Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

# КВАЛІФІКАЦІЙНА РОБОТА
# Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

**РОЗРОБКА МОБІЛЬНОЇ ГРИ НА UNITY З ВИКОРИСТАННЯМ SHADER GRAPH ДЛЯ СТВОРЕННЯ ВІЗУАЛЬНИХ ЕФЕКТІВ**
(тема)

Виконав:
студент 4 курсу, групи ІТІНФ-20-1
Несвітайло Д.А.
(прізвище, ініціали)

Спеціальності 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник доц. Шафроненко А.Ю.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Кобилін О.А.
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра Інформатики
(повна назва)

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2024 р.

# ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Несвітайлу Данилу Андрійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка мобільної гри на Unity з використанням Shader Graph для створення візуальних ефектів

затверджена наказом університету від 20 травня 2024 року № 464 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 1 червня 2024 р.

3. Вихідні дані до роботи науково-методична та науково-технічна література з програмування на Unity, дані інтернет-мереж, бібліотеки з GitHub, двигун для розробки ігор Unity, додаток для створення шейдерів Shadeg Graph

4. Перелік питань, що потрібно опрацювати в роботі
1. Огляд сучасної ігрової індустрії та дослідження ресурсів розробок гри.
2. Вибір формату гри та ігрового дизайну.
3. Аналіз існуючих можливостей розробок гри.
4. Опис процесу розробки гри.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) Актуальність проблеми розробки ігор, постановка задачі, тестові зображення.

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

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

**КАЛЕНДАРНИЙ ПЛАН**

| № з/п | Назва етапів роботи | Терміни виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Отримання завдання на кваліфікаційну роботу | 08.04.2024 | |
| 2 | Аналіз завдання, підбір літератури | 08.04.24-15.04.24 | |
| 3 | Аналіз літератури з досліджуваної проблеми | 16.04.24-18.04.24 | |
| 4 | Аналіз технічних засобів та ігрової індустрії | 19.04.24-25.04.24 | |
| 5 | Розробка методу | 26.04.24-14.05.24 | |
| 6 | Програмна реалізація | 15.05.24-23.05.24 | |
| 7 | Оформлення пояснювальної записки | 24.05.24-26.05.24 | |
| 8 | Перевірка на плагіат | 27.05.24 | |
| 9 | Рецензування | 28.05.24 | |
| 10 | Підготовка презентації та доповіді | 29.05.24-02.06.24 | |
| 11 | Занесення роботи в електронний архів | 05.06.24 | |
| 12 | Попередній захист кваліфікаційної роботи | 12.06.24 | |

Дата видачі завдання 8 квітня 2024 р.

Студент _____
(підпис)

Керівник роботи _____ _____доц. Шафроненко А.Ю.
(підпис) (посада, прізвище, ініціали)

# РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи 54 с., 2 табл., 26 рис., 25 джерело.

РОЗРОБКА ІГОР НА UNITY 3D, РОЗРОБКА ІГОР НА ANDROID, РОЗРОБКА INDI ІГОР, ВИКОРИСТАННЯ SHADER GRAPH .

Об’єктом роботи є Unity, програма для розробки ігор, а також її додатковий інтегрований елемент для створення шейдерів Shader Graph.

Метою роботи є створення діючою 2d ігри, орієнтованої під платформи Android, розробленої на Unity з використанням Shader Graph.

В роботі було проаналізовано ігрові тенденції у світі, зроблено аналіз існуючих ігор-прообразів, розроблено дизайн гри, вибрано відповідне програмне забезпечення для створення відповідної гри, прописано дизайн гри та приведено етапи її створення.

У результаті роботи було створено актуальну гру, відповідну до дизайну розробника.

GAME DEVELOPMENT ON UNITY 3D, GAME DEVELOPMENT ON ANDROID, INDI GAME DEVELOPMENT, USING SHADER GRAPH.

The object of work is Unity, a game development program, as well as its additional integrated element for creating shaders Shader Graph.

The purpose of the work is to create a working 2d game, oriented for Android platforms, developed on Unity using Shader Graph.

The work analyzes gaming trends in the world, analyzes existing prototype games, develops the game design, selects the appropriate software for creating the game, prescribes the game design and presents the stages of its creation.

As a result of the work, an actual game was created, corresponding to the developer's design.

# ЗМІСТ

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПК –персональний комп’ютер

VR – Virtual reality (віртуальна реальніст)

AR – Additional reality (доповнена реальність)

FPS – Frame per second (кількість кадрів в секунду)

# ВСТУП

У світі розробка ігор займає центральне місце серед індустрій розваг і технологічних галузей. Зі зростанням доступності високошвидкісного інтернету, потужних графічних рушіїв та мобільних пристроїв, ігри стали невід'ємною частиною повсякденного життя мільйонів людей по всьому світу. Цей сектор, що динамічно розвивається, приваблює талановитих розробників і творчих індивідуумів, які бажають втілити свої ідеї у віртуальній реальності. Розробка ігор пропонує унікальні можливості для інновацій та експериментів у галузі геймдизайну, штучного інтелекту, віртуальної та доповненої реальності. Соціальні та мультиплеєрні ігри створюють нові способи взаємодії та спілкування, а ігрові рушії, такі як Unity, надають розробникам потужні інструменти для втілення своїх ідей у життя.

Додаток Unity є професійним ігровим рушієм, який використовується у створенні відеоігор для різних платформ [1]. Це актуальне середовище для розробок ігор, яке визнається у всьому світі. За даними Unity, платформу використовують понад 6 мільйонів зареєстрованих розробників, а 770 мільйонів геймерів насолоджуються іграми, створеними Unity [3].

Однак ця величезна популярність має і негативний побічний ефект: поява явно загального зовнішнього вигляду Unity серед багатьох ігор, створених з використанням цього рушія. Цей «зовнішній вигляд Unity» є результатом того, що багато нових розробників використовують у своїх іграх ті самі матеріали, ресурси, світлові ефекти та шейдери Unity Engine за умовчанням. Віддані художники та програмісти, звичайно ж, можуть додати свій власний друк до проекта Unity, написавши свої власні ресурси та шейдери, щоб надати своєму проекту більш унікальної естетики. Але багато дрібних розробників не можуть або не хочуть докладати зусиль, щоб справді вивести свою гру за рамки «загального» виду, який може характеризувати проекти Unity [4].

Особливо актуальною ця проблема стала розробки мобільних ігор. У зв’язку з обмеженими параметрами мобільних пристроїв вимоги до графічних елементів ігор більш суворі, ніж до персональних комп’ютерів. У зв’язку з чим альтернатив Unity Engine у невеликих інді-компаній небагато.

Однією з таких альтернатив є Shader Graph. Shader Graph це новий інструмент для створення шейдерів у unity [6]. Shader Graph дозволяє візуально створювати шейдери та бачити результати в режимі реального часу. Ця система на основі вузлів відкриває поле для художників та інших членів команди [5].

У цій роботі було аргументовано актуальність ігрової індустрії у сучасному світі, проаналізовано її основні тенденції та закономірноті. Було описано процес створення гри для мобільного пристрою, на рушії Unity 3D і додатковим інструментом у вигляді Shader Graph. Проаналізовано альтернативи для використання інших графічних редакторів, описано ефективність найбільш популярних 3D рушіїв. Було детально описано усі аспекти розробки, включаючи backend та frontend розробку, опис логіки гри та її дизайну, проаналізовано декілька ігор з подібними механіками та жанрами, проведено обробку та моделювання, аналіз звуку та відео, coding та багато іншого.

# 1 АНАЛІЗ МОЖЛИВОСТЕЙ ІГРОВОЇ ІНДУСТРІЇ

## 1.1 Значення комп’ютерних ігор

Ігри, як частина людської культури, пройшли довгий шлях від простих настільних і карткових розваг до складних і технологічно просунутих відеоігор. У сучасному світі ігри відіграють важливу роль не тільки як форма дозвілля, а й як значущий соціальний, економічний і культурний феномен. Вони стали інструментом для навчання та розвитку, способом комунікації та соціальної інтеграції, а також потужним рушієм технологічного прогресу та інновацій.

Ігри впливають на безліч аспектів життя сучасної людини: вони сприяють розвитку когнітивних і моторних навичок, навчають стратегічного мислення та командної роботи, а також надають платформу для самовираження і творчості. Вони сприяють розвитку моторіки, аналітичного апарату та уважності дітей [22].

Крім того, ігри стали важливим інструментом у галузі освіти та професійної підготовки, даючи змогу створювати інтерактивні та мотивуючі навчальні середовища. Віртуальна і доповнена реальність, що застосовуються в іграх, відкривають нові можливості для медицини, архітектури, інженерії та багатьох інших галузей [23].

## 1.2 Ігрова індустрія та її тенденції

Ігрова індустрія сьогодні є одним із секторів світової економіки, що найдинамічніше розвивається, залучаючи мільйони користувачів і генеруючи багатомільярдні доходи [24]. У всьому світі відеоігри формують величезний ринок. Завдяки традиційним точкам доступу, таким як консолі, ПК та портативні пристрої, а також нескінченним додаткам, до яких можна

доторкнутися одним пальцем на телефоні, відеоігри вже кілька років поспіль оцінюються у майже 200 мільярдів доларів [24].

При цьому, не дивлячись на різноманітність платформ, я хочу звернути увагу на саме мобільні ігри, тому що вважаю їх перспективнішими.

Ринок мобільних ігор вже більший за ринки консолей та ПК разом узяті, і на нього припадає майже 57% світового доходу від відеоігор у 2021 році. Зростаюча зрілість стрімінгу, хмарних ігрових сервісів та мобільного кіберспорту – в поєднанні з тим, що мобільні платформи близькі до технічного паритету з ПК та консолями – означає, що більшість геймерів охоплять мобільні ігри в найближчі кілька років [26]. Тому, розробка ігор, орієнтованих на мобільні пристрої є актуальною та ефективною.

Традиційні видавці ігор для консолей та ПК, такі як Activision Blizzard, Sony та Electronic Arts, все більше уваги приділяють новому сегменту ринку через його популярність серед геймерів по всьому світу та можливості отримання прибутків [26].

Вони конкурують з такими компаніями, як Tencent, Sea та Perfect World, які в першу чергу зосереджені на мобільних іграх. Платформи соціальних мереж, такі як ByteDance, онлайн-рекламодавці, такі як AppLovin, та сервіси потокового відео, такі як Netflix, також переходять до видавництва мобільних ігор [26].

Зростання доступності мереж 5G у найближчі роки спонукатиме все більше користувачів до мобільних ігор, особливо багатокористувацьких. Це, в свою чергу, сприятиме зростанню мобільного кіберспорту [26].

## 1.3 Реалізація ігр інді-студіями

Створення ігор інді компаніями являє собою унікальний і захоплюючий процес, який багато в чому відрізняється від розробки ігор великими студіями. Інді-розробка часто асоціюється з меншими бюджетами

та командами, але саме ці обмеження нерідко стимулюють креативність і нестандартне мислення.

Інді-компанії, як правило, складаються з невеликих груп ентузіастів, які об’єднуються навколо спільної ідеї та пристрасті до ігор. Вони можуть працювати зі своїх будинків або невеликих офісів, використовуючи доступні ресурси і технології, щоб створити продукт, який здатний здивувати і надихнути гравців по всьому світу. На відміну від великих студій, інді-розробники мають свободу експериментувати, приймати ризики та втілювати в життя проєкти, які, можливо, ніколи не отримали б зеленого світла в умовах корпоративного середовища.

Створення ігор інді-компаніями може бути прибутковою справою. Незважаючи на обмежені ресурси і невеликі бюджети, інді-розробники часто доводять, що успіх на ринку відеоігор не обов'язково пов’язаний з великими вкладеннями. Багато інді-ігор, створених невеликими командами або навіть окремими розробниками, стають справжніми хітами і приносять значні доходи своїм творцям.

Один із яскравих прикладів такого успіху – гра Minecraft, розроблена шведським програмістом Маркусом Перссоном. Спочатку створена як хобі-проект, гра швидко завоювала популярність завдяки своїй унікальній концепції та свободі творчості, що надається гравцям. Згодом проєкт був придбаний корпорацією Microsoft за кілька мільярдів доларів, що наочно демонструє, як невеликий інді-проєкт може перетворитися на глобальний феномен.

Іншим прикладом є гра Undertale, створена практично поодинці розробником Тобі Фоксом. Фінансована через платформу Kickstarter, гра зібрала понад 2,7 мільйона доларів за перші два роки після релізу, перевищивши всі очікування і ставши культовим хітом серед гравців.

## 1.4 Постановка задачі

Об’єктом роботи є дослідження ігрового ринку та аналіз основних аспектів створення ігор у сучасному світі.

Метою роботи є створення повноцінної гри на мобільні пристрої, яка включає в себе всі ключові аспекти створення гри від концепту до фінального тестування

Для реалізації цієї мети необхідно виконати наступні завдання:

– вибір ігрового жанру та механік гри;

– аналіз існуючих конкурентів та створення ігрового дизайну на основі їх основних переваг та власних бажань;

– вибір середовища програмування та його додаткових елементів ;

– створення покрокового алгоритму реалізації гри;

– розробка візуальних ефектів, включаючи графіку, анімацію та інтерфейс користувача;

– реалізація гри, включаючи: програмування основної логіки, інтеграцію візуальних елементів та тестування.

# 2 СТВОРЕННЯ ДИЗАЙНУ ГРИ

## 2.1 Опис дизайну гри

Гра, яка розробляється буде жанру hybrid casual idle arcade, а значить, що акцент буде робитись на яскравих графічних елементах, мелодіях, анімаціях, та простому геймплеї. В ній неможливо буде програти, не буде обмежень у часі та будь-якої системи оцінки ефективності дій гравця.

В цій грі гравець буде керувати мисливцем з спеціальним пристроєм від третьої особи, що графічно буде схожим на Беатрікс ЛеБо з гри Slime Rancher. Головною відмінність від гри-прообраза буде те, що мисливцем ми будемо керувати від третьої особи, як у грі My Mini Mart. Це надасть плавності анімаціям та переходам, підстать казуальному жанру.

