Галицький фаховий коледж імені В’ячеслава Чорновола
відділення комп’ютерних технологій
циклова комісія інформатики та комп’ютерних дисциплін

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач відділення
комп’ютерних технологій
Наталія СТЕФУРАК/_____/
підпис
«___» ___ 2024 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи
освітньо-професійного ступеня «фаховий молодший бакалавр»
зі спеціальності 122 «Комп’ютерні науки»

на тему: “ Розробка комп’ютерної 3D-гри в жанрі “Sanbox””

Студент групи КН-41 Максим РИБЧАК _____
(підпис)

Керівник роботи Наталія КУЛЬЧИНСЬКА _____
(підпис)

Консультанти:
з техніко-економічного
обґрунтування Любов МЕЛЕНЧУК _____
(підпис)

нормоконтролер Надія ГАВРИШКІВ _____
(підпис)

Тернопіль - 2024

Галицький фаховий коледж імені В’ячеслава Чорновола
відділення комп’ютерних технологій
циклова комісія інформатики та комп’ютерних дисциплін

ЗАТВЕРДЖУЮ
Завідувач відділення
комп’ютерних технологій
Наталія СТЕФУРАК /_____/
підпис
«___» _____ 2023 р.

ЗАВДАННЯ

на кваліфікаційну роботу
на здобуття освітньо-професійного ступеня «фаховий молодший бакалавр»
студенту Рибчаку Максиму Анатолійовичу
(прізвище, ім’я та по-батькові студента)

1. Тема роботи Розробка комп’ютерної 3D-гри в жанрі “Sanbox”

затверджена наказом по коледжу від “27”листопада 2023р., № 234 а-н

2. Термін здачі студентом завершеної роботи “___” _____2024 р.

3. Вихідні дані до роботи результати дослідження 3D-ігор в жанрі “Sanbox”, Результат дослідження ігрових рушіїв, результат аналізу дослідження засобів моделювання

4. Перелік питань, які повинні бути розроблені:

а) основна частина аналіз та загальна характеристика комп'ютерних ігор, проєктування ігрового застосунку, реалізація ігрового продукту та його тестування

б) техніко-економічне обґрунтування аналіз ринку збуту продукту чи послуги, Розрахунок витрат на проєктування, обґрунтування необхідності розробки

5. Перелік графічного матеріалу діаграма класів, діаграма варіантів використання, діаграма діяльності

6. Консультанти роботи: Меленчук Любов Іванівна

Розділ	Консультанти	Підпис, дата	
		Завдання видано	Завдання прийнято
з техніко-економічного обґрунтування	_____ (вчена ступень, звання П.І.Б. _____ консультанта)		

КАЛЕНДАРНИЙ ПЛАН

виконання кваліфікаційної роботи

№ п/п	Найменування етапу	Терміни	
		початку	завершення
1.	Вибір теми кваліфікаційної роботи	23.11.23	
2.	Аналіз програмних рішень	24.11.23	05.01.24
3.	Вивчення технологій реалізації роботи	19.1.24	29.01.24
4.	Робота над програмним кодом, розробка графічної частини кваліфікаційної роботи	10.02.24	05.03.24
5.	Проєктування ігрового додатку	18.03.24	23.03.24
6.	Реалізація ігрового додатку	23.03.24	23.04.24
7.	Тестування застосунку і вирішення багів	24.04.24	20.05.24
8.	Аналіз економічної частини	20.05.24	30.05.24
9.	Оформлення пояснювальної записки	01.06.24	15.06.24
10.	Попередній захист кваліфікаційної роботи	17.06.24	
11.	Підготовка до захисту кваліфікаційної роботи	18.06.24	24.06.24
12.	Захист кваліфікаційної роботи	27.06.24	

7. Дата видачі завдання “_____” _____ 2023 р. Керівник _____/

Завдання прийняв до виконання _____/ /

Реферат

Кваліфікаційна робота. Розробка комп’ютерної 3D-гри в жанрі “Sanbox. 77с., 34 рисунків, 3 додатка, 7 джерел

Об’єкт дослідження – ігрові продукти спроєктовані і розроблені в жанрі “Sanbox”, засоби розробки ігор.

Метою кваліфікаційної роботи є розробка ігрового продукту в жанрі “Sanbox” на базі ігрового рушія Unity.

Завданням кваліфікаційної роботи є аналіз засобів реалізації комп’ютерних ігор, проєктування та розробка ігрового продукту в жанрі “Sanbox”.

Для розробки системи було використано середовище розробки Unity, продукт створено з допомогою програм, таких як:

– Visual Studio.

– Blender.

Написання скриптів та класів здійснюється за допомогою мови програмування C#.

Результат – запроєктований та розроблений ігровий продукт на базі ігрового рушія Unity. Продукт має простий і зручний користувацький інтерфейс а також зрозумілий геймплей.

СИСТЕМА, UNITY, C#, BLENDER, МОДЕЛЮВАННЯ, 3D-ГРА

Abstract

Qualification work. Development of a 3D computer game in the genre "Sanbox. 77 p., 34 drawings, 3 appendices, 7 sources.

The object of research is game products designed and developed in the "Sanbox" genre, game development tools.

The purpose of the qualification work is the development of a game product in the "Sanbox" genre based on the Unity game engine.

The task of the qualification work is to analyze the means of implementing computer games, design and development of a game product in the "Sanbox" genre.

The Unity development environment was used to develop the system, the product was created using programs such as:

– Visual Studio.

– Blender.

Scripts and classes are written using the C# programming language.

The result is a designed and developed game product based on the Unity game engine. The product has a simple and convenient user interface, as well as clear gameplay.

SYSTEM, UNITY, C#, BLENDER, MODELING, 3D GAME

ЗМІСТ

					КР.КН 24.564.16.000 ПЗ		
Зм.	*Арк.*	*№ докум.*	*Підпис*	*Дата*			
Розроб.		Рибчак М. А.			Розробка комп’ютерної 3D-гри в жанрі “Sanbox	*Літ.*	*Аркушів*
Перевір.		Кульчинська Н.З.				5	78
Реценз.		Сиротюк О. Б.				*ГФКімВЧ.ВКТ.ЦК ітаКД гр. КН - 41*	
Н.контр.		Гавришків Н.Г.					
Зав. відділ		Стефурак Н.А					

ВСТУП

Ігрова індустрія в сучасному світі є на досить високому рівні, з високотехнологічним програмним забезпеченням, технікою та на різноманітні платформи, починаю від мобільних пристроїв закінчуючи повноцінними ігровими кімнатами.

Ігрові додатки мають досить високі вимоги до графічного рівня, до аудіо ресурсів, до оптимізація застосунку враховуючи рівень графіки. Графічні елементи ігрових проєктів досягнули настільки високого рівня, яке інколи неможливо відрізнити від реальності. Високі потреби в продуктивності апаратного забезпеченні сприяє розвиток технічних характеристик пристроїв.

Розробка ігрового проєкту залучає велику команду працівників різних сфер. Розробка таких продуктів залучає не лише великий людський ресурс але й затрачує велику кількість фінансів та часу.

Розвиток саме Sanbox ігор в стилі фермерства спонукає деяких гравців купляти або розробляти власні ігрові консолі, наприклад консоль-автомобіль, або навіть цілих кімнат. Розробка таких проєктів може приносити доволі великий прибуток, враховуєчи що користувачі готові донатити в гру, купуючи різні моделі ігрових об’єктів або оформляти платні підписки.

Метою кваліфікаційної роботи є розробка ігрового додатку з хорошою оптимізацією, приємною графічною складовою та цікавим сценарієм і процесом гри.

1 РОЗРОБКА КОМП'ЮТЕРНОЇ 3D ГРИ В ЖАНРІ "SANDBOX"

1.1Аналіз та загальна характеристика комп'ютерних ігор

Комп’ютерні ігри – програмне забезпечення, має розважальний характер, іноді носить мету розвитку навичок за допомогою гри. Надає можливість взаємодіяти з віртуальним середовищем за допомогою різних видів гаджетів.

Існує широкий вибір жанрів комп’ютерних ігор, включаючи стратегії, шутери, квести та спортивні симулятори.

Комп’ютерні ігри поділяються на різні жанри, такі як бойовики, пригоди, стратегії, головоломки, рольові ігри і спортивні ігри. Кожен жанр має різні та індивідуальні характеристики та можливості. Сучасні ігри мають доволі реалістичні графічні рішення та якісні звукові ефекти. Такі технології, як реалістичне освітлення, відтворення тіней, фізика та анімація, роблять гру більш приємною та комфортною. Важливу роль відіграє те, як гравець взаємодіє з віртуальним світом та іншими персонажами.

Більшість комп'ютерних проєктів мають власну історію або сюжет, який розгортається під час гри. Деякі ігри мають лінійний сюжет, у якому гравець керує діями, заздалегіть продуманими розробниками, тоді як інші пропонують пропонують відкритий світ, в якому дії користувача абсолютно вільні та ні від чого не залежать.

Багато сучасних ігор мають функції для кількох гравців, які дозволяють гравцям спілкуватися та взаємодіяти через мережу. Це створює можливості для гри з друзями або знайомими.

У багатьох іграх є система прогресу та досягнень, яка заохочує гравців досліджувати ігровий світ і виконувати різноманітні завдання, щоб отримувати нагороди та покращувати рівень навичок. Сучасні ігри створюються на різні платформи, та надають можливість вибрати зручний спосіб для гри, який індивідуальний для кожного.

У центрі аналізу – технічні характеристики комп’ютерних ігор. Це включає:

- графіку;
- звуковий дизайн;
- інтерфейс користувача;
- фізику гри;
- оптимізацію.

Аналізуючи ці аспекти, можна зробити висновки про технічний рівень гри, її потенційну доступність для різної аудиторії та загальну якість виконання.

Крім технічних показників, не менш важливим є аналіз культурних аспектів комп'ютерних ігор. Це включає теми, сюжети, моральні аспекти та відображення різних культур і соціальних цінностей.

Також важливо враховувати роль ігор у формуванні спільноти, розвитку спільноти для кількох гравців та онлайн-культурі.

Короткий аналіз та загальна характеристика комп’ютерних ігор показує, що це складна та багатогранна тема дослідження, яка потребує ретельного та комплексного підходу.

Розуміння як технічних, так і культурних аспектів ігор допомагає не тільки оцінити якість ігор, але й зрозуміти їхній вплив на суспільство та особистісне зростання користувачів. Центральним аспектом аналізу та загальної характеристики комп'ютерних ігор є жанрова характеристика.

Жанр визначає типову механіку гри, елементи сюжету та основні цілі, а також впливає на сприйняття гравцем і взаємодію з ігровим світом.

Наприклад, рольові ігри наголошують на розвитку персонажа та глибоких сюжетних лініях, що дозволяє гравцям взаємодіяти з вигаданими світами та приймати важливі рішення, які впливають на хід подій.

З іншого боку, екшн-ігри зосереджені на напруженому ігровому процесі та швидкій реакції гравця, включаючи бойові сцени та місії, ініційовані ворогом.

Інші популярні жанри включають стратегічні ігри, де гравці повинні планувати та виконувати стратегічні дії, щоб досягти перемоги, і пригоди, де гравцям доводиться розгадувати головоломки та досліджувати нові місця.

Ігри-головоломки, симулятори, спортивні ігри та багато інших жанрів роблять свій внесок у різноманітність ігрового світу.

Аналізуючи конкретні жанри, ви можете краще зрозуміти, які аспекти гри привертають увагу гравців і як вони сприймають ігровий досвід. Крім того, знання жанрів допомагає розробникам краще розуміти свою цільову аудиторію та створювати привабливіші ігри. Жанри комп'ютерних ігор відображають різноманіття відомого світу та інтереси гравців.

Можна створити унікальну ігрову механіку та досвід, враховуючи як можливості, так і обмеження кожного жанру.

Екшн-ігри часто зосереджені на реакції та вправності гравця, з наголосом на швидкому та захоплюючому ігровому процесі.

Рольові ігри відображають глибокі сюжети та персонажів, дозволяючи гравцям впливати на розвиток історії та особистісне зростання своїх героїв.

Стратегія сприяє розвитку стратегічного мислення та планування, пропонуючи гравцям управляти ресурсами та вирішувати стратегічні завдання.

Пригодницькі ігри зосереджені на дослідженні світу та вирішенні головоломок, заохочуючи творчі здібності та аналітичні здібності гравців.

Ігри-головоломки розвивають логічне мислення та вирішення проблем, часто вимагаючи від гравців пошуку нестандартних рішень.

Ігри-симулятори дозволяють гравцям взяти на себе роль реальних або вигаданих персонажів, таким чином відчуваючи реалістичність життя та деталі певних видів діяльності.

Ці жанри – лише частина різноманітного світу комп'ютерних ігор.

Кожна з них має свої унікальні характеристики та приваблює різноманітних гравців, створюючи широкий вибір і можливості для нових ігрових вражень.

1.2 Аналіз існуючих розробок

Жанр комп’ютерних ігор «пісочниця» (від англ. «Sandbox» — пісочниця) відрізняється відкритим світом, у якому гравець має велику свободу дій і можливість взаємодіяти з навколишнім середовищем.

У цьому жанрі гравцям часто дають можливість вільно досліджувати ігровий світ, будувати структури, вирощувати зернові культури, виробляти ресурси та взагалі створювати та змінювати світ за бажанням.

Основними характеристиками ігор жанру «пісочниця» є відсутність чіткої лінійної структури, високий ступінь свободи у виборі дій гравця, акцент на творчість і самовираження.

Цей жанр дозволяє гравцям грати у власному темпі, експериментувати та брати участь у різноманітних заходах, таких як будівництво, виживання, дослідження та торгівля.

Популярні приклади ігор із «пісочницею» включають «Minecraft», де гравці можуть будувати будь-що за допомогою кубиків, і , яка поєднує будівництво, дослідження та боротьбу з монстрами в піксельному світі.

Жанр «пісочниця» приваблює гравців своєю відкритістю та можливістю створити щось унікальне та особливе у своєму віртуальному світі.

Він надає безпрецедентну творчу свободу і дозволяє гравцям реалізувати свої фантазії та ідеї в ігровому середовищі.

Основними характеристиками ігор у жанрі "Sandbox" є:

– Відкритий світ: Гравці мають можливість вільно досліджувати ігровий світ без обмежень, відвідуючи різноманітні локації та взаємодіючи з різними об'єктами.

– Воля дій: Гравці можуть впливати на світ гри шляхом будівництва, розміщення об'єктів, модифікації оточуючого середовища та інших дій, що відображають їхні уявлення та креативність.

– Творчість та експерименти: Головний акцент у таких іграх робиться на можливості гравця втілити свої ідеї та концепції в життя, будуючи, створюючи та експериментуючи з різноманітними аспектами гри.

– Монстри та вороги: У деяких іграх жанру "Sandbox" можуть зустрічатися монстри та вороги, які становлять загрозу для гравця. Проте їхній наявний вигляд та характеристики можуть варіюватися від ігри до ігри.

– Графічні стилі: Ігри цього жанру можуть мати як фентезійну, мальовану графіку, так і реалістичний стиль відображення світу.

Жанр "Sandbox" надає гравцям можливість відчути себе справжніми творцями та втілити свої уявлення в реальність у віртуальному середовищі без будь-яких обмежень.

Крім того, ігровий процес у жанрі пісочниці може зосереджуватися на таких аспектах, як виживання, дослідження та розвиток персонажа.

Гравці можуть зіткнутися з різними викликами, такими як пошук їжі та ресурсів, боротьба з небезпечними істотами та взаємодія з іншими гравцями в режимі для кількох гравців.

Одним із ключових аспектів успішної гри в пісочниці є її відкритість для модифікації. Багато з цих ігор дозволяють гравцям змінювати та розширювати гру за допомогою модів, надаючи необмежену кількість унікальних ігрових вражень.

Жанр пісочниці також надає можливості для співпраці та спілкування гравців. У деяких іграх гравці можуть будувати разом, розробляти масштабні

проєкти і навіть змагатися один з одним, щоб створити найкреативніші будівлі та механізми.

Загалом, жанр «пісочниця» відкриває перед гравцями безмежні можливості для творчості та самовираження у віртуальному світі, що робить його одним із найпопулярніших та захоплюючих напрямків у світі комп’ютерних ігор.

Прикладами ігор цього жанру можуть бути "Farm for your Life", " Acres". У цих іграх гравцям надається можливість вільно експериментувати та творити власну унікальну ферму, взаємодіяти з оточуючим середовищем та розвивати своє сільськогосподарське підприємство у власному темпі.

Farm for your Life – пісочниця яка поєднує елементи симулятора та стратегії.

Сюжет гри: після ураганну який пройшов по містечку і зруйнував будівлі, а також перетворив жителів на монстрів, герой повинен швидко відновити ферму і захищати містечко від монстрів.

Гра розроблена з використанням 3D-технологій, де гравець керує персонажем у виді зверху. Її графічний дизайн виконаний у стилі, що нагадує мультфільми, що надає проєкту естетичного та привабливого вигляду.

Метою гри є захист від монстрів ферми та урожаю. Для цього гравець повинен заробляти гроші продавши вирощені культури. Укріплення відбувається за допомогою захисник споруді гармат. Проєкт передбачає вирощування рослин і прокачування ферми(рисунок 1.1). Farm for your Life сподобається всім любителям симуляторів ферми.

