Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

Рівень вищої освіти перший (бакалаврський)

Ігровий застосунок у жанрі симулятора
з використанням рушія Unity

(тема)

Виконав:
здобувач 3 року навчання,
групи КІУКІу-22-1
Єгор МУХІН
(власне ім’я, прізвище)

Спеціальність
123 «Комп’ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма
Комп’ютерна інженерія
(повна назва освітньої програми)

Керівник: доц. Тетяна ФІЛІМОНЧУК
(посада, власне ім’я, прізвище)

Допускається до захисту

Завідувач кафедри ЕОМ _____ Андрій КОВАЛЕНКО
(підпис) (власне ім’я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти перший (бакалаврський)

Спеціальність 123 «Комп’ютерна інженерія»
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Комп’ютерна інженерія
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
“____” _____ 20__ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Мухіну Єгору Ігоровичу
(прізвище, ім’я, по батькові)

1. Тема роботи Ігровий застосунок у жанрі симулятора з використанням рушія Unity

затверджена наказом по університету від “ 25 ” травня 2025 р. № 425 Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 14 липня 2025 р.

3. Вхідні дані до роботи

1) рушій Unity; 2) мова програмування C#; 3) середа розробки Visual Studio 2022.

4. Перелік питань, що потрібно опрацювати у роботі

1) аналіз проблеми та огляд існуючих рішень;
2) вибір технології розробки та інструментальних засобів;
3) розробка алгоритмічного забезпечення;
4) розробка програмних модулів;
5) відлагодження програмних модулів;
6) висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій

Слайд-презентація – 11 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Аналіз існуючих методів вирішення задачі	10.06.2025-13.06.25	
2	Вибір програмного забезпечення та	14.06.2025-17.06.25	
	інструментів розробки		
3	Проєктування архітектури застосунку	18.06.25-21.06.25	
4	Розробка ігрової логіки	22.06.25-28.06.25	
5	Розробка графічного інтерфейсу користувача	29.06.25-02.07.25	
6	Тестування застосунку	03.07.25-05.07.25	
7	Подання кваліфікаційної роботи керівникам	06.07.25-09.07.25	
	для попереднього захисту		
8	Подання кваліфікаційної роботи на	10.07.25-11.07.25	
	рецензування		

Дата видачі завдання " 09 " червня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ доц. Тетяна ФІЛІМОНЧУК
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 65 с., 30 рис., 1 дод., 16 джерел.

РУШІЙ, КОРИСТУВАЦЬКИЙ ІНТЕРФЕЙС, ООП, CLEAN ARCHITECTURE, SOLID, СКРИПТОВІ ОБ’ЄКТИ, РЕНДЕРІНГ, СКРИПТ, BEHAVIOR TREE.

Метою кваліфікаційної роботи є створення ігрового додатку у жанрі симулятор фермера.

У ході виконання кваліфікаційної роботи було створено гру-симулятор фермера для Windows на Unity з використанням мови C#. Гра включає всі основні елементи жанру: вирощування рослин, систему речей, взаємодію з неігровими персонажами, будівництво, завдання та створення предметів.

При розробці використовувалися сучасні підходи до програмування: Clean Architecture та принципи SOLID, що робить код зрозумілим та легким для розвитку. Технічно проєкт побудований на основі скриптів контролерів та ScriptableObjects, власних редакторах та системі поведінки персонажів через Behavior Tree.

Особлива увага приділялася зручності інтерфейсу та простоті управління. В результаті отримали повноцінну гру з інтуїтивним управлінням та різноманітними можливостями, яка відповідає сучасним вимогам до фермерських симуляторів.

Створений проєкт демонструє успішне застосування теоретичних знань на практиці та може служити основою для подальшого розвитку.

ABSTRACT

Bachelor’s thesis: 65 pages, 30 figures, 1 appendix, 16 sources.

ENGINE, USER INTERFACE, OOP, CLEAN ARCHITECTURE, SOLID, SCRIPTABLE OBJECTS, RENDERING, SCRIPT, BEHAVIOR TREE.

The purpose of the qualification work is to create a game application in the farming simulator genre.

In order to demonstrate the ability to apply theoretical knowledge in practice and to develop a complete game product, a farming simulator game for Windows was created during the qualification work using the Unity engine and the C# programming language.

The game includes all the core elements of the genre: growing plants, managing an inventory system, interacting with non-playable characters, building structures, completing tasks, and crafting items.

During the development process, modern software engineering practices were applied — including Clean Architecture and SOLID principles. This helped ensure that the codebase remained well-structured, maintainable, and scalable. Technically, the architecture of the project is based on controller scripts and ScriptableObjects, supported by custom editor tools and a character behavior system implemented through Behavior Trees.

A particular focus was placed on user interface convenience and intuitive controls. As a result, the final product is a fully functional farming simulation game that offers a smooth user experience and features that meet the expectations of modern players within the genre.

The developed project clearly demonstrates the successful integration of theory and practice and can serve as a solid foundation for further expansion and improvement.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

AOT – компілятор (англ., Ahead-of-Time)

DRY – принцип програмування «не повторюйся» (англ., Don't Repeat Yourself)

FPS – кадрів за секунду (англ. Frames Per Second)

HDRP – технологія 3D рендерінгу (англ., High Definition Render Pipeline)

IDE – комплексне програмне рішення для розробки програмного забезпечення (англ., Integrated Development Environment)

KISS – принцип програмування «коротко та просто» (англ., Keep It Simple Stupid)

LINQ – запити інтегровані у мову (англ., Language-Integrated Query)

LMB – ліва кнопка миші (англ. Left Mouse Button)

NPC – не ігровий персонаж (англ., No Player Character)

PC – персональний комп’ютер (англ. Personal Computer)

SOLID – принципи об'єктно-орієнтованого програмування

UI – інтерфейс користувача (англ., User Interface)

UX – досвід користувача (англ., User Experience)

VFX – візуальні ефекти (англ. Visual Effects)

ВСТУП

Відеоігри – це інтерактивне програмне забезпечення, яке використовується для розваг, рольових ігор та симуляції. У відеоігри можна грати на комп’ютері з ОС Windows, мобільному пристрої або спеціальній ігровій консолі, наприклад Xbox або PlayStation [1].

У наш час відеоігри перетворилися на один із ключових елементів цифрової епохи та культурного простору. Охоплюючи величезну аудиторію, ігрова індустрія пропонує різноманітний контент: від легких аркад до масштабних онлайн-проектів з великою кількістю учасників. Занурення у віртуальні пригоди не лише розважає, але й розвиває: гравці постійно тренують творче мислення, опановують мистецтво розв'язання складних задач та вдосконалюють навички стратегічного планування, стикаючись із різними випробуваннями в динамічному ігровому середовищі.

Сучасні технології підняли відеоігри на новий рівень, забезпечивши їх вражаючою графікою та реалістичною фізикою, що суттєво збагачує ігровий досвід. Створення ігор перетворилося на комплексне мистецтво, де переплітаються програмування, дизайнерська майстерність, звукорежисура та анімація. Сьогодні віртуальні світи пропонують гравцям не просто розваги, а й глибоке занурення у простір, де порушуються важливі суспільні питання та відображаються культурні цінності.

Симулятори посідають визначну позицію в різноманітному світі відеоігор. Цей унікальний жанр дозволяє користувачам опанувати керування різноманітними віртуальними системами, які достовірно відтворюють реальні життєві процеси. Діапазон симуляцій вражає: від опанування транспортними засобами (автомобілями та літаками) до управління масштабними проєктами, як-от розбудова міст чи керування підприємствами. Поєднання достовірності та інтерактивності перетворює симулятори на потужний інструмент, що не лише розважає, а й навчає, допомагаючи розвивати практичні навички та відточувати управлінські здібності.

У світі симуляторів особливою популярністю користуються фермерські ігри, де гравці беруть на себе роль господаря сільськогосподарського підприємства. Ці проекти пропонують унікальний досвід взаємодії з природними циклами, включаючи вирощування культур, догляд за худобою та раціональне використання ресурсів. Фермерські симулятори – це не просто розвага, а й платформа для розвитку стратегічного мислення, де гравці вчаться долати реалістичні виклики: враховувати сезонні зміни, керувати фінансами та ефективно розподіляти ресурси. Така комбінація викликів робить ігровий процес одночасно захопливим та освітнім.

Ця кваліфікаційна робота має на меті створення ігрового застосунку в жанрі симулятора фермера на рушії Unity для операційної системи Windows.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

В ході виконання кваліфікаційної роботи слід вирішити задачі:

- проаналізувати існуючі на даний час симулятори фермерських господарств з точці зору недоліків, які ним присутні;
- розробити структуру застосунку з урахуванням всіх необхідних механік гри;
- побудовати застосунок, який буде мати характерні риси для симуляторів фермера;
- розробити дизайн світу гри, який буде привабливим для гравця та відповідати тематиці;
- забезпечити задовільний досвід використання UI та UX, шляхом впровадження інтуїтивної навігації та приємного для ока дизайну.

1.1 Аналіз існуючих рішень

На даний час існує кілька готовий рішень, що надають представлення того як мають виглядати та працювати ігри у жанрі симулятор фермера. Ці рішення мають необхідний функціонал, який задовольняє потреби гравця, проте кожне з них має свої переваги та недоліки. Розглянемо детальніше кілька популярних ігор цього жанру.

1.1.1 Гра-симулятор Slime Rancher

Slime Rancher (рисунок 1.1) – це гра у жанрі симулятор фермера, де гравцю, доведеться взяти під своє керування старе ранчо, на якому треба буде вирощувати кумедних створінь «слаймів», щоб отримувати від них «плорди» та продавати їх на ринку [2].

Гра Slime Rancher створила власну нішу в жанрі фермерських симуляторів, запропонувавши незвичний підхід до традиційного

господарства. Замість звичних сільськогосподарських тварин та культур, гравці опікуються веселими створіннями – «слаймами», що надає проєкту неповторного характеру. Гра вдало поєднує елементи фермерства з можливістю вільно досліджувати навколишній світ, розширюючи простір для творчості та експериментів. Привабливий візуальний дизайн з життєрадісними кольорами створює розслаблюючу атмосферу, а зрозумілі механіки керування роблять гру доступною як для новачків, так і для досвідчених гравців.

Рисунок 1.1 – Кадр з гри «Slime Rancher»

Проте гра має свої обмеження. Порівняно з традиційними фермерськими симуляторами, тут спрощена система управління ресурсами та економікою, що може розчарувати любителів складних стратегічних рішень. Досвідчені гравці можуть знайти ігровий процес занадто простим, а повторювані дії з догляду за слизами та збору ресурсів ризикують стати монотонними. Відсутність взаємодії з NPC, яка зазвичай збагачує подібні ігри соціальним елементом, створює відчуття самотності у грі. Ці фактори можуть обмежити довготривалий інтерес до проекту.

1.1.2 Серія ігор-симуляторів від Farming Simulator

Farming Simulator (рисунок 1.2) – це серія реалістичних симуляторів фермерства, де гравець керує сучасною сільськогосподарською технікою, вирощує культури, доглядає за тваринами та розвиває власне господарство, продаючи врожаї на ринку [3].

Рисунок 1.2 – Сільськогосподарська техніка в Farming Simulator

Farming Simulator встановлює високу планку в жанрі фермерських симуляторів завдяки своєму прагненню до максимальної достовірності. Гра пропонує глибоке занурення у світ сучасного сільського господарства, де гравці керують точними копіями реальної сільгостехніки та займаються всіма аспектами фермерської діяльності – від вирощування рослин до тваринництва та торгівлі. Особливу цінність грі додає активна спільнота модерів, яка постійно створює новий контент, розширюючи можливості гри.

