Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка відеогри на базі рушія Godot з використанням мови програмування C#

Виконав: студент IV курсу, групи СН-41
спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)

_____ Базюк Р.Л.
(підпис) (прізвище та ініціали)

Керівник _____ Матійчук Л.П.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Шимчук Г.В.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Боднарчук І.О.
(підпис) (прізвище та ініціали)

Рецензент _____ _____
(підпис) (прізвище та ініціали)

Тернопіль
2025

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«____» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)

за спеціальністю 122 Комп'ютерні науки
(шифр і назва спеціальності)

Студенту Базюку Роману Любомировичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка відеогри на базі рушія Godot з використанням мови програмування C#

Керівник роботи Матійчук Любомир Павлович, д. е. н., професор кафедри КН
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «*07*» *травня* 2025 року № *4/7-444*

2. Термін подання студентом завершеної роботи 23 червня 2025 р.

3. Вихідні дані до роботи базуються на основі досліджених літературних та інтернет-джерелах.

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1 Аналіз сфери розробки відеоігор. 1.1 Огляд сучасного стану GameDev-індустрії. 1.2 Аналіз рушіїв для розробки відеоігор. 1.3 Огляд існуючих 2D симуляторів фермерства. 1.4 Висновок до першого розділу. 2 Проектування відеогри у жанрі симулятора фермерства. 2.1 Дослідження та вибір жанру симулятора фермерства. 2.2 Розробка концепції гри. 2.2.1 Сюжет та історія гри. 2.2.2 Механіки та геймплей. 2.3 Побудова архітектури гри на основі C# у Godot. 2.4 Висновок до другого розділу. 3 Реалізація відеогри за допомогою Godot з використанням мови програмування C#. 3.1 Розробка візуальної складової відеогри. 3.2 Створення ігрових сцен для відеогри «Paw Isle» у Godot. 3.3 Написання механік гри з використанням мови програмування C#. 3.4 Висновок до третього розділу. 4 Безпека життєдіяльності, основи охорони праці. 4.1 Актуальність безпеки життєдіяльності людини в цифровому середовищі. 4.2 Організація безпечної роботи електроустановок. 4.3 Інженерно-Технічного рішення з охорони праці розробника. Висновки. Перелік джерел.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	Сенчишин В.С.	09.06.2025	13.06.2025

7. Дата видачі завдання 07 травня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	07.05.2025	*Виконано*
2.	Підбір та опрацювання літературних джерел по темі кваліфікаційної роботи	08.05.2025-15.05.2025	*Виконано*
3.	Виконання дослідження щодо відеоігор на різних платформах	16.05.2025-22.05.2025	*Виконано*
	Розроблення відеогри		
4.	Оформлення розділу «Аналіз сфери розробки	23.05.2025-01.06.2025	*Виконано*
	відеоігор»		
5.	Оформлення розділу «Проектування відеогри у жанрі	23.05.2025-01.06.2025	*Виконано*
	симулятора фермерства»		
6.	Оформлення розділу «Реалізація відеогри за	23.05.2025-01.06.2025	*Виконано*
	допомогою Godot з використанням мови		
	програмування C#»		
7.	Виконання завдання до підрозділу «Безпека життєдіяльності»	02.06.2025-05.06.2025	*Виконано*
8.	Виконання завдання до підрозділу «Основи охорони	08.06.2025-09.06.2025	*Виконано*
	праці»		
9.	Оформлення кваліфікаційної роботи	16.06.2025-18.06.2025	*Виконано*
10.	Нормоконтроль	13.06.2025-16.06.2025	*Виконано*
11.	Перевірка на плагіат	18.06.2025	*Виконано*
12.	Попередній захист кваліфікаційної роботи	20.06.2025	*Виконано*
13.	Захист кваліфікаційної роботи	24.06.2025	

Студент _____ Базюк Р.Л.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Матійчук Л.П.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розробка відеогри на базі рушія Godot з використанням мови програмування C# // Кваліфікаційна робота освітнього рівня «Бакалавр» // Базюк Роман Любомирович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СН-41 // Тернопіль, 2025 // С. 57, рис. – 32, табл. – 1, слайди. – , додат. – 14, бібліогр. – 35.

Ключові слова: розробка відеоігор, Godot Engine, C#, 2D-гра, симулятор фермерства, ігровий дизайн, візуальна складова, ігрові механіки.

Кваліфікаційна робота присвячена дослідженню та розробці 2D відеогри у жанрі симулятора фермерства на базі рушія Godot з використанням мови програмування C#. В першому розділі кваліфікаційної роботи проведено огляд сучасного стану GameDev-індустрії. Здійснено порівняльний аналіз провідних ігрових рушіїв. Висвітлено основні переваги рушія Godot для інді-розробників та 2D симуляторів.

В другому розділі кваліфікаційної роботи обґрунтовано вибір жанру симулятора фермерства, досліджено причини популярності та привабливості цього жанру. Розроблено концепцію гри, що включає основний сюжет, ключові ігрові механіки та обраний візуальний стиль.

В третьому розділі кваліфікаційної роботи описано процес розробки відеогри. Детально розглянуто створення візуальної складової, включаючи 2D-спрайти, об'єкти середовища, анімації персонажа та ігрових елементів. Описано логіку та взаємодії гравця з предметами довкілля. Об'єкт дослідження кваліфікаційної роботи – процес розробки 2D відеогри у жанрі симулятора фермерства. Предмет дослідження кваліфікаційної роботи – інструментарій та методи розробки відеоігор на базі рушія Godot з використанням мови програмування C#.

ANNOTATION

Development of a Video Game Based on the Godot Engine Using the C# Programming Language // Qualification work of the educational level «Bachelor» // Baziuk Roman // Ternopil Ivan Pulyu National Technical University, Computer and Information Systems and Software Engineering Faculty, Computer Sciences Department, group SN-41 // Ternopil, 2025 // P. 57, fig. – 32, tabl. –1, slides. – , annexes. – 14, references – 35.

Keywords: game development, Godot Engine, C#, 2D game, farming simulator, game design, visual design, game mechanics.

This qualification work is dedicated to the research and development of a 2D video game in the farming simulator genre, based on the Godot engine and using the C# programming language.

The first section of the qualification work provides an overview of the current state of the GameDev industry. A comparative analysis of major game engines is presented, highlighting the advantages of Godot for indie developers and 2D simulators.

The second section of the qualification work justifies the choice of the farming simulator genre and examines the reasons behind its popularity and appeal. A game concept is developed, which includes the main plot, core game mechanics, and a chosen visual style.

The third section of the qualification work describes the actual game development process. It details the creation of the visual components, including 2D sprites, environment objects, character animations, and other game elements. The logic of player interactions with the environment is also discussed. Object of the study: the process of developing a 2D video game in the farming simulator genre. Subject of the study: tools and methods for developing video games using the Godot engine and the C# programming language.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

HUD (Heads-Up Display) – екран, що відображає інформацію.

NPC (Non-player character) – не ігровий персонаж.

UE (Unreal Engine) – ігровий рушій.

UI (User Interface) – відповідає за візуальний вигляд та функціональність інтерфейсу.

ООП – Об’єктивно орієнтоване програмування.

ЗМІСТ

ВСТУП

Актуальність теми. Внаслідок швидкого розвитку технологій, зростання доступності інструментів для розробки та підвищення попиту на інтерактивний контент, створення якісних ігор перестало бути прерогативою лише великих студій. Особливої популярності набувають інді-проекти та ігри у нішевих жанрах, таких як симулятори, що пропонують гравцям глибоке занурення та унікальний досвід. Симулятори фермерства, зокрема, здобули широке визнання завдяки своїй здатності створювати затишну, медитативну атмосферу та надавати гравцям відчуття прогресу та досягнення.

Мета і задачі дослідження. Метою даної кваліфікаційної роботи освітнього рівня «Бакалавр» є розробка функціональної 2D відеогри у жанрі симулятора фермерства на базі рушія Godot з використанням мови програмування C#, що забезпечить цікавий ігровий процес та продемонструє можливості обраних технологій. Для досягнення поставленої мети потрібно виконати ряд завдань, зокрема:

- Проаналізувати сучасний стан GameDev-індустрії, існуючі жанри відеоігор та особливості їхнього розвитку.
- Дослідити та порівняти популярні ігрові рушії (Unity, Unreal Engine, Godot) з метою обґрунтування вибору Godot для 2D-розробки.
- Оглянути існуючі рішення та успішні проекти у жанрі симуляторів фермерства, визначити їхні ключові особливості та механіки.
- Розробити детальну концепцію гри, включаючи сюжет, основні ігрові механіки, цільову аудиторію та візуальний стиль.
- Спроектувати архітектуру гри та структуру її сцен, використовуючи принципи розробки на Godot з C#.
- Реалізувати візуальну складову гри, включаючи створення 2D-спрайтів, тайлових карт, анімацій персонажа та об'єктів, а також налаштування камери, освітлення та пост-обробки.

- Запрограмувати ігрову логіку та взаємодію гравця з об'єктами світу, включаючи систему завдань та прогресу.
- Розробити користувацький інтерфейс, включаючи головне меню, екран завантаження та вікно налаштувань.

Практичне значення одержаних результатів. Отримані результати кваліфікаційної роботи мають значне практичне значення, оскільки вони демонструють можливість створення повноцінної 2D відеогри у популярному жанрі симулятора фермерства з використанням відкритого рушія Godot та мови програмування C#. Розроблений проект може слугувати основою для подальшого розвитку, розширення функціоналу та комерціалізації. Набутий досвід у проектуванні ігрових механік, роботі з 2D-графікою, анімацією, розробці UI та програмуванні на C# у Godot є цінним для фахівців у галузі GameDev. Крім того, розроблена гра може стати інструментом для демонстрації потенціалу Godot Engine як конкурентоспроможного рішення для розробки інді-ігор.

Таким чином, результат цієї роботи не лише підтверджує ефективність використання Godot та C# для створення сучасних 2D-ігор, але й можуть бути практично застосовані у навчальному процесі, портфоліо розробника або як основа для запуску стартапу в індустрії відеоігор. Вони відкривають широкі можливості для подальших досліджень, вдосконалення технічних рішень та реалізації творчих ідей у сфері геймдеву.

РОЗДІЛ 1. АНАЛІЗ СФЕРИ РОЗРОБКИ ВІДЕОІГОР

1.1 Огляд сучасного стану GameDev-індустрії

Індустрія розробки відеоігор постійно зростає і стає одним з найдинамічніших секторів світової економіки. У 2024 році ігровий ринок оцінювався в 446,31 мільйона доларів США, а до 2033 року прогнозується зростання до 1198,43 мільйона доларів США [1]. Загальний ігровий ринок також швидко зростає, очікується, що до 2029 року він досягне 562,87 мільярда доларів США. За даними показниками можна оцінити, наскільки швидко та активно розвивається ігрова індустрія.

Загалом ринок відеоігор можна поділити на кілька основних платформ. Одним із ключивих є ПК-ігри, які займають 40% ринку, що пояснюється великим попитом на кіберспорт, покупки сильніших, так званих, ігрових комп'ютерів та самими користувачами, які створюють різні модифікації для ігор. Steam є також головною платформою для ПК-ігор, на яку припадає приблизно 70% усіх продаж. У свою чергу на мобільні ігри виходить близько 35% ринку, їхній розвиток підтримується за рахунок безкоштовних моделей, внутрішньоігровими покупками та хмарними ігровими сервісами, які зараз так швидко розвиваються. Android та iOS платформи разом відповідають за 90% усіх завантажень мобільних ігор. У середньому, щодня на мобільних пристроях грають 45 хвилин. Мобільний сегмент є лідером за доходом, заробивши 92,6 мільярда доларів США у 2024 році [2]. Головною причиною такого різкого зростання стало через великий попит на самі смартфони, наприклад, у Великобританії, де 91,43% сімей користуються смартфонами у 2021 році [3]. Також можна згадати і за ігри для телевізорів, які в себе включають консолі та інтеграцію з різними смарт-ТВ або станціями, займаючи при чому 15% ринку. Самі по собі консольні платформи, наразі, мають менший попит у 2024 році, хоча очікується, що у 2025 році консолі можуть стати кращими ніж ПК-ігри за

доходом. Ймовірно ринок ігрових консолей, зросте до 55,36 мільярда доларів США до 2032 року за прогнозами відповідних ресурсних даних, а все це тому, що сильно розвиваються такі напрямки, як: онлайн-мультиплеєр, кіберспорт, хмарний геймінг та розвиток VR/AR технологій [4].

На даний момент сучасний ринок розробки ігор формується завдяки кільком ключовим тенденціям:

- Інтеграція штучного інтелекту, яка на даний момент зросла до 33% розробників, які використовують інструменти на основі ШІ для підвищення ефективності роботи. Якщо брати більш широке опитування, то отриманий результат становитиме близько 96%, що студії інтегрують інструменти ШІ у свої робочі процеси, як правило, це для вирішення різного роду простих проблем, а не для написання чогось нового для полегшення розробки.
- Хмарний геймінг показує значне зростання на 40%, що в свою чергу дозволяє користувачам транслювати високоякісні ігри без зайвої необхідності придбанні дорогого обладнання.
- Крос-платформна розробка є також важливою тенденцією, оскільки близько 67% розробників вибирають дане рішення, щоб зменшити втрати та забезпечити доступність гри для різних платформ.
- Ремейки та ребути є досить прибутковою тенденцією в ігровій індустрії, адже даний метод напряму впливає на вже сформовану фан-базу, що набагато спрощує маркетинг продукту.
- Ранній доступ став невід'ємною частиною процесу розробки ігор. Це зв’язано з тим, що багато незалежних розробників не мають достатнього фінансування для завершення свого проекту. Також це потрібно, щоб зрозуміти про можливий попит на гру завдяки зворотному зв’язку від спільноти.

В загальному зростання інді-розробки сильно вплинуло на збільшення кількості платформ для розробки. Такі інструменте, як Unity, Unreal та Godot, суттєво знизила витрати та час на розробку відеоігор, дозволяючи розробникам більше зосередитися на творчості, цікавих механіках та сюжеті. Godot виділяється як безкоштовний, гнучкий та легкий варіант, ідеальний для інді-

розробників. Дане програмне забезпечення значно зменшує бар’єр для входу в середовище розробки, що в свою чергу призводить до створення більш унікальних механік, сюжетних ліній та художним стилем. Завдяки такому інструменту менші команди та окремі розробники мають можливість створювати високоякісні ігри, які часто оцінюються в AAA-проектами [5]. У даному середовищі, яке є більш сприятливим для творчих експериментів та інновацій у геймплеї, оскільки інді-розробники є менш обмеженими комерційним тиском або усталеними формулами великих студій.

GameDev-індустрія часто стикається з новими викликами, включаючи звільнення, які торкнулися близько 11% розробників ігор. Фінансові обмеження є основною і найбільшою проблемою для більшості розробників, при цьому самі витрати на розробку ігор значно зросли на 30% за останні три роки через великий попит на високоякісну графіку та інтеграцію ШІ. Інді-розробникам, які часто працюють з обмеженими ресурсами, важко встигати за сучасними тенденціями, що може суттєво перешкоджати довгостроковому успіху. Також економічні спади та інфляція можуть зменшити доходи споживачів, що може серйозно вплинути на продажі навіть для недорогих інді-ігор [6].

1.2 Аналіз рушіїв для розробки відеоігор

Перед початком розробки відеогри, спершу потрібно вибрати підходящий ігровий рушій, який буде основою для подальшого проектування. На ньому будуть базуватись увесь проект, від базової логіки та обробки подій до графіки, фізики та анімацій. Якщо розглядати рушії для 2D ігор, які розробляються на C#, особливо важливо врахувати їхні можливості у цьому напрямку, підтримку мови програмування, ліцензійні умови та активність спільноти. Для порівняння було вибрано три провідних ігрових рушіїв: Unity, Unreal Engine та Godot.

Розпочну аналіз з Unity, який є одним із самих популярних та багатофункціональних ігрових рушіїв. Він пропонує широкі можливості для

розробки, як 3D-ігор, так і 2D. У контексті 2D розробки в ньому багато різних функцій та спеціальних пакетів, такі як: 2D Animation, 2D Pixel Perfect та 2D Tilemap Editor, що значно полегшує створення візуальних ефектів та привабливих сцен. Рушій підтримує імпорт багатошарових зображень, покадрову та кісткову анімацію персонажів, а також надійну 2D-фізичну систему. Unity також вирізняється активною глобальною спільнотою та великою кількістю ресурсів для навчання, що робить його привабливим вибором для розробників. Однак, незважаючи на безкоштовну версію Unity Personal для невеликих проектів та скасування суперечливої «Runtime Fee», недавні зміни в моделі ціноутворення могли викликати певну недовіру серед інді-розробників, змушуючи їх розглядати альтернативи [7].

