МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА ANDROID ЗАСТОСУНКУ ЗАСОБАМИ UNITY ТА C#»**

Виконав: студент 2 курсу, групи 8.1212-з
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
Р.В. Сергєєв
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, доцент, к.ф.-м.н. Кудін О.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент доцент кафедри комп'ютерних наук, доцент, к.т.н. Матвіїшина Н.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Сергєєву Роману Володимировичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка Android застосунку засобами Unity та C#

керівник роботи Кудін Олексій Володимирович, к.ф.-м.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 01 » травня 2023 року № 643-с

2. Строк подання студентом роботи 27.11.2023 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Розробка Android застосунку засобами Unity та C#.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 03.05.2023 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	12.05.2023	
2.	Збір вихідних даних.	07.06.2023	
3.	Обробка методичних та теоретичних	21.06.2023	
	джерел.		
4.	Розробка першого та другого розділу.	23.08.2023	
5.	Розробка третього розділу.	27.10.2023	
6.	Оформлення та нормоконтроль	20.11.2023	
	кваліфікаційної роботи магістра.		
7.	Захист кваліфікаційної роботи.	15.12.2023	

Студент _____ Р.В. Сергєєв
(підпис) (ініціали та прізвище)

Керівник роботи _____ О.В. Кудін
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка Android застосунку засобами Unity та C#»: 71 с., 56 рис., 1 табл., 27 джерел.

АЛГОРИТМ, БАЗА ДАНИХ, ГРА НА ВИЖИВАННЯ, ЗАСТОСУНОК, ІГРОВИЙ ПРОЦЕС, ІНТЕРФЕЙС, КОНТРОЛЕР, ПЕРСОНАЖ, ПРОГРАМА, РУШІЙ, ХМАРА, ШТУЧНИЙ ІНТЕЛЕКТ, 3Д РЕДАКТОР, BLENDER, UNITY.

Об’єкт дослідження – мобільна гра на виживання.

Предмет дослідження – процес розробки, реалізації та випробування мобільної гри на виживання.

Мета роботи: вивчити сучасні технологій та підходи до розробки мобільних ігор, застосувати їх при створення захоплюючої мобільної гри на виживання.

Метод дослідження – збір та аналіз інформації, статистичний аналіз, експертні опитування, порівняння, спостереження за гравцями, аналіз рецензій та відгуків.

У магістерській кваліфікаційній роботі було розглянуто тенденції розвитку сучасних мобільних ігор та їх актуальність. Вивчено особливості геймплею та механіки подібних проєктів. Також було порівняно та обрано популярні інструменти для реалізації мобільної гри.

Було розроблено концепцію гри, написано алгоритм взаємодії компонентів у середині застосунку, а також безпосередньо з гравцем. Мобільну гру на виживання реалізовано за допомогою рушія Unity з використанням мови програмування C#. Додаток було протестовано та опубліковано на сервісі Google Play Market.

Цю роботу в подальшому можна застосувати як основу для побудови статистичної моделі поведінки людини в екстремальних ситуаціях.

SUMMARY

Master’s qualifying paper «Development of the Android Application Using Unity and C#»: 71 pages, 56 figures, 1 table, 27 references.

ALGORITHM, DATABASE, SURVIVAL GAME, APPLICATION, GAMEPLAY, INTERFACE, CONTROLLER, CHARACTER, PROGRAM, ENGINE, CLOUD, ARTIFICIAL INTELLIGENCE, 3D EDITOR, BLENDER, UNITY.

The object of the study is a mobile survival game.

The subject of research is the study is the process of development, implementation and testing of the mobile survival game.

The aim of the work is to study modern technologies and approaches to the development of mobile games, to implement them in the implementation of the creation of an exciting mobile survival game.

The methods of research are collection and analysis of information, statistical analysis, expert surveys, comparisons, observation of players, analysis of reviews and feedback.

In the Master’s qualification work, the development trends of modern mobile games and their relevance were considered. Peculiarities of gameplay and mechanics of similar projects have been studied. Popular tools for implementing a mobile game were also compared and selected.

The game concept was developed, the algorithm for interaction between components and player inside the application was written as well. The mobile survival game is implemented through the Unity engine with using of C# programming language. The application was tested and published on the Google Play Market service.

In the future, this game can be used for building of a basic statistical model of human behavior in extreme situations.

ЗМІСТ

Завдання на кваліфікаційну роботу 2
Реферат 4
Summary 5
Скорочення та умовні познаки 8
Вступ 9
1 Огляд і аналіз предметної області 11
1.1 Тенденції розвитку індустрії розваг 11
1.2 Огляд інструментів розробки 15
1.2.1 Вибір рушія 15
1.2.2 Вибір 3Д редактора 18
1.3 Збір та аналіз вимог 20
1.4 Висновки до розділу 1 23
2 Розробка алгоритму та компонентів проєкту 24
2.1 Концепція гри 24
2.2 Основні компоненти проєкту 26
2.3 Розробка алгоритму штучного інтелекту 32
3 Реалізація та тестування проєкту 37
3.1 Створення локації та інтерфейсу гри 37
3.2 Створення 3Д об’єктів 40
3.3 Програмна реалізація проєкту 45
3.3.1 Збереження та зчитування даних 45
3.3.2 Дії з предметами 48
3.3.3 Пошук та підняття предметів 50
3.3.4 Розпалення багаття 52
3.4 Застосування хмарних сервісів у проєкті 55
3.5 Оптимізація застосунку 58
3.6 Тестування готового продукту 61

Висновки 68
Перелік посилань..... 69

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

БД	База даних
FPS	Frames Per Second – кількість кадрів в секунду
Крафт	З англійської craft – ремесло, створення предмету власноруч
Корутина	Співпрограма або блок коду для асинхронного виконання
Рендеринг	Процес візуалізації, що виконується за допомогою програми
Скрипт	Невелика програма, яка виконує певну задачу
ШІ	Штучний інтелект

ВСТУП

Мобільні ігри стали невід’ємними супутниками нашого повсякденного життя, займаючи важливе місце в сфері розваг та відпочинку. Із стрімким розвитком технологій та поширенням смартфонів, популярність мобільних ігор зростає, пропонуючи гравцям різноманітні жанри та концепції гри. Один з таких жанрів – мобільна гра на виживання, що викликає інтерес користувачів та вимагає від них великої уваги та активності.

Кваліфікаційна робота присвячена дослідженню та розробці мобільної гри на виживання, яка спрямована на вивчення основних аспектів виживання в умовах обмежених ресурсів та складних ситуацій. Робота включає аналіз сучасних тенденцій у галузі мобільних ігор, огляд схожих проєктів, а також розробку та реалізацію концепції мобільної гри на виживання з використанням передових технологій та креативних рішень.

Тема "Мобільна гра на виживання" є надзвичайно актуальною через стрімкий ріст популярності мобільних ігор, постійний інтерес до жанру виживання, технологічний прогрес у мобільних пристроях та конкурентну боротьбу на ринку ігор.

Метою цього проєкту є:

- розробка та створення захоплюючої мобільної гри на виживання, яка привертає увагу гравців та надихає їх на взаємодію з ігровим світом;
- вивчення та впровадження сучасних технологій та підходів до розробки мобільних ігор з акцентом на жанр виживання;
- дослідження популярних механік та елементів гри на виживання з метою створення унікального та привабливого геймплею;
- підвищення рівня якості та цікавості мобільних ігор на ринку шляхом створення високоякісного продукту.

За допомогою інноваційних рішень та унікального підходу до геймдизайну, кваліфікаційна робота має на меті розкрити важливі аспекти

створення мобільних ігор на виживання та внести свій внесок у цю захоплюючу галузь розваг.

Об’єктом роботи обрано мобільну гру на виживання розглядаючи наступні аспекти:

– особистий інтерес (як розробник, маю особистий інтерес до ігор на виживання та відчуваю пристрасть до створення захоплюючих геймплеїв, які надихають гравців);
– популярність жанру (жанр виживання завжди викликав великий інтерес у гравців, а тому розробка гри в цьому жанрі повинна привернути велику увагу аудиторії);
– ринковий попит (мобільні ігри на виживання мають стабільний попит на ринку розважальних додатків);
– технічні можливості (сучасні технології дозволяють створювати складні та захоплюючі ігри з високоякісною графікою та функціоналом, що відкриває нові можливості для розробки захоплюючих геймплеїв).

Предметом дослідження є процес розробки, реалізації та випробування мобільної гри на виживання. Включає в себе вивчення особливостей геймплею, графічного оформлення і взаємодії користувача з грою.

Завданнями кваліфікаційної роботи є:

– провести аналіз популярних мобільних ігор на виживання та визначити їхні основні характеристики та особливості геймплею;
– розробити концепцію гри, включаючи сценарій, персонажів, рівні складності та геймплейні механіки;
– створити програмний код гри та графічне оформлення, враховуючи вимоги до мобільних платформ;
– провести тестування гри з метою виявлення та виправлення помилок, а також оптимізації геймплею;
– забезпечити можливість гравцям випробувати гру та зібрати їхні відгуки для подальшого вдосконалення.

1 ОГЛЯД І АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Тенденції розвитку індустрії розваг

Розваги – це невід'ємна частина життя, одна з потреб людини. Тому не дивно, що бізнесмени прагнуть заробити на цьому. Та і світ розваг в свою чергу не стоїть на місці, він розвивається.

Ще в далекі 80-і роки діти відмовилися від спортивних м'ячів, іграшок та велосипедів у своїх запитах на подарунки та почали просити відеоігри. Цей технологічний прогрес повністю переосмислив розваги. З кожним днем все більше компаній вважають геймерів своєю цільовою аудиторією, і це не дарма. Діти 80-х тепер стали незалежними дорослими людьми з роботою, які люблять відеоігри. Тож світ відеоігор охоплює різні вікові категорії, а не тільки дітей [1].

Зараз те, як люди знаходять і споживають розваги, кардинально відрізняється від того, які були навіть кілька років тому. Наприклад, захоплюючі технології, такі як доповнена реальність, віртуальна реальність та інший інтерактивний вміст, максимально залучають гравців [2].

Індустрія відеоігор розвивається з неймовірною швидкістю в останні роки, і ні для кого не секрет, що це прибутковий бізнес. Але ви можете бути здивовані, дізнавшись, скільки грошей заробляє ігрова індустрія порівняно з іншими галузями розваг. Згідно з останніми даними, ігрова індустрія зараз заробляє більше грошей, ніж музична та кіноіндустрія разом узяті.

Згідно зі звітом SuperData Research, у 2020 році світовий ігровий ринок оцінювався в 159,3 мільярда доларів США. Це включає дохід від консольних ігор, ігор для ПК, мобільних ігор та кіберспорту. Щоб поглянути на це в перспективі, у 2020 році музична індустрія оцінювалася в 19,1 мільярда доларів, тоді як кіноіндустрія оцінювалася в 41,7 мільярда доларів. Це означає, що ігрова індустрія заробляє у вісім разів більше, ніж музична індустрія, і

майже в чотири рази більше, ніж кіноіндустрія.

Однією з головних причин успіху ігрової індустрії є зростання популярності мобільних ігор. У 2020 році на мобільні ігри припадало більше половини світового ігрового ринку з доходом у 86,1 мільярда доларів. Зростання кількості смартфонів і планшетів полегшило людям доступ до ігор і можливість грати в них, а збільшення доступності високошвидкісного Інтернету зробило мобільні ігри більш складними та привабливими [3].

Причини успіху ігрової індустрії:

– технологічні досягнення;
– інтерактивний і соціальний досвід;
– різноманітний контент і доступність;
– кіберспорт та професіоналізація ігор;
– конвергенція індустрії розваг.

Розглянемо далі причини успіху ігрової індустрії детальніше.

Одним із головних факторів стрімкого зростання ігор як форми розваги є швидкі темпи технологічного прогресу. Розвиток високошвидкісного Інтернету, потужних ігрових консолей і передової графіки дозволив створити захоплюючий ігровий досвід, який конкурує з традиційними формами розваг. Ці досягнення очевидні в іграх казино. Наприклад, ви можете грати в ігри онлайн-казино в Нью-Джерсі, просто натиснувши кнопку. Віртуальна реальність (VR), доповнена реальність (AR) і хмарні ігри – це лише деякі приклади новаторських технологій, які трансформують ігрову індустрію. Ці інновації дозволяють гравцям увійти у повністю реалізовані віртуальні світи та взаємодіяти зі своїм середовищем способами, які колись вважалися неможливими. Оскільки технології продовжують розвиватися, ігри будуть ставати все більш витонченими та захоплюючими, зміцнюючи своє місце як майбутнього розваг.

Одним із найпривабливіших аспектів ігор є притаманна їм інтерактивність. На відміну від пасивних форм розваги, таких як перегляд фільмів або прослуховування музики, ігри дозволяють гравцям активно

взаємодіяти з вмістом і впливати на результат досвіду. Це почуття волі та контролю може зробити гру більш задовільною та насиченою формою розваги. Крім того, розвиток онлайн-ігор і багатокористувацьких платформ перетворив ігри на соціальну діяльність. Тепер гравці можуть спілкуватися з друзями або незнайомцями з усього світу, щоб співпрацювати, змагатися або просто спілкуватися, насолоджуючись улюбленими іграми. Ця здатність налагоджувати зв'язки та створювати спільноти навколо спільних інтересів робить ігри неймовірно привабливою формою розваги.

Ігрова індустрія розвинулася, щоб задовольнити широкий спектр уподобань і стилів гри, пропонуючи щось для кожного. Від насичених діями шутерів від першої особи та складних рольових ігор до розслаблюючих ігор-головоломок і навчальних ігор, є гра на будь-який смак. Крім того, ігри стають дедалі доступнішими, оскільки такі платформи, як мобільні пристрої, консолі та ПК, пропонують різноманітні цінові категорії та початкові рівні для гравців. Звичайні геймери можуть насолоджуватися безкоштовними іграми на своїх смартфонах, а віддані ентузіасти можуть інвестувати в потужні ігрові установки та зануритися в останні випуски ААА. Цей різноманітний діапазон контенту та доступність гарантує, що ігри можуть зацікавити широку аудиторію та зберегти свою позицію домінуючої форми розваги.

Поява кіберспорту ще більше зміцнила статус ігор як майбутнього розваг. Змагальні ігри перетворилися на мільярдну індустрію, коли професійні гравці, команди та ліги залучають величезну аудиторію та залучають значні інвестиції від спонсорів. Великі кіберспортивні події, такі як Чемпіонат світу з гри League of Legends, тепер конкурують із традиційними спортивними подіями за кількістю глядачів і грошовими призами. Зростання кіберспорту не тільки розширило охоплення ігор, але й узаконило їх як форму розваги та життєздатну кар'єру для талановитих гравців.

Іншим фактором, що сприяє домінуванню ігор у сфері розваг, є конвергенція різних індустрій розваг. Традиційні медіакомпанії, такі як Disney і Netflix, зараз інвестують в ігрові об'єкти та досліджують можливості створення

інтерактивного досвіду на основі своїх популярних франшиз. І навпаки, успішні відеоігри адаптуються до фільмів, телевізійних серіалів та інших форм медіа, ще більше стираючи межі між іграми та традиційними розвагами. Така конвергенція галузей підкреслює зростаючу важливість ігор в екосистемі розваг і їх потенціал змінити спосіб споживання контенту [4].

Потокове відео, соціальні мережі та ігри допомагають створювати нові бізнес-моделі та змінювати медіа та розваги. Але реальна історія 2023 року полягає в тому, що ці три сектори дедалі більше стають взаємозалежними як частина ширшої та багатшої екосистеми медіа та розваг [5].

Глобальна індустрія відеоігор – це мільярдний бізнес. Велика трійка в індустрії відеоігор – Nintendo, Microsoft і Sony. Велика трійка є найбільшими видавцями відеоігор у всьому світі, заробляючи на відеоіграх мільярди доларів США щороку. Згідно з останніми даними, 29 відсотків від загального доходу ігрової індустрії припадає на «велику трійку». Успіх в ігровій індустрії не обмежується лише власниками платформ. Прибуток від мобільних ігор навіть не обмежується публікацією мобільних ігор – незважаючи на те, що Apple і Google не є традиційними ігровими компаніями, Apple і Google також заробляють багато грошей на ігрових додатках [6].

В Україні ж дедалі більше інді-розробників випускають свої ігри. За рік цей показник зріс на 66,9%.

Розробники Unity випустили свою щорічну аналітику про властивості й тренди ігрової індустрії, та перелічили основні тренди останнього року в ігровій галузі:

- інді-розробники випускають ігри досить швидко, а розробники працюють менше годин (у 2022 році 62% інді-ігор вийшли менш ніж за рік);
- виходить більше ігор, які призначені винятково для мобільних пристроїв;
- великі студії збільшують кількість багатоплатформних ігор (у 2022-му їх виходило на 16% більше, ніж у 2021-му);

– у мобільні ігри грають більше людей, ніж у 2021 році (щоденна кількість активних користувачів у світі зросла на 8% для середньої гри);
– тривалість «життя» ігор зростає з року в рік (студії працюють на довгострокову перспективу) [7].

Ігри стали майбутнім розваг завдяки їх захоплюючій природі, технологічному прогресу, інтерактивності, соціальному досвіду, різноманітному контенту, доступності, професіоналізації через кіберспорт і конвергенції індустрій розваг. Оскільки технології продовжують розвиватися, а ігрова аудиторія стає все більш різноманітною, індустрія ставатиме все більш впливовою та невід'ємною частиною нашого дозвілля [4].

1.2 Огляд інструментів розробки

1.2.1 Вибір рушія

Розглянемо 3 рушія, які очолюють рейтинг у розробці ігор: Unity [8], Unreal Engine [9], GameMaker Studio [10].

Unity (Unity Technologies) з'явився вперше в 2005 році, як рушій лише для Mac OS X, тепер Unity підтримується як Windows, так і Linux, а також відомий своєю перехресною платформою для всіх мобільних, консольних платформ та навіть AR/VR. Це найпопулярніший двигун серед розробників ігор, як серед інді, так і великих студій. Рушій орієнтований на ідею "доступності": він має досить низький поріг входу, його легко освоїти, він безкоштовний для незалежних розробників. В Unity існує магазин готових ассетів та плагінів. Це дозволяє розробляти проєкти швидше і з меншими витратами (що, погодьтеся, дуже важливо для стартапу). Також програма має повноцінний графічний редактор, що дозволяє малювати карти, локації, розставляти персонажів. До прийнятного вигляду їх доводять у Photoshop. При створенні Unity 3д-гри

можна імпортувати 3D-моделі з більшості сторонніх редакторів, що полегшує процес роботи.

Це повноцінний рушій спрямований на створення гри в одному редакторі. Безліч популярних мобільних ігрових продуктів створені саме на цьому рушії: Hearthstone: Heroes of Warcraft, Age of Magic, Royal Blood, Among Us, Escape from Tarkov та інші.

Тому Unity підходить розробникам, які ще не «розжилися» великою командою, але вже готові взяти на себе більшість процесів.

Плюси рушія Unity:

- зрозумілий редактор та інструментарій: за декілька днів основні речі може освоїти навіть той, хто вперше стикається з розробкою мобільного додатка;
- сучасний рівень графіки, здатний конкурувати з більш дорогими рушіями (Unity, безумовно, програє UnrealEngine за можливостями, але радує deferred освітленням, стандартним набором постпроцесингових ефектів, SSAO (Англ. Screen space ambient occlusion – заломлення світла в екранному просторі), прискореної опрацюванням лайтмапів);
- ігровий рушій Unity надається умовно безкоштовно (платити потрібно буде тільки за розширення пакетів підписки);
- велике ком’юніті розробників, безліч випущених ігор;
- внутрішній Asset Store, де можна купити готові фрагменти коду, асети і звуки.
- можливість створення фотореалістичної графіки;
- розробка на Юніті дозволяє легко імпортувати між ОС Windows, Linux, OS X, Android, iOS, на консолі PlayStation, Xbox, Nintendo, на VR- і AR-пристрої.

