Міністерство освіти і науки України
Криворізький національний університет
Факультет інформаційних технологій
Кафедра автоматизації, комп'ютерних наук і технологій

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеню вищої освіти – магістр
за освітньо-професійною програмою
«Комп'ютерні науки»

зі спеціальності
122 – Комп'ютерні науки

тема роботи:

«Розробка дизайну та візуальної складової гри у жанрі «фермерський симулятор» з використанням рушія Unity»

Виконав студент гр. КН-24м. _____Гречко Є.В.

Керівник _____ Рубан С. А.

Нормоконтроль _____ Маринич І. А.

Завідувач кафедри _____ Рубан С. А.

Кривий Ріг – 2025

КРИВОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет: *інформаційних технологій*

Кафедра: *автоматизації, комп'ютерних наук і технологій*

Ступінь вищої освіти: *Магістр*

Спеціальність: *122 – Комп'ютерні науки*

ЗАТВЕРДЖУЮ

Зав. кафедри: *к.т.н. Рубан С.А.*

_____

« *15* » *травня* 2025 р.

ЗАВДАННЯ

на кваліфікаційну роботу магістра

студентові групи КН-24м *Гречко Єлизаветі Вадимівні*

1. Тема кваліфікаційної роботи: *«Розробка дизайну та візуальної складової гри у жанрі «фермерський симулятор» з використанням рушія Unity»*

затверджено наказом по університету № *257с* від *13.05.2025 р.*

2. Термін здачі кваліфікаційної роботи: *01.12.2025 р.*

3. Склад кваліфікаційної роботи: *Пояснювальна записка обсягом 83с., додатки, презентація у Microsoft PowerPoint (13 слайдів) в електронному та друкованому вигляді*

4. Консультанти кваліфікаційної роботи:

Розділ 1-3 доц. Рубан С. А.

_____

Нормоконтроль доц. Маринич І. А.

5. Календарний план:

№	Етапи роботи	Термін виконання
1	*Вступ*	*17.05.25*
2	*Розділ 1*	*15.07.25*
3	*Розділ 2*	*18.09.25*
4	*Розділ 3*	*19.11.25*
5	*Висновки*	*20.11.25*
6	*Оформлення кваліфікаційної роботи*	*25.11.25*
7	*Підготовка презентації та графічного матеріалу*	*28.11.25*
8	*Підготовка доповіді до захисту*	*01.12.25*

6. Дата видачі завдання: *13.05.2025р.*

Керівник _____ / Рубан С. А./

7. Запевнення: *Я, Гречко Єлизавета Вадимівна, запевняю, що ця кваліфікаційна робота виконана самостійно, не містить академічного плагіату, фабрикації, фальсифікації. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.*

Із чинним Положенням про академічну доброчесність Криворізького національного університету ознайомлений.

Чітко усвідомлюю, що в разі виявлення у кваліфікаційній роботі умисних порушень робота не допускається до захисту або оцінюється незадовільно.

Здобувач _____ **/ Гречко Є.В./**

АНОТАЦІЯ

Гречко Є. В. Розробка дизайну та візуальної складової гри у жанрі «фермерський симулятор» з використанням рушія Unity: кваліфікаційна робота магістра: 122 – Комп’ютерні науки. Кривий Ріг. Криворізький національний університет, 2025. 83 с.

Об’єктом дослідження є процес розробки та програмної реалізації візуальної складової та ігрового середовища комп’ютерних ігор.

У першому розділі проведено аналіз сучасного стану ігрової індустрії та жанру «фермерський симулятор», досліджено психологічні аспекти впливу візуальної стилістики на гравця, а також обґрунтовано вибір рушія Unity для реалізації проєкту.

В другому розділі розроблено загальну концепцію та візуальний стиль гри (естетика «затишних ігор»), створено графічні ресурси (середовище, персонажі-коти, архітектура) та спроєктовано дизайн інтерфейсу користувача.

В третьому розділі виконано програмну реалізацію ігрових механік мовою C#: створено архітектуру менеджерів гри, реалізовано систему грядок та росту рослин, налаштовано фізику руху персонажа, систему інвентаря, збереження даних та інтеграцію UI.

Ключові слова:

UNITY, C#, ФЕРМЕРСЬКИЙ СИМУЛЯТОР, ГЕЙМДИЗАЙН, АРХІТЕКТУРА ГРИ, SINGLETON, JSON.

ANNOTATION

Hrechko. Y.V. Development of design and visual component of a game in the «farming simulator» genre using the Unity engine: Master's Qualification Thesis: 122 – Computer Science. Kryvyi Rih. Kryvyi Rih National University, 2025. 83 p.

The object of the study is the process of development and software implementation of the visual component and game environment of computer games.

In the first section, the current state of the game industry and the "farming simulator" genre is analyzed, the psychological aspects of the visual style's impact on the

player are investigated, and the choice of the Unity engine for the project implementation is substantiated.

In the second section, the general concept and visual style of the game ("cozy games" aesthetics) are developed, graphic resources (environment, cat characters, architecture) are created, and the user interface design is projected.

In the third section, the software implementation of game mechanics in C# is performed: the architecture of game managers is created, the system of garden beds and plant growth is implemented, character movement physics, inventory system, data saving, and UI integration are configured.

Keywords:

UNITY, C#, FARMING SIMULATOR, GAME DESIGN, GAME ARCHITECTURE, SINGLETON, JSON.

ЗМІСТ

ВСТУП

У сучасній індустрії ігровий простір стрімко розвивається, трансформуючись із засобу розваги у повноцінний цифровий продукт, що поєднує технології, творчість та психологічний вплив. Серед різноманіття жанрів стабільно високу популярність утримують «фермерські симулятори», які дозволяють користувачеві зануритися у медитативний процес керування господарством . В умовах сьогодення, коли зростає потреба у психологічному розвантаженні, візуальна складова таких ігор набуває критичного значення, формуючи атмосферу затишку та безпеки . Однак створення якісного візуального продукту вимагає не лише художнього бачення, а й глибокого розуміння технічних можливостей сучасних інструментів, зокрема рушія Unity. Необхідність поєднання естетичних вимог жанру з технічною оптимізацією та ергономікою інтерфейсу становить складну проблему, що й обумовило вибір теми магістерської роботи: «Розробка дизайну та візуальної складової гри у жанрі «фермерський симулятор» з використанням рушія Unity».

Питання комп’ютерних ігор як об’єкта дослідження розглядаються у працях багатьох науковців, які аналізують історію та культурологічне значення відеоігор, а специфіка жанрів та їх монетизація висвітлені у загальних оглядах індустрії. Окрему увагу дослідники приділяють візуальній стилістиці та психології сприйняття гравця, зокрема впливу художніх стилів на залученість та естетиці «затишних ігор» як окремого феномену. Технічні аспекти реалізації графіки в Unity, включаючи оптимізацію та анімацію, детально описані в документації та працях спільноти розробників. Проте, незважаючи на наявність ґрунтовних праць з окремих аспектів геймдеву, питання комплексної розробки фермерського симулятора, що поєднує специфічну «милу» стилістику, психологічні аспекти утримання гравця та технічну реалізацію на Unity, розкрито недостатньо і потребує подальшого практичного дослідження.

Метою роботи є розробка цілісної концепції дизайну та програмна реалізація візуальної складової комп’ютерної гри у жанрі «фермерський симулятор» засобами

рушія Unity. Для досягнення поставленої мети необхідно проаналізувати особливості жанру та визначити вимоги до його візуального оформлення; дослідити принципи візуального дизайну та UI/UX для створення емоційно комфортного середовища; охарактеризувати інструментарій Unity для роботи з графічними ресурсами та анімацією; розробити концепцію проєкту, створити графічні ресурси (середовище, персонажі) та спроєктувати інтерфейс користувача; виконати програмну реалізацію ключових механік та інтегрувати візуальні компоненти в ігровий процес.

Об’єктом дослідження є процес розробки візуальної складової та ігрового середовища комп’ютерних ігор. Предметом дослідження є методи, засоби та інструменти рушія Unity, що застосовуються для створення графічного стилю, анімації та інтерфейсу гри жанру «фермерський симулятор».

Практичне значення одержаних результатів полягає у тому, що розроблено та програмно реалізовано діючий прототип гри, який включає унікальну візуальну концепцію з використанням затишної стилістики та персонажів-котів, оптимізовану систему рендерингу середовища (тайлова система) та анімації, архітектуру ігрових менеджерів та систему збереження даних, а також адаптивний інтерфейс користувача, інтегрований з Figma в Unity. Результати роботи можуть бути використані як база для створення повноцінного комерційного продукту.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ОСНОВИ РОЗРОБКИ ІГОР У ЖАНРІ «ФЕРМЕРСЬКИЙ СИМУЛЯТОР»

1.1 Комп’ютерні ігри як об’єкт дослідження

Перші комп’ютерні ігри почали з’являтися ще в 1950-х роках, серед них електромеханічна розробка Tennis for Two (рис. 1.1). Уже в 1970-х сформувався напрям відеоігор, який представили аркадні хіти такі як Pac-Man (рис. 1.2) чи Space Invaders (рис. 1.3). Наступне десятиліття ознаменувалося появою ігрових консолей, що дало змогу значно розширити коло гравців і відкрило нові можливості для розвитку індустрії. Відтоді відеоігри почали швидко еволюціонувати – з’являлися нові жанри, удосконалювалися технології, а сама індустрія перетворювалася на масштабне культурне й технологічне явище. [1]

Рисунок 1.1 – Зображення електромеханічної розробки Tennis for Two

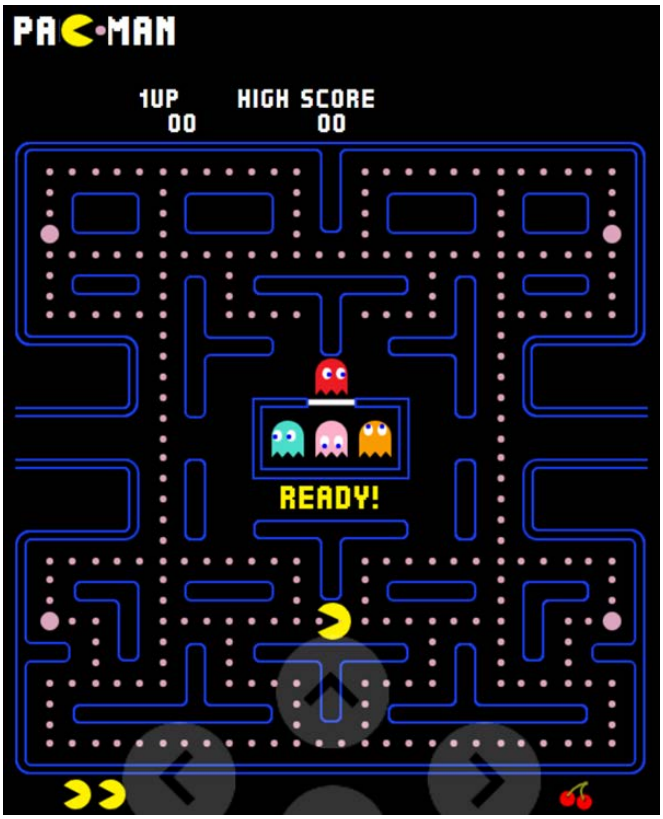

Рисунок 1.2 – Зображення екрану гри Pac-Man

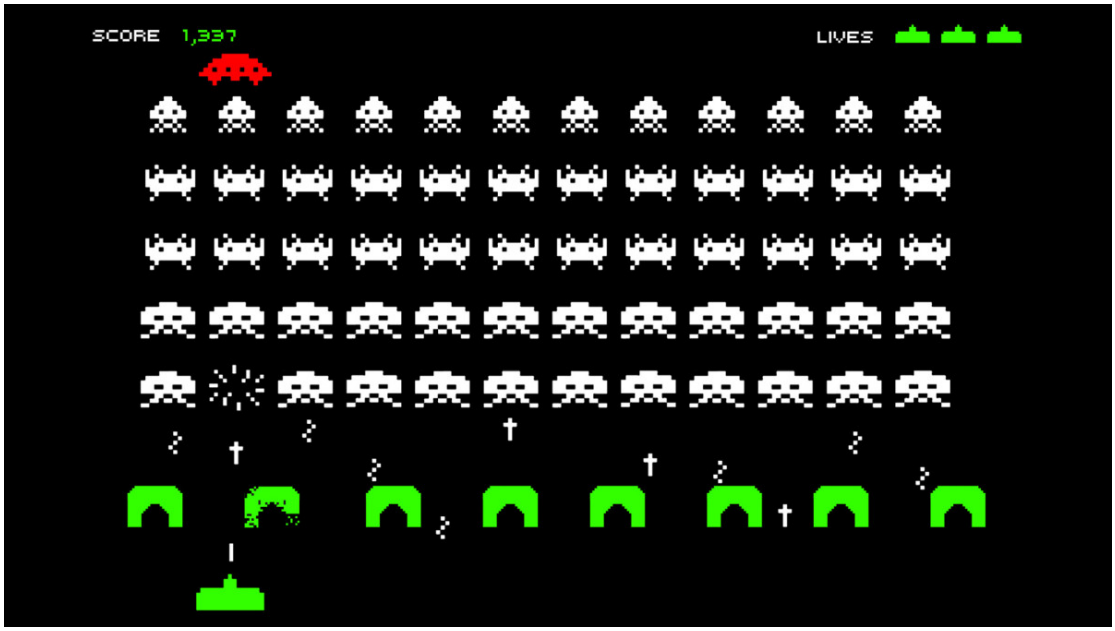

Рисунок 1.3 – Зображення екрану гри Space Invaders

Комп’ютерні ігри сьогодні можна розглядати не лише як розвагу, а й як складний продукт, у створенні якого поєднуються технології та творчість. Їхня розробка проходить кілька етапів – від планування та дизайну до програмування,

балансування й оптимізації. Для цього використовують сучасні рушії, такі як Unity чи Unreal Engine, різні мови програмування (C++, C#, Java, Python), а також інструменти для створення 3D-графіки, наприклад Blender або Maya. Така технічна багатогранність робить ігри не просто програмним продуктом, а явищем, яке впливає на культуру та суспільство. Саме тому вони все частіше стають об'єктом наукових досліджень у культурології, соціології, психології та педагогіці. В межах цього інтересу сформувався окремий напрям «Game studies», що вивчає історію ігор, їхні механіки, соціокультурне значення та вплив на користувачів, підтверджуючи важливість комп'ютерних ігор як повноцінного об'єкта дослідження. [2]

Геймдизайнери у цьому процесі відіграють ключову роль: вони формують концепцію гри, створюють ігрові механіки, визначають баланс і способи взаємодії персонажів. Але результат їхньої роботи можливий лише завдяки співпраці з програмістами, художниками, звуковими дизайнерами та тестувальниками, які разом перетворюють ідею на завершений продукт. На різних етапах команда створює прототипи, проводить бета-тестування, готує гру до релізу й обов'язково враховує особливості тієї платформи, для якої вона призначена – ПК, мобільних пристроїв чи консолей. Саме ця колективна та багаторівнева праця робить комп'ютерні ігри настільки комплексним явищем, що їх неможливо вивчати лише з технічної чи культурної точки зору окремо. [3]

Важливим аспектом дослідження також є жанрова різноманітність, адже саме вона відображає як підхід розробників до створення ігрового досвіду, так і очікування гравців. Класифікація за жанрами ґрунтується на типі геймплею, діях і цілях користувача. Традиційно виділяють шутери від першої чи третьої особи, рольові ігри (RPG), стратегії, платформери, пригодницькі ігри, симулятори, що відтворюють реальні процеси (керування транспортом, будівництво, моделювання життя), а також файтинги, головоломки, спортивні ігри і багато інших напрямів. Водночас ця система не є сталою – жанри постійно трансформуються та розширюються разом із розвитком індустрії. [4]

Окремої уваги заслуговують симулятори, адже їхньою головною метою є максимальне реалістичне відтворення умов та процесів з реального життя. Це може бути керування автомобілем чи літаком, управління містом, фермою або навіть польотом космічного апарата. Такі ігри поєднують розважальний і прикладний характер: вони не лише захоплюють, а й можуть використовуватися як навчальні чи професійні інструменти. Симулятори сприяють розвитку аналітичного мислення, уміння приймати рішення й вибудовувати стратегії, а в освітньому середовищі застосовуються спеціалізовані програми для відпрацювання практичних навичок, наприклад Simsoft (рис. 1.4) для менеджменту проєктів чи підготовки інженерів. [4]

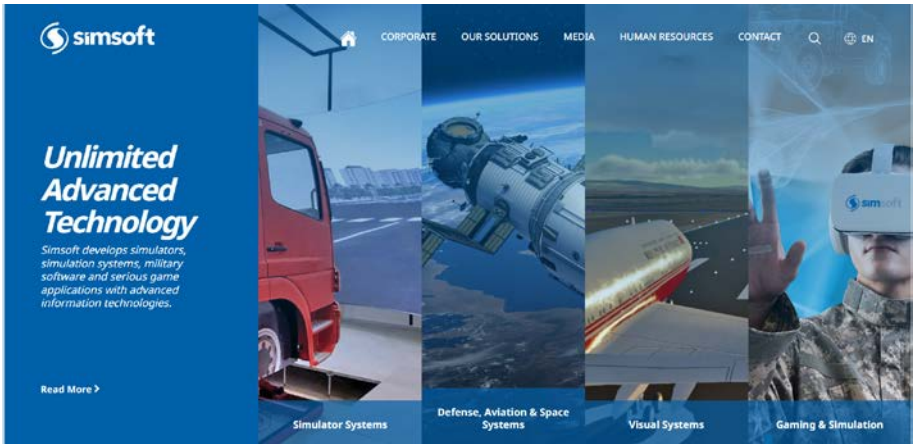

Рисунок 1.4 – Головний екран сайту Simsoft

1.2 Жанр «фермерський симулятор»

Жанр «фермерський симулятор» належить до піджанрів ігор-симуляторів, основною метою яких є управління фермою та здійснення різних видів сільськогосподарської діяльності. Ці ігри відтворюють процеси вирощування культур, догляду за тваринами, обробки землі та реалізації продукції. Найвідомішим представником жанру вважається серія Farming Simulator, що стала орієнтиром для подальшого розвитку подібних проєктів.