Unity Personal залишається безкоштовним для організацій з річним доходом та фінансуванням менше 200 000 доларів США. Для досвідчених команд та сольних розробників, що перевищують цей поріг, Unity Pro коштуватиме 2200,00 доларів США на рік за місце. Unity Enterprise доступний для великих організацій з річним доходом та фінансуванням понад 25 мільйонів доларів США, з індивідуальним ціноутворенням [8].

Наступним для порівняння буде рушій Unreal Engine (UE), який в першу чергу був розроблений для високоякісних 3D-ігор і відомий своєю реалістичною та насиченою графікою. Він також надає спеціалізований набір інструментів для 2D-розробки ігор під назвою Paper 2D [9]. Paper 2D підтримує спрайтову графіку, фізику, виявлення зіткнень та тайлові карти, дозволяючи розробнику створювати візуально красиві та механічно насичені 2D-ігри.

UE пропонує два можливих підходи до програмування, надаючи розробникам вибір між C++ та Blueprint Visual Scripting. C++ дозволяє контролювати функціональність гри та високоякісну візуалізацію, але для його використання потрібно мати чималий досвід роботи з ним, що в свою чергу робить його більш придатним саме для досвідчених розробників [10]. Blueprint Visual Scripting - це система на основі вузлів, яка спрощує розробку ігор, дозволяючи створювати складні ігрові механіки, елементи інтерфейсу та

анімації без написання великого коду для цього. Багато студій застосовують гібридний підхід, використовуючи Blueprints для швидкого прототипування, а потім допрацьовують елементи, що вимагають високої продуктивності на C++. Рендеринг у реальному часі в UE прискорює розробку, дозволяючи миттєво візуалізувати зміни [11].

UE є безкоштовним інструментом для студентів, викладачів, любителів та більшості компаній, що не займаються іграми, з річним валовим доходом менше 1 мільйона доларів США. Для розробників ігор та інших користувачів, які поширюють програми, яких продажі перевищують 1 мільйон доларів США валового доходу за весь термін продукту, стягується 5% роялті [12].

UE має велику та активну спільноту розробників, що налічує сотні тисяч розробників. Завдяки ним, створюється багато навчального матеріалу. Вони також часто обмінюються знаннями, допомагають з діагностикою поширених проблем та дають відповіді на форумах.

UE, попри його великі можливості та включення Paper 2D, має в собі певні проблеми, що стосується витрат при використанні для 2D-розробки через його дизайн як 3D-рушія. Це означає, що розробники витрачають більше часу на оптимізацію та навігацію через складний інтерфейс, який є доволі складним для простих вимог 2D-завдань. Крім того, інтерфейс та робочий процес є значно складнішим порівняно з такими рушіями, як Unity та Godot, що в свою чергу вимагає багато часу на освоєння.

Розглянув останню альтернативу, а саме Godot Engine, який вирізняється спеціалізованим та нативним 2D-робочим процесом. Нативний підхід означає, що рушій не просто підтримує 2D, а є повноцінним та оптимізованим інструментом для розробки, де всі ключові функції розраховані під двовимірну графіку. Godot включає в себе вбудований редактор тайлових карт, що спрощує створення 2D-світів як за допомогою процедурної генерації, так і за допомогою ручного створення [13]. Його інтерфейс можна описати, як дуже простий та інтуїтивний, а його 2D-реалізація сприймається набагато краще порівняно з Unity.

Godot пропонує підтримку платформи .NET, що дозволяє розробникам використовувати свій досвід роботи з C#, застосовувати привичні бібліотеки та при цьому підвищувати продуктивність. C# є строго типізованою мовою, що допомагає зменшити випадкові помилки та сприяє глибшому розумінню концепцій об'єктно-орієнтованого програмування (ООП). C# у Godot також надає доступ до великої кількості інструментів та дозволяє інтегрувати зовнішні бібліотеки чи пакети [14].

Godot є повністю безкоштовним та з відкритим вихідним кодом. Тобто у ньому відсутні приховані платежі, роялті чи договірні зобов'язання, надаючи розробникам повну свободу використання, модифікації та розповсюдження своїх проектів або навіть самого рушія [15].

Отож, рішення обрати саме рушій Godot було прийняте після ретельного аналізу наявних альтернатив, кожна з яких має свої переваги та сильні сторони. На сучасному ринку представлено низку сильних ігрових рушіїв, які не поступаються за функціональністю та підтримкою, однак саме Godot продемонстрував найбільшу відповідність вимогам до розробки 2D-гри з використанням мови програмування C#. Найкращим аргументом вибору є порівняльний аналіз із конкурентами, що дозволяє об’єктивно оцінити переваги та недоліки кожного з рушіїв. У таблиці 1.1 представлено узагальнене порівняння популярних ігрових рушіїв.

Таблиця 1.1 – Порівняння популярних ігрових рушіїв

Характеристика	**Unity**	**Unreal Engine**	**Gogot**
1	**2**	**3**	**4**
Мова програмування	C#	C++/Blueprint Visual Scripting	GDScripts/ C#
Підхід до 2D розробки	Адаптивний (3D-основа з 2D-інструментами)	Адаптивний (3D-основа з 2D-інструментами)	Нативний 2D

Продовження таблиці 1.1

1	2	3	4
Ключові 2D інструменти	2D Animation, Tilemap, SpriteShape, 2D Physics	Paper 2D (Sprite, Tilemap, Physics), Blueprint Visual Scripting	Tile Map Editor, 2D Lights, 2D Skeletons, Pixel-Based Unit System
Модель ліцензування	Freemium (Personal, Pro, Enterprise)	Freemium/Royalty	MIT License (повністю безкоштовно, без роялті)
Розмір інсталяції (орієнтовно)	<10ГБ	>40 ГБ	<100МБ
Легкість вивчення	Середня(розуміти адаптацію 3D для 2D)	Висока (складний інтерфейс, 3D-фокус)	Легка (інтуїтивний 2D-робочий процес)
Переваги для 2D ігор	Широкий набір інструментів, велика бібліотека ассетів	Потужні інструменти для візуалізації, гнучкість C++/Blueprint	Легкість, нативний 2D, низькі системні вимоги, безкоштовність

Отож, проаналізувавши всіх кандидатів і беручи до уваги, що апаратні ресурси обмежені, все ж рішенням буде вибрати ігровий рушій Godot для розробки відеогри.

1.3 Огляд існуючих 2D симуляторів фермерства

Для того, щоб зрозуміти, які саме проекти є привабливими для користувачів, потрібно проаналізувати успішні існуючі 2D симулятори фермерства та схожих ігор. Це важливий етап, адже потрібно оцінити продукти та придумати, щось особливе для залучення гравців.

Один із самим популярних рішень у даному контексті є «Stardew Valley», який широко відомий своєю здатністю створювати відчуття теплого світу, що

характеризується мирною атмосферою та чарівною піксельною графікою. Успіх даного проекту напряму походить від майстерного поєднання розслаблюючого геймплею зі значною глибиною, що дозволяє гравцям прогресувати у власному темпі без відчуття поспіху. Гра дуже захоплююча, адже вона керована позитивним циклом зворотного зв'язку, який ніколи не закінчується, що забезпечує постійне залучення та відчуття досягнення навіть після незначних невдач. Багатокористувацький режим ще більше підвищує задоволення, дозволяючи друзям разом працювати над різними фермерськими роботами та дослідженнями. Гра має характерний та чарівний піксельний художній стиль, доповнений красивими пейзажами, які змінюються з кожною порою року, що значно сприяє її затишній та привабливій атмосфері. «Stardew Valley» пропонує багатий набір цікавих ігрових механік. Вони включають основні фермерські дії, такі як: посадка, полив, збір різних культур, риболовлю, видобуток ресурсів, розширений крафтинг та налагодженню стосунків з різноманітними мешканцями міста. Гравці постійно мотивовані чіткою системою прогресу, з постійними цілями, такими як: модернізація своєї ферми, завершення громадського центру та дослідження нових територій.

Комерційний успіх «Stardew Valley» чітко демонструє, що для 2D фермерських симуляторів затишна або розслаблююча атмосфера є не просто поверхневим візуальним вибором, а ключовим етапом, що лежить в основі залучення гравців. «Stardew Valley» хвалять за мирну атмосферу, чарівний піксельний арт та загальне затишне відчуття.

Наступним об’єктом дослідження буде не менш відома відеогра, франшиза якої виникла в Японії у 1996 році як «Bokujō Monogatari» (що означає «фермерська історія»), з основною метою – відновити занедбану ферму до успішного підприємства. З часом серія розвивалася і зрештою розділилася на дві окремі франшизи: «Harvest Moon» та «Story of Seasons».

Основні ігрові механіки включають в себе фермерство, а саме: посадка, полив, збір різних культур, управління часом та побудову стосунків з мешканцями міста. Пізніші впровадили більш складні елементи, такі як:

розведення худоби, шлюб, сімейне життя та навіть дослідження відкритого світу з мобільними фермами.

Успіх «Harvest Moon» полягає у її здатності створювати відчуття прогресу та досягнення через виконання повсякденних завдань. Навіть найпростіші дії, як-от посадка моркви чи доїння корови, приносять відчуття задоволення, а довгострокові цілі, такі як розширення ферми або створення сім'ї, надають глибину геймплею [16]. Серія також була високо оцінена за її розслаблюючу атмосферу, що дозволяє гравцям грати у власному темпі без тиску. Ця формула виявилася надзвичайно успішною, продемонструвавши, що ігри про фермерське життя можуть бути не тільки розважальними, а й емоційно привабливими, створюючи затишний і приємний ігровий досвід, який продовжує надихати розробників і гравців по всьому світу.

Останнім прикладом дослідження буде «Graveyard Keeper» – це симулятор управління кладовищем, натхненний «Stardew Valley» та заснований на «Harvest Moon». Гра вирізняється своєю макабричною атмосферою та нестандартними діями, що є нечастим явищем в іграх. Візуальний стиль описується як GBA-стиль та милий, різноманітний та невідповідно темний , що створює унікальне поєднання. Графіка доповнює її вміщений відкритий світ, і вплив старих ігор «Zelda» простежується так само сильно, як і «Harvest Moon» або «Stardew Valley» [17].

Основні особливості геймплею «Graveyard Keeper» включають управління кладовищем, крафтинг, дослідження та побудову стосунків, які взаємопов'язані та є важливими для прогресу в грі. Гравець бере на себе роль людини, яка потрапила в аварію та прокинулася в незнайомому середньовічному фентезійному світі, де йому доручено доглядати за місцевим кладовищем та церквою. Крім догляду за могилами та очищення навколишнього середовища, гра також включає фермерство, ковальство, багаторівневі підземелля, риболовлю та розширену систему крафтингу. Прогрес у грі тісно пов'язаний з розблокуванням нових технологій через

«Дерево Технологій», що вимагає трьох типів очок досвіду: червоних, синіх та зелених.

У підсумку проробленого аналізу серед популярних 2D-симуляторів фермерства, таких як: «Stardew Valley», «Harvest Moon» та «Graveyard Keeper», дозволив виділити ключові елементи, які вплинули на їхню популярність серед гравців. Затишна атмосфера, приємна піксельна графіка, свобода дій, постійне відчуття прогресу, а також комбінація фермерських, соціальних та дослідницьких механік – все це робить гру привабливою. Дані проекти цього жанру демонструють, що саме поєднання спокійного геймплею з глибиною ігрових систем є основою для створення захопливого користувацького досвіду.

1.4 Висновок до першого розділу

У першому розділі кваліфікаційної роботи було проведено комплексний аналіз сучасної GameDev-індустрії, розглянуто її поточний стан, ключові тенденції, а також виконано порівняльний аналіз провідних ігрових рушіїв та успішних 2D-симуляторів фермерства.

Далі було проаналізовано популярні рушії – Unity, Unreal Engine та Godot з точки зору їхньої ефективності в розробці 2D-проектів із використанням C#, і на цій основі обґрунтовано вибір Godot як оптимального інструменту. Завершальною частиною став огляд успішних 2D-симуляторів фермерства, зокрема «Stardew Valley», «Harvest Moon» і «Graveyard Keeper», що дозволив виокремити ключові елементи, які приваблюють гравців, а саме: спокійна атмосфера, геймплейна прогресія та збалансоване поєднання фермерських і соціальних механік.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ВІДЕОГРИ У ЖАНРІ СИМУЛЯТОРА ФЕРМЕРСТВА

2.1 Дослідження та вибір жанру симулятора фермерства

Для початку потрібно чітко розставити цілі проекту. Це необхідно для кращої реалізації продукту. Розробка гри відбуватиметься у жанрі симулятор, тобто у даному форматі буде показано імітацію дій, як у реальному житті, але зі спрощенням. Зазвичай у симуляторах немає чітких цілей, а гравцям надається свобода вільно досліджувати середовище в якому вони перебувають.

Для проекту було обрано жанр симулятор фермерства. Зазвичай він поєднує елементи соціальної та фермерської симуляції. У таких іграх головний герой переїжджає в сільську місцевість і береться за ферму, часто через спадщину або через нудьгу від міського життя. Гравець вирощує врожай та розводить худобу, щоб заробляти гроші, а також може взаємодіяти з різноманітними персонажами, що живуть у місті. Сюжети часто включають відродження занепадаючих містечок.

Привабливість симуляторів фермерства полягає в їхній простоті та здатності викликати інтерес. Ці ігри стали несподіваним хітом у світі геймінгу, захопивши увагу гравців, а їхній шарм полягає в простоті та відчутті досягнення, яке вони пропонують. Вони є мирною альтернативою швидким та напруженим іграм, пропонуючи заспокійливий ритм догляду за врожаєм, спостереження за його ростом та збирання врожаю, що допомагає гравцям розслабитися та зняти стрес. У таких іграх немає перемог чи поразок, що усуває страх розчарування та дозволяє грати в розслабленому режимі без напруги .

Відчуття прогресу є ще одним ключовим аспектом перетворення невеликої, занедбаної ділянки землі на процвітаючу ферму є неймовірно цікаво. Кожне невелике досягнення, будь то вирощування першої партії моркви або придбання нового трактора, відчувається як велика перемога. Саме такі

моменти прогресу утримують гравців, оскільки завжди є до чого йти. Даний жанр дозволяє втекти від реальності, замінивши все на спів птахів та красиві зелені поля. Відсутність конкуренції та щоденних зобов'язань робить гру справжньою розвагою [18].

2.2 Розробка концепції гри

Розробки концепції гри є ключовим етапом, що визначає її основні елементи, а саме: сюжет, ігрові механіки та візуальний стиль. Для 2D фермерського симулятора на Godot з використанням мови програмування C# ці аспекти мають бути ретельно продумані, щоб створити захопливий та унікальний ігровий досвід.

2.2.1 Сюжет та світ гри

Сюжет гри розгортається на чарівному острові під назвою – «Paw Island» («Лапковий острів»), затишному куточку світу, де панують мир та гармонія. Колись цей острів був процвітаючим осередком котячого фермерства, відомим своїми неперевершеними врожаями та щасливими тваринами. Проте, з часом, через таємничий катаклізм, який місцеві жителі називають «Великим Шелестом», більшість полів заросла, ферми занепали, а старі традиції майже забулися. Колись жваве селище «Pawville» спорожніло, залишивши лише декількох жителів, які з сумом згадують колишню славу острова.

Головний герой, молодий та енергійний котик на ім'я «Лапчик», прибуває на «Paw Isle», наслідуючи старий, занедбаний будиночок і невелику ділянку землі від свого дідуся фермера. «Лапчик» мріє відродити сімейну ферму та повернути острову його колишню велич. Його завдання - не лише обробляти землю, вирощувати культури та доглядати за тваринами, але й допомогти місцевим жителям, розкрити таємницю «Великого Шелесту» та відновити довіру й співпрацю серед мешканців «Pawville».

Острів буде розділено на кілька унікальних зон:

- «Ферма Лапчика» – початкова ділянка гравця, що потребує відновлення. На рисунку 2.1 продемонстровано створений макет для локації.