Головна ціль гри – дослідити ігровий світ, з метою збору та поєднанню усіх існуючих в ньому слаймів. Акцент гри буде робитись на різноманітті та унікальності поєднання різних слаймів, вивченні їх розташування та можливостей поєднання.

В грі буде представлено 12 базових та 144 можливих поєднань слаймів. Кожен слайм матиме розмір, базову текстуру, колір та певний візуальний елемент, наприклад крила, вуха, інші за виглядом очі тощо.

Протягом ігрового процесу, гравець буде відкривати нові біоми з різноманітними видами слаймів, переносити їх на свою базу та аналізувати їх підвид в залежності від зовнішнього вигляду. Також, на базі, завдяки додатковим інструментам, гравець буде отримувати можливість поєднувати зібрані види слаймів для отримання нових, невідомих раніше. Поєднані види слаймів будуть наслідувати візуальні елементи поєднаних.

Знаходити слаймів гравець буде в одному з 4-х біомів: зимньому, лісному, пустинному та біомі хвойного лісу. В кожному з біомів буде власний набір слаймів, що матимуть окрас та унікальні елементи, відповідний до біому та його дизайну.

## 2.2 Slime Rancher

Slime Rancher – це пригодницька відеогра з видом від першої особи, розроблена та видана американською інді-студією Monomi Park. Гра вийшла в ранньому доступі в січні 2016 року, а офіційний реліз для Windows, macOS, Linux та Xbox One відбувся 1 серпня 2017 року [21].

Гра відбувається у відкритому світі з видом від першої особи. Гравець керує персонажем на ім'я Беатрікс ЛеБо, яка переїжджає на далеку від Землі планету під назвою «Far Far Diapason». Ігровий процес полягає в будівництві свого ранчо і дослідженні світу «Far Far Diapason». Акцент робиться на збиранні, вирощуванні, годуванні і розведенні слайми.

Слайми – це желатинові живі організми різних розмірів і характеристик. Щоб просуватися вперед, володарка використовує нотатки, залишені колишнім власником ранчо, які допомагають їй у подорожі через «Far Far Diapason» [21].

Рисунок 2.1 – Геймплей гри Slime Rancher

Як можна побачити з рисунку 2.1, головним інструментом Беатрікс є пушка, завдяки якій головна героїня може перетягувати, змінювати, транспортувати та обробляти слаймів. Різноманітність дій володаря ранчо полягає в різноманітності дій її унікального інструмента, який дозволяє

робити зі слаймами те, що захоче гравець. Ціль гри – відкрити усі існуючі елементи світу, всі види слимаків та всі можливості пушки Беатрікс.

Гра захоплює яскравими фарбами, гарними 3D анімаціями, проробленим відкритим світом та можливостями. Гравець може використовувати ресурси так, як він того захоче, робити зі слимаками те, що він захоче та йти туди, куди він захоче. Сюжет гри лише підштовхує до пізнання базових механік і ніяк не примушує.

Головним недоліком я можу виділити той факт, що завдяки складній графіці, цю гру можуть підтримувати далеко не всі пристрої. Обмежена кількість стаціонарних комп'ютерів показують стабільні результати, а на Android та IOS гра не була випущена взагалі.

## 2.3 My mini mart

My Mini Mart – це симуляційна гра, створена Supersonic Studios LTD, яка дозволяє вам взяти на себе управління власним невеликим магазином і з часом розвинути його, щоб стати успішним бізнес-магнатом [18].

Гра доступна на платформах iOS та Android.


Рисунок 2.2 – Геймплей гри My Mini Mart

Судячи з рисунку 2.2, та міркувань ігрових критиків, інтерфейс зручний для користувача, а графіка приваблива, що робить його приємним для дітей. Додаток можна завантажити та використовувати безкоштовно [19].

У цій грі ви можете вирощувати органічні рослини, доглядати за тваринами та продавати свою продукцію клієнтам. У міру розвитку ви можете наймати працівників, будувати і розширюватися, відкриваючи нові магазини і перетворюючи свою маленьку крамничку на величезну імперію. [18].

Головною стратегією гри є послідовне відкриття все більш і більш складних систем по зрощенню та обробці різноманітних органічних культур та тварин, з ціллю її подальшого продажу. Ціль гри – відкрити всі види рослин та тварин, і продавати якомога більше товарів з власної ферми.

Ідея гри в тому, що гравець відкриває нові види рослин та тварин у той момент, коли він вже виконав усі необхідні завдання, зв’язані з попереднім циклом вирощування. Результат дії гравця не лінійно перетворюється на ігрову валюту, тому ще все, що було зібрано на фермі треба продати за певною системою ефективності. Клієнти у магазина з’являються послідовно один за одним, та запрошують лише певні товари, що викликає необхідність підлаштовуватись під замовлення та вирощувати збутки в певних пропорціях. Така ігрова концепція дозволяє зробити геймплей різноманітнішим та складнішим.

Але, так як нові рослини та тварини ніяк не зв’язані з ефективністю магазину, а просто відкриваються на певному етапі, і при цьому, так як клієнти, тобто єдине джерело ігрової валюти, генеруються процедурно, то гра стає лінійною. В будь-якому випадку прийдеться проходити послідовно усі етапи, незалежно від ефективності гри. При такому форматі, за неможливості поразки та банкротства магазину, та за відсутності конкуренції гра перестає бути цікавою після першого проходження, оскільки всі складнощі та можливості вже досягнені.

# 3 РОЗРОБКА ГРИ

## 3.1 Аналіз існуючих рушіїв

Першочерговою задачею перед створенням гри є вибір середи, в якій буде створена гра, та мови програмування.

Вибір мови програмування та середовища розробки має величезне значення для програмістів. Мова програмування визначає не лише синтаксис коду, але й підходи до розв’язання проблем, що впливає на продуктивність та якість розробки. Крім того, обране середовище розробки визначає зручність та ефективність роботи програміста, забезпечує необхідні інструменти для аналізу, налагодження та відлагодження програмного забезпечення. Правильний вибір також впливає на можливості співпраці з іншими розробниками та інтеграцію з існуючими проектами. Тому, враховуючи всі ці аспекти, обдуманий вибір мови та середовища розробки є ключовим для досягнення успіху у сфері програмування.

Окрім середи програмування, аспект мови програмування є не менш необхідним до аналізу.

Так наприклад, C++ є однією з найпопулярніших мов для розробки ігор, особливо для високопродуктивних ігор AAA. Він пропонує низькорівневий контроль пам’яті, що є вирішальним для оптимізації продуктивності. Багато ігрових движків, таких як Unreal Engine і Unity (частково), використовують C++ [8].

C# – ще одна широко використовувана мова, особливо для розробки ігор за допомогою Unity. Він відомий своєю простотою використання та надійним середовищем розробки. Unity має потужний API сценаріїв, побудований на C#, що робить його доступним як для початківців, так і для досвідчених розробників [8].

Python – це універсальна мова, яка набуває популярності в розробці ігор, насамперед для розробки незалежних і мобільних ігор. Можливо, це не

найкращий вибір для ресурсомістких 3D-ігор, але він чудовий для 2D-ігор і створення прототипів. Pygame – популярна бібліотека для розробки ігор на Python [8].

Java зазвичай використовується для розробки ігор для Android за допомогою таких фреймворків, як LibGDX. Хоча це не так популярно для розробки ігор для ПК або консолей, це чудовий вибір, якщо ви націлені на мобільні платформи [8]. Але, Java не є потужною мовою для програмування 3D ігор, оскільки він не підтримується на більшості 3D рушіїв, і відповідно до своєї непопулярності, не є гнучкою та поширеною.

JavaScript необхідний для веб-ігор і часто використовується з HTML5 і WebGL для браузерних ігор. Ігрові механізми, такі як Phaser і Three.js, спрощують розробку ігор за допомогою JavaScript [8].

Таким чином, кожна з мов відповідає певним вимогам та має власний інструментарій та може покращувати процес розробки ігри в залежності від бажань розробника.

Задумана гра розроблювалась як 3D гра класу AA. Вона має нескладну графіку, просту механіку idle ігор, не розповсюджений режим мультіплеера, невелике навантаження на графічний процесор, та не потребує багато часу для розробки.

Тому з числа можливих мов програмування було обрано: C++, C#, JavaScript, оскільки вони найбільше за інших відповідають вимогам.

В таблиці наведеній нижче знаходиться перелік рушіїв для розробки 3D ігор, взятий з інтернет-ресурсу dragonflydb.io. Першочергово вони розташовані за популярністю. Окрім списку, також вказані такі параметри як: складність використання, мови реалізації, безкоштовність, орієнтована платформа, на якій будуть реалізовані ігри [7].

Більше ніж 9 рушіїв описувати немає сенсу, оскільки, розташовані за списком рушії йдуть від найбільш поширеного до найменш поширеного. Починаючи з 10го рушія, за моєю думкою, середа розробки буде неактуальною, оскільки непопулярні рушії не мають таких інформаційних

ресурсів, представлених розробниками-користувачами та розробниками-володарями, у порівнянні з першими дев'ятьма. Тому при виборі платформи я обмежився кількістю 9.

Таблиця. 3.1 – Список 3D рушіїв [7]

| № | Назва | | Платформа | Складність | Кошти |
|---|---|---|---|---|---|
| 0 | Unity | C# | Телефони, Комп'ютери, Консолі, VR, AR | 2/10 | Free |
| 1 | Unreal Engine | C++ | Телефони, Комп'ютери, Консолі, VR, AR | 3/10 | Free |
| 2 | Godot | C++, C# | Телефони, Комп'ютери, Консолі | 3/10 | Free |
| 3 | Game Maker | C++, Lua | Телефони, Комп'ютери, Консолі | 2/10 | Not free |
| 4 | Open 3D Engine | JavaScript | Комп'ютери, Консолі | 4/10 | Free |
| 5 | BuildBox | JavaScript | Телефони, Комп'ютери, Консолі | 3/10 | Not free |
| 6 | Haxe | Haxe | Телефони, Комп'ютери, Консолі, Браузерні ігри | 3/10 | Free |
| 7 | Babylon.js | JavaScript | Браузерні ігри | 3/10 | Free |
| 8 | Flax Engine | C#, C++ | Телефони, Комп'ютери, Консолі, VR, AR | 3/10 | Not free |

В таблиці закцентувано увагу на тому, що платформа для написання гри повинна бути безкоштовною, випускатись на телефоні та мати мову c# / c++ або JavaScript.

Таким чином зі списку було вибрано 3 рушія: Unity, Unreal Engine, та Godot.

Але Godot це переважно 2d орієнтована платформа для розробки, тому, з усього списку рушіїв вибираємо головні 2 конкурента – Unity 3D та Unreal Engine.

## 3.2 Порівняння Unity та Unreal Engine

### 3.2.1 Порівняння продуктивності

Розіб’ємо порівняння продуктивності Unreal Engine та Unity на декілька пунктів, такі як: порівняння продуктивності для стандартних графічних моделей та рухів, порівняння рівня графіки, порівняння можливостей мов програмування, які підтримуються на цих платформах, наявності або відсутності додаткових формуів та бібліотек, розповсюдженість платформ, тощо.

Зробимо порівняння продуктивності Unreal Engine та Unity на прикладі конкретного експериментального аналізу [15].

Об'єктом цього дослідження була покрокова стратегічна гра, в якій користувач керує персонажами на створеному рівні. Розрахунок траєкторії руху персонажа здійснювався за допомогою алгоритму «A* Pathfinding» [16]. Гравець також має доступ до інших типів пересування – очікування черги, стрільба з вогнепальної зброї, кидання гранати, атака з близької відстані або взаємодія з дверима. Карта відкривається користувачеві в міру того, як відкриваються двері до наступних кімнат. Гра була запрограмована на двох ігрових рушіях – Unity та Unreal Engine. Було використано останню версію (на момент дослідження) Unreal Engine 5.2.1, а також останній випуск довгострокової підтримки (LTS) Unity – 2021.3.18f1 [15].

Побудовані ігри тестувалися на двох комп’ютерах, параметри яких наведено в таблиці 3.2. На першій робочій станції гра запускалася в роздільній здатності 2560x1440 пікселів, на другій – в роздільній здатності 1920x1080 пікселів.

Таблиця 3.2 – порівняння двох моделей графічних рушіїв

| | № 1 | № 2 |
|---|---|---|
| Операційна система | Windows 11 | Windows 10 |
| Процесор | Intel Core i7-13700K, 16 rdzeni (8 Performance + 8 Efficient) | Intel Core i5-9300HF, 4 rdzenie |
| ОЗУ оперативної пам’яті | 32 GB DDR4, 4000 MHz, CL 18-22-22-42 | 8 GB DDR4, 2400 MHz, CL 17-17-17-39 |
| Відеокарта | NVIDIA RTX 4080, 16 GB GDDR6X | NVIDIA GeForce GTX 1650, 4 GB GDDR5 |

Категоріями порівняння, на прикладі ігор, які запускались на цих двох платформах були: кількість кадрів в секунду, використання відсотків центрального процесору, середнє використання оперативної пам’яті, середнє використання пам’яті відеокарти.

Ресурси вимірювались в трьох випадках, при неруховому стані персонажа, при базовому русі та при проведенні атаки.