Мінуси середовища розробки Unity:

- розробка гри на Unity вимагає навичок програмування;
- безліч вбудованих компонентів роблять продукт об’ємним;

– у розробників немає доступу до вихідного коду власної гри;
– немає інтеграції із зовнішніми сервісами та бібліотеками (наприклад, Facebook), розробники змушені налаштовувати це вручну;
– неможливість додати до рушія сторонню фізику, або SpeedTree.

Unreal Engine (Epic Games). Названий на честь гри 1998 року, в якій він і був вперше використаний, Unreal Engine з роками все більше знижував ліцензійні збори, тому тепер доступний практично для кожного, хто хоче створювати на ньому ігри. Тим не менш, частіше він використовується все-таки для проєктів AAA.

У ньому закладено практично той же інструментарій, що і в Unity: робота з фізикою, 3D-графікою і не тільки, але існують і деякі інші рішення, здатні схилити розробників на його користь. Втім, для цього вже потрібен певний рівень скіллів. Це потужний рушій для створення високореалістичних ігор «з коробки», що підтримує швидке прототипування та візуалізований кодинг, а також має велику кастомізацію. Його широко використовують не тільки в ПК, консольній та мобільній ігровій розробці, а й поза геймдевом: наприклад, у кіно, архітектурі та автомобільній промисловості. Приклади тайтлів: Battlegrounds, Sea of Thieves, Fortnite, Final Fantasy VII Remake, Dead by Daylight, BioShock: Infinite, Star Wars Jedi: Fallen Order.

Переваги рушія Unreal Engine:

– увесь код проєкту пишеться на C++, що робить його дуже швидким, є вбудований редактор Blueprints;
– програмне забезпечення максимально стабільне з відсутністю багів;
– добре підходить як для створення ігор, так і кінематографічних спец ефектів;
– багатоплатформений.

Недоліки рушія Unreal Engine:

– високі вимоги до навичок розробника;
– рушій орієнтовано на потужні високовиробничі системи, що складає труднощі при оптимізації для мобільних пристроїв;

– великий об’єм програмного забезпечення;

– висока ціна на контент для геймдева.

GameMaker Studio (YoYo Games). Випущений в 1999 році, GameMaker орієнтований на розробників-початківців і володіє інтуїтивно зрозумілим Drag & Drop: для його використання немає необхідності написання будь-яких скриптів і тонн коду, як і взагалі знання мов програмування. Готову гру можна відразу експортувати до Steam.

У ньому немає таких можливостей для роботи з 3D, як у Unity та Unreal: натомість він фокусується на 2D-іграх. Інший недолік – висока ціна під час роботи з кількома платформами.

Зважаючи на те, що гра розрахована для роботи на мобільних пристроях для розробки проєкту було обрано рушій Unity, який дає змогу працювати застосунку з меншими вимогами до параметрів пристроїв. В ньому використовується C# для написання скриптів, що робить це легшим ніж на C++ в Unreal Engine.

1.2.2 Вибір 3Д редактора

Для реалізації проєкту нам потрібен 3д редактор. Розглянемо найпоширеніші з них та зведемо дані про них у таблицю 1.1.

Розглянувши функціонал, переваги та недоліки кожного з 3д редакторів для реалізації проєкту було обрано програму Blender [11], враховуючи його доступність, інтуїтивно зрозумілий інтерфейс та широкий спектр способів створення та редагування 3Д об’єктів.

Таблиця 1.1 – Програмні засоби для створення 3Д-об’єктів

ПЗ	Autodesk 3DsMax	Cinema 4d	Autodesk Maya	Blender
Сайт	https://www.autodesk.com/products/3ds-max/overview	https://www.maxon.net/en/cinema-4d	https://www.autodesk.com/products/maya/overview	https://www.blender.org
Розробник	Autodesk	Maxton Computer GmbH	Autodesk	Blender Foundation
Підтримка ОС	Microsoft Windows 7 та вище	Microsoft Windows, Mac OS	Linux, Microsoft Windows, Mac OS	Linux, Microsoft Windows, Mac OS, Solaris, FreeBSD, OpenBSD, Irix
Вартість	Безкоштовна тільки демо-версія			Безкоштовна
Основні можливості	Використовується моделювання на основі полігонів, сплайнів і NURBS. Має багато інформації та додатків.	Дозволяє рендерити об’єкти за методом Гуро. Підтримує анімації і високоякісний рендеринг. Простий інтерфейс.	Має широкі можливості для створення 3Д анімації, графіки руху і візуальних ефектів.	Включає засоби моделювання скульптингу, анімації, симуляції, рендерингу, пост обробки і монтажу відео зі звуком, компонування за допомогою «вузлів», а також створення 2Д анімації.
Сфера застосування	Здебільш застосовують в архітектурній візуалізації.	Часто використовують у сфері моушен, реклами.	Використовується зазвичай професіоналами та великими компаніями для створення анімації, мультфільмів, реклами, розробки ігор.	Застосовується для створення анімаційних фільмів, витворів мистецтв, моделей на основі 3д друку, візуальних ефектів, відеоігор.
Переваги	Величезний функціонал, безліч плагінів і навчальної інформації, гнучке керування частками, фотореалістична візуалізація.	Легкий в освоєнні, інтуїтивний інтерфейс, відмінний функціонал, безліч навчальних матеріалів, зручні інструменти моделювання, система фотореалістичної візуалізації.	Величезний функціонал і можливості, може моделювати фізику твердих часток, прораховувати поведінку тканини, рідкі ефекти.	Доступність, відкритий код, кросплатформеність, невеликий розмір, широкий функціонал, можливість створення ігор, постійно оновлюється, не потребує потужних ресурсів.
Недоліки	Важкий в освоєнні, потребує високих технічних можливостей, громіздкий інтерфейс.	Неналагоджена система переходу між версіями.	Тривале і складне навчання, високі вимоги до системи, висока ціна.	Відсутність документації в базовому постачанні.

1.3 Збір та аналіз вимог

Було проведено опитування 60 осіб на те, в який жанр ігор вони б скоріш зіграли. Отримані результати зведено до діаграми (див. рис. 1.1).

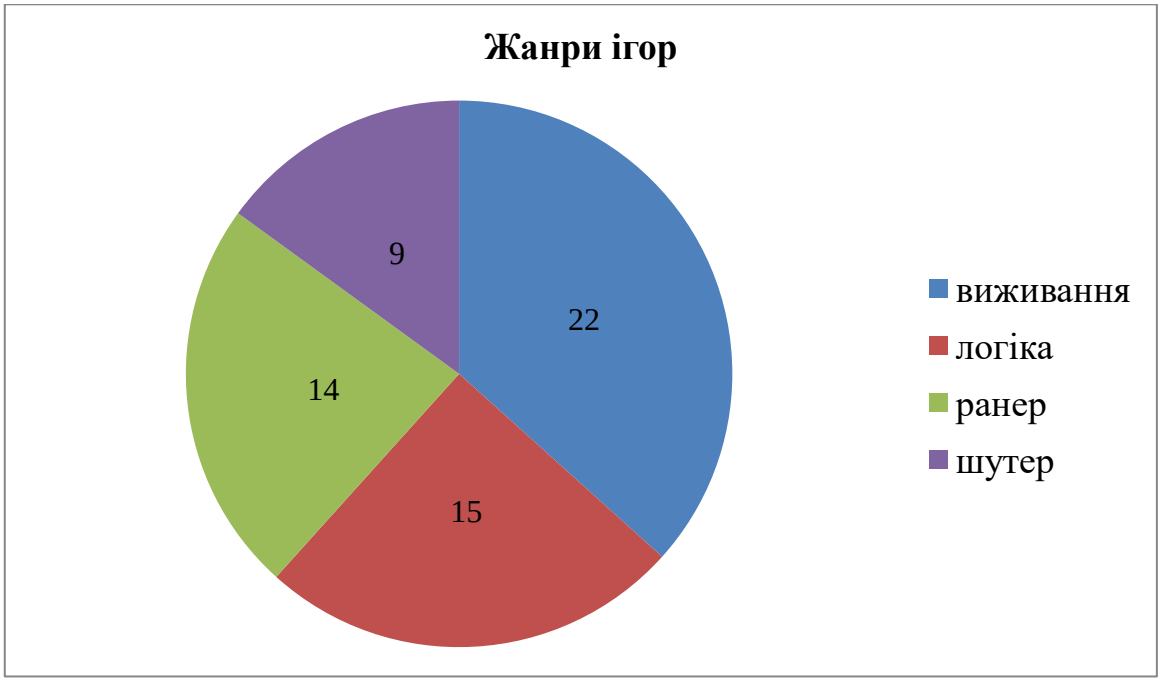

Рисунок 1.1 – Результати опитування респондентів

За результатами опитування було обрано створення проєкту гри Survival, або на виживання.

Було проведено огляд серії ігор на виживання, та виявлено деякі особливості ігор цього жанру. Виживання – один із найстаріших інстинктів притаманних тваринам і людям. Тому не дивно, що людство за свою історію виплутувалось із найнебезпечніших ситуацій, від смертельних епідемій до катаклізмів.

У симуляторах виживання гравець має досліджувати ігровий світ, добувати ресурси, необхідні йому для виживання, а ще намагатися уникнути небезпеки. Симулятори виживання дуже часто звертаються до теми робінзонади, де головний герой опиняється на безлюдному острові чи в дикій місцевості та дають можливість самостійно знаходити їжу та воду,

облаштовувати житло та захищатися від небезпеки, такої як негода, дикі тварини, злодії чи фантастичні вороги, як мутанти чи зомбі. Загибель персонажа в симуляторах зазвичай є непоправною: якщо гравець не зміг зберегти життя віртуальному герою, він має почати проходження з нуля, втрачаючи всі здобуті раніше ресурси.

Локація відеоігор може бути різною. Це може бути безлюдний острів, постапокаліптична пустеля, ліс або інша планета. Гравець знаходиться у цьому віртуальному світі наодинці або з іншими гравцями, які також борються за виживання. Гравець може збирати матеріали з навколишнього середовища, які потім використовуються для виготовлення знарядь, зброї, ліків, захисних споруд та інших корисних речей. Будівництво дозволяє гравцеві створювати укриття, бази або склади для захисту від небезпеки. Дослідження карти відкриває нові місця, дозволяючи знайти цінні ресурси або зустріти інших гравців.

Можна відокремити 4 піджанри ігор на виживання.

Перший – «класичний» survival. Сюжет в проєктах такого типу переважно відсутній. Зазвичай вони мають відкритий світ та можливість грати у мережі. Найяскравішим представником цього піджанру є Minecraft [12]. Це відеогра від незалежної студії Mojang 2011 року жанру «пісочниця» у відкритому світі з виглядом від першої/третьої особи. Minecraft дає в розпорядження гравцеві тривимірний процедурно генерований світ, що складається з кубічних блоків, які можливо використовувати на свій розсуд. Один з важливих аспектів Minecraft – це його режими гри, їх аж п'ять: творчість (англ. Creative), виживання (англ. Survival), пригода (англ. Adventure), гардкор (англ. Hardcore) і спостерігач (англ. Spectator).

У режимі виживання гравець повинен подорожувати і шукати потрібні блоки, боротися з мобами, вести господарство, виготовляти нові матеріали і майструвати інструменти. Тут персонаж має, крім інвентарю, шкали здоров'я, голоду і досвіду.

Minecraft також пропонує мультиплеєрний режим, де гравці можуть грати

разом у спільному світі. Це відкриває безліч можливостей для спільної творчості, експедицій і сутичок.

Іншим піджанром виживання є Survival Horror, тобто жахи. Тут можливість виживання поєднується з напругою та раптовістю появи ворогів. Більшість проєктів є орієнтованими на одного гравця, та мають сюжетну складову, яка займає головне місце. Яскравим прикладом є серія відеоігор Resident Evil розроблена компанією Capcom. Історія концентрується на боротьбі із зомбі та мутантами.

У 2023 році вийшов римейк Resident Evil 4 [13] де з'явилася система крафту (створення предметів), яка дозволяє гравцеві створювати предмети та боєприпаси, використовуючи зібрані ресурси. Також можна купувати, покращувати та обмінювати предмети у купця, та отримати від нього нові додаткові завдання, які можна виконати під час гри.

Третім піджанром виживання є Survival Action (бойовик). Він є найменш пов’язаним з концепцією Survival. В таких іграх активно розвинена геймплейна складова, яка часто опирається на механіку стрільби. Механіка виживання тут зазвичай полягає у можливості майструвати предмети та зброю з підручних матеріалів, та не є обов’язковою для використання.

Яскравим представником цього піджанру є The Last of Us [14]. Гра розроблена студією Naughty Dog. Це пригодницька гра від третьої особи, де гравець перетинає постапокаліптичні місцевості, такі як міста, будівлі, ліси та підземні комунікації. Для боротьби з ворогами можна використовувати вогнепальну зброю серійних і саморобних зразків, рукопашний бій, а також ховатися. Проєкт містить складний сюжет, що досліджує теми виживання, моралі, відповідальності та насильства. Гравець зможе пережити емоційно напружені моменти та складні моральні вибори, які впливають на подальший розвиток історії. Також, гра надає можливість вдосконалювати навички та зброю головного героя.

Останнім з піджанрів є стратегічне виживання. Цей жанр є мало пов’язаним з виживанням, але він все одно використовує механіки виживання

стратегії.

Одним із прикладів стратегічного виживання є Surviving Mars [15]. Це стратегічний симулятор, розроблений компанією Haemimont Games. У цій грі перед гравцями стоїть завдання колонізувати та підтримувати людську колонію на планеті Марс. Гра пропонує унікальний і захоплюючий досвід, оскільки гравці стикаються з проблемами виживання в несприятливому середовищі та управління ресурсами, щоб забезпечити виживання і зростання своєї колонії.

Гравці повинні керувати життєво важливими ресурсами, такими як вода, кисень і їжа, не забуваючи при цьому про добробут і щастя колоністів. У грі пропонується цілий ряд ігрових можливостей, включаючи можливість досліджувати нові технології, вивчати марсіанський ландшафт і стикатися з несподіваними подіями та викликами.

Дослідивши чотири піджанри ігор на виживання було встановлено, що їх межі поступово стираються, і нові ігри місять механіки, що притаманні усьому жанру виживання. Це також означає появу нових піджанрів у майбутньому.

1.4 Висновки до розділу 1

У цьому розділі було розглянуто тенденції розвитку індустрії розваг. Виявлено актуальність розробки та впровадження мобільних ігор, враховуючи великий попит та стрімкий розвиток індустрії розваг, та зокрема те, що мобільні ігри є актуальним та перспективним напрямком, який має потенціал для подальшого зростання та інновацій. Проаналізовано і зібрано інформацію про жанри, вимоги до геймплею та механіки сучасних ігор.

Виходячи з результатів аналізу та проведених опитувань респондентів було обрано об’єктом дослідження мобільну гру на виживання.

Також було розглянуто і порівняно найбільш популярні інструменти для реалізації проєкту. На основі чого обрано для створення застосунку рушій Unity та 3Д редактор Blender.

2 РОЗРОБКА АЛГОРИТМУ ТА КОМПОНЕНТІВ ПРОЄКТУ

2.1 Концепція гри

Гра уявляє собою «класичний» survival, сюжетом якої є потрапляння персонажу на безлюдний острів, який він повинен залишити, виживаючи у складних ситуаціях.

Основною ідеєю гри є відтворити максимально наближену до реалії симуляцію знаходження людини на безлюдному острові з великим переліком небезпек для його життя, таких як:

– голод;
– спрага;
– хвороби;
– небезпечні тварини.

Метою проєкту є примусити гравця поглинути в систему гри, переживати разом з персонажем реалістичні події.

Від гравця вимагається зосередитись на досягненні основної цілі – покинути безлюдний острів, постійно стежити за життєвими показниками персонажу та виконати вимоги для досягнення цієї цілі.

Для відтворення реалістичності розроблені та використані об’єкти та події з реального життя:

– у грі відбувається чергування дня та ночі;
– вигляд та поведінка тварин відповідають дійсності (вони рухаються як реальні тварини, можуть бути наляканими або агресивними, і навіть завдати шкоди персонажу);
– рослини відповідають місцевості та можуть мати лікарські властивості (також з них можна виготовляти предмети);
– кидання предметів враховує силу тяжіння;

– персонаж втомлюється, голодніє, відчуває спрагу;
– та інші.

Щоб не полишати гравця сам на сам у грі присутні підказки:

– підсвічування предметів, що можна використати;
– щоденник попереднього мешканця острову;
– інформація про те, як можна використати предмети.

Гравець зможе досягти основну ціль гри двома шляхами:

– старанно працювати, добувати ресурси, побудувати пліт, зібрати усі необхідні речі по завданню та самостійно покинути острів;
– покладаючись на свою вдачу, збільшити шанс того, що його помітять та заберуть з острову.

Другий спосіб потребує від гравця меншої кількості затрачених зусиль на збирання ресурсів та полювання але менш залежить від самого гравця та може збільшити час на проходження гри.

Наш гравець – це цілеспрямована людина, що не відступає перед труднощами. Гра вимагає від гравця зусилля та стимулює розвиток важливих навичок, як:

– спостережливість;
– старанність;
– цілеспрямованість;
– уважність;
– кмітливість.

Аналізуючи обраний метод проходження гри та тактику гравця, в майбутньому можна побудувати статистичну модель поведінки людини в екстремальних ситуаціях.

Гра зберігає дані гравця на сервері автоматично з інтервалом або при натисканні кнопки «Зберегти». Тому вона може бути перервана у будь-який момент та відтворена з місця останнього збереження.

2.2 Основні компоненти проєкту

Розглянемо основні компоненти проєкту та зв’язки між ними, яка наведена на рисунку 2.1.

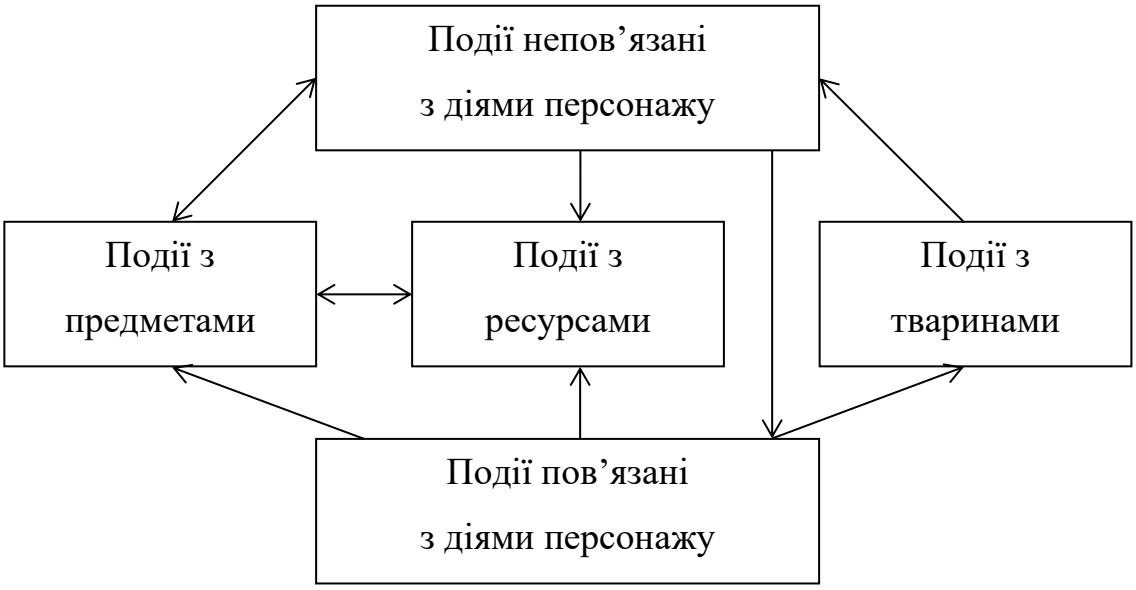

Рисунок 2.1 – Схема внутрішніх подій в проєкті

До подій непов’язаних з діями персонажу відноситься весь ігровий процес, в якому персонаж не приймає участь, та який від нього не залежить, такі як:

– графічні налаштування;
– збереження та завантаження інформації;
– відлік часу;
– зміна доби;
– поява ресурсів.