Поява фермерських симуляторів припадає на початок 2000-х років, однак справжнього поширення жанр набув після виходу серії Farming Simulator,

розробленої компанією Giants Software у 2008 році. З того часу гра регулярно отримувала нові версії (окрім 2021 року), кожна з яких розширювала функціонал, удосконалювала графіку, ускладнювала ігрові механіки та збільшувала масштаби відкритого світу. Особливу увагу розробники приділяли реалізму – у грі з'явилася ліцензована сільськогосподарська техніка, точні фізичні моделі ґрунту та руху машин, а також економічні системи, що моделюють діяльність реального фермерського господарства. [5]

Основною характеристикою фермерських симуляторів є високий рівень деталізації та орієнтація на реалістичне відтворення процесів. Гравець має змогу керувати різноманітною технікою, вирощувати культури, займатися тваринництвом, лісозаготівлею чи переробкою сировини. В іграх цього жанру, як правило, відсутня чітко визначена кінцева мета – користувач самостійно формує власні завдання, спрямовані на розвиток ферми, збільшення прибутку або розширення виробництва. Важливою особливістю є відкритий світ, який сприяє зануренню у процес гри та формує відчуття свободи дій. [5, 6]

Серія Farming Simulator (рис. 1.5) вирізняється підтримкою активної спільноти модифікацій: користувачі можуть створювати власні карти, об'єкти чи транспорт, що істотно розширює ігрові можливості. Використання реальних брендів сільськогосподарської техніки надає проєкту додаткової достовірності та підсилює ефект занурення. Такі особливості поєднують у собі технічну точність і креативний підхід до побудови ігрового середовища. [5]

Рисунок 1.5 – Зображення екрану гри Farming Simulator 25

Фермерські симулятори посідають помітне місце в сучасній ігровій індустрії завдяки поєднанню розважальної, стратегічної та навчальної складових. Вони приваблюють не лише геймерів, але й людей, які мають інтерес до сільського господарства або прагнуть розвинути навички планування й управління ресурсами. Такі ігри виконують і певну освітню функцію, оскільки дозволяють на практиці зрозуміти основи аграрних процесів, економічного мислення та організації виробництва.

Крім того, фермерські симулятори вирізняються спокійним темпом гри та медитативною атмосферою, що робить їх популярними серед користувачів, які шукають розслаблення або психологічне розвантаження. У цьому контексті жанр має і соціокультурне значення – він сприяє зниженню рівня стресу, розвитку концентрації уваги та формує відчуття стабільності.

Таким чином, жанр «фермерський симулятор» можна охарактеризувати як синтез технічної точності, естетичного комфорту та навчального потенціалу. Його привабливість полягає у можливості поєднати ігрову взаємодію з реалістичними механіками та візуальною гармонією, що забезпечує унікальний ігровий досвід та формує окрему нішу в сучасній культурі відеоігор.

1.3 Аналіз візуальної складової ігор

Візуальна складова відіграє одну з найважливіших ролей у сприйнятті гри. Саме через неї формується перше враження, настрій і глибина занурення у віртуальний світ. До неї входять графічний стиль, кольорова гама, анімація, освітлення, тіні, інтерфейс – усе, що створює відчуття атмосфери й цілісності. Візуальний стиль не обмежується «красивою картинкою» – він допомагає передати емоції, характер світу й навіть розповісти історію без слів. Кожен жанр і тип аудиторії має свої візуальні уподобання: наприклад, фотореалістична графіка (рис. 1.6) частіше використовується у симуляторах чи пригодницьких іграх, тоді як стилізовані (малюнкові, піксельні або мінімалістичні) підходять для більш експериментальних або казкових проєктів (рис. 1.6). [7]

Рисунок 1.6 – Зображення фотореалістичної та стилізованиї нрафіки на прикладі ігор Detroit become human та Terraria

Сучасна ігрова візуалізація базується переважно на тривимірній графіці, яка дає змогу створювати об’ємні простори, деталізовані об’єкти та складні світлові ефекти. Технологічні можливості таких движків, як Unreal Engine чи Unity, дозволяють розробникам будувати масштабні, живі світи з реалістичними матеріалами, погодними умовами та динамікою рухів. Анімація персонажів і навколишніх елементів робить ігровий простір переконливим і «живим». Водночас важливою частиною дизайну є оптимізація – навіть найкрасивіша гра не повинна втрачати у швидкодії, адже плавність процесу напряму впливає на комфорт гравця. [7]

Добре продуманий візуальний дизайн також допомагає орієнтуватися в інтерфейсі: гравець інтуїтивно розуміє, куди натиснути чи як взаємодіяти з елементами. Успішне поєднання художнього стилю, технічної реалізації та юзабіліті створює не просто «картинку», а повноцінний досвід, який підтримує сюжет, емоційний настрій і мотивацію до гри. [7]

1.4 Аналіз графічних особливостей відомих ігор жанру фермерський симулятор

Під час розробки власної гри важливо аналізувати вже наявні проєкти в межах обраного жанру. Це допомагає зрозуміти, які підходи до графіки, стилізації та подачі геймплею виявилися успішними, а також визначити, які рішення будуть найбільш доречними для майбутньої гри. Фермерські симулятори демонструють широкий спектр візуальних стилів – від максимально реалістичних до виразно стилізованих, тому вивчення цих прикладів дає можливість краще орієнтуватися у можливостях жанру.

Серія Farming Simulator від Giants Software (рис. 1.7) демонструє послідовну еволюцію графічної якості, зосереджену на фотореалізмі, деталізації техніки та природного середовища. Починаючи з FS17, де було зроблено помітний стрибок у якості текстур ґрунту, рослинності та фізики, кожна наступна частина суттєво розширювала візуальні можливості рушія. У FS19 це проявилося у впровадженні нового графічного ядра, детальніших ландшафтів і можливості модифікації рельєфу, тоді як FS22 запропонував покращений штучний інтелект, масштабні карти та нові алгоритми обробки ґрунту. Остання версія FS25 стала найтехнологічнішою, додавши оптимізацію під апаратні можливості нового покоління, реактивні на дії гравця поля, зношуваність техніки, реалістичне освітлення та динамічні погодні ефекти. Візуальна складова серії вирізняється високою деталізацією ліцензованої техніки зі складними анімаціями механізмів, реалістичною грою світла, відображеннями та поступовим забрудненням поверхонь. Середовище також постійно вдосконалюється: волюметричний туман, хмарні тіні, динамічна трава й культури, нічне небо зі зірками, а також багатошарова система погоди формують атмосферність симуляції. У FS25 вперше застосовано ray-traced освітлення, покращений HDR та сучасні методи згладжування. Попри наявні оптимізаційні рішення, такі як адаптивні LOD-моделі та спрощені геометрії рослинності, спільнота досі критикує деякі технічні вади, зокрема флікер текстур, поп-ін об’єктів і некоректну взаємодію опадів з будівлями.

Загалом графічний розвиток Farming Simulator спрямований не на досягнення рівня ААА-проєктів, а на підтримання функціональної реалістичності, де головний акцент зроблено на атмосферу сільського господарства, правдоподібність техніки та занурення гравця у процес фермерства.[8, 9]

Рисунок 1.7 – Приклад графіки у грі Farming Simulator

Stardew Valley (рис. 1.8) побудована на ретро-піксель-арті, що одразу формує теплу та ностальгічну атмосферу, характерну для класичних 2D RPG. Попри видиму простоту, візуальний стиль є глибоко продуманим: персонажі, рослини та інтер’єри створені вручну й вирізняються акуратною деталізацією, а мінімалістичні анімації залишаються достатньо плавними, щоб підтримувати інді-спрямованість гри та не відволікати гравця від процесу. Візуальна складова проєкту спирається на естетику 16-бітної ери SNES, але реалізована за допомогою сучасних 32-бітних можливостей, що забезпечує широку кольорову палітру та приємну глибину зображення. Спрайти варіюються від 32×32 до 128×128 пікселів, мають ручну анімацію з 4-8 кадрів та доповнюються паралакс-рухом фону, який створює відчуття багатошаровості середовища. Гра використовує цикли дня і ночі, сезонні палітри, погодні ефекти та локальні джерела світла, що формують живу атмосферу

без значного навантаження на систему, адже Stardew Valley працює навіть на дуже слабкому обладнанні. Інтерфейс також витриманий у мінімалістичному піксельному стилі з текстурами дерева, ретро-шрифтом та інвентарем, побудованим за принципом простої ґратки, що підсилює загальну естетику гри. Піксель-арт у Stardew Valley демонструє поєднання ретро-підходу та сучасної оптимізації: вручну намальовані елементи, плавність роботи на будь-якому пристрої та атмосферні візуальні ефекти роблять графіку гри впізнаваною, виразною та довговічною, попри її технічні обмеження. [10]

Рисунок 1.8 – Приклад графіки у грі Stardew Valley

Серія Harvest Moon (рис. 1.9), одна з перших у жанрі фермерських симуляторів, активно використовує мультиплікаційний, дружній для сприйняття стиль. Простота форм і яскрава кольорова гама створюють атмосферу затишку та легкості. Плавні рухи персонажів і впізнаваний дизайн містечок, полів і будівель роблять гру теплою й приємною, а також формують емоційний образ ідеалізованого сільського життя. [11]

Рисунок 1.9 – Приклад графіки у грі Harvest Moon

Coral Island (рис. 1.10) поєднує стилізований 3D-візуал із яскравими кольорами та динамічними анімаціями. Гра робить акцент на природі й океанічних середовищах: барвисті рослини, тварини та персонажі створюють атмосферу живого, дихаючого світу. Візуальний стиль позитивний і насичений, що підсилює відчуття пригод і дослідження, характерне для сучасних фермерських симуляторів з екологічним фокусом. [12]

Рисунок 1.10 – Приклад графіки у грі Coral Island

Різноманітність графічних рішень у жанрі фермерських симуляторів показує, наскільки по різному можуть виглядати ігри з подібною тематикою. Від піксельної естетики до фотореалістичних просторів – вибір стилю дозволяє орієнтуватися на різну аудиторію та підбирати візуальне подання, що найкраще відповідає задуму гри.

1.5 Психологічні аспекти залучення гравця

Психологічні аспекти залучення гравця відіграють ключову роль у проєктуванні фермерських симуляторів, оскільки саме вони визначають рівень мотивації, емоційної залученості та довгострокового повернення користувачів. Одним із важливих чинників є різниця між психотипами гравців. Відповідно до типології Річарда Бартла, користувачі можуть належати до дослідників, нагромаджувачів, кілерів або соціалістів. У контексті фермерських симуляторів найбільш активними виявляються нагромаджувачі, які отримують задоволення від накопичення ресурсів і поступового розвитку господарства, а також соціалісти, для яких важливими є елементи взаємодії та відчуття спільноти. Врахування потреб цих груп дозволяє розробникам підвищувати утримання гравців і забезпечувати стабільний інтерес до проєкту. [13]

Помітний вплив на мотивацію мають інвестиції гравця у власний прогрес. Розвиток ферми, будівництво споруд і поступове відкриття нових територій створюють відчуття досягнення, а потенційна втрата напрацьованого результату формує своєрідний психологічний бар’єр перед виходом із гри. У цьому контексті особливо ефективним є впровадження динамічних подій – тимчасових акцій, сезонних активностей чи тематичних оновлень, що підтримують інтерес завдяки своїй обмеженості у часі та стимулюють регулярне повернення до ігрового процесу. До таких подій можуть належати святкові ярмарки, сезонні фестивалі або спеціальні завдання, які доступні лише протягом певного періоду й пропонують унікальні нагороди чи рідкісні предмети. Крім того, регулярні щоденні або щотижневі виклики формують стабільний ритм взаємодії з грою, заохочуючи до

систематичної участі. Оновлення контенту у вигляді нових культур, тварин, техніки чи сюжетних ліній забезпечує різноманітність і стимулює гравців до дослідження та експериментування. У деяких випадках такі події поєднуються з елементами соціальної взаємодії – спільною участю у конкурсах або кооперативних активностях, що підсилює відчуття причетності до спільноти. Завдяки цим динамічним механікам ігровий світ залишається «живим», що знижує ризик одноманітності та формує стійку звичку повертатися до гри. [13]

Важливою складовою залучення гравця є соціальні зв'язки. Навіть у переважно одиночних фермерських симуляторах часто реалізовані механіки обміну ресурсами, рейтинги, кооперативні завдання чи змагання між користувачами, які формують відчуття причетності до спільноти. Додатково залученість підтримують сезонні події та тимчасові активності, що стимулюють взаємодію між гравцями через спільні фестивалі, конкурси чи кооперативні проєкти. Наприклад, у Farm Together користувачі можуть паралельно вести власні ферми, долучатися до кооперативних активностей та виконувати унікальні щоденні завдання. Поєднання соціальних механік із динамічним контентом посилює емоційне занурення, створює стабільні точки повернення та формує довготривалу прихильність до гри. [13, 14]

Окрему групу психологічних механізмів становлять системи поведінкової мотивації. Змінні винагороди – від рідкісних предметів до випадкових бонусів – підтримують інтерес завдяки ефекту непередбачуваності, тоді як щоденні, тижневі чи місячні завдання стимулюють регулярний вхід у гру та формують стійку ігрову звичку. Саме ці механіки забезпечують емоційне підкріплення і підтримують відчуття прогресу у довгостроковій перспективі. У сучасних фермерських симуляторах, зокрема Farm Together, Echoes of the Plum Grove чи Farming Simulator 25, такі системи працюють як основа регулярної взаємодії з грою: вони урізноманітнюють геймплей, забезпечують відчуття винагороди та сприяють тривалому утриманню користувачів без необхідності постійних соціальних або подійних стимулів. [13]

Суттєвий вплив на досвід гравця має затишна естетика та загальна атмосфера фермерських симуляторів. Тепла кольорова палітра, м’яке освітлення, спрощені форми й плавні анімації створюють відчуття затишку, яке сприяє розслабленню та зниженню рівня стресу. Природні звуки, образи домашнього побуту та доброзичливі персонажі посилюють емоційний комфорт і занурюють у спокійний ігровий простір. У цьому жанрі важливим є й повільний темп: відсутність суворих часових обмежень дає можливість приймати рішення у власному ритмі, зменшує когнітивне навантаження і підсилює відчуття контролю над процесом. [15]

Довгострокове повернення у «фермерських симуляторах» забезпечується насамперед циклічністю внутрішніх процесів – зміною сезонів, добовим ритмом, поетапністю вирощування культур та тривалими проєктами на кшталт будівництва чи розвитку господарства. Такі цикли створюють чіткий темп гри, близький до реальних фермерських процесів, і природно формують звичку повертатися, адже завершення одного етапу відкриває наступний. Довготривалі завдання запускають сталі «дофамінові петлі» очікування й винагороди та активують психологічні механізми – ефект власності, ефект незавершеності, інвестицію часу. Додаткове посилення retention забезпечують соціальні елементи: обмін ресурсами, кооперативні завдання чи взаємодія з фермами інших гравців, що створює відчуття спільноти й підвищує мотивацію повертатися у гру. [13]

Психологічні аспекти, від типології гравців до атмосфери затишку та механік поведінкової мотивації, формують фундамент, на якому будується привабливість і ефективність фермерських симуляторів. Саме завдяки цим інструментам ігри жанру створюють стійкий емоційний зв’язок із користувачем та підтримують інтерес у довгостроковій перспективі.