Втім, така відданість реалізму має і зворотний бік. Неспішний темп геймплею та зосередженість на технічних деталях можуть відлякувати гравців, які прагнуть більш жвавого та різнопланового ігрового процесу. Новачкам може бути складно освоїтися в грі через її специфічність. Крім того, обмежені можливості соціальної взаємодії створюють відчуття певної

відірваності від живого світу, що контрастує з іншими іграми цього жанру. Однак саме ця увага до деталей та реалістичність зробили гру популярною серед справжніх ентузіастів сільського господарства, які цінують можливість випробувати себе у ролі професійного фермера.

1.1.3 Інді-гра про сільське господарство Stardew Valley

Stardew Valley (рисунок 1.3) – гра, в якій гравець успадковує занедбану ферму, і його задачою, маючи лише не великі фінанси на початку гри, розвинути власне фермерське господарство [4].

Рисунок 1.3 – Кадр з гри Stardew Valley

Stardew Valley приваблює своєю багатогранністю, де кожен гравець може знайти улюблене заняття: від традиційного фермерства та тваринництва до риболовлі, створення ремісничих виробів та захопливих пригод у підземеллях. Така різноманітність дозволяє створювати унікальний шлях розвитку власного господарства. Гра виділяється глибокою системою взаємовідносин: спілкування з жителями містечка, можливість розвивати дружбу та навіть будувати романтичні стосунки додають емоційного виміру ігровому процесу. Розгалужена система квестів та подій забезпечує постійне

відчуття розвитку та просування вперед.

Проте не всім може припасти до душі піксельна графіка в ретро-стилі, а початкові етапи гри можуть здатися надто повільними. Деякі гравці можуть втомитися від циклічності щоденних фермерських завдань. Спрощена система створення предметів може розчарувати тих, хто шукає більш комплексні механіки створення речей. Хоча Stardew Valley створює чарівну та затишну атмосферу, її базові механіки можуть здатися надто простими для гравців, які звикли до більш складних симуляторів.

1.1.4 Онлайн гра про фермерське господарство FarmVille

FarmVille (рисунок 1.4) – це онлайн-гра у жанрі симулятора фермера, де гравець керує власною віртуальною фермою: вирощує культури, доглядає за тваринами та прикрашає ферму, виконуючи завдання для розширення господарства і взаємодії з друзями [5].

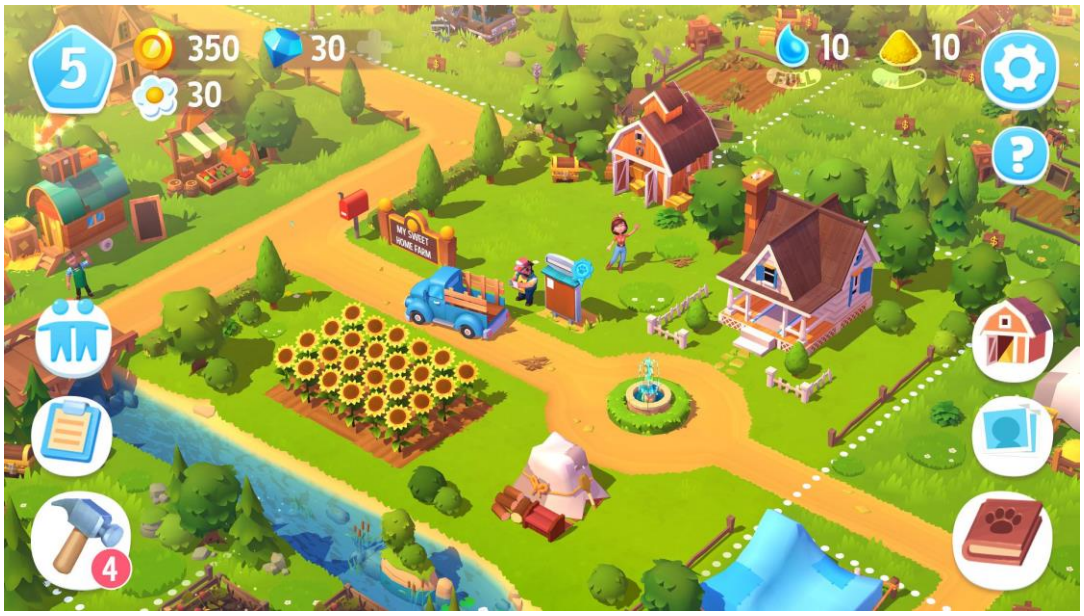


Рисунок 1.4 – Кадр з гри FarmVille

FarmVille відрізняється від інших фермерських симуляторів своїм акцентом на соціальній складовій. Гра заохочує взаємодію між користувачами через систему взаємодопомоги у розвитку господарств, що

формує активну спільноту та підвищує задоволення від гри. Інтуїтивно зрозумілі механіки та можливість грати короткими сесіями роблять гру привабливою для casual-гравців та тих, хто має обмежений час. Постійний потік нового контенту та спеціальних подій допомагає підтримувати залученість аудиторії.

Водночас бізнес-модель гри, побудована на системі внутрішньоігрових покупок, може стати каменем спотикання для тих, хто надає перевагу традиційній системі оплати. Одноманітність щоденних завдань поступово знижує ентузіазм гравців, а базові механіки управління фермою та прості економічні взаємозв'язки навряд чи зацікавлять любителів приймати складні рішення для проходження та глибокої симуляції господарства.

1.2 Постановка задачі кваліфікаційної роботи

З огляду на опис задачі, треба розробити гру на PC, яка матиме наступний набір можливостей:

- відкриття головного меню при запуску гри;
- налаштування гри (звук, видимість камери);
- пересування та керування персонажем у грі;
- вирощування та збір врожаю;
- продаж та купівля товару в NPC;
- взаємодія з NPC;
- купівля та розміщення будівель;
- система створення речей;
- можливість взяття та виконання внутрішньоігрових квестів;
- система збереження речей (інвентар).

При створенні фермерського симулятора на Unity ключову роль відіграє дотримання фундаментальних принципів розробки програмного забезпечення. Зокрема, впровадження принципів SOLID забезпечує створення надійної архітектурної основи, яка легко піддається модифікаціям

та розширенню. Такий підхід гарантує мінімальну зв'язаність між компонентами, що особливо важливо для ігор, які потребують регулярних оновлень та масштабування.

Реалізація принципу DRY допомагає уникнути надлишковості в коді, що особливо актуально для ігрових проектів. У контексті фермерського симулятора це стосується, наприклад, створення універсальних механік взаємодії з ігровими об'єктами – будь то рослини, неігрові персонажі чи інвентар. Такий підхід значно полегшує процес оновлення та розширення функціоналу гри.

Принцип KISS закликає до максимальної простоти у реалізації проєкту на Unity. Це означає, що як ігрові механіки, так і інтерфейс користувача мають бути інтуїтивно зрозумілими для всіх учасників процесу – від програмістів до кінцевих користувачів. Простота рішень мінімізує ймовірність помилок, спрощує тестування та знижує ризики майбутніх ускладнень [6].

Використання цих принципів створює міцний фундамент для подальшого розвитку проєкту. Добре структурований код полегшує впровадження нових функцій та виправлення помилок без необхідності суттєвого переписування існуючої бази. У контексті ігрової розробки, де вимоги постійно еволюціонують, така гнучка архітектура дозволяє легко адаптувати гру до нових потреб – від додавання нового контенту до впровадження складних ігрових механік що робить процес розробки та підтримки симулятора фермера більш ефективним як для команди розробників, так і для задоволення очікувань гравців.

2 АНАЛІЗ ТЕХНОЛОГІЙ, ЩО ВИКОРИСТОВУЮТЬСЯ ПРИ РОЗРОБЦІ ІГРОВОГО ДОДАТКУ

2.1 Огляд засобів розробки

2.1.1 Мова програмування C#

C# постає сучасною об'єктно-орієнтованою мовою програмування, розробленою Microsoft для створення різноманітного програмного забезпечення в екосистемі .NET. Вона була представлена у 2000 році як складова стратегії розвитку міжмовної платформи для Windows, мова успадкувала синтаксичні риси Java, C++ та інших високорівневих мов, що робить її легкозасвоюваною для професійних програмістів [7].

Визначальною характеристикою C# є її потужні мовні конструкції, спрямовані на створення чіткого та легкочитного коду. Автоматичне керування пам'яттю через механізм збирача сміття мінімізує ризики помилок пам'яті, а підтримка складних концепцій, як-от делегати, події, анонімні методи та LINQ, забезпечує гнучкість та ефективність розробки додатків.

Мова підтримує різноманітні стратегії компіляції, що розширює її універсальність. Just-In-Time (JIT) компіляція дозволяє оптимізувати код безпосередньо під час виконання програми, що є стандартним підходом у .NET. Додаткова можливість AOT-компіляції виявляється корисною при створенні високопродуктивних додатків для пристроїв з обмеженими ресурсами.

Завдяки багатому інструментарію, інтуїтивному синтаксису та потужній спільноті розробників, C# перетворилася на провідну мову для створення широкого спектра застосунків: від настільних програм до мобільних застосунків, вебсервісів, ігрових проєктів (зокрема на основі Unity) та хмарних сервісів через .NET Core та Azure. Постійний розвиток мови робить її актуальним вибором для сучасних розробників.

2.1.2 Ігровий рушій Unity

Unity – це провідний ігровий рушій, який надає розробникам унікальні можливості для створення інтерактивних застосунків та відеоігор на різноманітних платформах. Він був представлений у 2005 році, швидко завоював провідні позиції у світі розробки ігор завдяки простоті використання, потужному функціоналу та активній підтримці глобальної спільноти розробників [8].

Визначальною перевагою Unity є його кросплатформенність. Рушій дозволяє створювати програмні продукти, які легко адаптуються більш ніж до 25 різних платформ – від комп'ютерів та мобільних пристроїв до ігрових консолей та систем віртуальної (VR) та доповненої реальності (AR). Така гнучкість суттєво оптимізує процес розробки та мінімізує витрати на адаптацію проектів.

Підтримка мови програмування C# робить Unity зручним інструментом для реалізації складних ігрових механік та логіки. Вбудована система компонентів дозволяє розробникам швидко додавати функціональність до ігрових об'єктів через інтуїтивно зрозумілий інтерфейс, що значно прискорює процес створення прототипів та налаштування взаємодії між елементами гри.

Графічні можливості Unity вражають своєю універсальністю. Рушій підтримує як двовимірну, так і тривимірну графіку, надаючи професійні інструменти для створення реалістичних сцен. Такі додаткові інструменти, як Shader Graph, система постобробки зображень та технології високопродуктивного рендерингу HDRP (High Definition Render Pipeline) та URP (Universal Render Pipeline), дозволяють досягати вражаючих візуальних ефектів. Анімаційна система Unity, що включає в себе Mecanim та Timeline, надає розробникам зручні засоби для створення плавних анімацій та кінематографічних сцен.

Unity є унікальною платформою, що підходить для розробників будь-

якого рівня підготовки – від початківців, які роблять перші кроки у створенні ігор, до професійних команд, які працюють над складними масштабними проєктами. Його гнучкість, доступність та потужний функціонал роблять Unity неперевершеним інструментом у світі розробки інтерактивних додатків.