Персонаж може впливати на стан та режими тварин, на предмети та ресурси.

Розглянемо основні події та зв’язки між ними всередині проєкту більш детально. На рисунку 2.2 зображена схема подій ігрового процесу не пов’язаного з діями персонажу.

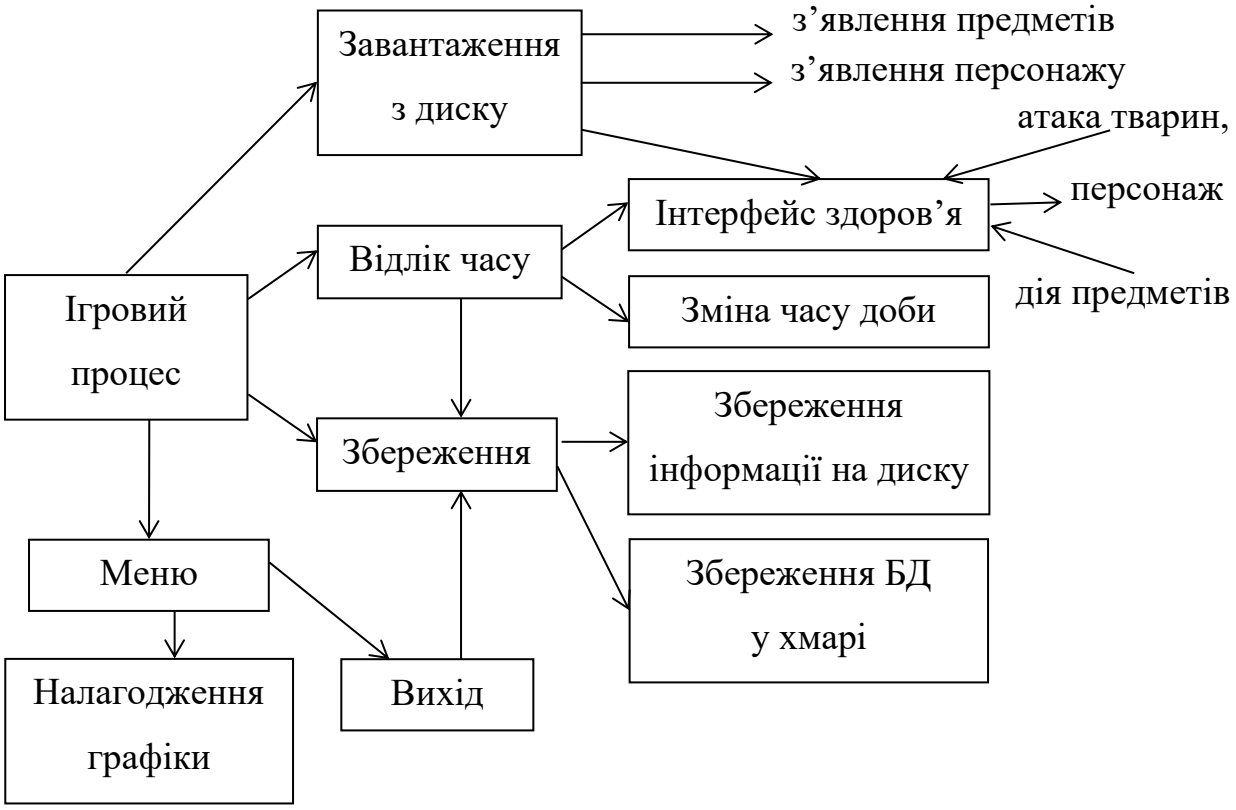

Рисунок 2.2 – Схема подій непов’язаних з діями персонажа

При запуску гри в ігровому процесі завантажуються усі збережені дані на диску, а саме:

– час з початку гри;
– стан життєвих показників персонажу;
– вміст інвентарю;
– предмети та їх координати у локації;
– координати знаходження персонажу;
– деякі стани подій ігрового процесу.

Після завантаження застосовуються усі збережені стани, створюються усі предмети, розставляються усі об’єкти по координатах.

Важливим процесом є постійний відлік та відстеження часу ігрового процесу. Це необхідно для повторення подій, створення та розташування ресурсів, впливу на стан показників здоров’я персонажу.

Збереження всіх даних на диск та у хмару у вигляді БД відбувається або періодично з відліком часу, або під час виходу з гри.

На рисунку 2.3 наведена схема процесів пов’язаних з дією персонажу.

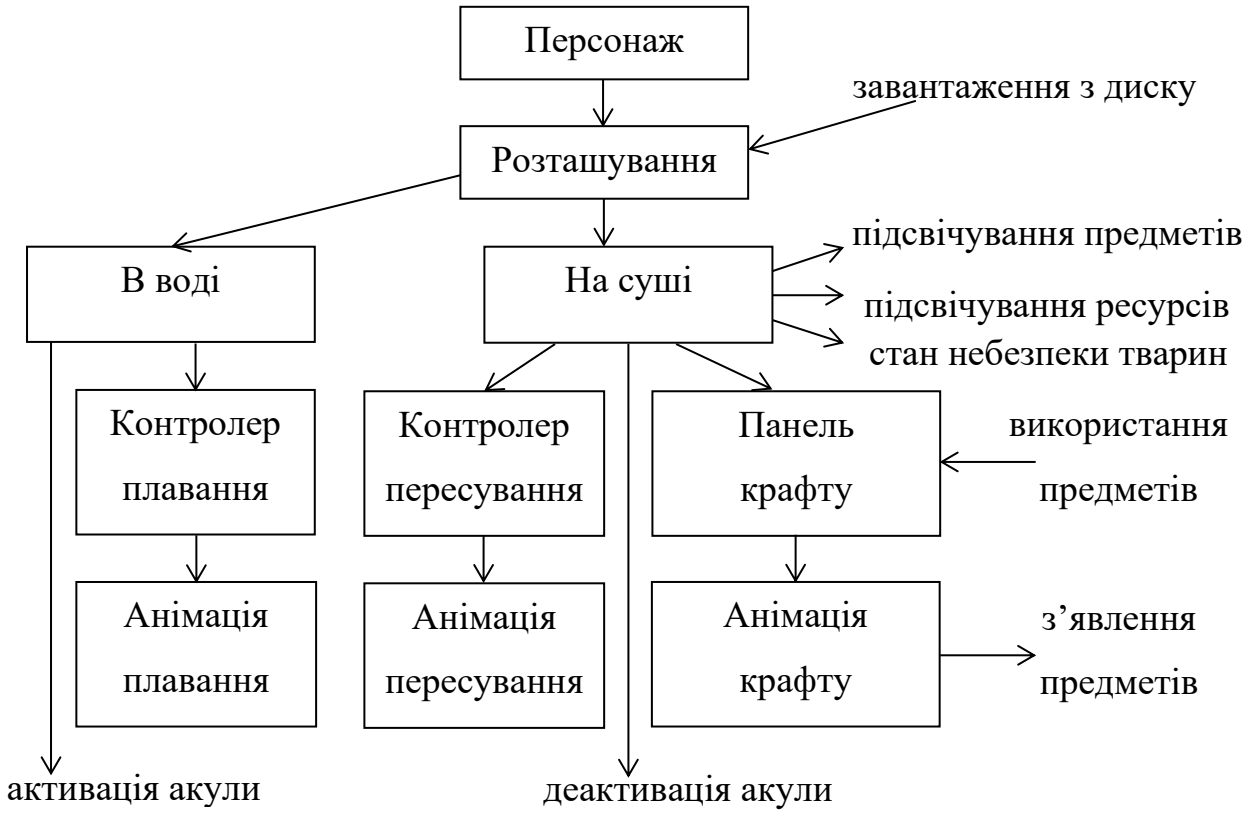

Рисунок 2.3 – Схема подій пов’язаних з діями персонажа

В залежності від розташування персонажу в воді чи на суші перемикаються контролери та аніматори на відповідні до цих стихій. Коли він знаходиться у воді, то активує акулу, яка його атакує та не дає упливти з острова. Коли персонаж на суші, він може взаємодіяти з предметами, ресурсами, тваринами тощо. Розглянемо як відбувається взаємодія персонажу з предметами. У програмі реалізовано механізм періодичного пошуку всіх предметів на острові, та розрахунку найближчого з них до персонажу. Якщо той знаходиться в радіусі 20 метрів, то він починає підсвічуватися. Підійшовши ще ближче на відстань витягнутої руки персонаж може підняти предмет. Схема взаємодії з предметами надана на рисунку 2.4.

Така ж взаємодія виникає і з ресурсами, тільки пошук відбувається не всіх водночас, а тільки певних в залежності від того, який інструмент тримає у руці персонаж.

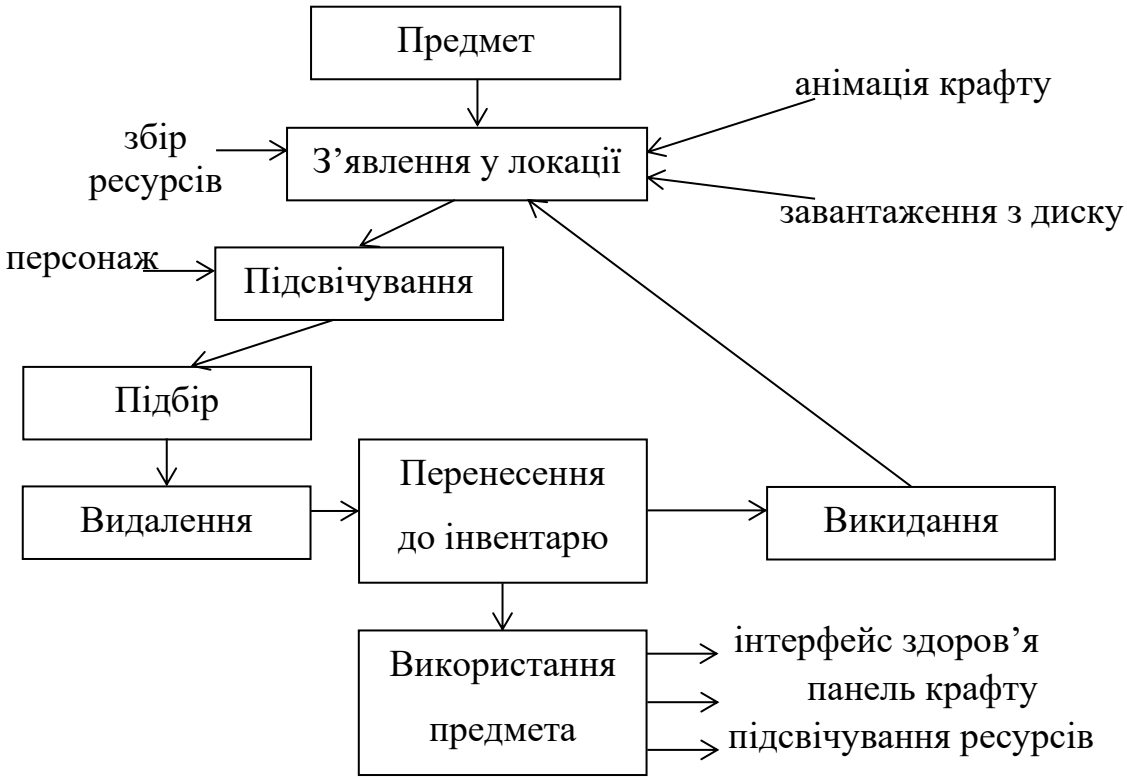

Рисунок 2.4 – Схема дій з предметами

Предмети розміщуються у локації після завантаження збережених координат та кількості останніх, після збору ресурсів або після крафту. Коли предмет підібрано, він видаляється з локації та створюється в інвентарі його опис. Після прямого використання, наприклад персонаж з’їв їжу чи використав ліки, предмет видаляється з інвентарю, а його властивість змінює показники здоров’я персонажу через інтерфейс здоров’я. Якщо предмет викинуто з інвентарю, то його опис видаляється, а сам предмет створюється у локації з шаблону об’єкту. Аналогічно створюється і розташовується відповідний предмет, який очікується після крафту, а задіяні в цьому процесі предмети видаляються з інвентарю.

Процес збору ресурсів схожий з процесом підйому предметів, тільки для кожного запускається своя анімація, ресурс видаляється з місця розташування та поруч створюється предмет йому відповідний з шаблону об’єкту. Також запускається процес відліку часу та шансу з’явлення ресурсу. Спочатку шанс малий, але ресурс все одно з’явиться, тому як с часом шанс появи збільшується,

доки не дойде до 100%. Нижче наведена схема взаємодії з ресурсами на рисунку 2.5.

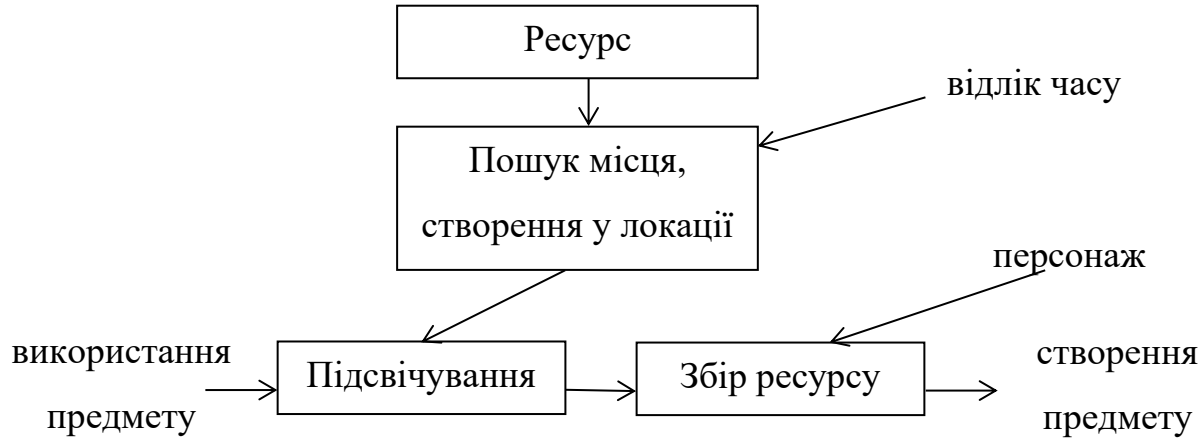

Рисунок 2.5 – Схема дій з ресурсами

Розглянемо, як відбувається взаємодія з тваринами, яка зображена на рисунку 2.6.

Рисунок 2.6 – Схема дій тварин

Тварини мають два стани: спокою та небезпеки. В стані спокою вони живуть своїм життям. Тобто пересуваються, вмикається анімація, яка імітує поведінку тварин, де вони приймають їжу, видивляються, приводять себе до ладу тощо.

При наближенні персонажу до тварин відбувається зміна їх режиму у стан небезпеки. В якому в залежності від тварини та додаткових подій, вони або втікають, або атакують персонажа. При віддаленні знов перемикається на режим спокою.

Майже така сама ситуація трапляється і з птахами, які сидять у гнізді. Тільки з різницею, що у стані спокою чайка ще й відкладає яйця та генерує пір’я. В стані небезпеки може або атакувати персонажа, або втікти від нього активувавши режим польоту. Це показано на рисунку 2.7.

Рисунок 2.7 – Схема дій чайки

Дія акули спрямована тільки на атаку, тобто у неї не існує стану спокою. Коли персонаж заходить у воду, то ведеться прорахунок, де з’явиться акула. Коли остання з’являється, то вона відразу починає атаку. Наближається до

персонажа, кусає його та відпливає в сторону для повторної атаки. Акула зникне тільки тоді, коли персонаж вийде на сушу. Схема поведінки акули зображена на рисунку 2.8.

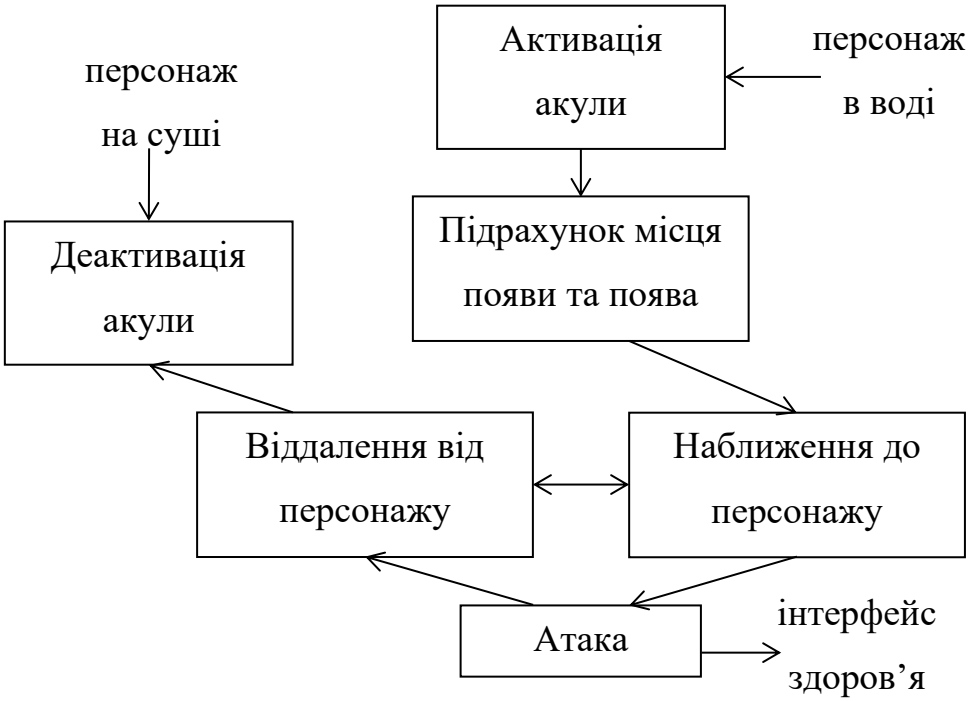

Рисунок 2.8 – Схема дій акули

2.3 Розробка алгоритму штучного інтелекту

Для реалізації поведінки тварин застосуємо штучний інтелект. Для цього використаємо NavMesh Agent. Це компонент рушія Unity, який допомагає об'єктам у пошуку шляхів через складні простори, створюючи «навігаційні» мапи в ігровому світі [16].

За допомогою NavMesh Agent можна запекти ландшафт у локації та розставити ділянки на ній, де певні об'єкти можуть чи не можуть пересуватися, або повинні зупинитися, чи перестрибнути. Також цей компонент допомагає налагодити швидкість пересування, розгортання, прискорення та зупин об'єктів.

Розглянемо принцип створення ШІ на прикладі поведінки кроля. На рисунку 2.9 наведено налагодження NavMesh Agent для кроля.