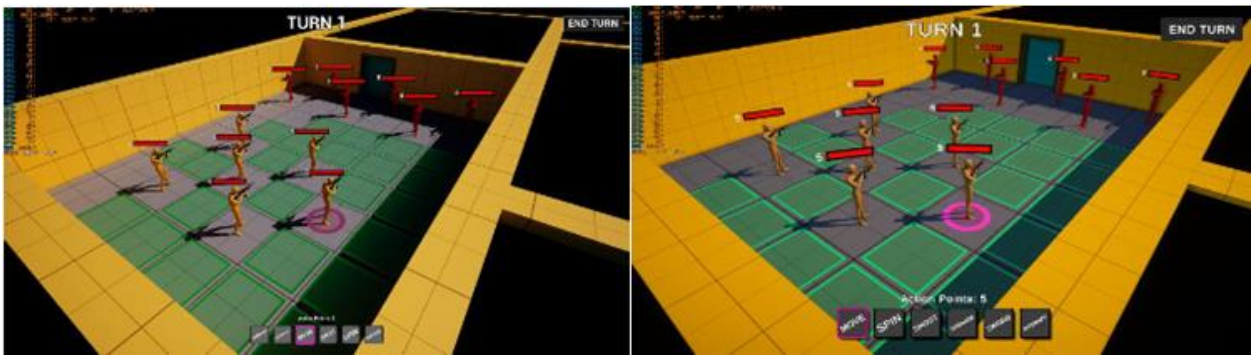



Рисунок 3.1 – Порівняння ефективності Unity (зліва) та Unreal Engine (зправа)

Результатом дослідження були наступні висновки:

– рушій Unity показує більший результат в обох дослідженнях з точки зору середнього значення Fps, але з точки відхилення від середнього

значення в критичних графічних навантаженнях, Unreal Engine має кращі характеристики;

– використання ресурсів процесору відрізняється менше ніж на 4 відсотка, тобто в цьому аспекті рушії майже рівні за характеритиками;

– використання оперативної пам’яті більш ефективне у Unity в обох випадках, а ніж у Unreal Engine.

Таким чином можна зробити висновки, що при наявності потужної відеокарти та процесору, а також шини підкачки оперативної пам’яті, Unreal Engine показує більші але нестабільні результати, тоді як Unity отримує результати, що є кращими з точки зору середнього результату, але з більшою амплітудою відхилення в критичних точках.

З точки зору розробки гри характеристики Unity більш рекомендовані, оскільки гра розроблялась для платформи телефону, яка є більш обмеженою з точки зору відеокарти та процесору, ані ж альтернативні платформи. Окрім того, так як гра має клас AA, а не AAA, але потребує 3D графіки, то моментів критичного навантаження буде менше, і більший середній результат Unity буде мати перевагу перед біьшим максимальним результатом Unreal Engine

На Unity представлена мова C#, а на Unreal Engine представлена мова C++.

Правильна мова кодування та скриптів має важливе значення для створення гри, яка робить те, що ви хочете, без потреби в нескінченно складному коді [17].

C++ – це високопродуктивна мова програмування загального призначення, відома своєю швидкістю та гнучкістю. C++ відома своєю винятковою продуктивністю. Здатність мови безпосередньо маніпулювати пам’яттю та оптимізувати код під конкретне апаратне забезпечення дає їй значну перевагу у ресурсоємних додатках. C++ найкраще підходить для сценаріїв, де абсолютна продуктивність і контроль над системними ресурсами мають першорядне значення [17].

C# – це керована, об’єктно-орієнтована мова, розроблена компанією Microsoft. Вона є частиною фреймворку .NET і відома своєю простотою використання та надійністю. C# абстрагується від багатьох низькорівневих деталей, пропонуючи простіший синтаксис та автоматичне керування пам’яттю за допомогою збирача сміття. В порівнянні с C++, C# пропонує більш спрощений і дружній до початківців підхід, особливо в поєднанні з ігровими рушіями, такими як Unity [17].

### 3.2.2 Asset Store

Однією з додаткових переваг C# перед C++ є наявність платформи Asset Store, яка представляє з себе інтернет ресурс з набором даних (Asset – їх назва), які представляють з себе аудіо, відео та графічні ресурси під потреби розробника. Наприклад, в Asset Store я скопіював ефекти під назвою Epic Tune FX, які являють з себе набір графічних ефектів, таких як вибухи, іскри, води та інші.

Асети, завантажені з платформи Asset Store розроблені під інтегровану в C# біблотеки, що робить можливим ефективніше використовувати інтернет ресурси відкритого доступу у власних розробках. І так як гра позіціонує себе як невелика 3D гра орієнтована під телефони, наявніть такої платформи як Asset Store значно спрощує процес розробки.

Аналогії Asset Store для C++ немає, через складність та неукомплектованість стандартних бібліотек, тому Asset Store є великою перевагою C# перед C++. І так як C# підтримується лише на Unity, то Asset Store стає перевагою і Unity над Unreal Engine.

### 3.2.3 Порівняння популярності двигунів

Обидва рушії мають солідну присутність на сцені розробників ігор, тому не дивно, що вони мають великі спільноти. Давайте подивимось, скільки підписників у сабреддітах: 238 тис. учасників у підредакторі Unity проти 111 тис. у Unreal Engine. Можна подумати, що ці цифри не говорять багато. Але це все ж таки слушна точка для порівняння: вдвічі більше людей у спільноті Unity, а це означає більше запитань і, що важливіше, більше відповідей. І, звичайно, більша база знань [17].

### 3.2.4 Компонентна архітектура

Компонентна архітектура (Component based architecture з англ.) – це методологія обробки коду, інтегрована в Unity, яка використовувалась мною при розробці гри.

Компонентна архітектура фокусується на декомпозиції проекту на окремі функціональні або логічні компоненти, які представляють чітко визначені комунікаційні інтерфейси, що містять методи, події та властивості. Він забезпечує вищий рівень абстракції та розділяє проблему на підпроблеми, кожна з яких пов’язана з розділами компонентів [11].

Основною метою компонентної архітектури є забезпечення повторного використання компонентів. Компонент інкапсулює функціональність і поведінку елемента програмного забезпечення в бінарний блок, який можна повторно використовувати та самостійно розгортати [11].

Кожен елемент виконує завдання архітектурно визначеним способом. Для простоти компоненти зберігаються в бібліотеці, зібраній розробниками, і спілкуються один з одним через API [11].

Посередник запитів об’єктів, який іноді називають «програмною шиною», полегшує зв’язок, надаючи єдину площину зв’язку, яку використовують усі компоненти. Комунікація може відбуватися декількома

способами, наприклад асинхронно, через трансляцію, через систему, керовану повідомленнями, або як частину поточного потоку даних [12].

Перевагами компонентної архітектури перед іншими способами зберігати дані:

– гнучкість: компоненти можна легко замінити або модифікувати, не впливаючи на всю систему, забезпечуючи гнучкість адаптації до мінливих вимог;

– повторне використання: компоненти можна повторно використовувати в різних проектах, заощаджуючи час і зусилля на розробку;

– масштабованість: оскільки компоненти незалежні, їх додавання або видалення не впливає на систему в цілому;

– ремонтопридатність: завдяки чітким межам між компонентами стає легше підтримувати та налагоджувати систему;

– паралельна розробка, завдяки якій, у компонентній архітектурі різні команди можуть працювати над різними компонентами одночасно. Цей підхід до паралельної розробки покращує співпрацю та зменшує залежності, забезпечуючи швидші цикли розробки та кращий розподіл ресурсів [13].

Недоліками компонентної архітектури є:

– ізольовані компоненти, які IT-адміністратори тестують окремо, це життєво важливий процес, щоб керувати ними та впорядковувати їх, але він все одно виснажливий і трудомісткий;

– повторення компонентів. використання яких у різних програмах робить їх менш настроюваними;

– не варіативний компонентний підхід, який потребує значного обслуговування. Першою проблемою може бути пошук компонента, який відповідає потребам програми. Крім того, оновлювати та підтримувати бібліотеки компонентів непросто, оскільки вони потребують частого моніторингу [14].

Якщо проаналізувати головні переваги та недоліки, то можна дійти до висновку, що компонентна архітектура буде гарним кроком до використання

у розробці ігри класу AA з використанням 3D графіки, оскільки ані навантаження на систему розробки, ані складність тестування ані не варіативність не є перешкодами при розробці гри такого рівня. Проте, переваги, які надає компонентна архітектура є значимими та необхідними.

І так як компонентна архітектура інтегрована в Unity 3D, то вона є ще одним аргументом на користь Unity 3D перед Unreal Engine та Godot.

## 3.3 Створення гри

Тепер перейдемо безпосередньо до створення гри. На цьому етапі необхідно розділити реалізацію на декілька основних аспектів: візуальний аспект, ігрову логіку, та реальзацію коду.

Візуальний аспект буде враховувати в себе: створення моделей та анімацій персонажу, слаймів, локацій; додатковий інструментарій поєднання слаймів, їх зовнішній вигляд тощо. Ігрова логіка буде враховувати в себе закономірності генерації біомів, створення слаймів, систему порталів, закони поєднання слаймів та можливості гравця. В цьому розділі будуть введені норми розмірів ігрових полей, часові та додаткові обмеження, тощо

В другому етапі буде розповідаться про реалізацію гри на мові програмування C# в графічному рушії Unity 3D. Будуть показані моменти використання Shader Graph, приведені результати коду та інше.

## 3.4 Візуальне оформлення гри

### 3.4.1 Дизайн персонажу

Основною метою було створити візуально простого, але функціонально насиченого персонажа, здатного збирати «слайми» за допомогою спеціалізованої пушки.

Персонажа було задумано як симпатичного, легко впізнаваного героя з мінімалістичними, але виразними рисами, характерними для стилю My Mini Mart. Його основний атрибут, пушка для збору слаймів, вносить унікальний елемент інтерактивності, який запозичено з Slime Rancher. Ця пушка дозволяє персонажу взаємодіяти з різними видами слаймів у грі, збираючи їх для виконання завдань чи удосконалення своєї бази.

Розробка персонажа почалася з етапу скетчінгу, де було створено кілька варіантів дизайну, щоб визначити оптимальне співвідношення між простотою та функціональністю. Остаточний дизайн був реалізований у Blender, де персонаж отримав текстури та анімації, призначені для підсилення його характеру та інтерактивності.

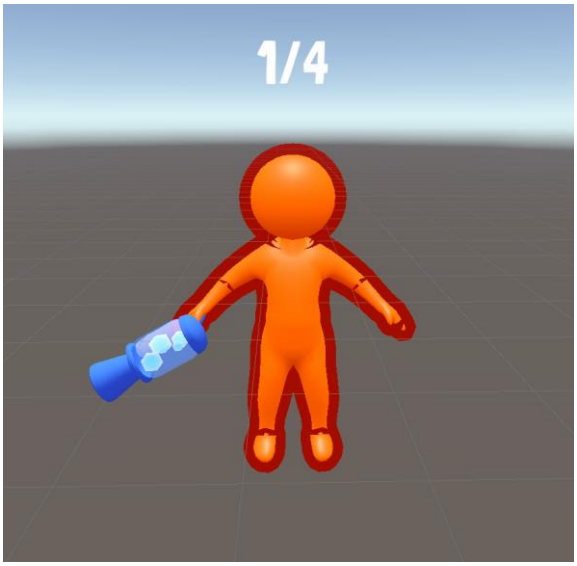



Рисунок 3.3 – Модель ігрового перонажу

Загалом, було розроблено 12 унікальних слаймів, кожен з яких має свій колір та індивідуальний візуальний атрибут, що підкреслює його характер. Візуальні атрибути включають елементи, такі як хвости, носики, роги, що не

тільки забезпечують унікальність кожного слайма, але й відображають їх емоційний стан та поведінку.

Процес створення кожного слайма починався з вибору кольору та відповідного атрибуту, який би підходив до його особливостей. Наприклад, червоний слайм має злі брови, що символізують його агресивність, тоді як рожевий слайм з крилами відображає його дружність та легкість. Кожен слайм був розроблений у Blender, де були створені деталізовані моделі та текстури для кожного персонажа.

Для економії ресурсів, та спрощення розробки, усі атрибути слайму були розміженні на одній моделі. Для того ,щоб під час гри можна було просто вмикати потрібний атрибут, та назначати відповідний колір.

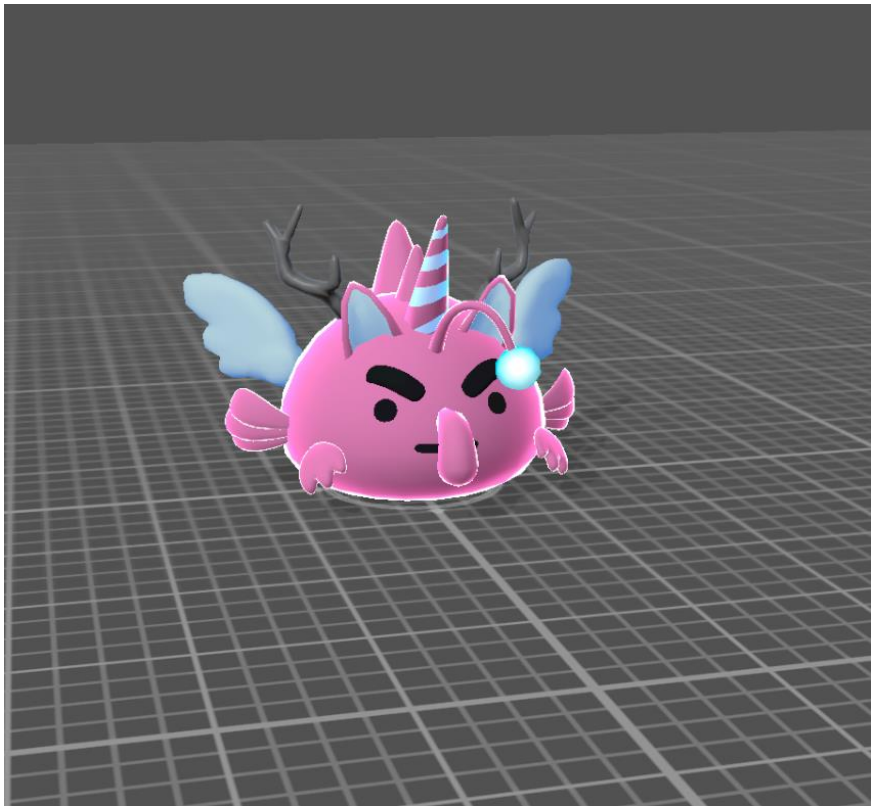


Рисунок 3.4 – Слайм, який наслідував усі унікальні графічні елементи гри.

У грі представлено чотири біоми: пустеля, ліс, зимовий та хвойний ліс, кожен з яких має унікальне середовище і впливає на колірну гаму слаймів, що в них мешкають. Слайми адаптують свій зовнішній вигляд до умов біома,

в якому вони живуть, що додає різноманітності геймплею та занурення в ігровий світ.