1.6 Значення візуальної складової у формуванні атмосфери гри

Візуальний стиль відіграє ключову роль у створенні атмосфери гри, адже саме він задає емоційний тон і визначає, як гравець сприйматиме ігровий світ з перших секунд взаємодії. Багато користувачів оцінюють якість гри ще до того, як

зануряться у геймплей, а згідно з дослідженням Entertainment Software Association, 72% гравців зазначають, що саме візуальні ефекти суттєво впливають на їхнє задоволення від процесу. Це стосується не лише естетичного враження – продумана візуальна складова допомагає глибше взаємодіяти з історією та персонажами. [16]

Додаткові дослідження свідчать, що приблизно 85% користувачів орієнтуються насамперед на візуальну інформацію. Саме тому якісна графіка здатна надовго утримувати увагу, сприяючи зануренню та підвищуючи показники повернення гравців. Перше враження, яке формує загальна стилістика, має прямий вплив на те, чи захоче користувач повернутися до гри. Колірна палітра, освітлення, текстури, форми об’єктів та анімація працюють комплексно, створюючи відчуття реалістичності, фантастичності, затишку або напруги – залежно від задуму розробника. Наприклад, холодні сині та сірі відтінки посилюють атмосферу відстороненості чи таємничості, тоді як теплі зелені й золотисті кольори формують більш природне, комфортне середовище (рис. 1.11). Візуальний стиль також виконує функцію навігації: він допомагає гравцеві інтуїтивно зрозуміти правила світу та помітити важливі елементи без зайвих пояснень. Унікальна стилістика робить гру впізнаваною і підсилює емоційну залученість, формуючи стійку мотивацію продовжувати взаємодію з ігровим світом. [16]

Рисунок 1.11 – Палітри теплих та холодних кольорів

Сучасні тенденції у візуальному оформленні ігор спрямовані на поєднання реалістичних елементів зі стилізованими художніми рішеннями, що дозволяє створювати впізнавану атмосферу та підтримувати індивідуальність проєкту. Технічний прогрес – зокрема використання рейтрейсингу та вдосконалених систем динамічного освітлення – забезпечує підвищену якість графіки без суттєвого зниження продуктивності. Популярності набувають мінімалістичні інтерфейси, орієнтовані на зрозумілу та інтуїтивну візуальну комунікацію, що полегшує взаємодію користувача з грою. Застосування процедурної генерації дає змогу створювати великі та різноманітні ігрові світи, у яких кожна сесія може пропонувати унікальний досвід. Водночас удосконалені системи анімації та динамічні візуальні ефекти роблять ігрове середовище більш живим, підсилюючи занурення та емоційну залученість гравця. [17]

Отже, візуальний стиль виступає ключовим чинником формування атмосфери та глибини ігрового досвіду, визначаючи емоційне сприйняття й залученість користувача. Сучасний розвиток індустрії демонструє прагнення поєднувати технологічні можливості з унікальними художніми рішеннями, що дозволяє іграм залишатися водночас технічно досконалими та виразними за стилем.

1.7 Ключові елементи ігрового середовища

Ключові компоненти ігрового середовища, зокрема ландшафт, об’єкти, рослинність та архітектура, формують цілісність і унікальність віртуального світу. Кожен із них робить внесок у його реалістичність та вимагає відповідної оптимізації, необхідної для забезпечення стабільної роботи гри.

Ландшафт у грі охоплює природні елементи – землю, пагорби, рівнинні ділянки, водойми, які формують простір пересування та взаємодії гравця. Рельєф, розташування водних об’єктів та загальна структура території впливають як на атмосферу, так і на особливості ігрового досвіду, створюючи відчуття цілісного,

реального або фантазійного світу. Під час розробки враховують природну логіку середовища, композицію та екологічний контекст. [18]

Елементи оточення – будівлі, техніка, предмети, персонажі наповнюють світ змістом і визначають його ігрову динаміку. Вони мають значення як для сюжету, так і для механік, тому їх моделі та текстури створюються з урахуванням оптимізації: застосовуються рівні деталізації (LOD) та технології стиснення, щоб зберегти продуктивність.

Рослинність формує природність середовища – додає глибини, підсилює атмосферність і часто виступає ресурсом або елементом взаємодії. Для її оптимізації використовують різні рівні опрацювання та динамічне завантаження залежно від відстані до гравця.

Архітектурні елементи визначають стиль і культурний характер локацій, допомагають орієнтуватися в просторі та підтримують логіку світу. Це можуть бути будівлі, мости, оглядові майданчики чи інші малі архітектурні форми.

Оптимізація візуальних компонентів є невід'ємною умовою формування комфортного ігрового процесу та стабільної продуктивності, особливо з огляду на різноманітність апаратних платформ.

Передусім такі заходи спрямовані на зниження навантаження на графічний процесор (GPU). Одним із базових методів є використання рівнів деталізації (LOD), коли для об'єктів, віддалених від камери, автоматично підставляються менш деталізовані моделі. Це дозволяє підтримувати високу частоту кадрів без істотної втрати якості зображення. [19]

Крім того, застосовується occlusion culling – технологія, що відсіює об'єкти, приховані іншими елементами сцени та невидимі для гравця, завдяки чому скорочується кількість елементів, які необхідно обробляти під час рендерингу. Це особливо важливо для великих відкритих локацій, характерних для фермерських симуляторів, де на екрані потенційно можуть бути тисячі об'єктів рослинності. Додатково використовуються методи стиснення текстур і оптимальний добір роздільної здатності, що зменшує навантаження на відеопам'ять та пришвидшує завантаження графічних ресурсів. У поєднанні ці підходи забезпечують суттєву

економію пам’яті, у кілька разів, без помітної втрати візуальної якості, що робить їх ключовими для підтримання стабільної продуктивності. [19, 20]

Для анімації нерідко застосовують вертексну анімацію, яка потребує значно менше ресурсів порівняно зі скелетною, але водночас забезпечує достатню плавність та виразність рухів без перевантаження системи. Такий підхід ґрунтується на технології Vertex Animation Texture, коли траєкторії руху вершин заздалегідь записуються у текстуру, а їх відтворення здійснюється безпосередньо шейдером на GPU. Це дозволяє уникнути обчислень, пов’язаних із трансформацією кісток, зменшити кількість викликів рендеру та мінімізувати використання оперативної пам’яті, що робить метод ефективним для великої кількості однотипних об’єктів – рослинності, води чи динамічних ефектів. Водночас скелетна анімація, заснована на ієрархії кісток і ваговій прив’язці вершин, забезпечує вищий рівень контролю над рухами персонажів, підтримує інверсну кінематику, Blend Trees та ретаргетинг, що робить її незамінною для складних керованих моделей. Попри високу продуктивність, вертексний підхід має низку обмежень: фіксовану тривалість анімацій, складність у редагуванні під час виконання та значні вимоги до розміру текстур у разі роботи зі складними моделями. Тому оптимальним для сучасних проєктів вважається гібридний підхід, за якого персонажі та NPC анімуються скелетними методами, тоді як рослинність, вода та вторинні ефекти реалізуються засобами вертексної анімації, що забезпечує баланс між контролем, якістю та продуктивністю. [21]

Окремо варто згадати про вибір стилю візуалізації: ізометрична проекція чи стилізована графіка, особливо у мобільних проєктах, допомагають зберегти привабливість ігрового світу за меншої рендерної складності. Ізометрична подача сцени, що традиційно використовується у багатьох мобільних фермерських симуляторах, поєднує високу читабельність із низькими вимогами до апаратних ресурсів. Через відсутність перспективних обчислень та мінімальне використання буфера глибини така проекція значно зменшує навантаження на графічний процесор, дозволяючи ефективно застосовувати sprite batching і скорочувати кількість викликів рендерингу. Це забезпечує суттєву економію GPU-часу

порівняно з повноцінною тривимірною графікою. Додатковою перевагою є зручність взаємодії на сенсорних екранах: великі спрайти та рівномірна побудова сцени спрощують навігацію й роблять ігровий простір добре оглядовим без потреби постійного масштабування. Завдяки цьому навіть пристрої початкового рівня здатні стабільно підтримувати високу частоту кадрів, особливо за використання атласів спрайтів та оптимізованих 2D-анімацій. Такий підхід водночас зберігає естетичну привабливість, характерну для жанру, та забезпечує технічну ефективність, необхідну для мобільних платформ. [19, 22, 23]

Сучасні рушії, такі як Unity та Unreal Engine, надають широкий набір інструментів для впровадження зазначених оптимізацій. Паралельно вони підтримують і передові технології, зокрема рейтрейсинг та DLSS (Deep Learning Super Sampling), які дають змогу підвищувати якість зображення без критичного впливу на продуктивність. Водночас ці платформи поєднують традиційні методи оптимізації – такі як occlusion culling, рівні деталізації (LOD) та стиснення текстур – із новітніми рішеннями на кшталт Nanite, Lumen та технологій масштабування зображення DLSS/FSR. Завдяки цьому навіть проєкти з високою деталізацією можуть стабільно працювати на пристроях середнього класу. Для жанру фермерських симуляторів це означає можливість відтворювати великі й візуально насичені простори, включно з реалістичними ефектами освітлення чи рейтрейсингу, зберігаючи продуктивність на рівні 60 кадрів за секунду і вище. [19, 24]

1.8 Інструменти створення графічних ресурсів для гри

Інструменти для створення графічних ресурсів у відеоіграх значною мірою залежать від типу графіки, 2D чи 3D, та складності проєкту, проте серед них можна виділити ключові програми, які широко використовуються у геймдеві.

Для тривимірного моделювання та анімації особливу популярність мають Blender, Autodesk Maya, Autodesk 3ds Max та ZBrush. Blender є безкоштовним і відкритим рішенням, що поєднує функції моделювання, текстурування, анімації та

рендерингу, часто обирається завдяки універсальності, активній спільноті та постійному оновленню. Autodesk Maya вважається індустріальним стандартом для складної анімації, ріггінгу та інтеграції з VFX-програмами і переважно використовується професійними студіями для анімації персонажів та спецефектів. 3ds Max популярний для архітектурної візуалізації та полігонального моделювання, підтримує різноманітні рендерери і дозволяє створювати фотореалістичні сцени. ZBrush спеціалізується на цифровій скульптурі та високодеталізованому моделюванні, що робить його незамінним для органічних форм, персонажів і складних поверхонь. Часто ці програми використовуються у комбінації: моделі створюють і деталізують у ZBrush, анімують у Maya або Blender, а потім інтегрують у ігрові рушії, як Unity чи Unreal Engine. [25]

Для 2D-графіки та текстурування застосовують інші програми. Clip Studio Paint залишається популярним інструментом цифрового малювання завдяки великому набору пензлів, векторній підтримці й гнучкому інтерфейсу, що робить його зручним для створення складної графіки. Adobe Photoshop, індустріальний стандарт растрової обробки, застосовується для текстурування, UI-елементів та концепт-арту завдяки потужним можливостям роботи зі шарами й фільтрами. Krita, безкоштовна альтернатива Photoshop, підходить художникам із будь-яким бюджетом і пропонує розширений набір кистей, шари та можливість базової анімації. Pencil2D слугує простим інструментом для покадрової анімації у растровому й векторному форматах, популярним серед початківців. Aseprite спеціалізується на піксель-арті та анімації, пропонуючи інтуїтивний інтерфейс і роботу з шарами. Сукупно ці інструменти покривають повний спектр задач – від простих малюнків до високодеталізованої графіки, а вибір конкретного рішення залежить від вимог проєкту, бюджету та навичок розробника. [26, 27]

Анімаційна складова також охоплює широкий спектр інструментів. Для 2D-анімації популярними є Adobe Animate, Toon Boom Harmony, Pencil2D та мобільний FlipaClip, які підтримують покадрову та векторну графіку, дозволяють створювати плавні переходи і інтерактивні елементи. Для складної 3D-анімації застосовують Blender і Autodesk Maya, а для анімації персонажів із захопленням

рухів через вебкамеру – Adobe Character Animator. Для композитингу і створення ефектів часто використовується Adobe After Effects, що дозволяє поєднувати 3D-анімацію та відео для формування складних сцен. [28, 29]

Пайплайн експорту та імпорту в Unity – це систематичний процес перенесення моделей, текстур, анімацій та інших графічних ресурсів між зовнішніми програмами і Unity. Він забезпечує коректну інтеграцію створених активів у проєкт, зберігаючи їх якість, масштаб, матеріали та структуру. Такий процес дозволяє команді розробників працювати ефективно та узгоджено, мінімізує ризик технічних помилок і спрощує оновлення або заміну ресурсів на будь-якому етапі роботи. Крім того, правильно організований пайплайн полегшує взаємодію між художниками, технічними дизайнерами та програмістами, забезпечуючи єдині стандарти під час підготовки та імпорту активів.