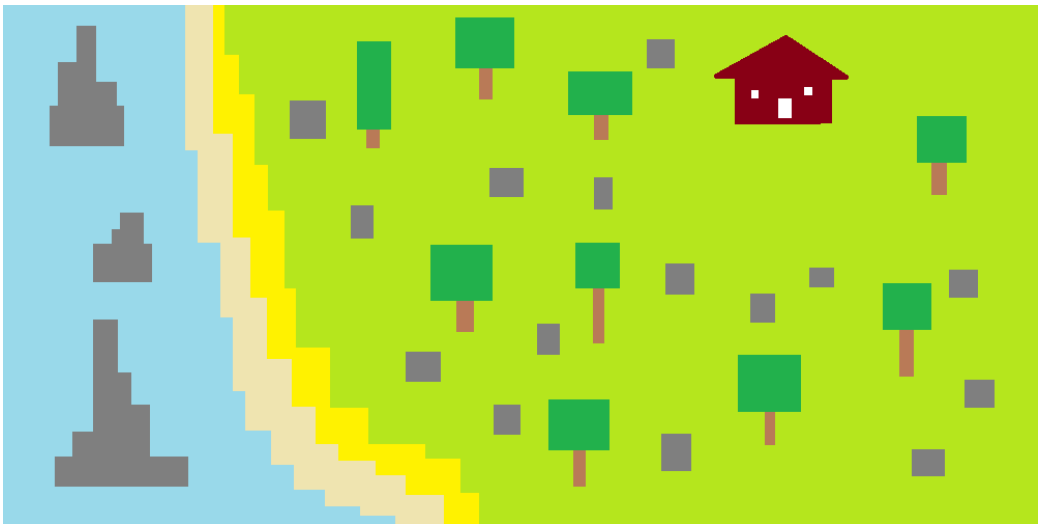

Рисунок 2.1 – Прототип локації «Ферма Лапчика»

- «Дальній пляж» – куди «Лапчик» прибуває на човні. Тут він вперше ступає на острів і знаходить дорогу до своєї занедбаної ферми. На рисунку 2.2 зображено прототип початкової локації.

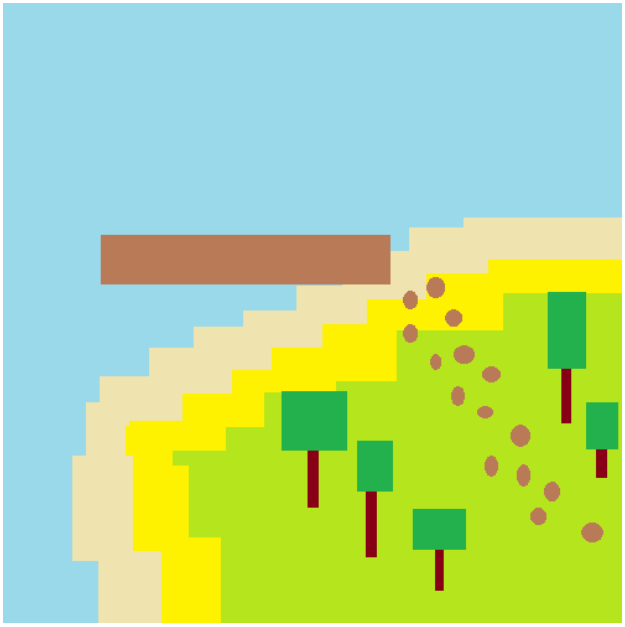

Рисунок 2.2 – Прототип локації «Дальній пляж»

- «Pawville» – невелике поселення з кількома будинками, крамничкою та таверною. На рисунку 2.3 зображено прототип локації містечка.

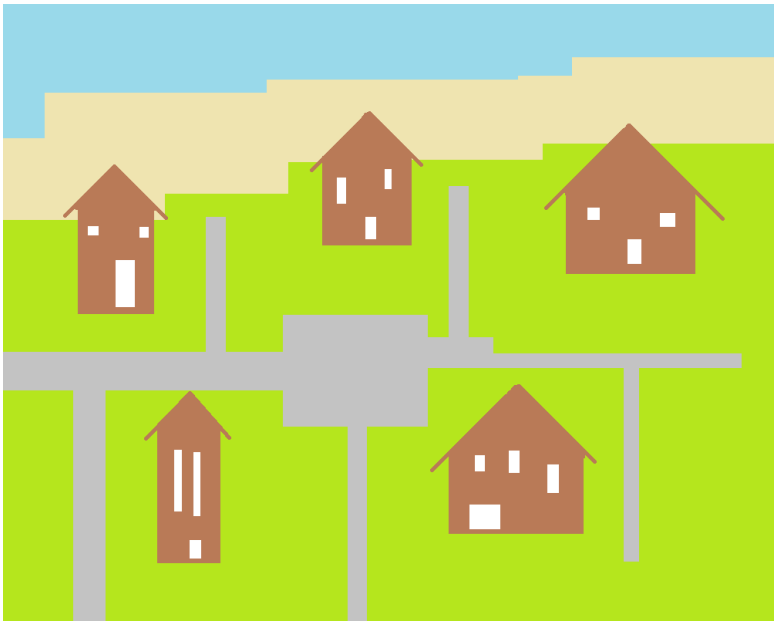

Рисунок 2.3 – Прототип локації «Pawvill»

- «Ліс Шепотів» – зарослий ліс на півночі, де можна знайти рідкісні ресурси. На рисунку 2.4 зображено прототип локації лісу.

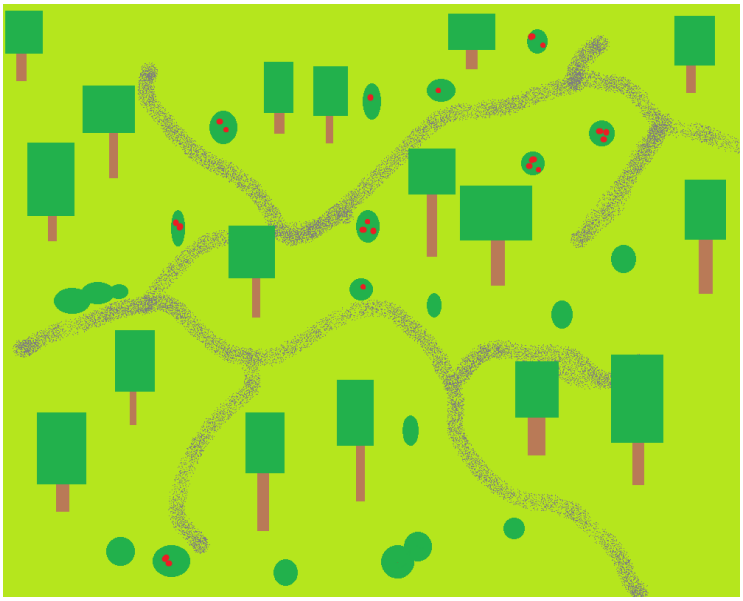

Рисунок 2.4 – Прототип локації «Ліс Шепотів»

- «Озеро Срібних Риб» – озеро, що кишить рибою, ідеальне для риболовлі. На рисунку 2.5 зображено прототип локації озера.

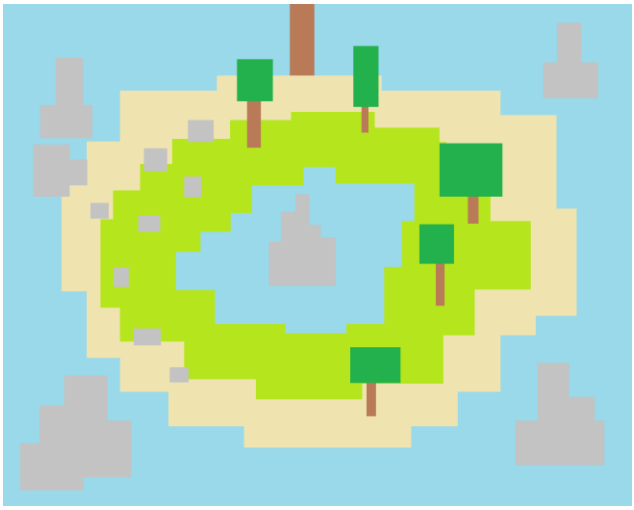

Рисунок 2.5 – Прототип локації «Озеро Срібних Риб»

- «Приховані Печери» – Таємничі підземелля, де можуть ховатися як цінні ресурси, так і небезпеки. На рисунку 2.6 зображено прототип локації з печерою.

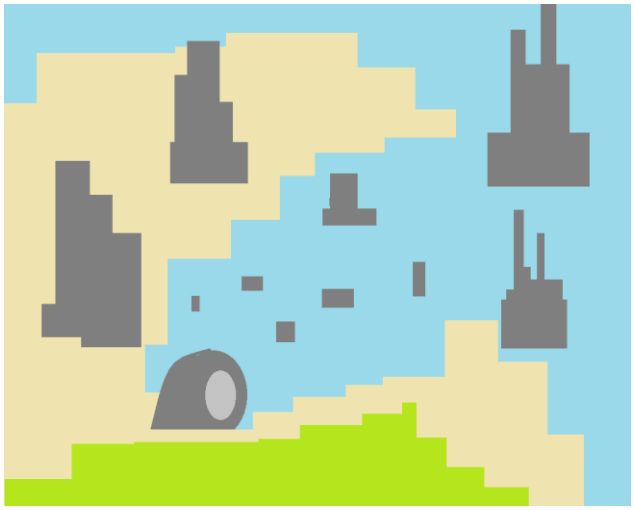

Рисунок 2.6 – Прототип локації «Приховані Печери»

Маючи приблизну структуру кожної унікальної зони острова, зробив загальну карту ігрового світу. Всі локації будуть мати свою особливість, хоч і

на прототипах цього не видно, але кожна сцена міститиме унікальні ресурси та різні красиві пейзажі.

Загальний макет карти острова (див. рис. 2.7) відображає логічне розташування всіх ключових зон. Подорож «Лапчика» починається на «Дальньому пляжі», звідки відкривається шлях до центрального поселення «Pawville». Звідти гравець отримує доступ до «Ферми Лапчика» та ресурсних локацій, таких як: «Ліс Шепотів» та «Озеро Срібних Риб», а також до таємничих «Прихованих Печер». Таке розташування дозволяє плавно розширювати ігровий процес, поступово відкриваючи нові локації та можливості для розвитку ферми.

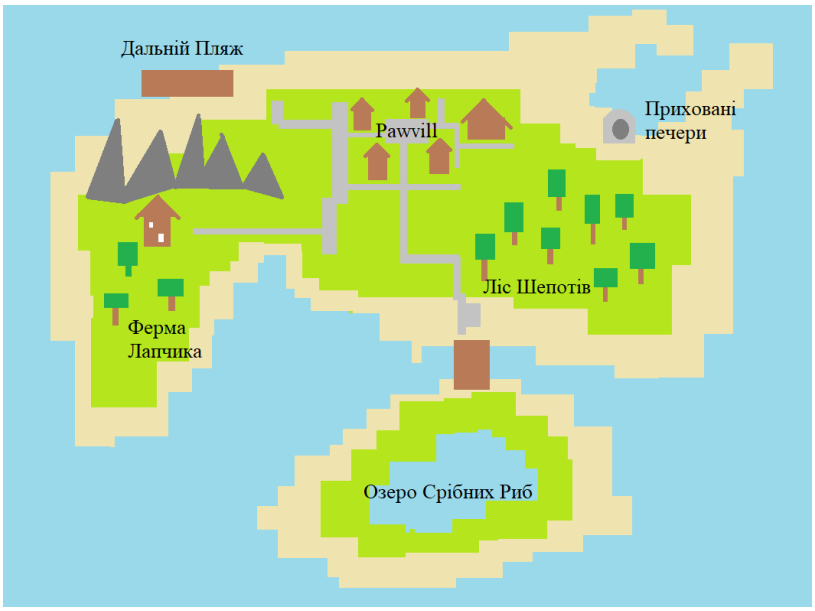

Рисунок 2.7 – Прототип загальної карти острова

Для прискорення розробки та фокусування на основних ігрових механіках, у проекті буде використано готовий шаблон спрайтів для персонажів та ігрових об'єктів. Цей підхід дозволяє швидко створити візуальну основу гри, не витрачаючи багато часу на розробку кожного графічного елемента з нуля.

Єдине, що буде виконано для забезпечення унікальності це – незначна перемальовка та адаптація вибраних спрайтів. Зокрема, для кожного котика-персонажа буде створено унікальні прототипи. Це дозволить гравцеві легко

розрізняти мешканців острова, зберігаючи при цьому цілісний візуальний стиль. Всі представлені графічні елементи на даному етапі є прототипами, призначеними для ілюстрації концепції та подальшої роботи.

Основні персонажі-котики та їхні імена:

- «Лапчик» – головний герой, молодий, сповнений ентузіазму котик, що прибув на острів. На рисунку 2.8 зображено прототип спрайта головного героя.

Рисунок 2.8 – Прототип спрайта «Лапчик»

- «Бабуся Мурка» – мудра стара кішка, що мешкає в «Pawville». Вона пам'ятає часи процвітання острова і дає «Лапчику» цінні поради та старі рецепти. Вона є хранителькою знань про острів. На рисунку 2.9 зображено прототип спрайта для «Бабусі Мурки».

Рисунок 2.9 – Прототип спрайта «Бабуся Мурка»

- «Кицюня» – енергійна та жвава молода кішечка, власниця місцевого магазину. Вона продає насіння, інструменти та інші необхідні товари. Кицюня завжди в курсі останніх пліток. На рисунку 2.10 зображено прототип спрайта для «Кицюні».

Рисунок 2.10 – Прототип спрайта «Кицюня»

- «Дядько Лапсон» – вельми поважний, але трохи буркотливий старий кіт, який колись був найкращим садівником на острові, але зараз занепав духом. Він неохоче ділиться своїми знаннями, доки «Лапчик» не доведе свою щирість. На рисунку 2.11 зображено прототип спрайта для «Дядька Лапсона».

Рисунок 2.11 – Прототип спрайта «Дядько Лапсон»

- «Мишко» – сором'язливий та працьовитий котик, який мешкає біля «Лісу Шепотів». Він майстер по дереву та може покращувати інструменти «Лапчика». Він також любить рибалити. На рисунку 2.12 зображено прототип спрайта для «Мишка».

Рисунок 2.12 – Прототип спрайта «Мишко»

- «Рудик» – трохи дивакуватий кіт-винахідник, що живе на околиці селища. Він створює дивні, але корисні пристрої та дає «Лапчику» унікальні завдання. На рисунку 2.13 зображено прототип спрайта для «Рудика».

Рисунок 2.13 – Прототип спрайта «Рудик»

Кожен з цих персонажів відіграватиме ключову роль у світі гри. Їхні унікальні особистості та взаємодії з головним героєм, не лише поглиблять сюжет, але й нададуть гравцеві можливість розвивати стосунки, отримувати допомогу та розкривати таємниці острова, створюючи насичений ігровий досвід.

2.2.2 Механіка та геймплей

Геймплей гри націлений на фермерській діяльності, дослідженні світу та взаємодії з його мешканцями, пропонуючи гравцеві цікавий досвід проведення часу. Основні механіки мають бути ретельно продумані для забезпечення балансу між повсякденними фермерськими справами та елементами пригод.

До основних механік гри будуть відноситись:

- Фермерство – це буде головною механікою, що включає повний цикл догляду за землею та вирощуванням культур. Гравець почне з обробки занедбаних ділянок на «Фермі Лапчика», використовуючи мотику для орання землі та лійку для її поливу. Доступні будуть різноманітні культури – від основних овочів до рідкісних фруктів та квітів, кожен з яких матиме свої вимоги до вирощування та час дозрівання. Зібраний врожай можна буде продавати для отримання прибутку або дарувати жителям острова для покращення відносин. Окрім рослинництва, важливим аспектом є догляд за тваринами. Гравець зможе розводити курей для отримання яєць, корів для молока та, можливо, інших тварин, що даватимуть цінні продукти. Кожна тварина потребуватиме регулярного годування та догляду, а їхнє благополуччя безпосередньо впливатиме на якість та кількість отримуваної продукції.
- Ресурси та крафтинг – дослідження острова тісно пов'язане зі збором ресурсів. Гравець зможе зрубувати дерева для отримання деревини, розбивати каміння для каменю та, можливо, добувати руду у Прихованих Печерах. Ці ресурси є основними для системи крафтингу. За допомогою верстака Лапчик зможе створювати нові інструменти, ремонтувати наявні та навіть будувати

нові споруди на своїй фермі, такі як курники, хліви чи комори, що дозволить розширювати виробництво.

- Дослідження світу є невід'ємною частиною геймплею. Починаючи з «Дальнього пляжу» та «Ферми Лапчика», гравець поступово відкриватиме нові локації на острові, такі як: «Озеро Срібних Риб» або «Прихованих Печер». Кожна нова зона приховує унікальні ресурси, нових NPC та, можливо, загадки чи секрети, пов'язані з історією острова. Крім того, гравець може натрапляти на приховані скарби, які містять рідкісні предмети, цінні рецепти, що стимулює допитливість та ретельне вивчення навколишнього середовища.