2.1.3 Інтегроване середовище розробки Visual Studio 2022

Visual Studio 2022 є сучасним потужним інтегрованим середовищем розробки від Microsoft, яке пропонує комплексні інструменти для створення програмних додатків на різних платформах. Підтримка 64-бітної архітектури забезпечує надійну роботу з масштабними проєктами, гарантуючи високу стабільність та продуктивність розробки.

Середовище підтримує широкий спектр мов програмування, зокрема C#, C++, Python, JavaScript та інші, і має глибоку інтеграцію з .NET 6, що дозволяє створювати кросплатформні додатки.

Visual Studio 2022 пропонує потужні можливості для розробників ігор, які працюють з Unity. Редактор забезпечує комфортне середовище для написання ігрових скриптів на мові C#. Крім того, вбудований налагоджувач Unity дозволяє розробникам ретельно вивчати процес виконання скриптів, маючи змогу миттєво спостерігати за значеннями змінних та відстежувати кожен крок коду безпосередньо під час роботи програми, що значно полегшує процес налагодження та оптимізації ігрового коду.

2.1.4 Система контролю версій Git

Git є сучасною системою контролю версій, яка надає розробникам широкі можливості для спільної роботи та відстеження еволюції програмного коду. Система підтримує два типи репозиторіїв: локальні, що розміщені безпосередньо на комп'ютері розробника, та віддалені, які знаходяться на

серверах таких платформ, як GitHub [9].

При розробці даного проєкту використовувалася система Git та сервіс GitHub. Центральним елементом є гілка main, у яку зливаються всі інші гілки зі змінами. Для полегшення роботи з Git використовується внутрішній інтсрументарій Visual Studio, який полегшує роботу.

Застосовується стратегія іменування гілок у форматі TYPE/N-TASK, де TYPE – тип задачі, N – номер задачі, а TASK – іменування завдання. На початку кожній задачі присвоюється тип (наприклад back-end, fix та ін.), номер та ім’я, після чого створюється окрема гілка. Такий підхід забезпечує прозору систему організації коду, що дає змогу команді легко відстежувати поточний стан проєкту та орієнтуватися в множині паралельних гілок. Особливої ефективності цей метод набуває саме в командній розробці, де важлива чітка структура та узгодженість роботи над різними компонентами проєкту. GitHub дозволяє налаштувати проєкт так, щоб усі зміни в гілку dev вносилися через Pull request. Такий підхід дає змогу іншим розробникам переглядати запропоновані зміни, залишати коментарі чи зауваження щодо можливих помилок або неточностей. Автор змін, у свою чергу, має врахувати отриманий зворотний зв'язок та доопрацювати код перед затвердженням.

2.1.5 3D-редактор Blender

Blender – це багатофункціональна програма для розробки 3D-графіки, яка дає творцям, митцям та проєктувальникам численні інструменти для тривимірного моделювання, створення анімацій, візуалізації та спецефектів. Запущений у 1995 році, Blender перетворився на один з найпопулярніших засобів у галузі тривимірної графіки завдяки безкоштовній ліцензії, регулярним удосконаленням та підтримці світової спільноти користувачів [10].

Програма містить комплексний інструментарій для створення 3D-моделей, нанесення текстур, цифрової скульптури, анімування та фізичного

моделювання. Вбудований модуль Geometry Nodes забезпечує можливості для створення процедурних об'єктів та комплексних ефектів без написання коду, що значно полегшує роботу над складними проєктами.

Система візуалізації є важливою перевагою Blender. Програма включає Cycles – рендерер на основі фізично коректного трасування променів, а також Eevee – швидкий рендерер реального часу, який забезпечує якісні результати без тривалих обчислень.

Через сумісність з іншим програмним забезпеченням, зокрема Unity, Blender стає універсальним рішенням для розробки комп'ютерних ігор, кіноефектів, архітектурних візуалізацій та моделей для 3D-друку.

Blender задовольняє потреби як новачків у світі 3D-графіки, так і досвідчених фахівців, які працюють над масштабними проєктами. Його потужні можливості, адаптивність та вільна екосистема роблять Blender одним із провідних інструментів у сфері цифрової графіки.

2.1.6 Магазин асетів Unity AssetStore

Unity Asset Store – це офіційний інтернет-магазин компонентів для ігрового двигуна Unity, який пропонує творцям ігор доступ до розмаїття готових елементів, що допомагають пришвидшити розробку ігрових та інтерактивних проєктів. Відкритий у 2010 році, магазин швидко став незамінним сервісом як для незалежних розробників, так і для великих компаній, дозволяючи суттєво економити ресурси та концентруватися на оригінальних аспектах своїх проєктів [11].

Головна цінність Unity Asset Store полягає в його обширній бібліотеці ресурсів, що включає тривимірні моделі, анімаційні послідовності, текстурні матеріали, аудіоефекти, музичні треки, шейдерні програми, програмний код, утиліти для розробки, а також готові системи для штучного інтелекту, фізичного моделювання, інтерфейсу користувача та багатьох інших ігрових компонентів. Це дає можливість розробникам оперативно впроваджувати

готові рішення у свої проєкти, не витрачаючи час на створення базових елементів.

Особливістю Asset Store є доступність як безплатних, так і платних преміум-ресурсів, що робить платформу корисною для розробників з різними фінансовими можливостями. Безкоштовні ресурси дозволяють пробувати нові підходи та вчитися, в той час як професійні платні рішення забезпечують високу якість та часто використовуються у комерційних проєктах.

Завдяки безперебійній взаємодії з Unity, ресурси можна легко завантажувати та впроваджувати безпосередньо в середовищі розробки Unity, що раціоналізує робочий процес. Крім того, магазин має систему рейтингів та коментарів, яка допомагає користувачам виявляти якісні ресурси та уникати неякісних продуктів.

Unity Asset Store є незамінним ресурсом для розробників усіх рівнів – від новачків, які шукають швидкі рішення для навчальних проєктів, до професійних команд, які використовують магазин для оптимізації робочих процесів та зменшення витрат. Його великий асортимент, простота використання та тісна інтеграція з Unity роблять Asset Store одним із найважливіших інструментів для сучасних розробників відеоігор.

2.2 Архітектурні підходи та принципи

2.2.1 Архітектурний шаблон Clean Architecture

Clean Architecture, розроблений Робертом С. Мартіном, є архітектурним підходом, який забезпечує гнучкість, зручність тестування та простоту підтримки програмних систем. Його головна ідея полягає у чіткому поділі відповідальностей між рівнями архітектури, що знижує зв’язність компонентів і дозволяє незалежно змінювати різні частини системи. У центрі архітектури знаходиться бізнес-логіка, ізольована від деталей реалізації,

таких як фреймворки, бази даних або зовнішні сервіси [10].

Ключовим принципом є спрямованість залежностей всередину: зовнішні шари можуть залежати від внутрішніх, але не навпаки. Наприклад, бізнес-логіка не залежить від конкретної бази даних чи інтерфейсу користувача, що дозволяє змінювати ці елементи без впливу на основну функціональність системи.

Переваги Clean Architecture включають гнучкість, легкість тестування та незалежність від інструментів. Кожен шар архітектури можна тестувати окремо, що спрощує створення модульних та інтеграційних тестів, що робить систему легшою у підтримці та масштабуванні, оскільки зміни зовнішніх технологій, наприклад, заміна бази даних чи фреймворка, не вимагають змін у бізнес-логіці.

2.2.2 Використання патерну проєктування

Патерн проєктування – це стандартний підхід до вирішення типових задач, які виникають під час розробки архітектури ПЗ. На відміну від готових функцій або бібліотек, патерн не є готовим до використання кодом, який можна просто вставити в програму. Він представляє собою загальну концепцію або принцип вирішення проблеми, що вимагає адаптації до конкретних потреб та особливостей проєкту [12].

Одним з таких патернів є Singleton, який забезпечує створення лише одного екземпляра класа, та надає глобальний доступ до нього для всієї програми. Це корисно для взаємодії з глобальними скриптами (менеджери та контролери), які виконують основну логіку гри.

Використання патернів проєктування сприяє створенню добре структурованих систем, які прості у підтримці та розширенні. Вони допомагають дотримуватися принципів SOLID та впроваджують найкращі практики програмування, що робить код більш зрозумілим та зручним для подальшого обслуговування.

2.2.3 Базовий клас MonoBehavior

MonoBehaviour – це основний клас в Unity, від якого наслідуються всі скрипти, що повинні взаємодіяти з ігровими об'єктами та компонентами в сцені. Цей клас є ключовим для створення взаємодії з ігровим середовищем, оскільки дозволяє скриптам функціонувати в контексті ігрових об'єктів Unity, використовувати можливості компонентів та взаємодіяти з іншими елементами гри, такими як фізика, анімація або введення від користувача. Значення MonoBehaviour полягає в тому, що він дозволяє автоматично прив'язувати об'єкти та компоненти до ігрового об'єкта в редакторі, забезпечуючи зручне управління життєвим циклом об'єктів [13].

Життєвий цикл в Unity є послідовністю подій, що відбуваються з об’єктом від його початку появу на ігровій сцені й до його знищення, або закриття сцени. Завдяки цієї послідовності можна налаштувати дані та інші компоненти, що присутні на ньому, під час ініціалізації, ігрового процесу чи закінчення гри, що дозволяє гнучко маніпулювати логікою гри. Крім цього, життєвий цикл дозволяє позитивно вплинути на оптимізацію проєкту, очіщуючи посилання на неактивний об’єкт та непотрібні ресурси, які дуже довго не використовувалися гравцем.

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ ДОДАТКУ

3.1 Огляд структури проєкту

У рамках проєкту використовується загальна архітектура, що характерна Unity проєктам – розділення кожного файлу на свої спеціалізовані папки. На рисунку 3.1 представлено структуру проєкту. Такий підхід дозволяє отримати просту навігацію по проєкту та дозволяє масштабувати його без втрати важливих файлів.

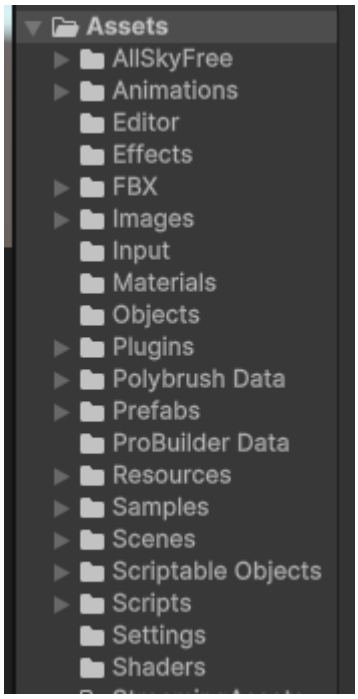


Рисунок 3.1 – Структура проєкту

Для коду застосується архітектурний підхід Clean Achitecture, що передбачає розділення коду на логічно розділені шари з чіткими відповідальностями. На рисунку 3.2 представлено структура коду.

Скрипти-контролери – верхній шар, що напряму взаємодіють з об’єктами гри. Всі інші скрипти, що знаходяться в інших папках, знаходяться на нижчому шарі, в якому опрацьовуються дані потрібні для контролерів, й не як на пряму не взаємодіють з об’єктами. Такий підхід дозволяє зберегти

незалежність компонентів, полегшує навігацію та дозволяє легко вносити необхідні зміни в проєкті не ризукуючи отримати серйозні помилки в роботі.