Рисунок 1.1 – Зображення гри Farm for your Life

У грі існує система розвитку ферми, яка створює можливості для героя. Фермерська діяльність не лише забезпечує необхідними ресурсами, але й служить ефективним засобом захисту від нападів зомбі. Ця система допомагає гравцеві збільшувати ефективність боротьби з небезпекою та розвивати свої стратегічні навички.

З нюансів можна підмітити застарілий інтерфейс та низька якість моделей. Також проблемою є те що гра не оновлюється та деякі баги не виправлені.

Гра Acres про догляд за посівами та плекання землі для щедрого врожаю. Успіх залежить від здатності реагувати на ринкові замовлення, вирощуючи потрібні культури в потрібних кількостях, щоб задовольнити попит.

Ця пісочниця орієнтована на бізнес-симулятор, який дає можливість перевірити ваші навички у керуванні фермою та стратегічні здібності (рис. 1.2). Можна випробувати свої знання та вміння в управлінні ресурсами,

вирощуванні рослин, управлінні персоналом і вирішенні стратегічних завдань для ефективного функціонування ферми.

Рисунок 1.2 – Зображення гри Acres

Потрібно виконувати замовлення по вирощуванню різних сільськогосподарських культур. Створена система оновлення транспорту та обладнання. Реалізована автоматизація роботи над полями.

Мінусом даного програмного продукту є важка для розуміння бізнес модель, для більшості користувачів така система буде складною.

1.3 Визначення вимог до проєкту

Після ретельного аналізу існуючих рішень були сформульовані наступні вимоги до цього проєкту:

- якісно розроблений дизайн карти;
- реалізація динамічної зміни дня та ночі;
- впровадження системи валюти;
- система управління персонажем;
- система керування транспортом;
- механізм купівлі та продажу;

– зручний та інтуїтивно зрозумілий інтерфейс;

– реалізація головного меню гри;

– впровадження живого навколишнього середовища;

– реалізація фермерських угідь та їх обробки.

Вищезазначені вимоги вимагають ретельної розробки всіх аспектів гри, від дизайну мапи до реалізації системи ботів. Важливо забезпечити якісний дизайн мапи, щоб створити привабливе та цікаве візуальне середовище для гравця. Динамічна зміна дня і ночі додає грі реалістичності та атмосфери.

Впровадження валютної системи дозволяє гравцям здійснювати різноманітні обміни та торги, роблячи гру більш динамічною та цікавою. Система управління персонажем і транспортом дозволяє гравцям вільно пересуватися в межах гри і взаємодіяти з навколишнім світом.

Механізми купівлі та продажу різних предметів і ресурсів розвивають ігровий процес більш глибоко. Зручний та інтуїтивно зрозумілий інтерфейс зробить гру комфортною для всіх гравців.

Реалізація головного меню гри дозволить гравцям легко обирати налаштування та режими гри. Реалізація системи ботів зробить гру більш захоплюючою та цікавою для гравців і створить живе та непередбачуване середовище.

2 ПРОЄКТУВАННЯ ІГРОВОГО ЗАСТОСУНКУ

2.1 Аналіз наявних рушіїв на ринку

Для створення гри можна використовувати різні інструменти та технології, залежно від потреб та можливостей. Було розглянуто такі варіанти:

– Unity;

– Unreal Engine;

– Godot Engine;

– GameMaker Studio;

Редактор Unity – безкоштовний рушій для створення сучасних ігрових проєктів. Рушій має великий функціонал для реалізації графічних та фізичних складових.

Інтерфейс Unity складається з різних вікон, які ви можете розміщувати на свій розсуд. Це дозволяє налагоджувати ігри та програми безпосередньо в редакторі. До основних вікон належать провідник ресурсів проєкту, Інспектор поточних об'єктів, вікно попереднього перегляду, провідник сцен та провідник ієрархії ресурсів.

Проєкти Unity розділені на сцени (рівні).Він містить окремий файл, що містить кожен ігровий світ та набір об'єктів, сценаріїв та налаштувань для кожного з них. Сцена може містити як модельні об'єкти (ландшафт, персонаж, об'єкти оточення і т.д.), так і порожній ігровий об'єкт — той, який не має моделі, але задає поведінку інших об'єктів (тригери подій, точки прогресу і т. д.). З їх допомогою фізичні процеси реалізують взаємодію з кодом, взаємодія між ігровими об'єктами. Об'єктам на робочій панелі потрібен компонент перетворення для зберігання координат положення, повороту і розмірів по всіх трьох осях. Об'єкти, що відображають геометрію, за замовчуванням також мають компонент візуалізації сітки, який робить видимою модель. Різні моделі можна об'єднувати в Набори (assets) для швидкого доступу до них.

Unreal Engine — ігровий рушій, який розробляється та підтримується компанією Epic Games.

Написаний мовою програмування C++, рушій дає змогу створювати ігри для більшості операційних систем і платформ таких як Microsoft Windows, Linux, Mac OS і Mac OS X, і консолей: Xbox, Xbox 360, PlayStation 2, PlayStation Portable, PlayStation 3, Wii, Dreamcast і Nintendo GameCube. У грудні 2009 Марк Рейн продемонстрував як працює рушій Unreal Engine 3 на iPod Touch і iPhone 3GS. А у березні 2010 робота рушія була продемонстрована на комунікаторі Palm Pre, що базується на мобільній платформі webOS.

Задача Godot –найбільш інтегроване та самодостатнє середовище для розробки ігор. У цьому середовищі розробники можуть створювати ігри з нуля, не використовуючи ніяких інструментів, крім інструментів, необхідних для створення ігрового контенту (графічні елементи, музичні треки і т.д.).). Процес програмування також не вимагає жодних зовнішніх інструментів (за бажанням Ви можете скористатися зовнішнім редактором, але це дуже просто).

Загальна архітектура Godot побудована навколо концепції дерева з наслідуваних "сцен". Кожен елемент сцени (вузол) може бути повноцінною сценою в будь-який час. Тому в процесі розробки легко повністю змінити всю архітектуру проєкту, розширити його елементи в будь-якому напрямку і маніпулювати складними сценами на рівні простої Абстракції.

GameMaker: Studio – один з найпопулярніших ігрових рушіїв, що дозволяє розробляти додатки під безліч платформ. GameMaker: Studio є серйозним розвитком його попередника – Game Maker і головною відмінністю є додавання платформ, завдяки якій, а також іншим істотним доопрацюванням, GameMaker: Studio став потужним інструментом для професійної розробки. Творець і головний розробник перших шести версій

оригінального конструктора Game Maker – Марк Овермарс, наступні версії, включаючи GameMaker: Studio, розробляються компанією YoYo Games.

Безкоштовна версія (Standard) обмежена компіляцією під Windows. У порівнянні з нею, Professional версія має безліч переваг, включаючи управління ресурсами, компіляцію для macOS, Ubuntu і запуск на Android. Також, в професійній версії можна купувати окремі модулі, що розширюють функціональність програми. Версія Master Collection містить всі поточні модулі і майбутні доповнення версії 1.x.

2.2 Формалізація вимог

Фермерські симулятори в пісочниці – це унікальний жанр комп'ютерних ігор, де гравці мають багато свободи і можливість експериментувати на власних фермах. Вони можуть будувати і розширювати свої ферми на власний розсуд, вирощувати різні культури, розводити тварин і розвивати свої ферми відповідно до власних цілей і бажань.

Симулятори ферм у пісочниці дозволяють гравцям по-своєму взаємодіяти з навколишнім середовищем. Вони можуть вирішувати, які культури садити, як розвивати і розширювати будівлі та обладнання, як організувати землю. Гра дозволяє гравцям керувати всіма аспектами ферми, створюючи унікальний інтерактивний досвід.

Проаналізувавши дані ігрового жанру, та визначивши вимоги до застосунку,було спроєктовано ряд функцій, виконання яких повинен забезпечити даний проєкт (рисунок 2.1).

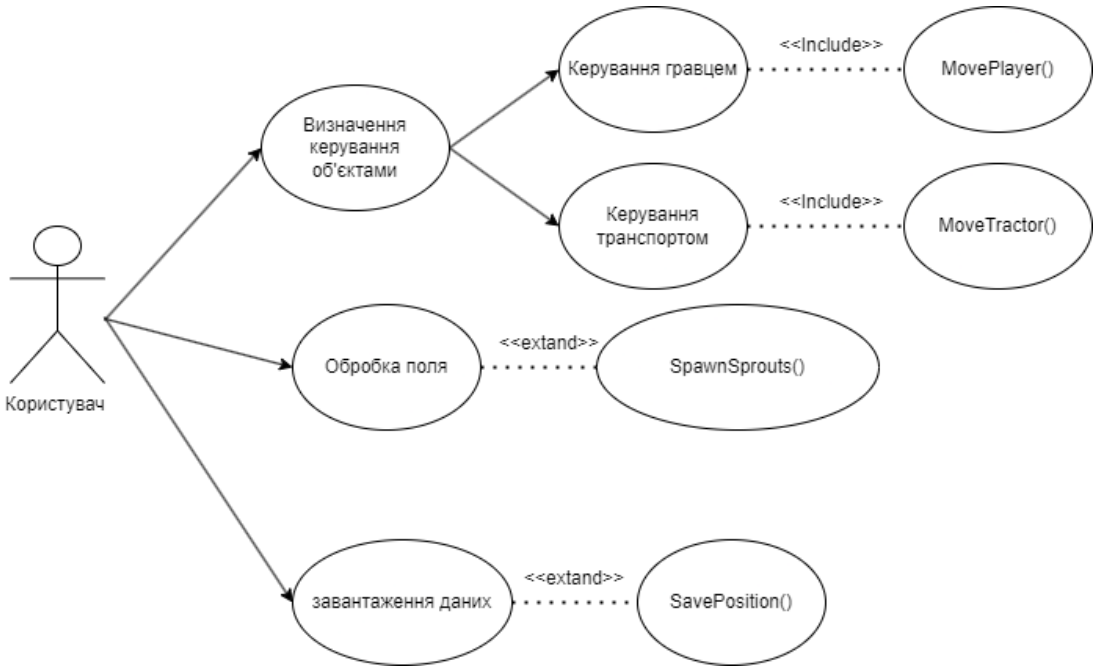


Рисунок 2.1 – Діаграма варіантів використання

Після запуску гри користувачу відкривається головне меню з наступними кнопками:

“Продовжити гру”: Гравець може продовжити попередньо збережений ігровий процес та всі дані.

“Почати гру”: Гравець вибирає карту та рівень важкості гри. Це буде безпосередньо впливати на ігровий процес.

Далі гравець вибирає рівень важкості та карту . Вибір користувача буде впливати на ігровий процес. Наприклад зростанням цін на товар та зменшення вартості товарів персонажа та дозрівання рослин.

"Налаштування": Кнопка "Налаштування" відкриває меню з різними налаштуваннями, такими як звук, графіка, контролі та інші параметри, які користувач може налаштувати за своїм вибором.

"Вихід": Натискання цієї кнопки завершує гру і виходить з неї.

Узагальнений алгоритм роботи подано на рисунку 2.2.

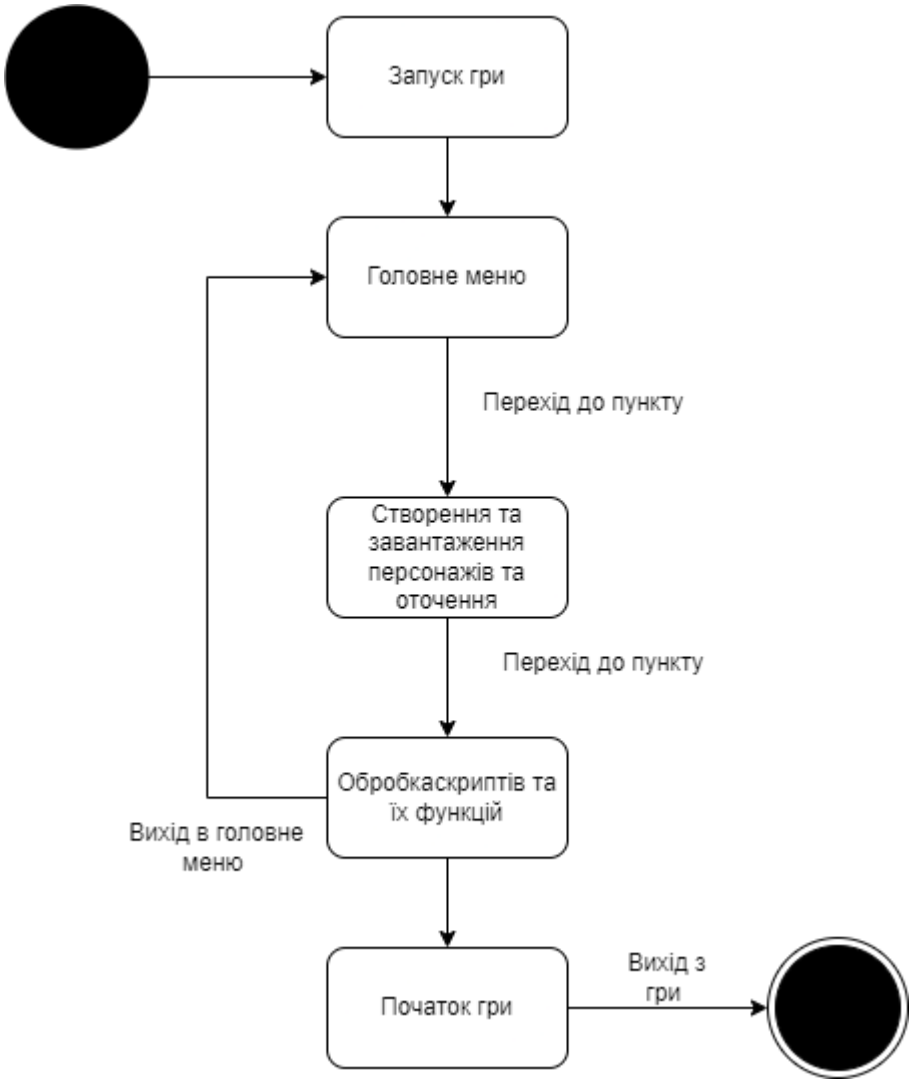


Рисунок 2.2 – Діаграма діяльності

2.3 Розробка сценарію гри

У відокремленому селі Рівнобережжя, розташованому в мальовничій долині між горами і річками, жив літній фермер на ім'я Джон. Його ферма була джерелом життя і радості для всього села. Протягом багатьох років Джон піклувався про свої поля, сади та тварин з любов'ю та відданістю. Він був відомий своєю мудрістю і добрим серцем, і кожен селянин завжди міг розраховувати на його допомогу.

Одного разу Джон отримав лист від далекого родича, в якому він повідомляв, що йому терміново потрібно виїхати за кордон, щоб вирішити сімейні проблеми. Не маючи можливості повернутися на свою ферму, він

вирішив передати її комусь, хто міг би продовжувати свою роботу з такою ж любов'ю та відданістю.

Він написав листа молодому ентузіасту на ім'я Алекс, який завжди мріяв керувати власною фермою. У ньому він побачив потенціал і щирість, необхідні для процвітання ферми Рівнобережжя..

Тепер настав час Джону продовжувати свою роботу. Новий власник ферми повинен навчитися доглядати за рослинами, вирощувати врожаї, доглядати за тваринами та підтримувати зв'язок із жителями села. Крім того, йому доведеться вирішувати різні проблеми, що виникають в господарстві, і шукати способи зробити його ще більш успішним.

Його місія-зробити Рівнобережжя багатим місцем, де кипить життя, і світ щедро подякує вам за увагу.

2.4 Обґрунтування технологій та засобів реалізації

Для реалізації даної гри було обано ігровий рушій Unity.

Unity – середовище розробки, яке поєднує функціональні можливості для створення ігор та інших інтерактивних програм. Його гнучкість є вигідною, оскільки його можна використовувати на багатьох платформах, включаючи ПК, консолі, мобільні пристрої та віртуальну реальність. Unity має різноманітні інструменти для розробки ігор, включаючи графічний редактор, фізичний движок і звукову систему. Активна спільнота користувачів і постійні оновлення сприяли популярності Unity як одного з найпопулярніших інструментів для створення ігор на планеті.

Програмний код було реалізовано засобами середовища Visual Studio та мови програмування C#.

Visual Studio – це комбінована платформа розробки програмного забезпечення, яка підтримує кілька мов програмування, включаючи C#. Visual Studio поєднується з Unity, ця комбінація дозволяє розробникам програмувати та налагоджувати свої ігри безпосередньо з Blender IDE.

Моделі ігрових об’єктів створено з використанням 3D-моделера Blender.

Blender – програмне забезпечення з повним та безкоштовним функціоналом, яке використовується для 3D-моделювання, анімації та графіки. Він здатний створювати різноманітні об’єкти та анімованих персонажів, у тому числі моделі навколишнього середовища та персонажів. Blender поєднується з Unity, що полегшує прямий імпорт моделей і анімацій у проєкт. Він також є середовищем для активної спільноти користувачів і регулярно оновлюється, що робить його популярним інструментом для розробників, які хочуть використовувати графіку та анімацію.

Крім того, ці три інструменти надають обширну документацію, навчальні посібники та онлайн-ресурси, що полегшує навчання та вдосконалення ваших здібностей щодо розробки ігор. Ці інструменти також мають спільноти активних користувачів, які завжди готові допомогти вам, якщо у вас виникнуть запитання чи проблеми.