Наприклад, слайми пустелі мають відтінки від жовтого до глибокого червоного, відображаючи піщаний ландшафт цього біома, а улісовому біомі переважають зелені та рожеві кольори, що дозволяє слаймам гармонійно вписуватися в густу рослинність. Це покращує досвід взаємодії з грою.

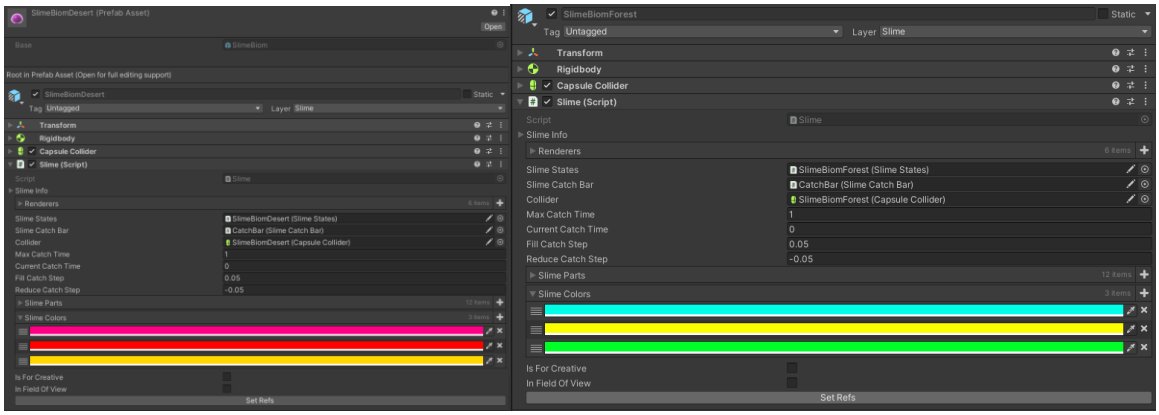



Рисунок 3.5 – Програма, що поєднує кольори відповідних біомів та розфарбовує слаймів у відповідні відтінки.

Кожен з 4-х біомів буде мати своє власне насичення.

Біом пустелі матиме перешкоди на шляху гравця у вигляді пальм, та кактусів.

Ліс матиме перешкодами одиничні дерева, або рощиці у вигляді поєднання декількох дерев; зима матиме великі кам’яні глиби; хвойний ліс матиме ялинки.

Окрім перешкод кожен з біомів матиме певну структуру підлоги, оформлену у кольорі під стать назви. Так, хвойний ліс матиме фіолетовий окрас, такий самий як і окрас його ялинок, ліс матиме зелений, пустеля жовтий колір, а зимній біом – білий.

Рисунок 3.6 – Графічні об’єкти, які належать до ігрових біомів

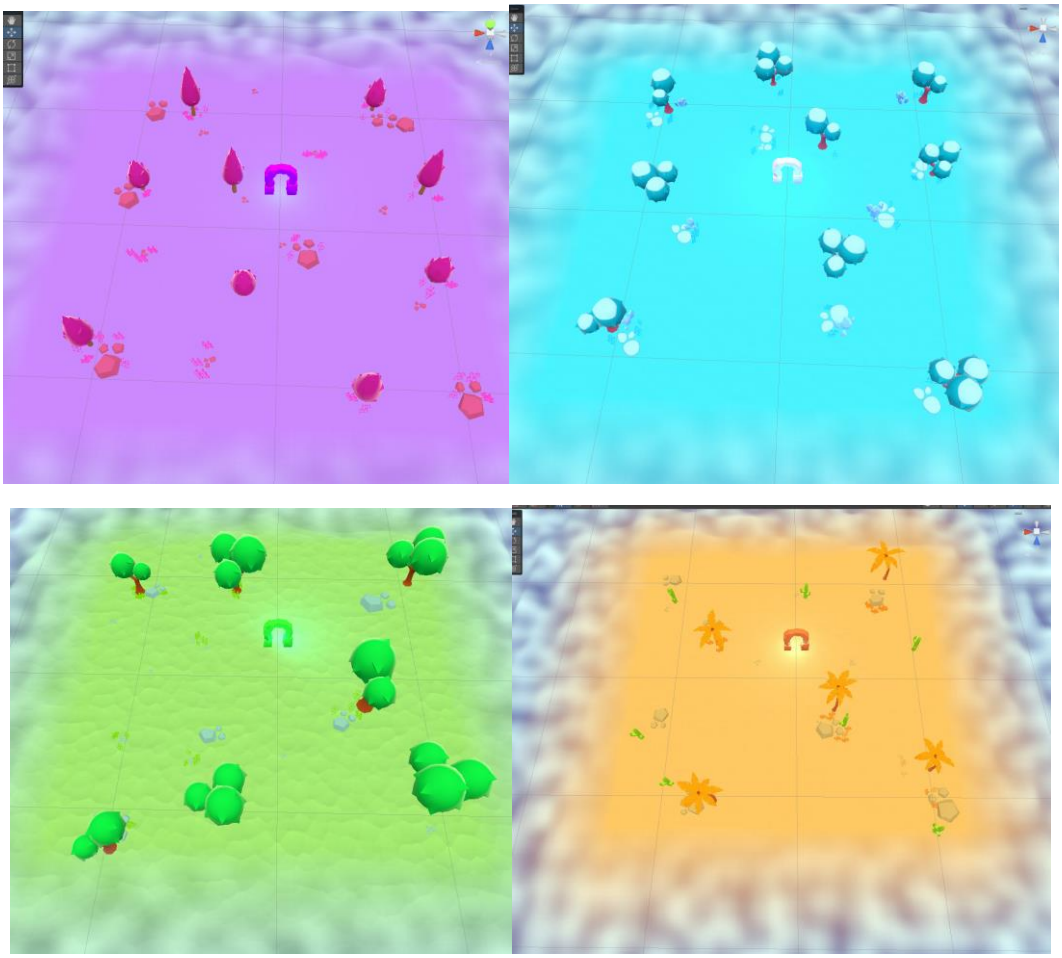


Рисунок 3.7 – Зовнішній вигляд біомів гри.

Ферма слаймів буде місцем, де гравець буде поєднувати отриманих слаймів один з одним у певних загонах.

Кожен з загонів буде представляти з себе місце, представлене у вигляді набору клітин 4 на 4. Кожна клітина матиме місце для одного слайму, та кожен з загонів матиме окрему можливість для вирощування слаймів. Загони огорожені заборами та будуть відкриватись за ігрову валюту послідовно.

Також на місці ферми знаходяться 4 портали, які ведуть у відповідні біоми для збору слаймів


Рисунок 3.8 – Зовнішній вигляд ферми слаймів

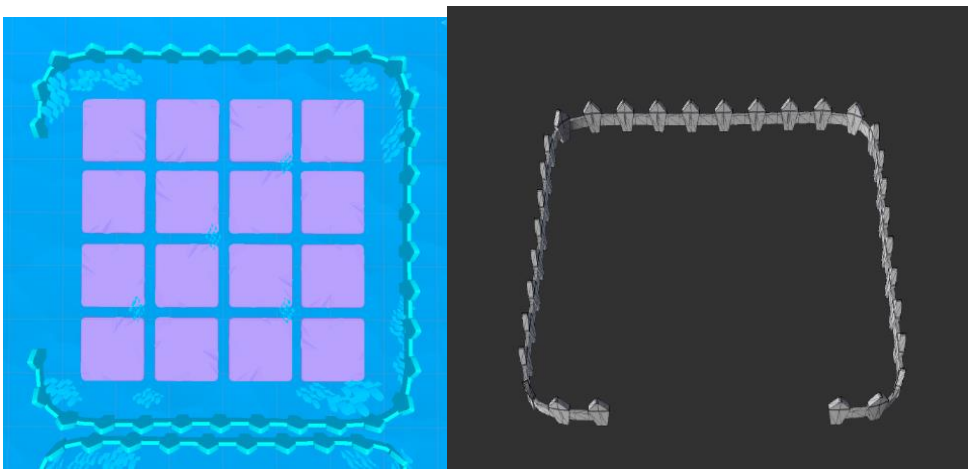


Рисунок 3.9 – Зовнішній вигляд загону


Рисунок 3.10 – Зовнішній вигляд порталу

### 3.4.2 Ігрові закономірності

Персонаж рухається за допомогою джойстику, а взіємодія з усім ігровим світом реалізована через пересування гравця та зупинках на візуально виділених місцях у самій грі, що робить геймплей дуже простим.

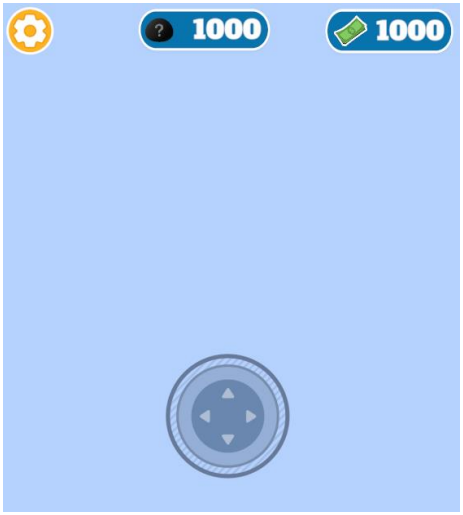



Рисунок 3.11 – Управління джойстиком

На базі є 4 портали, перший одразу відкритий. Інші треба купити за ігрову валюту, яку ми заробляємо зі спійманих слаймів у загоні.

При вході в портал, ми потрапляємо в відповідний порталу біом. Біом генерує 12 слаймів, опираючись на шанси. 50, 30 та 20 відсотків на спавн відповідних видів слаймів у кожному біомі. Це зроблено для того, щоб із часом гри, було складніше спіймати потрібні слайми для мержу нових видів.

У гравця є обмеження на кількістьсть зібраних слаймів. Максимум можна із собою носити лише 6 штук. Після цього їх треба віднести на базу, і закинути в загін.

Коли гравець виходить з біому, то починається відлік на відновлення цього біому. Якщо одразу зайти в попередній біом, то на ньому залишаться слайми, які були до виходу з нього.  Для того, щоб в біомі знову з’явились нові, треба почекати 60 секунд. За цей час гравець може або відвідати інший біом, або зайнятись мержом слаймів, або перенесеням їх між загонами для зручного розташування. Після потраплання слайму в загін, він починає генерувати ігрову валюту, яку можна буде витратити на нові ферми або нові портали.

У грі реалізована Merge  та Drag and Drop механіки. Після того, як гравець стає на зону одної з ферм, камера перемикається на вид зверху, і ми бачимо сітку.

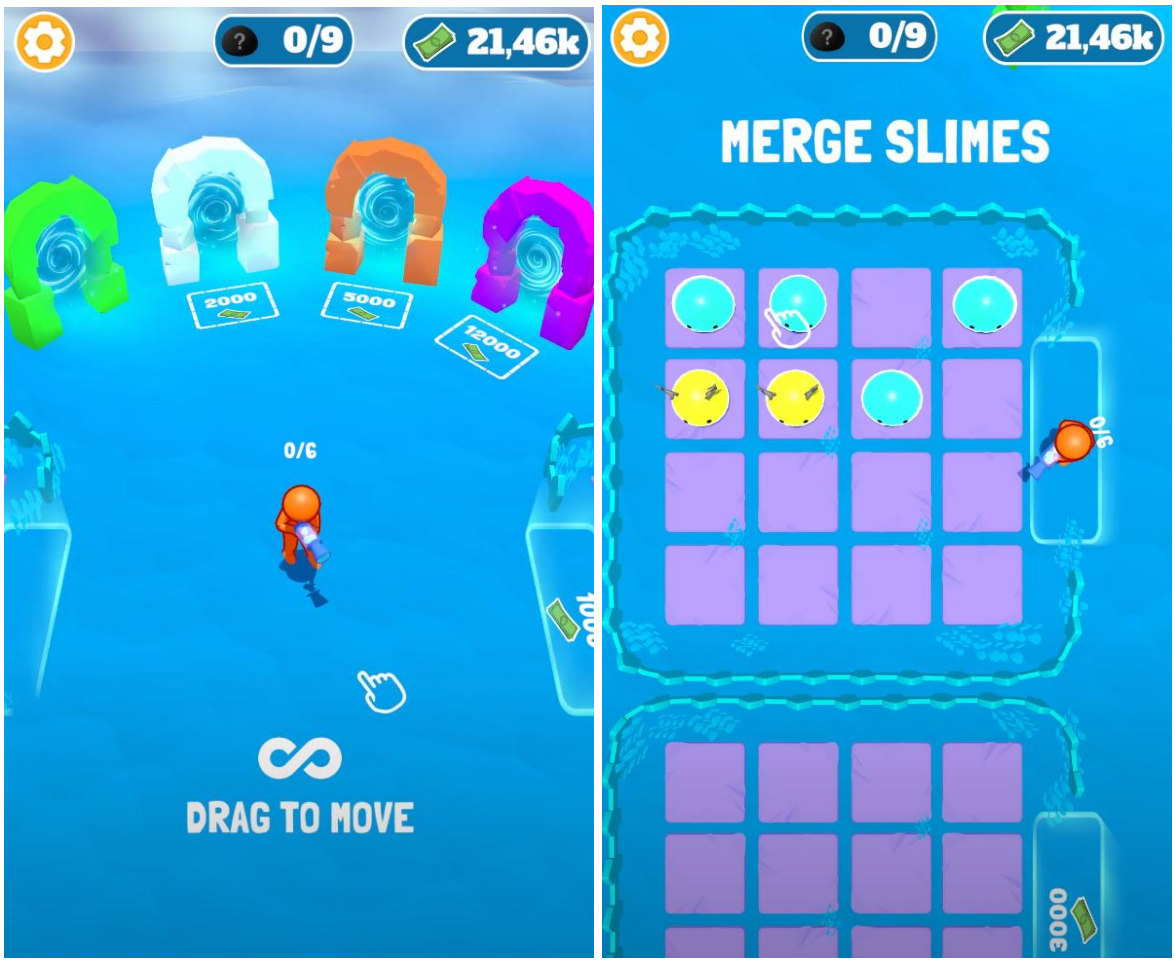



Рисунок 3.12 – Інтерфейс ферми для поєднання слаймів

У цьому режимі гравець може:

– переміщувати слаймів, перетаскуючи їх з клітинки на клітинку (рисунок 3.13);

Рисунок 3.13 – Переміщення слаймів на фермі

– викинути, перемістивши слайм за межі загону, слайм відсвітиться червоним та видалиться (рис 3.13);

– змержити, перемістити один слайм, на клітинку, в якій вже є інший слайм, після цього візуальні елементи двох створінь поєднаються, їх кольори стануть градієнтом, а розмір збільшиться (рис 3.14);

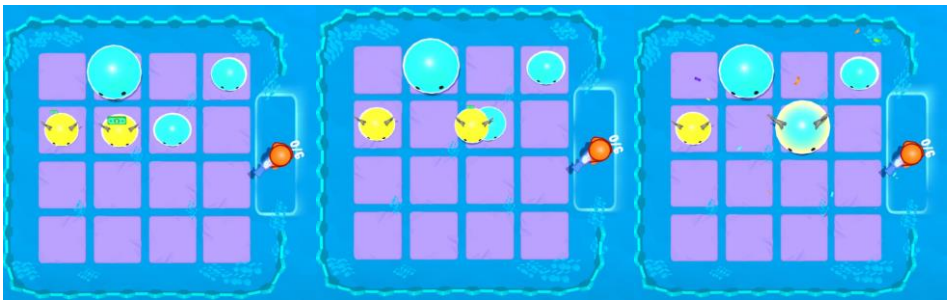


Рисунок 3.14 – Поєднання слаймів на фермі для слаймів

– додати слайм в рюкзак гравцю, перемістивши слайм на самого гравця (рис 3.15).