Рисунок 2.9 – Налагодження NavMesh Agent

Реалізуємо поведінку тварини у програмному коді (див. рис. 2.10).

```
28      private void Start()
29      {
30          //використовує агент ШІ
31          _navMeshAgent = GetComponent<NavMeshAgent>();
32          _navMeshAgent.speed = movementSpeed;
33          _animator = GetComponent<Animator>();
34          pers = GameObject.FindGameObjectWithTag("Player");
35          startPosition = transform.position;//позиція кроля на початку
36      }
37
38      void OnEnable()
39      {
40          //повторює функцію пересування тварини
41          InvokeRepeating(nameof(MoveAnimal), changePositionTime, changePositionTime);
42      }
43
```

Рисунок 2.10 – Ініціалізація на початку

При першому запуску застосовується NavMesh Agent, аніматор кроля, для відображення його анімації, пошук об’єкту персонаж для взаємодії з ним, та встановлюється початкова крапка тварини, щоб повернути її в процесі на це місце. Кожного разу при активації кроля запускається метод OnEnable(), в якому розпочинається корутина з встановленим інтервалом (для кроля 6 секунд) для зміни події (див. рис. 2.11).

```
100     private void MoveAnimal()
101     {
102         //якщо кроль не активний, то відміняє корутину
103         if (!transform.gameObject.activeInHierarchy)
104         {
105             CancelInvoke();//відміняє таймер
106             return;
107         }
108         countRepeat++;
109         //після кожного другого бігу вмикається доп. анімація
110         //і якщо плеєр не поруч
111         if (countRepeat > 2)
112         {
113             isDopAnim = true;
114             int doing = Random.Range(1, 4);//рандомно обираються дії
115             switch (doing)
116             {
117                 case 1:
118                     _animator.Play("Guarding");
119                     break;
120                 case 2:
121                     _animator.Play("Sitting1");
122                 break;
123                 case 3:
124                     _animator.Play("Sitting2");
125                 break;
126             }
127             countRepeat = 0;
128         }
129         else
130         {
131             isDopAnim = false;
132             _navMeshAgent.SetDestination(RandomNavSphere(moveDistance));
133             //обмежує радіус віддалення від спавну
134             if (startDistance > maxDistance)
135             {
136                 _navMeshAgent.SetDestination(startPosition);
137             }
138             else
139             {
140                 _navMeshAgent.SetDestination(RandomNavSphere(moveDistance));
141             }
142         }
143     }
```

Рисунок 2.11 – Метод корутини подій

Якщо кріль активний, то кожні 6 секунд два рази він пересувається, а на третій раз вмикається рандомно одна з анімацій. Також постійно відстежується відстань, на яку він відбіг від початкової своєї крапки. Якщо відстань більше допустимої, то тварина повертається до початку точки з'явлення.

Пересування кроля відбувається за допомогою NavMesh Agent, який прораховує шлях до вказаної крапки на ландшафті. Для цього застосуємо метод генерації рандомної координати на мапі (див. рис. 2.12).

```
89      //функція створює сферу та бере рандомно точку у цій сфері
90      Vector3 RandomNavSphere(float distance)
91      {
92          Vector3 randomDirection = UnityEngine.Random.insideUnitSphere * distance;
93          randomDirection += transform.position;
94          NavMeshHit navHit;
95          NavMesh.SamplePosition(randomDirection, out navHit, distance, -1);
96          return navHit.position;
97      }
```

Рисунок 2.12 – Метод генерації рандомної координати

Цей метод генерує уявну сферу з вказаним діаметром (максимально допустима відстань пересування), обирає на ній рандомно крапку та проєктує перпендикулярно на ландшафт локації з корегуванням через NavMesh Agent до найближчої доступної запеченої області, і повертає отримані координати.

Далі у методі Update(), який викликається кожного кадру, і відбувається саме переміщення тварини (див. рис. 2.13).

В ньому постійно відстежується місцезнаходження персонажу у локації та прораховується відстань від нього до кроля. Якщо відстань менше 5 метрів, то відміняється корутина з додатковими анімаціями, та тварина починає тікати. Коли досягнена безпечна відстань, то корутина знову розпочинається і кріль продовжує анімації спокою.

```
45      private void Update()
46      {
47          if (isDie) {return;}
48          Vector3 diff = pers.transform.position - transform.position;
49          diff.y = 0;
50          curDistance = diff.sqrMagnitude;//відстань між кролем та плеєром
51          //якщо відстань до персонажа менше 5 починає тікати
52          if (curDistance < 5 && !isDie)
53          {
54              if (!isRunAway)
55              {
56                  CancelInvoke();//відміняє таймер
57                  isRunAway = true;
58                  posAway = RandomNavSphere(moveDistance);//точка куди тікати
59                  _navMeshAgent.SetDestination(posAway);
60                  countRepeat = 0;
61              }
62          }
63          //якщо вже тікає то відстежую чи прибіг
64          if (isRunAway)
65          {
66              Vector3 diffRun = transform.position - posAway;
67              diffRun.y = 0;
68              float runAwayDistance = diffRun.sqrMagnitude;//відстань до точки втікання
69              if (runAwayDistance < 1)
70              {
71                  isRunAway = false;
72                  isStartCorutine = true;
73              }
74          }
75          //якщо втік то починає знову корутину анімацій
76          if (curDistance > 5 && isStartCorutine)
77          {
78              isStartCorutine = false;
79              InvokeRepeating(nameof(MoveAnimal), changePositionTime, changePositionTime);
80          }
81          _animator.SetFloat("Speed", _navMeshAgent.velocity.magnitude / movementSpeed);
82      }
```

Рисунок 2.13 – Метод Update()

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОЄКТУ

3.1 Створення локації та інтерфейсу гри

За допомогою редактора ландшафту у рушії Unity було створено ландшафт острову, додано основні текстури та розмальовано пісок, траву, гори тощо (див. рис. 3.1). Також із бібліотеки Unity Asset Store скачано та додано воду до локації [17].

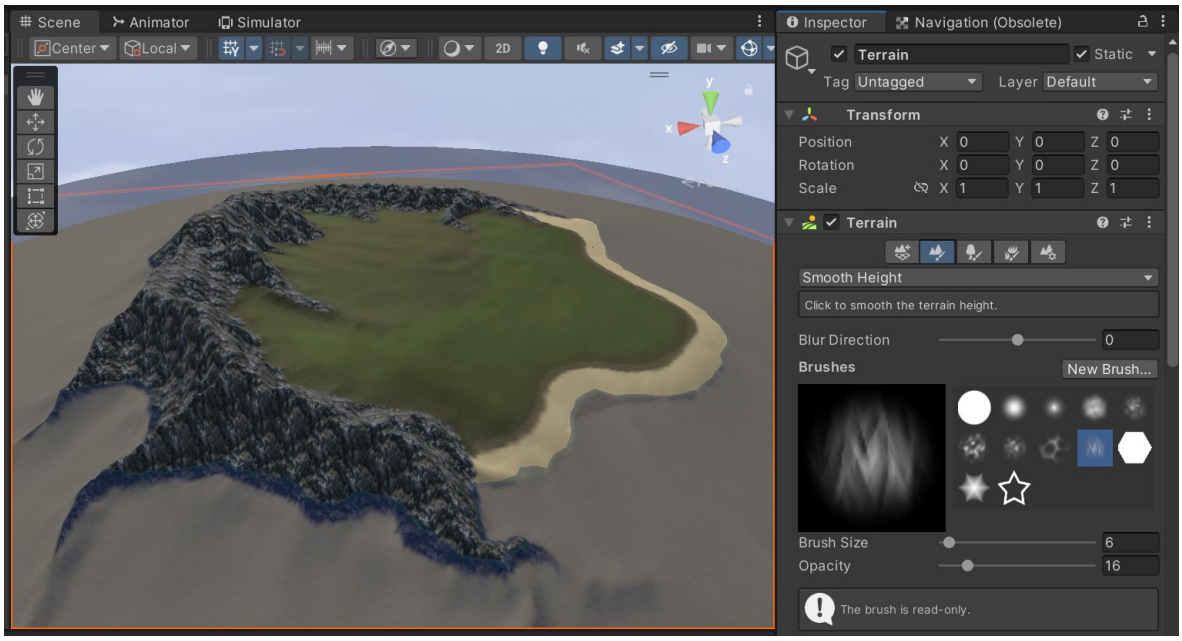

Рисунок 3.1 – Локація гри

До проєкту додано область Canvas, всередині якої будуть знаходитись всі елементи інтерфейсу користувача. Зроблено усі необхідні графічні та фізичні налагодження елементів інтерфейсу, такі як: кнопки, поля для виводу тексту та іконок (див. рис. 3.2).

До інтерфейсу додано додаткові меню та панелі:

– основне меню;
– панель налагодження графіки;
– панель крафту;

– меню входу та виходу з гри;
– панель інвентарю та збереження речей у сундуку;
– панель інтерфейсу здоров’я;
– панель відображення мапи.

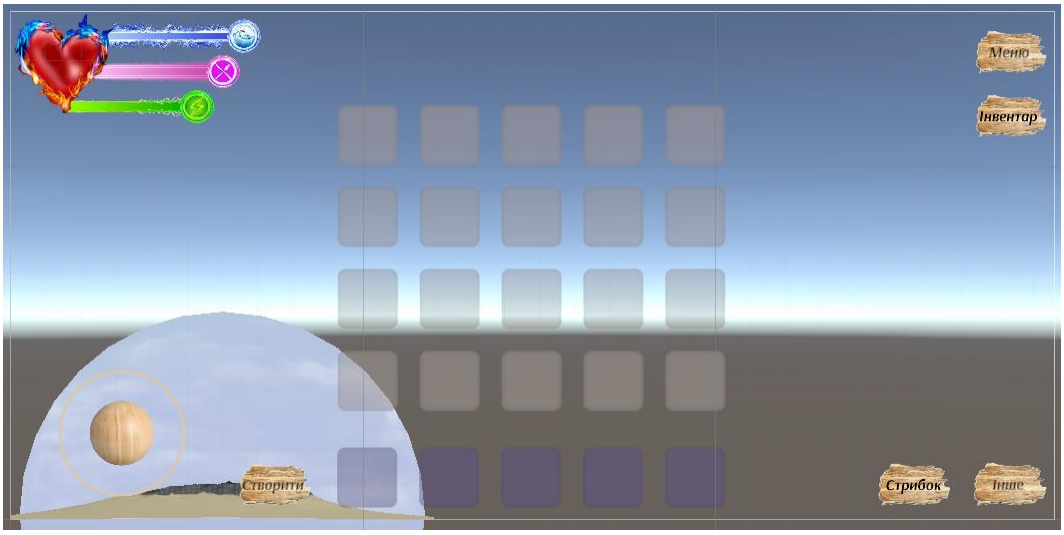

Рисунок 3.2 – Основний інтерфейс гри

Панель інтерфейсу здоров’я містить в собі чотири шкали життєвих показників персонажа. Синя шкала відображає показник спраги, з наростанням якої показник зменшується, тобто він вказує не на саму спрагу, а навпаки на втамування її. Так само відбувається з рожевим – він вказує на ситість. Зелений – вказує на життєву енергію, він постійно зменшується з втомою персонажу. Червоне серце – показник рівня життя, який залежить від попередніх.

Через основне меню можна активувати панель налагодження графіки, яка має достатньо багато параметрів, щоб виставити якість та зменшити вимогливість її до системи, тим самим дати змогу працювати застосунку навіть на слабких гаджетах (див. рис. 3.3).

В цій панелі можна виставити роздатну здатність обравши одну зі стандартних і тонко масштабувати її. Обрати якість зображення, яке змінить якість ренедерингу, відображення тіней та прорисування в залежності від відстані. Також активувати динамічну зміну розданої здатності, яка змінюється автоматично в залежності від показника FPS.

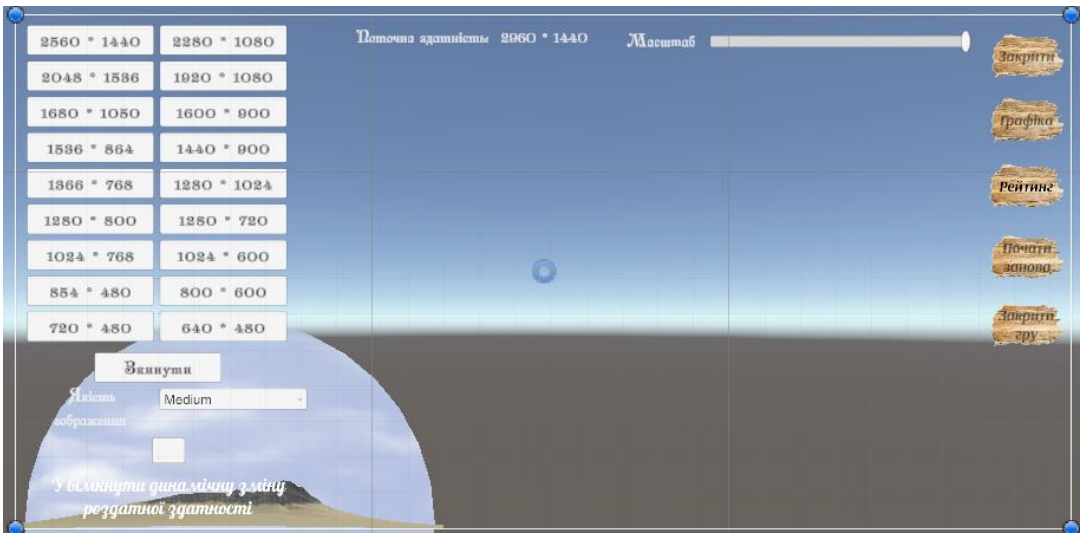

Рисунок 3.3 – Панель налагодження графіки

Складну структуру має панель крафту (див. рис. 3.4).

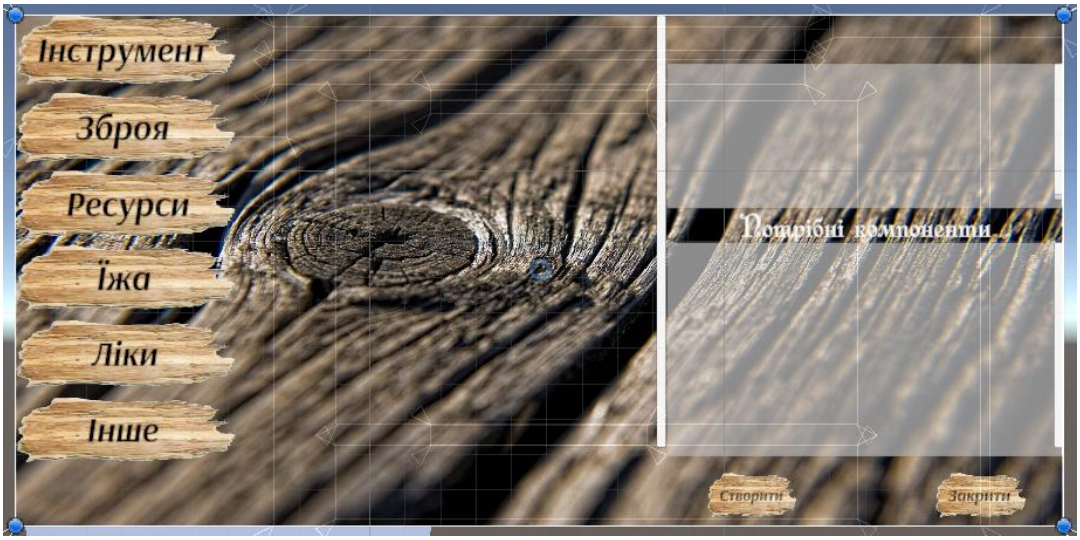

Рисунок 3.4 – Панель крафту

Панель крафту можна поділити на 3 частини:

– вибір типу об’єкту крафту;

– відображення усіх можливих об’єктів крафту;

– опис об’єкту крафту та його складових.

Також панель має динамічну частину, яка прораховує кількість часу на відтворення та чи вистачає ресурсів на крафт.

Для збереження речей додано панель сундуку (див. рис. 3.5).

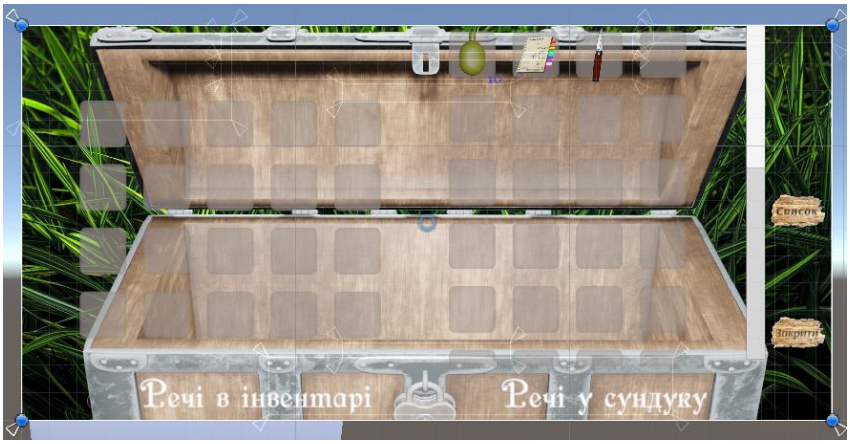

Рисунок 3.5 – Панель збереження речей у сундуку

Через цю панель гравець може зберігати речі у сундуку, які не вмістились в інвентарі, або потрібні для виконання завдання зі списку, а також подивитись на цей список.

3.2 Створення 3Д об’єктів

Для реалізації проєкту потрібно багато 3Д об’єктів, таких як: персонаж, тварини, рослини, предмети тощо. Більшість з них було створено у 3Д редакторі Blender (див. рис. 3.6).

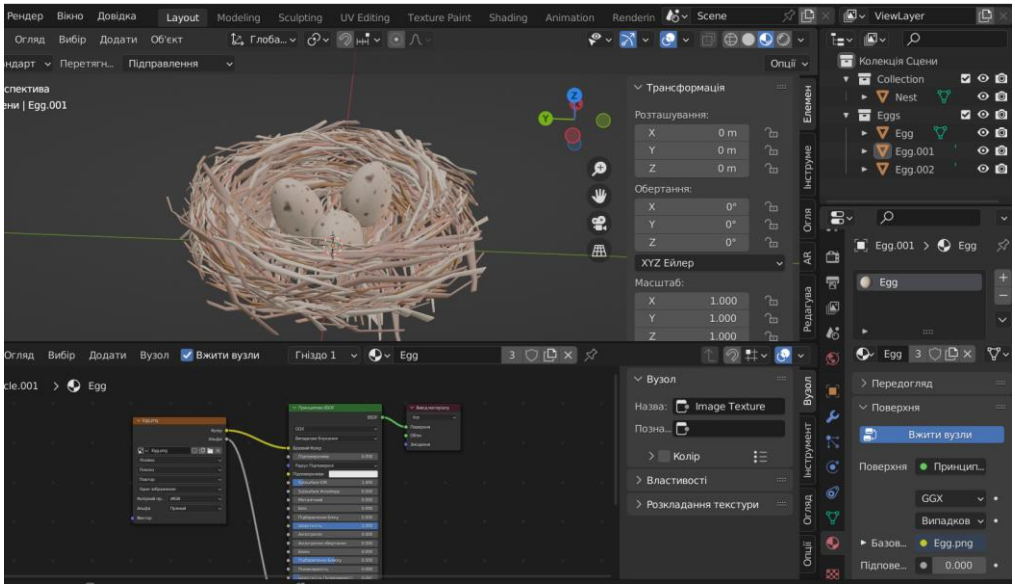

Рисунок 3.6 – 3Д об’єкт гнізда

Усі об’єкти додано до проєкту в Unity, створено їх шаблони (див. рис. 3.7). Налагоджено їх основні графічні та фізичні властивості. Встановлені коллайдери – компоненти, які визначають форму об’єкту для цілей фізичних зіткнень [18].