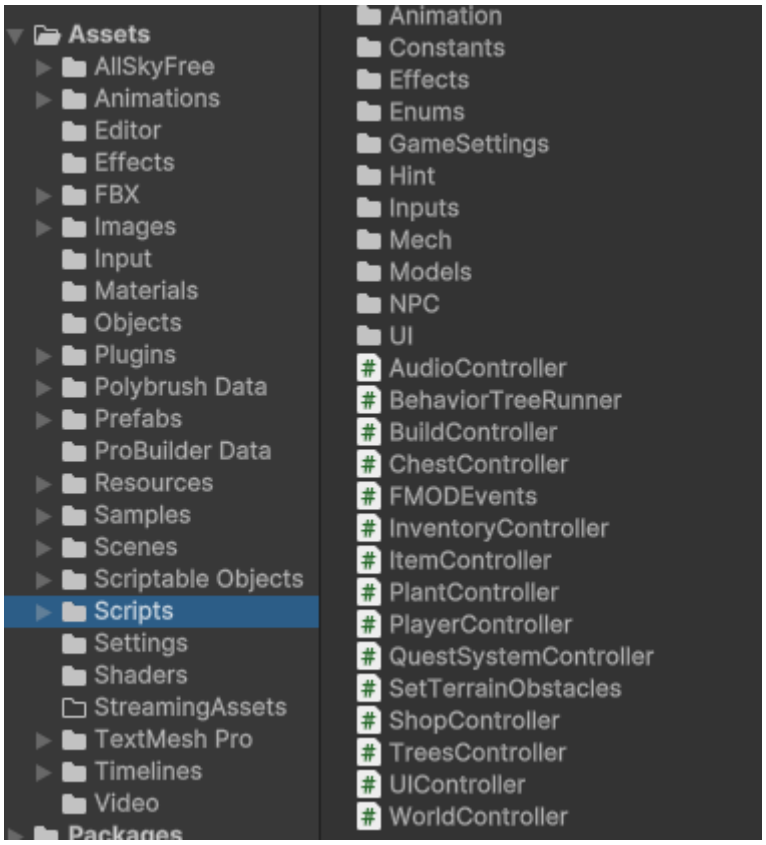


Рисунок 3.2 – Структура коду

3.2 Каталог Animations

Каталог Animations містить анімаційні файли: анімаційні кліпи (Animation Clip) та анімаційні контролери (Animation Controller). Анімаційні кліпи містять інформацію про зміну об’єкта, без використання скриптів, між двома або більшою кількістю ключових кадрів (Key frame). В кліпах можуть змінюватися різні властивості об’єкта: положення, розмір, оберт, колір та ін.

Іноді для одного об’єкта може бути декілька анімаційних кліпів, й для того щоб їх поєднати існують анімаційні контролери. Завдяки ним, можна налаштувати переходи між кліпами та виставити умови переходу, щоб анімація була плавнішою та логічно зв’язаною. Також з контролерами можна зв’язуватися через скрипт, що допомагає змінювати деякі дані анімації,

наприклад швидкість програвання анімації. На рисунку 3.3 можна побачити анімаційний контролер, що відповідає за анімації гравця.

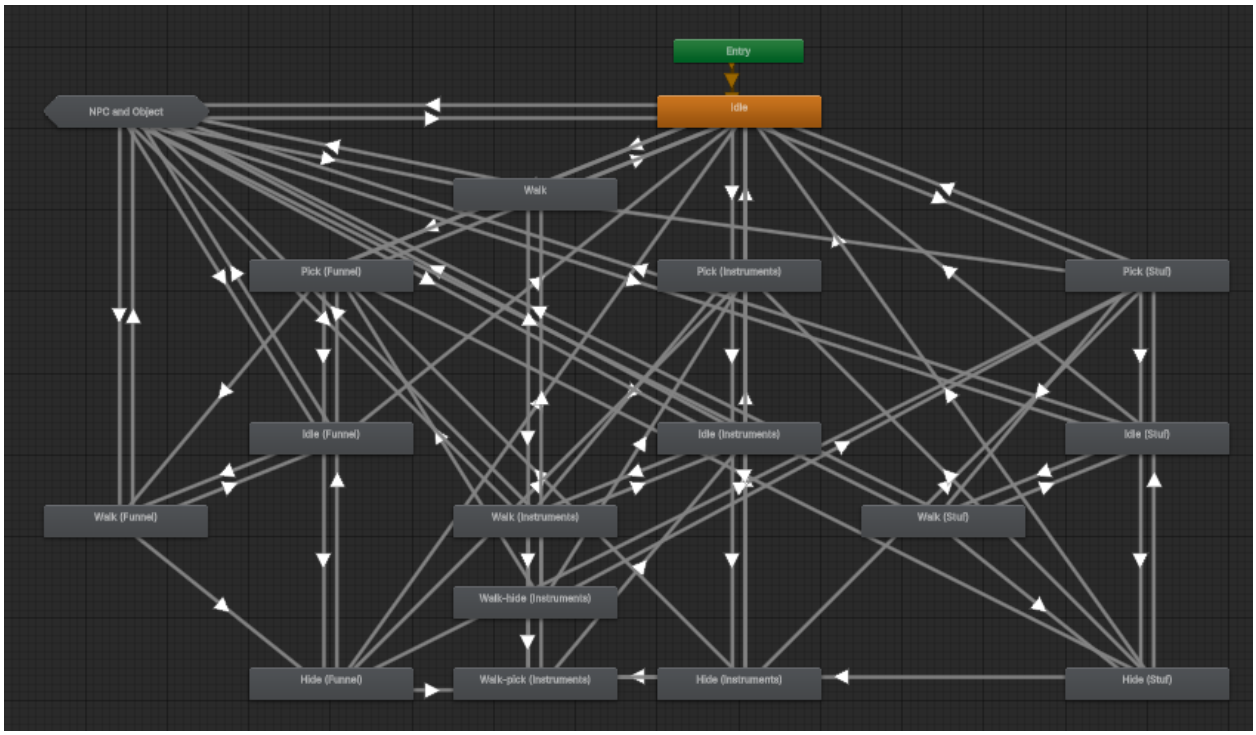


Рисунок 3.3 – Анімаційний контролер гравця

3.3 Каталог Editor

Каталог Editor у проєкті містить скрипти, що розширюють функціональність Unity Editor та використовуються виключно в середовищі розробки. У ньому зосереджено інструменти, які полегшують створення та налагодження ігрових механік. Зокрема, в цьому каталозі знаходяться кастомні вікна та компоненти, що дозволяють розробникам працювати з внутрішніми системами гри безпосередньо через редактор.

Для побудови графічного інтерфейсу використовуються файли UXML, які визначають структуру елементів UI подібно до HTML, а також файли USS, що задають стилі оформлення, аналогічно до CSS. Одним із ключових елементів цього каталогу є вікно для створення та редагування дерева поведінки NPC (лістинг 3.1). Завдяки цьому редакторському інструменту

можливо зручно керувати логікою штучного інтелекту, додаючи й налаштовуючи вузли дерева візуально, без необхідності втручання в програмний код.

Лістинг 3.1 – Метод PopulateView скрипта BehaviorTreeView

```
internal void PopulateView(BehaviorTree tree)
{   _tree = tree;
    graphViewChanged -= OnGraphViewChanged;
    DeleteElements(graphElements);
    graphViewChanged += OnGraphViewChanged;
    if(_tree.RootNode == null)
    {   _tree.RootNode = _tree.CreateNode(typeof(RootNode)) as
RootNode;
        EditorUtility.SetDirty(_tree);
        AssetDatabase.SaveAssets();
    }
    _tree.Nodes.ForEach(n => CreateNodeView(n));
    _tree.Nodes.ForEach(n =>
    {   var children = _tree.GetChildren(n);
        children.ForEach(c =>
        {   NodeView parentView = FindNodeView(n);
            NodeView childView = FindNodeView(c);
            Edge edge =
parentView.Output.ConnectTo(childView.Input);
            AddElement(edge);
        });
    });
}
```

3.4 Каталог Effects

Каталог Effects містить візуальні ефекти (VFX), що використовуються для створення яскравих та динамічних сцен у грі. У каталозі зберігаються заготовки ефектів, створені з використанням систем частинок або інших візуальних засобів Unity. Вони застосовуються для підсилення враження від дій гравця або оточення, таких як удари, магічні дії, вибухи тощо.

Одним із ключових елементів цього каталогу є ефект вибуху, який використовується для візуалізації руйнування об'єктів або сильних ударів. Завдяки таким ефектам ігровий процес стає більш виразним та емоційно насиченим.

3.5 Каталоги FBX, Images та Input

Каталог FBX містить 3D моделі формату fbx, їх текстури та матеріали, що накладають текстуру в самій грі. Формат fbx є одним з найпростіших для налаштування в Unity. Він дозволяє розкласти об'єкт на окремі елементи (матеріали, текстури, сітка, анімації та ін.), що дозволяє їх редагувати згідно необхідних для проєкту вимог.

Каталог Images містить усі файли зображень, що виконують тимчасову роль текстур для об’єктів гри. Але крім цих зображень, в каталозі може міститися спрайти – зображення, які відображаються у користувацькому інтерфейсі користувача.

Каталог Input містить файли InputAction (рисунок 3.4), в яких закладено управління грою. InputAction – це доволі простий спосіб налаштувати керування за допомогою вводу майже з будь-якого перефірійного пристрою (клавіатура, миш, джойстик, дисплей телефону тощу) використовуючи мінімальний об’єм коду.

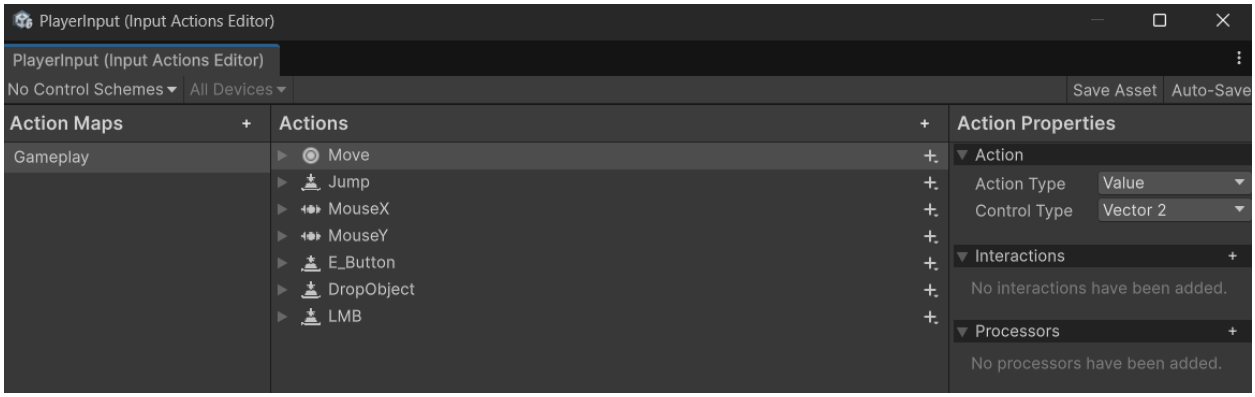


Рисунок 3.4 – InputAction для гравця

Також крім відслідковування натискання клавіш чи кнопок, InputAction дозволяє отримувати додадкові дані, наприклад, координати курсору миші, що спрощує обрахунок даних, які залежать від вводу. Це дає гнучкість для створення зручного управління, що дає розробникам легко змінювати або додавати нові схеми без переписування логіки.