Під час підготовки графічних елементів у програмах на кшталт Blender, Maya або 3ds Max художники проводять оптимізацію моделей: створюють коректні UV-розгортки, призначають матеріали, налаштовують скелетну анімацію. Для експорту зазвичай використовують формат FBX, якщо модель має анімації чи складну структуру, або OBJ для статичних об’єктів. Текстури найчастіше зберігають у PNG чи TGA зі стандартними розмірами потужності двійки (1024×1024, 2048×2048), що гарантує сумісність та коректну роботу в Unity. [30]

Імпорт у Unity зазвичай відбувається просто: активи перетягують у папку Assets, після чого Unity автоматично створює шаблон ігрового об’єкту, імпортує матеріали та підтягує текстури. Для зручності модель та її текстури зберігають у сусідніх папках, щоб Unity правильно пов’язав їх між собою. Після імпорту художники налаштовують параметри текстур у Inspector – Max Size, Compression, Mip Maps – щоб збалансувати якість і продуктивність. [31]

Наступним кроком є налаштування імпортованих моделей та анімацій. У Model Importer задають параметри Scale Factor, Mesh Compression, Read/Write Enabled. Для анімацій вибирають тип (Generic або Humanoid), створюють або імпортують Avatar. Матеріали адаптують під Standard, URP або HDRP шейдери, призначають карти Albedo, Normal Map, Metallic/Smoothness. Якщо

використовується PBR-пайплайн, Roughness зазвичай інтегрують в Alpha-канал Metallic-текстури. [30]

Для експорту ресурсів із Unity застосовують два основні підходи. Asset Bundles дозволяють динамічно завантажувати активи під час роботи гри, що критично для великих проєктів. Unity Packages використовують для передачі готових активів між проєктами. Їх можна експортувати через меню Unity: відкривши розділ Assets і вибравши пункт Export Package. Важливо, що Unity зберігає посилання на оригінальні джерельні файли, тому будь-які зміни в Assets автоматично оновлюються в сценах. [32, 33]

У великих командних проєктах пайплайн часто автоматизують: застосовують Unity Automation, плагіни на кшталт Pixyz для оптимізації CAD-моделей або USD-формат для спільної роботи між художниками, аніматорами та технічними спеціалістами. Такі інструменти допомагають уникати ручних помилок, прискорюють обмін файлами та забезпечують узгодженість між різними етапами виробництва. Оновлені активи синхронізують через Reimport або Refresh, а інколи інтегрують системи контролю версій (Git, Plastic SCM), що дозволяє підтримувати порядок, відстежувати зміни та зберігати стабільність у великих довготривалих проєктах. [34]

1.9 Unity як основний рушій для реалізації візуальної складової

Unity є одним із найпоширеніших кросплатформних рушіїв, який завдяки широкому набору інструментів для роботи з графікою, освітленням, анімацією та взаємодіями забезпечує зручне середовище для створення візуально цілісних 2D і 3D ігрових проєктів. Рушій дозволяє ефективно поєднувати художні й технічні елементи, забезпечуючи водночас продуктивність і високу якість зображення. [35]

Основою роботи в Unity є сцени, що виконують роль контейнерів для моделей, джерел світла, камер, інтерфейсу та інших об'єктів. Кожна сцена може представляти окрему локацію або рівень, а їх перемикання здійснюється через SceneManager. Система ієрархічного управління сценами та використання Prefab-

об’єктів дозволяють ефективно масштабувати проєкт і зберігати структурованість навіть у складних ігрових середовищах. Рушій забезпечує повноцінну підтримку як 2D, так і 3D графіки: у 2D доступні Sprite Renderer, Tilemap, інструменти 2D Physics і система освітлення URP, тоді як 3D середовище включає роботу з мешами, PBR-матеріалами, скелетною анімацією та якісним освітленням у URP або HDRP. [33, 36]

Unity також пропонує розвинені засоби постобробки зображення. Система Post Processing у межах URP і HDRP надає доступ до ефектів, таких як Bloom, Depth of Field, Motion Blur, Color Grading та інших, а Volume Framework дозволяє застосовувати їх не лише глобально, а й локально в межах конкретних зон сцени. За потреби розробник може створювати власні кастомні ефекти за допомогою шейдерів. Система освітлення підтримує динамічні, змішані та запечені режими, що дозволяє оптимально поєднувати якість і продуктивність. У HDRP доступні й технології трасування променів, які надають можливість реалізувати фотореалістичне освітлення. [35]

Засоби створення візуальних ефектів включають Particle System, що дозволяє формувати динамічні ефекти на кшталт вогню, диму, води чи магічних частинок. Модулі системи та підтримка Sub-Emitters забезпечують широкі можливості налаштування, а Visual Effect Graph дає змогу створювати процедурні GPU-ефекти у вузловому інтерфейсі. Анімаційна система Unity базується на Animator Controller, підтримує стани, переходи, Blend Trees та параметри. Для персонажів використовується Mecanim, для кат-сцен – Timeline, а в 2D передбачено інструменти кісткової анімації та Sprite Skin. Усі ці можливості можуть доповнюватися програмним керуванням через API. [37, 38]

Базові взаємодії в Unity реалізуються через компонентну систему. Collider та Rigidbody забезпечують фізичні зіткнення та рух, Input System відповідає за збирання й обробку сигналів управління, а UI Toolkit надає сучасний підхід до побудови інтерфейсів на основі UXML та стилів USS. Крім того, Event System керує інтерактивністю UI та об’єктів через події взаємодії, включно з обробкою натискань, наведень і променевих перевірок. [33, 35]

Завдяки універсальності, масштабованості та підтримці широкого спектра платформ Unity залишається одним із найзручніших інструментів для створення ігор будь-якого формату – від простих 2D і мобільних проєктів до VR-досвіду та складних 3D-систем із високою якістю візуалізації.

1.10 UI/UX у «фермерських симуляторах» та принципи його створення в Unity

UI/UX у фермерських симуляторах традиційно орієнтується на простоту, інтуїтивність і створення затишної атмосфери. Оскільки подібні ігри передбачають тривалі ігрові сесії, інтерфейс має забезпечувати швидкий доступ до інформації про ресурси, інструменти, техніку та загальний прогрес ферми. Особливу роль у цьому відіграє HUD – постійна інформаційна панель, що накладається поверх ігрової сцени та надає гравцеві ключові відомості без необхідності переривати процес гри. У фермерських симуляторах HUD зазвичай відображає кількість ресурсів, стан інструментів, сезонність, таймери та інші необхідні дані. Його дизайн, як правило, мінімалістичний, з великими іконками та добре читабельними шрифтами, що дозволяє комфортно сприймати інформацію на різних пристроях. [39]

Побудова HUD (рис. 1.12) у цьому жанрі ґрунтується на принципах читабельності та контекстності. Текстові елементи повинні залишатися чіткими та легко помітними, що досягається завдяки використанню достатнього розміру шрифтів і контрастних кольорів. Візуальні елементи, зокрема іконки, мають бути достатньо деталізованими, щоб гравець без труднощів розпізнавав їх у динамічному середовищі. Водночас інтерфейс повинен реагувати на дії користувача: відображати актуальні інструменти в залежності від контексту, підсвічувати активні елементи чи доповнювати дію короткими підказками. Такий підхід дає змогу підтримувати баланс між інформативністю та візуальним комфортом, що є критично важливим для ігор із повільним, розслабленим темпом. [39]

Рисунок 1.12 – Приклад HUD у грі Meow Meow Star Acres

Значну роль у сучасній розробці інтерфейсів відіграє використання Figma як основного середовища для проєктування та первинного прототипування HUD. Завдяки векторній природі, підтримці автоматичних макетів і компонентному підходу інструмент дозволяє швидко створювати масштабовані й адаптивні макети. Інтеграція між Figma та Unity істотно прискорила робочий процес: інтерфейс, спроєктований у дизайнерському середовищі, може бути перенесений до Unity майже без змін у структурі та пропорціях. Це забезпечує точне відтворення елементів інтерфейсу, коректну адаптацію до різних роздільностей екрана та зменшує потребу у тривалій ручній верстці на стороні рушія.

Використання компонентів у Figma спрощує роботу з повторюваними елементами: зміни в одному компоненті автоматично застосовуються до всіх його копій. У поєднанні з Prefab у Unity це дозволяє швидко оновлювати UI. Крім того, Figma підтримує зручну спільну роботу – дизайнери й розробники можуть працювати в одному середовищі та відстежувати зміни в реальному часі.

Попри численні переваги такого підходу, варто враховувати, що інтеграція між Figma та Unity має певні технічні обмеження. Щоб забезпечити стабільність роботи та уникнути помилок при імпорті інтерфейсу, важливо розуміти типові труднощі та способи їх подолання. Основні проблеми, рекомендовані рішення та

інструменти, які допомагають оптимізувати цей процес, узагальнено у таблиці нижче.

Таблиця 2.1 – Обмеження та їх рішення при імпорті інтерфейсу з Figma у Unity

Обмеження	Опис проблеми	Рішення	Інструмент
Анімація	Figma Smart Animate не конвертується в Unity Animator	1. Додати Unity Animator Controller на імпортовані елементи 2. Tweening через DOTween/Cinemachine 3. Timeline для cutscene HUD	Unity Animator, DOTween
Drag & Drop	Figma прототипи не передають логіку	1. IDragHandler/IBeginDragHandler скрипти 2. EventSystem + Physics Raycaster 3. Canvas Group для drag поверхні	Unity EventSystem
Динамічний текст	Text layers фіксовані	1. TextMeshPro Dynamic текст замість Image 2. TMP Atlas для кастомних шрифтів 3. Rich Text markup	TextMeshPro
9-Slice scaling	Figma не має Unity 9-slice	1. Експорт з PNG до Unity Sliced Sprite 2. Figma Plugin з 9-slice export 3. UI Toolkit BorderImage	Unity Sprite Editor

Продовження табл. 2.1

Performance	Багато Image елементів	1. Canvas Groups + Batching 2. Atlas текстури (Sprite Packer) 3. UI Toolkit замість uGUI	Unity Profiler
Shaders	Градієнти з Figma відтворюються в Unity некоректно	1. Custom Shader Graph для HUD 2. URP Unlit шейдери 3. Material swapping	Shader Graph
Fonts	Шрифти не відтворюються в Unity один в один	1. Google Fonts SDF import 2. Custom TMP fonts 3. Dynamic Font fallback	TextMeshPro
Responsive	Різні aspect ratios	1. Canvas Scaler (Scale With Screen Size) 2. Safe Area для notch 3. Figma Auto Layout конвертація	Canvas Scaler

Висновки до розділу

У першому розділі здійснено теоретичний аналіз особливостей розробки комп’ютерних ігор у жанрі «фермерський симулятор». Встановлено, що даний жанр поєднує елементи економічної стратегії та медитативного геймплею, а його популярність базується на створенні психологічно комфортного середовища для гравця.

Визначено, що ключову роль у формуванні атмосфери гри відіграє візуальна складова. Аналіз аналогів (Farming Simulator, Stardew Valley, Coral Island) показав,

що незалежно від обраної стилістики (реалізм чи стилізація), графіка повинна бути «затишною», з використанням теплої колірної гами та м’яких форм.

Обґрунтовано вибір інструментарію для реалізації проєкту. Рушій Unity визначено як оптимальну платформу завдяки його універсальності, підтримці сучасних технологій рендерингу (URP) та широким можливостям для оптимізації графіки . Також досліджено принципи побудови UI/UX, де пріоритетом є мінімалізм та інтуїтивність інтерфейсу.

Таким чином, сформовано теоретичну та методологічну базу для практичної розробки дизайну та програмної реалізації власного ігрового проєкту в наступному розділі.

РОЗДІЛ 2
РОЗРОБКА КОНЦЕПЦІЇ, ВІЗУАЛЬНОГО СТИЛЮ ГРИ У ЖАНРІ «ФЕРМЕРСЬКИЙ СИМУЛЯТОР»

2.1 Концепція проєкту та формування візуального стилю

Концепція мого проєкту ґрунтується на створенні гри у жанрі фермерського симулятора, головними персонажами якої виступатимуть коти. Вибір саме цього стилю та сетингу є невипадковим: фермерські симулятори традиційно асоціюються з розміреним темпом гри, передбачуваністю і можливістю повного контролю над процесами. Це робить жанр особливо привабливим у контексті психологічної підтримки гравця.

У сучасних умовах, коли життя людей в Україні супроводжується війною та регулярними обстрілами, питання емоційної стабільності та пошуку безпечного простору набуває особливої актуальності. Тому в процесі проєктування я ставлю перед собою мету створити цифрове середовище, яке буде виконувати функцію умовного «тихого місця» для користувача – простору, де все передбачувано, зрозуміло та не викликає тривоги.

Вибір котів як основних персонажів базується на їхній сильній асоціації з турботою, затишком та емоційною підтримкою. Візуально приємні, милі та доброзичливі образи виступають своєрідним «емоційним буфером», який пом’якшує переживання та допомагає знижувати рівень стресу. Механіки гри також будуть підпорядковані цій концепції: розмірений темп, відсутність різких негативних подій та акцент на стабільності й турботі про віртуальних котиків.

Таким чином, основна ідея проєкту полягає не лише у створенні милої фермерської гри, а й у формуванні підтримувального і комфортного візуального середовища, здатного забезпечити користувачеві відчуття психологічної безпеки та дозволити хоча б на певний час відволіктися від зовнішніх стресових факторів.

Враховуючи обрану концепцію, поєднання фермерського симулятора та персонажів котів, потенційна цільова аудиторія проєкту є досить широкою.

Комбінація популярного жанру, спокійного і передбачуваного ігрового досвіду та привабливої візуальної естетики дозволяє охопити різні вікові й інтересові групи. Нижче узагальнено ключові аудиторні сегменти, на які орієнтується проєкт.

1. Діти та сімейні користувачі

М’який, неагресивний контент та образи котів роблять гру придатною для дітей 6-12 років, а також для родин, які шукають безпечні, зрозумілі та емоційно нейтральні ігри. Такий формат дозволяє батькам контролювати ігрове середовище та бути впевненими у його безпечності.

2. Підлітки та молодь (13-25 років)

Ця група активно цікавиться затишною естетикою, системами колекціонування та крафтингу, а також спільнотами, пов’язаними з тваринами. Образи котів органічно вписуються в сучасну інтернет культуру, що простежується у популярності мемів, фан-артів та тематичних соціальних спільнот.

3. Жінки віком 25-40 років

Для цієї аудиторії особливо важливими є релаксуючий темп гри, доброзичлива атмосфера та можливість відволіктися від повсякденного стресу. Комбінація передбачуваного геймплею та приємного візуального стилю створює комфортний користувацький досвід.

4. Казуальні гравці всіх вікових груп

Це користувачі, які віддають перевагу простим, ненапруженим ігровим механікам і прагнуть отримувати задоволення без додаткових когнітивних навантажень. Для них важливими є стабільність, милі персонажі та можливість грати у зручному темпі.

5. Шанувальники жанру «фермерський симулятор»

Ця категорія гравців цінує класичні механіки ведення ферми, але водночас прагне нових тематичних рішень. Наявність котиків як активних персонажів надає жанру свіжого та більш емоційно забарвленого вигляду.

6. Любителі тварин і представники онлайн-спільнот.

Гра орієнтована також на користувачів, які долучені до ком’юніті, пов’язаних із котами: від тематичних фан-груп до культури мемів про тварин. Такий контекст

сприяє швидкому поширенню проєкту через соціальні мережі та взаємодію між користувачами.

Окремо варто зазначити психологічні фактори, які впливають на залучення аудиторії. До них належать: формування емоційної прихильності до персонажів котів, створення затишної атмосфери з легкою візуальною стилістикою, можливість соціальної взаємодії між гравцями та загальний релаксуючий темп гри. У комплексі ці елементи забезпечують високий рівень емоційного комфорту та сприяють довготривалому утриманню користувача.

Таким чином, цільова аудиторія проєкту охоплює казуальних та сімейних гравців, шанувальників тварин, поціновувачів затишних ігор і тих, хто шукає стабільний, емоційно комфортний і тривалий ігровий досвід.

Відповідно до концепції проєкту, ігровий процес фермерського симулятора з котами як головними персонажами поєднує традиційні фермерські механіки з елементами колекціонування та м’якого, затишного ігрового процесу. Основний акцент робиться на емоційній привабливості персонажів, кастомізації та відчутті поступового, ненапруженого прогресу.

Основний цикл гри базується на виконанні повсякденних фермерських завдань, адаптованих під котячий світ. Після кожної успішної дії персонажі реагують емоційно – наприклад, радісно підстрибують, що підсилює ефект залучення та формує позитивний досвід взаємодії. До класичних видів діяльності додається риболовля, яка слугує окремою системою отримання ресурсів. Надалі ці ресурси можуть бути використані для створення предметів, обміну або розширення ферми. Таким чином формується послідовність «отримання ресурсів, їх подальша переробка чи реалізація та послідовне розширення господарства», яка і визначає основний цикл гри.

Тематика котів інтегрована в усі аспекти ігрового процесу. Користувач може змінювати зовнішність персонажів за допомогою кастомізації – нашийників, бантиків, декоративних елементів чи варіантів забарвлення шерсті. На фермі розміщуються спеціальні котячі споруди: рибні ставки, будиночки, іграшки та

інший декор, що водночас впливає на ігрову економіку й формує атмосферу затишку.

Традиційні механіки фермерського симулятора інтерпретуються через котячий світ: разом із звичайним ринком функціонує «рибний базар», а замість стандартних свят можуть проводитися ярмарки котячих талантів. Такий підхід дозволяє зберегти структуру жанру, водночас роблячи її унікальною та впізнаваною.

Соціальна взаємодія реалізується через систему обміну ресурсами та декоративними об’єктами між гравцями. Регулярні конкурси, такі як визначення найкращого врожаю чи найкреативнішої ферми, створюють механізми мотивації та сприяють довгостроковому залученню користувачів. Додатково передбачена рейтингова система, яка підтримує елемент змагальності без порушення загального спокійного темпу гри.

Система монетизації базується переважно на косметичних елементах: преміальні варіанти одягу, декору чи особливі види кастомізації. Для підтримки регулярної активності впроваджуються щоденні квести та сезонні події, пов’язані з тематичними святами, наприклад, «Котячий Гелловін» або «Літній котячий пікнік». Такий підхід посилює утримання і водночас не створює тиску на гравця.

Візуальна частина реалізується у форматі ізометричної 2.5D-графіки з використанням системи плиток для формування полів і системи Spine 2D для анімації котів. GPU Instancing дає змогу ефективно відображати велику кількість повторюваних об’єктів, що є важливим для фермерських ігор. Завдяки системам колекціонування та м’якого прогресу очікувані показники утримання користувачів можуть становити приблизно день 1 – близько 50%, день 30 – близько 15%.