- Соціальна взаємодія з мешканцями «Pawville» та інших локацій відіграє ключову роль у просуванні сюжету та розвитку персонажа. Спілкування з NPC розкриватиме історію острова, надаватиме цінні поради та відкриватиме завдання (квести). Ці завдання можуть бути різноманітними: від простого прохання виростити певну культуру або знайти втрачений предмет до складніших місій, що розкривають таємницю «Великого Шелесту». Виконання квестів не лише приноситиме винагороди, а й покращуватиме стосунки з NPC, що може відкривати доступ до нових товарів у магазинах, унікальних рецептів чи особливих подій.

- Економіка та часовий цикл дозволяє гравцям торгувати зібраним врожаєм та ресурсами в магазині Кицюні в «Pawville», отримуючи кошти для подальшого розвитку. Зароблені гроші можна потратити, щоб перейти до системи покращень, такі як: купувати нові інструменти, розширювати ферму, будувати нові споруди або навіть покращувати свій будинок. Цикл дня і ночі додають реалізму геймплею. Час доби впливає на доступні можливості.

Отже, дана система механік, що включає в себе фермерство, дослідження, крафтинг, соціальну взаємодію та економіку, у поєднанні з циклом зміни дня і ночі, створює цікавий геймплей. Такий підхід забезпечує не лише захопливий ігровий досвід, але й дозволяє гравцеві відчути справжній прогрес, відновлюючи занедбану ферму та повертаючи життя на острів.

2.3 Побудова архітектури гри на основі C# у Godot

Побудова архітектури на базі рушія Godot з використанням мови програмування C# забезпечить гнучкість, модульність та ефективність розробки. Godot зі своєю унікальною архітектурою на основі вузлів та підтримкою C# добре підходить для створення 2D фермерського симулятора.

Рушій використовує архітектуру, де кожен елемент гри, будь то персонаж, об'єкт світу, світло чи звук, є вузлом (Node). Ці вузли організовані в ієрархічне дерево сцен, що забезпечує високу модульність та організованість проекту. Така структура є інтуїтивно зрозумілою та полегшує процес навчання, особливо для початківців. Вузли та сцени в Godot дозволяють створювати компоненти, які легко можна використовувати повторно та керувати ними, що є перевагою над традиційними префабами (prefabs) в інших рушіях.

Використання C# у Godot дозволяє розробникам використовувати свій досвід роботи з мовою програмування, застосовувати вже знайомі бібліотеки. C# є строго типізованою мовою, що допомагає зменшити кількість випадкових помилок та сприяє глибшому розумінню ООП. Вона надає доступ до інструментів та дозволяє інтегрувати зовнішні бібліотеки та пакети NuGet, значно розширюючи функціональність рушія. Починаючи з Godot 4.4, пакети C# орієнтовані на .NET 8, що забезпечує довгострокову підтримку та сумісність з новішими версіями .NET.

Як і будь-яка повноцінна відеогра, «Paw Isle» буде оснащений базовими функціями керування ігровим процесом. Проект передбачає можливість паузи в грі та зручного виходу до головного меню чи на робочий стіл у будь-який момент. Для забезпечення максимального комфорту гравця буде реалізовано вікно налаштувань, де користувач зможе регулювати рівні гучності фонової музики та звукових ефектів, а також налаштовувати клавіші управління для всіх ігрових дій персонажа.

Навігація гравця у відеогрі наведено в діаграмі на рисунку 2.14.

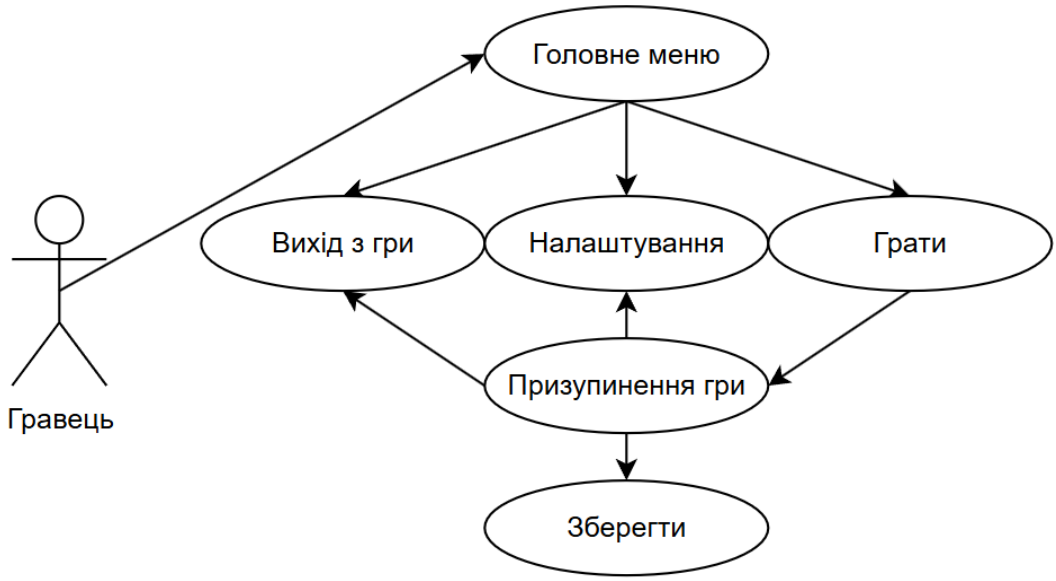

Рисунок 2.14 – UML-діаграма навігація у грі «Paw Isle»

Все це буде розбавлене анімаціями, приємним дизайном, звуками та музикою аби перебування на затишному острові передавало відчуття спокою та гармонії.

Тепер потрібно спроектувати початковий екран або, іншими словами, головне меню – це перше, що бачить гравець після запуску «Paw Isle», тому ця сцена має бути візуально привабливою та інтуїтивно зрозумілою. Її дизайн має одразу занурювати гравця в затишну атмосферу котячого фермерського симулятора. Головними інтерактивними елементами є кнопки, що дозволяють гравцеві:

- Розпочати гру.
- Завантажити існуючу гру.
- Відкрити налаштування.
- Вийти з гри.

Отож, користуючись програмою Krita, було розроблено прототип початкового екрану, який наведено на рисунку 2.15.

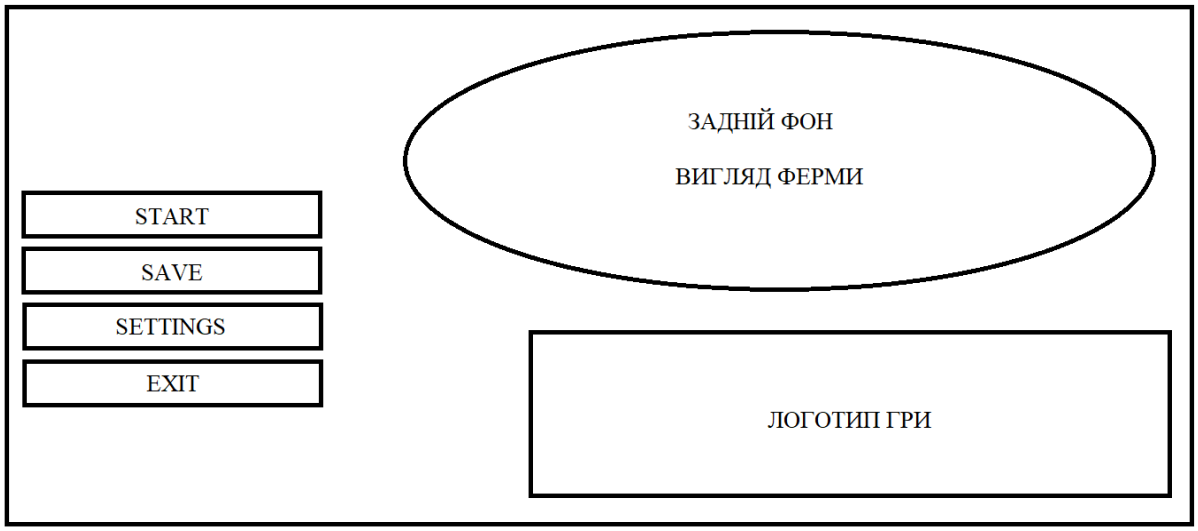

Рисунок 2.15 – Прототип початкового екрану «Paw Isle»

За допомогою даного прототипу можна легко розробити такий дизайн за допомогою інструментів Godot.

Наступним етапом проектування буде дисплей HUD (Heads-Up Display), який складається з кількох різних елементів інтерфейсу, які накладаються безпосередньо на ігровий екран. Основна мета HUD – надати гравцю важливу та актуальну інформацію про поточний стан гри та головного героя при цьому не заважаючи основному геймплею.

Для фермерського симулятора це включатиме:

- Час – відображає поточного часу доби, що є важливим для фермерських робіт.
- Гроші – поточний баланс гравця.
- Інвентар – швидкий доступ до предметів.
- Інструментарій – швидкий доступ до інструментів, якими володіє головний герой.

Отож, маючи такий набір потрібних елементів, можна проектувати прототип ігрового HUD (див. рис. 2.16).

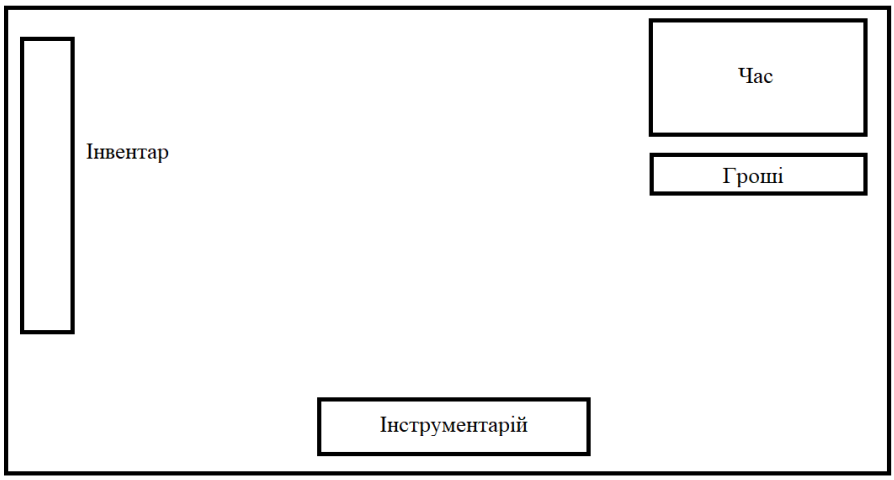

Рисунок 2.16 – Прототип HUD-а для відеогри «Paws Isle»

Тепер гравець з легкістю буде бачити всю потрібну йому інформацію на своєму екрані.

Тепер потрібно спроектувати меню параметрів гри або вікно налаштувань, є важливою складовою будь-якої відеогри, оскільки воно дає користувачам можливість змінювати різноманітні параметри відповідно до їхніх вподобань. Кількість доступних опцій може бути різною, проте головна мета завжди полягає у забезпеченні персоналізації та комфорту гравця.

У випадку «Paw Isle» буде реалізовано просте меню налаштувань, що дозволить гравцю:

- Регулювати рівні гучності – ця функція є важливою, адже не завжди стандартні рівні гучності фонової музики чи звукових ефектів є комфортними для гравця. Можливість їх зміни забезпечить приємніше занурення в атмосферу гри.

- Змінювати розкладку клавіш керування – кожен гравець має свої вподобання щодо цього. Надання можливості перепризначення клавіш дозволить адаптувати керування під індивідуальні потреби, усуваючи потенційний дискомфорт від стандартної розкладки.

Для візуалізації майбутнього вигляду вікна налаштувань було змодельовано його приблизний вигляд. Результат готового макету зображено на рисунку 2.17.

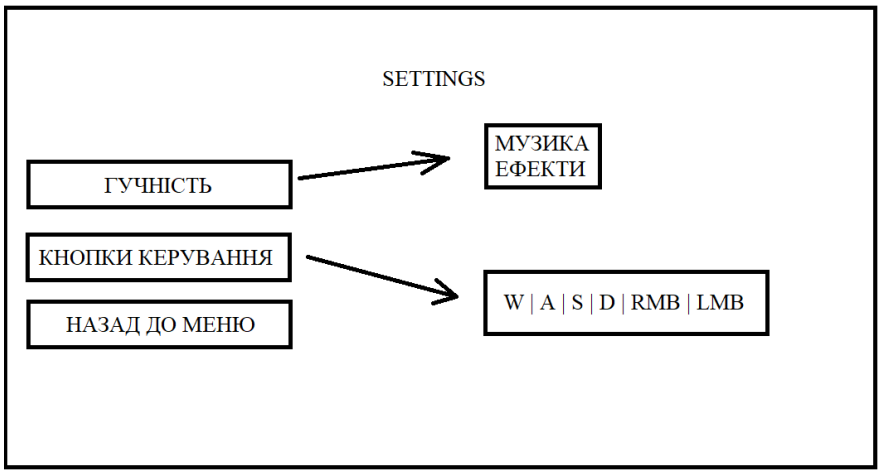

Рисунок 2.17 – Прототип вікна налаштувань для відеогри «Paw Isle»

Меню налаштувань дає великі можливості у разі ігрових доповнень, тому ним ніколи не потрібно нехтувати.

2.4 Висновок до другого розділу

У другому розділі кваліфікаційної роботи було детально проаналізовано процес створення 2D-симулятора фермерства «Paw Isle» на рушії Godot з використанням мови C#. Основну увагу приділено вивченню жанру симуляторів, його привабливості для гравців та особливостям реалізації ключових механік, таких як фермерство, крафтинг, дослідження світу, соціальна взаємодія та динамічна зміна дня і ночі.

Окремо було розглянуто візуальну складову гри, яка базується на піксельному стилі з адаптацією готових спрайтів. Спроектовано всі основні локації острова, кожна з яких має свою функціональну роль у геймплеї, а також розроблено інтерфейсні елементи: HUD, головне меню та меню налаштувань. Усі ці компоненти реалізовано у середовищі Godot, що ще раз підтверджує його ефективність і зручність у створенні 2D-ігор.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ВІДЕОГРИ ЗА ДОПОМОГОЮ GODOT З ВИКОРИСТАННЯМ МОВИ ПРОГРАМУВАННЯ C#

3.1 Розробка візуальної складової відеогри

Розробка візуальних компонентів є важливою частиною процесу створення гри. Вона включає в себе кілька етапів: концепт-арт, створення спрайтів та текстур, анімацію, візуальні ефекти та добавлення всіх компонентів у ігровий рушій. Саме завдяки графіці гравець отримує перше враження від гри, і саме вона формує атмосферу та емоційний зв’язок із віртуальним світом. Процес розробки візуальних елементів включає кілька етапів: від створення концепт-арту до фінальної реалізації спрайтів, анімацій та інтеграції графіки в рушій Godot.

Графіка – це все, коли йдеться про створення ігор. А у світі графіки можливості безмежні. Від яскравих і барвистих мультяшних ілюстрацій до більш докладних і реалістичних робіт. Ігровий художній стиль може сподобатися користувачам і покращити ігровий процес. Він дозволяє зануритися в історію з захоплюючими краєвидами, які переносять гравця у інший світ. Різні художні стилі гри дійсно вражають і дозволяють розробникам створювати унікальні графічні дива.

Серед візуальних робіт у проекті є такі елементи як: іконки для позначення UI-елементів та текстур, 2D-об’єкти середовища (тайли, спрайти), головний персонаж, анімації ходьби, фермерських дій. Саме завдяки ретельно продуманому стилю та якісній реалізації графіки гравець має змогу повністю поринути у гру, відчути її чарівність і провести в ній години із захопленням.

У проекті «Paw Isle», який розробляється у жанрі симулятора фермерства, має містити елементи, які відповідають ідеї гри. Дерева, каміння, квіти, трава, різні культури та тварини – все це необхідне для передавання відчуття усамітнення з природою та спокою.

Для розробки спрайтів було обрано програму Krita. На рисунку 3.1 наведено роботу над створенням невеликого дерева.