Poly.pizza – це веб-сервер, що спеціалізується на пошуку та застосуванні префабів, матеріалів та моделей для різних проєктів. Цей ресурс надає можливість користувачам знаходити як безкоштовні, так і платні асети, які розробники застосовують в моделюванні, архітектурних візуалізаціях, анімаціях та інших цифрових проєктах.

Unity Asset Store – це офіційний онлайн-магазин для розробників, що використовують Unity, один з найпопулярніших ігрових двигунів. Магазин пропонує широкий асортимент цифрових активів, включаючи 3D-моделі, текстури, анімації, скрипти, музичні треки. Це допомагає значно прискорити процес розробки та інших інтерактивних додатків.

2.5 Проєктування структури системи

Розробка фермерської гри в Unity починається з аналізу вимог проєкту. Кожна складова гри представлена у вигляді об'єкта, який має свої атрибути. Кожен об’єкт є екземпляром класу, який є нащадком класу GameObject.

Клас “Animal”: Скрипт надає функціонал вільного пересування тварин по територію. Наявна можливість зміни курсу шляху та обертання в залежності від напрямку руху. Клас дозволяє його використовувати незалежно від типу об’єкту та його розмірів. Прив’язка скрипту до об’єкту забезпечує взаємодію з іншими об’єктами які мають колайдери.

Клас культур “SingleCropFieldManager”: виконує роль засадження території рослинами. Код діє на певній території з певним тегом. Розмір поля є універсальним. Забезпечується можливість зміни моделей саджанців на велику рослинність, тобто дозрівання культури. Засівання відбувається шляхом заповнення матриці території по координатах. Відбувається купівля насіння та продаж культури. Збереження валюти відбувається автоматично. Завантаження збережених даних також відбувається автоматично.

Структурна схема взаємодії елементів гри подана на рисунку 2.3.

Скрипт “MovemantPlayer”: Програмний код який здійснює логіку переміщення гравця по карті, можливість стрибків, взаємодія з колайдерами. Також даний скрипт використовується для переключення на транспорт гравця та взаємодію з ігровим компонентом скрипта гравця та транспорту.

Клас “moveCamera”: прикріплена до камери яка в свою чергу прикріплена до гравця. Скрипт дозволяє керувати полем зору гравця кругом осі.

Скрипт “TractorMovemant”: Реалізує роботу з колайдерами коліс транспорту. Задає їм значення потужності, прискорення, та взаємодію з поверхнею. Дає змогу повертати колесами транспорта та самим транспортом.

Після аналізу Скриптів було створено діаграму класів (рисунок 2.3).

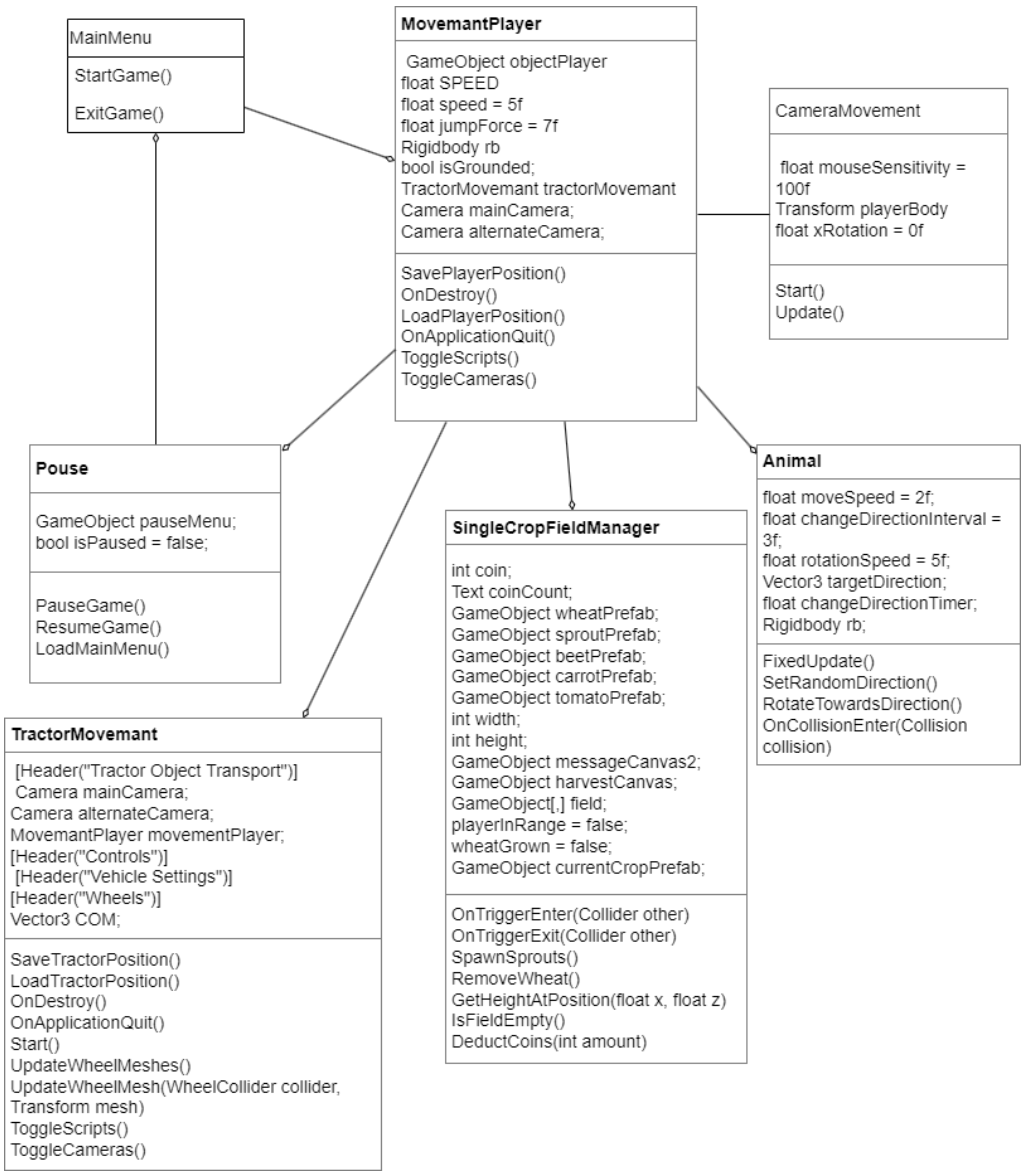


Рисунок 2.3 – Діаграма класів

Отже, проаналізувавши вимоги та визначивши основні класи та об'єкти в середовищі Unity, можна створити основу для фермерської гри, зокрема – переміщення тварин, посадка сільськогосподарських культур, переміщення акторів, керування камерами та жестами і т.д.

Важливими також є економічні елементи гри, включаючи покупку насіння і продаж врожаю за рахунок автоматичного зберігання і завантаження даних. У процесі розробки важливо постійно тестувати кожен компонент, щоб переконатися, що він працює належним чином і взаємодіє між собою. Завдяки чіткій структурі та продуманій архітектурі можна створити захоплюючу та функціональну фермерську гру.

3 РЕАЛІЗАЦІЯ ІГРОВОГО ПРОДУКТУ ТА ЙОГО ТЕСТУВАННЯ

3.1 Створення основних об'єктів гри

Основними об’єктами гри будуть рослини, тварини, споруди, поля, транспорт. За основу було взято ассети з спеціальних вебзастосунків. Після чого потрібно їх імпортувати в ігровий рушій (рисунок 3.1).

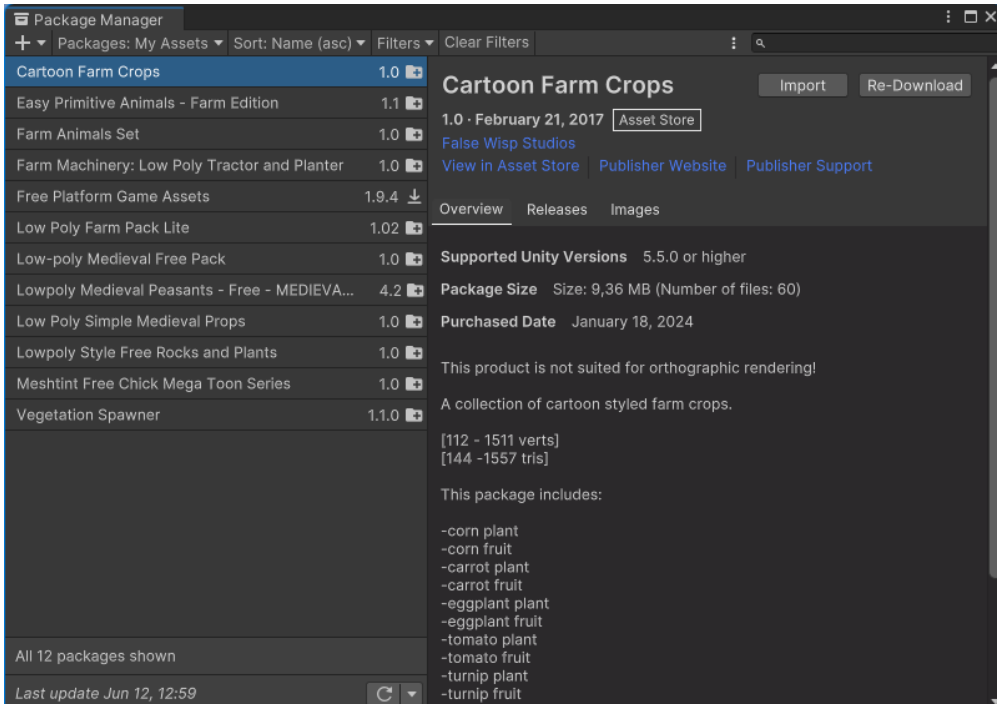


Рисунок 3.1 – Імпорт ассетів

Після чого за можна віредагувати моделі та матеріали, яке це показано на рисунку 3.2 в середовищі “Blender”. Після редагування модель імпортується в Unity та перетворювалась в Prefab.

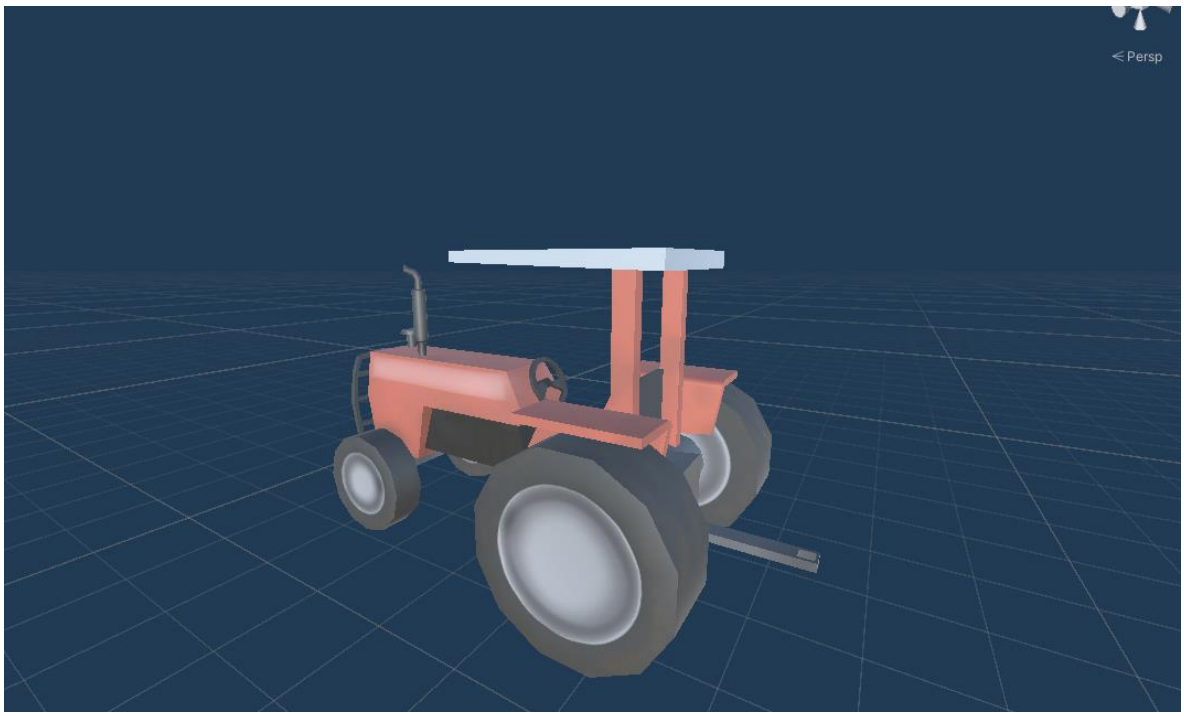


Рисунок 3.2 – Моделювання

Після створення та редагування потрібних моделей можна переходити до створення навколишнього середовища та його функціоналу.

Для початку необхідно розробити освітлення середовища. Для цього потрібно створити моделі ліхтариків (рисунок 3.3). Для їх функціонування додамо Point Light світла та налаштовуємо цей елемент (рисунок 3.4). Визначаємо його колір інтенсивність та яскравість, а також вибираємо чи буде це напрямлене світло чи загальне.

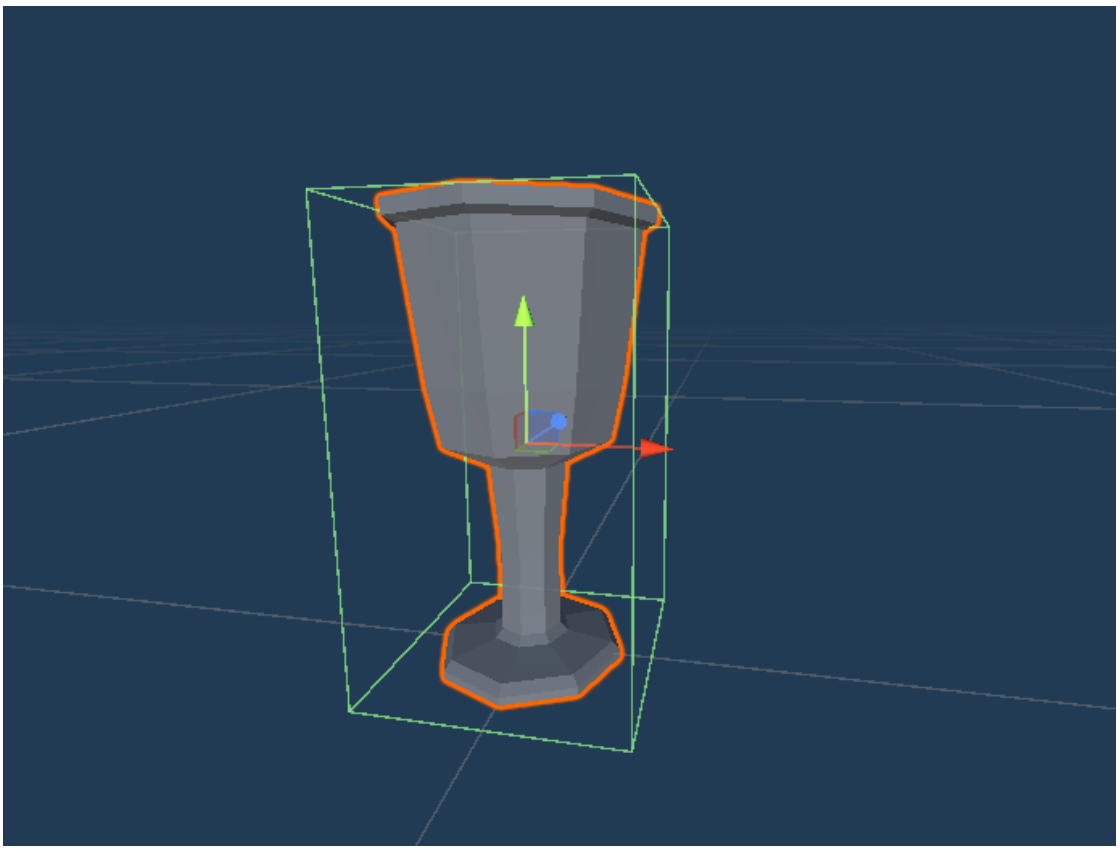


Рисунок 3.3 – Модель ліхтаря

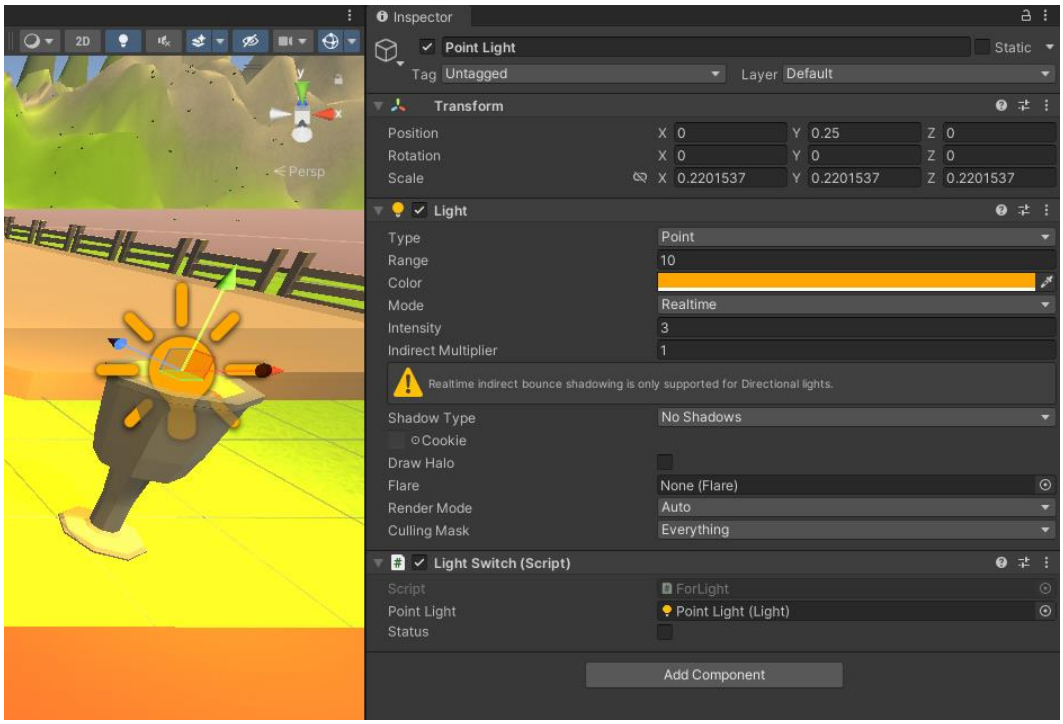


Рисунок 3.4 – Налаштування світла

Також створюємо об’єкт “Terrain”, він призначений для створення ландшафту. Він створює деталізовані та великі форми рель’єфу які добре оптимізовані та не навантажують систему. Має обширне меню налаштувань об’єкту (рисунок 3.5). За допомогою цього елементу ми заповнюємо карту травою, деревами, кущами та іншими об’єктами. Також даний об’єкт виконує текстурування поверхні. Великим плюсом є те що в цьому об’єкті є підтримування тіней, колайдерів та фізичних властивостей. Створене навколишнє середовище показано на рисунку 3.6.