Вибір затишної стилістики для проєкту не є виключно естетичним рішенням, а має під собою чітке концептуальне та психологічне обґрунтування. Ігри cozy-напряму давно стали предметом академічних досліджень, зокрема у сфері інклюзивного та феміністського дизайну, де вони розглядаються як форма «безпечного цифрового середовища». У сучасних соціально-політичних умовах, коли рівень стресу в суспільстві зростає, популярність затишних ігор демонструє

стабільну кореляцію з потребою користувачів у прогнозованому, підтримувальному та емоційно м’якому просторі. Саме тому застосування затишної естетики для фермерського симулятора з котиками є логічним кроком, який підсилює основну ідею створення комфортного для психіки ігрового середовища.[40]

Візуальні характеристики цього стилю формують особливу атмосферу, що ґрунтується на пастельній палітрі кольорів, м’якому освітленні та відсутності агресивних контрастів. Округлі форми об’єктів і персонажів, плавні анімації та легкі звукові ефекти (наприклад, муркотіння або м’які природні шуми) створюють чуттєво привабливий простір. Саме така композиція візуальних і аудіальних елементів сприяє розслабленню та підтримує емоційний комфорт користувача.

Особливо важливо, що затишна естетика органічно поєднується з образом котів як центральних персонажів. Коти традиційно асоціюються з домашнім теплом і безпекою, а м’яка стилістика додатково підсилює їхній чарівний образ. Більш реалістичний або контрастний стиль міг би нівелювати цю емоційну привабливість, тоді як затишна естетика посилює ефект «психологічної віддушини» – ключової цінності гри.

З технічного погляду обрана стилістика також є практично доцільною. Затишний 2D/2.5D-візуальний підхід добре оптимізується під мобільні пристрої та забезпечує стабільну роботу навіть на слабких системах. Використання універсального рендер-пайплайну Unity та ефективна робота зі спрайтами прискорюють арт-пайплайн і дозволяють зосередитися на створенні великої кількості варіативних декоративних елементів, що особливо важливо для ігор, орієнтованих на кастомізацію.

Таким чином, затишна естетика не лише відповідає загальній концепції створення безпечного та емоційно підтримувального ігрового простору, а й забезпечує стабільні показники залученості, сприяє формуванню психологічної прив’язаності до персонажів і відкриває широкі можливості для технічної реалізації та художнього розвитку проєкту.

Для визначення візуальної концепції гри були створені чотири мудборди, що охоплюють ключові напрямки: середовище, архітектура, персонажі та інтерфейс. Мудборди використовуються як інструмент візуального аналізу та дозволяють узагальнити стильові рішення, характерні для затишної естетики.

На мудборді середовища (рис. 2.1) узагальнено візуальні рішення, що визначають атмосферу ігрового простору: природні ландшафти, рослинність, водойми та декоративні елементи. Переважають округлі форми, пастельні кольори та м’які текстури, що відповідають принципам затишної естетики. Аналіз матеріалів дозволив сформувати напрямок для подальшого стилю середовища: відсутність агресивних контрастів, приглушена палітра та наявність дрібних «заспокійливих» деталей, таких як квіти та декоративні об’єкти.

Рисунок 2.1 – Мудборд для свторення середовища

Мудборд архітектури (рис. 2.2) містить референси невеликих будівель із м’якою геометрією, округлими дахами та декоративними елементами. Представлені матеріали допомогли визначити ключові принципи архітектурного стилю гри: простота форм, теплота кольорів та «дружній» вигляд конструкцій. Такий підхід дозволяє створити безпечний і затишний простір, де гравець не відчуває візуального напруження.

Рисунок 2.2 – Мудборд для створення архітектури

На мудборді персонажів (рис. 2.3)зібрано візуальні підходи до стилізації котів, що виступають головними героями гри. Проаналізовано пропорції, характер шерсті, емоції та можливі варіанти кастомізації. Виділено особливості дизайну, які підсилюють відчуття милоти та спокою: великі очі, округлі силуети, мінімалістичні риси обличчя. Мудборд дозволив сформувати напрямок дизайну персонажів як м’яких, доброзичливих і емоційно підтримувальних.

Рисунок 2.3 – Мудборд для створення персонажів

Мудборд інтерфейсу (рис. 2.4) репрезентує приклади дружнього, мінімалістичного UI, характерного для затишних ігор. Серед ключових елементів – округлі кнопки, пастельні акцентні кольори, м’які анімації та проста візуальна ієрархія. На основі аналізу визначено, що інтерфейс гри повинен уникати різких форм і високого контрасту, підтримуючи загальну атмосферу спокою.

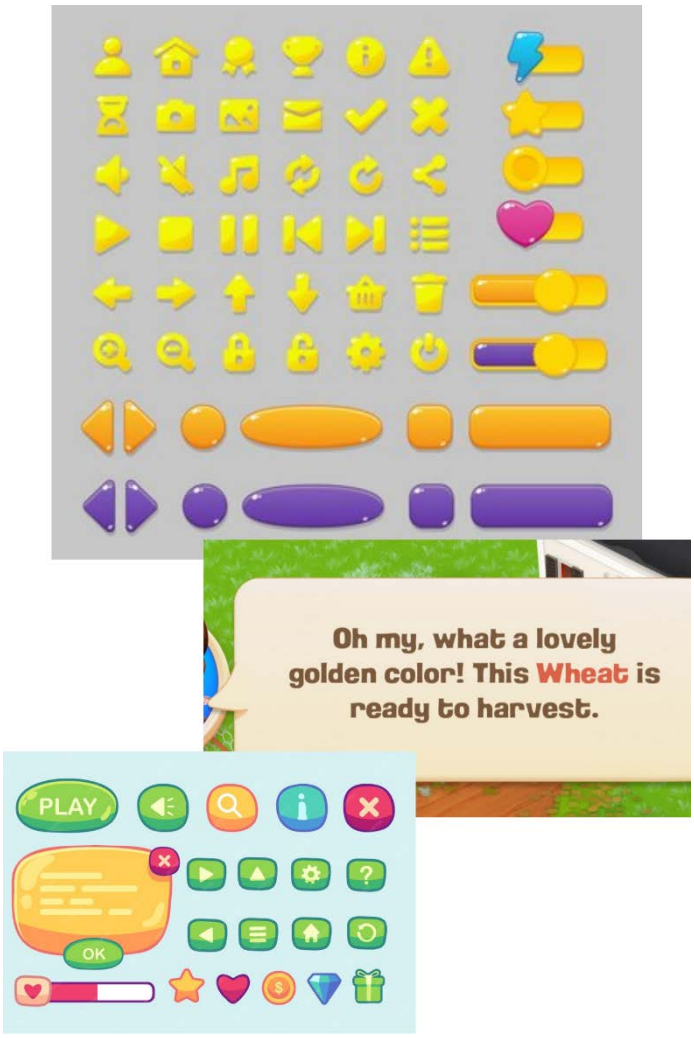

Рисунок 2.4 – Мудборд для створення інтерфейсу

Створення й аналіз мудбордів дозволили систематизувати стилістичні рішення та визначити єдину візуальну концепцію гри. На основі цих матеріалів сформовано загальні принципи графічного стилю, що орієнтований на затишність, емоційну безпеку та позитивне сприйняття гравцем. Мудборди стали основою для вибору кольорової палітри, форми персонажів і подальшого дизайну середовища та інтерфейсу.

2.2 Розробка графічних ресурсів: середовище, об’єкти та персонажі

У процесі розробки графічних ресурсів було використано підхід, що поєднує затишну естетику, котячу тематику та вимоги до оптимізації гри. Основним завданням стало сформувати візуальний стиль, який не буде створювати емоційне напруження й підтримає відчуття спокою та безпеки. Для цього використовувалися м’які форми, пастельна палітра, низька контрастність і спрощені силуети, які добре читаються в ізометричній перспективі.

Створення середовища (рис. 2.5) базувалося на принципах упізнаваності та м’якості форм. Для ґрунту, трави, дерев і декоративних об’єктів використовувалися округлі контури, приглушені кольори та мінімалістичні текстури. Основна увага приділялася читабельності елементів у масштабі ферми та їх гармонійній інтеграції в загальний візуальний стиль.

Рисунок 2.5 – Зображення середовища гри

Об’єкти ферми створювалися з урахуванням того, що гравець має сприймати їх як «комфортні» та ненав’язливі. Архітектура виконана в спрощеній манері з домінуванням округлих дахів, широких силуетів і декоративних деталей.

Рисунок 2.6 – Зображення дошки завдань у грі

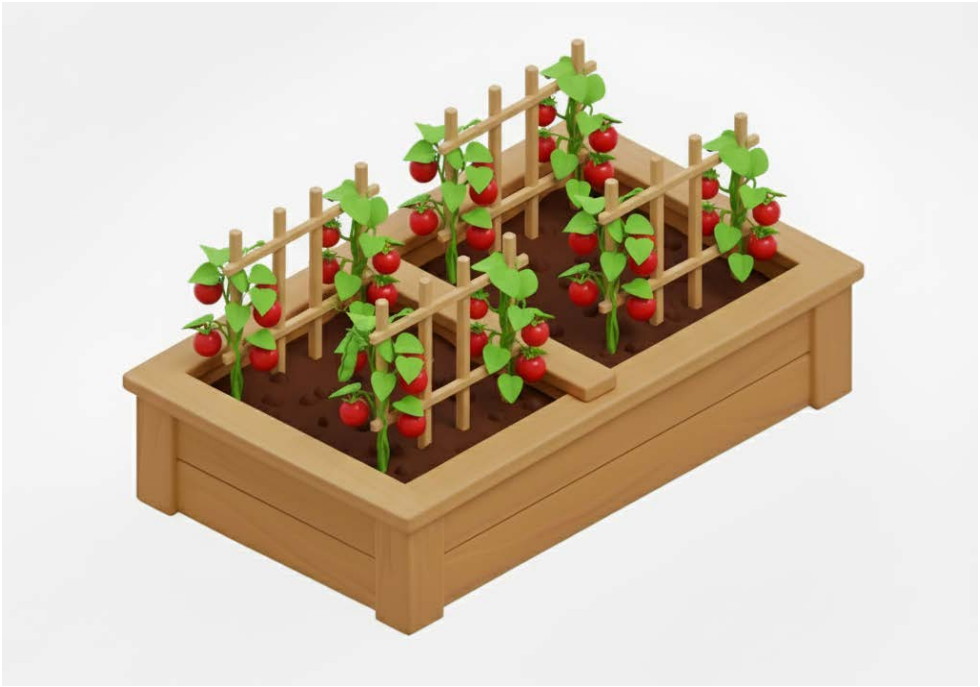

Рисунок 2.7 – Зображення грядок із помідорами у грі

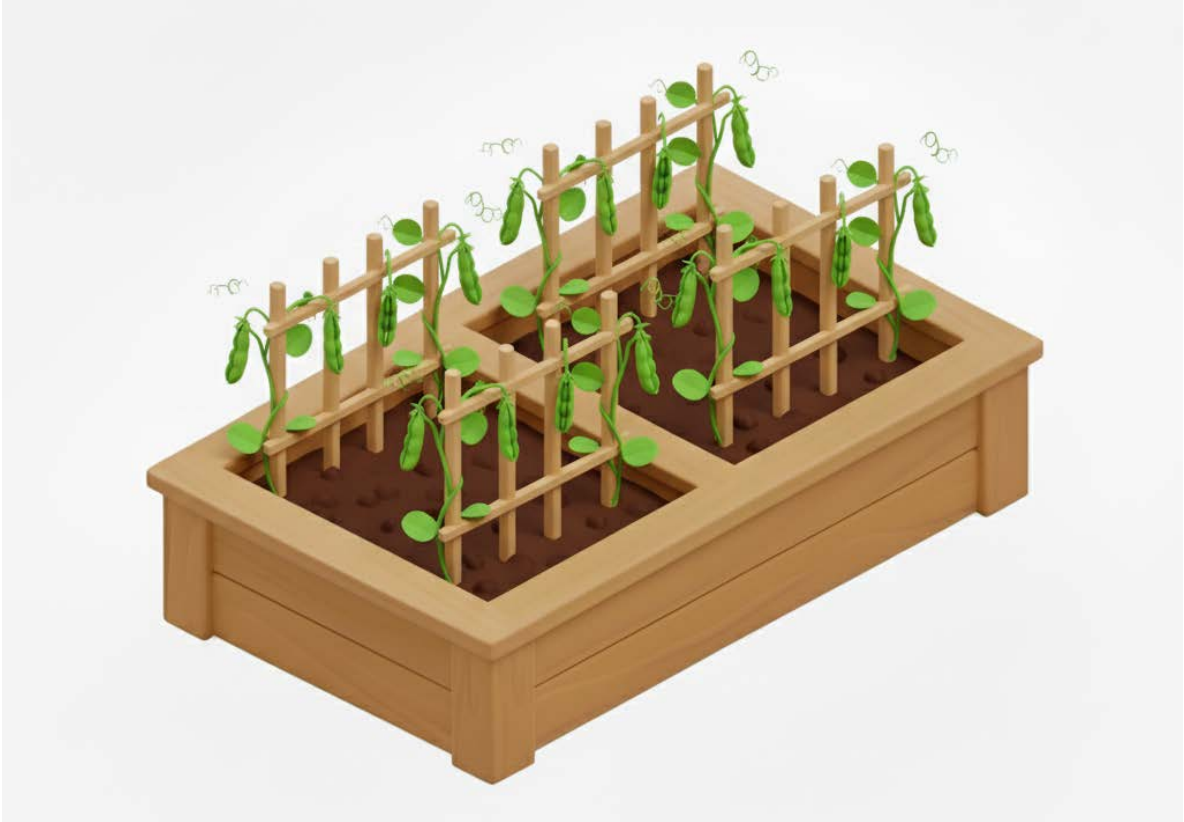

Рисунок 2.8 – Зображення грядки з горохом у грі

Рисунок 2.9 – Зображення ринку для продажу у грі

Рисунок 2.10 – Зображення кухні у грі

Рисунок 2.11 – Зображення поштового ящика у грі

Рисунок 2.12 – Зображення амбару у грі

Рисунок 2.13 – Зображення базових споруджень у грі

Рисунок 2.14 – Зображення базових споруджень та прикрашеного простору у грі

Рисунок 2.15 – Зображення базових споруджень у грі

Головним завданням у створенні персонажів було формування візуально привабливих і м’яких за стилістикою героїв, здатних викликати у гравця стале позитивне емоційне ставлення. Такий підхід безпосередньо відповідає загальній концепції милої гри, де персонаж виконує не лише функціональну, а й терапевтичну роль – створює атмосферу безпеки, турботи та спокою.

Візуальна мова побудована на принципах «милого дизайну»: збільшені очі, компактні пропорції, округлі силуети та мінімалістична міміка. Таке рішення забезпечує високу читабельність образу на різних пристроях і сприяє швидкому емоційному приєднанню користувача.

Дизайн персонажів узгоджено із середовищем гри: м’які кольори, спрощені форми та плавні контури підсилюють загальну естетику затишності й дозволяють героям органічно існувати в ігровому просторі. На початку розробки було створено нарис персонажів на папері (рис. 2.16). Потім цей малюнок було переведено у комп’ютерне 2D зображення (рис. 2.17), на основі якого будувалися 3D моделі головних персонажів для гри (рис. 2.18).

Для індивідуалізації ігрового досвіду реалізовано систему кастомізації: зміна кольору шерсті, аксесуари та декоративні елементи. Це не лише розширює експресивність персонажів, а й підвищує мотивацію гравця повертатися в гру.