Рисунок 3.15 – Додавання слайму до рюкзака

Коли гравець, відкриває нове створіння, лічильник відкритих слаймів збільшується, та програється відповідна анімація.


Рисунок 3.16 – Анімація додавання

Для того, щоб купити нові портали, і отримати доступ до нових світів, потрібно лише мати достатню кількість ігрової валюти, та стати на візуально помічений цінник навпроти порталу. Гроші автоматично будуть списуватись, і в результаті ми відкриємо портал.

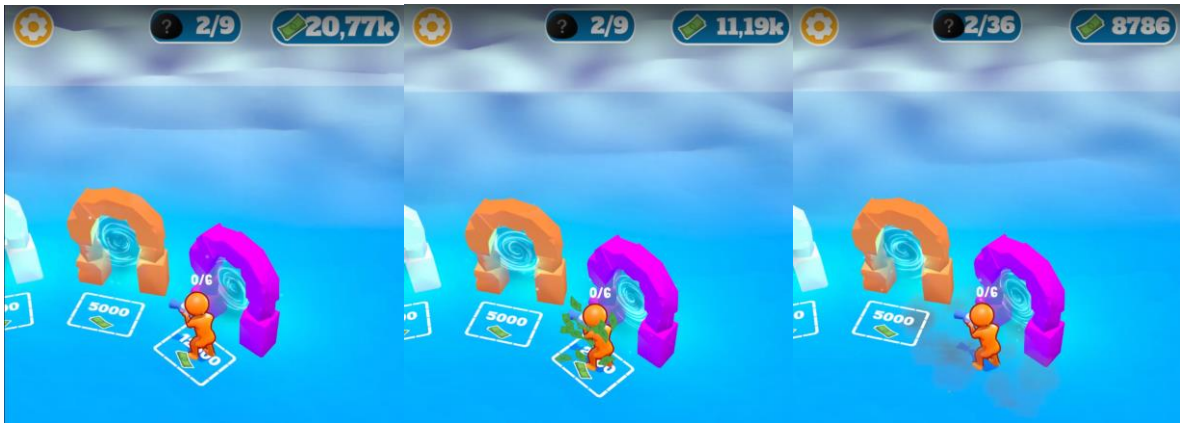



Рисунок 3.17 – Портали для переходу в інші локації

## 3.5 Реалізація гри

Для початку потрібно створити проект в Unity, де буде далі проводитись наша робота. Спочатку треба завантажити Unity Hub з сайту Unity. Далі потрібно встановити потрібну нам версію (2020.3.38f1) на комп’ютер, потім створити та назвати наш проєкт (рис 3.18).

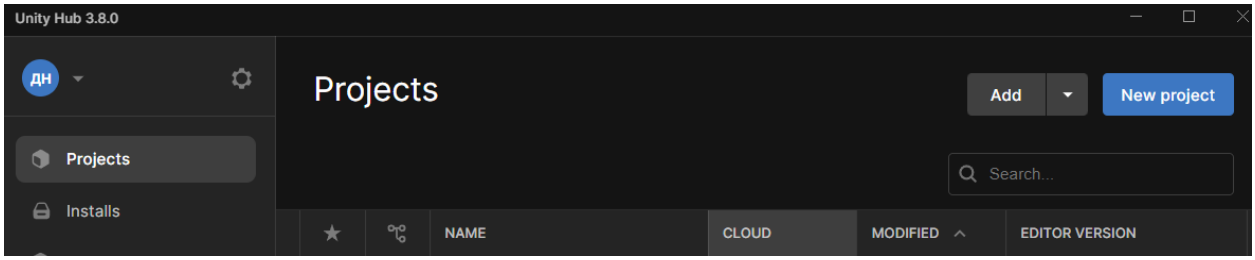



Рисунок 3.18 – Створення проекту в Unity Hub

При створенні нового проекту в Unity Hub, необхідно обрати відповідний темплейт, що задовольняє вимоги вашого проекту. Темплейт проєкту в Unity Hub це заздалегідь сконфігурована версія, яка включає специфічні налаштування, бібліотеки та іноді активи чи зразки сцен.

Використання темплейтів ефективно спрощує процес розробки, також вони забезпечують основну структуру та конфігурацію, оптимізовану для певних цілей або платформ, що значно прискорює запуск проекту.

З огляду на те, що ідея гри включає 3D оточення та об’єкти, необхідно обрати із списку темплейтів саме «3D with URP» (Universal Render Pipeline). Темплейт URP оптимізований для створення високоефективних 3D–ігор із використанням універсальної системи рендерингу, що пропонує поліпшені візуальні можливості та більш гнучкі налаштування рендерингу (рис. 3.19).

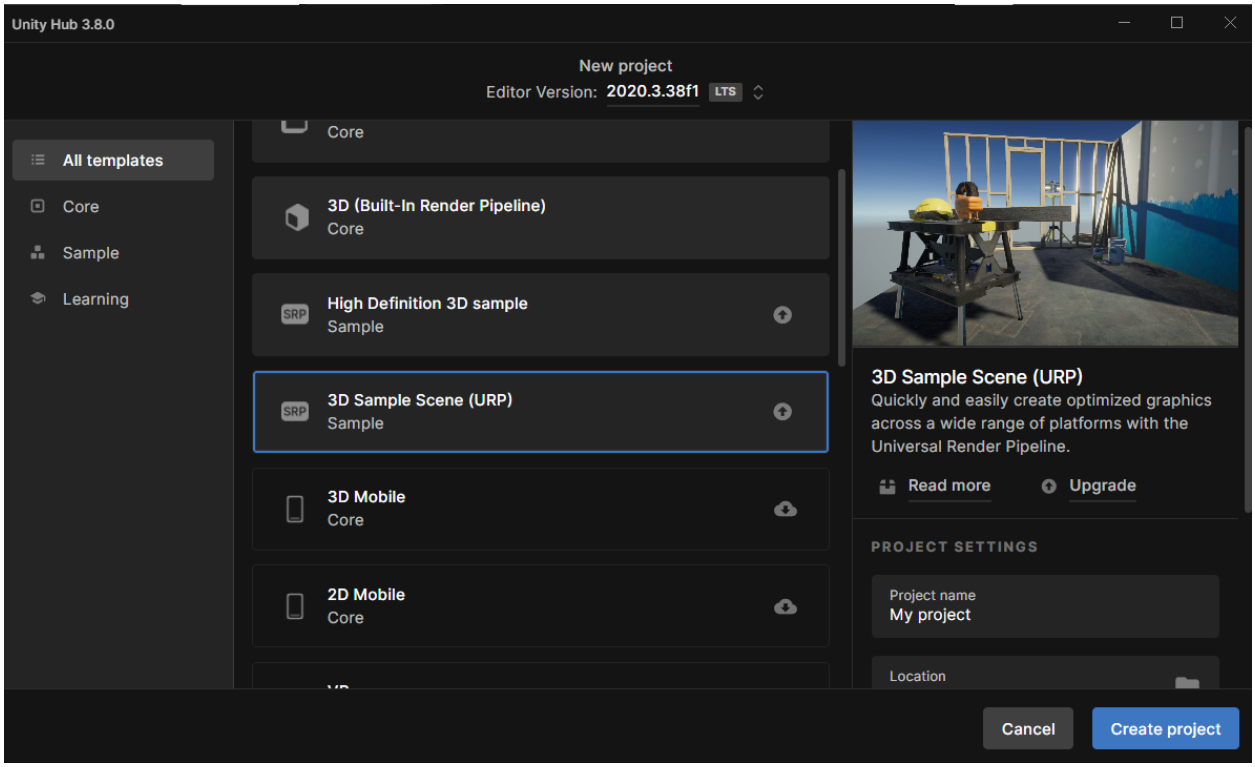



Рисунок 3.19 – Вибір URP темплейту

Після вибору темплейту задаємо назву нашому проєкту, шлях, де будуть знаходитись файли, а також потрібно вказати назву організації, яка буде використовувати двигун. Створення організації можливе в обліковому записі на платформі Unity, що необхідно для подальшого використання сервісів платформи і рушія.

Після створення проекту відкриється сам двигун Unity, де ми зможемо побачити багато інструментів, допоміжних вікон та багато іншого, що ми будемо використовувати поступово (рис. 3.20).

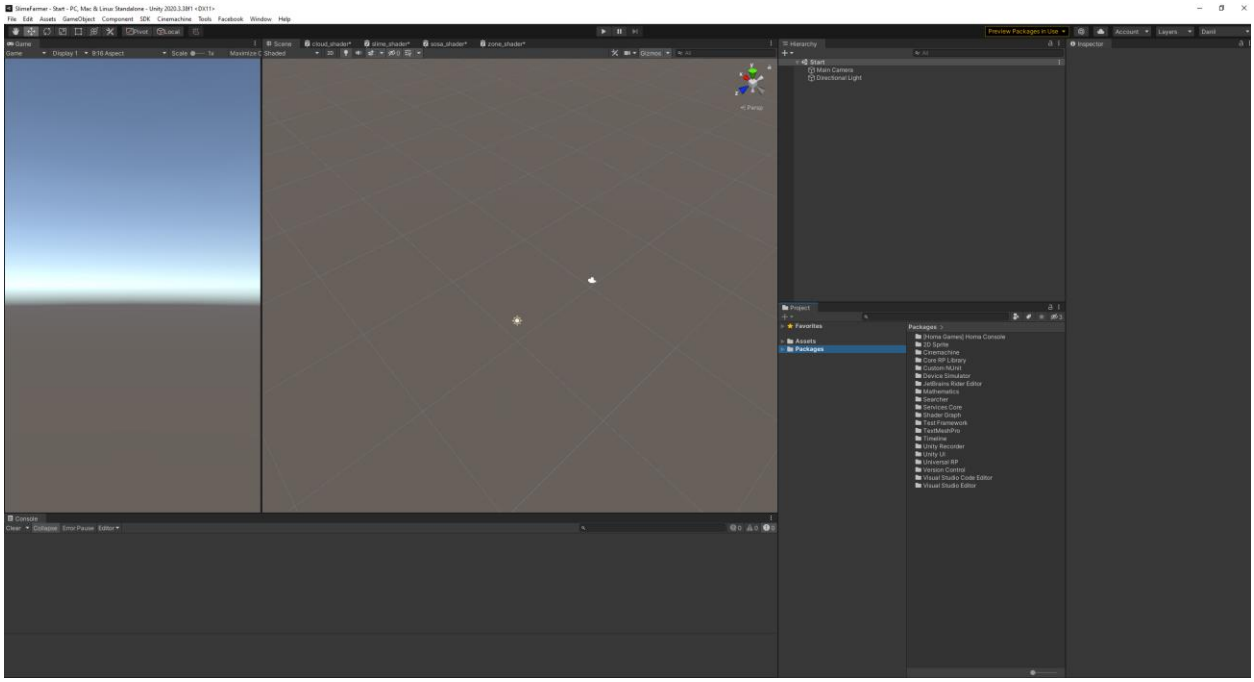


Рисунок 3.20 – Інтерфейс рушія Unity

Оскільки проект буде випускатись на платформу Android, то потрібно змінити налаштування платформи в самому Unity. Зробити це можна, перейшовши по вкладці на верхній панелі інструментів Unity. Загальний шлях: File – Build Settings – Обрати у списку платформ, платформу Android – Натиснути кнопку Switch Platform.

Цей процес переключення платформи є важливим, адже він адаптує всі налаштування проекту та ресурси для оптимальної роботи на вибраному обладнанні.

### 3.5.1 Використання Shader Graph для створення візуальних ефектів

Окрім стандартниї елементів Unity, в цій платформі для розробки ігор інтегровані можливості під’єднання додаткових елементів, які можуть

покращити процес розробки ігор у багатьох сенсах. Одним з таких є Shader Graph.

Shader Graph — це інструмент візуального редагування Unity для шейдерів. Він дає вам змогу створювати багато ефектів, які раніше можна було створити лише за допомогою шейдерного коду, але натомість він використовує вузли, що полегшує деяким людям опанування Shader Graph і часто прискорює ітерацію порівняно з написанням коду [9].

Будучи інтегрованим в unity 3D, Shader Graph є ефективним інструменом, що оптимізує та полегчує працю програміста-розробника.

Ось декілька переваг у використанні Shader Graph перед звичайним кодуванням шейдерів:

– графік шейдерів забезпечує набагато простіший спосіб створення шейдерів порівняно з традиційним кодуванням шейдерів;

– замість того, щоб писати складні рядки коду, можливо візуалізувати операції на графі на основі вузлів;

– він забезпечує миттєвий зворотний зв’язок і дозволяє набагато швидше та ефективніше розробляти шейдери [10].