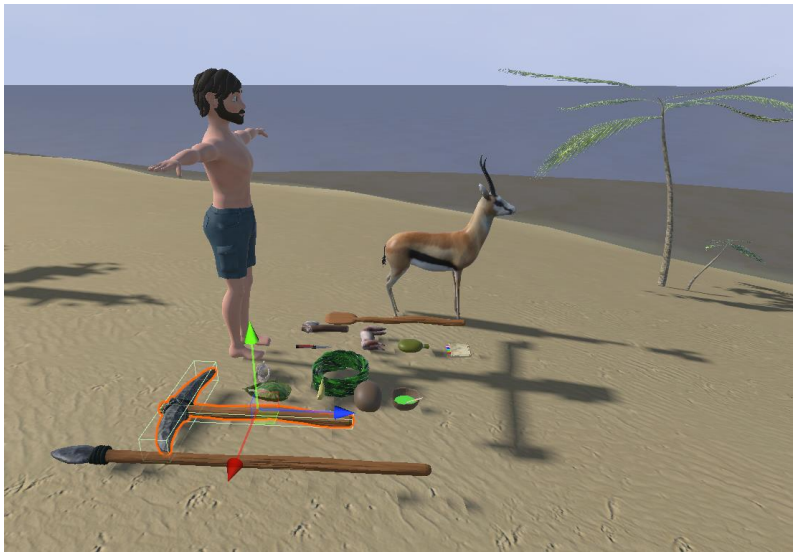

Рисунок 3.7 – Деякі об’єкти проєкту

Для персонажу та тварин було зроблено та додано у проєкт усі основні та додаткові анімації, створено контролери та аніматори, розставлені зв’язки та алгоритми (див. рис. 3.8).

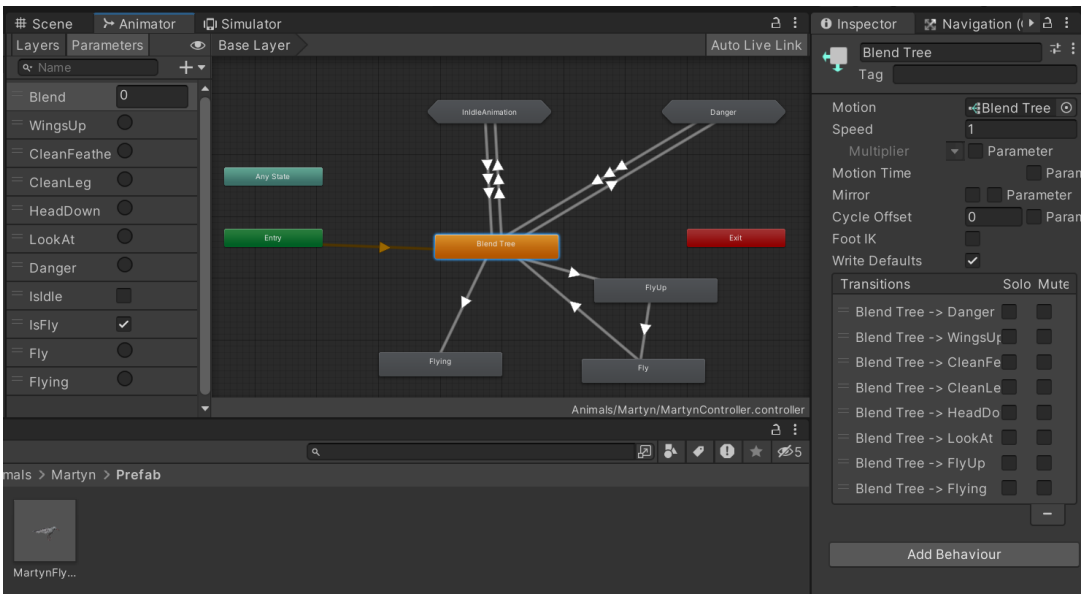

Рисунок 3.8 – Аніматор птаха

До проєкту також додано більш складні об’єкти та налагоджено їх властивості, наприклад динамічне небо зі зміною дня та ночі або багаття.

Для реалізації вогню, диму та іскор вогнища використано систему часток (див. рис. 3.9).

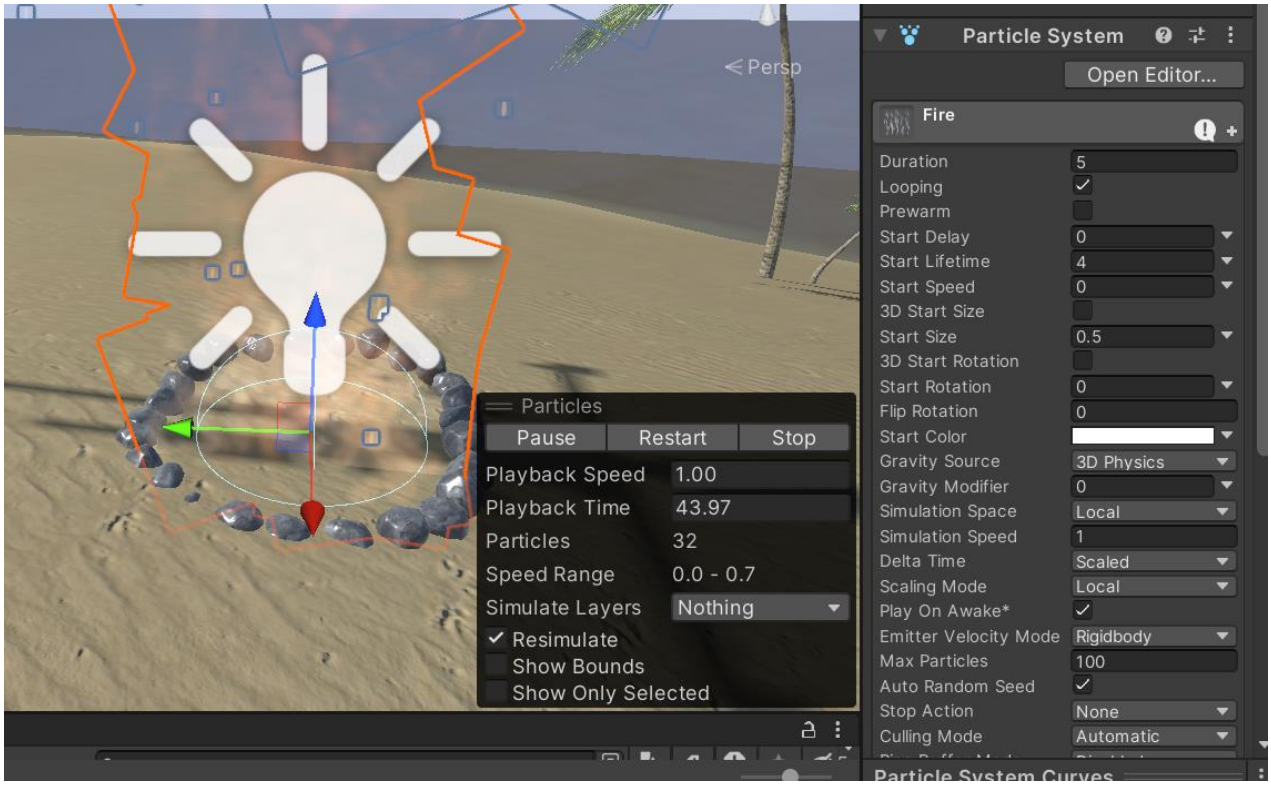

Рисунок 3.9 – Система часток у багатті

Частки генеруються з заданого місця. Їм задаються такі параметри як: кількість часток, швидкість початкова та розгін, траєкторія руху, затухання та інші. До кожної частки додається набір графічних елементів, кожен з яких відображає язик полум’я. Ці елементи постійно змінюються кадр за кадром та створюють ефект живого полум’я.

Для реалізації диму та іскор використовується така ж сама система, тільки на частки накладаються інші графічні елементи та інший характер поведінки часток.

Для оптимізації ігри та створення візуальних графічних ефектів у проєкт додано Universal Render Pipeline (URP) – це масштабований багатоплатформенний конвеєр візуалізації, який завдяки масштабованості,

налагодженості та багатому набору функцій пропонує творчу свободу в будь-якому типі проєкту, від стилізованих візуальних елементів до фізичного рендерингу [19].

URP – це крута можливість розробникам самостійно через скрипти змінювати спосіб рендеру, не поринаючи у глибини математики шейдерів та фізики світла в Unity. Він має величезну кількість налаштувань, вбудований постпроцесинг, змінений рушій рендерингу світла і матеріалів.

Використаємо можливості URP для створення динамічного неба та зміни дня і ночі.

Динамічне небо являє собою сферу, яка ніби окутує локацію проєкту. На сферу накладено текстури хмар та шейдери (див. рис. 3.10). За допомогою шейдерів та обертаючи сферу можна змінювати відображення хмар на небі. Основне джерело світла, яке знаходиться за сферою, освітлює небо та хмари, та проєктує тіні на ландшафт та об’єкти.

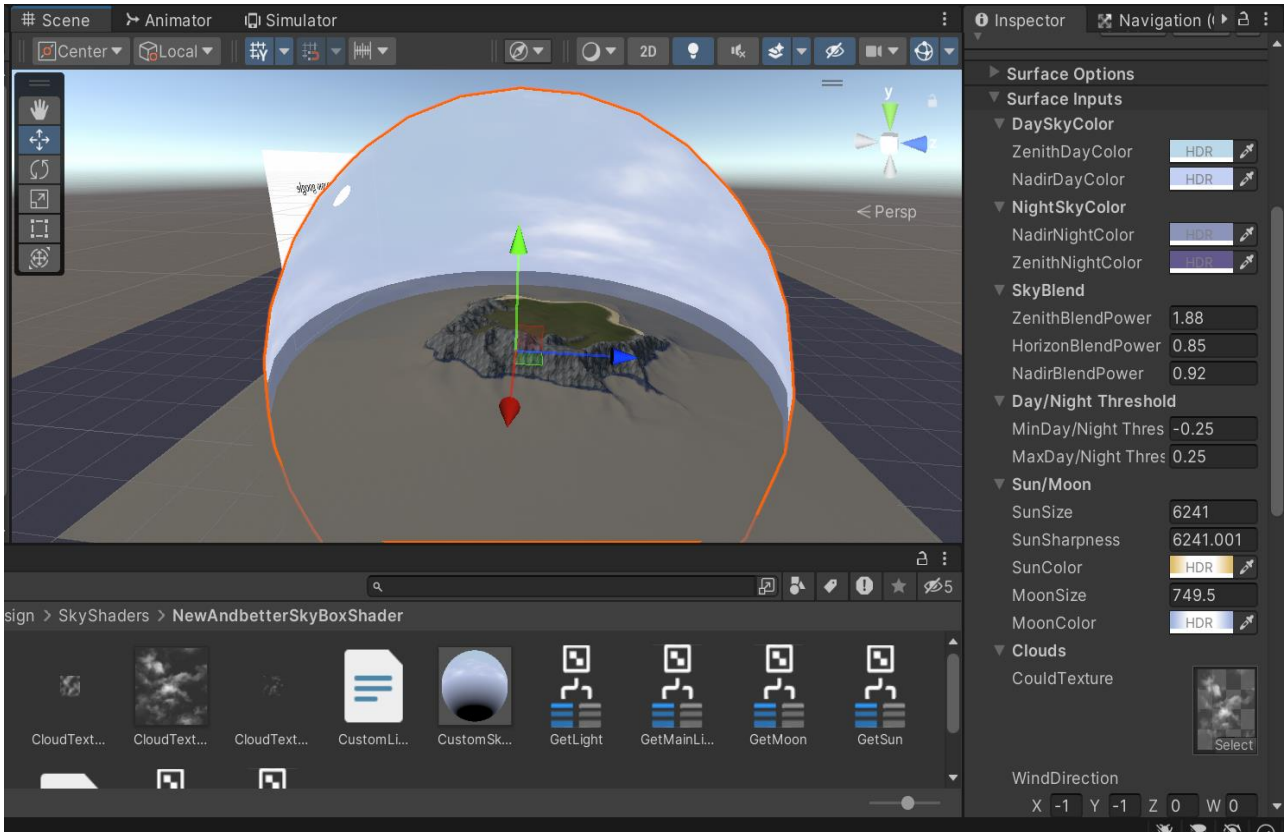

Рисунок 3.10 – Відображення неба у проєкті

За допомогою постпроцесингу та скрипту реалізуємо зміну дня та ночі (див. рис. 3.11).

```
40      void UpdateTime()
41      {
42          //якщо день то світліше та повільніше пересуваєься світло
43          if (isDay)
44          {
45              sunAngle += 0.15f;
46              //кожного разу змінюється інтенсивність світла 90 градусів - макс 1.2
47              _intensity = 1.2f - Mathf.Abs(90 - sunAngle) * 0.0077f;
48              //змінюється розмір
49              sunSize = 6241 - Mathf.RoundToInt(Mathf.Abs(90 - sunAngle) * 0.12f);
50              //активує пост процесінг вечора
51              if (sunAngle > 150 && !isEvening)
52              {
53                  transform.GetChild(1).gameObject.SetActive(true);
54                  isEvening = true;
55              }
56              if (sunAngle > 30 && isMorning)
57              {
58                  transform.GetChild(1).gameObject.SetActive(false);
59                  isMorning = false;
60              }
61                  //перемикаємо на ніч
62                  if (sunAngle > 178f)
63              {
64                  sunSize = 6241;
65                  isDay = false;
66                  sunAngle = 3;
67                  isEvening = false;
68                  transform.GetChild(1).gameObject.SetActive(false);
69                  volume.profile.TryGet(out bloom);
70                  bloom.intensity.value = 0f;
71                  //активуємо нічний пост процессінг
72                  transform.GetChild(2).gameObject.SetActive(true);
73                  //виставляємо параметри шейдера на ніч
74                  transform.GetChild(0).gameObject.GetComponent<MeshRenderer>().material.SetFloat("_MinDay_Night_Threshold", -2f);
75                  transform.GetChild(0).gameObject.GetComponent<MeshRenderer>().material.SetFloat("_HorizonBlendPower", 0.25f);
76                  transform.GetChild(0).gameObject.GetComponent<MeshRenderer>().material.SetFloat("_CloudSharpness", -0.84f);
77              }
78              transform.GetChild(0).gameObject.GetComponent<MeshRenderer>().material.SetInt("_SunSize", sunSize);
79          }
80          else
```

Рисунок 3.11 – Частина коду зміни дня та ночі

Метод UpdateTime() запускається кожну секунду. В ньому постійно змінюється розмір та кут нахилу сонця відносно локації. Також в залежності від куту змінюються параметри накладеного постпроцесингу. Коли сонце в зеніті, то освітлення максимальне, а коли на горизонті – йде затухання та накладання ефекту заходу та сходу сонця.

З точністю до навпаки відбуваються події у нічний час, тільки зміна триває швидше ніж у денний.

3.3 Програмна реалізація проєкту

Для реалізації проєкту було написано біля 90 скриптів на мові C# [20, 21] з використанням Microsoft Visual Studio 2022 – це інтегроване середовище розробки програмного забезпечення для редагування, налагодження та складання коду, а також для публікації програми [22].

Розглянемо деякі з цих скриптів.

3.3.1 Збереження та зчитування даних

При вході та виході з гри, а також періодично дані про персонажа, усі предмети та деякі стани завантажуються і записуються на диск.

Розглянемо частину коду з реалізацією цих процесів. Для цього було створено 2 скрипта: SaveLoadManager та SaveData.

У SaveLoadManager саме записуються та зчитуються необхідні дані, а у SaveData ці дані серіалізуються.

Серіалізація – це процес перетворення стану об'єкта у форму, придатну для збереження або передачі [23].

Шляхом серіалізації дані зберігаються у вигляді послідовності байт, що дуже ускладнює процес їх взлому.

Також можна зберігати самі об’єкти, що дуже полегшує процес збереження та обробки завантажених даних.

Розглянемо процес серіалізації даних на прикладі збереження стану панелі здоров’я персонажу та його розташування в локації (див. рис. 3.12).

Для цього створимо дві структури Vec3 та HealthStat. Vec3 для збереження вектора положення персонажу у просторі.

HealthStat буде містити в собі дані про стан здоров’я персонажу, голоду, спраги, енергії, хворобу та кровотечу, а також про час проведений у грі.

```
12      //структура вектора для збереження вектора3 положення
13      [System.Serializable]
14      public struct Vec3
15      {
16          public float x, y, z;
17          public Vec3(float x, float y, float z)
18          {
19              this.x = x;
20              this.y = y;
21              this.z = z;
22          }
23      }
24
25      //структура життєвих показників для збереження показників персонажа та часу
26      [System.Serializable]
27      public struct HealthStat
28      {
29          public float health, water, eat, energy;
30          public int kindOfIll,timeHour, timeSeconds;
31          public bool isIll, isBleeding;
32          public HealthStat(float health, float water, float eat, float energy,
33              bool isIll, int kindOfIll, bool isBleeding, int timeHour, int timeSeconds)
34          {
35              this.health = health;
36              this.water = water;
37              this.eat = eat;
38              this.energy = energy;
39              this.isIll = isIll;
40              this.kindOfIll = kindOfIll;
41              this.isBleeding = isBleeding;
42              this.timeHour = timeHour;
43              this.timeSeconds = timeSeconds;
44          }
45      }
46      public Vec3 persPosition;
47      public HealthStat healthStat;
```

Рисунок 3.12 – Частина коду серіалізації даних

Отримаємо необхідні дані для серіалізації, а саме: розташування персонажу у просторі та показники здоров’я від класу HealthPanel (панель здоров’я), який розташовано на персонажі (див. рис. 3.13).

```
100     //збирає данні про персонажа
101     public void PersData()
102     {
103         GameObject pers = GameObject.FindGameObjectWithTag("Player");
104         persPosition = new Vec3(pers.transform.position.x,
105             pers.transform.position.y, pers.transform.position.z);
106         var go = pers.GetComponent<HealthPanel>();
107         healthStat = new HealthStat(go.GetHealth(), go.GetWater(),
108             go.GetEat(), go.GetEnergy(), go.GetIsIll(),
109             go.GetKindOfIll(), go.GetIsBleeding(), go.GetTimeHour(),
110             go.GetTimeSeconds());
111     }
```

Рисунок 3.13 – Частина коду збору даних для серіалізації

Серіалізовані дані записуємо у файл з ім’ям save.dat з шляхом по замовчуванню у методі SaveGame(), перетворюючи дані у бінарний формат (див. рис. 3.14).

```
24      void Start()
25      {
26          pers = GameObject.FindGameObjectWithTag("Player");
27          filePath = Application.persistentDataPath + "/save.dat";
28      }
29
30      public void SaveGame()
31      {
32          BinaryFormatter bf = new BinaryFormatter();
33          FileStream fs = new FileStream(filePath, FileMode.Create);
34
35          SaveData saveData = new SaveData();
36          saveData.PersData();
37          saveData.SetSunData(dynamicSky);
38          SaveInventoryPanel(saveData);
39          SaveQuickSlots(saveData);
40          SaveChestSlots(saveData);
41          SaveAllItems(saveData);
42          SaveIsMapActive(saveData);
43          SaveIsRaftCreated(saveData);
44          bf.Serialize(fs, saveData);
45
46          fs.Close();
47          _database.GetComponent<Database>().SaveDatabase("Playing");
48      }
```

Рисунок 3.14 – Метод збереження даних у файл

Процес завантаження даних виконується з точністю до навпаки. Дані зчитуються з файлу, десеріалізуються та встановлюються усі показники та стани у відповідних класах через методи Set() (див. рис. 3.15).

```
 99      //метод повертає персонажа в гру та всі його параметри
100      private void SetPersData(SaveData saveData)
101      {
102          Vector3 vector3 = new Vector3(saveData.persPosition.x, saveData.persPosition.y,
103              saveData.persPosition.z);
104          //вимикає керування персонажем
105          pers.GetComponent<Invector.vCharacterController.vThirdPersonInput>().enabled = false;
106          pers.transform.position = vector3;//пересуває персонажа на збережену позицію
107          Invoke("OnPersonInput", 0.5f);
108          var go = pers.GetComponent<HealthPanel>();
109          go.SetHealth(saveData.healthStat.health);
110          go.SetWater(saveData.healthStat.water);
111          go.SetEat(saveData.healthStat.eat);
112          go.SetEnergy(saveData.healthStat.energy);
113          if (saveData.healthStat.isIll)
114          {
115              go.Illness(saveData.healthStat.kindOfIll);
116          }
117          if (saveData.healthStat.isBleeding)
118          {
119              go.SetIsBleed();
120          }
121          go.SetTime(saveData.healthStat.timeHour, saveData.healthStat.timeSeconds);
122      }
```

Рисунок 3.15 – Метод застосування завантажених даних

3.3.2 Дії з предметами

Для взаємодії з предметами та створення інвентарю було застосовано ScriptableObject. ScriptableObject – це клас, який дозволяє зберігати велику кількість інформації, що передається незалежно від зразків скрипта. Він є одним із способів реалізації модульності та повторного використання коду в проєкті [24].