Рисунку 3.6 – Об’єкт Terrain

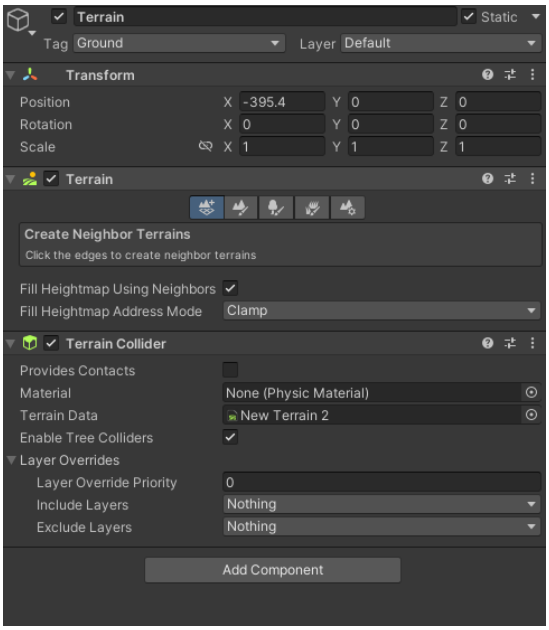


Рисунок 3.5 – Налаштування “Terrain”

В результаті було заповнене навколишнє середовище. Додано фізичні властивості, тіні, матеріали та текстури.

3.2 Розробка функціоналу гри

Для того щоб керувати гравцем, потрібно написати певний скрипт, проте для того щоб він міг взаємодіяти з навколишнім середовищем потрібно додати колайдери. Це надасть твердості текстурам та дасть змогу виконувати певний функціонал та використовувати теги.

До самого гравця додаємо наступні елементи:

– “Collider”;

– “Rigidody”;

– “Pouse Manager (Script)”;

– “Movemant Player (Script)”.

Ці елементи нададуть фізичних властивостей персонажу. “Pouse Manager (Script)” створює можливість відкриття паузи.

Для реалізації паузи створимо об’єкт “Canvas” який буде спочатку неактивний та невидимий до його активації. Він повинен зупиняти ігровий процес, для цього змінюємо стан часу на - 0, для продовження гри повертаємо значення. Також пауза активовує можливість виходу з гри в головне меню. Програмний код скрипта поданий в лістингу 3.1.

Лістинг 3.1 – Код скрипта паузи

```
void PauseGame()
{
    pauseMenu.SetActive(true); // Відображення меню паузи
    Time.timeScale = 0; // Зупинка часу в грі
    isPaused = true; // Встановлюємо прапорець паузи в true
}
public void ResumeGame()
{
    pauseMenu.SetActive(false); // Приховання меню паузи
    Time.timeScale = 1; // Відновлення часу в грі
    isPaused = false; // Встановлюємо прапорець паузи в
false
}
```

```
    public void LoadMainMenu()
    {
        Cursor.visible = true; // Показуємо курсор при виході в
головне меню
        Cursor.lockState = CursorLockMode.None; // Розблоковуємо
курсор
        isPaused = false; // Перевстановлюємо прапорець паузи в
false перед завантаженням головного меню
        SceneManager.LoadScene("Menu");      //      Завантаження
головного меню
    }
    ”Movemant Player (Script)”
```

В даному скрипті використано такі функції:

– “PauseGame()” – це функція яка зупиняє час та активує “canvas”.

– “ResumeGame()” – це функція яка відновлює час та дизактивує “canvas”.

– “LoadMainMenu()” – функція яка знімає прапорець, що гра на паузі, розблокування курсора та завантажується сцена з головним меню.

“Movemant Player (Script)” надає можливість пересування персонажа по карті, стрибок, біг, перехід в транспортний засіб, збереження та завантаження даних про позицію персонажа на карті. Він приймає наступні значення:

– SPEED – швидкість бігу;

– Speed – швидкість ходьби;

– Jump Force – сила стрибку;

– Main Camera – об’єкт камери гравця;

– Alternete camera – об’єкт камери транспорту.

Функціонал пересування показано в лістингу 3.2.

Лістинг 3.2 – код пересування гравця

```
    float moveHorizontal = Input.GetAxis("Horizontal");
    float moveVertical = Input.GetAxis("Vertical");
    Vector3 movement = new Vector3(moveHorizontal, 0.0f,
moveVertical);
```

```
movement = movement.normalized * speed * Time.deltaTime;
transform.Translate(movement);
isGrounded = Physics.Raycast(transform.position,
Vector3.down, 1f);
if (Input.GetKeyDown(KeyCode.Space) && isGrounded)
{
    rb.AddForce(Vector3.up * jumpForce, ForceMode.Impulse);
}
if (Input.GetKey(KeyCode.LeftShift))
{
    speed = SPEED; // Задаємо швидкість спрінту
}
else
{
    speed = 5f; // Повернути звичайну швидкість руху, якщо
клавіша не утримується
}
```

Функціонал перемикання від гравця на транспорт показано в лістингу 3.3.

Лістинг 3.3 – код зміни об’єкту

```
void ToggleScripts()
 {
     if (tractorMovemant != null)
     {
         bool isPlayerScriptEnabled = this.enabled;
         this.enabled = !isPlayerScriptEnabled;
         tractorMovemant.enabled = isPlayerScriptEnabled;
     }
     else
     {
         Debug.LogWarning("TractorMovemant script is not
assigned in the inspector.");
     }
```

```
    }
    void ToggleCameras()
    {
        if (mainCamera != null && alternateCamera != null)
        {
            mainCamera.enabled = !mainCamera.enabled;
            alternateCamera.enabled = !alternateCamera.enabled;
        }
        else
        {
            Debug.LogWarning("Cameras are not assigned in the
inspector.");
        }
    }
```

ToggleScripts() функціонал вимкнення скрипту керування гравцем та перемикання на керування транспортом.

ToggleCameras() функціонал перемикання на інший об’єкт камери.

Функціонал збереження даних про місцерозташування гравця подано в літстингу 3.4.

Лістинг 3.4 – код збереження позиції гравця.

```
void ToggleScripts()
    {
        if (tractorMovemant != null)
        {
            bool isPlayerScriptEnabled = this.enabled;
            this.enabled = !isPlayerScriptEnabled;
            tractorMovemant.enabled = isPlayerScriptEnabled;
        }
        else
        {
            Debug.LogWarning("TractorMovemant script is not
assigned in the inspector.");
        }
```

```
    }
    void ToggleCameras()
    {
        if (mainCamera != null && alternateCamera != null)
        {
            mainCamera.enabled = !mainCamera.enabled;
            alternateCamera.enabled = !alternateCamera.enabled;
        }
        else
        {
            Debug.LogWarning("Cameras are not assigned in the
inspector.");
        }
    }
```

Даний скрипт зберігає дані при закритті гри при переході на сцену в головне меню. При повторному вході в гру дані заватажуються з пам’яті пристрою.

Так як камера прив’язана до гравця то потрібно створити функціонал огляду за допомогою камери. Камера повинна керуватися мишкою навколо осі та мати налаштовану чутливість. Функціонал камери подано в лістингу 3.5.

Лістинг 3.5 – код керування камерою

```
    float mouseX = Input.GetAxis("Mouse X") * mouseSensitivity *
Time.deltaTime;
    float mouseY = Input.GetAxis("Mouse Y") * mouseSensitivity *
Time.deltaTime;

    // Обернути гравця на основі руху миші по осі X
    playerBody.Rotate(Vector3.up * mouseX);

    // Обернути камеру на основі руху миші по осі Y
    xRotation -= mouseY;
```

```
    xRotation = Mathf.Clamp(xRotation, -90f, 90f); // Обмежити обертання камери, щоб не дивилась занадто вгору або вниз
    transform.localRotation = Quaternion.Euler(xRotation, 0f, 0f);
```

Він приймає наступні значення:

– Mouse Sensitivity – чутливість мишки;

– Player Body – модель гравця.

Наступним важливим кроком стане розробка функціоналу посіву рослин. Скрипт буде аналізувати контакт гравця з полем. Це відбувається за допомогою тегів. Скрипт визначає чи стикаються колайдери з відповідним тегом. Для виставлення тегу потрібно щоб об’єкт мав колайдер, на колайдері має стояти позначка “IsTrigger”. Це надає можливість програмному коду бачити взаємодію між двома колайдерами та активує певний фрагмент коду. Тоді надається можливість засіювати поле, та виводиться на екран певний текст. Для цього було створено дві функції. Функціонал першої подано в лістингу 3.6.

Лістинг 3.6 – код перевірки контакту об’єктів

```
private void OnTriggerEnter(Collider other)
 {
     if (other.CompareTag("Field"))
     {
         playerInRange = true;
         bool isFieldEmpty = IsFieldEmpty();
         if (isFieldEmpty)
         {
             messageCanvas2.SetActive(true);
         }
         else
         {
             messageCanvas2.SetActive(false);
             if (wheatGrown)
```

```
            {
                harvestCanvas.SetActive(true);
            }
        }
```

В ньому відбувається перевірки чи колайдери контактують, після чого активується текст так надається можливість засаджання культури.

В другій функції використано наступний лістинг 3.7.

Лістинг 3.7 – код перевірки виходу з площі поля

```
private void OnTriggerExit(Collider other)
{
    if (other.CompareTag("Field"))
    {
        playerInRange = false;
        messageCanvas2.SetActive(false);    //    Приховуємо
канвас
        harvestCanvas.SetActive(false); // Приховуємо канвас
    }
}
```

Тепер розглянемо функцію яка засіває поле спочатку саджанцями які потім перетворюються в дорослу культуру. Також відбувається керування текстом на екрані користувача. Створення моделей відбувається по матричній системі на площі поля. Два цикли проходять по повній довжині та ширині поля. Для кожної точки обчислюється позиція для спавну (поява гравців і мобів в певній точці ігрового світу), після чого викликається функція для створення об’єкта паростка. Для переходу на великі рослини використовується такий ж метод з циклами, але відбувається додатково знищення об’єкта паростка та заміна його на дорослу культуру. Лістинг даного функціоналу подано в лістингу 3.8.

Лістинг 3.8 – код посіву рослин

```
private IEnumerator SpawnSprouts()
 {
```

```
            messageCanvas2.SetActive(false); // Приховуємо канвас з
повідомленням про спавн
            if (playerInRange) // Перевіряємо, чи гравець на полі
            {
                Vector3 startPosition = new
Vector3(transform.position.x - width / 2, transform.position.y,
transform.position.z - height / 2);
                for (int x = 0; x < width; x++)
                {
                    for (int y = 0; y < height; y++)
                    {
                        Vector3 spawnPosition = new
Vector3(startPosition.x + x, GetHeightAtPosition(startPosition.x
+ x, startPosition.z + y), startPosition.z + y);
                        field[x, y] = Instantiate(sproutPrefab,
spawnPosition, Quaternion.identity);
                    }
                }
                yield return new WaitForSeconds(10f); // Зачекати
10 секунд
                for (int x = 0; x < width; x++)
                {
                    for (int y = 0; y < height; y++)
                    {
                        if (field[x, y] != null)
                        {
                            Destroy(field[x, y]);
                            Vector3 spawnPosition = new
Vector3(startPosition.x + x, GetHeightAtPosition(startPosition.x
+ x, startPosition.z + y), startPosition.z + y);
                            field[x, y] =
Instantiate(currentCropPrefab, spawnPosition,
Quaternion.identity);
                        }
                    }
```

```
            }
            wheatGrown = true; // Встановлюємо прапорець, що
зійшла пшениця
            if (playerInRange)
            {
                harvestCanvas.SetActive(true);
            }
        }
    }
```

Наступна функція видаляє рослини з поля та додає валюту гравцеві та зберігає її в пам’яті. Для видалення використовується аналогічний метод з циклами як в попередній функції (лістинг 3.9).

Оновлення кількості монет відбувається збиранням даних про наявну валюту, після чого відбувається зароблена валюта та зберігається в пам’ять пристрою. Після цього приховується текст з повідомленням про збирання врожаю.

Відбувається перевірка на присутність гравця на полі, якщо так то виводиться текст з повідомленням про взаємодію з полем.

Лістинг 3.9 – код видалення рослин з поля

```
private void RemoveWheat()
{
    for (int x = 0; x < width; x++)
    {
        for (int y = 0; y < height; y++)
        {
            if (field[x, y] != null)
            {
                Destroy(field[x, y]);
                field[x, y] = null;
            }
        }
    }
```

```
            coin  =  PlayerPrefs.GetInt("coins");  //  призначення
зміної
            PlayerPrefs.SetInt("coins", coin + 10);
            coinCount.text = (coin + 10).ToString(); // додавання
грошей в пам'ять
            wheatGrown = false; // Reset wheatGrown flag
            harvestCanvas.SetActive(false); // Приховуємо канвас з
повідомленням про збирання
            if (playerInRange)
            {
                messageCanvas2.SetActive(true);    //    Показуємо
канвас з повідомленням про взаємодію, якщо гравець ще в зоні
            }
        }
```

Важливу роль відіграє функція яка робить перевірку на те, чи поле засаджене і чим. Також переглянемо функцію на віднімання валюти при купівлі саджанців на лістингу 3.10.

Лістинг 3.10 – код перевірки стану поля та збору валюти

```
private bool IsFieldEmpty()
    {
        // Перевіряємо, чи всі елементи поля порожні
        for (int x = 0; x < width; x++)
        {
            for (int y = 0; y < height; y++)
            {
                if (field[x, y] != null)
                {
                    return false;
                }
            }
        }
        return true;
    }
```

```
private void DeductCoins(int amount)
{
    coin -= amount;
    PlayerPrefs.SetInt("coins", coin);
    coinCount.text = coin.ToString();
}
```

Для правильної роботи скрипту потрібно передати відповідні дані та вказати правильний розмір поля в скрипт рушія гри (рисунок 3.6).

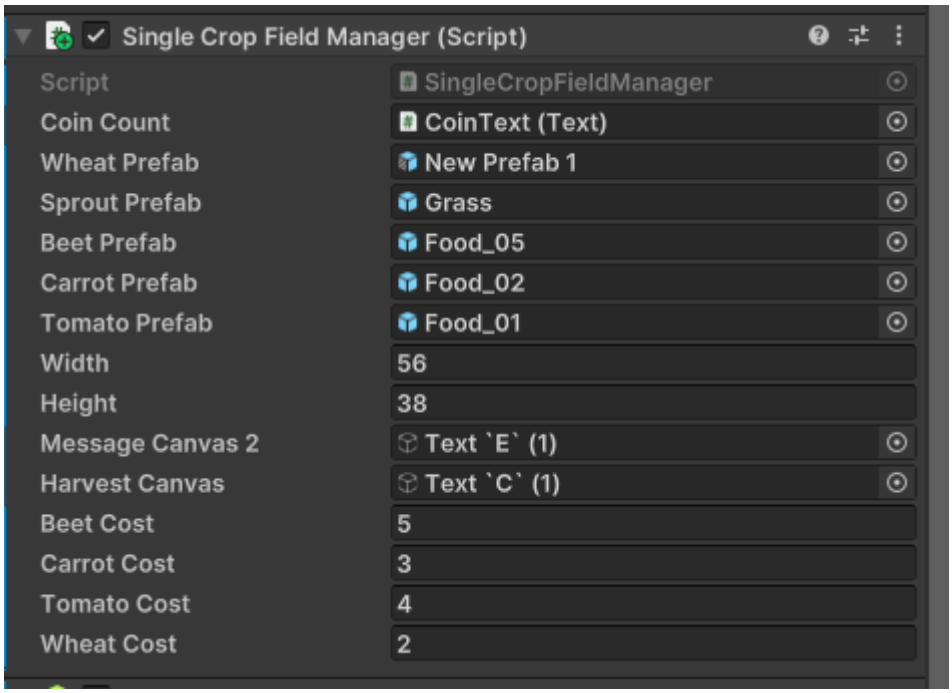


Рисунок 3.6 – Дані скрипту

Наступним розглянемо функціонал транспорту. Насамперед потрібно створити окремо компоненти “Wheel Collider”. Це компонент для моделювання коліс в Unity, забезпечує реалістичну поведінку транспорту та змогу ним керувати.