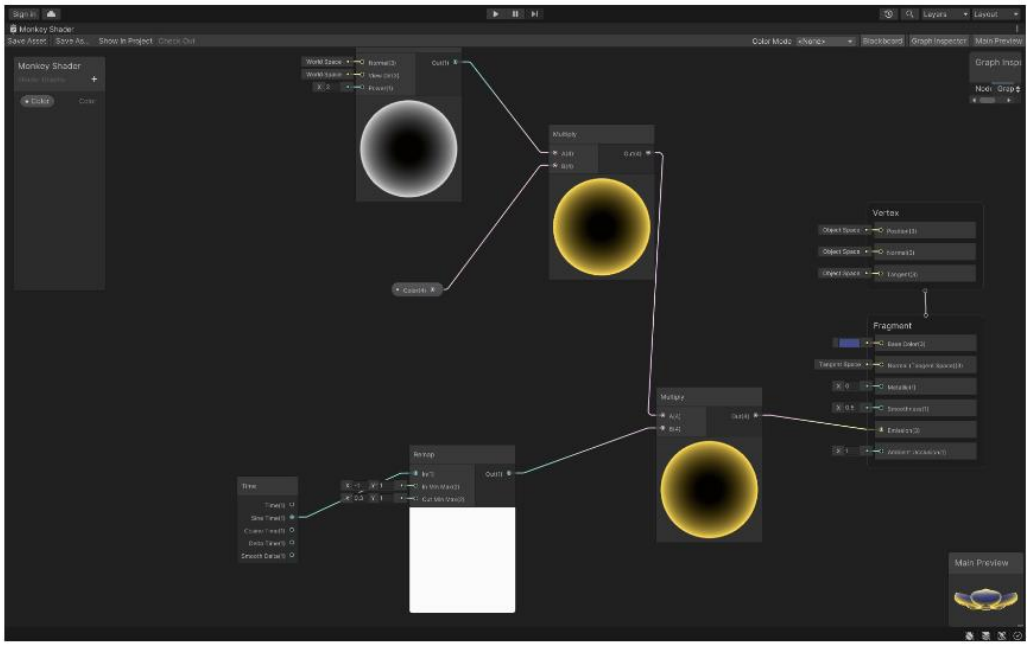



Рисунок 3.21 – Використання Shader Graph для створення візуальних ефектів

У рамках створення гри треба було створити візуальний елемент, який дозволив би гравцеві інтуїтивно визначати межі ігрової локації. Для цього було вирішено використати шейдер для створення візуального ефекту хмар, який би делікатно вказував на те, що подальший рух за межі поточної зони неможливий. Це особливо важливо у казуальних іграх, де інтуїтивне розуміння інтерфейсу та ігрового простору суттєво знижує поріг входження для користувачів.

Процес розробки починається зі створення нового Shader Graph, після чого поетапно додаються вузли (nodes), кожен з яких виконує певну функцію в загальному процесі генерації візуального ефекту.

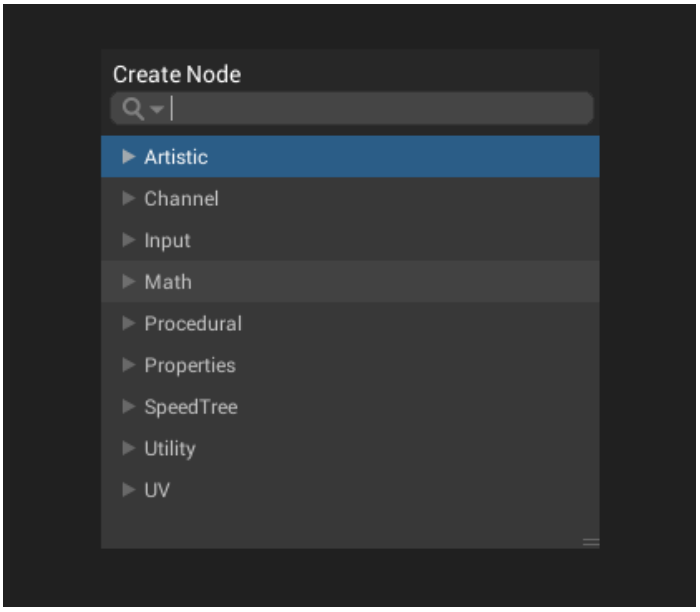



Рисунок 3.18 – Панель додавання node у Shader Graph.

Додаються параметри для управління кольором хмар, їхньою густотою та іншими візуальними аспектами, які можуть бути змінені в редакторі Unity для досягнення бажаного ефекту.

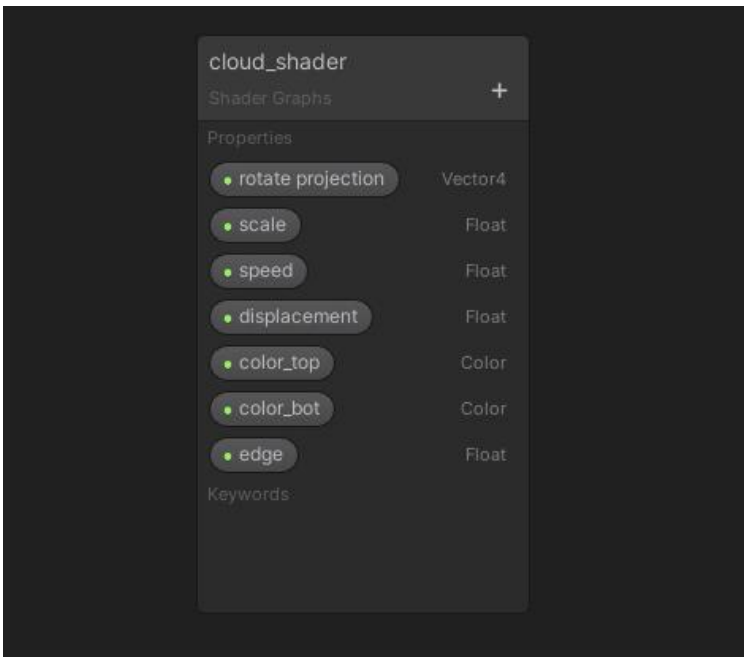



Рисунок 3.19 – Панель з параметрами у Shader Graph

Використовуються вузли для створення процедурної текстури, що імітує хмари. Додаємо шум, зміщення за часом, плавні зміни кольорі для реалістичного візуального стилю.

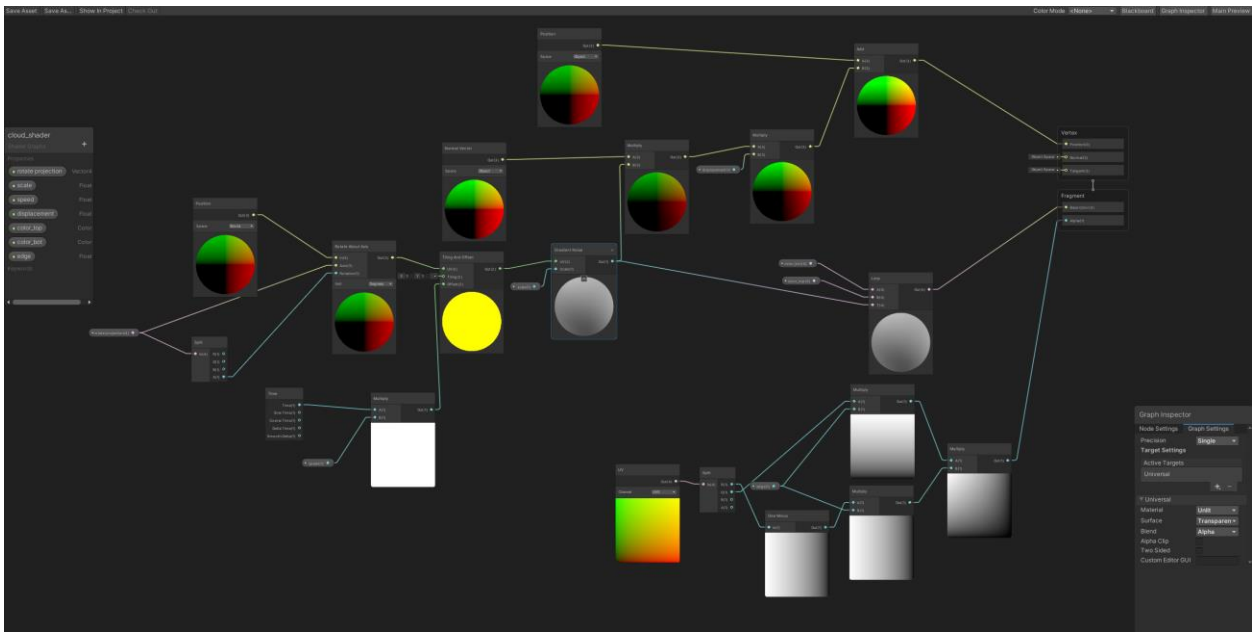


Рисунок 3.20 – Реалізація хмар у Shader Graph

Далі створюється матеріал на основі створенного шейдеру, налаштовуються параметри для більш реалістичної поведінки хмари і додається на модель у сцені

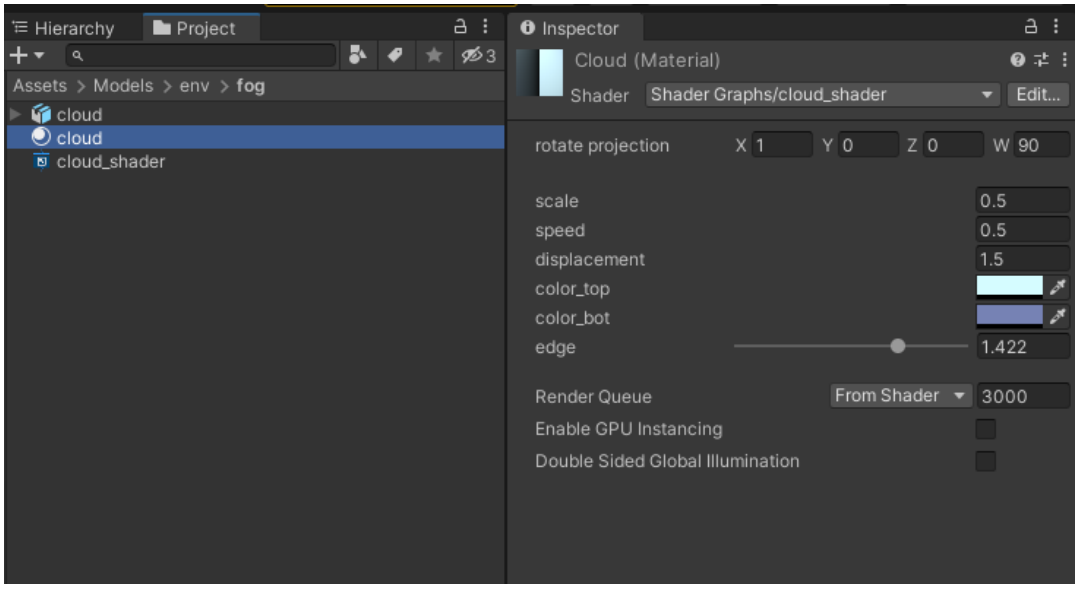



Рисунок 3.21 – Створений матеріал та параметри налаштувань шейдеру

Для створення одного з головних елементів гри, а саме слайма, потрібно було розробити шейдер, який візуалізаціє комбінацію кольорів під час злиття слаймів, а також створює ефект блискучої слизу. Це повинно сприяти глибшому залученню гравців і збільшенню візуальної привабливості гри. Додаємо параметри для налаштування первинного стану кожного зі слаймів, а також зміни візуалу матеріалу під час злиття двох слаймів в один.

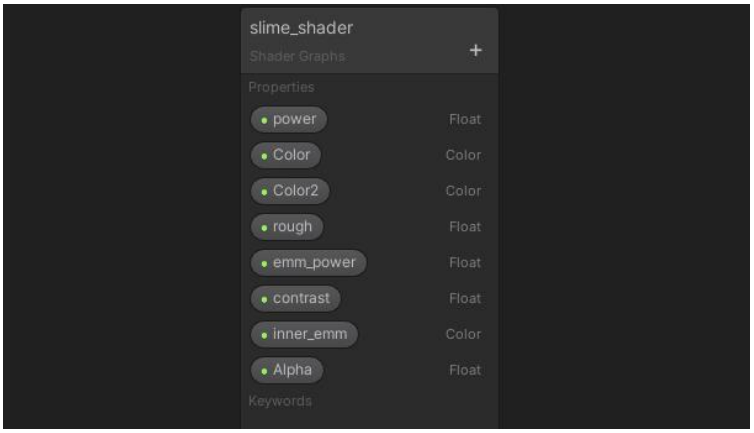



Рисунок 3.22 – Параметри налаштувань матеріалу слайму

Ефекту слизу можна досягти використовуючи нод Freshel Effect. Він додає більше блиску поверхні обьекту при збільшенні куту спостерігання, в результаті створюється ілюзія вологого, блискучого покриття. Також додаємо необхідні ноди для змішення кольорів та параметрів контрасту, емісії.