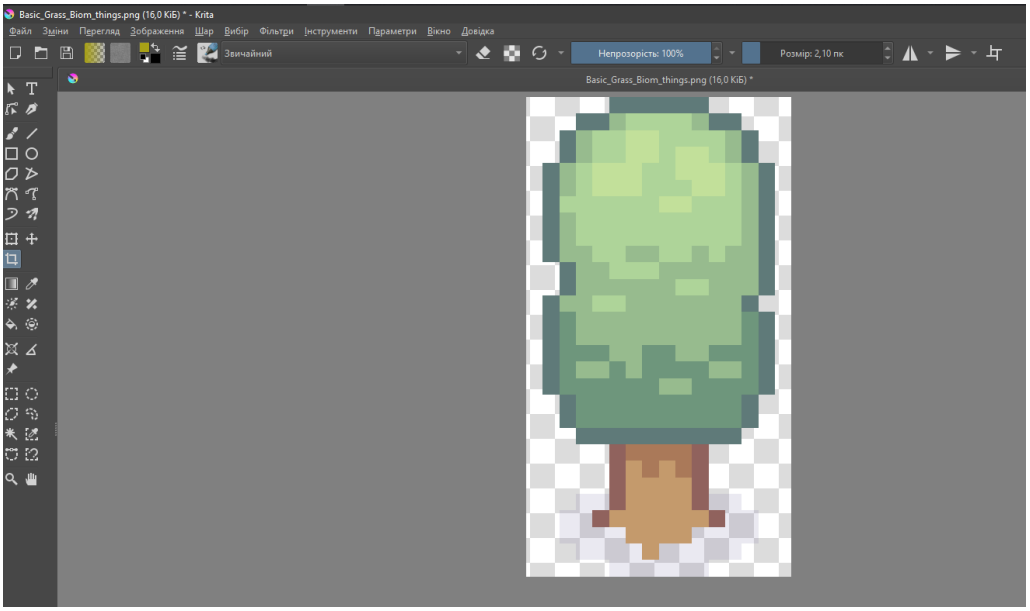

Рисунок 3.1 – Спрайт дерева розроблений у Krita

По аналогії до попереднього спрайту, було створено і наступні об’єкти, які різноманітять навколишнє середовище. На рисунку 3.2 зображено один із готових наборів спрайтів, які будуть використовуватись у грі.

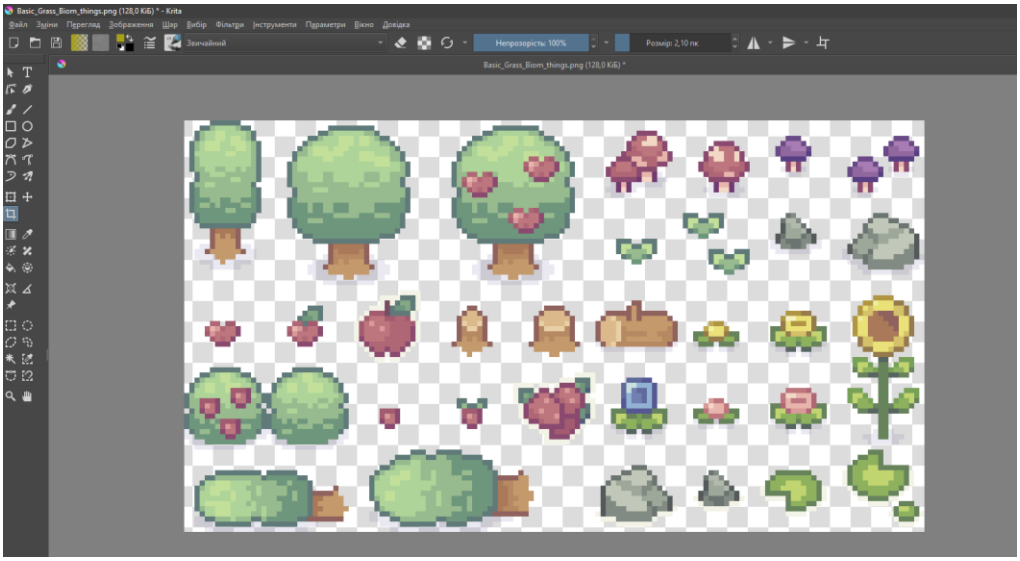

Рисунок 3.2 – Набір спрайтів розроблений у Krita

Даний набір містить мінімальну кількість різних елементів, але для початку цього буде достатньо, щоб трошки розбавити сцени гри. Якщо постаратись і розставити всі спрайти, то має вийти гарна візуальна картинка.

Всі 2D-спрайти збережено у форматі PNG та імпортовано в ігровий рушій Godot. На рисунку 3.3 зображено перенесені елементи у відповідну вкладку TileSets.

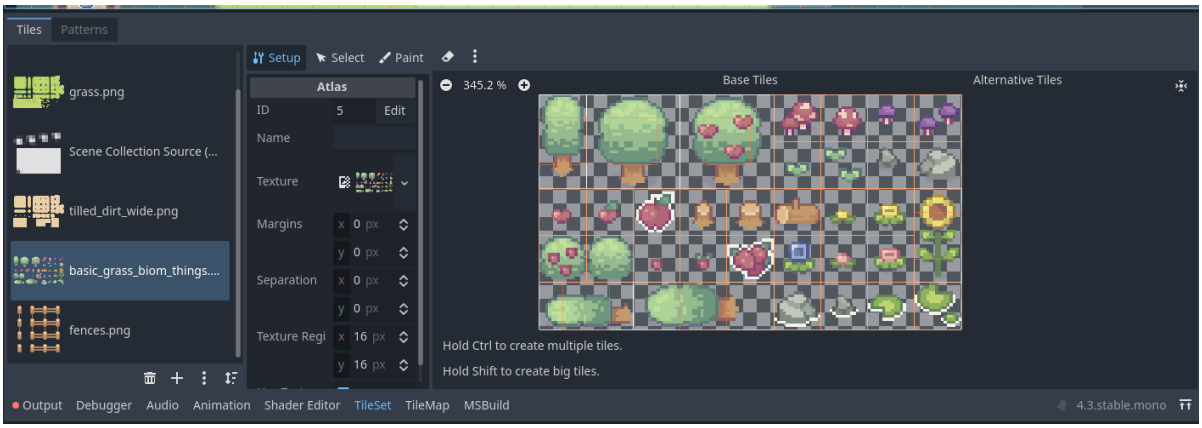

Рисунок 3.3 – Імпортовані срайти у середовищі Godot

Для побудови локацій гри необхідно також розробити тайли «землі», які слугуватимуть основою для більшості сцен. На рисунку 3.4 наведено готовий варіант тайлу, який буде використовуватись у проекті.

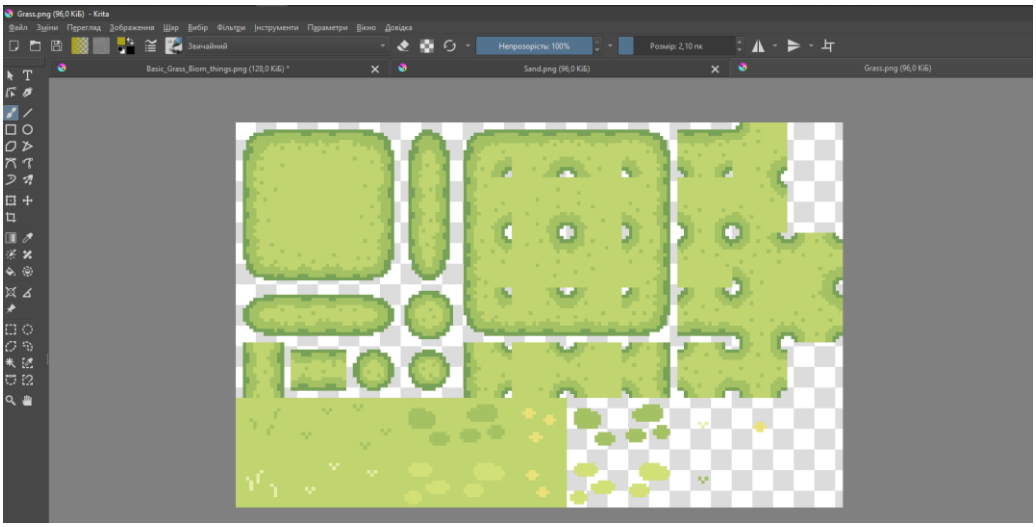

Рисунок 3.4 – Тайли «землі» спроектовані у Krita

Отже, маючи розроблені 2D-спрайти ігрових об'єктів та базові тайли «землі» імпортовані в Godot, можна переходити до наступного етапу, а саме безпосереднього створення ігрових сцен та проектування локацій.

3.2 Створення ігрових сцен для відеогри «Paw Isle» у Godot

Проектування та реалізація ігрових сцен є важливим етапом у розробці «Paw Isle», оскільки саме вони формують візуальний простір, в якому відбуватиметься ігровий процес та взаємодія гравця зі світом.

«Дальній пляж» є початковою точкою подорожі «Лапчика» на острів «Paw Isle». Ця сцена розроблена як затишне місце, що символізує початок нової пригоди. Вона включає в себе елементи прибережного ландшафту та перші ознаки дикої рослинності острова. Тут гравець вперше ступає на землю після прибуття на човні. На рисунку 3.5 зображено готову сцену «Дальнього пляжу».

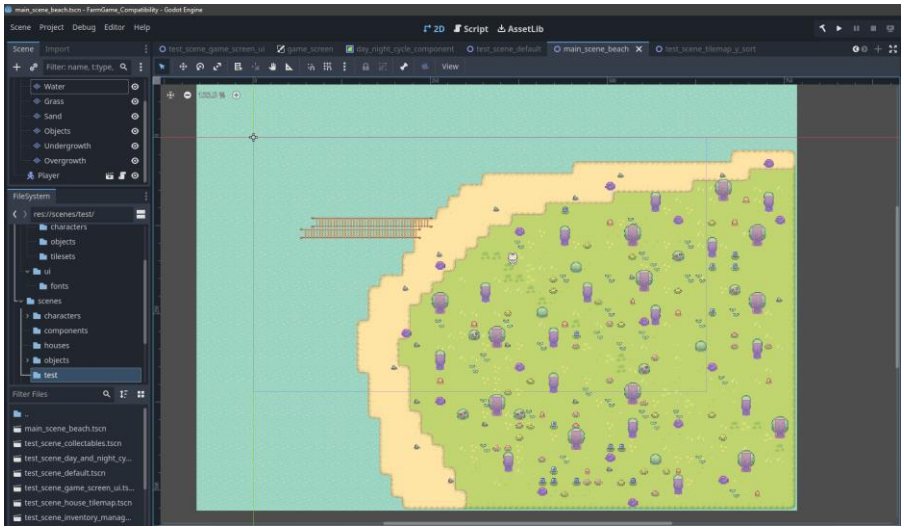

Рисунок 3.5 – Локація «Дальній пляж»

«Pawville» – це невелике поселення, що колись було серцем острова, а нині є майже покинутим місцем, де живуть лише віддані жителі, що пам'ятають його колишню славу. Ця сцена розроблена так, щоб передати відчуття занепаду, але й збереженої надії. Хоча тут залишилися лише декілька

унікальних будівель, таких як магазин Кицюні та занедбана таверна, кожен дім розповідає історію минулого процвітання. На рисунку 3.6 зображено сцену «Pawvill».

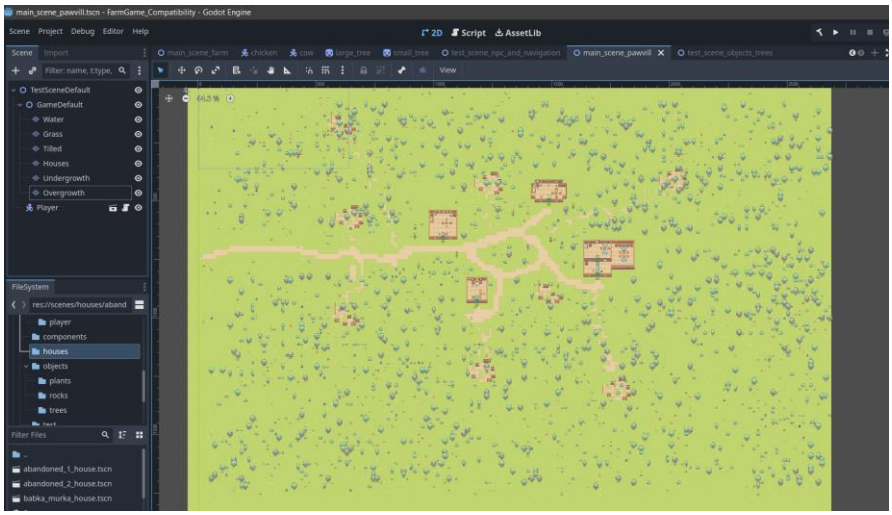

Рисунок 3.6 – Локація «Pawvill»

«Ферма Лапчика» – це головна локація у гри, яка є особистим володінням гравця та головним об'єктом для розвитку. Сцена відображає початковий занедбаний стан ферми. Вона включає руїни старих грядок, зарослу траву та скромний будиночок «Лапчика». Проектування цієї сцени передбачає багато вільного простору для майбутнього розширення, будівництва нових споруд та полів. На рисунку 3.7 зображено сцену «Ферма Лапчика».

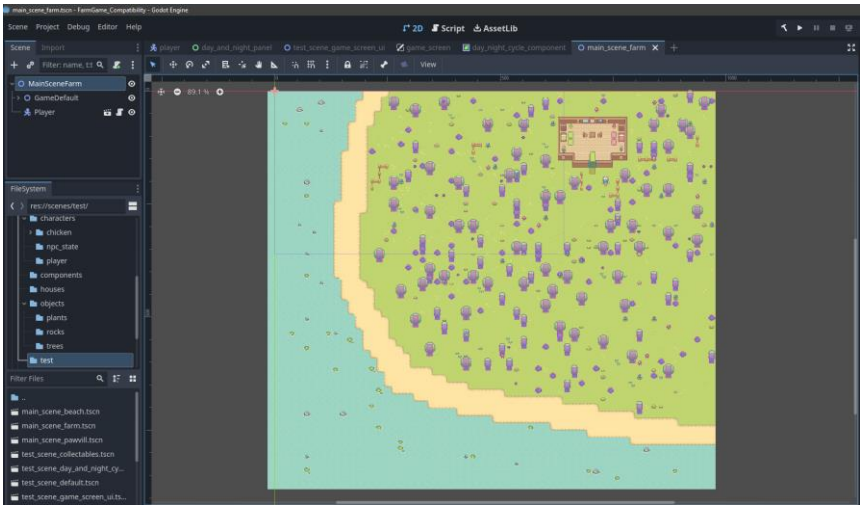

Рисунок 3.7 – Локація «Ферма Лапчика»

«Озеро Срібних Риб» є центральною та мальовничою локацією на острові, що ідеально підходить для риболовлі та збору водних ресурсів. Як видно на рисунку 3.8, сцена озера відображає велику, спокійну водну гладь, яка займає значну частину внутрішньої області острова. Воно оточене береговою лінією, яка органічно переходить у зелену траву та поодинокі дерева, створюючи відчуття замкнутого, але просторого простору.

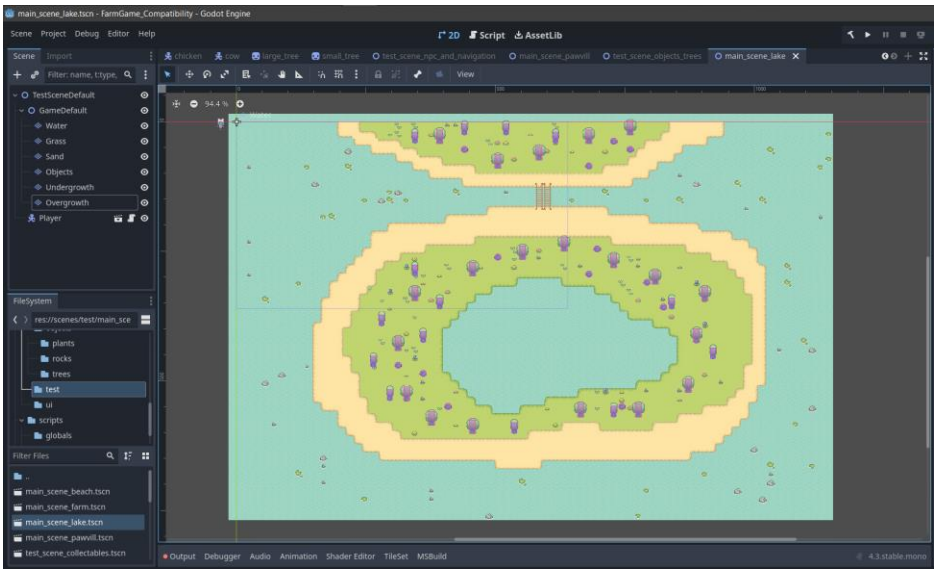

Рисунок 3.8 – Локація «Озеро Срібних Риб»

Подальші сцени були спроектовані у такому ж стилі. Кожна з унікальних локацій – від «Дальнього пляжу», що слугує точкою прибуття, до «Ферми Лапчика», поселення «Pawville», зон «Лісу Шепотів» та «Озера Срібних Риб», були продумані та створенні з врахуванням їх функціонального призначення та сюжетної ролі.