Властивості колеса:

– Radius (радіус): Визначає радіус колеса. Ця властивість впливає на те, наскільки високо колесо піднімає транспортний засіб над землею;

– Suspension Distance (відстань підвіски): Визначає максимальну відстань, на яку колесо може рухатися вверх-вниз, імітуючи підвіску транспортного засобу;

– Suspension Spring (пружина підвіски): Включає властивості пружини (жорсткість та демпфірування), які визначають, як підвіска реагує на нерівності дороги;

– Mass (маса): Маса колеса. Хоча колеса зазвичай мають невелику масу порівняно з кузовом транспортного засобу, ця властивість все одно може впливати на фізику руху;

– Forward Friction (поздовжнє зчеплення): Визначає, як колесо взаємодіє з поверхнею при прискоренні та гальмуванні;

– Sideways Friction (бічне зчеплення): Визначає, як колесо взаємодіє з поверхнею при поворотах.

Після цього ми співставляємо колайдер з сіткою колеса (рисунок 3.7). Важливим аспектом є те що сітка колеса під час початку гри починає ставати на місце колайдера. Це потрібно врахувати, тому що можуть бути негативні наслідки.

Рисунок 3.7 – Колайдер коліс

Далі налаштовуємо характеристики колеса (рисунок 3.8).

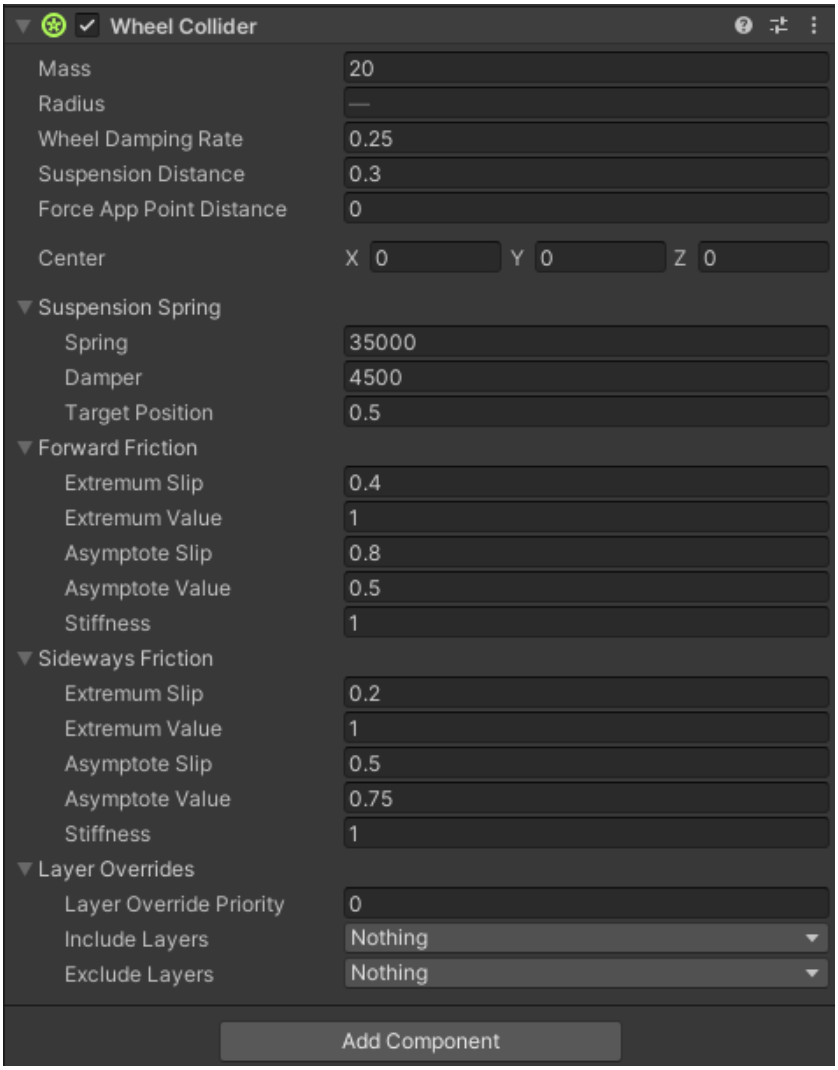


Рисунок 3.8 – Характеристика колеса

Налаштування для об'єкта:

– public Transform objectToTransport - об'єкт, який будемо переносити;

– public float transportDistance - відстань транспортування.

Посилання на камери та скрипт руху гравця:

– public Camera mainCamera - головна камера;

– public Camera alternateCamera - альтернативна камера;

– public MovemantPlayer movementPlayer - скрипт руху гравця.

Rigidbody та керування рухом:

– public Rigidbody rb - Rigidbody трактора;

– public float Accel – Прискорення;

– public float Brakes – Гальма;

– public float Steering – Керування;
– public float EnginePower - Потужність двигуна;
– public float BrakeForce - Сила гальм;
– public float SteerAngle - Кут повороту;
– public WheelCollider[] FrontWheels - Передні колеса;
– public WheelCollider[] RearWheels - Задні колеса;
– public Transform[] FrontWheelMeshes - Моделі передніх коліс;
– public Transform[] RearWheelMeshes - Моделі задніх коліс.

Тепер розглянем функціональний код для збереження та завантаження позиції транспорту на мапі. Збереження відбувається під час закриття програми. Зберігаються координати транспорту. Після повторного входу в гру координати зчитуються в функції “Start()” та об’єкт переноситься на певну точку вказану координами. Відповідний функціонал описаний в лістингу 3.11.

Лістинг 3.11 – код збереження та завантаження позиції транспорта

```
void SaveTractorPosition()
{
    PlayerPrefs.SetFloat("TractorPosX",
transform.position.x);
    PlayerPrefs.SetFloat("TractorPosY",
transform.position.y);
    PlayerPrefs.SetFloat("TractorPosZ",
transform.position.z);
    Debug.Log("Tractor position saved: " +
transform.position);
}
void LoadTractorPosition()
{
    float tractorPosX = PlayerPrefs.GetFloat("TractorPosX",
0f);
    float tractorPosY = PlayerPrefs.GetFloat("TractorPosY",
0f);
```

```
        float tractorPosZ = PlayerPrefs.GetFloat("TractorPosZ",
0f);
        Debug.Log("Tractor position loaded: " + tractorPosX + ",
" + tractorPosY + ", " + tractorPosZ);
        transform.position    =    new    Vector3(tractorPosX,
tractorPosY, tractorPosZ);
    }
```

OnDestroy() і OnApplicationQuit(): Викликають збереження позиції при руйнуванні об'єкта або виході з програми лістинг 3.12.

Лістинг 3.12 – код збереження при виході з гри

```
void OnDestroy()
{
    SaveTractorPosition();
}
void OnApplicationQuit()
{
    SaveTractorPosition();
}
```

Наступна функція реалізовує обертання колеса транспорту. Приймає змінні про потужність двигуна, прискорення, силу тормозів, та кут повороту коліс, керування транспортом та реалізовує даний функціонал. Оновлюється візуальні моделі коліс. Активується режим “IsKinematic” який не дає транспорту котитися після виходу з нього. Надає можливість повернутися на керування гравцем та його камерою Функціонал відбувається на кожному кадрі гри. Нижче наведено лістинг 3.13.

Лістинг 3.13 – код керування колесами

```
void Update()
 {
     rb.isKinematic = false;

     // Toggle scripts and cameras on Enter key press
     if (Input.GetKeyDown(KeyCode.Return))
```

```
            {
                if (objectToTransport != null)
                {
                    objectToTransport.gameObject.SetActive(true);
                    // Визначаємо позицію, куди треба перемістити
об'єкт
                    Vector3 targetPosition = transform.position +
transform.up * desiredHeightAboveTractor;

                    // Переміщуємо об'єкт до цієї позиції
                    objectToTransport.position = targetPosition;
                }
                foreach (WheelCollider wheel in FrontWheels)
                {
                    wheel.motorTorque = 0;
                }
                foreach (WheelCollider wheel in RearWheels)
                {
                    wheel.motorTorque = 0;
                }
                rb.isKinematic = true;
                ToggleScripts();
                ToggleCameras();
            }

            Accel = Input.GetAxis("Vertical"); // Use the vertical
axis for forward/reverse
            Brakes = Input.GetAxis("Jump");    // Use the jump axis
(typically space bar) for brakes
            Steering = Input.GetAxis("Horizontal");

            // Apply motor torque to front and rear wheels
            foreach (WheelCollider wheel in FrontWheels)
            {
                wheel.motorTorque = EnginePower * Accel;
```

```
}

foreach (WheelCollider wheel in RearWheels)
{
    wheel.motorTorque = EnginePower * Accel;
}

// Apply brake force
if (Brakes > 0)
{
    foreach (WheelCollider wheel in FrontWheels)
    {
        wheel.brakeTorque = BrakeForce * Brakes;
    }
    foreach (WheelCollider wheel in RearWheels)
    {
        wheel.brakeTorque = BrakeForce * Brakes;
    }
}
else
{
    foreach (WheelCollider wheel in FrontWheels)
    {
        wheel.brakeTorque = 0;
    }
    foreach (WheelCollider wheel in RearWheels)
    {
        wheel.brakeTorque = 0;
    }
}

// Apply steering angle to front wheels
foreach (WheelCollider wheel in FrontWheels)
{
    wheel.steerAngle = SteerAngle * Steering;
```

```
            }

            // Update wheel meshes position and rotation
            UpdateWheelMeshes();
        }
    void UpdateWheelMeshes()
    {
        for (int i = 0; i < FrontWheels.Length; i++)
        {
            UpdateWheelMesh(FrontWheels[i],
FrontWheelMeshes[i]);
        }
        for (int i = 0; i < RearWheels.Length; i++)
        {
            UpdateWheelMesh(RearWheels[i], RearWheelMeshes[i]);
        }
    }
```

Друга функція реалізується в попередній. Вона використовує метод колайдера, щоб отримати позицію і обертання колеса а потім встановлює текстуру колеса відповідно до колайдера. Це забезпечує обертання текстури. Функція показана в лістингу 3.15.

Лістинг 3.15 – код співставлення текстури з колайдером

```
void UpdateWheelMesh(WheelCollider collider, Transform mesh)
{
    Vector3 position;
    Quaternion rotation;
    collider.GetWorldPose(out position, out rotation);
    mesh.position = position;
    mesh.rotation = rotation;
}
```

Також потрібно оформляти перемикання активності скрипта трактора та гравця лістинг 3.16.

Лістинг 3.16 – код перемикання стану скриптів

```
void ToggleScripts()
{
    if (movementPlayer != null)
    {
        bool isPlayerScriptEnabled = this.enabled;
        this.enabled = !isPlayerScriptEnabled;
        movementPlayer.enabled = isPlayerScriptEnabled;
    }
    else
    {
        Debug.LogWarning("TractorMovemant script is not assigned in the inspector.");
    }
}
```

А також потрібно змінювати камеру. Функціонал подано в лістинг 3.17.

Лістинг 3.17 – код зміни камери

```
void ToggleCameras()
{
    if (mainCamera != null && alternateCamera != null)
    {
        mainCamera.enabled = !mainCamera.enabled;
        alternateCamera.enabled = !alternateCamera.enabled;
    }
    else
    {
        Debug.LogWarning("Cameras are not assigned in the inspector.");
    }
}
```

Таким чином було реалізовано основний функціонал пересування гравця, його взаємодія з навколишнім середовищем та іншими об’єктами. Створено керування транспортним засобом, керування його колайдерами та

текстурами. Реалізація вирощення культури рослин та їх збір. Збереження та завантаження даних.

Тепер переглянемо функціонал зміни дня та ночі. В наступному скрипті ми створимо можливість змінювати активний стан об’єкту світла. Зміна буде встановлюватися відповідно до вказаного часу в функції. Функціонал вказано в лістингу 3.18.

Лістинг 3.18 – код освітлення

```
    public Light pointLight; // Посилання на елемент типу Point
Light
    public bool status;
    void Start()
    {
        pointLight.enabled = !status;

        // Запускаємо метод ToggleLight кожні 6 секунд
        InvokeRepeating("ToggleLight", 0f, 360f);//ToggleLight",
0f, 360f
    }
    void ToggleLight()
    {
        // Зміна стану світла (увімкнено або вимкнено)
        pointLight.enabled = !pointLight.enabled;
    }
```

3.3 Розробка меню гри

Тепер створимо головне меню гри. В ньому повинен бути створений основний функціонал старту та виходу з гри. Для початку створимо елеменет UI елемент “Canvas” та додамо до фон меню (рисунок 3.9).

Рисунок 3.9 – Фонове зображення меню

Тепер створюємо пустий об’єкт та називаємо його “MainMenu” який стане зоною для створених кнопок. Та дає змогу на них реагувати. Після цього створюємо кнопки (рисунок 3.10) та додаємо їх до пустого об’єкта (рисунок 3.11).

Рисунок 3.10 – Кнопка старту

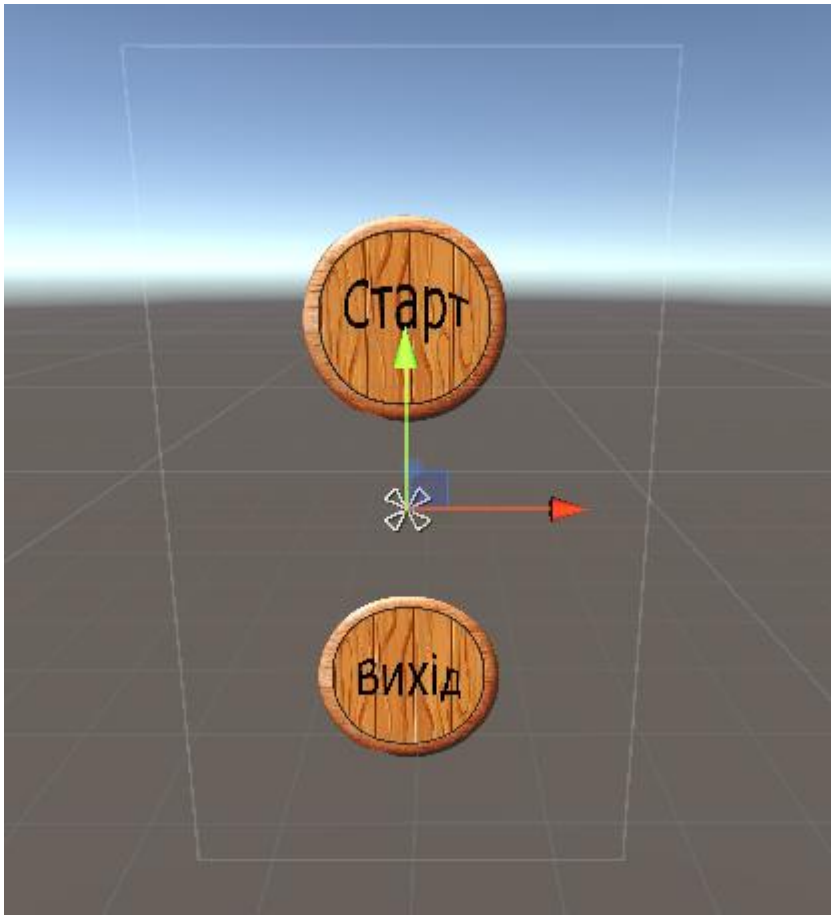


Рисунок 3.11 - Область кнопок

Тепер розглянемо функціонал меню який надасть взаємодіяти з кнопками та для кожної створимо окрему функці. Розглянемо функції в лістингу 3.19.

Лістинг 3.19 – код кнопок меню

```
public void StartGame()
{
    Time.timeScale = 1;

SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex
+ 1);
}
public void ExitGame()
{
    Application.Quit();
```

```
}
```

Функція старту гри відновлює показник часу та завантажує наступну сцену в білдері гри.

Тепер треба прив’язати ці функції до кнопок за допомогою “OnClick” (рисунок 3.12).

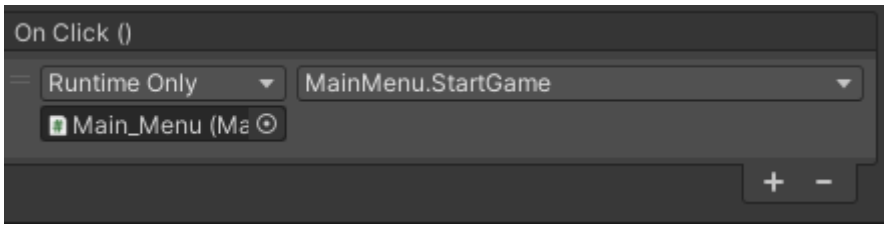


Рисунок 3.12 – Метод “OnClick”

Після цього кнопки відповідають відповідній функції.

Отже, було створено меню гри з активними UI-елементами кнопок, та підключено відповідний функціонал до кожної з них. Розроблено можливість переходу між сценами та вихід з сцени.

3.4 Тестування комп’ютерної гри

Тестування розпочнемо з взаємодії гравця з полями. Під час розташування гравця на полі повинно виводитися текстове поле (Рисунок 3.13) та має бути надана можливість засаджувати поле (Рисунок 3.14).