3.6 Каталоги Materials та Object

Каталог Materials містить тимчасові матеріали, для об’єктів, які були створені всередині редактора Unity. Матеріали визначають, як повинна відтворюватися поверхня, включаючи посилання на текстури, які вона використовує, інформацію про мозаїку, відтінки кольорів тощо [15].

Каталог Object містить лише об’єкти інших форматів (не fbx об’єкти), а також вони є тимчасовими, й використовується лише для тестування деякого функціоналу.

3.7 Каталоги Plagins та Prefabs

Каталог Plugins є стандартним системним каталогом Unity, призначеним для зберігання сторонніх плагінів, бібліотек та зовнішнього коду, який інтегрується в проєкт. У цьому каталозі можуть міститися як готові рішення у вигляді .dll файлів, так і цілі підкаталоги з кодом, що забезпечує додаткову функціональність, не пов’язану безпосередньо з основною логікою гри. Unity автоматично обробляє вміст цієї папки особливим чином, зокрема може виділяти її залежності при збірці проєкту для різних платформ. Розміщення плагінів у цьому каталозі допомагає підтримувати порядок у проєкті та забезпечує коректну інтеграцію зовнішніх інструментів.

Каталог Prefabs містить шаблони об’єктів, які можна використовувати для побудови сцен або під час роботи проєту. Префаби (Prefab) – це дуже зручний інструмент в Unity. Він дозволяє створювати шаблон об’єкту зі всім потрібним йому функціоналом, компонентами та змінами, які зробив розробник. Крім того якщо у проєкті буде існувати декілька копій одного шаблону, які треба змінити, то достатньо внести зміни лише в одну копію та зберегти їх, щоб усі інші копії прийняли ці зміни. Це дозволяє економити велику купу часу в роботі над проєктом, й дозволяє не змінювати поштучно кожну копію одного шаблону.

3.8 Каталоги Resources та Scenes

Каталог Resources є одним з зазервованих каталогів в Unity. Він зберігає файли, які потрібно завантажувати під час виконання гри, забезпечуючи доступ до ресурсів (текстур, моделей, звуків тощо) без необхідності попередньої прив'язки до сцени чи об'єктів. Але її використання слід обмежувати через вплив на продуктивність і розмір збірки, оскільки всі файли з цієї папки включаються в гру незалежно від того, чи використовуються вони.

Каталог Scenes містить файли сцен, що використовуються в проєкті. Сцена в Unity – це основна робоча одиниця, яка представляє середовище гри або окрему її частину, де розміщуються і взаємодіють об'єкти такі як моделі, освітлення, камери, звуки та скрипти. Вона використовується для побудови рівнів, меню чи інших частин гри, з можливістю переходів між ними.

3.9 Каталог ScriptableObjects

Каталог ScriptableObjects містить спеціальні скриптовані об’єкті (Scriptable Object). Scriptable Object в Unity – це спеціальний тип об'єкта, який зберігає дані в окремих ресурсах поза сценою, дозволяючи організовувати, зберігати та повторно використовувати налаштування або дані (наприклад, параметри персонажів, конфігурації рівнів чи інвентар) без прив'язки до конкретного ігрового об'єкта, що сприяє зручності управління даними та оптимізації пам'яті.

3.9.1 ScriptableObject BehaviorTree

Скрипт BehaviorTree реалізує систему дерева поведінки для NPC, що використовується для створення гнучкого та візуально редагованого штучного інтелекту (лістинг 3.2). ScriptableObject зберігає кореневий вузол дерева, список усіх вузлів та "чорну дошку" (Blackboard) – спільне сховище

даних, яке можуть використовувати різні вузли.

Під час виконання гри дерево оновлюється через кореневий вузол, і його стан зберігається. У редакторі Unity цей скрипт також надає функції для створення, видалення, копіювання та зв’язування вузлів, що дозволяє зручно будувати і редагувати структуру дерева прямо в інтерфейсі редактора. Крім того, реалізована можливість обходу дерева з виконанням певної дії на кожному вузлі, клонування дерева разом з усіма його елементами, а також прив’язка дерева до конкретного NPC, щоб кожен вузол мав доступ до контролера персонажа та до загального сховища даних. Завдяки цьому скрипту забезпечується модульна та масштабована система прийняття рішень для NPC, яку можна легко редагувати та повторно використовувати.

Лістинг 3.2 – Метод CreateNode скрипта BehaviorTree

```
public Node CreateNode(System.Type type)
{    Node node = ScriptableObject.CreateInstance(type) as Node;
    node.name = type.Name;
    node.guid = GUID.Generate().ToString();
    Undo.RecordObject(this, "Behavior Tree (CreateNode)");
    Nodes.Add(node);
    if(!Application.isPlaying)
    {    AssetDatabase.AddObjectToAsset(node, this);
    }
    Undo.RegisterCreatedObjectUndo(node, "Behavior Tree
(CreateNode)");
    AssetDatabase.SaveAssets();
    return node;
}
```

3.9.2 ScriptableObject Node

Скрипт Node є базовим класом для всіх вузлів у дереві поведінки NPC. Він описує загальну структуру та логіку виконання вузла, включаючи життєвий цикл: початок, оновлення та завершення (лістинг 3.3). Кожен вузол має свій стан (виконується, успішно завершено або завершено з помилкою), а також зберігає службову інформацію, таку як позиція у редакторі, унікальний ідентифікатор, посилання на NPC і чорну дошку з даними. Завдяки цьому

класу забезпечується єдиний підхід до реалізації вузлів у дереві поведінки, що дозволяє легко масштабувати та підтримувати систему штучного інтелекту.

Лістинг 3.3 – Метод Update скрипта Node

```
public StateEnum Update()
{    if (!Started)
    {   OnStart();
        Started = true;
    }
    State = OnUpdate();
    if(State == StateEnum.Failure ||  State ==
StateEnum.Success)
    {   OnStop();
        Started = false;
    }
    return State;
}
```

3.10 Каталог Scripts

Каталог Scripts містить усі основні скрипти, які працюють на пряму з об’ктами (скрипти-контролери) та ті, що опрацьовують дані для отримання потрібного результату, що буде використовуватися скриптами-контролерами.

3.10.1 Скрипт AudioController

Скрипт AudioController відповідає за керування звуком у грі. Він централізує роботу зі звуковою системою FMOD [16], дозволяючи контролювати гучність музики, звукових ефектів та загальну гучність (лістинг 3.2). Після запуску гри він автоматично починає відтворювати фонову музику. Скрипт також забезпечує можливість створення окремих звукових подій та гарантує, що звуки коректно завершуються при виході зі сцени. Такий підхід дозволяє легко та гнучко керувати аудіо у проєкті, забезпечуючи якісне звукове оформлення.

Лістинг 3.4 – Метод Awake скрипта AudioController

```
private void Awake()
{if (instance == null){
        instance = this;
    }
    studioSystem = RuntimeManager.StudioSystem;
    studioSystem.getCoreSystem(out coreSystem);
    coreSystem.getMasterChannelGroup(out masterChannelGroup);
    musicBus = RuntimeManager.GetBus("bus:/Music");
    sfxBus = RuntimeManager.GetBus("bus:/SFX");
    _music = CreateInstance(FMODEvents.instance.BackroundMusic);
    _music.start();
}
```

3.10.2 Скрипт BuildController

Скрипт BuildController відповідає за управління будівлями у грі, включаючи перевірку ресурсів для крафту та оновлення будівлі, а також взаємодію з інвентарем і гравцем. Він має приватні поля для зберігання колекцій крафтів та покращень, рівня будівлі та властивість для отримання коефіцієнта зменшення витрат ресурсів залежно від рівня, ключові методи включають перевірку наявності необхідних предметів у гравця (лістинг 3.5).

Скрипт також відповідає за оновлення інтерфейсу при відкритті меню крафту, перевірку можливості здійснення дії та виконання апгрейду. Він взаємодіє з InventoryController, PlayerController та UIController, забезпечуючи централізовану логіку керування будівлями.

Лістинг 3.5 – Метод IsHasNeededItem скрипта BuildController

```
private bool IsHasNeededItem(List<Item> items, InformationModel
infoModel, List<int> amount = null)
{for (int i = 0; i < infoModel._neededItems.Count; i++){
NeededItem neededItem = infoModel._neededItems[i];
List<Item> neededItems = items.Where(item =>{
if (item == null) return false;
if (item._id != neededItem._item._id) return false;
return true;}).ToList();
if (neededItems.Count == 0) return false;
int itemsAmount = neededItems.Select(item => item.Amount).Sum();
if (itemsAmount < (amount == null ? neededItem._amount :
amount[i])) return false;}
return true;}
```

3.10.3 Скрипт ChestController

Скрипт ChestController відповідає за управління інвентарем скрині та його інтеграцію з системою інвентарю й інтерфейсом користувача (лістинг 3.6), забезпечуючи коректну взаємодію між скринею та іншими компонентами гри.

Лістинг 3.6 – Метод LoadChestInventoryToUI скрипта ChestController

```
public void LoadChestInventoryToUI()
{InventoryController.GetInstance().SetChestInventory(_inventory);
UIController.GetInstance().SwitchActiveChestMenu();    }
```

3.10.4 Скрипт InventoryController

Скрипт InventoryController є ключовим елементом для управління інвентарями в грі, забезпечуючи роботу з головним інвентарем, інвентарем гравця та інвентарем скрині. Він контролює всі операції, пов’язані з додаванням, переміщенням, видаленням та скиданням предметів, а також відповідає за зміну активного предмета, який тримає гравець. Скрипт реалізує синглтон, що дозволяє глобальний доступ до цього контролера.

В інвентарі гравця відображаються предмети, які можна обирати для використання, а головний інвентар та інвентар скрині підтримують більші обсяги зберігання предметів. Всі взаємодії між інвентарями реалізовані через модель переміщення предметів, що фіксує дані про осередок і тип інвентарю для обраного предмета. Контролер забезпечує переміщення предметів між комірками одного інвентаря або різних інвентарів, підтримуючи механіку стекування (об'єднання однакових предметів). Також скрипт дозволяє скидати предмети у світ гри, створюючи їх копію у вигляді шаблонів та розміщуючи біля руки гравця. Крім того, він може видаляти предмети з інвентарю, зокрема під час виходу з гри, щоб викликати їх деструкцію.

Зміна активного предмета забезпечує взаємодію гравця з інвентарем, дозволяючи перемикатися між комірками інвентаря гравця, а також оновлює

візуальне відображення через інтерфейс користувача (лістинг 3.7). Інтеграція з UI реалізована для оновлення відображення інвентарів після будь-яких змін, як-от переміщення чи видалення предметів.

Лістинг 3.7 – Метод ChangeActiveItem скрипта InventoryController

```
public void ChangeActiveItem(bool isPositiv = true, int index =
-1){if (!_isCanChangeActiveItem) return;
int oldActivePlayerItemIndex = _activePlayerItemIndex;
if (index == -1){
_activePlayerItemIndex += isPositiv ? 1 : -1;}
else{_activePlayerItemIndex = index;}
_activePlayerItemIndex = Mathf.Clamp(_activePlayerItemIndex,
MechConstants.MAX_ITEMS_IN_PLAYER - 1);
if(_activePlayerItemIndex != oldActivePlayerItemIndex)
ApplyActiveItem();}
```

Крім того, контролер підтримує додавання предметів до головного інвентаря, перевіряючи доступні комірки, і дозволяє зчитувати вміст різних інвентарів для подальшої взаємодії або виведення в UI. Взаємодія з гравцем реалізується через активний предмет, який зберігається та оновлюється відповідно до обраної комірки.