3.3 Написання механік гри з використанням мови програмування C#

У процесі розробки «Paw Isle» всі ключові ігрові механіки реалізовуються за допомогою мови програмування C# у середовищі Godot. Вибір C# дозволяє створювати надійний, розширюваний та продуктивний код.

Одиним із ключових механік є переміщення головного героя та його анімації, яка була реалізована за допомогою патерну «State Machine». Цей підхід дозволяє чітко керувати поведінкою персонажа, переходячи між різними станами, такі як: «Idle», «Walk», «Chopping», «Tilling», «Watering». Кожен стан відповідає за певний набір анімацій та логіки взаємодії. Наприклад, у стані «Walk» спрацьовує анімація ходьби, а у стані «Chopping» – анімація рубання сокирою. Такий підхід забезпечує плавність переходу між діями та гнучкість у додаванні нових анімацій або поведінок.

У лістингу 3.1 наведено фрагмент коду, який відслідковує натискання відповідних клавіш, що відповідають за ходьбу персонажа.

Лістинг 3.1 – Фрагмент скрипту «WalkState.cs»

```
public override void OnPhysicsProcess(double delta)
    {
        Vector2 direction = GameInputEvents.GetMovementInput();
        if(direction == Vector2.Up)
        {
            AnimatedSprite.Play("walk_back");
        }
        else if(direction == Vector2.Right)
        {
            AnimatedSprite.Play("walk_right");
        }
        else if(direction == Vector2.Down)
        {
            AnimatedSprite.Play("walk_front");
        }
        else if(direction == Vector2.Left)
        {
            AnimatedSprite.Play("walk_left");
        }

        if(direction != Vector2.Zero)
        {
            Player.PlayerDirection = direction;
        }
        Player.Velocity = direction * Speed;
        Player.MoveAndSlide();
    }
```

За схожим принципом було розроблено інші стани персонажа та підключенні до сцени з головним героєм.

Наступним етапом стала розробка механіки збору ресурсів, рубання дерев та розбивання каміння. Вони були реалізовані за допомогою взаємодії персонажа з об'єктами в світі, які мають відповідні налаштування для цього. При взаємодії з деревом, за наявності сокири, або каменем запускається відповідна анімація роботи. Після певної кількості ударів відбувається «руйнування» об'єкта, внаслідок чого, з нього випадають ресурси, а саме: деревина та камінь. Ці ресурси з'являються у вигляді невеликих спрайтів, які гравець може підібрати.

На рисунку 3.9 зображено результат на тестовій сцені даних механік.

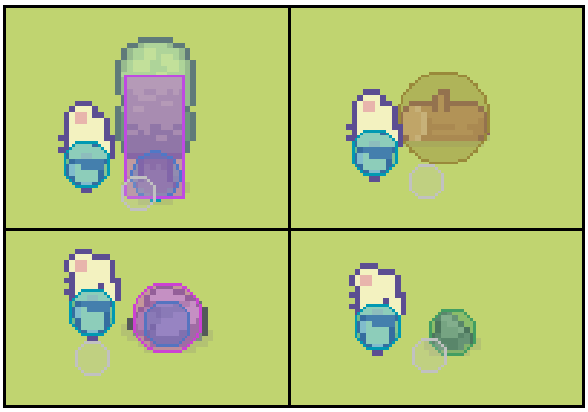

Рисунок 3.9 – Демонстарція розбиття та випадання певних ресурсів

Як можна побачити з рисунку, кожний елемент має свою колізію та області в якій він стає активним для взаємодії.

Наступна потірбна механіку у грі є підбирання ресурсів та було реалізовано за допомогою зон взаємодії (Area2D). Коли персонаж заходить у область дії випавшого ресурсу, відбувається його автоматичний або можна зробити за натисканням кнопки підбір, і ресурс додається до інвентарю гравця. Ця механіка є простою та інтуїтивно зрозумілою, дозволяючи гравцеві швидко збирати врожай та ресурси. На рисунку 3.10 наведено приклад підбирання предметів.

Рисунок 3.10 – Демонстрація підбору та зарахування ресурсів

Як можна побачити з рисунку, кожний елемент має свою колізію та області в якій він стає активним для взаємодії. Якщо персонаж зайде в дану зону, то об’єкт пропаде з сцени та добавиться до інвентару. У додатку А наведено всі написані скрипти для головного персонажа.

Також, одна із важливих механік – це логіка поведінки NPC-тварин, таких як кури та корови, які базуються на використанні зон навігації (NavigationRegion2D). Для початку потрібно створити сам об’єкт «Chicken» і провести його налашутвання, а саме: додати спрайт, анімацію спокою та ходьби, скрипи для реалізації хаотичного руху в вибраній зоні. На рисунку 3.11 зображено вже створений об’єкт.

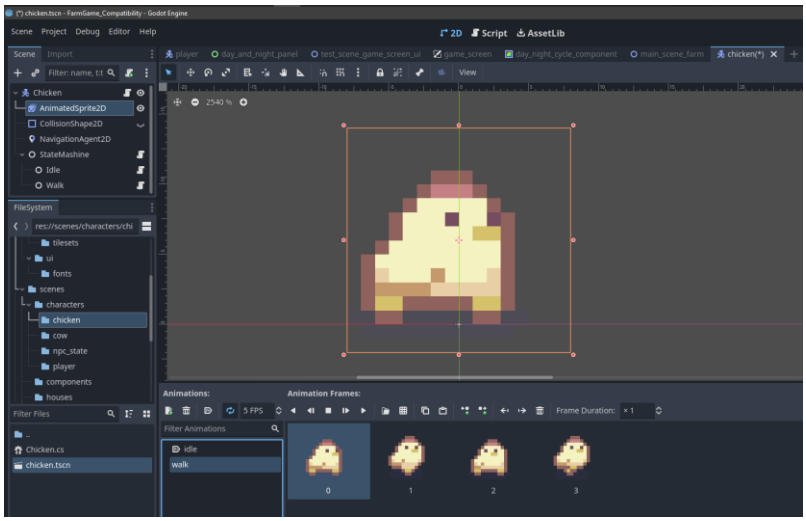

Рисунок 3.11 – Спроектована курка для відеогри «Paw Isle»

Аналогічним методом створив об’єкт «Cow», який можна побачити на рисунку 3.12.

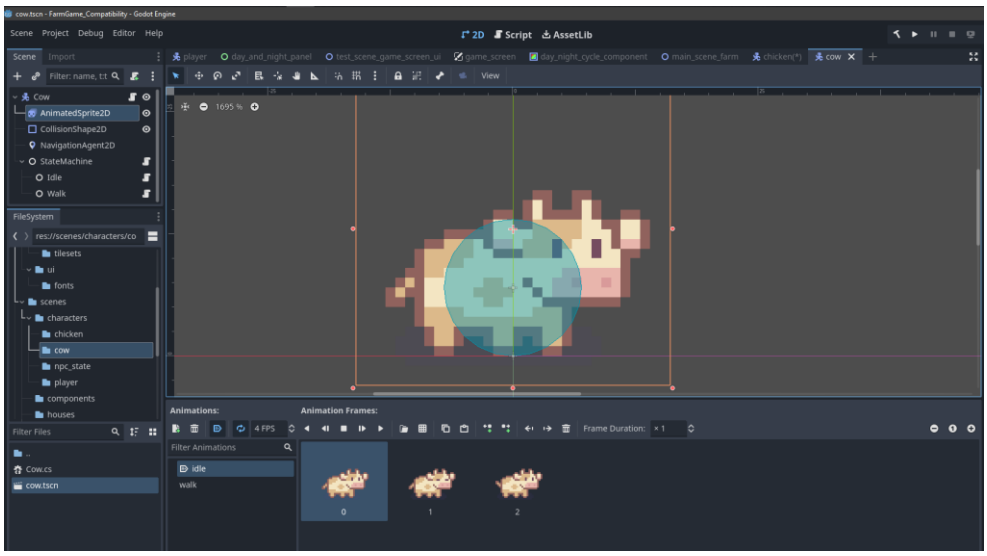

Рисунок 3.12 – Спроектована корова для відеогри «Paw Isle»

Для тестування було створено окрему сцену. На ній додав дві зони для курей та корів. Після чого на ній створив спеціальні загони, які відповідно мають вже промальовану колізію. За допомогою спеціального інструмента у рушії можна побачити територію, яку було вибрано. На рисунку 3.13 зображено вигляд території по якій будуть ходити тварини.

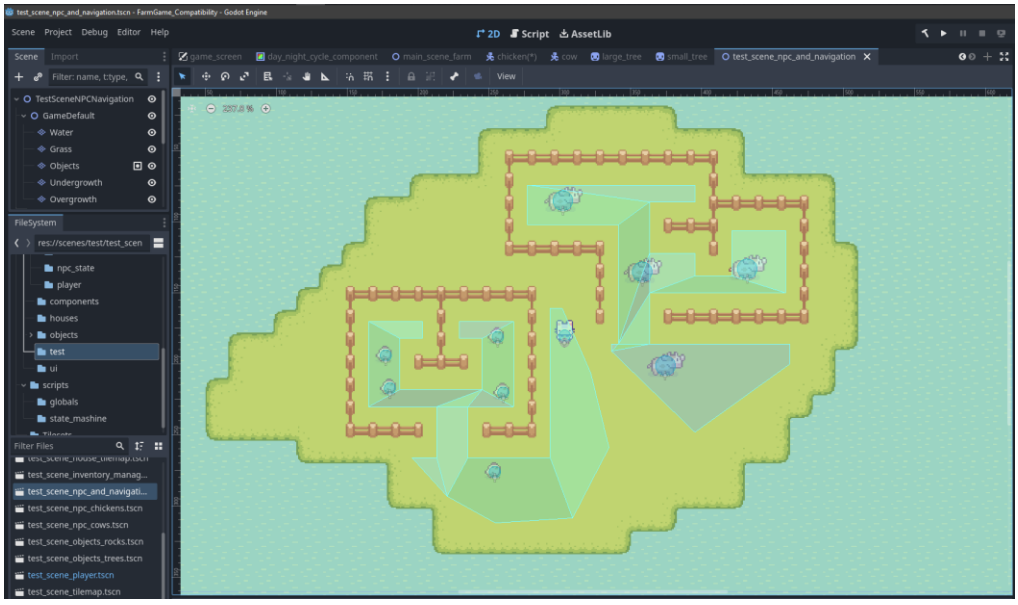

Рисунок 3.13 – Зони тварин

Тобто, кожна тварина знаходиться у своєму вузлі з обмеженою зоною, де випадковим чином для кожного об’єкту вибирається точка до якої буде він буде прямувати. У додатку Б наведено напсиані всі скрипти, які пов’язані з механікою переміщення тварин.

Зміна часу доби є однією з ключових систем, що впливає на ігровий процес та атмосферу. Ця механіка реалізована за допомогою програмного відстеження часу, яке змінюється поступово. На рисунку 3.14 наведено цикли зміни часу від світанку до повної ночі.

Рисунок 3.14 – Демонстарція зміни циклу дня

Тепер, маючи розроблений і робочий інтерфейс, злегкістю можна керувати часом нажимаючи на відповідні кнопки, які пришвидшують або, наборот, сповільнюють швидкість часу.

Робочий інструментарій та інвентар головного героя включає в себе основні інструменти для фермерства та збору ресурсів. Механіка перемикання інструментів реалізована таким чином, що гравець може швидко обирати потрібний інструмент зі свого інвентарю або за допомогою гарячих клавіш або наводячись мишкою на відповідну іконку. Кожен інструмент має унікальну функціональність і відповідну анімацію «Лапчика» при його використанні, що дозволяє виконувати різні типи взаємодій зі світом та об'єктами. На рисунку 3.15 зображено вигляд розробленого інтерфейсу.

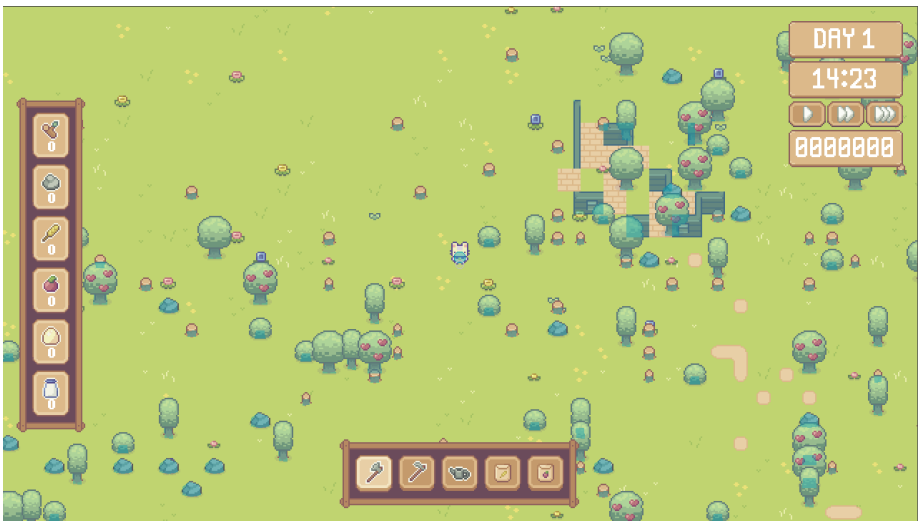

Рисунок 3.15 – Робочий інтерфейс персонажа у відеогрі «Paw Isle»

Даний інтерфейс було розроблено за відповідним, раніше спроектованим, макетом. Було дописано функціональну частину, за допомогою якого гравець може з легкістю переключатись між інструментами. У додатку В наведено глобальні скрипти, що використовуються для зберігання даних.

3.4 Висновок до третього розділу

У третьому розділі кваліфікаційної роботи було розглянуто процес реалізації відеогри «Paw Isle» на рушії Godot з використанням мови програмування C#. Розробка охоплювала всі ключові аспекти: від створення візуальної складової, включно з розробкою спрайтів, тайлів і сцен, до проектування ігрових локацій. Значну увагу приділено художньому стилю гри, який через піксельну графіку передає затишну та чарівну атмосферу симулятора фермерства.

Крім візуальної частини, в розділі було описано реалізацію основних геймплейних механік, а саме: переміщення персонажа, взаємодію з об’єктами, збирання ресурсів, поведінку NPC-тварин, зміну часу доби та систему інвентарю з перемиканням інструментів. Усі ці компоненти формують цілісну й інтерактивну ігрову систему, яка відповідає сучасним вимога.

РОЗДІЛ 4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Актуальність безпеки життєдіяльності людини в цифровому світі

У сучасному світі, де цифрові технології є основою багатьох професій, актуальність безпеки життєдіяльності стрімко зростає. Це особливо помітно серед фахівців, чия робота передбачає тривалу та інтенсивну взаємодію з комп'ютерною технікою, як-от розробники відеоігор, що працюють на рушії Godot. Така інтенсивна робота часто призводить до розвитку втоми, яка може мати серйозні наслідки для здоров'я.

Українське законодавство чітко регламентує ці аспекти. Згідно з ДСанПіН 3.3.2.007-98 «Гігієнічні вимоги до організації роботи з візуальними дисплейними терміналами (ВДТ)» [31], ігнорування встановлених норм щодо положення користувача, режимів праці та параметрів робочого середовища може спричинити значні негативні наслідки. Серед найпоширеніших проблем буває порушення зору, що виникають через постійну напругу очей та недостатнє моргання. Також значно зростає ризик розвитку захворювань опорно-рухового апарату, таких як: остеохондроз або синдром карпального тунелю, через неправильну позу та повторювані рухи. Крім того, тривала робота в умовах стресу без належного відпочинку може призвести до психоемоційних розладів та емоційного вигорання.

Зазначені норми вимагають чіткого врахування оптимальних режимів праці та відпочинку. Це включає обов'язкові 5–10-хвилинні перерви щогодини роботи за комп'ютером. Ці короткі паузи є не просто перервою від роботи, а можливістю для очей сфокусуватися на далеких об'єктах, а для м'язів розслабитись або зробити легкі вправи. Додатково, необхідно забезпечувати регулювання освітлення на робочому місці, уникаючи відблисків на екрані та забезпечуючи достатній рівень яскравості. Рівень шуму має бути мінімізований, а мікроклімат – оптимальним для підтримки комфорту та концентрації.