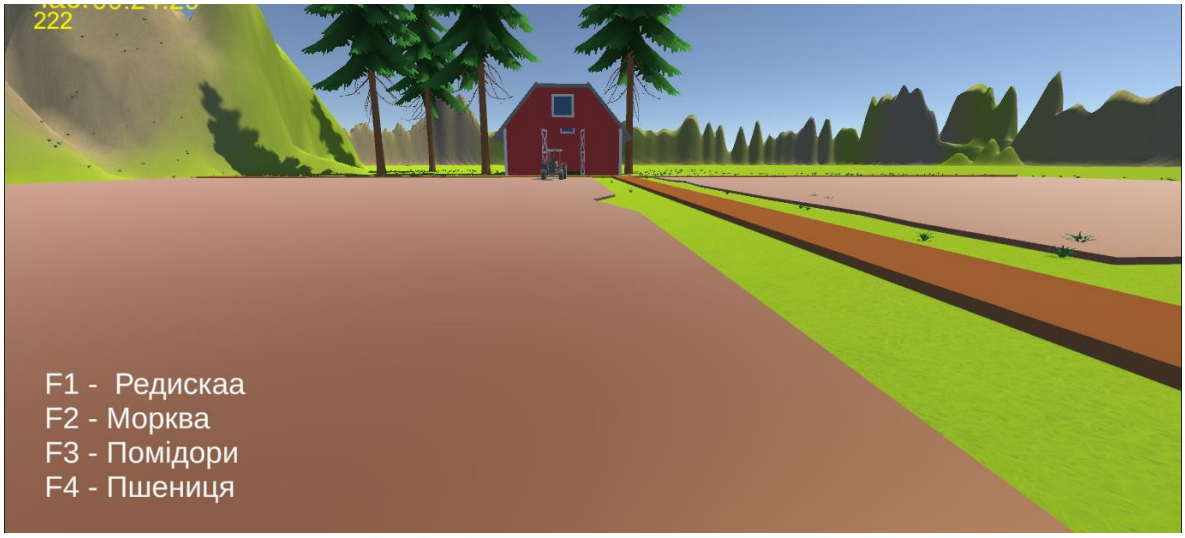


Рисунок 3.13 – Тестування виводу текстового повідомлення

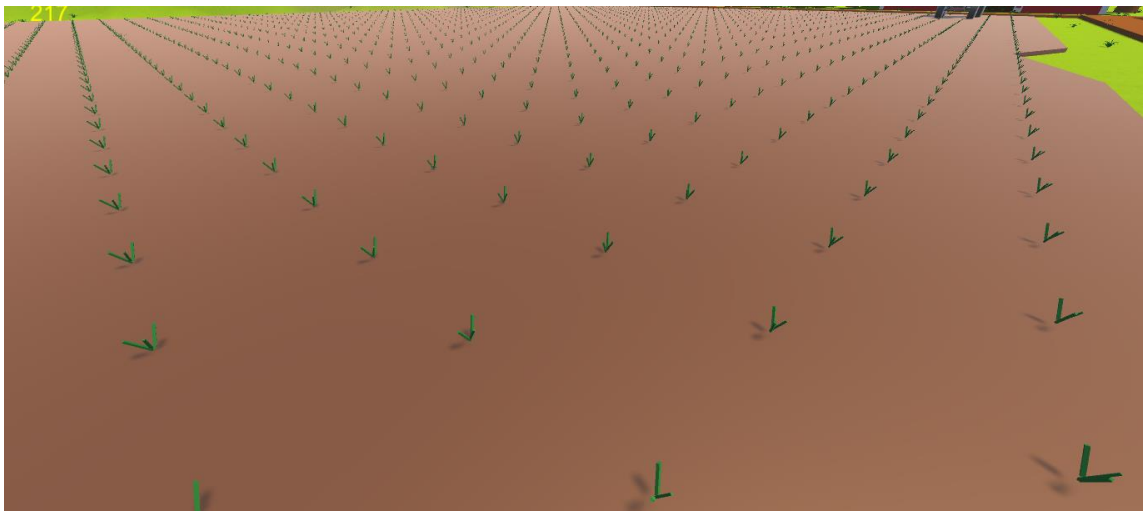


Рисунок 3.14 – Активна можливість засівання поля

Наступним етапом стане перевірка паузи. Під час натискання кнопки “ESC” гра повинна зупинятися та виводитися діалогове меню (рисунок 3.15).

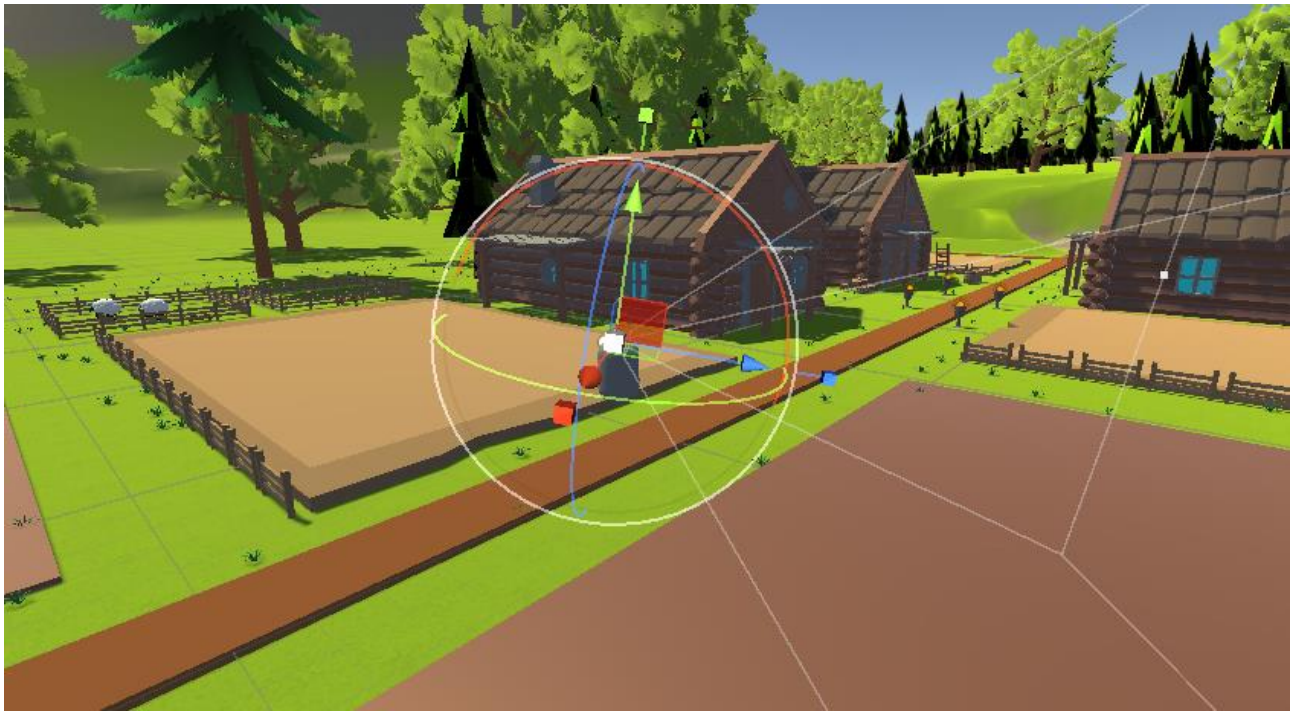


Рисунок 3.15 – Зупинений час гри

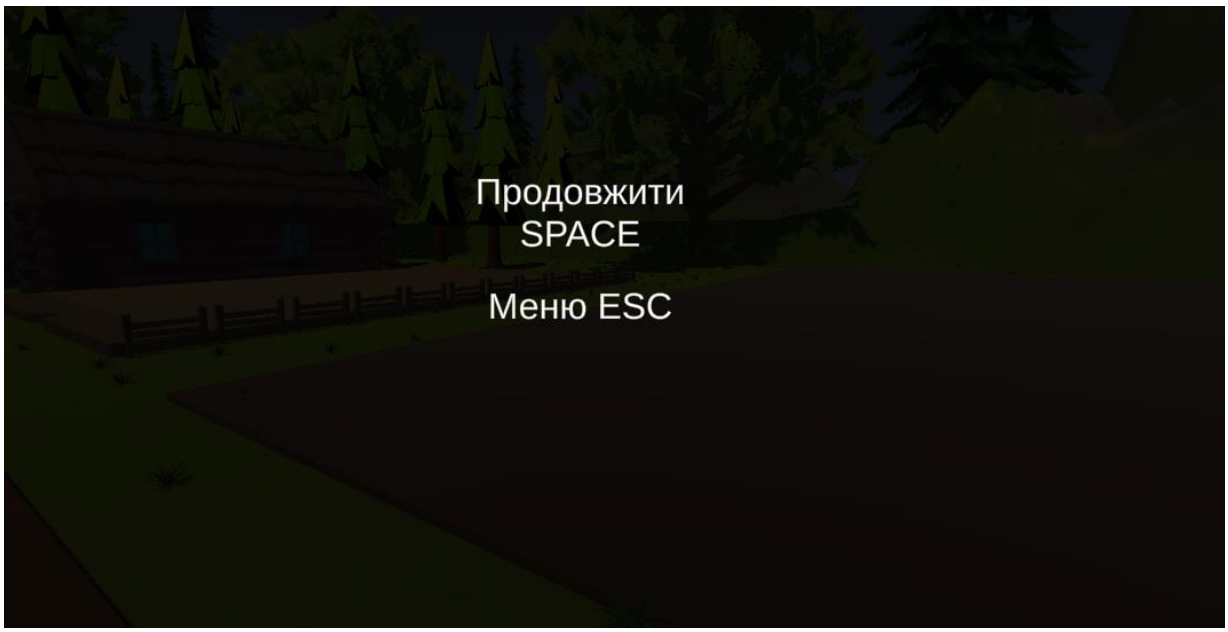


Рисунок 3.16 – Діалогове меню паузи

Тепер перевіримо одну з найважливіших функцій гри – перевірити засадження рослинністю поля. Поле повинно спочатку засаджуватися паростками майбутньої рослини. Поле повинно бути повністю заспавнене моделями саджанців (рисунок 3.17).

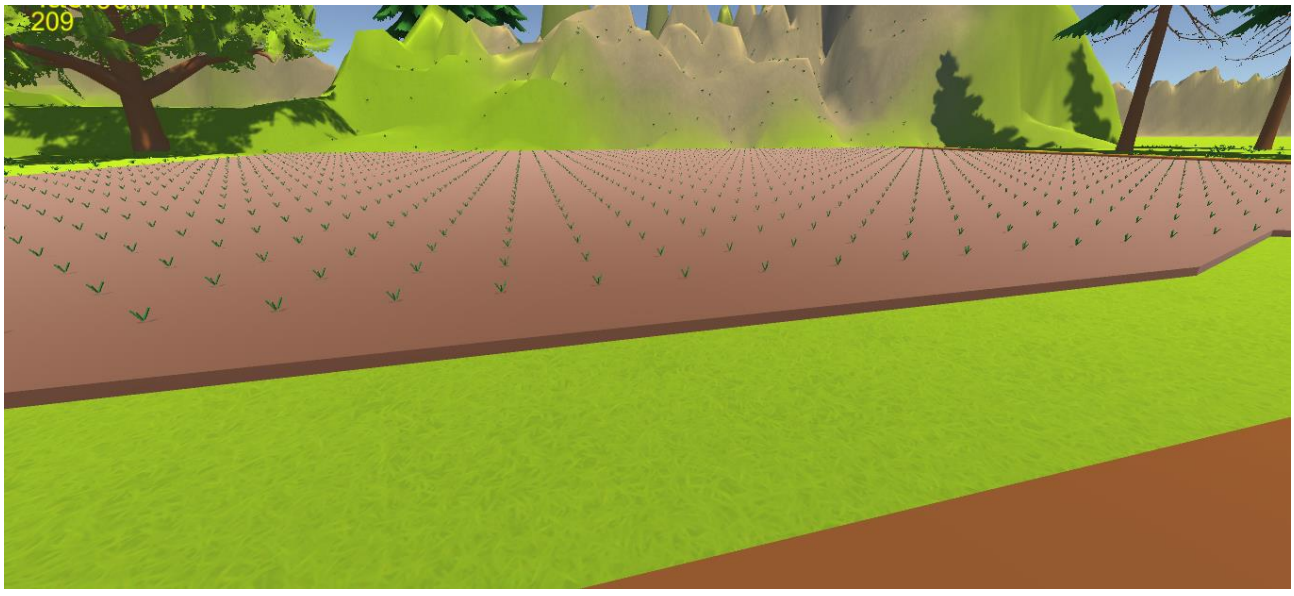


Рисунок 3.17 – Засаджене поле

Після паростків має бути здійснений перехід на дорослу культуру. При цьому старі моделі паростків мають бути видалені та коректно створенні нові матеріали. Дорослі пагони культур показані на рисунку 3.18.

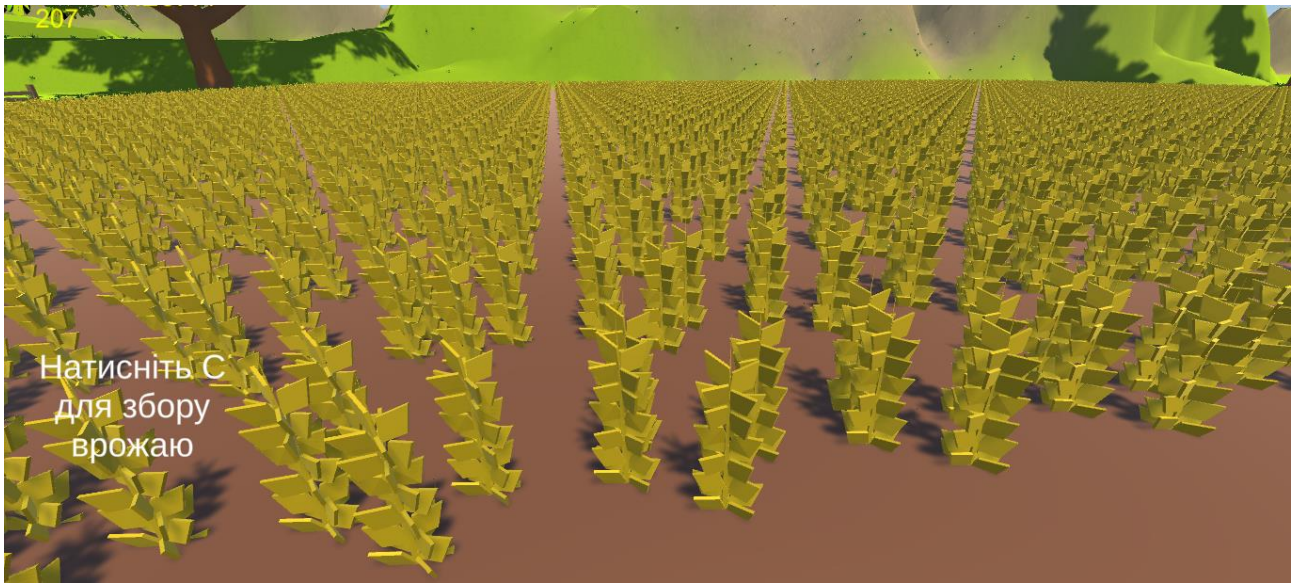
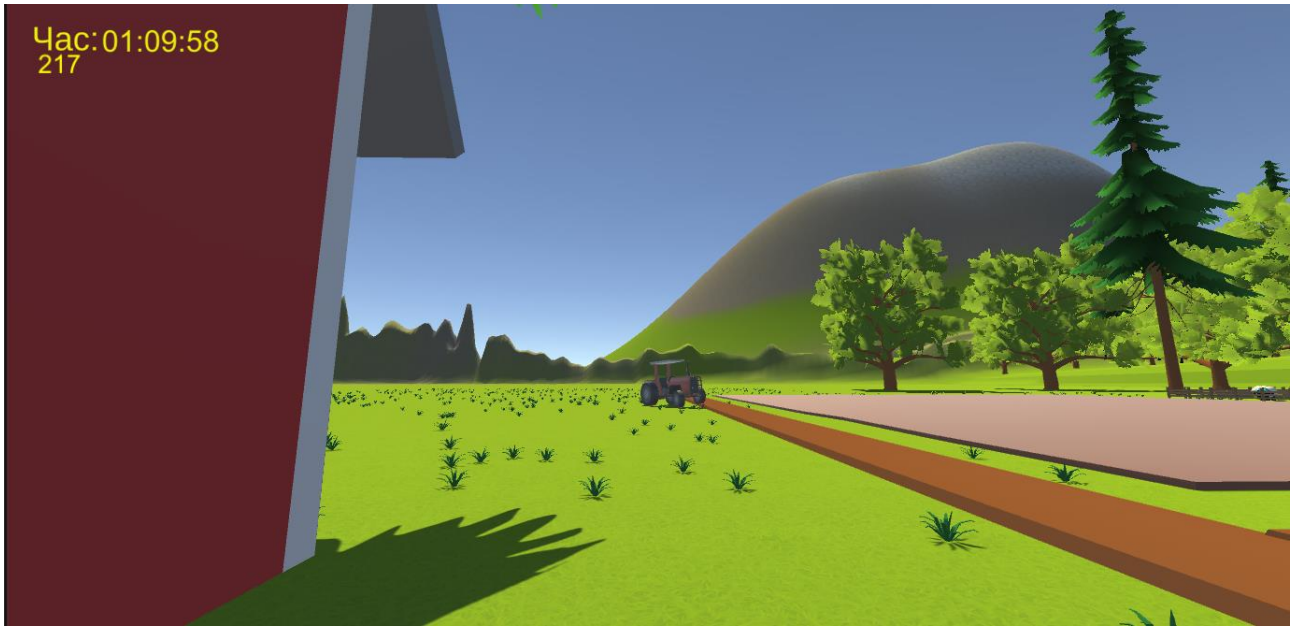


Рисунок 3.18- Створена дозріла культура

Тепер потрібно перевірити збір культури на відповідну клавішу. Кожна модель має бути видалена та поле повинно залишитися повністю пустим. Тестування збору врожаю зображено на рисунку 3.19.