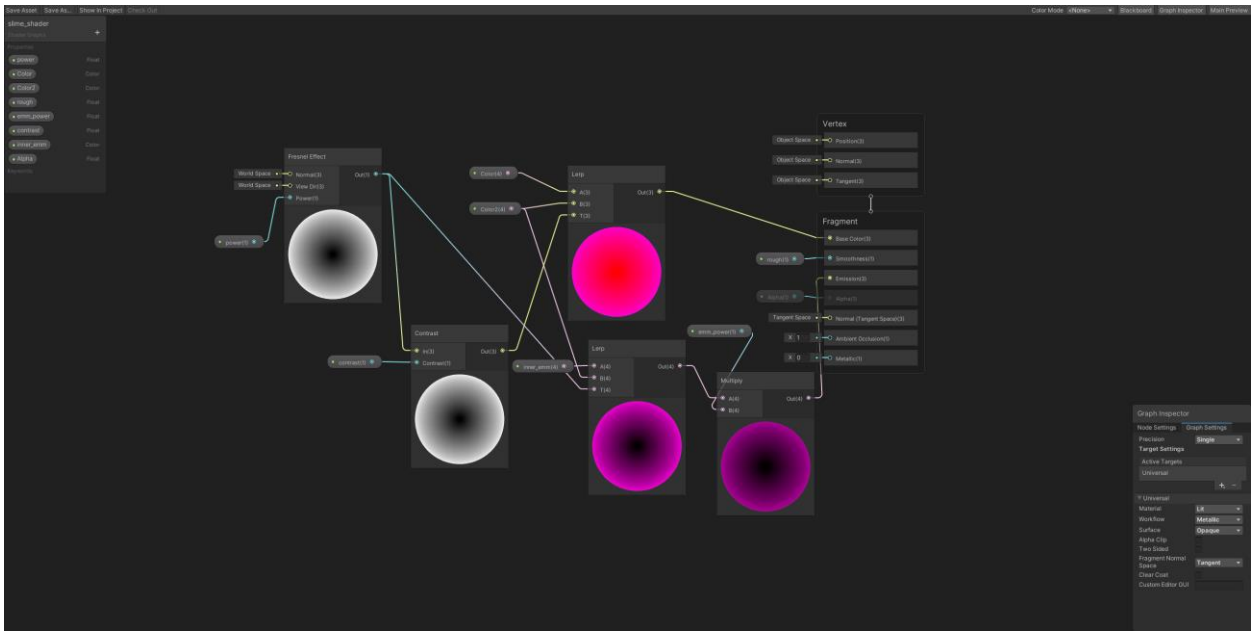


Рисунок 3.23 – Реалізація візуалу слайму у Shader Graph

Також важливим елементом гри э реалізація зон, зрозумілих гравцю для чого вони існують, не використовуючи текст. Основною метою шейдера є створення візуального ефекту, який одночасно виконує дві функції: ідентифікацію зони покупок та взаємодії з фермою слаймів. Шейдер повинен відразу привертати увагу гравця і давати зрозуміти, що перебування в цій зоні активізує певні ігрові механіки.

Шейдер розробляється таким чином, щоб краї зони покупки та взаємодії випромінювали сяйво, створюючи враження світіння, що обрамляє ділянку.

Додаваємо головні параметри для назначення кольорів та текстури шуму. Також цифромі параметри для швидкості зміни, плавності, розмір

еміссії. Це дозволяє адаптувати візуал під загальний стиль ігри та зробити його помітним, але не відволікаючим.

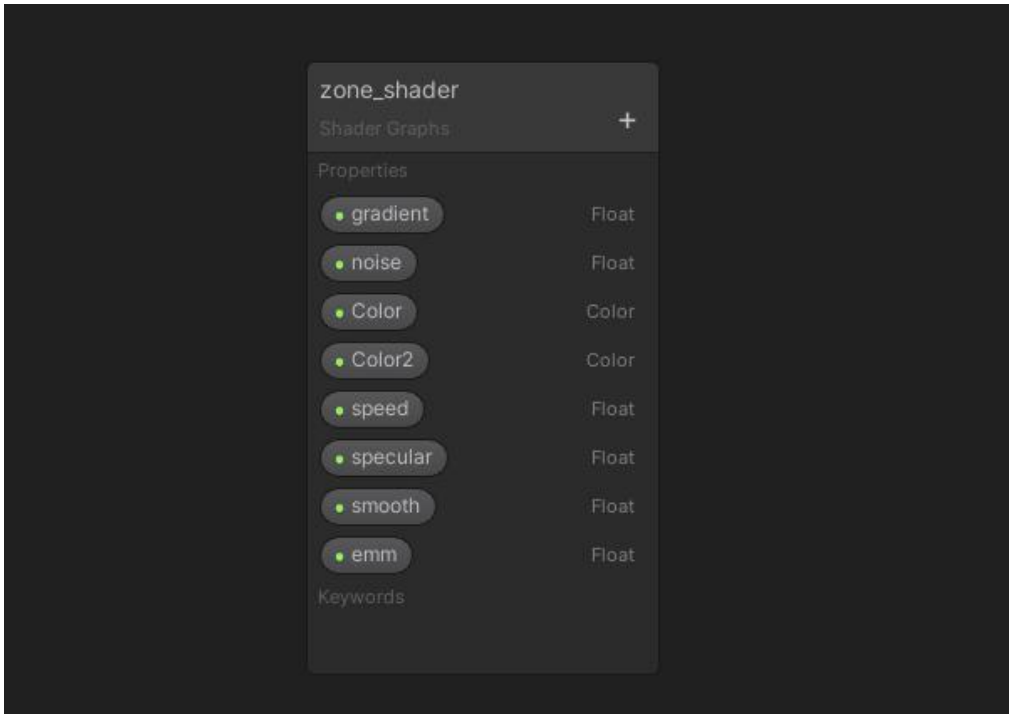



Рисунок 3.24 – Параметри налаштувань зони

Основними нодами шейдеру будуть градієнт та градієнтний шум. До них ми додаємо ноди комбінації кольорів та зміщення шуму з плином часу.

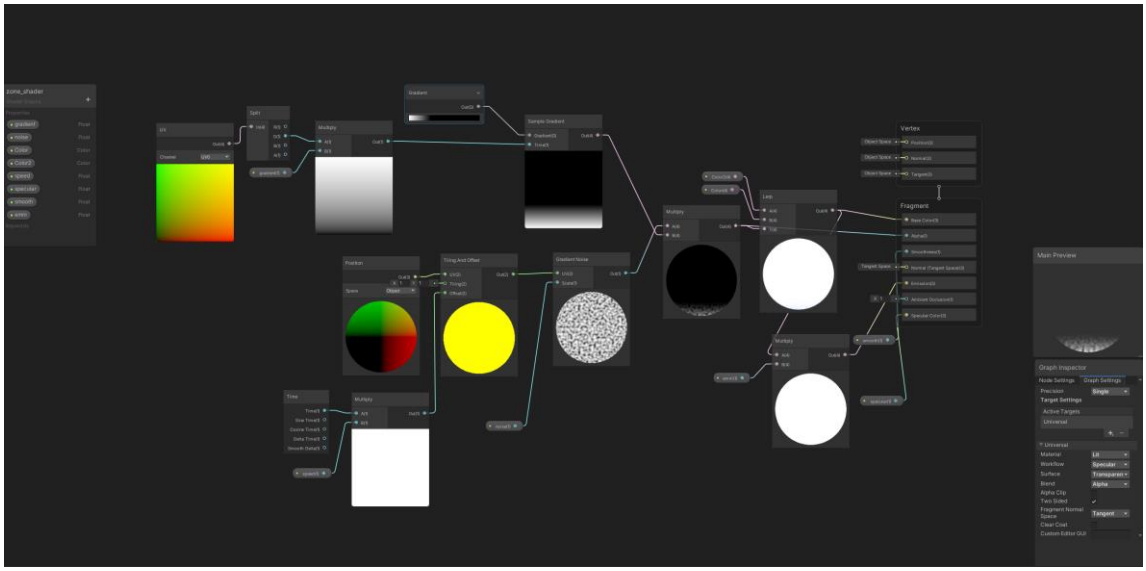


Рисунок 3.24 – Реалізація візуалу зони у Shader Graph

У процесі розробки мобільної гри одним з ключових аспектів було реалізація плавного та інтуїтивно зрозумілого управління персонажем. Для цього було використано Joystick Pack, який дозволив інтегрувати віртуальний джойстик у гру.

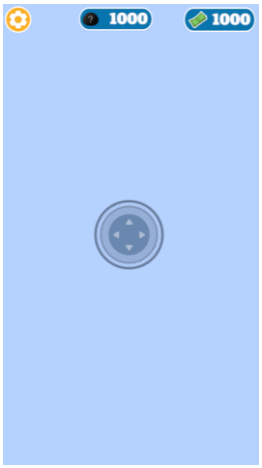



Рисунок 3.18 – Джойстик

Нижче представлено код скрипта JoystickForMovement, який активує джойстик та обробляє вхідні дані від користувача для управління пересуванням персонажа.

```
private void EnableJoystick()
{
    m_Joystick.gameObject.SetActive(true);
}

Event function  DaniilHiberry
private void Update()
{
    if(!m_CanMove)
        return;
    var direction:Vector2 = InputManager.Instance.JoystickDirection;
    m_Movement.MoveCharacter(moveDirection:new Vector3(-direction.x, y:0, z:-direction.y));

    if (direction.x != 0 || direction.y != 0)
        m_Movement.RotateCharacter(moveDirection:new Vector3(-direction.x, y:0, z:-direction.y));
}

2 usages  DaniilHiberry
private void CanMove(bool state)
{
    m_CanMove = !state;
    m_Joystick.gameObject.SetActive(!state);
}
```

Рисунок 3.19 – Обробка даних з джойстику

Основна логіка пересування та обертання персонажа виконується через скрипт CharacterMovement, який забезпечує реалізацію фізичної моделі руху, включаючи обробку гравітації. Рух персонажа відбувається шляхом

нормалізації вектора швидкості та використання контролера персонажа для його пересування у просторі, а обертання виконується через обчислення нового напрямку обличчям до напрямку руху. Переміщення виконується за допомогою Unity компонента CharacterController

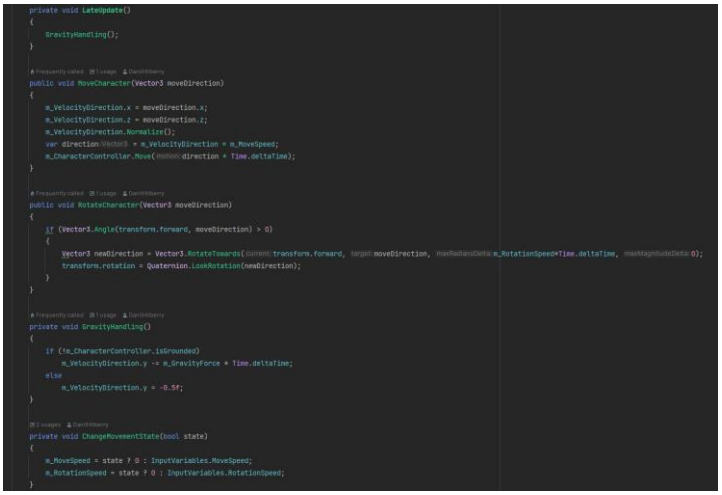



Рисунок 3.20 – Реалізація пересування гравця використовуючи вхідні дані з джойстику.

Цей код відповідає за переміщення слаймів у грі, використовуючи компонент NavMeshAgent для навігації. Коли гравець наближається на певну відстань, слайм активно уникає зустрічі, переміщаючись у протилежний бік. У випадку достатньої віддалі, слайм повертається до нормального стану і розпочинає випадковий рух в різних напрямках. Код також включає логіку для анімацій та зміни швидкості руху слаймів в залежності від їхнього стану.

```
Event function | DanilHitberry
private void Update()
{
    float distance = Vector3.Distance( a: transform.position, b: Character.transform.position);
    if (distance < SlimeDistanceRun)
    {
        Vector3 dirToChar = transform.position - Character.transform.position;
        Vector3 newPos =  transform.position + dirToChar;
        m_MeshAgent.SetDestination(newPos);

        if (isIdle)
        {
            if (m_MoveWonder != null)
                StopCoroutine(m_MoveWonder);

            m_MeshAgent.speed = m_RunSpeed;
            Animator.SetTrigger(c_Jump);
            isIdle = false;
        }
    }
    else
    {
        if (!isIdle)
        {
            Animator.SetTrigger(c_Idle);
            m_MeshAgent.speed = m_WalkSpeed;
            m_MoveWonder = StartCoroutine( routine: MoveWonderDirection());
            isIdle = true;
            return;
        }
        if (m_MoveWonder == null)
        {
            m_MeshAgent.speed = m_WalkSpeed;
            m_MoveWonder = StartCoroutine( routine: MoveWonderDirection());
        }
    }
}

Frequently called | 3 usages | DanilHitberry
private IEnumerator MoveWonderDirection()
{
    m_CurrentWonderTime = Random.Range(m_WonderTime.x,m_WonderTime.y);
    newPos = transform.position + new Vector3( x: Random.Range(-5,5), y: 0, z: Random.Range(-5,5));
    m_MeshAgent.SetDestination(newPos);
    yield return new WaitForSeconds(Random.Range(1, 2));
    while (m_CurrentWonderTime > 0)
    {
        m_CurrentWonderTime -= 0.1f;
        yield return new WaitForSeconds(0.1f);
    }
    WonderDirection();
}

Frequently called | 1 usage | new *
private void WonderDirection()
{
    StopCoroutine(m_MoveWonder);
    StartCoroutine( routine: MoveWonderDirection());
}
```

Рисунок 3.21 – Реалізація поведінки слаймів

У грі передбачена унікальна можливість злиття слаймів, що дозволяє гравцям створювати нові види з особливими характеристиками. Функціонал злиття реалізовано через метод MergeSlimeDrop, який обробляє інформацію про два слайми і створює з них один об’єднаний слайм з новими візуальними та ігровими властивостями. Створюється новий матеріал для зображення слайма, який включає градієнтний колір, формований від обох поєднаних слаймів, та налаштування візуальних ефектів.