Основною ідеєю ScriptableObject є те, що він є спеціальним типом об'єкта, який може бути створений, збережений та використаний в редакторі Unity. ScriptableObject має значну перевагу, оскільки його екземпляри можуть бути створені на основі класів, які успадковуються від нього, але можуть бути використані безпосередньо в сценах, як самостійні об'єкти.

Створимо клас ItemScriptableObject, який успадкує властивості ScriptableObject. В ньому створимо змінні, які будуть нести властивості та опис наших предметів (див. рис. 3.16).

```
7   public class ItemScriptableObject : ScriptableObject
8   {
9       public enum ItemType { Default, Food, Weapon, Instrument, Resource, Medicament, Other }
10      public string itemName;
11      public int maximumAmount;
12      public GameObject itemPrefab;
13      public Sprite icon;
14      public ItemType itemType;
15      public string itemDescription;
16      public bool isConsumable;
17      public string inHandName;
18
19      [Header("Consumable Characteristics")]
20      public int changeHealth;
21      public int changeHunger;
22      public int changeThirst;
23      public int changeEnergy;
24
25  }
```

Рисунок 3.16 – Клас ItemScriptableObject

Усі предмети матимуть такі властивості: назву, максимальну кількість у стаку, шаблон цього предмету, іконку, тип (їжа, зброя, інструмент, ресурс, медикамент тощо), опис, чи можна його використати, назва предмету у руці, та чи змінює життєві показники і на скільки при використанні.

Було написано скрипт, який дає змогу створювати певний скриптований об’єкт, та зберігати його (див. рис. 3.17).

```
6   //меню створення об'єкту
7   [CreateAssetMenu(fileName ="Item",menuName ="Inventory/Items/New Item")]
8
9   public class ItemCreator : ItemScriptableObject
10  {
11      public void Start()
12      {
13          itemType = ItemType.Food;
14      }
15  }
```

Рисунок 3.17 – Клас створення скриптованих об’єктів

Було створено усі необхідні скриптовані об’єкти, надано їм властивості та описи. Вони будуть застосовуватися при створені певних предметів та дії з ними (див. рис. 3.18).

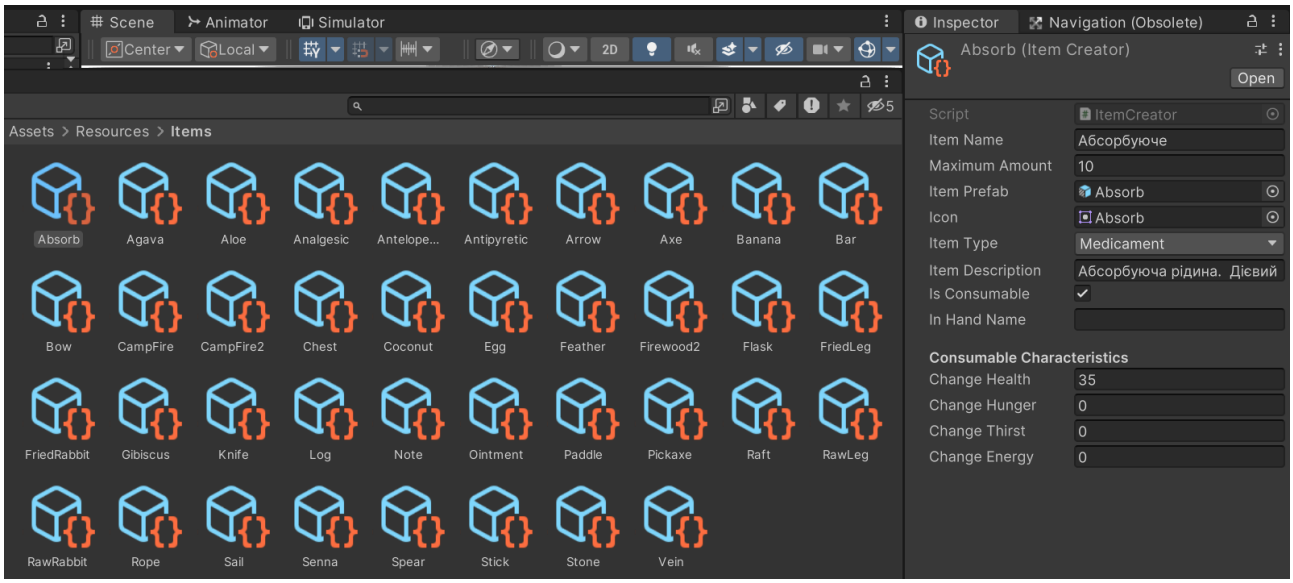

Рисунок 3.18 – Скриптовані об’єкти

Також було створено ще один скрипт Item, та додано його до всіх шаблонів предметів. Він буде нести інформацію про екземпляр скриптованого об’єкта, його кількість у локації та чи потрібно зберігати про нього інформацію на диску (див. рис. 3.20).

```
6   public class Item : MonoBehaviour
7   {
8       public ItemScriptableObject item;
9       public int amount;
10      public bool isNotSave;//яке не буде зберігатися у файл
11  }
```

Рисунок 3.20 – Код скрипту Item

Коли гравець підіймає предмет, то сам предмет видаляється, з екземпляру береться необхідна інформація про нього, а саме: назва предмету, кількість, іконка. Ця інформація додається до слоту в інвентарі. Тобто в інвентарі знаходиться не сам об’єкт, а його властивості та опис з екземпляру. Коли гравець викидає предмет, то з екземпляру береться посилання на шаблон об’єкту, створюється новий об’єкт у локації, встановлюється кількість, а зі слоту видаляється вся інформація.

3.3.3 Пошук та підняття предметів

Для пошуку на локації до всіх предметів додано тег Item. За цим тегом і буде виконуватися пошук. Також на предмети накладено скрипт Outline для відображення обведення останнього.

Розглянемо реалізацію пошуку предметів у програмному коді (див. рис. 3.21).

На початку створюється масив з усіх об’єктів під тегом Item та запускається корутина, яка повторюється кожні пів секунди.

Метод FindClosestItem() перебирає усі об’єкти з масиву, обчислює відстань кожного з них до персонажу та встановлює і повертає найближчий. Для підрахунку відстані було використано функцію sqrMagnitude(), вона повертає квадратне значення, не обчислюючи корінь, що менше навантажує систему.

```
24      void Start()
25      {
26          item = GameObject.FindGameObjectsWithTag("Item");//масив всіх предметів
27          InvokeRepeating(nameof(UpdateTime), 0.5f, 0.5f);
28      }
29
30      //метод пошуку найближчого предмету
31      GameObject FindClosestItem()
32      {
33          distance = 500;//максимальна відстань для пошуку
34          Vector3 position = transform.position;
35          foreach (GameObject go in item)
36          {
37              Vector3 diff = go.transform.position - position;
38              float curDistance = diff.sqrMagnitude;//відстань між предметом та плеєром
39              if (curDistance < distance)
40              {
41                  closest = go;
42                  distance = curDistance;
43              }
44          }
45          return closest;//повертає найближчий об'єкт
46      }
```

Рисунок 3.21 – Частина коду пошуку предметів

Далі в методі UpdateTime(), який запущено корутиною кожні пів секунди, викликається метод FindClosestItem() та отримується найближчий предмет (див. рис. 3.22).

```
51      void UpdateTime()
52      {
53          FindClosestItem();
54          //якщо не змінився найближчий об'єкт
55          if (curClosest == closest)
56          {
57              if (distance > 100)
58              {
59                  curClosest.gameObject.GetComponent<Outline>().enabled = false;
60                  text.text = "";
61                  isNear = false;
62                  StopOutlineAnimator();
63              }
64              else
65              {
66                  if (curClosest.gameObject.GetComponent<Outline>().enabled == false)
67                  {
68                      curClosest.gameObject.GetComponent<Outline>().enabled = true;
69                      text.text = "поруч " + curClosest.GetComponent<Item>().item.itemName;
70
71                      if (!isNear)
72                      {
73                          SetOutlineAnimator();
74                      }
75                  }
76              }
77          }
78          else
79          {
80              curClosest.gameObject.GetComponent<Outline>().enabled = false;
81              if (distance < 100)
82              {
83                  closest.gameObject.GetComponent<Outline>().enabled = true;
84                  text.text = "поруч " + closest.GetComponent<Item>().item.itemName;
85                  SetOutlineAnimator();
86              }
87              curClosest = closest;
88          }
89      }
```

Рисунок 3.22 – Метод UpdateTime() в коді пошуку предметів

Якщо відстань до найближчого предмету менше квадрату 100, тобто менша 10 метрів, то активується обведення цього предмету. Коли відстань більше 10 м, обведення деактивується.

Потім також ведеться порівняння відстані, і якщо вона менша ніж 1 м, то активується кнопка підняття предмету. При натисненні цієї кнопки предмет видаляється та інформація про нього з’являється у панелі інветарю.

Аналогічно до цього принципу реалізовано пошук та збір ресурсів. Тільки у кожного ресурсу свій окремий тег, в залежності від інструменту, яким він збирається.

При застосуванні інструменту активується пошук ресурсу відповідного цьому інструменту. При відстані меншої за 1 м, активується кнопка збору ресурсу.

3.3.4 Розпалення багаття

Для розпалення багаття потрібно попередньо підготувати місце, тобто розкласти каміння через панель крафту. Саме розпалення виконується також через цю панель, треба персонажем узяти у руку кременеве каміння, мати дрова, стояти поруч з місцем розведення та натиснути кнопку для розпалення багаття. Після потрібної анімації у програмному коді накладеному на об’єкт багаття активується система часток та воно розпалюється (див. рис. 3.23).

В об’єкті багаття активується дочірній об’єкт, яким є система часток. Встановлюються попередні параметри трьох систем часток, а саме для вогню, диму та іскор, їх розмір та сила. Також розпочинається корутина с періодом в 1 секунду, в якій відстежується час горіння та сила вогню.

Принцип горіння вогню такий, що його сила та час залежить від кількості покладених до багаття дров. Одна дровина горить 40 секунд та збільшує силу вогню на 1. По закінченню 40 секунд дровина зникає та сила вогню зменшується на 1.

```
53      //розпалення вогню
54      public void StartFire()
55      {
56          transform.GetChild(2).gameObject.SetActive(true);
57          fireStartSize = 0.3f;
58          fireLifeTime = 1f;
59          fireShapeRadius = 0.1f;
60          smokeStartSize = 0.3f;
61          smokeLifeTime = 0.05f;
62          smokeShapeRadius = 0.05f;
63          sparkLifeTime = 1f;
64          firePower = 0;
65          isFire = true;
66          SetFirePower();
67          timeFireLife = 40;
68          //відрахунок секунд у перериваннях по 1с
69          InvokeRepeating("UpdateTime", 0f, 1f);
70      }
```

Рисунок 3.23 – Програмний код активації системи часток

В методі SetFirePower() прораховуються та встановлюються параметри систем часток в залежності від сили вогню та активується підсвітлення. Якщо сила вогню більше, то відповідно і радіус, швидкість та час життя часток збільшується (див. рис. 3.24).

```
19      private void SetFirePower()
20      {
21          ParticleSystem ps = transform.GetChild(2).gameObject.GetComponent<ParticleSystem>();
22          if (firePower > 6) { firePower = 6; }
23          if (firePower < 0) { firePower = 0; }
24          //розрахункові проміжки між силами вогню
25          fireStartSize = (0.5f - 0.3f) / 6 * firePower + 0.3f;
26          fireLifeTime = (4f - 1f) / 6 * firePower + 1f;
27          fireShapeRadius = (0.3f - 0.1f) / 6 * firePower + 0.1f;
28          smokeStartSize = (1f - 0.3f) / 6 * firePower + 0.3f;
29          smokeLifeTime = (5f - 0.05f) / 6 * firePower + 0.05f;
30          smokeShapeRadius = (0.3f - 0.05f) / 6 * firePower + 0.05f;
31          sparkLifeTime = (5f - 1f) / 6 * firePower + 1f;
32          //встановлюємо вогонь
33          ps.startLifetime = fireLifeTime;
34          ps.startSize = fireStartSize;
35          var sh = ps.shape;
36          sh.radius = fireShapeRadius;
37          //встановлюємо дим
38          ps = transform.GetChild(2).GetChild(0).gameObject.GetComponent<ParticleSystem>();
39          ps.startLifetime = smokeLifeTime;
40          ps.startSize = smokeStartSize;
41          sh = ps.shape;
42          sh.radius = smokeShapeRadius;
43          //встановлюємо іскри
44          ps = transform.GetChild(2).GetChild(1).gameObject.GetComponent<ParticleSystem>();
45          ps.startLifetime = sparkLifeTime;
46          dirLight.intensity = (firePower + 1) * 0.25f;
47      }
```

Рисунок 3.24 – Метод встановлення параметрів системи часток

В методі UpdateTime(), який запущено корутиною кожну секунду зменшується час життя вогню на 1 та відстежується його сила горіння (див. рис. 3.25).

Кожні 40 секунд 1 дровина з багаття прибирається та сила вогню зменшується на 1.

При зменшені сили вогню до 0 система часток та підсвітлення деактивуються.

```
75      void UpdateTime()
76      {
77          timeFireLife--;
78          //якщо час вийшов
79          if (timeFireLife <= 0)
80          {
81              firePower--;
82              //якщо закінчилися дрова
83              if (firePower < 0)
84              {
85                  transform.GetChild(1).GetChild(0).gameObject.SetActive(false);
86                  transform.GetChild(2).gameObject.SetActive(false);
87                  isFire = false;
88                  dirLight.intensity = 0f;
89                  CancelInvoke();//відміняє таймер
90              }
91              else
92              {
93                  //зменшує силу на 1 та прибирає 1 дровину
94                  timeFireLife = 40;
95                  transform.GetChild(1).GetChild(firePower + 1).gameObject.SetActive(false);
96                  SetFirePower();
97              }
98          }
99      }
```

Рисунок 3.25 – Метод відстеження сили вогню

Для підтримання та збільшення вогню персонажу потрібно підкидати дрова до багаття.

Для цього йому необхідно стояти поруч з вогнищем, та мати у панелі швидкого доступу дрова.

При застосуванні дров з панелі швидкого доступу ведеться перевірка чи поруч багаття та чи розведено вогнище. Якщо умови виконано, то у програмному коді збільшується сила вогню та зменшується кількість дров у слоті (див. рис. 3.26).

```
166             //якщо використовує дрова
167             if (slot.item.itemName.ToString() == "Дрова")
168             {
169                 //отримує чи поруч багаття
170                 NearestCampFire nearestCampFire = FindObjectOfType<NearestCampFire>();
171                 bool isNearCamp = nearestCampFire.GetCampFireIsNear();
172                 //отримує чи багаття горить
173                 CampFirePower campFirePower = FindObjectOfType<CampFirePower>();
174                 bool isFire = campFirePower.GetIsFire();
175             if (isNearCamp && isFire)
176                 {
177                     GameObject campFire = nearestCampFire.GetCampFire();
178                 //розгортаємо персонаж для анімації
179                 Vector3 targetPosition = new Vector3(campFire.transform.position.x,
180                     pers.transform.position.y, campFire.transform.position.z);
181                 pers.transform.LookAt(targetPosition);
182                 //додає дрова у вогонь
183                 campFire.GetComponent<CampFirePower>().AddFirePower();
184                     if (slot.amount <= 1)
185                     {
186                         slot.GetComponentInChildren<DragAndDropItem>().NullifySlotData();
187                     }
188                     else
189                     {
190                         slot.amount--;
191                         slot.itemAmountText.text = slot.GetComponent<InventorySlot>().amount.ToString();
192                     }
193                 }
194             }
```

Рисунок 3.26 – Використання дров з панелі швидкого доступу

3.4 Застосування хмарних сервісів у проєкті

Для створення рейтингу гравців у грі необхідно додати БД у проєкт та розташувати її у хмарі для постійного доступу до неї всіх клієнтів. Для цього використаємо безкоштовний сервіс Google Firebase – це платформа для розробки мобільних і вебдодатків, що дає змогу створювати високоякісні додатки, залучати лояльних користувачів і збільшувати прибутки [25].

Скористаємось службами для розробки Firebase Auth та Realtime Database. Firebase Auth – це служба, яка може аутентифікувати користувачів, використовуючи лише код на стороні клієнта. Realtime Database надає в режимі реального часу базу даних та бекенд як службу. Ця служба надає розробникам за стосунків API, який дозволяє синхронізувати дані застосунків між клієнтами та зберігати їх у хмарі Firebase. Створимо на сервісі Google Firebase новий проєкт та під’єднаємо до нього наш додаток, зробимо усі налагодження доступу та застосуємо необхідні служби (див. рис. 3.27).

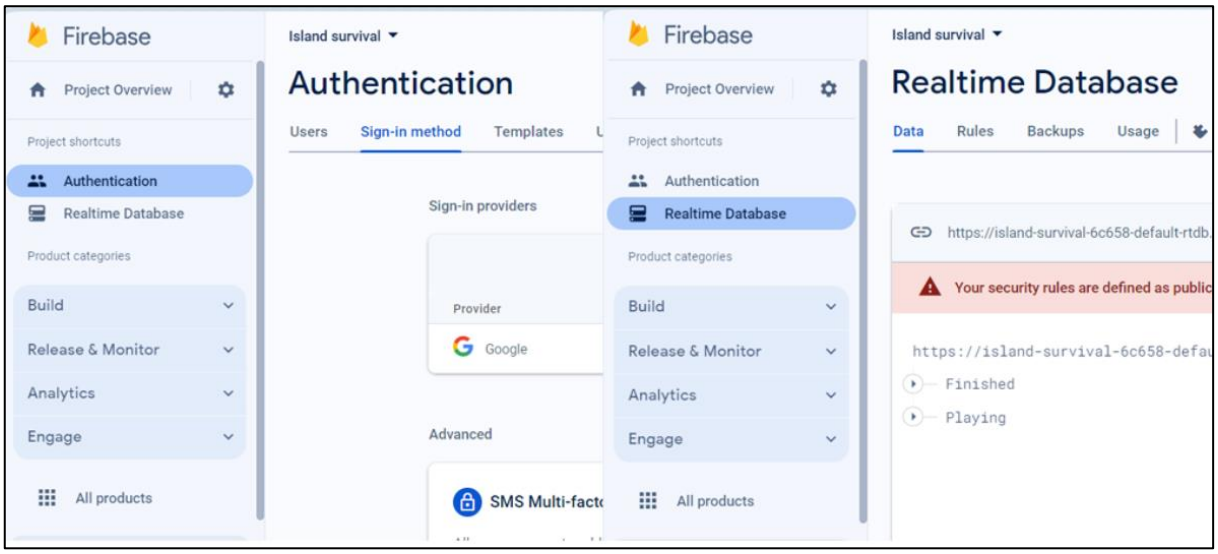

Рисунок 3.27 – Застосування хмарного ресурсу Firebase

Для аутентифікації та доступу до БД застосуємо метод Google sign-in. Це дуже зручний та легкий метод для мобільних додатків, тому як в більшості випадків усі мобільні пристрої завжди зостаються аутентифікованими в Google, та процес входу у додатку зводиться до одного – двох натиснень кнопки.