Рисунку 3.19 – Зібране поле

Далі перевіримо зміну керування гравцем та транспортом. Даний функціонал повинен включати та виключати відповідні скрипти керування та змінювати камеру гри. Під час керування транспортом не повинен керуватися персонаж та навпаки – трактор, коли ми керуємо гравцем. Результати тестування зображено на рисунку 3.20.



Рисунку 3.20 – До зміни керування об’єктом

Рисунку 3.21 – Після змінення керування об’єктом

Також важливо зазначити що об’єкт гравця не повинен залишатися на мапі карти.

Після керування транспортом змінна назад на гравця повинна відбуватися в вигляді перенесення гравця до транспортного засобу (рисунок 3.22).

Рисунку 3.22 – Тестування повернення керуванням гравцем

Наступним потрібно перевірити як реагують колайдери коліс при натискані на клавішу повороту (рисунок 3.23).

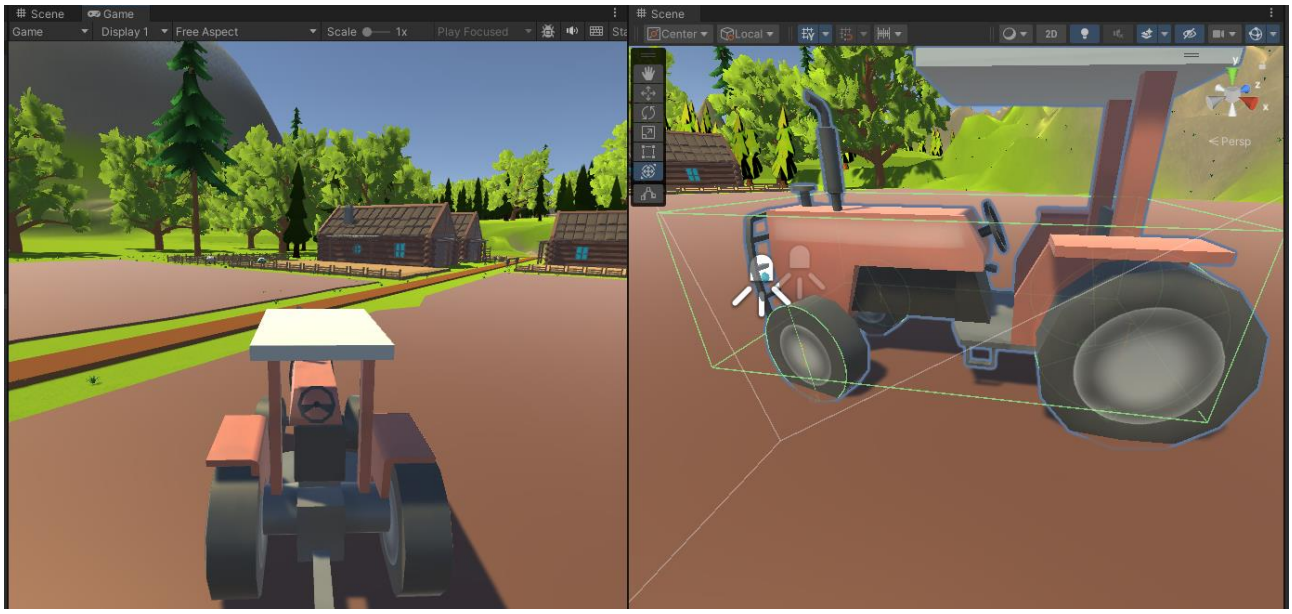


Рисунок 3.23 – Поворот керма

На зображені чітко видно що при повороті колайдери коліс повертаються разом з їхньою текстурою яка відповідно зв’язана.

Наступним кроком стане перевірка збереження позиції гравця та транспорту при виході гравця з гри. Для цього позначимо початкову точку(Рисунок 3.24), точку в якій об’єкти мають зберегти позицію(Рисунок 3.25), точку завантаження об’єктів(Рисунок 3.26).

Рисунок 3.24 – Початкова точка

Рисунок 3.25 – Точка збереження позиції

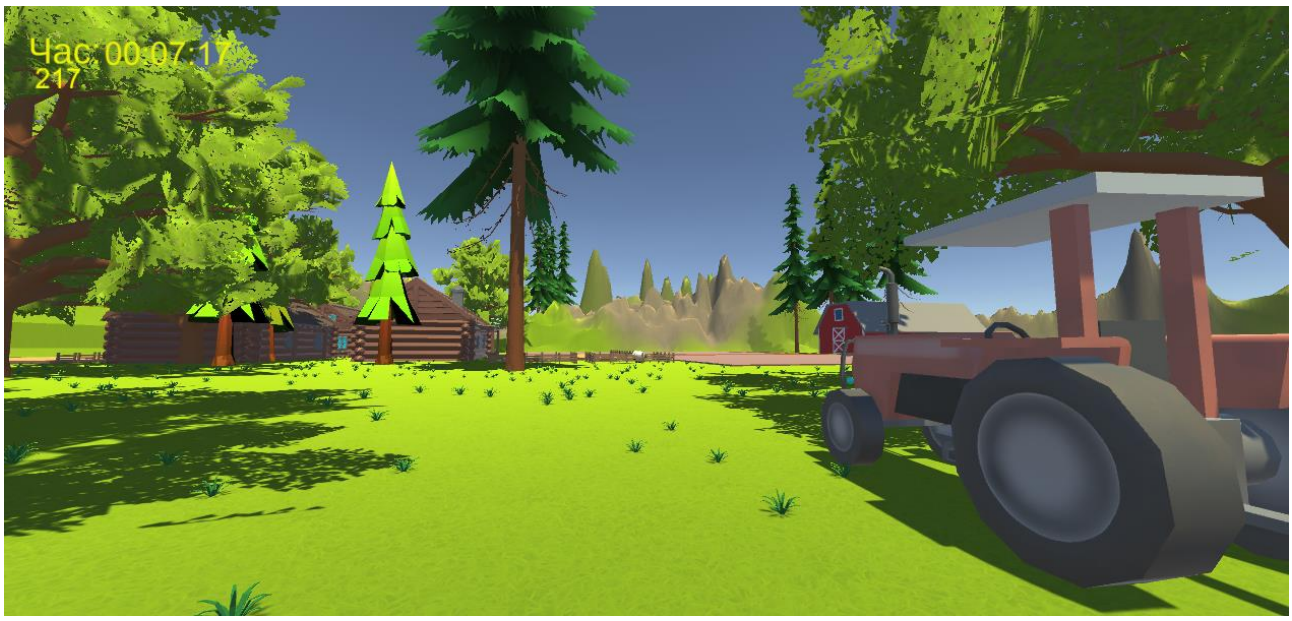


Рисунок 3.26 – Точка відновлення позиції

Наступним переглянемо тестування зміни дня та ночі. Для прикладу час зміни встановимо меншим. Світло повинно включатися на вулиці, в транспорті, та в ангарі.

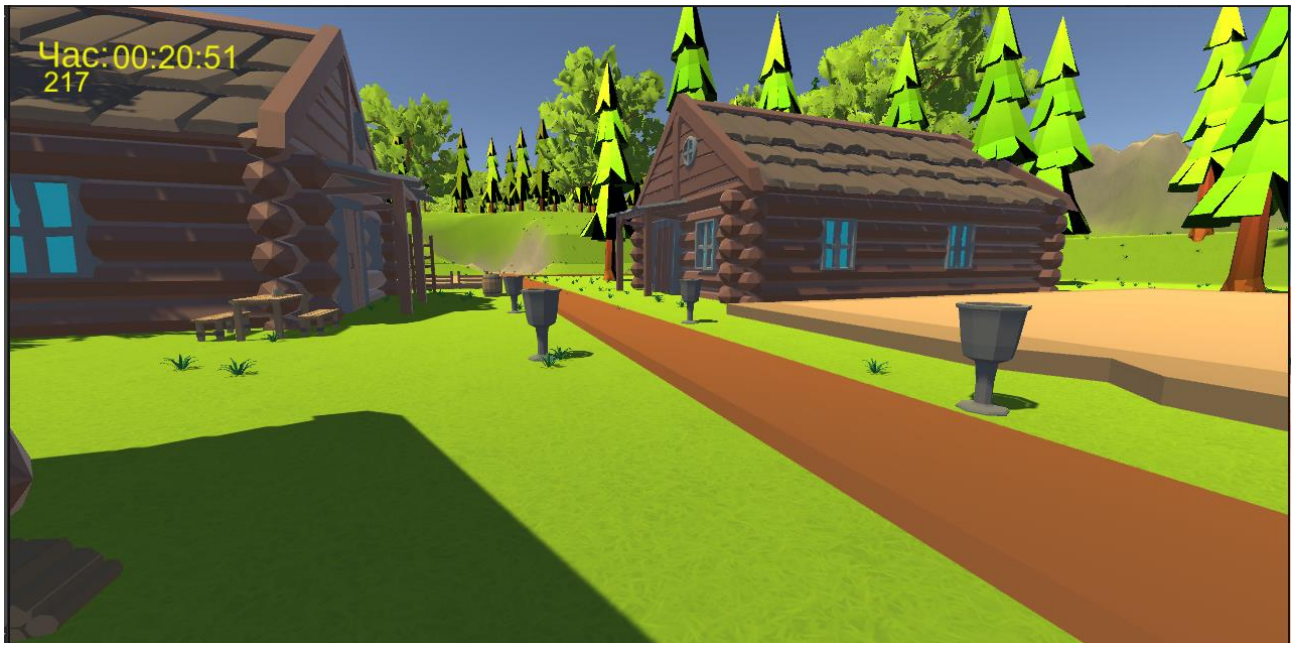


Рисунок 3.27 – Денний варіант освітлення

Як ми можемо побачити вдень елементом освітлення є об’єкт сонця.Тепер розглянемо нічне освітлення (рисунок 3.28).

Рисунок 3.28 – Нічне освітлення

В нічний час освітлення надає ліхтарі біля будинків та на самих будинках присутнє легке світіння.

В результаті тестування гри було успішно досліджено функціонал гри. Отримано позитивні результати в основних та допоміжних системах та було виправленно незначну кількість багів.

4 ТЕХНІКО-ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ

4.1 Аналіз ринку збуту продукту чи послуги.

3D гра про фермерство є відеогрою розроблена спеціально для молодшої аудиторії, яка носить жанр “Sanbox”. Гра реалізована під платформу ПК з використанням яскравої та барвистої приємної графіки. В грі присутній спокійний та приємний навколишній світ. Плюсом для молодшої вікової категорії є відсутність жорстокості та схожих аспектів. Інтуїтивно зрозумілий ігровий процес включає вирощування рослин , догляд за тваринами, організація валютних процесів.

Гра має традиційні аспекти симулятора фермерства з адаптацією до малих дітей, та безпеку.

Основні цільові групи це діти віком від 3 до 10 років, батьки які шукають забезпечення цікавого часу для своїх дітей. Батьки, які шукають безпечні та освітні ігри для своїх дітей, складають основну аудиторію.

Гра може бути реалізована на глобальному ринку, з нахилом на Європу – розвинений ринок з акцентом на освітні продукти для дітей; Азія – Динамічний ринок з великим числом молодих користувачів.

З огляду на популярність мирних і розважальних ігор для дітей, очікується високий попит з боку батьків і можливо освітніх установ. Попит підкріплюється відкритим світом та простим інтерфейсом гри, які розважають дітей.

Організація обслуговування буде відбуватися допродажно та після продажно.

– Допродажне обслуговування це обслуговування та вдосконалення проєкту до початку випуску його в мережу. Виправляється технічні проблеми проєкту.

– Післяпродажне обслуговування включає в себе регулярні оновлення гри, аналіз та усунення помилок гри які були виявленні після випуску, та технічна підтримка проєкту.

Головними конкурентами є:

– Toca Life: Farm: Освітня гра для дітей, що дозволяє керувати фермою. Відома своєю інтерактивністю та безпекою для дітей.

– My Little Farm: Симулятор фермерства для дітей з простим геймплеєм та яскравою графікою.

– Farmville Kids: Дитяча версія відомої гри Farmville, адаптована для молодшої аудиторії.

Оцінюючи життєвий цикл нового продукту, важливо враховувати етапи запуску, зростання, зрілості та занепаду. Очікується, що гра буде у фазі зростання протягом перших двох років і може перейти в стадію зрілості завдяки постійним оновленням і додаткового контенту. Завдяки оновленням і впровадженню нових функцій ви зможете значно продовжити життєвий цикл продукту і забезпечити постійний інтерес дітей і батьків до гри.

4.2 Розрахунок витрат на проєктування

Заробітна плата обчислюється за рівнем професіоналізму працівника, розмірів, умов та важкості виконуваної роботи. На рівень зарплатні впливає швидкість виконання поставленого завдання та рівень його якості виконаної роботи.

Таблиця 4.1 – Кошторис витрат на проєктування

Найменування статей витрат	Сума, грн	Обґрунтування
1. Зарплата проєктувальників	100000	Зарплата розробників, дизайнерів, програмістів та інших фахівців, залучених до проєкту
2. Відрахування на соціальні потреби	20000	Відрахування на соціальні потреби, зокрема, пенсійне страхування, страхування на випадок безробіття та інші соціальні виплати (20% від суми зарплати проєктувальників)
3. Контрагентські роботи і послуги	200000	Залучення зовнішніх фахівців, таких як звукорежисери, художники, тестувальники тощо
4. Витрати на відрядження	50000	Витрати на поїздки, зустрічі та інші виїзні заходи, пов'язані з розробкою гри
5. Інші прямі витрати	100000	Купівля ліцензійного ПЗ, обладнання, витрати на сертифікацію, рекламні матеріали
6. Усього прямих витрат	470000	Загальна сума прямих витрат
7. Накладні витрати	94000	Витрати на утримання офісу, комунальні послуги, амортизація обладнання, адміністративні витрати (20% від суми прямих витрат)
8. Планові накопичення	100000	Резерви для непередбачених витрат та інвестування у подальший розвиток проєкту
9. Усього, кошторисна вартість проєкту	664000	Загальна вартість проєкту
10. Податок на додану вартість	132800	Податок на додану вартість (20% від кошторисної вартості проєкту)
11. Загалом, договірна ціна розробки	796800	Загальна договірна вартість розробки проєкту, включаючи ПДВ

Заробітна плата співробітника визначається виходячи з кількості виконавців, поточного посадового окладу і кількості місяців, витрачених на розробку проєкту.

Для розробки даного ігрового проєкту були залучені наступні спеціалісти:

– Unity Developer.
– Animation Lead (Blender).
– C# Developer.

Розрахунок зарплати продемонстровано в таблиці 4.2.

Таблиця 4.2 – Розрахунок заробітної плати проєктувальників

№	Посада	Оклад	Відрахування	Кількість		Сума
1	Unity Developer	12500	2437	1	2,5 місяці	31250
2	Animation Lead (Blender)	15000	2925	1	2,5 місяці	37500
3	C# Developer	12500	2437	1	2,5 місяці	31250
	Усього зарплати:					100000

Витрати компанії на соціальні заходи виникають в результаті виникнення сум, що залежать від розміру фактичних витрат на оплату праці співробітників, включаючи витрати на основну і додаткову заробітну плату та інші види заохочень і виплат.

Unity Developer:

– Податок на доходи фізичних осіб: 12,500 грн * 18% = 2,250 грн.
– Військовий збір: 12,500 грн * 1.5% = 187.5 грн.
– Єдиний внесок: 12,500 грн * 22% = 2,750 грн.
– Утримання становлять: 2,250 грн + 187.5 грн = 2,437.5 грн.
– До виплати працівникові: 12,500 грн - 2,437.5 грн = 10,062.5 грн.

Animation Lead (Blender):

– Податок на доходи фізичних осіб: 15,000 грн * 18% = 2,700 грн.

– Військовий збір: 15,000 грн * 1.5% = 225 грн.

– Єдиний внесок: 15,000 грн * 22% = 3,300 грн.

– Утримання становлять: 2,700 грн + 225 грн = 2,925 грн.

– До виплати працівникові: 15,000 грн - 2,925 грн = 12,075 грн.

C# Developer:

– Податок на доходи фізичних осіб: 12,500 грн * 18% = 2,250 грн.

– Військовий збір: 12,500 грн * 1.5% = 187.5 грн.

– Єдиний внесок: 12,500 грн * 22% = 2,750 грн.

– Утримання становлять: 2,250 грн + 187.5 грн = 2,437.5 грн.

– До виплати працівникові: 12,500 грн - 2,437.5 грн = 10,062.5 грн.