3.10.5 Скрипт ItemController

Скрипт ItemController керує поведінкою предмета в грі, використовуючи прив’язку до об'єкта, який представляє предмет. Він забезпечує різноманітні операції, такі як ініціалізація, оновлення, знищення (лістинг 3.8) та припинення роботи з предметом. Скрипт забезпечує відстеження стану предмета, а також здійснює його взаємодію з іншими системами гри.

Лістинг 3.8 – Метод ApplyItemBroke скрипта ItemController

```
private void ApplyItemBroke(){
if (!_itemObject.IsItemCountZero()) return;
InventoryController.GetInstance().RemoveItem();
Destroy(gameObject);}
```

3.10.6 Скрипт PlantController

Цей скрипт управляє рослинами в грі – від посадки до збору врожаю. Він слідкує за тим, чи треба рослину полити, чи не засохла вона, чи виросла, і чи була підживлена добривами (лістинг 3.9). Програма показує іконку, коли рослина хоче пити, запускає анімацію росту, змінює вигляд землі після удобрення та вираховує, скільки врожаю дасть рослина. Коли гравець поливає рослину, вона перестає "просити" воду і починає рости. Якщо довго не поливати – рослина засихає та змінює свій вигляд. Коли приходить час збирати врожай, гравець отримує врожай, а рослина зникає. Скрипт поєднує всі ці процеси разом, щоб гра працювала плавно і зрозуміло для гравця.

Лістинг 3.9 – Метод PatchHarvesting скрипта PlantController

```
public void PatchHarvesting()
{   if (_isPlantGrow)
    { RuntimeManager.PlayOneShot(FMODEvents.instance.Harvest,
transform.position);
InventoryController.GetInstance().AddItemToMainInventory(_harves
t.Copy(), _harvestAmount);
        Destroy(transform.parent.gameObject);    }
    if (_isPlantDry)
    {  Destroy(transform.parent.gameObject);    }}
```

3.10.7 Скрипт PlayerController

Скрипт PlayerController керує поведінкою персонажа в грі, зокрема його рухами, взаємодією з об'єктами, управлінням інвентарем та використанням предметів. Він має багато змінних, що зберігають інформацію про стан руху, обертання, утримувані об'єкти та гроші. Важливими компонентами є CharacterController, який використовується для керування рухами персонажа (лістинг 3.10) та різні змінні для обробки об'єктів, які персонаж може піднімати або відпускати. Скрипт обробляє різні аспекти гри, зокрема рухи персонажа (за допомогою введення з клавіатури або контролера), управління гравцем у повітрі (гравітація, стрибки), а також

утримання і переміщення об'єктів. Якщо персонаж тримає об'єкт, скрипт дозволяє його переміщати або скинути, а також визначає, чи можна піднімати нові об'єкти залежно від їх маси.

Крім того, скрипт дозволяє переміщати та використовувати предмети з інвентаря, змінюючи активний предмет у руці персонажа. Перемикання між можливістю рухатися, обертатися та використовувати предмети контролюється через спеціальні методи. Скрипт також взаємодіє з інвентарем, додаючи або видаляючи предмети залежно від дій персонажа.

Лістинг 3.10 – Метод ApplyMovement скрипта PlayerController

```
private void ApplyMovement(){
if (!_isCanMoving) {
_inputMovement = Vector2.zero;
return;}
Vector3 direction = (_inputMovement.y * transform.forward) +
(_inputMovement.x * transform.right);
_direction.x = direction.x;
_direction.z = direction.z;
_chController.Move(_direction * PlayerConstants.MOVEMENT_SPEED *
Time.deltaTime);}
```

3.10.8 Скрипт QuestSystemController

Скрипт QuestSystemController відповідає за управління системою квестів у грі, включаючи ініціалізацію, оновлення та перевірку виконання квестів. Він використовує дві основні колекції для квестів: денну колекцію (квести, доступні для виконання протягом дня) та колекцію квестів гравця (прогрес гравця у виконанні завдань). Під час запуску гри денна колекція ініціалізується випадковими квестами з загальної бази (лістинг 3.11).

Скрипт також взаємодіє з інвентарем гравця, перевіряючи, чи є у нього достатньо необхідних предметів для завершення конкретного квесту. Метод перевірки використовує LINQ для пошуку відповідних предметів у колекції інвентаря, порівнюючи їх за кількістю та ідентифікаторами.

Лістинг 3.11 – Метод InitializeDayQuests скрипта QuestSystemController

```
public void InitializeDayQuests(){
_dayContainer.Container.Clear();
for (int i = 0; i < MechConstants.MAX_DAY_QUEST; i++){
bool questIsAdded = false;
while (!questIsAdded){
int indexQuest = Random.Range(0, _collection.Quests.Count);
questIsAdded =
_dayContainer.AddQuest(_collection.Quests[indexQuest]);}}}
```

Крім того, через взаємодію з інтерфейсом користувача скрипт відповідає за відображення активних квестів у відповідних меню. Він гарантує, що денні квести оновлюються на початку кожного дня і що список прогресу гравця доступний для перегляду.

3.10.9 Скрипт ShopController

Скрипт ShopController реалізує функціонал магазину в грі, дозволяючи гравцям купувати та продавати предмети. Магазин має два основні списки: повний список товарів, доступний для продажу та список товарів дня, який оновлюється щодня або залежно від налаштувань. Під час старту гри товари ініціалізуються з базових даних, включаючи їх кількість та ціну. Гравець може купувати товари з денного асортименту, перевіряючи наявність достатньої кількості грошей (лістинг 3.12), а також продавати предмети з інвентаря. Для управління продажем використовується словник, який зберігає товари, доступні для продажу, їх кількість та ціну. Магазин інтегрований з інвентарем та інтерфейсом користувача, що дозволяє оновлювати список доступних товарів у режимі реального часу та відображати їх у відповідному UI.

Лістинг 3.12 – Метод BuyItemFromShop скрипта ShopController

```
public void BuyItemFromShop(int index){
if (PlayerController.GetInstance().Money <
_goodsForDay[index].Price) return;
_goodsForDay[index].Count--;
```

```
PlayerController.GetInstance().Money -=
_goodsForDay[index].Price;
Item soldGoods = _goodsForDay[index].Goods.Copy();
soldGoods.Init();
InventoryController.GetInstance().AddItemToMainInventory(soldGoo
ds, 1);
if (_goodsForDay[index].Count == 0) _goodsForDay[index] = null;
LoadGoodsForDayToUI();}
```

3.10.10 Скрипт TreeController

Скрипт TreeController керує деревом у грі, яке може рости, давати плоди, потребувати поливу, засихати або бути зрубаним.

Програма відстежує різні стани дерева: чи хоче воно пити, чи повністю виросло, чи з'явилися плоди, скільки врожаїв уже зібрано та скільки ударів нанесено для рубання (лістинг 3.13). Коли дерево довго не поливають, запускається таймер, після якого воно помирає – стає чорним, анімації зупиняються та дерево перестає працювати. При поливі молодого дерева запускається анімація росту, а якщо дерево вже доросле – почне з'являтися урожай. Коли плоди дозріли, гравець може зібрати випадкову кількість фруктів, які потрапляють в інвентар. Після певної кількості зборів дерево вмирає. Також можна зрубати дерево – кожен удар рахується, і коли набирається потрібна кількість, дерево зникає, а гравець отримує дрова.

Скрипт показує іконку, коли дерево потребує води, керує анімаціями через спеціальні компоненти і взаємодіє з системою інвентаря. Це створює реалістичну поведінку дерева, з яким можна по-різному взаємодіяти в грі.

Лістинг 3.13 – Метод TreeHarvesting скрипта TreeController

```
public void TreeHarvesting()
{   if (_isFruitGrow)
    {   int fruitCount =
Random.Range(MechConstants.MIN_TREE_HARVEST,
    MechConstants.MAX_TREE_HARVEST);
        _collectTimes += CollectFruit();
        if (_collectTimes == MechConstants.MAX_COUNT_OF_HARVEST)
        {TreeDead();}
InventoryController.GetInstance().AddItemToMainInventory(_harves
t.Copy(), fruitCount);    }}
```

3.10.11 Скрипт UIController

Скрипт UIController відповідає за управління інтерфейсом користувача в грі, забезпечуючи функціонал для різних меню, таких як інвентар, магазин, квести, ремесло та взаємодія зі скринями. У ньому реалізовано логіку для оновлення текстової інформації, включаючи час, кількість грошей і квестів, а також відображення прогрес-бару з можливістю виконання дії після завершення таймера (лістинг 3.14).

Скрипт дозволяє перемикати активний стан різних елементів інтерфейсу, оновлювати їхній вміст та керувати інтерактивними елементами, такими як списки предметів у магазині чи списки квестів. Крім того, тут реалізовано функцію закріплення предметів на курсорі миші, а також вибір клітинок інвентарю гравця із зміною кольору під час вибору. Основна логіка оновлюється щокадрово, включаючи функції для руху курсора, оновлення текстових полів та контролю стану прогрес-бару. Завдяки цьому забезпечується інтерактивність та динамічність взаємодії з користувачем у реальному часі.

Лістинг 3.14 – Метод UpdateProgressBar скрипта UIController

```
private void UpdateProgressBar()
{
if (PlayerInputSystem.holdingLMB && _usingProgressBar){
_indicatorTimer += Time.deltaTime / _timerMultiple;
radialIndicatorUI.enabled = true;
radialIndicatorUI.fillAmount = _indicatorTimer;
if (_indicatorTimer >= _maxIndicatorTimer){
_indicatorTimer = 0.0f;
radialIndicatorUI.fillAmount = 0.0f;
eventProgressBar.Invoke();
}}
else {
_indicatorTimer = 0.0f;
radialIndicatorUI.fillAmount = 0.0f;
radialIndicatorUI.enabled = false;
_usingProgressBar = false;
eventProgressBar.RemoveAllListeners();}
}
```

3.10.12 Скрипт WorldController

Скрипт WorldController відповідає за динамічну зміну освітлення та часу доби в ігровому світі. Він автоматично оновлює візуальні параметри, такі як кольори освітлення, туману та напрямок основного джерела світла, залежно від плину часу (лістинг 3.15). Система реалізує плавну зміну часу і відображає перехід між різними фазами доби.

Лістинг 3.15 – Метод UpdateLighting скрипта WorldController

```
private void UpdateLighting(float timePercent){
RenderSettings.ambientLight =
_preset._ambientColor.Evaluate(timePercent);RenderSettings.fogCo
lor = _preset._forColor.Evaluate(timePercent);
if(_directionalLight != null){
_directionalLight.color =
_preset._directionalColor.Evaluate(timePercent);_directionalLigh
t.transform.localRotation = Quaternion.Euler(new
Vector3((timePercent * 360f) - 100f, 170, 0));}}
```

4 ІНТСРУКЦІЯ КОРИСТУВАЧА

Після заходу в гру, користувачу відкриється головне меню гри (рисунок 4.1), де він може почати ігровий процес, налаштувати гру або вийти з гри, щоб закінчити сесію. У випадку, якщо гравець натиснув на кнопку «settings» то йому відкривається меню налаштування (рисунок 4.2). В данному меню користувач може налаштувати звук, чутливість миші (швидкість миші), а також поле зору (налаштування камери).