Додамо програмний код до проєкту для аутентифікації у Firebase та роботи з БД.

Для побудови рейтингу гравців необхідно до БД занести дані про гравців, а потім їх використовувати. Розглянемо частину коду роботи з базою (див. рис. 3.28).

```
18      //зберігає дані у БД фаєрбейз
19      public void SavaDatabase(string where, string name, string id, string imageURL, int hour, int seconds)
20      {
21          dbRef = FirebaseDatabase.DefaultInstance.RootReference;
22          dbRef.Child(where).Child(id).Child("Name").SetValueAsync(name);
23          dbRef.Child(where).Child(id).Child("Image").SetValueAsync(imageURL);
24          int minutes = seconds / 60 + hour * 60;
25          dbRef.Child(where).Child(id).Child("Time").SetValueAsync(minutes);
26      }
```

Рисунок 3.28 – Частина коду запису у БД

Після аутентифікації та отримання доступу до Realtime Database зберігаємо дані у БД. Для рейтингу гравців достатньо зберегти дані про ідентифікатор користувача Google аккаунту, ім’я, посилання на аватарку та

кількість витраченого часу на проходження гри.

Як показано на рисунку 3.27 БД поділена на 2 частини: Playing (ті що грають) та Finished (ті що закінчили). Кожні 5 хвилин та при виході з гри БД оновлюється даними про гравця, коли той успішно пройшов гру, то дані про нього видаляються з частини тих, що грають і переносяться у частину закінчивши гру з остаточним часом витраченим на проходження.

Кожен гравець має можливість через основне меню переглянути таблицю рейтингу гравців, що успішно залишили острів за часом витраченим на проходження гри. Розглянемо частину коду з побудовою рейтингу гравців (див. рис. 3.29).

```
29      public IEnumerator LoadDatabase(string id)
30      {
31          dbRef = FirebaseDatabase.DefaultInstance.RootReference;
32          //відсортируємо за часом та обмежимо до 10 кращих
33          var user = dbRef.Child("Finished").OrderByChild("Time").LimitToFirst(10).GetValueAsync();
34          yield return new WaitUntil(predicate: () => user.IsCompleted);
35          if (user.Exception != null)
36          {
37              Debug.LogError(user.Exception);
38          }
39          else
40          {
41              DataSnapshot snapshot = user.Result;
42              int i = 0;
43              //цикл перебирає 10 гравців та заносить дані у таблицю
44              foreach (DataSnapshot childSnapshot in snapshot.Children)
45              {
46                  string text2 = childSnapshot.Child("Name").Value.ToString();
47                  int minutes = int.Parse(childSnapshot.Child("Time").Value.ToString());
48                  int hour = minutes / 60;
49                  minutes = minutes % 60;
50                  string text3 = hour.ToString() + "год " + minutes.ToString() + "хв";
51                  panelRating.GetChild(i).GetChild(2).gameObject.GetComponent<Text>().text = text2;
52                  panelRating.GetChild(i).GetChild(3).gameObject.GetComponent<Text>().text = text3;
53                  //загружає дані аватарки з гугл аккаунта
54                  if (childSnapshot.Child("Image").Value.ToString() != "NULL")
55                  {
56                      var request = UnityWebRequestTexture.GetTexture(childSnapshot.Child("Image").Value.ToString());
57                      yield return request.SendWebRequest();
58                      if (request.result == UnityWebRequest.Result.Success)
59                      {
60                          Texture2D tex = new Texture2D(1, 1);
61                          tex = DownloadHandlerTexture.GetContent(request);
62                          if (tex != null)
63                          {
64                              Sprite image = Sprite.Create(tex, new Rect(0, 0, tex.width, tex.height), new Vector2(0.5f, 0.5f));
65                              panelRating.GetChild(i).GetChild(0).gameObject.GetComponent<Image>().sprite = image;
66                          }
67                      }
68                  }
69                  i++;
70              }
71          }
72      }
```

Рисунок 3.29 – Частина коду з побудовою рейтингу гравців

Отримавши доступ до БД сортуємо частину з гравцями, що закінчили гру, обмежуємо кількість до 10 та завантажуємо ці дані. Якщо після запиту інформація отримана успішно, то розпочинаємо цикл, в якому заповнюємо панель рейтингу гравців з отриманої інформації. Так як Realtime Database є динамічною, то кожного разу при зчитуванні даних з БД отримуємо дані у вигляді моментального скриншоту бази DataSnapshot, і вже з цим екземпляром працюємо.

У циклі беремо дані з екземпляру DataSnapshot та зведемо до таблиці панелі рейтингу гравців, водночас завантажуючи та перетворюючи на графічний елемент картинку з аватаркою клієнта Google.

3.5 Оптимізація застосунку

Одним із дуже важливих етапів розробки проєкту є його оптимізація. Це процес модифікації програмної системи для вдосконалення та підвищення її ефективності.

Наш проєкт містить багато 3Д об’єктів одночасно розташованих на локації, особливо цю роль відіграють дерева у лісі, і на обробку них витрачається дуже багато ресурсів.

Більшість мобільних приладів не володіють потужними графічними ресурсами.

Тому оптимізація здебільш буде спрямована на зниження вимог до обробки графіки та рендерингу.

Тому у проєкт було додано URP, який вже було вище розглянуто. Та створено декілька варіантів його налагодження (див. рис. 3.30).

У кожному із варіантів було налагоджено якість рендерингу, глибину текстур, ступені та якість проєктування тіней тощо.

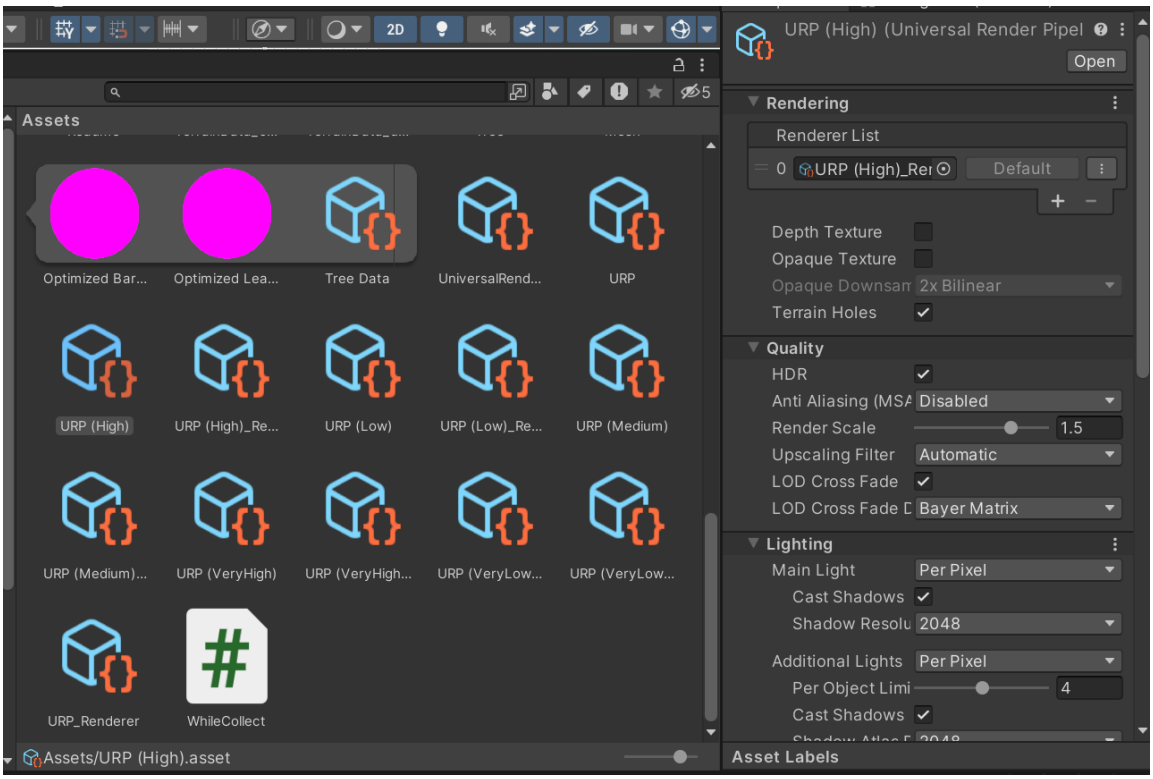

Рисунок 3.30 – Налагодження URP

Ці варіанти було додано до основних налагоджень проєкту у категорії якості графіки, та в панелі самої гри до вибору якості графіки (див. рис. 3.31).

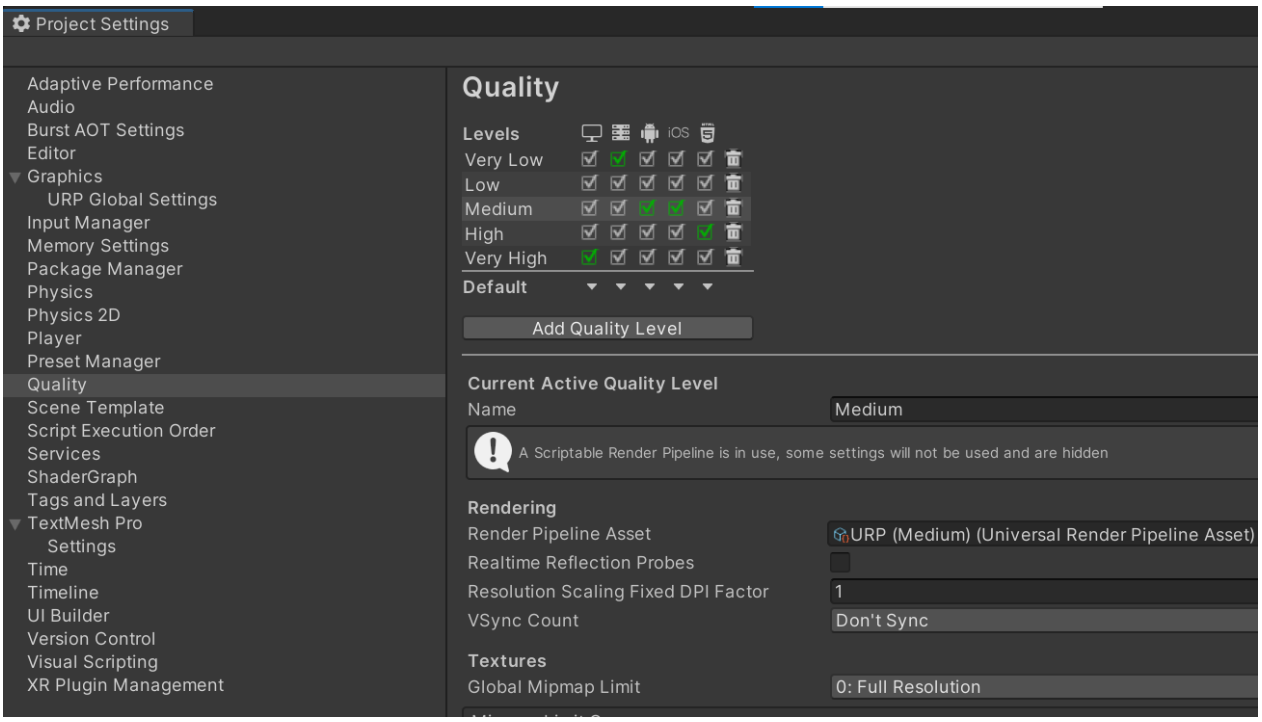

Рисунок 3.31 – Встановлення якості графіки проєкту

У застосунку було збільшено продуктивність шляхом додавання до нього системи мешей різного рівня деталізації LOD (Level of Detail).

Цей метод оптимізації використовує кілька мешей кожного об'єкта. Меші являють собою один і той же об'єкт із зменшеною геометричною деталізацією. Ідея полягає в тому, що низько-деталізовані меші показуються, коли об'єкт знаходиться далеко від камери. Тому різницю між ними та високою деталізацією не буде помічено, при цьому продуктивність зросте [26].

Для дерев було встановлено LOD трьох рівнів з різною кількістю полігонів для кожного з них. Коли персонаж далеко від 3Д об’єкту, то він взагалі не відображається і система не витрачає ресурсів на його рендеринг (див. рис. 3.32).

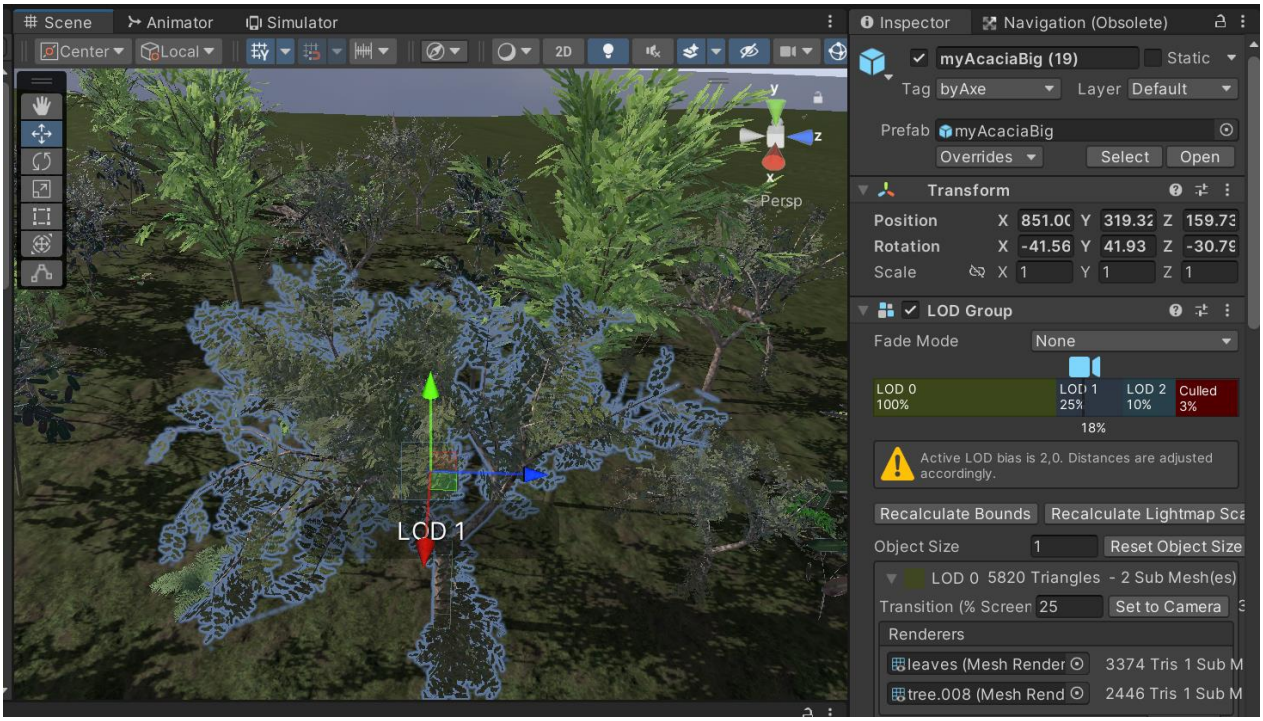

Рисунок 3.32 – Система LOD

Для кожного варіанту якості графіки було встановлено свою систему відображення LOD. При низькій якості кількість ступенів більша, а при високій менша, аж до не використання системи взагалі при ультра налагодженнях якості графіки.

3.6 Тестування готового продукту

Тестування програмного забезпечення – це процес, під час якого проводяться експерименти для виявлення помилок і дефектів у програмі. Воно дає змогу переконатися, що ПЗ працює коректно, відповідає вимогам і очікуванням користувачів, а також працює надійно і безпечно.

Для тестування продукту було обрано метод ручного тестування – це процес, у якому тестувальники виконують тестові сценарії та перевіряють функціональність програмного продукту вручну. Вони стежать за кожним кроком тестового процесу й активно взаємодіють із застосунком, перевіряючи його працездатність, користувацький інтерфейс і відповідність вимогам.

Перевагами ручного тестування є:

– гнучкість і адаптивність: тестувальники можуть швидко адаптуватися до змін і вносити корективи в тестові сценарії;
– інтуїтивне розуміння: ручне тестування дає змогу виявляти проблеми, які можуть бути упущені в автоматичному тестуванні.

Гру було завантажено до Google Play Market [27], пройдено усі внутрішні перевірки та схвалено на викладення для закритого тестування (див. рис. 3.33).

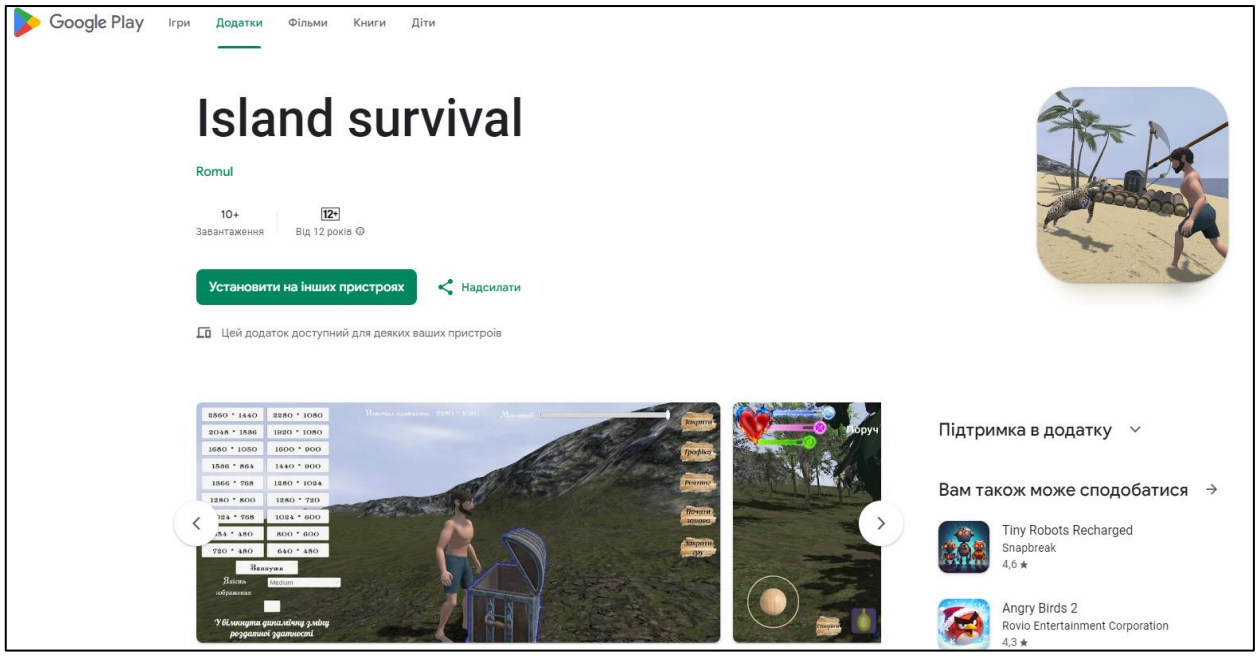

Рисунок 3.33 – Відображення застосунку у Goole Play

Для закритого тестування застосунку було залучено 33 особи з різними мобільними пристроями, такими як:

– смартфони;

– планшети.

