

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Комп'ютерних наук і технологій
(повне найменування факультету)

Комп'ютерні системи та мережі
(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалаврський
(ступінь вищої освіти)

на тему: РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ ГРИ-КВЕСТУ
НА БАЗІ UNITY

Виконав(ла): студент(ка) 4 курсу,
групи КНТз-513сп

Спеціальності
123 Комп'ютерна інженерія
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Комп'ютерна інженерія
(назва освітньої програми (спеціалізації))

ЗАБРОДІН О.О.
(ПРИЗВИЩЕ та ініціали)

Керівник ТЯГУНОВА М.Ю.
(ПРИЗВИЩЕ та ініціали)

Рецензент ЩЕРБАКОВ В.І.
(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет Комп'ютерних наук і технологій
Кафедра комп'ютерних систем та мереж
Ступінь вищої освіти бакалаврський
Спеціальність 123 Комп'ютерна інженерія
(код і найменування)
Освітня програма (спеціалізація) Комп'ютерна інженерія
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри КУДЕРМЕТОВ Р.К.

«14» квітня 2026 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ЗАБРОДІНА Олега Олексійовича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка мобільного додатку гри-квесту на базі Unity

керівник проєкту (роботи) к.т.н., доцент, Тягунова М.Ю.,

(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від «08» квітня 2026 року № 151

2. Строк подання студентом проєкту (роботи) 01.06.2026 р.

3. Вихідні дані до проєкту (роботи) інструментальне середовище розробки Unity, ігрові механіки 2D гри у жанрі квест, технічна документація рушія, графічні ресурси у стилі піксельної графіки

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз предметної області;

2. Проєктування та налаштування програмного забезпечення

3. Реалізація гри;

4. Тестування роботи системи

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

12 слайдів презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4	ТЯГУНОВА М.Ю.		
Нормоконтроль	ДЬЯЧУК Т.С.		

7. Дата видачі завдання «14» квітня 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Аналіз предметної області	до 18.04.2026	
2	Визначення та аналіз вимог до програмного забезпечення	до 22.04.2026	
3	Аналіз програмного забезпечення	до 24.04.2026	
4	Вибір програмного та апаратного забезпечення системи	до 28.04.2026	
5	Збір ассетів для розробки	до 01.05.2026	
6	Проектування архітектури програмного забезпечення	до 05.05.2026	
7	Проектування інтерфейсу користувача	до 22.05.2026	
8	Тестування гри	до 24.05.2026	
9	Оформлення пояснювальної записки	до 25.05.2026	
10	Проходження нормоконтролю	до 01.06.2026	
11	Перевірка на наявність академічного плагіату	до 03.06.2026	
12	Проходження рецензування	до 10.06.2026	

Студент(ка)

_____ (підпис)

Олег ЗАБРОДІН

(Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

_____ (підпис)

Марія ТЯГУНОВА

(Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
74 с., 2 табл., 64 рис., 15 джерел.

МОБІЛЬНИЙ ДОДАТОК, ГРА-КВЕСТ, UNITY ENGINE, C#, ПІКСЕЛЬНА ГРАФІКА, ПРОГРАМУВАННЯ

Об'єкт розробки – мобільний додаток на базі Unity Engine.

Метою роботи є забезпечення проведення вільного часу людини з користю за рахунок використання розробленої мобільної гри-квесту на базі Unity Engine.

Структурно робота побудована за принципом послідовного переходу від теоретичного обґрунтування до до практичного втілення.

Робота починається з аналізу існуючих технологічних рішень та порівняння ігрових рушіїв, в результаті якого було визначено інструментарій Unity як оптимальну базу для проєкту.

Далі описано архітектурне проєктування системи, детально розкрито потенціал обраної платформи та сформовано каркас майбутнього додатку.

Заклучна частина роботи повністю присвячена практичній інженерії. В ній описано систематизацію та підготовку графічних компонентів, кодування логіки додатку.

Логічним підсумком роботи став робочий прототип гри-квесту.