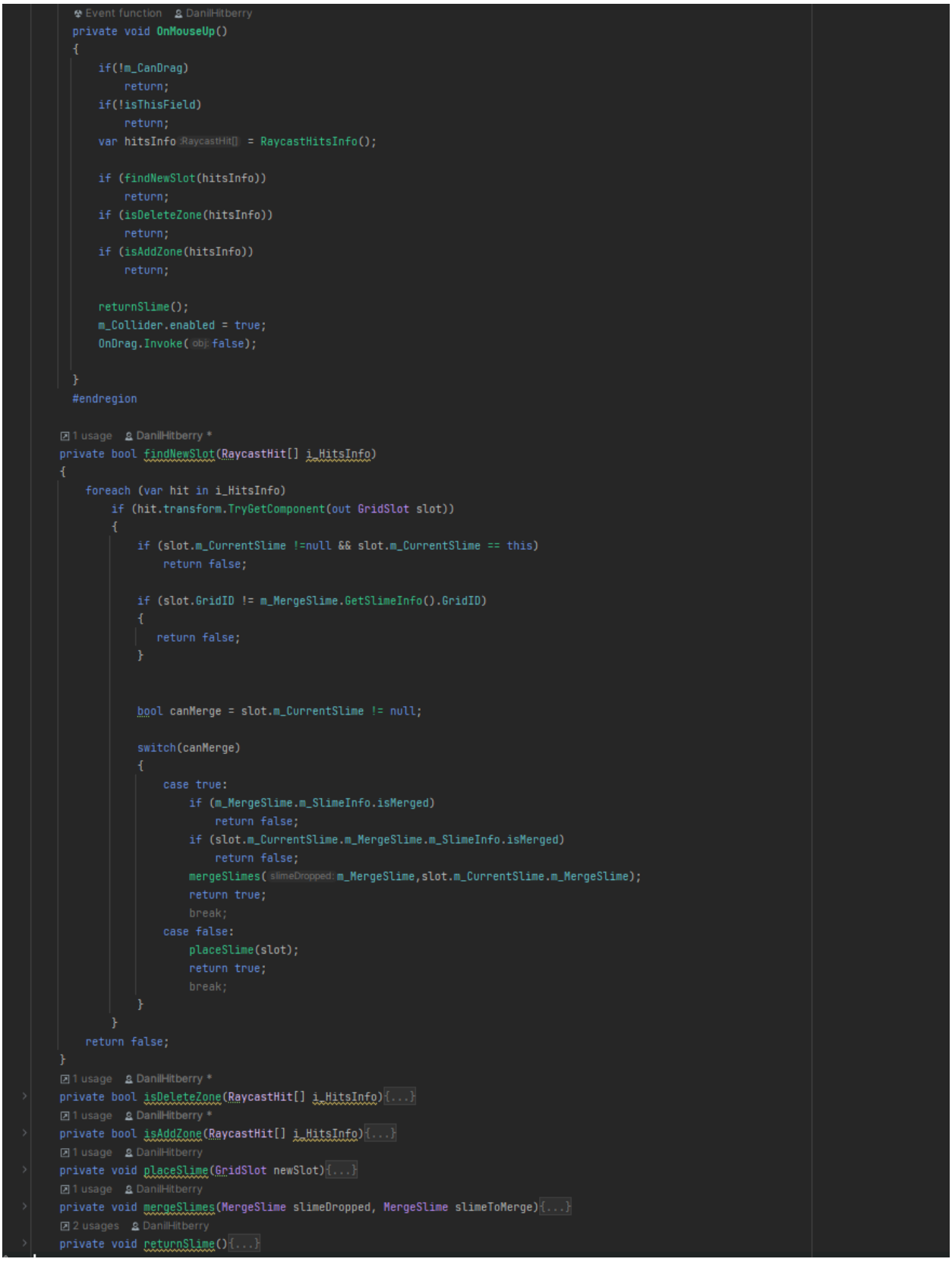
```
public void MergeSlimeDrop(SlimeInfo slimeInfo)
{
    m_SlimeInfo = new SlimeInfo(slimeInfo);

    var indexToActivateParent :int  = (int)m_SlimeInfo.e_SlimeMainType * m_SlimeTypesCount + m_SlimeInfo.SlimeMainIndex;
    var indexToActivateMerged :int  = (int)m_SlimeInfo.e_SlimeMergedType * m_SlimeTypesCount + m_SlimeInfo.SlimeMergedIndex;

    if(m_SlimeParts[indexToActivateParent]!=null)
        m_SlimeParts[indexToActivateParent].SetActive(true);
    if(m_SlimeParts[indexToActivateMerged]!=null)
        m_SlimeParts[indexToActivateMerged].SetActive(true);

    var tempDefaultMaterial = new Material(m_Renderers[0].materials[0]);
    tempDefaultMaterial.SetColor( name: c_Color1, m_SlimeColors[m_SlimeInfo.SlimeMainIndex]);
    tempDefaultMaterial.SetColor( name: c_Color2, slimeInfo.GradientColor);
    tempDefaultMaterial.SetFloat( name: "_Rough",  value: 0.6f);

    m_DefaultMaterial = tempDefaultMaterial;

    foreach (var meshRenderer in m_Renderers)
    {
        var tmp = new List<Material>( meshRenderer.materials);
        tmp[0].SetColor( name: c_Color1,m_SlimeColors[m_SlimeInfo.SlimeMainIndex]);
        tmp[0].SetColor( name: c_Color2,slimeInfo.GradientColor);
        meshRenderer.materials = tmp.ToArray();
    }
    m_SlimeIncome.TimeIncome(slimeInfo);
    m_DragDropSlime.isThisField = true;
}
```

Рисунок 3.22 – Злиття слаймів

Центральним елементом є логіка обробки дій гравця зі слаймами в сітці ферми.

Ключові функції включають перевірку можливості перетягування слайма, а також знаходження нового місця для слайма через метод findNewSlot. Цей метод використовує променевий пошук, тобто використання Raycast для ідентифікації слотів.

Також реалізовані зони спеціальних дій, такі як зона видалення, де слайми можуть бути видалені з ігрової сцени, та зона додавання, де слайми можуть бути повернуті в рюкзак гравця. Обидві ці зони реагують на перетягування слай ма в атктивну зону, і виконують відповідні функції.



```
Event function  DanilHitberry
private void OnMouseUp()
{
    if(!m_CanDrag)
        return;
    if(!isThisField)
        return;
    var hitsInfo RaycastHit[] = RaycastHitsInfo();

    if (findNewSlot(hitsInfo))
        return;
    if (isDeleteZone(hitsInfo))
        return;
    if (isAddZone(hitsInfo))
        return;

    returnSlime();
    m_Collider.enabled = true;
    OnDrag.Invoke( obj false);

}
#endregion

1 usage  DanilHitberry *
private bool findNewSlot(RaycastHit[] i_HitsInfo)
{
    foreach (var hit in i_HitsInfo)
        if (hit.transform.TryGetComponent(out GridSlot slot))
        {
            if (slot.m_CurrentSlime !=null && slot.m_CurrentSlime == this)
                return false;

            if (slot.GridID != m_MergeSlime.GetSlimeInfo().GridID)
            {
               return false;
            }


            bool canMerge = slot.m_CurrentSlime != null;

            switch(canMerge)
            {
                case true:
                    if (m_MergeSlime.m_SlimeInfo.isMerged)
                        return false;
                    if (slot.m_CurrentSlime.m_MergeSlime.m_SlimeInfo.isMerged)
                        return false;
                    mergeSlimes( slimeDropped: m_MergeSlime,slot.m_CurrentSlime.m_MergeSlime);
                    return true;
                    break;
                case false:
                    placeSlime(slot);
                    return true;
                    break;
            }
        }
    return false;
}
1 usage  DanilHitberry *
private bool isDeleteZone(RaycastHit[] i_HitsInfo){...}
1 usage  DanilHitberry *
private bool isAddZone(RaycastHit[] i_HitsInfo){...}
1 usage  DanilHitberry
private void placeSlime(GridSlot newSlot){...}
1 usage  DanilHitberry
private void mergeSlimes(MergeSlime slimeDropped, MergeSlime slimeToMerge){...}
2 usages  DanilHitberry
private void returnSlime(){...}
```

Рисунок 3.23 – Реалізація можливостей гравця в сітці

Коли гравець переходить у певний біом, усі інші менеджери спавну деактивуються, а менеджер спавну обраного біому активується. Залежно від типу переходу, визначеного в BiomTransitionUI.TransitionType, реалізується

логіка спавну слаймів. Якщо перехід відбувається додому, активується таймер для цього менеджера спавну, інакше запускається процес загрузки та спавну слаймів через CurrentBiomManager.

```
private void SpawnCurrentBiomSlimes(int biomID, BiomTransitionUI.TransitionType type)
{
    foreach (var manager in m_SpawnManagers)
        manager.BiomParent.gameObject.SetActive(false);

    m_SpawnManagers[biomID].BiomParent.gameObject.SetActive(true);
    if (type == BiomTransitionUI.TransitionType.Home)
    {
        Character.OnPlayerTeleportedHome.Invoke(biomID);
        m_SpawnManagers[biomID].StartTimer();
        return;
    }
    Character.OnTeleportedBiom.Invoke(biomID);
    CurrentBiomManager currentBiomManager = CurrentBiomManager.Instance;
    currentBiomManager.GetSlimes(m_SpawnManagers[biomID], m_SpawnManagers[biomID].SlimeList);
    currentBiomManager.SpawnSlimes();
}
```

Рисунок 3.24 – Реалізація генерації слаймів у біомах

# ВИСНОВКИ

У рамках кваліфікаційної роботи було підкреслено важливість ігрової індустрії в сучасному світі, проведено аналіз основних тенденцій та закономірностей цієї галузі. Описано процес створення гри для мобільного пристрою на рушії Unity 3D із використанням додаткового інструменту Shader Graph. Розглянуто альтернативи для застосування інших графічних редакторів та проаналізовано ефективність найпопулярніших 3D рушіїв. Детально висвітлено всі аспекти розробки, включаючи backend та frontend розробку, опис логіки гри та її дизайну. Проведено аналіз кількох ігор із подібними механіками та жанрами. Розглянуто процес обробки та моделювання, аналіз звуку та відео, написання коду та багато іншого.

Крім того, було проведено порівняльний аналіз різних підходів до розробки ігор, включаючи використання альтернативних рушіїв, таких як Unreal Engine та Godot. Оцінено продуктивність та зручність цих інструментів для розробки мобільних ігор. Описано методології управління проєктами, які застосовуються у процесі розробки, та проаналізовано їх вплив на кінцевий результат. Розглянуто аспекти монетизації ігор та їх маркетингове просування, а також вплив різних стратегій на успішність гри на ринку.

Результатом роботи стала повноцінна гра під назвою «Slime Farmer», розрахована на безкоштовний геймінг на телефонах.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Unity 3D. URL: https://en.wikipedia.org/wiki/Unity_(game_engine) (дата звернення 12.05.2024).

2. JOSEPH HOCKING Unity in action Multiplatform game development in C# Second Edition: Shelter Island, 2018. – 402 p.

3. Unity at 10: For better—or worse—game development has never been easier. URL: https://arstechnica.com/gaming/2016/09/unity-at-10-for-better-or-worse-game-development-has-never-been-easier (дата звернення 12.05.2024).

4. How new graphics effects can make Unity Engine games look less generic. URL: https://arstechnica.com/gaming/2016/03/how-new-graphics-effects-can-make-unity-engine-games-look-less-generic/ (дата звернення 11.05.2024).

5. Shader Graph. URL: https://unity.com/features/shader-graph (дата звернення 10.05.2024).

6. Shader Graph. URL: https://medium.com/game-dev-channel/work-with-shader-graph-300a8c55025f# (дата звернення 12.05.2024).

7. Top 56 3D game engines compared. URL: https://www.dragonflydb.io/game-dev/engines/3d (Дата звернення 12.05.2024).

8. Top programming languages for game developing URL: https://forum.freecodecamp.org/t/top-programming-languages-for-game-development/639986/4 (дата звернення 10.05.2024).

9. Unity Shader Graph Basics (Part 1 - Your First Graph) URL: https://danielilett.com/2023-09-26-tut7-3-intro-to-shader-graph/ (дата звернення 10.05.2024).

10. What is a shader graph in Unity? URL: https://www.educative.io/answers/what-is-a-shader-graph-in-unity (дата звернення 10.05.2024).

11. Component-based Architecture. URL: https://www.tutorialspoint.com/software_architecture_design/component_based_architecture.htm (дата звернення 11.05.2024).

12. What is Component-Based Architecture? https://www.mendix.com/blog/what-is-component-based-architecture/# (дата звернення 12.05.2024).

13. Exploring the Benefits and Limitations of Component-Based Software Architecture. URL: https://moldstud.com/articles/p-exploring-the-benefits-and-limitations-of-component-based-software-architecture (дата звернення 11.05.2024).

14. Mastering Component-Based Architecture: Everything You Need to Know. URL: https://nandbox.com/all-you-need-to-know-about-component-based-architecture/# (дата звернення 10.05.2024).

15. Kamil Abramowicz, Comparative analysis of the performance of Unity and Unreal Engine game engines in 3D games: Lublin University of Technology, 2024m 8 pp.

16. X.Cui, H.Shi, A*-based pathfinding in modern computer games, International Journal of Computer Science and Network Security 11(1) (2011) p 125–130.

17. Unity vs Unreal Engine: What Works For You? URL: https://www.incredibuild.com/blog/unity-vs-unreal-what-kind-of-game-dev-are-you (дата звернення 11.05.2024).

18. My mini mart APK. URL: https://my-mini-mart.en.softonic.com/ (дата звернення 11.05.2024).

19. Is My Mini Mart safe for kids? An honest My Mini Mart review for parents. URL: https://www.bark.us/app-overview/my-mini-mart/ (дата звернення 11.05.2024).

20. Under the hood: Deconstructing one of the top idle arcade games URL: https://www.pocketgamer.biz/feature/81292 (Дата звернення 11.05.2024).

21. Slime Rancher. URL: https://en.wikipedia.org/wiki/Slime_Rancher (Дата звернення 11.05.2024).

22. Are there benefits of video games for kids URL: https://intermountainhealthcare.org/blogs/are-there-benefits-of-video-games-for-kids (дата звернення 07.05.2024).

23. Future of virtual reality in everyday life. URL: https://www.lsbuk.com/future-of-virtual-reality-vr-in-everyday-life/ (дата звернення 09.05.2024).

24. State of video games. URL: https://variety.com/vip-special-reports/state-of-video-games-a-special-report-1235884124/ (дата звернення 22.04.2024).

25. Mobile gaming VS PC gaming, key differences. URL: https://whimsygames.co/blog/mobile-gaming-vs-pc-gaming-key-differences/ (дата звернення 18.05.2024).