Кожен з гаджетів мав свої технічні параметри:

– роздатна здатність;

– центральний процесор;

– графічний процесор;

– версія операційної системи Android;

– та інші.

Усі тестувальники встановили гру на свій пристрій та зареєструвалися у Firebase, використовуючи метод Google sign-in та свій акаунт Google (див. рис. 3.34).

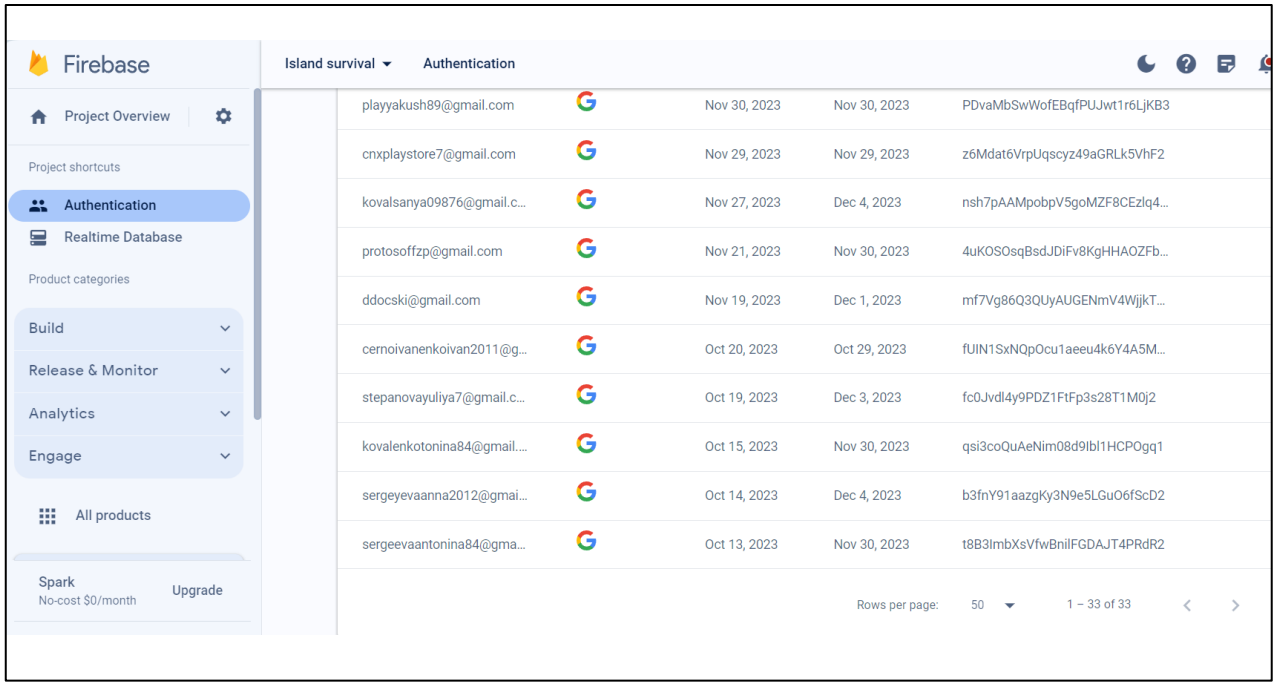

Рисунок 3.34 – Список зареєстрованих тестувальників

Так як гру було адаптовано під усі мобільні пристрої, то у всіх тестувальників не виникло жодних проблем при встановленні, реєстрації та графічному налагоджені застосунку під свій пристрій (див. рис. 3.35).

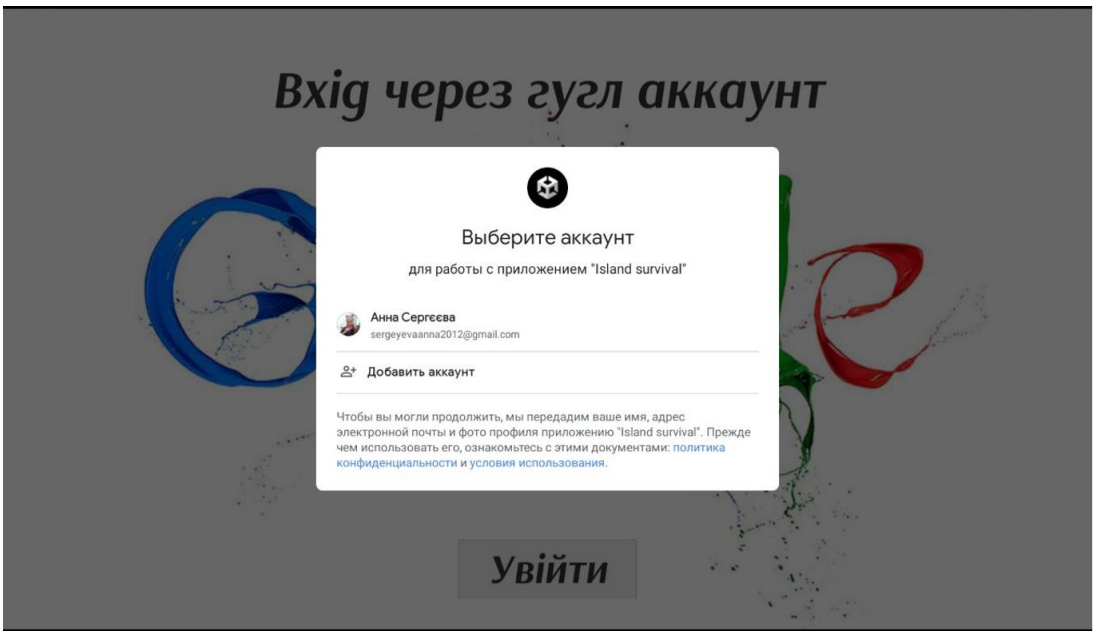

Рисунок 3.35 – Приклад реєстрації через Google sign-in

Також було протестовано зміну графічних налагоджень: перемикання розданої здатності, масштабування, встановлення різної якості зображення та активацію динамічної зміни розданої здатності (див. рис. 3.36).

Рисунок 3.36 – Тестування зміни графічних налагоджень

У всіх тестувальників при проходженні гри, виході та вході в неї успішно відбувалось збереження та завантаження прогресу. Проблем у синхронізації з БД Firebase та її оновленні не виявлено (див. рис. 3.37).

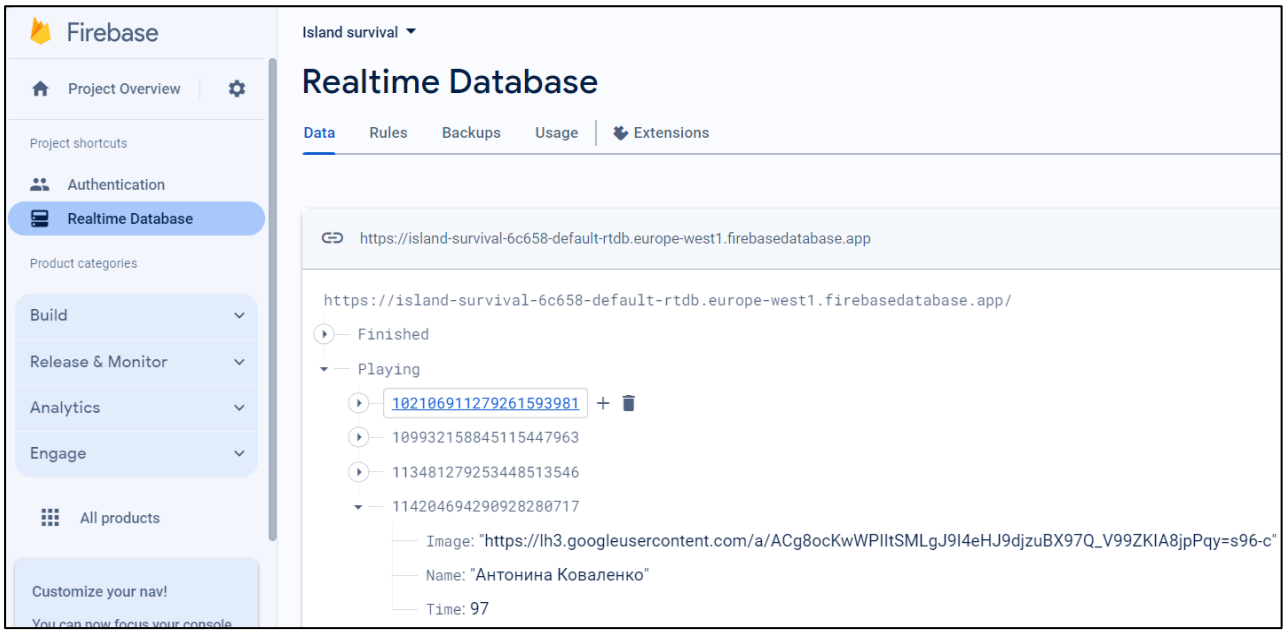

Рисунок 3.37 – Збережені дані у БД Realtime Database

Протестовано усі види збору ресурсів. Не виявлено помилок та багів (див. рис. 3.38).

Рисунок 3.38 – Тестування збору ресурсів

Тестування системи та роботи панелі крафту не виявило жодних помилок (див. рис. 3.39).

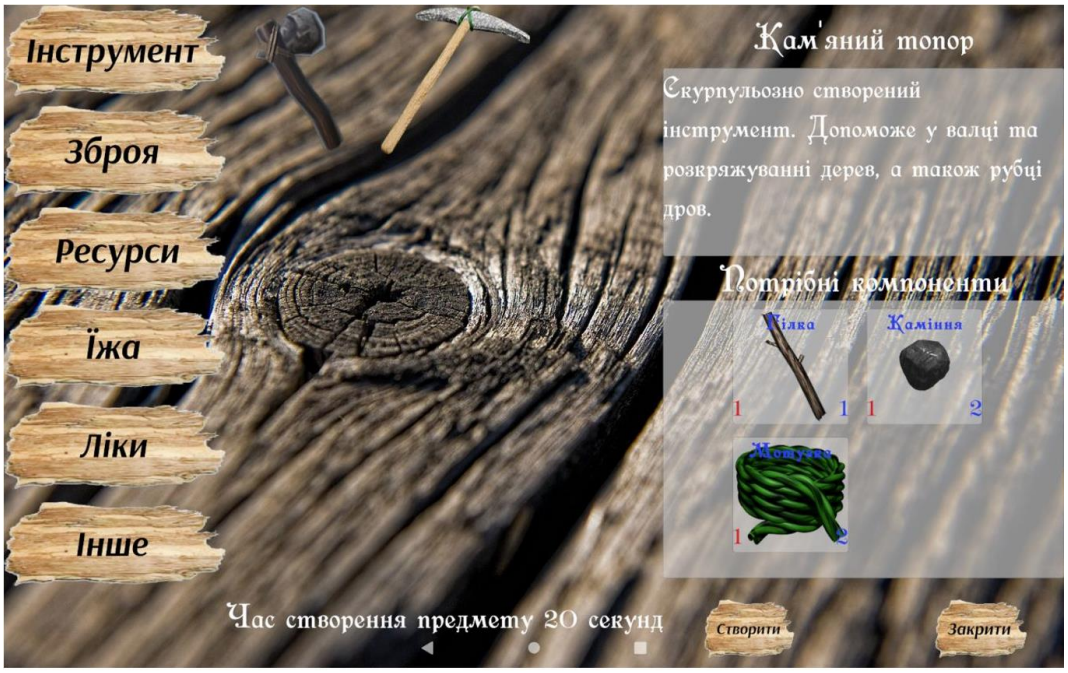

Рисунок 3.39 – Тестування панелі крафту

Приємне враження пособі залишає захід та схід сонця (див. рис. 3.40).

Рисунок 3.40 – Захід сонця у грі

Система зміни дня та ночі допомагає гравцеві зануритись у систему гри та робить проходження більш реалістичним (див. рис. 3.41).

Рисунок 3.41 – Система зміни дня та ночі

При полюванні більшістю тестувальників виявлено, що цей процес доволі складний, бо треба враховувати фізику польоту зброї та поведінку тварин, що надає невелику напруженість у проходжені, але кожному вдалось вполювати здобич (див. рис. 3.42).

Рисунок 3.42 – Тестування режиму полювання

Взагалі в процесі тестування було виявлено декілька незначних багів та помилок, які в процесі оптимізації та відлагодження було усунено.

Загальне враження від проходження гри у більшості тестувальників склалось позитивне.

Відгук одного з тестувальників: «Потрапляючи у гру занурюєшся у світ цікавих пригод, відкриваєш для себе нові відчуття. Це дає уявлення а що буде, якщо потрапити на безлюдний острів. Ніби пройти курс для скаутів: розвести багаття, вполювати здобич, вберегтись від хижаків, побудувати пліт тощо. Далеко не все вдається легко, але ж в житті це не може бути легким. За рівнем емоцій гра має 10 з 10, бо відчувається зв’язок з персонажем, переживання за його благополуччя та успіх у досягненні цілі. Дякую за докладені зусилля в створенні якісного продукту.»

Середній час проходження гри складає близько 4 годин (див. рис. 3.43).

Рисунок 3.43 – Рейтинг кращих гравців

ВИСНОВКИ

У даній кваліфікаційній роботі було досліджено та обґрунтовано актуальність створення мобільних ігор. Розглянуто та визначено основні характеристики та особливості геймплею сучасних ігор.

Для реалізації мобільного застосунку «Island survival» було розроблено концепцію, побудовано алгоритм взаємодії компонентів, опрацьовано механіку гри. Додаток створено засобами Unity, Blender та C#.

В процесі розробки проєкту було створено 76 3Д об’єктів, 56 анімацій, написано 92 скрипти та написано штучний інтелект поведінки тварин.

Мобільну гру було завантажено до сервісу Google Play Market, протестовано та перевірено на працездатність. Зібрано відгуки та перше враження від тестувальників.

У майбутньому застосунок планується розширити додатковими функціями та можливостями, такими як:

– додати різноманіття рослин та тварин;
– збільшити кількість факторів небезпеки: природні стихії, небезпечні тварини та комахи;
– збільшити перелік для крафту;
– додати у гру мультиплеєрний режим.

«Island survival» є представником класичної гри на виживання зі своєю власною цікавою механікою та способом проходження. Гра є захоплюючою та направлена не відриваи від реальності, надає змогу пройти випробування реалістичні, але недоступні в реальному житті. Ніби побудувати модель поведінки у складних умовах з визначенням «а що, якби..». На основі цих даних ми можемо побудувати статистичну модель.

Як виконавець цього проєкту вважаю його успішним. Застосунок уявляє собою готовий продукт, придатний для використання.

ПЕРЕЛІК ПОСИЛАНЬ

1. Ugami B. Not Just For Kids: Video Games Lead The Entertainment Industry. Forbs. 2022. URL: https://www.forbes.com/sites/forbeseq/2022/07/15/not-just-for-kids-video-games-lead-the-entertainment-industry/?sh=5965b041112c (дата звернення: 11.09.2023).
2. Top 8 Entertainment Trends in 2023. StartUs Insights. URL: https://www.startus-insights.com/innovators-guide/entertainment-trends-innovations/ (дата звернення: 11.09.2023).
3. Divers G. Gaming Industry Dominates as the Highest-Grossing Entertainment Industry. Gamerhub. 2023. URL: https://gamerhub.co.uk/gaming-industry-dominates-as-the-highest-grossing-entertainment-industry/ (дата звернення: 11.09.2023).
4. Why Gaming Is The Future Of Entertainment. Daily Trust. 2023. URL: https://dailytrust.com/why-gaming-is-the-future-of-entertainment/ (дата звернення: 11.09.2023).
5. Byers K. The 7 Biggest Entertainment Trends (2023-2025). Exploding Topics. 2023. URL: https://explodingtopics.com/blog/entertainment-trends (дата звернення: 11.09.2023).
6. Clement J. Video game industry – Statistics & Facts. Statista. 2023. URL: https://www.statista.com/topics/868/video-games/#topicOverview (дата звернення: 11.09.2023).
7. Harbuzova A. В Україні на 66% більше нових інді-ігор. Головне зі звіту Unity про ігрову індустрію. Gamedev. 2023. URL: https://gamedev.dou.ua/news/unity-gaming-report-2023-trend-and-changes/ (дата звернення: 11.09.2023).
8. Unity Game Engine. URL: https://unity.com (дата звернення: 12.09.2023).
9. Unreal engine. URL: https://www.unrealengine.com/en-US (дата звернення: 12.09.2023).

10. GameMaker Studio. URL: https://gamemaker.io/en (дата звернення: 12.09.2023).
11. Blender. URL: https://www.blender.org (дата звернення: 13.09.2023).
12. Minecraft. URL: https://www.minecraft.net (дата звернення: 15.09.2023).
13. Resident Evil 4. URL: https://store.steampowered.com/app/2050650/Resident_Evil_4/ (дата звернення: 15.09.2023).
14. The Last of Us Part I. URL: https://store.steampowered.com/app/1888930/The_Last_of_Us_Part_I/ (дата звернення: 15.09.2023).
15. Surviving Mars. URL: https://www.paradoxinteractive.com/games/surviving-mars/about (дата звернення: 15.09.2023).
16. Navigation and Pathfinding (AI Navigation). Unity User Manual. 2015. URL: https://docs.unity3d.com/Packages/com.unity.ai.navigation@2.0/manual/index.html (дата звернення: 28.09.2023).
17. Simple Water Shader URP. Unity Asset Store. 2020. URL: https://assetstore.unity.com/packages/2d/textures-materials/water/simple-water-shader-urp-191449 (дата звернення: 20.09.2023).
18. Mesh Collider component reference. Unity User Manual. 2015. URL: https://docs.unity3d.com/Manual/class-MeshCollider.html (дата звернення: 21.09.2023).
19. Universal Render Pipeline (URP). URL: https://unity.com/srp/universal-render-pipeline (дата звернення: 08.10.2023).
20. Troelsen A. Pro C# 7: With .NET and .NET Core. Minneapolis, Minnesota. 2017. 1372 p.
21. C# documentation. URL: https://docs.microsoft.com/uk-ua/dotnet/csharp/ (дата звернення: 17.09.2023).
22. Microsoft Visual Studio 2022. 2023. URL: https://visualstudio.microsoft.com/ (дата звернення: 17.09.2023).
23. Serialization in .NET. .NET fundamentals documentation. 2023. URL:

https://learn.microsoft.com/en-us/dotnet/standard/serialization/ (дата звернення: 19.09.2023).

24. ScriptableObject. Unity User Manual. 2015. URL: https://docs.unity3d.com/Manual/class-ScriptableObject.html (дата звернення: 19.09.2023).
25. Connecting apps to the Firebase platform . Google reference center. 2023. URL: https://support.google.com/admob/answer/6360054?hl=uk (дата звернення: 14.10.2023).
26. Level of Detail (LOD) for meshes. Unity User Manual. URL: https://docs.unity3d.com/Manual/LevelOfDetail.html (дата звернення: 18.10.2023).
27. Serhieiev R. Island survival. Google Play. URL: https://play.google.com/store/apps/details?id=com.Romul.Islandsurvival (дата звернення: 30.10.2023).