4.3 Обґрунтування необхідності розробки

Даний ігровий проєкт був розроблений для дітей молодшого віку (від 3 до 10 років). Застосунок може приносити вигоду компанії або організації за допомогою системи реклами. Використовуючи лише безпечний вид реклами для дитячого віку. Наприклад реклама іграшок, навчальних матеріалів або інших дитячих товарів.

Вважаючи що сучасні діти проводять багато часу за різними гаджетами, то можна зробити висновок що проєкт, який не містить в собі жорстоких алементів, або інших елементів які б могли погано повпливати на думку дити, стане популярним серед користувачів.

Гра побудування на популярному та доволі хорошому ігровому рушії – Unity Engine 6 Preview . Це дає змогу створити сучасну та оптимізовану гру, з підтримкою проєкту в подальшому. В свою чергу це збільшує популярність застосунку.

Ціль даного проєкту, випустити таку гру яка б була приємною та нешкідливою, з бажанням досліджувати віртуальний світ, та виконувати завдання.

ВИСНОВКИ

Метою кваліфікаційної роботи була розробка ігрового додатку в жанрі sandbox з хорошою оптимізацією, приємною графічною складовою, яка повинна забезпечити гравцям відкритий світ, творчу свободу, ефективну взаємодію, мотивацію до дослідження та збалансований ігровий процес. Ці елементи сприяють створенню захоплюючого та тривалого ігрового досвіду.

В проєкті гравець керує фермою, вирощує рослини, тварин та заробляє умовні «кошти» працюючи з ними.

Під час виконання кваліфікаційної роботи було:

– проаналізовано ігрові продукти інших розробників у даному жанрі;

– досліджено різні типи ігрових елементів;

– створено моделі та текстури в графічному редакторі;

– реалізовано фізичні властивості гравця, тварин, транспорту;

– реалізовано взаємодію між об’єктами в ігровому світі через середовище Unity та програмний код на мові C#;

– проведено тестування та виправлено помилки.

Проєкт був реалізований згідно поставлених вимог. Застосунок буде допрацьовуватися, відбудеться розширення функціоналу гри, вдосконалення графічної частини, розширення навколишнього середовища та його деталізація, зокрема буде розширено список тварин, рослин та транспорту, додано більше знарядь праці. Також планується збільшити кількість мап, розробити онлайн режим. Загалом, проєкт є основою для розробки більш масштабного продукту в даному жанрі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. NSDG. Створення руху гравця в Unity. NSDG : Вебсайт. URL : https://uk.sharpcoderblog.com/blog/creating-player-movement-in-unity (дата звернення: 02.05.2024).

2. UnityDocumentation. Unity User Manual 2022.3 (LTS). Вебсайт. URL : https://docs.unity3d.com/Manual/index.html (дата звернення: 05.05.2024).

3. Unity - GameObjects. Unity - Manual: Unity User Manual 2022.3 (LTS). Вебсайт. URL : https://docs.unity3d.com/ru/530/Manual/ControllingGameObjectsComponents.html (дата звернення: 19.05.2024).

4. Unity Asset Store. Unity Asset Store - The Best Assets for Game Making. Вебсайт.URL : https://assetstore.unity.com/ (дата звернення: 25.05.2024).

5. Poly Pizza. Free 3D models for everyone. Вебсайт. URL : https://poly.pizza/ (дата звернення: 19.05.2024).

6. Unity - Теги і слої. Unity - Manual: Unity User Manual 2022.3 (LTS). Вебсайт. URL: https://docs.unity3d.com/ru/530/Manual/class-TagManager.html (дата звернення: 28.05.2024).

7. SlideShare. Приклади реалізації алгоритмів управління в середовищі UNITY PRO. Вебсайт. URL: https://www.slideshare.net/slideshow/unity-pro-67934067/67934067 (дата звернення: 30.05.2024).

ДОДАТКИ

Додаток А

Приклад коду об’єкту “Field” у середовищі Visual Studio

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;

public class SingleCropFieldManager : MonoBehaviour
{
    private int coin;
    public Text coinCount;
    public GameObject wheatPrefab; // Префаб моделі пшениці
    public GameObject sproutPrefab; // Префаб паростка пшениці
    public GameObject beetPrefab; // Префаб буряка
    public GameObject carrotPrefab; // Префаб моркви
    public GameObject tomatoPrefab; // Префаб помідорів
    public int width; // Ширина області пшениці
    public int height; // Висота області пшениці
    public GameObject messageCanvas2; // Канвас для взаємодії з полем
    public GameObject harvestCanvas; // Канвас для повідомлення про
збирання урожаю
    private GameObject[,] field; // Поле, де кожен елемент є "слотом"
для пшениці
    private bool playerInRange = false; // Змінна для відстеження чи
гравець у зоні
    private bool wheatGrown = false; // Прапорець для відстеження, чи
зійшла пшениця
    private GameObject currentCropPrefab; // Поточний префаб культури

    // Define the cost of each crop
    public int beetCost = 5;
    public  int carrotCost = 3;
    public  int tomatoCost = 4;
    public int wheatCost = 2;

    void Start()
    {
        // Отримання кількості монет з PlayerPrefs
        coin = PlayerPrefs.GetInt("coins");

        // Перевірка, чи прив'язані всі необхідні об'єкти
        if (coinCount == null)
        {
            Debug.LogError("Coin Count Text is not assigned in the
inspector.");
            return;
        }

        if (messageCanvas2 == null)
        {
            Debug.LogError("Message Canvas 2 is not assigned in the
inspector.");
```

```
            return;
        }

        if (harvestCanvas == null)
        {
            Debug.LogError("Harvest Canvas is not assigned in the
inspector.");
            return;
        }

        // Прив'язка тексту кількості монет
        coinCount.text = coin.ToString();

        // Ініціалізація поля
        field = new GameObject[width, height];

        // Приховуємо канваси спочатку
        messageCanvas2.SetActive(false);
        harvestCanvas.SetActive(false);
    }

    void Update()
    {
        if (playerInRange && Input.GetKeyDown(KeyCode.F1) &&
IsFieldEmpty() && !harvestCanvas.activeSelf)
        {
            if (coin >= beetCost)
            {
                currentCropPrefab = beetPrefab;
                DeductCoins(beetCost);
                StartCoroutine(SpawnSprouts());
            }
        }

        if (playerInRange && Input.GetKeyDown(KeyCode.F2) &&
IsFieldEmpty() && !harvestCanvas.activeSelf)
        {
            if (coin >= carrotCost)
            {
                currentCropPrefab = carrotPrefab;
                DeductCoins(carrotCost);
                StartCoroutine(SpawnSprouts());
            }
        }

        if (playerInRange && Input.GetKeyDown(KeyCode.F3) &&
IsFieldEmpty() && !harvestCanvas.activeSelf)
        {
            if (coin >= tomatoCost)
            {
                currentCropPrefab = tomatoPrefab;
                DeductCoins(tomatoCost);
                StartCoroutine(SpawnSprouts());
            }
        }
```

```
        if (playerInRange && Input.GetKeyDown(KeyCode.F4) &&
IsFieldEmpty() && !harvestCanvas.activeSelf)
        {
            if (coin >= wheatCost)
            {
                currentCropPrefab = wheatPrefab;
                DeductCoins(wheatCost);
                StartCoroutine(SpawnSprouts());
            }
        }

        if (playerInRange && Input.GetKeyDown(KeyCode.C) &&
!IsFieldEmpty() && harvestCanvas.activeSelf)
        {
            RemoveWheat();
        }
    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Field"))
        {
            playerInRange = true;
            bool isFieldEmpty = IsFieldEmpty(); // Перевіряємо, чи
поле порожнє

            if (isFieldEmpty)
            {
                messageCanvas2.SetActive(true); // Показуємо канвас
тільки, якщо поле порожнє
            }
            else
            {
                messageCanvas2.SetActive(false); // Приховуємо канвас,
якщо є паростки або пшениця
                if (wheatGrown)
                {
                    harvestCanvas.SetActive(true); // Показуємо канвас
з повідомленням про збирання, якщо є пшениця
                }
            }
        }
    }

    private void OnTriggerExit(Collider other)
    {
        if (other.CompareTag("Field"))
        {
            playerInRange = false;
            messageCanvas2.SetActive(false); // Приховуємо канвас
            harvestCanvas.SetActive(false); // Приховуємо канвас
        }
    }

    private IEnumerator SpawnSprouts()
    {
```

```
        messageCanvas2.SetActive(false); // Приховуємо канвас з
повідомленням про спавн

        if (playerInRange) // Перевіряємо, чи гравець на полі
        {
            Vector3 startPosition = new Vector3(transform.position.x -
width / 2, transform.position.y, transform.position.z - height / 2);
            for (int x = 0; x < width; x++)
            {
                for (int y = 0; y < height; y++)
                {
                    Vector3 spawnPosition = new
Vector3(startPosition.x + x, GetHeightAtPosition(startPosition.x + x,
startPosition.z + y), startPosition.z + y);
                    field[x, y] = Instantiate(sproutPrefab,
spawnPosition, Quaternion.identity);
                }
            }

            yield return new WaitForSeconds(10f); // Зачекати 10
секунд

            for (int x = 0; x < width; x++)
            {
                for (int y = 0; y < height; y++)
                {
                    if (field[x, y] != null)
                    {
                        Destroy(field[x, y]);
                        Vector3 spawnPosition = new
Vector3(startPosition.x + x, GetHeightAtPosition(startPosition.x + x,
startPosition.z + y), startPosition.z + y);
                        field[x, y] = Instantiate(currentCropPrefab,
spawnPosition, Quaternion.identity);
                    }
                }
            }

            wheatGrown = true; // Встановлюємо прапорець, що зійшла
пшениця
            if (playerInRange)
            {
                harvestCanvas.SetActive(true); // Показуємо канвас з
повідомленням про збирання тільки, коли гравець на полі
            }
        }
    }

    private void RemoveWheat()
    {
        for (int x = 0; x < width; x++)
        {
            for (int y = 0; y < height; y++)
            {
                if (field[x, y] != null)
                {
                    Destroy(field[x, y]);
```

```
                    field[x, y] = null;
                }
            }
        }
        coin = PlayerPrefs.GetInt("coins"); // призначення зміної
        PlayerPrefs.SetInt("coins", coin + 10);
        coinCount.text = (coin + 10).ToString(); // додавання грошей в
пам'ять
        wheatGrown = false; // Reset wheatGrown flag
        harvestCanvas.SetActive(false); // Приховуємо канвас з
повідомленням про збирання

        if (playerInRange)
        {
            messageCanvas2.SetActive(true); // Показуємо канвас з
повідомленням про взаємодію, якщо гравець ще в зоні
        }
    }

    private float GetHeightAtPosition(float x, float z)
    {
        // Отримання висоти на певній позиції через колайдер
        RaycastHit hit;
        if (Physics.Raycast(new Vector3(x, 10000f, z), Vector3.down,
out hit, Mathf.Infinity))
        {
            return hit.point.y;
        }
        else
        {
            Debug.LogError("No collider found at position: " + new
Vector3(x, 0f, z));
            return 0f;
        }
    }
    private bool IsFieldEmpty()
    {
        // Перевіряємо, чи всі елементи поля порожні
        for (int x = 0; x < width; x++)
        {
            for (int y = 0; y < height; y++)
            {
                if (field[x, y] != null)
                {
                    return false;
                }
            }
        }
        return true;
    }
    private void DeductCoins(int amount)
    {
        coin -= amount;
        PlayerPrefs.SetInt("coins", coin);
        coinCount.text = coin.ToString();
    }
}
```

Додаток Б

Програмний код елементу “transport” у середовищі Visual Studio

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class TractorMovemant : MonoBehaviour
{


    [Header("Tractor Object Transport")]
    public Transform objectToTransport; // Об'єкт, який будемо
переносити
    public float transportDistance = 2f;



    public Camera mainCamera; // Reference to the main camera
    public Camera alternateCamera; // Reference to the alternate
camera
    public MovemantPlayer movementPlayer; // Reference to the player
movement script

    public Rigidbody rb;

    [Header("Controls")]
    public float Accel;
    public float Brakes;
    public float Steering;

    [Header("Vehicle Settings")]
    public float EnginePower = 1000f;
    public float BrakeForce = 15000f;
    public float SteerAngle = 35f;

    [Header("Wheels")]
    public WheelCollider[] FrontWheels;
    public WheelCollider[] RearWheels;
    public Transform[] FrontWheelMeshes;
    public Transform[] RearWheelMeshes;

    public Vector3 COM;
    void SaveTractorPosition()
    {
        PlayerPrefs.SetFloat("TractorPosX", transform.position.x);
        PlayerPrefs.SetFloat("TractorPosY", transform.position.y);
        PlayerPrefs.SetFloat("TractorPosZ", transform.position.z);
        Debug.Log("Tractor position saved: " + transform.position);
    }

    void LoadTractorPosition()
    {
        float tractorPosX = PlayerPrefs.GetFloat("TractorPosX", 0f);
        float tractorPosY = PlayerPrefs.GetFloat("TractorPosY", 0f);
        float tractorPosZ = PlayerPrefs.GetFloat("TractorPosZ", 0f);
```

```
        Debug.Log("Tractor position loaded: " + tractorPosX + ", " +
tractorPosY + ", " + tractorPosZ);
        transform.position = new Vector3(tractorPosX, tractorPosY,
tractorPosZ);
    }

    void OnDestroy()
    {
        SaveTractorPosition();
    }

    void OnApplicationQuit()
    {
        SaveTractorPosition();
    }

    void Start()
    {
        LoadTractorPosition();
        rb = GetComponent<Rigidbody>();
        rb.centerOfMass = COM;


    }
    public float desiredHeightAboveTractor = 2f; // Величина зміщення
вгору для спавну об'єкту

    void Update()
    {


        rb.isKinematic = false;

        // Toggle scripts and cameras on Enter key press
        if (Input.GetKeyDown(KeyCode.Return))
        {
            if (objectToTransport != null)
            {
                objectToTransport.gameObject.SetActive(true);
                // Визначаємо позицію, куди треба перемістити об'єкт
                Vector3 targetPosition = transform.position +
transform.up * desiredHeightAboveTractor;

                // Переміщуємо об'єкт до цієї позиції
                objectToTransport.position = targetPosition;
            }
            foreach (WheelCollider wheel in FrontWheels)
            {
                wheel.motorTorque = 0;
            }

            foreach (WheelCollider wheel in RearWheels)
            {
                wheel.motorTorque = 0;
            }
            rb.isKinematic = true;
            ToggleScripts();
```

```
            ToggleCameras();
        }

        Accel = Input.GetAxis("Vertical"); // Use the vertical axis
for forward/reverse
        Brakes = Input.GetAxis("Jump");    // Use the jump axis
(typically space bar) for brakes
        Steering = Input.GetAxis("Horizontal");

        // Apply motor torque to front and rear wheels
        foreach (WheelCollider wheel in FrontWheels)
        {
            wheel.motorTorque = EnginePower * Accel;
        }

        foreach (WheelCollider wheel in RearWheels)
        {
            wheel.motorTorque = EnginePower * Accel;
        }

        // Apply brake force
        if (Brakes > 0)
        {
            foreach (WheelCollider wheel in FrontWheels)
            {
                wheel.brakeTorque = BrakeForce * Brakes;
            }
            foreach (WheelCollider wheel in RearWheels)
            {
                wheel.brakeTorque = BrakeForce * Brakes;
            }
        }
        else
        {
            foreach (WheelCollider wheel in FrontWheels)
            {
                wheel.brakeTorque = 0;
            }
            foreach (WheelCollider wheel in RearWheels)
            {
                wheel.brakeTorque = 0;
            }
        }

        // Apply steering angle to front wheels
        foreach (WheelCollider wheel in FrontWheels)
        {
            wheel.steerAngle = SteerAngle * Steering;
        }

        // Update wheel meshes position and rotation
        UpdateWheelMeshes();
    }

    void UpdateWheelMeshes()
    {
        for (int i = 0; i < FrontWheels.Length; i++)
```

```
        {
            UpdateWheelMesh(FrontWheels[i], FrontWheelMeshes[i]);
        }
        for (int i = 0; i < RearWheels.Length; i++)
        {
            UpdateWheelMesh(RearWheels[i], RearWheelMeshes[i]);
        }
    }

    void UpdateWheelMesh(WheelCollider collider, Transform mesh)
    {
        Vector3 position;
        Quaternion rotation;
        collider.GetWorldPose(out position, out rotation);
        mesh.position = position;
        mesh.rotation = rotation;
    }

    void ToggleScripts()
    {
        if (movementPlayer != null)
        {
            // Toggle the activation of the scripts
            bool isPlayerScriptEnabled = this.enabled;
            this.enabled = !isPlayerScriptEnabled;
            movementPlayer.enabled = isPlayerScriptEnabled;
        }
        else
        {
            Debug.LogWarning("TractorMovemant script is not assigned
in the inspector.");
        }
    }

    void ToggleCameras()
    {
        if (mainCamera != null && alternateCamera != null)
        {
            mainCamera.enabled = !mainCamera.enabled;
            alternateCamera.enabled = !alternateCamera.enabled;
        }
        else
        {
            Debug.LogWarning("Cameras are not assigned in the
inspector.");
        }
    }
}
```

Додаток В

Таблиця 4.3 “Посадові оклади”

ПОСАДА	Розряд	Ктар
C# Developer	15	2,58
Animation Lead	14	2,42
BluePrint Dev, Unreal Engine Lead	12	2,12