Рисунок 4.1 – Головне меню гри

Рисунок 4.2 – Меню налаштування гри

4.1 Основний ігровий процес

Як тільки гравець натисне на кнопку «Play» він переміститься на ігрову сцену, де й почнеться основний ігровий процес. Користувач опиняється в стартовому будинку, де він бачить ігровий інтерфейс (рисуннок 4.3):

- знизу-посередині: інвентар швидкого доступу;
- знизу-зліва: кількість грошей та час дня;
- зверху-зліва: версія гри та підказки з управління;
- зверху-справа: кількість FPS.

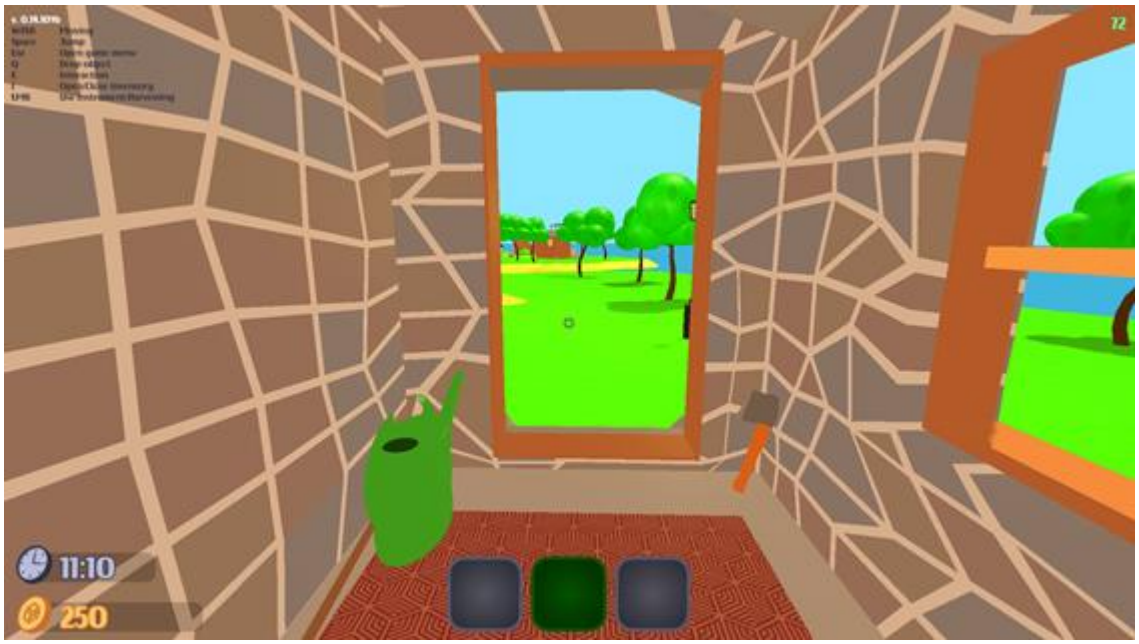


Рисунок 4.3 – Статовий будинок та ігровий інтерфейс

Всередині будинку гравець може знайти перші інтрументи: лійка, сапка та сокира. Якщо направити камеру в сторону інтрументу (або будь-якого об’єкту з яким можна взаємодіяти) посеред екрану напише назву предмету (або дію, яку треба зробити), кнопку на клавіатурі або миші, яку треба натиснути, щоб взаємодіяти з об’єктом, рівень об’єкту (якщо це інтрумент) та його кількість (рисунок 4.4). Інформація зникає автоматично, щойно гравець відводить камеру від об’єкта, що дозволяє не перевантажувати інтерфейс та зберігати увагу на геймплеї.

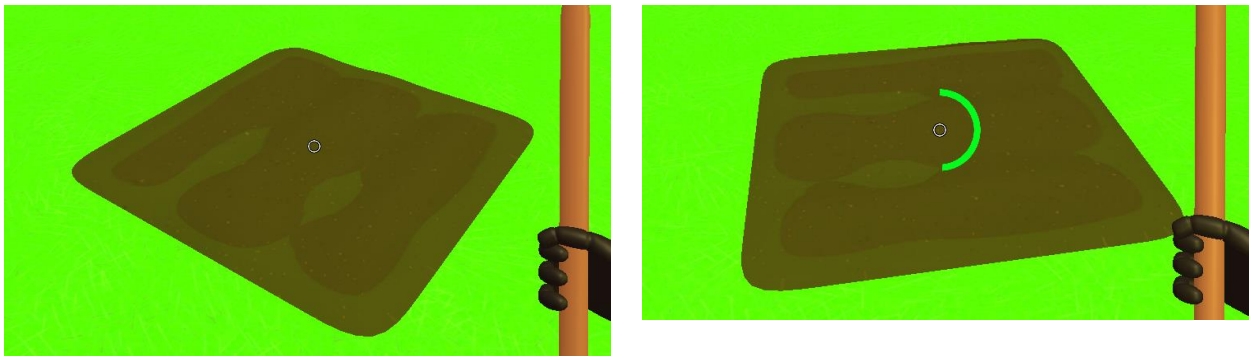
Рисунок 4.4 – Інформація про предмет

Після того як предмет буде підібрано, він опиняється в інвентарі, який можна відкрити за допомогою клавіші «I» (рисунок 4.5). Слід зазначити, що в інвентарі, гравець бачить усі свої предмети. Він їх може переміщати між комірчин, побачити інформацію про них, або перемістити в інвентар швидкого доступу. Коли предмет перемістили в активну комірчину інвентаря швидкого доступу (зелена комерчина), то він одразу береться в руки (рисунок 4.6), що дозволяє використовувати його, якщо це можливо. Щоб перемикатися між комірчинами інвентаря швидкого доступу, треба використати колесо миші, або скористатися клавішами клавіатури, а саме цифрами від 1 до 3.

Рисунок 4.5 – Інвентар

Рисунок 4.6 – Взяття предмету в активну комірчину

Основна ціль гри – вирощування врожаю, продажа його, отримання прибутку та збільшення своєї ферми. Щоб почати це робити, треба скористатися всіма інструментами, що є в грі. За допомоги сапи, гравець може робити грядки. Навівши камеру на землю, тримаючи сапу, на землі з’явиться прозора грядка, яка вказує де її можна розмістити (рисунок 4.7 а). Якщо грядка приймає червоного віддінку то значить на цьому місці її розмістити не можна, й треба її перемістити. Якщо нічого не заважає її розмістити, то зашавши LMB, через певний час зробиться грядка (рисунок 4.7 б).



а) б)

Рисунок 4.7 – Робота з грядками: а) візуалізація; б) процес створення

Для того щоб почати вирощувати рослини, треба сходити до центра мапи, де будуть знаходитися магазини. Серед них буде магазин, що продає насіння (рисунок 4.8). При взаємодії з ним, відкривається меню магазину, де ми можемо покупати та продавати предмети (рисунок 4.9). У продавця товар змінюється кожний новий день. У меню магазину, можна побачити що він продає, скільки воно коштує та яка кількість товару в нього доступна. Якщо в гравця достатньо грошей, то він може купити необхідний предмет.

Рисунок 4.8 – Зовнішний вигляд магазину

Рисунок 4.9 – Меню магазину

Щоб посадити насіння, треба натиснути LMB по грядці тримаючи насіння. У кожної рослини є 5 етапів росту (рисунок 4.10), які змінюються через певний час. На певних етапах росту, рослину треба полити. Коли це необхідно, то над грядкою з’являється значок води (рисунок 4.11). Поки рослина вимагає води, вона рости не буде. Якщо за певний проміжок часу не полити рослину, то вона почорніє, й не дасть врожай (рисунок 4.12).

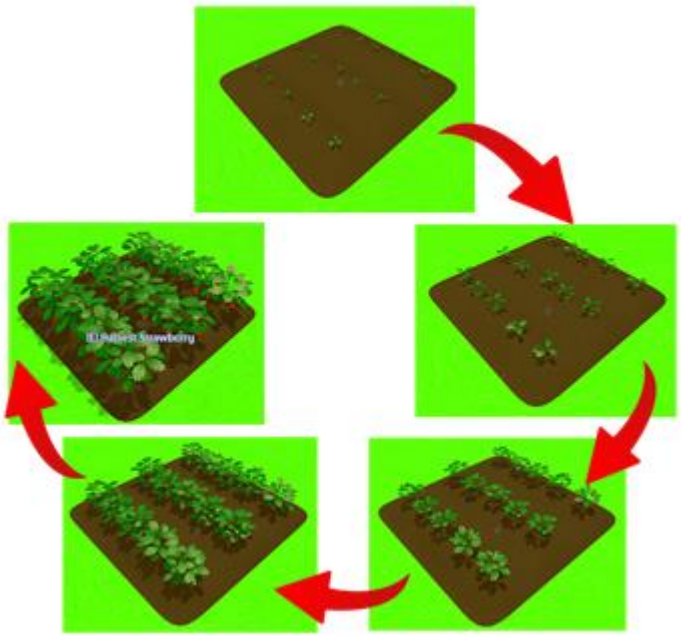


Рисунок 4.10 – Етапи росту певної рослини

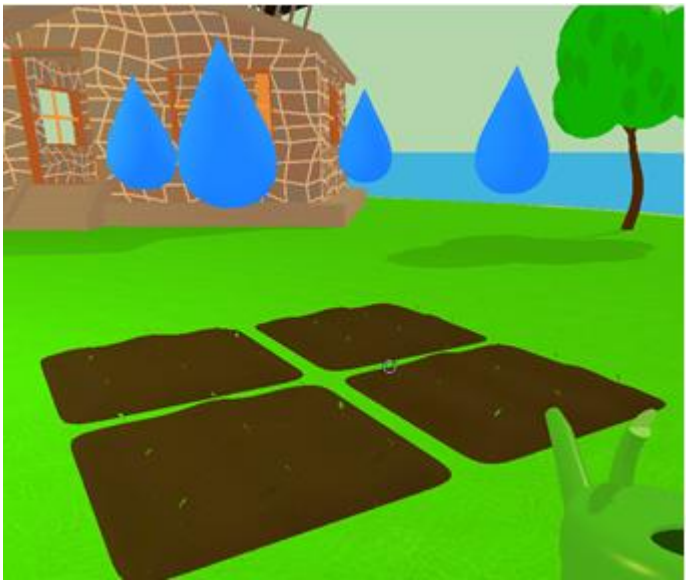


Рисунок 4.11 – Необхідність полити рослину

Рисунок 4.12 – Мертва рослина

Щоб поливати рослину, треба скористатися лійкою. В лійці обмежена кількість води, тому щоб її поновити, треба підійти до криниці.

Коли рослина помирає її необхідно прибрати. Для цього треба підійти до грядки та натиснути на кнопку «E», щоб прибрати її. Теж саме треба зробити, коли рослинан повністю виросла (рисунок 4.13). Натиснувши клавішу «E», грядка зникне, а врожай опиниться в інвентарі.

Рисунок 4.13 – Рослина, що виросла

Як тільки у гравця в інвентарі опиниться предмет, який можна продати, в магазині, в кладці «Sell» з’явиться предмети на продаж (рисунок 4.14). В магазині показується загальна кількість предметів, які можна продати, а

також дають вибір бажаної кількості предметів на продаж: 1, 10 та усі. Після продажу усіх предметів, вони зникають з магазину.