Важливим аспектом є і дотримання ергономічних вимог до робочого місця, що охоплює правильне розташування монітор.

Забезпечення безпеки життєдіяльності в цифровій ері є не лише обов'язковим елементом відповідності чинному законодавству, а й головною умовою для підтримки здоров'я фахівців. Це безпосередньо впливає на ефективність та якість процесу розробки, оскільки здоровий та відпочилий розробник є більш продуктивним та креативним.

4.2 Організація безпечної роботи електроустановок

Безпечна експлуатація електроустановок є абсолютною вимогою для кожного робочого місця, оснащеного персональними комп'ютерами та іншим електрообладнанням. Це життєво важливо для запобігання нещасним випадкам, таким як ураження електричним струмом, а також для уникнення пожеж та пошкодження дорогоцінного обладнання. В Україні ключові вимоги та процедури у цій сфері регулюються Наказом Міністерства праці України від 09.01.1998 № 4 «Про затвердження Правил безпечної експлуатації електроустановок споживачів» (ДНАОП 0.00-1.21-98). Доповнюють ці положення Правила улаштування електроустановок (ПУЕ), офіційне видання 2017 року, затверджені Наказом Міненерговугілля України від 21.07.2017 № 476. Цей фундаментальний документ, розроблений Відокремленим підрозділом «Науково-проектний центр розвитку Об'єднаної енергетичної системи України» ДП «НЕК «Укренерго», слугує заміною застарілих Правил устройства электроустановок 1986 року (за винятком глав 7.5 та 7.7), адаптуючи українські стандарти до сучасних технологій та міжнародних вимог електробезпеки [32].

Згідно з цими правилами, заземлення всіх комп'ютерів та пов'язаного з ними електрообладнання є обов'язковим. Це критичний захід, що забезпечує надійний відвід електричного струму у випадку пошкодження ізоляції, тим самим запобігаючи ризику ураження електричним струмом [33]. Система заземлення повинна включати використання спеціальних захисних пристроїв,

таких як: пристрої захисного відключення (ПЗВ) та автоматичні вимикачі, які оперативно реагують на витік струму, миттєво знеструмлюючи пошкоджену ділянку. Крім того, всі заземлення повинні мати чітку перенумерацію для легшої ідентифікації та контролю.

Категорично забороняється використання пошкоджених дротів, кабелів чи подовжувачів з будь-якими ознаками порушення ізоляції. Такі елементи є прямою загрозою виникнення коротких замикань та пожеж. Усі подовжувачі та розгалужувачі, що використовуються для підключення комп'ютерної техніки, повинні бути обладнані захистом від перенапруги (мережевими фільтрами-подовжувачами). Це захищає дороге обладнання від різких стрибків напруги в електромережі, які можуть призвести до його виходу з ладу.

Розміщення електронних пристроїв також має свої вимоги. Їх не слід встановлювати поблизу джерел значного тепла, таких як: радіатори опалення або прямі сонячні промені, оскільки це може призвести до їх перегріву та збою. Важливо також забезпечити захист обладнання від надмірного пилу та вологи, які можуть спричинити корозію контактів та внутрішні короткі замикання. Регулярне очищення вентиляційних отворів та внутрішніх компонентів комп'ютера від пилу є важливою профілактичною мірою.

Увесь персонал, який працює з електроустановками, включаючи комп'ютерну техніку, зобов'язаний проходити періодичні перевірки знань та відповідні інструктажі з електробезпеки. Це повинно відбуватися згідно з вимогами нормативно-правових актів з охорони праці (НПАОП) та правил технічної експлуатації електроустановок споживачів (ПТЕЕС). Такі інструктажі гарантують, що працівники усвідомлюють потенційні ризики та знають, як діяти в нештатних ситуаціях.

У разі виникнення будь-яких ознак аварійної ситуації, таких як: запах горілого, іскріння, нехарактерний шум, або у випадку короткого замикання чи пожежі, необхідно негайно відключити всі електропристрої від мережі. Після цього слід забезпечити безпечну евакуацію персоналу з небезпечної зони та

негайно повідомити відповідальних осіб – керівництво, інженера з охорони праці або пожежну службу.

4.3 Інженерно-технічне рішення з охорони праці розробника

Для всебічної оптимізації умов праці розробника, впровадження сучасних інженерно-технічних рішень з охорони праці є не просто бажаним, а необхідним заходом. Ці рішення не тільки допомагають відповідати нормативним вимогам, а й суттєво підвищують продуктивність, знижуючи потенційні ризики для здоров'я та цілісності даних.

Одним з ключових елементів є використання джерел безперебійного живлення (UPS) [34]. UPS не лише захищає комп'ютер та інше цінне обладнання від стрибків напруги, що можуть призвести до їх пошкодження, але й гарантує безпечне завершення робочих сесій у разі несподіваного відключення електроенергії. Це критично важливо для запобігання втраті даних та забезпечення безперервності процесу розробки.

Важливим аспектом є також автоматизація оновлень програмного забезпечення та створення зовнішніх резервних копій. Регулярні оновлення системи та програм дозволяють усувати вразливості та підвищувати стабільність роботи. Водночас, автоматичні бекапи значно підвищують рівень збереження даних проекту, захищаючи їх від випадкових видалень, збоїв обладнання чи кібератак. Цей підхід мінімізує рутинні операції, дозволяючи розробнику зосередитися на основних завданнях.

Комфорт та безпека роботи з периферійними пристроями значно покращується завдяки застосуванню ергономічних аксесуарів. Прикладами таких аксесуарів є антивібраційні килимки під клавіатуру та мишу, які зменшують передачу вібрації на руки, а також USB-хаби із вбудованим захистом від короткого замикання та перенапруги, що забезпечують безпечне підключення зовнішніх пристроїв.

Для підтримки оптимального робочого середовища важливе значення мають системи вентиляції та кондиціонування [35]. Вони забезпечують стабільний мікроклімат у приміщенні, підтримуючи комфортну температуру та вологість, що запобігає перегріву та духоті. Це безпосередньо впливає на самопочуття розробника, знижуючи втому та підвищуючи його концентрацію. Додатково, впровадження автоматичних моніторів шуму та температури дозволяє здійснювати постійний контроль за відповідністю цих параметрів санітарним нормам, забезпечуючи своєчасне реагування на будь-які відхилення.

Сукупно ці інженерно-технічні заходи мають на меті не лише зниження ризиків, пов'язаних з експлуатацією обладнання, а й створення максимально комфортних та сприятливих умов праці. Такий підхід позитивно впливає на концентрацію та загальну продуктивність розробника, сприяючи ефективній та якісній реалізації проектів.

4.4 Висновок до четвертого розділу

У четвертому розділі кваліфікаційної роботи детально описано комплекс заходів із забезпечення безпеки життєдіяльності та охорони праці, враховуючи специфіку роботи розробника в сучасному цифровому середовищі. Особлива увага приділена процесу створення відеоігор на рушії Godot, що передбачає тривалу та інтенсивну роботу за комп'ютером.

Безпеки та охорони праці є запорукою успішної розробки. Цей підхід охоплює всі етапи: від детального аналізу системи до реалізації конкретних фізичних засобів захисту та неухильного дотримання нормативної бази. Такий підхід не лише забезпечує відповідність законодавству, але й прямо впливає на підвищення продуктивності, якості виконуваних робіт та збереження здоров'я розробника.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи освітнього рівня «Бакалавр» було розроблено 2D відеогру в жанрі симулятора фермерства на базі рушія Godot з використанням мови програмування C#

В першому розділі кваліфікаційної роботи освітнього рівня «Бакалавр»:

- Проаналізовано сучасний стан GameDev-індустрії, існуючі жанри відеоігор та особливості їхнього розвитку.
- Досліджено популярні ігрові рушії.
- Оглянуто існуючі рішення та успішні проекти у жанрі симуляторів фермерства, визначено їхні ключові особливості та механіки.

В другому розділі кваліфікаційної роботи:

- Розроблено детальну концепцію гри, включаючи сюжет, основні ігрові механіки, візуальний стиль.

В третьому розділі кваліфікаційної роботи:

- Спроектовано архітектуру гри та структуру сцен.
- Реалізовано візуальну складову гри, включаючи створення 2D-спрайтів, тайлових карт, анімацій персонажа та об'єктів, а також налаштування камери, освітлення та пост-обробки.
- Розроблено ігрову логіку та взаємодію гравця з об'єктами світу.
- Створено користувацький інтерфейс, включаючи головне меню, екран завантаження та вікно налаштувань.

У розділі «Безпека життєдіяльності, основи охорони праці» розглянуто ключові аспекти захисту здоров’я розробника програмного забезпечення в умовах цифрового середовища. Акцент зроблено на вплив тривалої роботи за комп’ютером на організм, зокрема на зір, опорно-руховий апарат та психоемоційний стан.

ПЕРЕЛІК ДЖЕРЕЛ

1 Game Developer Market Size, Share, Growth, and Industry Analysis, By Types (C++, Java, Others), Applications (PC Games, Mobile Games, TV Games, Other) and Regional Insights and Forecast to 2033. Global Growth Insights. [Електронний ресурс]. Режим доступу: https://www.globalgrowthinsights.com/market-reports/game-developer-market-109099 (дата звернення: 01.06.2025).

2 Results and Trends of the Gaming Market in 2024. LogrusIT. [Електронний ресурс]. Режим доступу: https://games.logrusit.com/en/news/game-industry-trends/ (дата звернення: 01.06.2025).

3 Gaming Global Market Report 2025. The Business Research Company. [Електронний ресурс]. Режим доступу: https://www.thebusinessresearchcompany.com/report/gaming-global-market-report (дата звернення: 01.06.2025).

4 Gaming Console Market Size to grow USD. *GlobeNewswire*. [Електронний ресурс]. Режим доступу: https://www.globenewswire.com/news-release/2025/01/31/3018850/0/en/Gaming-Console-Market-Size-to-grow-USD-55-36-Billion-by-2032-at-a-CAGR-of-8-46-Research-by-SNS-Insider.html (дата звернення: 01.06.2025).

5 Indie Game Market Size & Share Analysis - Growth Trends & Forecasts (2025 - 2030). MordorIntelligence. [Електронний ресурс]. Режим доступу: https://www.mordorintelligence.com/industry-reports/indie-game-market (дата звернення: 01.06.2025).

6 Growth Trends & Risks Within the Gaming Industry. Forvis Mazars. [Електронний ресурс]. Режим доступу: https://www.forvismazars.us/forsights/2025/06/growth-trends-risks-within-the-gaming-industry-q1-2025 (дата звернення: 01.06.2025).

7 Godot Engine vs Unity: Which One Suits You Best in 2025. Rocket Brush Studio. [Електронний ресурс]. Режим доступу: https://rocketbrush.com/blog/godot-vs-unity (дата звернення: 01.06.2025).

8 Godot проти Unreal: Який двигун обрати для розробників ігор?. Meshy. [Електронний ресурс]. Режим доступу: https://www.meshy.ai/blog/godot-vs-unreal (дата звернення: 01.06.2025).

9 Unreal Engine 2024 Subscription Pricing Announced. GameFromScratch.com. [Електронний ресурс]. Режим доступу: https://gamefromscratch.com/unreal-engine-2024-subscription-pricing-announced/ (дата звернення: 01.06.2025).

10 Frequently asked questions. UnrealEngine. [Електронний ресурс]. Режим доступу: https://www.unrealengine.com/en-US/faq (дата звернення: 01.06.2025).

11 Is Unreal Engine Development Good for 2D Games?. Chic Mic Studio. [Електронний ресурс]. Режим доступу: https://www.chicmicstudios.in/blogs/is-unreal-engine-development-good-for-2d-games/ (дата звернення: 01.06.2025).

12 Why Godot is right for you. Godot. [Електронний ресурс]. Режим доступу: https://godotengine.org/features/ (дата звернення: 02.06.2025).

13 Godot C# packages move to .NET 8. Godot. [Електронний ресурс]. Режим доступу: : https://godotengine.org/article/godotsharp-packages-net8/ (дата звернення: 02.06.2025).

14 10 Reasons Why People Love Farming Simulator Games. GameSpace. [Електронний ресурс]. Режим доступу: https://gamespace.com/all-articles/news/10-reasons-why-people-love-farming-simulator-games/ (дата звернення: 02.06.2025).

15 The psychology of why video game farming is so satisfying. Popular Science. [Електронний ресурс]. Режим доступу: https://www.popsci.com/health/stardew-valley-psychology-farming/ (дата звернення: 02.06.2025).

16 A Brief History of the Harvest Moon Franchise. Cozy Game Reviews. [Електронний ресурс]. Режим доступу: https://cozygamereviews.com/history-of-harvest-moon/ (дата звернення: 02.06.2025).

17 Produce. Nookipedia. [Електронний ресурс]. Режим доступу: https://nookipedia.com/wiki/Produce (дата звернення: 02.06.2025).

18 Ostrovska, H., Strutynska, I., Sherstiuk, R., Petukhova, O., & Yasinetska, I. (2023). Development of collective intelligence in the enterprises' digital transformation (дата звернення: 03.06.2025).

19 Harchenko, A., Bodnarchuk, I., & Halay, I. (2013). Decision support system of software architect. 2013 IEEE 7th International Conference on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS), 1, 265–269 (дата звернення: 03.06.2025).

20 Kharchenko, A., Bodnarchuk, I., & Yatcyshyn, V. (2014). The method for comparative evaluation of software architecture with accounting of trade-offs. American Journal of Information Systems, 2(1), 20–25 (дата звернення: 03.06.2025).

21 Harchenko, A., Bodnarchuk, I., & Yatcyshyn, V. (2012). The modeling and optimization of software engineering processes. Proceedings of International Conference on Modern Problem of Radio Engineering, Telecommunications and Computer Science, 326 (дата звернення: 03.06.2025).

22 Kharchenko, A., Halay, I., Zagorodna, N., & Bodnarchuk, I. (2015). Trade-off optimal decision of the problem of software system architecture choice. 2015 Xth International Scientific and Technical Conference" Computer Sciences and Information Technologies"(CSIT), 198–205 (дата звернення: 03.06.2025).

23 Харченко, О., Яцишин, В., & Боднарчук, І. (2013). Експертна система проектування архітектури програмного забезпечення. Комп'ютерні Технології Друкарства, (29), 10–26 (дата звернення: 03.06.2025).

24 Harchenko, A., Halay, I., & Bodnarchuk, I. (2012). Stability of the solutions of the optimization problem of software systems architecture. Computer

Science and Information Technologies, VIIth International and Science Conference CSIT 2012, 47–48 (дата звернення: 04.06.2025).

25 Kharchenko, A., Halay, I., & Bodnarchuk, I. (2016). Multicriteria architecture choice of software system under design and reengineering. 2016 XIth International Scientific and Technical Conference Computer Sciences and Information Technologies (CSIT), 4–8 (дата звернення: 04.06.2025).

26 An Optimal Trade-off Solution of the Software Architecture Choice Problem. (2016). Journal of Information and Computing Science, 11(4), 281–290 (дата звернення: 04.06.2025).

27 Kharchenko, O., Raichev, I., Bodnarchuk, I., & Zagorodna, N. (2018). Optimization of software architecture selection for the system under design and reengineering. 2018 14th International Conference on Advanced Trends in Radioelecrtronics, Telecommunications and Computer Engineering (TCSET), 1245–1248 (дата звернення: 04.06.2025).

28 Харченко, О. Г., Боднарчук, І. О., & Галай, І. О. (2012). Метод багатокритеріальної оптимізації програмної архітектури на основі аналізу компромісів. Інженерія Програмного Забезпечення, (3–4), 5–11 (дата звернення: 04.06.2025).

29 Шимчук, Г. В., Маєвський, О. В., & Назаревич, О. Б. (2016). Конспект лекцій з дисципліни Комп’ютерна графіка для студентів освітнього рівня «бакалавр» спеціальності 125 «Кібербезпека» (дата звернення: 04.06.2025).

30 Марценко, С. В. (2011). Математичне моделювання та статистичні методи обробки даних вимірювань в задачах моніторингу електронавантаження (дата звернення: 04.06.2025).

31 ДЕРЖАВНИЙ КОМІТЕТ УКРАЇНИ З НАГЛЯДУ ЗА ОХОРОНОЮ ПРАЦІ Н А К А З 16.03.2004 N 81. Верховна Рада України. [Електронний ресурс]. Режим доступу: https://zakon.rada.gov.ua/laws/show/z0620-04#Text (дата звернення: 06.06.2025).