Рисунок 2.16 – Нарис майбутніх персонажів гри

Рисунок 2.17 – Комп’ютерне графічне зображення персонажів гри

Рисунок 2.18 – 3D зображення персонажів гри

Висновки до розділу

У другому розділі розроблено загальну концепцію та візуальний стиль гри, що базується на естетиці «cozy games» із використанням персонажів-котів для психологічного комфорту гравця. Створено пайплайн розробки графічних ресурсів (3D-моделі, анімації, елементи оточення) та спроєктовано дизайн інтерфейсу користувача, підготувавши візуальну базу для подальшої технічної реалізації.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ ГРИ У ЖАНРІ «ФЕРМЕРСЬКИЙ СИМУЛЯТОР»

3.1 Програмна реалізація ключових механік гри

У проєкті реалізовано клас GameManager, який виконує функцію центрального керуючого модуля. Він відповідає за ініціалізацію гри, роботу зі збереженнями та загальну координацію базових систем сцени. Клас побудовано з використанням патерну Singleton, що дозволяє забезпечити наявність лише одного екземпляра протягом усього життєвого циклу застосунку. Це видно у фрагменті на рисунку 3.1. Така структура гарантує глобальний доступ до менеджера та збереження його стану між змінами сцен.

```
public static GameManager Instance { get; private set; }

private void Awake()
{
    // Singleton патерн
    if (Instance != null && Instance != this)
    {
        Destroy(gameObject);
        return;
    }
    Instance = this;
    DontDestroyOnLoad(gameObject);
}
```

Рисунок 3.1 – Зображення коду класу GameManager

Після запуску сцени викликається метод Start, де відбувається початкова підготовка. Логіка визначення стартового стану побудована на перевірці наявності збережених даних. Даний підхід, що зображено на рисунку 3.2, дозволяє

автоматично продовжити попередню сесію або створити нову, забезпечуючи плавність роботи і комфорт користувача.

```
private void Start()
{
    InitializeGame();
}

private void Update()
{
    // Автозбереження
    if (autoSaveEnabled)
    {
        autoSaveTimer += Time.deltaTime;
        if (autoSaveTimer >= autoSaveInterval)
        {
            SaveGame();
            autoSaveTimer = 0f;
        }
    }

    // Швидке збереження на F5
    if (Input.GetKeyDown(KeyCode.F5))
    {
        SaveGame();
        Debug.Log("Гру збережено (F5)");
    }

    // Завантаження на F9
    if (Input.GetKeyDown(KeyCode.F9))
    {
        LoadGame();
        Debug.Log("Гру завантажено (F9)");
    }
}

private void InitializeGame()
{
    // Спроба завантажити збережену гру
    if (SaveSystem.HasSaveData())
    {
        LoadGame();
    }
    else
    {
        StartNewGame();
    }
}
```

Рисунок 3.2 – Зображення коду методу Start

Одна з ключових функцій GameManager (рис. 3.3) – це управління збереженнями. Клас містить параметри, зображені на рисунку, які визначають активність і частоту автозбережень

```
[SerializeField] private bool autoSaveEnabled = true;
[SerializeField] private float autoSaveInterval = 60f; // секунди
```

Рисунок 3.3 – Зображення коду функції GameManager

Сам механізм виконання автозбережень працює в методі Update, де відбувається накопичення часу та періодичний виклик збереження (рис. 3.4)

```
autoSaveTimer += Time.deltaTime;
if (autoSaveTimer >= autoSaveInterval)
{
    SaveGame();
    autoSaveTimer = 0f;
}
```

Рисунок 3.4 – Зображення коду методу Update

Також реалізовано можливість ручного збереження й завантаження через гарячі клавіші, що особливо зручно на етапі тестування прототипу (рис. 3.5)

```
// Швидке збереження на F5
if (Input.GetKeyDown(KeyCode.F5))
{
    SaveGame();
    Debug.Log("Гру збережено (F5)");
}

// Завантаження на F9
if (Input.GetKeyDown(KeyCode.F9))
{
    LoadGame();
    Debug.Log("Гру завантажено (F9)");
}
```

Рисунок 3.5 – Зображення коду для ручного збереження і завантаження через гарячі клавіші

Для уникнення втрати даних передбачено автоматичне збереження під час виходу з гри (рис. 3.6)

```
private void OnApplicationQuit()
{
    if (autoSaveEnabled)
    {
        SaveGame();
    }
}
```

Рисунок 3.6 – Зображення коду методу OnApplicationQuit

Клас FarmManager виконує роль центрального модуля, що керує сіткою ферми, генерацією грядок та взаємодією з ними. Він відповідає за створення структурованого ігрового поля, збереження даних кожної клітинки та їх відновлення під час завантаження гри. Подібно до GameManager, цей компонент реалізує патерн Singleton, що забезпечує доступ до нього з будь-якої частини системи (рис. 3.7)

```
public static FarmManager Instance { get; private set; }

private void Awake()
{
    if (Instance != null && Instance != this)
    {
        Destroy(gameObject);
        return;
    }
    Instance = this;
}

private void Start()
{
    InitializeFarm();
}
```

Рисунок 3.7 – Зображення коду класу FarmManager

Основною функцією класу є автоматичне формування фермерської території. Параметри сітки задаються в інспекторі Unity (рис. 3.8)

```
[SerializeField] private int gridWidth = 10;
[SerializeField] private int gridHeight = 10;
[SerializeField] private float tileSize = 1f;
```

Рисунок 3.8 – Параметри налаштування сітки ферми

Під час запуску сцени викликається ініціалізація ферми (рис. 3.9)

```
private void Start()
{
    InitializeFarm();
}
```

Рисунок 3.9 – Ініціалізація ферми

Метод InitializeFarm генерує сітку грядок, створюючи об’єкти за допомогою префабу farmTilePrefab. Для кожної позиції системно обчислюється світова координата. Кожна створена клітинка отримує відповідну координату в сітці та зберігається у словнику (рис. 3.10). Таким чином формується повноцінне поле з gridWidth × gridHeight грядок, що дозволяє ігровим системам звертатися до будь-якої клітинки через сіткові координати.

```
private void InitializeFarm()
{
    if (farmParent == null)
    {
        GameObject parent = new GameObject("Farm");
        farmParent = parent.transform;
    }

    for (int x = 0; x < gridWidth; x++)
    {
        for (int y = 0; y < gridHeight; y++)
        {
            Vector2Int gridPos = new Vector2Int(x, y);
            Vector3 worldPos = GridToWorldPosition(gridPos);

            GameObject tileObj = Instantiate(farmTilePrefab, worldPos, Quaternion.identity, farmParent);
            tileObj.name = $"Tile_{x}_{y}";

            FarmTile tile = tileObj.GetComponent<FarmTile>();
            if (tile != null)
            {
                tile.SetGridPosition(gridPos);
                farmTiles[gridPos] = tile;
            }
        }
    }

    Debug.Log($"Ферму ініціалізовано: {gridWidth}x{gridHeight} грядок");
}
```

Рисунок 3.10 – Зображення коду методу InitializeFarm

Менеджер забезпечує універсальні методи конвертації координат, що дозволяють визначати клітинку як за сітковими, так і за світовими координатами (рис. 3.11). Ці методи є ключовими для взаємодії гравця з грядками (посадка, взаємодія з інструментами), оскільки ігрові об’єкти працюють у світових координатах, а логіка працює у сіткових.

```
/// Конвертація позиції сітки у світову позицію

public Vector3 GridToWorldPosition(Vector2Int gridPosition)
{
    return new Vector3(
        gridPosition.x * tileSize,
        gridPosition.y * tileSize,
        0f
    );
}

/// Конвертація світової позиції у позицію сітки

public Vector2Int WorldToGridPosition(Vector3 worldPosition)
{
    int x = Mathf.FloorToInt(worldPosition.x / tileSize);
    int y = Mathf.FloorToInt(worldPosition.y / tileSize);
    return new Vector2Int(x, y);
}
```

Рисунок 3.11 – Методи перетворення координат між сіткою та ігровим світом

Для організації взаємодії різних систем гри з елементами ферми використовується метод доступу до конкретної клітинки-грядки. У класі, що відповідає за управління всією картою ферми, реалізовано функцію GetTileAt, яка повертає об’єкт певної грядки за її координатами у сітці (рис. 3.12). Метод перевіряє, чи існує клітинка з переданими координатами. Якщо так, то повертається відповідний об’єкт, якщо ні – null, що дозволяє уникнути помилок та безпечно опрацьовувати недійсні позиції. Такий підхід забезпечує зручний і

централізований доступ до елементів поля для всіх систем гри (посадка, взаємодії, збір врожаю тощо).

```
public FarmTile GetTileAt(Vector2Int gridPosition)
{
    if (farmTiles.ContainsKey(gridPosition))
    {
        return farmTiles[gridPosition];
    }
    return null;
}
```

Рисунок 3.12 – Зображення коду функції GetTileAt

Клас відповідає за формування структурованих даних для збереження стану всього поля. Метод GetFarmData збирає TileData кожної грядки. Відповідно, метод LoadFarmData відновлює дані під час завантаження гри (рис. 3.13) Така реалізація дозволяє коректно відтворювати стадії росту рослин, стан ґрунту та інші параметри навіть після завершення сеансу.

```
public FarmData GetFarmData()
{
    FarmData data = new FarmData();
    data.tileDataList = new List<TileData>();

    foreach (var kvp in farmTiles)
    {
        data.tileDataList.Add(kvp.Value.GetTileData());
    }

    return data;
}

public void LoadFarmData(FarmData data)
{
    if (data == null || data.tileDataList == null) return;

    foreach (TileData tileData in data.tileDataList)
    {
        FarmTile tile = GetTileAt(tileData.gridPosition);
        if (tile != null)
        {
            tile.LoadTileData(tileData);
        }
    }

    Debug.Log($"Завантажено {data.tileDataList.Count} грядок");
}
```

Рисунок 3.13 – Зображення коду методів GetFarmData та LoadFarmData

Для створення нової гри передбачено метод, який виконує скидання грядок (рис. 3.14). Цей метод залишено максимально мінімалістичним, оскільки перегенерація грядок при повторній ініціалізації гарантує коректне оновлення поля.

```
public void ClearFarm()
{
    foreach (var tile in farmTiles.Values)
    {
        // Тайли скинуться автоматично при новій ініціалізації
    }
}
```

Рисунок 3.14 – Зображення коду методу ClearFarm

Скрипт FarmTile.cs реалізує логіку одного тайлу грядки та керує його станами, типом вирощуваної культури, прогресом росту та оновленням візуального відображення. Він є ключовою частиною фермерської системи, оскільки кожен тайл представляє окрему клітинку з власним життєвим циклом.

Для логічного розділення стадій життя ґрунту і рослини: від порожнього тайлу до готового до збору врожаю, визначається перелік можливих станів ґрунту через enum TileState (рис. 3.15).

```
public enum TileState
{
    Empty,        // Порожній
    Tilled,       // Зораний (лопатою)
    Prepared,     // Підготований (сапкою)
    Planted,      // Посаджено
    Watered,      // Політий
    ReadyToHarvest // Готовий до збору
}
```

Рисунок 3.15 – Зображення коду enum TileState

Для відображення розвитку рослини використовується прогрес росту (growthProgress) і стадії росту (growthStage) (рис. 3.16).

```
[Range(0f, 1f)]
public float growthProgress = 0f;
public int growthStage = 0; // 0-3 стадії росту
```

Рисунок 3.16 – Параметри прогресу та стадії росту культури

Метод Update (рис. 3.17) відповідає за автоматичне зростання рослини після поливу. У даному методі враховується добриво, яке прискорює ріст. Прогрес росту переводиться у стадії росту. Коли прогрес досягає 100%, рослина стає готовою до збору. Таким чином, реалізовано циклічну зміну стадій росту рослин з автоматичним оновленням візуального стану.

```
private void Update()
{
    // Ріст рослини
    if (currentState == TileState.Planted && isWatered)
    {
        float growthSpeed = isFertilized ? fertilizerBoost : 1f;
        growthTimer += Time.deltaTime * growthSpeed;
        growthProgress = growthTimer / baseGrowthTime;

        // Оновлення стадії росту
        int newStage = Mathf.FloorToInt(growthProgress * 4);
        if (newStage != growthStage && newStage < 4)
        {
            growthStage = newStage;
            UpdateVisuals();
        }

        // Готово до збору
        if (growthProgress >= 1f)
        {
            currentState = TileState.ReadyToHarvest;
            growthStage = 3;
            UpdateVisuals();
        }
    }
}
```

Рисунок 3.17 – Зображення коду методу Update

Для взаємодії гравця з грядками створено метод InteractWithTool (рис. 3.18), який обробляє всі доступні інструменти. Його головна функція – виступати єдиною точкою взаємодії з тайлом, забезпечуючи обробку всіх дій гравця через інструменти. Такий підхід робить код більш структурованим і дозволяє легко додавати нові інструменти у майбутньому. Метод реалізований через switch-case, де кожен інструмент викликає відповідний приватний метод. Це централізує логіку дій гравця, а приватні функції відповідають за конкретну реалізацію, що забезпечує чистоту коду та зручність його підтримки.

```
public bool InteractWithTool(ToolType tool, CropType seedType = CropType.None)
{
    switch (tool)
    {
        case ToolType.Shovel:
            return TillSoil();

        case ToolType.Hoe:
            return PrepareSoil();

        case ToolType.WateringCan:
            return WaterTile();

        case ToolType.Seeds:
            return PlantSeed(seedType);

        case ToolType.Fertilizer:
            return ApplyFertilizer();

        case ToolType.None:
            return HarvestCrop();

        default:
            return false;
    }
}
```

Рисунок 3.18 – Зображення коду методу InteractWithTool

Наприклад, метод PlantSeed (рис. 3.19) відповідає за посадку рослини. Він перевіряє стан тайлу, оновлює його стан та тип рослини, ініціалізує ріст, оновлює спрайт через UpdateVisuals і повертає результат дії (true або false), забезпечуючи зворотний зв’язок для гравця. Така реалізація гарантує логічну послідовність дій:

рослина не може бути посаджена на непідготовлений ґрунт, а гравець не зможе посадити кілька культур одночасно.

```
private bool PlantSeed(CropType seedType)
{
    if (currentState == TileState.Prepared && seedType != CropType.None)
    {
        currentState = TileState.Planted;
        cropType = seedType;
        growthProgress = 0f;
        growthStage = 0;
        growthTimer = 0f;
        UpdateVisuals();
        return true;
    }
    return false;
}
```

Рисунок 3.19 – Зображення коду методу PlantSeed

Метод UpdateVisuals (рис. 3.20) оновлює спрайт тайлу залежно від його стану (Empty, Tilled, Prepared, Planted, Watered, ReadyToHarvest). Для посаджених культур він викликає UpdateCropVisuals(), щоб відобразити правильну стадію росту конкретної рослини.

Це дозволяє гравцю одразу бачити результат своїх дій – оранку, підготовку ґрунту, полив чи збір врожаю – та забезпечує наочну і логічну взаємодію з грядкою, спрощуючи підтримку коду і додавання нових культур або станів.

```
private void UpdateVisuals()
{
    if (spriteRenderer == null) return;

    switch (currentState)
    {
        case TileState.Empty:
            spriteRenderer.sprite = emptySprite;
            break;

        case TileState.Tilled:
            spriteRenderer.sprite = tilledSprite;
            break;

        case TileState.Prepared:
            spriteRenderer.sprite = preparedSprite;
            break;

        case TileState.Planted:
        case TileState.Watered:
        case TileState.ReadyToHarvest:
            UpdateCropVisuals();
            break;
    }
}
```

Рисунок 3.20 – Зображення коду методу UpdateVisuals

Метод UpdateCropVisuals (рис. 3.21) відповідає за оновлення зовнішнього вигляду рослини відповідно до поточної стадії росту. Він отримує масив спрайтів конкретної культури та встановлює спрайт, який відповідає значенню змінної growthStage. Це забезпечує плавну візуальну зміну рослини від початкової стадії до повної зрілості. Така реалізація дозволяє гравцеві візуально відстежувати ріст рослини без додаткових індикаторів.

```
private void UpdateCropVisuals()
{
    Sprite[] stages = GetCropStages();
    if (stages != null && stages.Length > growthStage)
    {
        spriteRenderer.sprite = stages[growthStage];
    }
}
```

Рисунок 3.21 – Зображення коду методу UpdateCropVisuals

Метод GetCropStages (рис. 3.22) використовується для вибору набору спрайтів залежно від типу посіяної культури. Такий підхід дозволяє зручно працювати з кількома видами рослин, уникати дублювання коду та легко додавати нові культури, просто доповнюючи перелік спрайтів без змін у загальній логіці роботи системи.

```
private Sprite[] GetCropStages()
{
    switch (cropType)
    {
        case CropType.Wheat: return wheatStages;
        case CropType.Carrot: return carrotStages;
        case CropType.Tomato: return tomatoStages;
        case CropType.Potato: return potatoStages;
        default: return null;
    }
}
```

Рисунок 3.22 – Зображення коду методу GetCropStages

Метод HarvestCrop (рис. 3.23) відповідає за збір врожаю з грядки. Він перевіряє, чи тайл перебуває у стані ReadyToHarvest, тобто чи рослина повністю дозріла. У разі успішної перевірки метод додає зібрану культуру до інвентаря гравця через InventoryManager, після чого виконує скидання стану тайлу.