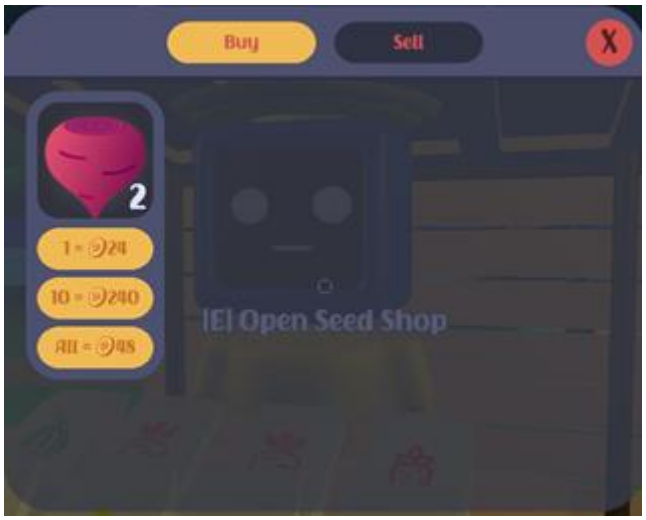


Рисунок 4.14 – Продаж предметів

Ще один тип рослин, які можна вирощувати – це дерева. Щоб їх посадити треба взяти паросто в руки та навестись на землю (рисунок 4.15). Якщо візуалізація дерева не червона, то значить його можна посадити на цьому місці.

Рисунок 4.15 – Візуалізація дерева

Дерева ростуть довше за інші рослини, а також їх треба поливати частіше. Проте вони дають врожай 5 разів за своє життя (рисунок 4.16). Коли всі врожаї зібрані, або якщо дерево вчасно не полити, воно помирає та стає чорним, й щоб його прибрати треба скористатися сокирою (рисунок 4.17). Утримуючи LMB, тримаючи сокиру в руках, дерево почне рубатися й після певної кількості ударів воно зникне, а в інвентарі з’являться колоди.

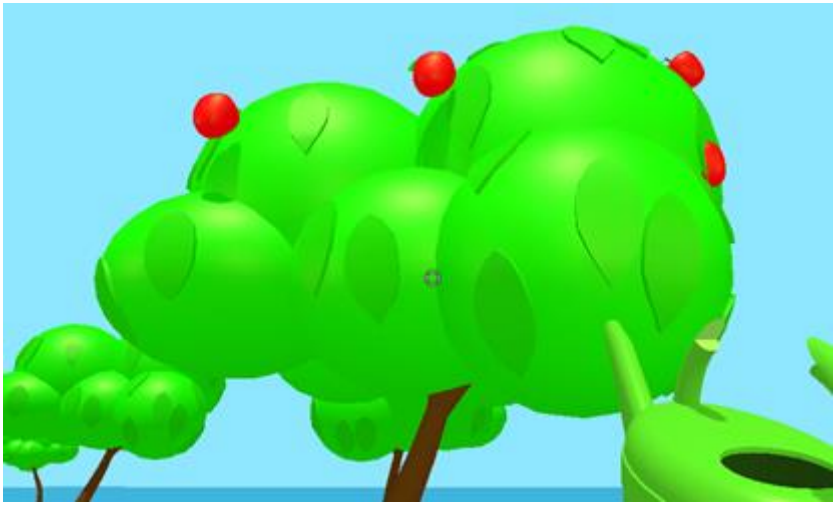


Рисунок 4.16 – Ріст плодів на дереві

Рисунок 4.17 – Мертве дерево

4.2 Інструменти для гри

В грі присутні три типи інтрументів: сапа, лійка та сокира. У кожного з інструментів є свої певні характеристики. У сапи це прочність та час викопування грядки. В сокири – прочність, час рубки, кількість ударів для зрублення дерева. В лійки – прочність та ємність води. Значення цих характеристик залежить від рівня ісінтрумента. Так на першому рівні вони найгірші, а на 5 – найліпші. Рівень інструмента можна дізнатися за кількістю зірок біля іконки. Також під іконкою можна помітити різні шкали, які вказують на прочність що залишилася в інструмента (або кілкість наявної води, якщо це лійка). Всі інструменти можна подивитися та купити в окремому магазині (рисунок 4.18). Чим вищий рівень, тим дорожчий буде інтрумент.

Рисунок 4.18 – Магазин інструментів

4.3 Ігрові квести

Підійшовши до поштового ящика, гравець може відкрити меню квестів. В даному меню, будуть квести, суть яких зібрати певну кількість предметів. Коли гравець їх всі збирає, то отримає нагороду у вигляді грошей. При

натисканні на квест, внизу меню можна буде побачити інформацію про нього: кількість предметів необіхних для закінчення квесту, а також нагорода (рисунок 4.19).

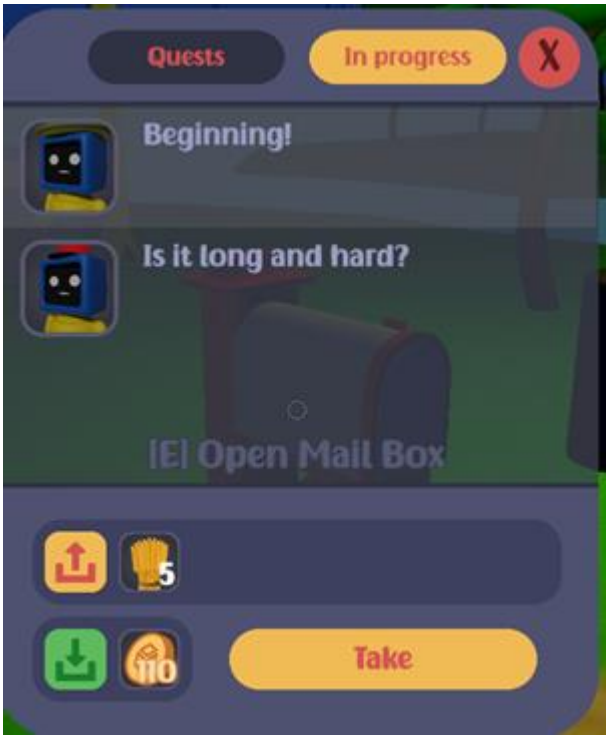


Рисунок 4.19 – Меню квестів

Всього гравець може взяти до 5 квестів. Щоб подивитися квести, які він взяв, треба відкрити вкладку «In Progress», де буде показані всі взяті квести, а також кількість взятих квестів. Коли гравець виконує квест, то в нього з’являється місце щоб взяти ще один. Квести оновлюються на початку дня, але якщо вони були взяті на виконання, то вони залишаться у гравця, поки він їх не виконає.

4.4 Будівлі та крафт

У спеціальному магазині, можна купити будівлі, які можуть переробити врожай, зібраний з грядок або дерева, на складніші продукти, які можна продати дорожче. Щоб розмістити будівлю треба помістити її в руки

та навестись на землю (рисунок 4.20). Якщо візуалізація не приймає червоного віддінку, то значить будівлю можна розмістити.

Рисунок 4.20 – Розміщення будівлі

Відкривши меню будівлі, можна побачити продукти, які можна скрафтити з врожаю, або інших предметів. Щоб зробити одну одиницю продукту, треба витратити певну кількість врожаю, вказаного у меню. Якщо в гравця достатньо врожаю, він зможе зробити новий продукт.

4.5 Використання неігрових персонажів (NPC)

Під час гри, можна побачити, що по мапі переміщаються неігрові персонажі. Також їх можна побачити біля магазинів. У кожного з не ігрових персонажів, є свій розклад дня: з 8 ранку до 8 вечора вони ходять по мапі або продають речі, а з 8 вечора й до 8 ранку – йдуть до дому. NPC можна взяти на руки, й віднести у будь-яке місце на мапі (рисунок 4.21). Коли гравець відпускає не ігрового персонажа, то він намагається повернутися на дорогу, щоб продовжити гуляти мапою. Якщо гравець взяв NPC, що стоїть біля магазину, то після того як його поставлять на землю він повернеться до магазину.

Якщо біля магазина не буде NPC, який стояв біля нього, то гравець нічого не зможе купити в ньому. Те саме стосується нічного часу, коли не ігровий персонаж йде до дома – магазин не працює. Проте, якщо гравець принесе NPC до свого магазину, то він зможе або щось купити чи продати.

Якщо віднести NPC у воду, він знищеться та з’явиться у себе дома. Після повторної появи, він знову буде переміщатися по мапі.

Рисунок 4.21 – Тримання NPC на руках

ВИСНОВОК

Дана кваліфікаційна робота присвячена створенню комп'ютерної гри в жанрі фермерського симулятора на основі ігрового движка Unity для операційної системи Windows. Дослідження включало весь спектр етапів ігрової розробки – починаючи з вивчення наявних продуктів на ринку та завершуючи впровадженням повнофункціонального ігрового прототипу.

Для реалізації проєкту було задіяно передові технології розробки: мову C#, платформу Unity, інтегроване середовище Visual Studio 2022, Git для версійного контролю та Unity Asset Store для отримання додаткових ресурсів. Проєкт включає основні ігрові системи симулятора: навігацію персонажа, управління інвентарем, механіку спорудження, систему рослинництва, комерційну взаємодію з NPC, квестову систему та додаткові геймплейні компоненти.

Структура проєкту грунтується на концепції Clean Architecture із широким використанням шаблонів проєктування та суворим дотриманням принципів SOLID, DRY та KISS. Такий підхід гарантує модульну будову, адаптивність та простоту майбутнього обслуговування й удосконалення системи. Особливу увагу було зосереджено на структуруванні програмного коду, розробці інтерфейсу користувача та оптимізації роботи з системними ресурсами.

Підсумком роботи стала надійна, масштабована ігрова платформа, здатна служити фундаментом для подальших удосконалень або комерційного використання.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. PC MAG: Definition of video game. URL: https://www.pcmag.com/ encyclopedia/term/video-game.

2. Slime Rancher в Steam. URL: https://store.steampowered.com/ app/433340/ Slime_Rancher.

3. Official Website: Farming Simulator. URL: https://www.farming-simulator.com/about.php.

4. Official Website: Stardew Valley. URL: https://www.stardewvalley.net.

5. Official Website: FarmVille. URL: https://www.farmville3.com.

6. KISS, DRY, S.O.L.I.D, YAGNI – навіщо дотримуватись принципів програмування? URL: https://senior.ua/articles/kiss-dry-solid-yagni--navscho-dotrimuvatis-principv-programuvannya.

7. Skeet J. C# in Depth. Publisher: Manning; 4th edition. 2019. 528 p.

8. Hocking J. Unity in Action. Multiplatform game development in C#. Publisher: Manning; 3e edition. 2022. 414 p.

9. Official Website: Git. URL: https://git-scm.com.

10. Official Website: Blender. URL: https://www.blender.org/.

11. Unity AssetStore. URL: https://assetstore.unity.com/.

12. Martin R., Henney K. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Publisher: Pearson; 1st edition, 2017. 432 p.

13. Патерни проектування. URL: https://refactoring.guru/uk/design-patterns.

14. Unity Documentation: MonoBehaviour. URL: https://docs.unity3d.com/ 6000.0/Documentation/ScriptReference/MonoBehaviour.html.

15. Unity Documentation: Meshes, Materials, Shaders and Textures. URL: https://docs.unity3d.com/ru/2019.4/Manual/Shaders.html.

16. Official Website: FMOD. URL: https://www.fmod.com/.