32 ДЕРЖАВНИЙ КОМІТЕТ УКРАЇНИ З НАГЛЯДУ ЗА ОХОРОНОЮ ПРАЦІ Н А К А З 16.03.2004 N 82. Верховна Рада України. [Електронний

ресурс]. Режим доступу: https://zakon.rada.gov.ua/laws/show/z062004#Text (дата звернення: 06.06.2025).

33 Правила Улаштування Електроустановок. МІНЕНЕРГОВУГІЛЛЯ УКРАЇНИ. [Електронний ресурс]. Режим доступу: https://zakon.isu.net.ua/sites/default/files/normdocs/pue.pdf (дата звернення: 07.06.2025).

34 Як працюють джерела безперебійного живлення?. Карат ЛТД. [Електронний ресурс]. Режим доступу: https://karatltd.com.ua/iak-pratsiuiut-dzherela-bezperebiinoho-zhyvlennia (дата звернення: 07.06.2025).

35 Вентиляції та кондиціювання повітря у робочій зоні виробничих приміщень. Державна служба України з питань праці. [Електронний ресурс]. Режим доступу: https://dsp.gov.ua/ventyliatsii-ta-kondytsiiuvannia-povitria-u-robochii-zoni-vyrobnychykh-prymishchen/ (дата звернення: 07.06.2025).

ДОДАТКИ

Додаток А

Лістинги для реалізації ходьби персонажа

Лістинг А.1 – Код файлу «NodeStateMachine.cs»

```
using Godot;
using System;
using System.Collections.Generic;

public partial class NodeStateMachine : Node
{
    [Export]
    public NodeState InitialNodeState { get; set; }

    private Dictionary<string, NodeState> _nodeStates = new
Dictionary<string, NodeState>();

    private NodeState _currentNodeState;
    private string _currentNodeStateName;
    private string parentNodeName;

    public override void _Ready()
    {
     parentNodeName = GetParent().Name;

     foreach (Node child in GetChildren())
     {
          if (child is NodeState nodeState)
          {
               _nodeStates[nodeState.Name.ToString().ToLower()] =
nodeState;
               nodeState.Transition += TransitionTo;
          }
     }

     if (InitialNodeState != null)
     {
          ChangeState(InitialNodeState);
     }
     else
     {
          GD.PushError("InitialNodeState не встановлено для
StateMachine!");
     }
    }

    public override void _Process(double delta)
    {
     _currentNodeState?.OnProcess(delta);
    }
```

```
        public override void _PhysicsProcess(double delta)
        {
         if (_currentNodeState != null)
         {
              _currentNodeState.OnPhysicsProcess(delta);
              _currentNodeState.OnNextTransitions();
              //GD.Print($"{parentNodeName} - current state:
{_currentNodeStateName}");
         }
        }

        public void ChangeState(NodeState newState)
        {
         if (newState == _currentNodeState)
         {
              return;
         }

         if (_currentNodeState != null)
         {
              _currentNodeState.OnExit();
              _currentNodeState.Transition -= TransitionTo;
         }

         _currentNodeState = newState;

         if (_currentNodeState != null)
         {
              _currentNodeState.Transition += TransitionTo;
              _currentNodeState.OnEnter();
              _currentNodeStateName =
_currentNodeState.Name.ToString().ToLower();
              //GD.Print($"Поточний стан: {_currentNodeStateName}");
         }
        }

        private void TransitionTo(string nodeStateName)
        {
         if (_currentNodeState != null && nodeStateName.ToLower() ==
_currentNodeState.Name.ToString().ToLower())
         {
              return;
         }

         if (_nodeStates.TryGetValue(nodeStateName.ToLower(), out
NodeState new_node_state))
         {
              ChangeState(new_node_state);
         }
         else
         {
              GD.PrintErr($"NodeStateMachine: Не знайдено стану з
назвою: {nodeStateName}");
```

```
        }
    }
}
```

Лістинг А.2 – Код файлу «Player.cs»

```
using Godot;
using System;

public partial class Player : CharacterBody2D
{
    public Vector2 PlayerDirection {get; set;}

    [Export]
    public Tools CurrentTool {get; set; } = Tools.None;

    private HitComponent hitComponent;
    private ToolManager toolManager;

    public override void _Ready()
    {
        hitComponent =
GetNode<HitComponent>("HitComponent");
        toolManager =
GetNode<ToolManager>("/root/ToolManager");

        toolManager.ToolSelected += OnToolSelected;
    }

    public void OnToolSelected(Tools tool)
    {
        CurrentTool = tool;
        hitComponent.currentTool = tool;
        GD.Print($"Tool: {tool}");
    }
}
```

Лістинг А.3 – Код файлу «IdleState.cs»

```
using Godot;
using System;

public partial class IdleState : NodeState
{
[Export]
public Player Player {get; set; }
[Export]
public AnimatedSprite2D AnimatedSprite {get; set; }

public override void _Ready()
{
```

```
    }

    public override void OnPhysicsProcess(double delta)
    {
    if(Player.PlayerDirection == Vector2.Up)
    {
    AnimatedSprite.Play("idle_back");
    }
    else if(Player.PlayerDirection == Vector2.Right)
    {
    AnimatedSprite.Play("idle_right");
    }
    else if(Player.PlayerDirection == Vector2.Down)
    {
    AnimatedSprite.Play("idle_front");
    }
    else if(Player.PlayerDirection == Vector2.Left)
    {
    AnimatedSprite.Play("idle_left");
    }
    else
    {
    AnimatedSprite.Play("idle_front");
    }
    }

    public override void OnNextTransitions()
    {
    GameInputEvents.GetMovementInput();

    if(GameInputEvents.IsMovementInput())
    {
    EmitSignal(SignalName.Transition, "Walk");
    }

    if(Player.CurrentTool == Tools.AxeWood &&
GameInputEvents.UseTool())
    {
    EmitSignal(SignalName.Transition, "Chopping");
    }

    if(Player.CurrentTool == Tools.TillGround &&
GameInputEvents.UseTool())
    {
    EmitSignal(SignalName.Transition, "Tilling");
    }

    if(Player.CurrentTool == Tools.WaterCrops &&
GameInputEvents.UseTool())
    {
    EmitSignal(SignalName.Transition, "Watering");
    }
    }
```

```
	public override void OnExit()
	{
	AnimatedSprite.Stop();
	}
	}
```

Лістинг А.4 – Код файлу «ChoppingState.cs»

```
	using Godot;
	using System;

	public partial class ChoppingState : NodeState
	{
		[Export]
		public Player Player {get; set; }
		[Export]
		public AnimatedSprite2D AnimatedSprite {get; set; }
		[Export]
		public CollisionShape2D hitComponentCollisionShape {get;
set; }

		public override void _Ready()
		{
			hitComponentCollisionShape.Disabled = true;
			hitComponentCollisionShape.Position = new Vector2(0,
0);
		}

		public override void _Process(double delta)
		{

		}

		public override void OnProcess(double delta)
		{

		}

		public override void OnPhysicsProcess(double delta)
		{

		}

		public override void OnNextTransitions()
		{
			if(!AnimatedSprite.IsPlaying())
			{
				EmitSignal(SignalName.Transition, "Idle");
			}
		}
```

```
        public override void OnEnter()
        {
            if(Player.PlayerDirection == Vector2.Up)
            {
                AnimatedSprite.Play("chopping_back");
                hitComponentCollisionShape.Position = new
Vector2(0, -18);
            }
            else if(Player.PlayerDirection == Vector2.Right)
            {
                AnimatedSprite.Play("chopping_right");
                hitComponentCollisionShape.Position = new
Vector2(9, 0);
            }
            else if(Player.PlayerDirection == Vector2.Down)
            {
                AnimatedSprite.Play("chopping_front");
                hitComponentCollisionShape.Position = new
Vector2(0, 3);
            }
            else if(Player.PlayerDirection == Vector2.Left)
            {
                AnimatedSprite.Play("chopping_left");
                hitComponentCollisionShape.Position = new
Vector2(-9, 0);
            }
            else
            {
                AnimatedSprite.Play("chopping_front");
                hitComponentCollisionShape.Position = new
Vector2(0, 3);
            }

            hitComponentCollisionShape.Disabled = false;
        }

        public override void OnExit()
        {
            AnimatedSprite.Stop();
            hitComponentCollisionShape.Disabled = true;
        }
    }
```

Лістинг А.5 – Код файлу «WalkState.cs»

```
using Godot;
using System;

public partial class WalkState : NodeState
{
    [Export]
    public Player Player {get; set; }
```

```
        [Export]
        public AnimatedSprite2D AnimatedSprite {get; set; }
        [Export]
        public int Speed = 50;

        public override void OnPhysicsProcess(double delta)
        {
            Vector2 direction =
GameInputEvents.GetMovementInput();

            if(direction == Vector2.Up)
            {
                AnimatedSprite.Play("walk_back");
            }
            else if(direction == Vector2.Right)
            {
                AnimatedSprite.Play("walk_right");
            }
            else if(direction == Vector2.Down)
            {
                AnimatedSprite.Play("walk_front");
            }
            else if(direction == Vector2.Left)
            {
                AnimatedSprite.Play("walk_left");
            }

            if(direction != Vector2.Zero)
            {
                Player.PlayerDirection = direction;
            }

            Player.Velocity = direction * Speed;
            Player.MoveAndSlide();
        }

        public override void OnNextTransitions()
        {
            if(!GameInputEvents.IsMovementInput())
            {
                EmitSignal(SignalName.Transition, "Idle");
            }
        }

        public override void OnExit()
        {
            AnimatedSprite.Stop();
        }
    }
```

Додаток Б

Лістинги для NPC-тварин

Лістинг Б.1 – Код файлу «IdleStateNPC.cs»

```
using Godot;
using System;

public partial class IdleStateNPC : NodeState
{
    [Export]
    public CharacterBody2D character {get; set;}
    [Export]
    public AnimatedSprite2D animatedSprite {get; set;}
    [Export]
    public float idleStateTimeInterval {get; set;} = 5.0f;

    private Timer idleStateTimer;
    private bool idleStateTimeout = false;

    public override void _Ready()
    {
     idleStateTimer = new Timer();
     idleStateTimer.WaitTime = idleStateTimeInterval;

     idleStateTimer.Timeout += OnIdleStateTimeout;

     AddChild(idleStateTimer);
    }

    public override void _Process(double delta)
    {

    }

    public override void OnProcess(double delta)
    {

    }

    public override void OnPhysicsProcess(double delta)
    {

    }

    public override void OnNextTransitions()
    {
     if(idleStateTimeout)
     {
          EmitSignal(SignalName.Transition, "walk");
     }
```

```
    }

    public override void OnEnter()
    {
     animatedSprite.Play("idle");

     idleStateTimeout = false;
     idleStateTimer.Start();
    }

    public override void OnExit()
    {
     animatedSprite.Stop();
     idleStateTimer.Stop();
    }

    private void OnIdleStateTimeout()
    {
     idleStateTimeout = true;
    }
}
```

Лістинг Б.2 – Код файлу «WalkStateNPC.cs»

```
using Godot;
using System;

public partial class WalkStateNPC : NodeState
{
    [Export]
    public NonPlayableCharacter character {get; set;}
    [Export]
    public AnimatedSprite2D animatedSprite {get; set;}
    [Export]
    public NavigationAgent2D navigationAgent {get; set;}
    [Export]
    public float minSpeed {get; set;} = 5.0f;
    [Export]
    public float maxSpeed {get; set;} = 10.0f;

    private float speed;

    public override void _Ready()
    {
     navigationAgent.VelocityComputed += OnSafeVelocityComputed;

     CallDeferred("CharacterSetup");
    }

    private async void CharacterSetup()
    {
     await ToSignal(GetTree(), SceneTree.SignalName.PhysicsFrame);
```

```
          SetMovementTarget();
         }

         private void SetMovementTarget()
         {
          Vector2 targetPosition =
NavigationServer2D.MapGetRandomPoint(navigationAgent.GetNavigationMap(
), navigationAgent.NavigationLayers, false);
          navigationAgent.TargetPosition = targetPosition;
          speed = (float)GD.RandRange(minSpeed, maxSpeed);

         }

         public override void OnPhysicsProcess(double delta)
         {
          if(navigationAgent.IsNavigationFinished())
          {
               character.currentWalkCycle += 1;
               SetMovementTarget();
               return;
          }

          Vector2 targetPosition =
navigationAgent.GetNextPathPosition();
          Vector2 targetDirection =
character.GlobalPosition.DirectionTo(targetPosition);

          Vector2 velocity = targetDirection * speed;

          if(navigationAgent.AvoidanceEnabled)
          {
               animatedSprite.FlipH = velocity.X < 0;
               navigationAgent.Velocity = velocity;
          }
          else
          {
               character.Velocity = velocity;
               character.MoveAndSlide();
          }
         }

         private void OnSafeVelocityComputed(Vector2 safeVelocity)
         {
          animatedSprite.FlipH = safeVelocity.X < 0;
          character.Velocity = safeVelocity;
          character.MoveAndSlide();
         }

         public override void OnNextTransitions()
         {
          if(character.currentWalkCycle == character.walkCycles)
          {
               character.Velocity = Vector2.Zero;
```

```
            EmitSignal(SignalName.Transition, "Idle");
        }
    }

    public override void OnEnter()
    {
        animatedSprite.Play("walk");
        character.currentWalkCycle = 0;
    }

    public override void OnExit()
    {
        animatedSprite.Stop();
    }
}
```

Додаток В

Лістинги глобальних скриптів

Лістинг В.1 – Код файлу «DayAndNightCycleManager.cs»

```
using Godot;
using System;

public partial class DayAndNightCycleManager : Node
{
    public const int MINUTES_PER_DAY = 24 * 60;
    public const int MINUTES_PER_HOUR = 60;
    public const float GAME_MINUTE_DURATION = Mathf.Tau /
MINUTES_PER_DAY;

    public float gameSpeed = 5.0f;

    public int initialDay = 1;
    public int initialHour = 12;
    public int initialMinute = 30;

    public float time = 0.0f;
    public int currentMinute = -1;
    public int currentDay = 0;

    [Signal]
    public delegate void GameTimeEventHandler(float time);
    [Signal]
    public delegate void TimeTickEventHandler(int day, int
hour, int minute);
    [Signal]
    public delegate void TimeTickDayEventHandler(int day);

    public void SetInitialTime()
    {
     int initialTotalMinutes = initialDay * MINUTES_PER_DAY +
(initialHour * MINUTES_PER_HOUR) + initialMinute;

     time = initialTotalMinutes * GAME_MINUTE_DURATION;
    }

    public void RecalculateTime()
    {
     int totalMinutes = (int)(time / GAME_MINUTE_DURATION);
     int day = (int)(totalMinutes / MINUTES_PER_DAY);
     int currentDayMinutes = totalMinutes % MINUTES_PER_DAY;
     int hour = (int)(currentDayMinutes / MINUTES_PER_HOUR);
     int minute = (int)(currentDayMinutes % MINUTES_PER_HOUR);

     if(currentMinute != minute)
     {
```

```
                currentMinute = minute;
                EmitSignal(SignalName.TimeTick, day, hour, minute);
            }

            if(currentDay != day)
            {
                currentDay = day;
                EmitSignal(SignalName.TimeTickDay, day);
            }
        }

        public override void _Ready()
        {
            SetInitialTime();
        }

        public override void _Process(double delta)
        {
            time += (float)(delta * gameSpeed *
GAME_MINUTE_DURATION);
            EmitSignal(SignalName.GameTime, time);

            RecalculateTime();
        }
    }
```

Лістинг Б.2 – Код файлу «InventoryManagemtnet.cs»

```
using Godot;
using System;
using Godot.Collections;

public partial class InventoryManagement : Node
{
   public Dictionary  inventory = new Dictionary();

   [Signal]
   public delegate void InventoryChangedEventHandler();

   public override void _Ready()
   {
   }

   public void AddCollectable(string collectableName)
   {
     if (inventory.ContainsKey(collectableName))
     {
          int currentCount = (int)inventory[collectableName];
          inventory[collectableName] = currentCount + 1;
          GD.Print($"collectableName: {collectableName}");
     }
     else
```

```
      {
           inventory[collectableName] = 1;
           GD.Print($"collectableName: else");
      }

      EmitSignal(SignalName.InventoryChanged);
   }
}
```

Лістинг Б.3 – Код файлу «ToolManager.cs»

```
using Godot;
using System;

public partial  class ToolManager : Node
{
   public Tools selectedTool {get; set;} = Tools.None;

   [Signal]
   public delegate void ToolSelectedEventHandler(Tools tool);

   public void SelectTool(Tools tool)
   {
     EmitSignal(SignalName.ToolSelected, (int)tool);
   }
}
```