Після збору врожаю викликається метод ResetTile, який повертає грядку до початкового стану: очищає тип рослини, прогрес росту та додаткові ефекти (полив і добрива). Такий підхід забезпечує циклічність ігрового процесу, дозволяючи повторно використовувати один і той самий тайл без створення нових об’єктів.

```
private bool HarvestCrop()
{
    if (currentState == TileState.ReadyToHarvest)
    {
        // Додати врожай до інвентаря
        if (InventoryManager.Instance != null)
        {
            InventoryManager.Instance.AddCrop(cropType, 1);
        }

        // Скинути тайл
        ResetTile();
        return true;
    }
    return false;
}
```

Рисунок 3.23 – Зображення коду методів HarvestCrop та ResetTile

Для підтримки прогресу гри кожен тайл може зберігати та відновлювати свій стан. TileData містить інформацію про: стан тайлу, тип рослини, прогрес і стадію росту, полив та добрива, позицію на сітці (рис. 3.24). Це дозволяє забезпечити відновлення гри після перезапуску та інтегрувати тайли у масштабну систему ферми.

```
public TileData GetTileData()
{
    return new TileData
    {
        state = currentState,
        cropType = cropType,
        growthProgress = growthProgress,
        growthStage = growthStage,
        isWatered = isWatered,
        isFertilized = isFertilized,
        growthTimer = growthTimer,
        gridPosition = gridPosition
    };
}

public void LoadTileData(TileData data)
{
    currentState = data.state;
    cropType = data.cropType;
    growthProgress = data.growthProgress;
    growthStage = data.growthStage;
    isWatered = data.isWatered;
    isFertilized = data.isFertilized;
    growthTimer = data.growthTimer;
    gridPosition = data.gridPosition;
    UpdateVisuals();
}
```

Рисунок 3.23 – Методи отримання та відновлення даних тайлу гри

Скрипт InventoryManager відповідає за управління інвентарем гравця, зокрема вибором активного інструмента, обліком насіння, добрив та зібраного врожаю. Клас реалізовано як Singleton, що забезпечує глобальний доступ до інвентаря з інших частин гри (наприклад, із тайлів грядок). Такий підхід дозволяє централізувати роботу з ресурсами та уникнути дублювання логіки. Під час ініціалізації створюється базовий стан інвентаря, а також словник для збереження кількості зібраних культур кожного типу.

Важливою частиною роботи менеджера є обробка введення гравця. У методі HandleInput (рис. 3.24) реалізовано перемикання інструментів за допомогою клавіш клавіатури, а також зміну типу насіння при виборі інструмента Seeds. Це дозволяє

швидко керувати діями персонажа без складних меню. При зміні активного інструмента або типу насіння викликаються події (OnToolChanged, OnSeedTypeChanged), які можуть використовуватися для оновлення інтерфейсу користувача.

```
private void HandleInput()
{
    // Перемикання інструментів клавішами 1-6
    if (Input.GetKeyDown(KeyCode.Alpha1))
    {
        SetCurrentTool(ToolType.Shovel);
    }
    else if (Input.GetKeyDown(KeyCode.Alpha2))
    {
        SetCurrentTool(ToolType.Hoe);
    }
    else if (Input.GetKeyDown(KeyCode.Alpha3))
    {
        SetCurrentTool(ToolType.WateringCan);
    }
    else if (Input.GetKeyDown(KeyCode.Alpha4))
    {
        SetCurrentTool(ToolType.Seeds);
    }
    else if (Input.GetKeyDown(KeyCode.Alpha5))
    {
        SetCurrentTool(ToolType.Fertilizer);
    }
    else if (Input.GetKeyDown(KeyCode.Alpha6))
    {
        SetCurrentTool(ToolType.None); // Руки (для збору врожаю)
    }

    // Перемикання типу насіння (коли вибрано Seeds)
    if (currentTool == ToolType.Seeds)
    {
        if (Input.GetKeyDown(KeyCode.Q))
        {
            CycleSeedType();
        }
    }
}
```

Рисунок 3.24 – Зображення коду методу HandleInput

Ключовим методом взаємодії з ігровим світом є TryUseTool (рис. 3.25). Він виступає посередником між інвентарем та грядкою (FarmTile) і перевіряє можливість виконання дії з урахуванням наявних ресурсів. Наприклад, при посадці

насіння додатково перевіряється його кількість, а після успішної дії ресурс зменшується. Аналогічно реалізовано використання добрив. Такий підхід дозволяє відокремити логіку ресурсів від логіки тайлу, що підвищує масштабованість і зрозумілість коду.

```
public bool TryUseTool(FarmTile tile)
{
    if (tile == null) return false;

    bool success = false;

    switch (currentTool)
    {
        case ToolType.Shovel:
        case ToolType.Hoe:
        case ToolType.WateringCan:
            success = tile.InteractWithTool(currentTool);
            break;

        case ToolType.Seeds:
            if (HasSeeds(selectedSeedType))
            {
                success = tile.InteractWithTool(currentTool, selectedSeedType);
                if (success)
                {
                    UseSeeds(selectedSeedType, 1);
                }
            }
            else
            {
                Debug.Log($"Немає насіння: {GetCropName(selectedSeedType)}");
            }
            break;

        case ToolType.Fertilizer:
            if (fertilizer > 0)
            {
                success = tile.InteractWithTool(currentTool);
                if (success)
                {
                    fertilizer--;
                    OnInventoryChanged?.Invoke();
                }
            }
            else
            {
                Debug.Log("Немає добрив");
            }
            break;

        case ToolType.None:
            success = tile.InteractWithTool(ToolType.None);
            break;
    }

    return success;
}
```

Рисунок 3.25 – Зображення коду методу TryUseTool

Окремо реалізовано методи для роботи з ресурсами: перевірка наявності насіння, зменшення та додавання ресурсів, а також облік зібраного врожаю. Для

збереження прогресу гри передбачено методи серіалізації GetInventoryData (рис. 3.26) та LoadInventoryData (рис. 3.27), які дозволяють відновити стан інвентаря під час завантаження гри. Таким чином, InventoryManager забезпечує цілісну систему управління ресурсами та тісно інтегрується з іншими ігровими механіками.

```
public InventoryData GetInventoryData()
{
    return new InventoryData
    {
        currentTool = currentTool,
        selectedSeedType = selectedSeedType,
        wheatSeeds = wheatSeeds,
        carrotSeeds = carrotSeeds,
        tomatoSeeds = tomatoSeeds,
        potatoSeeds = potatoSeeds,
        fertilizer = fertilizer,
        wheatHarvested = GetCropCount(CropType.Wheat),
        carrotHarvested = GetCropCount(CropType.Carrot),
        tomatoHarvested = GetCropCount(CropType.Tomato),
        potatoHarvested = GetCropCount(CropType.Potato)
    };
}
```

Рисунок 3.26 – Зображення коду методу GetInventoryData

```
public void LoadInventoryData(InventoryData data)
{
    currentTool = data.currentTool;
    selectedSeedType = data.selectedSeedType;
    wheatSeeds = data.wheatSeeds;
    carrotSeeds = data.carrotSeeds;
    tomatoSeeds = data.tomatoSeeds;
    potatoSeeds = data.potatoSeeds;
    fertilizer = data.fertilizer;

    harvestedCrops[CropType.Wheat] = data.wheatHarvested;
    harvestedCrops[CropType.Carrot] = data.carrotHarvested;
    harvestedCrops[CropType.Tomato] = data.tomatoHarvested;
    harvestedCrops[CropType.Potato] = data.potatoHarvested;

    OnToolChanged?.Invoke(currentTool);
    OnInventoryChanged?.Invoke();
}
```

Рисунок 3.27 – Зображення коду методу LoadInventoryData

Скрипт PlayerController відповідає за керування персонажем гравця, зокрема за його переміщення по ігровій сцені та взаємодію з об’єктами ферми.

Переміщення реалізовано на основі фізики Unity з використанням компонента Rigidbody2D, що забезпечує плавний і стабільний рух. У методі Update (рис. 3.28) зчитується введення з клавіатури, а безпосередній рух застосовується у FixedUpdate, що відповідає рекомендаціям Unity для роботи з фізикою. Додатково реалізовано візуальний розворот спрайта залежно від напрямку руху персонажа, що підвищує відчуття контролю.

```
private void Update()
{
    HandleMovementInput();
    HandleInteractionInput();
    UpdateTargetTile();
}

private void FixedUpdate()
{
    Move();
}

private void HandleMovementInput()
{
    // WASD або стрілки
    moveInput.x = Input.GetAxisRaw("Horizontal");
    moveInput.y = Input.GetAxisRaw("Vertical");
    moveInput.Normalize();
}

private void Move()
{
    Vector2 movement = moveInput * moveSpeed;
    rb.velocity = movement;

    // Відображення спрайту в залежності від напрямку руху
    if (moveInput.x < 0)
    {
        spriteRenderer.flipX = true;
    }
    else if (moveInput.x > 0)
    {
        spriteRenderer.flipX = false;
    }
}
```

Рисунок 3.28 – Зображення коду методів Update та Move

Окрему увагу приділено системі взаємодії з грядками. У кожен момент часу контролер визначає найближчий тайл, з яким гравець може взаємодіяти, враховуючи напрямок руху та заданий радіус взаємодії. Якщо тайл знаходиться в межах досяжності, він зберігається як поточна ціль, а над ним відображається

індикатор взаємодії. При натисканні клавіші взаємодії (Space або ліва кнопка миші) викликається метод TryUseTool (рис. 3.29) менеджера інвентаря, який вже визначає можливість виконання дії. Така архітектура дозволяє чітко розділити відповідальність між керуванням персонажем, інвентарем і логікою грядок, що спрощує підтримку та розширення проєкту.

```
public bool TryUseTool(FarmTile tile)
{
    if (tile == null) return false;

    bool success = false;

    switch (currentTool)
    {
        case ToolType.Shovel:
        case ToolType.Hoe:
        case ToolType.WateringCan:
            success = tile.InteractWithTool(currentTool);
            break;

        case ToolType.Seeds:
            if (HasSeeds(selectedSeedType))
            {
                success = tile.InteractWithTool(currentTool, selectedSeedType);
                if (success)
                {
                    UseSeeds(selectedSeedType, 1);
                }
            }
            else
            {
                Debug.Log($"Немає насіння: {GetCropName(selectedSeedType)}");
            }
            break;

        case ToolType.Fertilizer:
            if (fertilizer > 0)
            {
                success = tile.InteractWithTool(currentTool);
                if (success)
                {
                    fertilizer--;
                    OnInventoryChanged?.Invoke();
                }
            }
            else
            {
                Debug.Log("Немає добрив");
            }
            break;

        case ToolType.None:
            success = tile.InteractWithTool(ToolType.None);
            break;
    }

    return success;
}
```

Рисунок 3.29 – Зображення коду методу TryUseTool

Скрипт SaveSystem реалізує систему збереження та завантаження стану гри. Він виконаний у вигляді статичного класу, що дозволяє звертатися до нього з будь-якої частини проєкту без необхідності створення окремого об’єкта на сцені. Збереження даних відбувається у файл формату JSON, який розміщується в директорії Application.persistentDataPath, що гарантує коректну роботу на різних платформах. Для зберігання використовується окремий клас GameSaveData (рис.

3.30), який об’єднує всі ключові дані гри: стан ферми, інвентар гравця та час створення збереження.

```
public class GameSaveData
{
    public string saveTime;
    public FarmData farmData;
    public InventoryData inventoryData;
}
```

Рисунок 3.30 – Зображення коду класу GameSaveData

Метод SaveGame (рис. 3.31) збирає поточний стан гри, отримуючи дані від менеджерів ферми та інвентаря, після чого серіалізує їх у JSON і записує у файл. Завантаження реалізовано в методі LoadGame (рис. 3.32), який перевіряє наявність файлу збереження, зчитує його вміст і відновлює стан гри шляхом передачі даних відповідним менеджерам. Додатково система містить допоміжні методи для перевірки наявності збереження, його видалення та отримання інформації про останній час збереження. Такий підхід забезпечує надійну та масштабовану систему збереження, яку легко розширити новими типами даних без суттєвих змін у загальній архітектурі проєкту.

```
public static void SaveGame()
{
    try
    {
        GameSaveData saveData = new GameSaveData();

        // Збереження ферми
        if (FarmManager.Instance != null)
        {
            saveData.farmData = FarmManager.Instance.GetFarmData();
        }

        // Збереження інвентаря
        if (InventoryManager.Instance != null)
        {
            saveData.inventoryData = InventoryManager.Instance.GetInventoryData();
        }

        // Збереження часу
        saveData.saveTime = DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss");

        // Серіалізація в JSON
        string json = JsonUtility.ToJson(saveData, true);

        // Запис у файл
        File.WriteAllText(SavePath, json);

        Debug.Log($"Гру збережено: {SavePath}");
    }
    catch (Exception e)
    {
        Debug.LogError($"Помилка при збереженні гри: {e.Message}");
    }
}
```

Рисунок 3.31 – Зображення коду методу SaveGame

```
public static void LoadGame()
{
    try
    {
        if (!File.Exists(SavePath))
        {
            Debug.LogWarning("Файл збереження не знайдено");
            return;
        }

        // Читання з файлу
        string json = File.ReadAllText(SavePath);

        // Десеріалізація з JSON
        GameSaveData saveData = JsonUtility.FromJson<GameSaveData>(json);

        // Завантаження ферми
        if (saveData.farmData != null && FarmManager.Instance != null)
        {
            FarmManager.Instance.LoadFarmData(saveData.farmData);
        }

        // Завантаження інвентаря
        if (saveData.inventoryData != null && InventoryManager.Instance != null)
        {
            InventoryManager.Instance.LoadInventoryData(saveData.inventoryData);
        }

        Debug.Log($"Гру завантажено (збережено: {saveData.saveTime})");
    }
    catch (Exception e)
    {
        Debug.LogError($"Помилка при завантаженні гри: {e.Message}");
    }
}
```

Рисунок 3.32 – Зображення коду методу LoadGame

Для відображення стану гри та взаємодії з гравцем реалізовано скрипт UIManager, який відповідає за оновлення користувацького інтерфейсу в реальному часі. Він відображає поточний вибраний інструмент, кількість ресурсів (насіння, добрива), зібраний врожай, а також текстові підказки з керування. Скрипт працює як посередник між логікою гри та UI, отримуючи актуальні дані з InventoryManager.

Однією з ключових особливостей є використання подій. У методі Start UIManager (рис. 3.33) підписується на події OnToolChanged, OnInventoryChanged та OnSeedTypeChanged, що дозволяє оновлювати інтерфейс лише тоді, коли дані

справді змінюються, а не кожен кадр. Такий підхід зменшує навантаження на систему та робить архітектуру більш модульною.

```
private void Start()
{
    // Підписка на події
    if (InventoryManager.Instance != null)
    {
        InventoryManager.Instance.OnToolChanged += UpdateToolDisplay;
        InventoryManager.Instance.OnInventoryChanged += UpdateInventoryDisplay;
        InventoryManager.Instance.OnSeedTypeChanged += UpdateSeedDisplay;
    }

    UpdateAllDisplays();
    UpdateHints();
}
```

Рисунок 3.33 – Зображення коду методу Start

Метод UpdateToolDisplay (рис. 3.34) відповідає за відображення активного інструменту. Якщо обрано насіння, додатково показується тип культури, що підкреслює контекст дії гравця.

```
private void UpdateToolDisplay(ToolType tool)
{
    if (toolNameText != null && InventoryManager.Instance != null)
    {
        string toolName = InventoryManager.Instance.GetToolName(tool);

        if (tool == ToolType.Seeds)
        {
            CropType seedType = InventoryManager.Instance.GetSelectedSeedType();
            string cropName = InventoryManager.Instance.GetCropName(seedType);
            toolNameText.text = $"{toolName}: {cropName}";
        }
        else
        {
            toolNameText.text = toolName;
        }
    }
}
```

Рисунок 3.34 – Зображення коду методу UpdateToolDisplay

Оновлення ресурсів і врожаю виконується в методі UpdateInventoryDisplay (рис. 3.35), де кількість насіння, добрив і зібраних культур відображається у текстових елементах UI. Також у скрипті реалізовано панель інвентаря, яку гравець

може приховувати або показувати натисканням клавіші Tab, що покращує зручність користування інтерфейсом.

```
private void UpdateInventoryDisplay()
{
    if (InventoryManager.Instance == null) return;

    // Насіння
    if (wheatSeedsText != null)
        wheatSeedsText.text = $"Пшениця: {InventoryManager.Instance.GetSeedCount(CropType.Wheat)}";

    if (carrotSeedsText != null)
        carrotSeedsText.text = $"Морква: {InventoryManager.Instance.GetSeedCount(CropType.Carrot)}";

    if (tomatoSeedsText != null)
        tomatoSeedsText.text = $"Помідор: {InventoryManager.Instance.GetSeedCount(CropType.Tomato)}";

    if (potatoSeedsText != null)
        potatoSeedsText.text = $"Картопля: {InventoryManager.Instance.GetSeedCount(CropType.Potato)}";

    if (fertilizerText != null)
        fertilizerText.text = $"Добрива: {InventoryManager.Instance.GetFertilizerCount()}";

    // Врожай
    if (wheatHarvestText != null)
        wheatHarvestText.text = $"Зібрано: {InventoryManager.Instance.GetCropCount(CropType.Wheat)}";

    if (carrotHarvestText != null)
        carrotHarvestText.text = $"Зібрано: {InventoryManager.Instance.GetCropCount(CropType.Carrot)}";

    if (tomatoHarvestText != null)
        tomatoHarvestText.text = $"Зібрано: {InventoryManager.Instance.GetCropCount(CropType.Tomato)}";

    if (potatoHarvestText != null)
        potatoHarvestText.text = $"Зібрано: {InventoryManager.Instance.GetCropCount(CropType.Potato)}";
}
```

Рисунок 3.35 – Зображення коду методу UpdateInventoryDisplay

Окремо реалізовано метод UpdateHints (рис. 3.36), який формує статичний блок підказок із керуванням. Це дозволяє гравцю швидко ознайомитися з основними механіками без необхідності додаткових меню.

```
private void UpdateHints()
{
    if (hintsText != null)
    {
        hintsText.text =
            "КЕРУВАННЯ:\n" +
            "WASD - Рух\n" +
            "Space/ЛКМ - Взаємодія\n" +
            "1 - Лопата (зорати)\n" +
            "2 - Сапка (підготувати)\n" +
            "3 - Лійка (полити)\n" +
            "4 - Насіння (посадити)\n" +
            "5 - Добрива (прискорити)\n" +
            "6 - Руки (зібрати)\n" +
            "Q - Змінити насіння\n" +
            "Tab - Інвентар\n" +
            "F5 - Зберегти\n" +
            "F9 - Завантажити";
    }
}
```

Рисунок 3.36 – Зображення коду методу UpdateHints

Висновки до розділу

У третьому розділі виконано програмну реалізацію основних механік гри мовою C# в середовищі Unity. Розроблено модульну архітектуру на базі патерну Singleton для ключових менеджерів (GameManager, FarmManager), що забезпечує централізоване керування станом гри.

Реалізовано повний агротехнічний цикл через систему тайлів (FarmTile) із логікою зміни станів (посів, ріст, збір). Впроваджено динамічний інтерфейс на основі системи подій для оптимізації продуктивності, а також систему збереження прогресу у формат JSON. У результаті створено стабільний робочий прототип із налагодженою взаємодією всіх ігрових систем.

ВИСНОВКИ

У кваліфікаційній роботі вирішено актуальне науково-прикладне завдання розробки дизайну та візуальної складової комп'ютерної гри у жанрі «фермерський симулятор» на базі рушія Unity. За результатами виконаних досліджень та практичної розробки встановлено, що жанр «фермерський симулятор» займає стійку нішу на ринку завдяки поєднанню економічної стратегії та медитативного геймплею. Аналіз аналогів показав, що успішність проєкту залежить від гармонійного поєднання ігрових механік та візуального стилю, який має створювати атмосферу психологічного комфорту, а також від врахування цільової аудиторії та психологічних патернів гравців, що впливають на їх утримання.

У ході роботи обґрунтовано вибір «затишної» естетики з використанням теплої колірної гами та м'яких форм. Розроблено оригінальну концепцію гри, де головними персонажами виступають коти, що підсилює емоційну привабливість проєкту. Створено пайплайн розробки графічних ресурсів: від мудбордів і скетчів до 3D-моделей та їх імпорту в Unity. Також розроблено та інтегровано інтерфейс користувача (HUD), попередньо спроєктований у Figma, з урахуванням принципів UX для забезпечення інтуїтивної взаємодії.

Підтверджено ефективність використання рушія Unity для створення ігор цього жанру. Використання інструментів URP, Shader Graph та Particle System дозволило досягти високої якості зображення, а застосування методів оптимізації, таких як GPU Instancing, LOD та атласи спрайтів, забезпечило стабільну продуктивність гри навіть при великій кількості об'єктів на сцені.

Розроблено гнучку та масштабовану архітектуру проєкту на мові C#, що включає реалізацію ключових систем: менеджменту гри через використання патерну Singleton для класів GameManager, FarmManager та InventoryManager; механіки фермерства із системою тайлів та логікою зміни станів (оранка, посів, полив, ріст, збір); системи збереження даних через серіалізацію у формат JSON; а також контролера персонажа з фізикою руху та контекстною системою використання інструментів.

Таким чином, мета роботи досягнута: створено функціональний прототип гри, який поєднує привабливу візуальну складову, обґрунтовану психологічними аспектами сприйняття, з надійною технічною реалізацією. Отримані результати можуть бути використані як основа для подальшої розробки комерційного ігрового продукту.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Маринін М. Історія розвитку комп'ютерних ігор: від пакмена до кіберспорту. Історія розвитку комп'ютерних ігор. URL: https://computerrgameshistory.blogspot.com/2023/04/blog-post.html#more (дата звернення: 08.06.2025).

2. Хайло А. Об'єкт дослідження: відеоігри. Вісник Книжкової палати. 2021. No. 11. P. 25–36. URL: https://doi.org/10.36273/2076-9555.2021.11(304).25-36 (дата звернення: 08.06.2025).

3. Степорук Н. Як соціолог світчнувся в геймдизайн. Розробник Den of Wolves – про пізній вхід у геймдев і складний пошук роботи в Європі. DOU. URL: https://gamedev.dou.ua/articles/interview-with-gamedesigner-10-chambers/ (дата звернення: 08.06.2025).

4. Жанри комп'ютерних ігор і системи їх монетизації. TCH.ua. URL: https://tsn.ua/cybersport/vidi-komp-yuternih-igor-zhanri-ta-monetizaciya-1710949.html (дата звернення: 08.06.2025).

5. Учасники проектів Вікімедіа. Farming Simulator (серія ігор) – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Farming_Simulator_(серія_ігор) (дата звернення: 10.06.2025).

6. ТОП ігор про фермерство / сільське господарство – Adler Agro Vision. Adler Agro Vision. URL: https://adler.com.ua/novyny/top-ihor-pro-fermerstvo-silske-hospodarstvo/ (дата звернення: 10.06.2025).

7. Світлична О., Кравченко Н., Малюк І. Дослідження графічної стилістики пригодницьких 2d відеоігор. Humanities science current issues. 2023. Т. 2, № 64. С. 91–97. URL: https://doi.org/10.24919/2308-4863/64-2-15 (дата звернення: 15.06.2025).

8. Nitrado gameserver. Nitrado Gameserver. URL: https://server.nitrado.net/en-US/news/a-brief-history-of-farming-simulator-a-look-back-from-2008-to-fs25/ (дата звернення: 09.08.2025).

9. Throwback: when was farming simulator at it's peak?. Reddit. URL: https://www.reddit.com/r/farmingsimulator/comments/19a8nss/throwback_when_was_farming_simulator_at_its_peak/?show=original (дата звернення: 15.06.2025).

10. Descriptions of characters and art style?. Reddit. URL: https://www.reddit.com/r/StardewValley/comments/sy84sw/descriptions_of_characters_and_art_style/ (дата звернення: 15.06.2025).

11. Why has the art direction in Harvest Moon games gone so far down hillm. Reddit. URL: https://www.reddit.com/r/StardewValley/comments/71fhb4/why_has_the_art_direction_in_harvest_moon_games/ (дата звернення: 20.06.2025).

12. Contributors to Wikimedia projects. Coral Island (video game) - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Coral_Island_(video_game) (дата звернення: 20.06.2025).

13. Механіки утримання в казуальних іграх: як залучати користувачів із різними психотипами • VOKI Games. Voki Games. URL: https://vokigames.com/ua/mehaniky-utrymannya-v-kazualnyh-igrah-yak-zaluchaty-korystuvachiv-iz-riznymy-psyhotypamy/ (дата звернення: 20.06.2025).

14. Іванченко А. Найкращі фермерські симулятори 2025 року - Acer Corner. Acer Corner. URL: https://blog.acer.com/ua/discussion/2918/naykraschi-fermerski-simulyatori-2025-roku (дата звернення: 20.06.2025).

15. Іванченко А. Ігри-симулятори життя з родзинкою для ПК-геймерів - Acer Corner. Acer Corner. URL: https://blog.acer.com/ua/discussion/2189/igri-simulyatori-zhittya-z-rodzinkoyu-dlya-pk-geymeriv (дата звернення: 25.06.2025).

16. Valeriu Crudu & MoldStud Research Team. Exploring the impact of diverse art styles on player engagement in various game genres. MoldStud. URL: https://moldstud.com/articles/p-exploring-the-impact-of-diverse-art-styles-on-player-engagement-in-various-game-genres (дата звернення: 25.06.2025).

17. Shiyan A. The transformation of stylization in video games. 80 Level. URL: https://80.lv/articles/stylization-in-video-games-a-deep-dive-analysis (дата звернення: 25.06.2025).

18. Будова ландшафту і стін в Unity - IT-школа. IT-школа. URL: https://go-mother.com/building-landscape-and-walls-in-unity/ (дата звернення: 25.06.2025).

19. Про оптимізацію простими словами. Куток. URL: https://kutok.io/icebang/pro_optymizaciu_prostymy_slovamy-34l8 (дата звернення: 25.06.25).

20. Оптимізація продуктивності в Unity: Прискорення ігор на ПК і мобільних пристроях. FoxmindEd. URL: https://foxminded.ua/optymizatsiia-produktyvnosti-v-unity/ (дата звернення: 25.06.2025).

21. Черевичний Р. Використання технології вертексної анімації VAT для створення комплексних візуальних ефектів. gamedev.dou. URL: https://gamedev.dou.ua/blogs/using-vertex-animation-visual-effects/ (дата звернення: 25.06.2025).

22. Isometric animation: A guide to stylized 3D motion. GarageFarm: Render Farm & Cloud Rendering Services. URL: https://garagefarm.net/blog/isometric-animation-breathing-life-into-stylized-worlds (дата звернення: 26.06.2025).

23. RocketBrush Studio. Isometric video games and how isometry benefits developers. RocketBrush. URL: https://rocketbrush.com/blog/isometric-games-how-isometry-benefits-game-developers (дата звернення: 26.06.2025).

24. Розробка ігор на Unreal та Unity: який двигун вибрати в 2025 році?. itproger. URL: https://itproger.com/ua/news/razrabotka-igr-na-unreal-i-unity-kakoy-dvizhok-vibrat-v-2025-godu (дата звернення: 27.06.2025).

25. ZBrush vs blender vs maya. Reddit. URL: https://www.reddit.com/r/3Dmodeling/comments/1h9chpx/zbrush_vs_blender_vs_maya/ (дата звернення: 27.06.2025).

26. Skoczylas N. 7 most used 2D graphics tools in indie games. IndieGameCloud. URL: https://indiegamecloud.com/7-most-used-2d-graphics-tools-in-indie-games/ (дата звернення: 27.06.2025).

27. Top 10 software for 2D art to explore in 2022. EJAW. URL: https://ejaw.net/top-10-software-for-2d-art (дата звернення: 27.06.2025).

28. 15 найкращих додатків для анімації у 2025. Komarov.Design. URL: https://www.komarov.design/15-naikrashchikh-dodatkiv-dlia-animatsiyi-u-2025/ (дата звернення: 27.06.2025).

29. Редакція Академії ITSTEP. Найкращі програми для створення анімації | Огляд програм для анімації. ITSTEP Академія Online освіта. URL: https://cloud.itstep.org/blog/top-8-programs-for-creating-animation (дата звернення: 28.06.2025).

30. Can't import 3D model with textures. Reddit. URL: https://www.reddit.com/r/Unity3D/comments/29p791/cant_import_3d_model_with_textures/ (дата звернення: 28.06.2025).

31. Unity - manual: importing textures. Unity documentation. URL: https://docs.unity3d.com/2019.3/Documentation/Manual/ImportingTextures.html (дата звернення: 28.06.2025).

32. Exporting packages - unity manual. Unity documentation. URL: https://docs.unity3d.com/ru/2018.4/Manual/HOWTO-exportpackage.html (дата звернення: 28.06.2025).

33. Unity - manual: asset workflow. Unity documentation. URL: https://docs.unity.cn/Manual/AssetWorkflow.html (дата звернення: 29.06.2025).

34. Create a pipeline for asset manager and unity asset transformer using unity automation, unity cloud, unity docs. Unity Documentation. URL: https://docs.unity.com/en-us/cloud/asset-manager/create-pipeline-asset-manager-pixyz (дата звернення: 29.06.2025).

35. Unity 2019.3: new features and updates for graphics | [site:name]. Unity. URL: https://unity.com/releases/2019-3/graphics (дата звернення: 29.06.2025).

36. 2D and 3D mode settings - Unity Manual. Unity Documentation. URL: https://docs.unity.cn/ru/2021.1/Manual/2DAnd3DModeSettings.html (дата звернення: 29.06.2025).

37. Unity - manual: particle animations. Unity Documentation. URL: https://docs.unity3d.com/Manual/particle-animation.html (дата звернення: 29.06.2025).

38. Unity - manual: using the built-in particle system. Unity Documentation. URL: https://docs.unity3d.com/2019.4/Documentation/Manual/PartSysUsage.html (дата звернення: 29.06.2025).

39. Farbod A. Mastering game HUD design : the best guide for you. Polydin. URL: https://polydin.com/game-hud-design/ (дата звернення: 15.08.2025).

40. Waszkiewicz A., Bakun M. Towards the aesthetics of cozy video games. Journal of gaming & virtual worlds. 2020. Т. 12, № 3. С. 225–240. URL: https://doi.org/10.1386/jgvw_00017_1 (дата звернення: 14.10.2025).

41. Маринич І. А., Тронь В. В. Методичні рекомендації до виконання кваліфікаційної роботи магістра для студентів спеціальності 151 “Автоматизація та комп’ютерно-інтегровані технології”. Кривий Ріг : Видавничий центр КНУ, 2022. 50с.

42. ДСТУ 3008:2015. Звіти у сфері науки і техніки. Структура і правила оформлення. Київ, ДП «УкрННЦ», 2015. 26с. (Інформація та документація).

43. ДСТУ 8302:2015. Бібліографічне посилання. Загальні вимоги та правила складання Київ, ДП «УкрННЦ», 2016. 16 с. (Інформація та документація).

44. ДСТУ 3582:2013. Бібліографічний опис. Скорочення слів і словосполучень в українській мові.

45. Загальні вимоги та правила. Київ, ДП «УкрННЦ», 2013. 23 с. (Інформація та документація)

46. ДСТУ 3651.0-97 Метрологія. Одиниці фізичних величин. Основні одиниці фізичних величин Міжнародної системи одиниць. Основні положення, назви та позначення Київ, Держстандарт України, 1998. 27 с. (Інформація та документація).