ЗМІСТ

Скорочення та умовні позначки	6
Вступ	7
1 Аналіз предметної області	9
1.1 Цілі розробки гри	10
1.2 Вибір платформ та мови програмування	11
1.3 Вибір двигунів для мобільних ігор	16
1.3.1 Рушій Unity Engine	15
1.3.2 Рушій Defold	16
1.3.3 Рушій Cocos Creator.....	16
2 Проєктування та налаштування програмного забезпечення	18
2.2 Скриптинг у Unity	21
2.3 Переваги Unity	26
2.4 Архітектура програмного забезпечення гри	27
3.1 Керування асетами та структура файлової системи додатка	29
3.2 Створення Game Object.....	33
3.3 Написання скрипта.....	42
4 Тестування роботи системи	51
Висновки.....	57
Перелік джерел посилання	58
Додаток А Лістинги скриптів	60
Додаток Б Слайди презентації.....	69

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ФС	–	Файлова система
IDE	–	Integrated Development Environment, Інтегроване середовище розробки
UI	–	User interface, Інтерфейс користувача
GO	–	Game Object, Ігровий об'єкт
VCS	–	Version Control System, Система контролю версій
GUI	–	Graphical User Interface, Графічний інтерфейс користувача
ISA	–	Input System Actions, Система вводу дій

ВСТУП

Сучасний етап розвитку інформаційного суспільства характеризується тотальною цифровізацією, яка дуже сильно змінює людську взаємодію з навколишнім світом. За короткий проміжок часу мобільні технології пройшли еволюцію від вузькоспеціалізованих засобів зв'язку до персональних високопродуктивних обчислювальних машин. Смартфон у сучасних реаліях став обов'язковою частиною повсякденності, виступаючи головним інструментом для вирішення широкого спектру побутових, комунікаційних, освітніх та професійних завдань. Портативність, постійна можливість вийти в інтернет та висока обчислювальна потужність мобільних пристроїв змістили вектор споживання цифрового контенту у бік мобільних платформ, сформувавши абсолютно нові сценарії поведінки користувачів.

В умовах безперервної цифровізації особливого масштабу набула індустрія мобільних ігор. Сьогодні мобільний геймінг вже не можна розглядати виключно як сферу розваг або пасивного проведення часу. Він перетворився на потужне інтерактивне середовище та соціальний і технічний феномен. Динамічний темп життя та високе інформаційне навантаження вимагають від сучасної людини інструментів для швидкого розумового переключення та розвантаження, або навпаки, стимуляції. Мобільні ігрові застосунки стали ефективним засобом навчання та розвитку логічного мислення, адже вони доступні користувачу у будь-який момент часу та пропонують миттєве занурення в ігровий процес.

Особливе місце в структурі мобільного геймінгу посідають сюжетні логічні ігри, зокрема жанру гри-квесту. Користувач під час взаємодії з грою вчиться виявляти приховані закономірності, прогнозувати результати своїх дій та приймати рішення в межах заданого ігрового всесвіту. Таким чином, мобільний квест виступає як корисна альтернатива монотонному споживанню інформації, перетворюючи дозвілля на процес активного інтелектуального тренінгу.

Ефективним та найбільш затребуваним інструментом для створення мобільних ігор у сучасній індустрії розробки ігор є середовище розробки Unity Engine. Використання цього рушія дозволяє значно оптимізувати процес створення гри завдяки потужним вбудованим підсистемам роботи з 2D графікою, анімацією, фізикою та звуковим супроводом. Мова програмування C#, що є базовою для Unity, забезпечує високу швидкість написання логіки та об'єктно орієнтований підхід до структурування проєкту. Таким чином, розробка інтерактивного мобільного квесту на базі Unity Engine є актуальним напрямком, який відповідає сучасним вимогам ринку програмного забезпечення та концепції корисного використання цифрового простору.

Метою роботи є комплексне проєктування, розробка архітектури та програмна реалізація мобільного ігрового застосунку в жанрі 2D квесту на базі сучасного рушія Unity, що спрямований на організацію корисного та інтелектуального дозвілля користувачів мобільних пристроїв.

Для досягнення поставленої мети в межах кваліфікаційної роботи необхідно виконати такі завдання:

- провести детальний аналіз предметної області;
- обґрунтувати вибір середовища розробки Unity, мови програмування C# та супутнього інструментарію для підготовки графічного контенту;
- визначити та систематизувати технічні вимоги до цільових мобільних платформ для забезпечення стабільного функціонування додатку;
- спроектувати програмну архітектуру застосунку, розробити схему взаємодії класів, підготувати візуальні асети;
- реалізувати програмну логіку, включаючи системи взаємодії з предметами та обробки станів гри;
- виконати комплексне тестування розробленого додатку, провести аналіз результатів і сформулювати висновки щодо ефективності розробки.

В ході виконання дипломної роботи було використано ДСТУ3008,8320 та бланки [1-3].

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Процес створення сучасного мобільного додатку є комплексною інженерною та творчою діяльністю, яка поєднує в собі передові практики програмування, комп'ютерної графіки та проєктування взаємодії з користувачем. Структура індустрії розробки інтерактивних додатків зазнала суттєвих змін: якщо на етапі становлення ринку створення програмного продукту могло здійснюватися одноосібно або невеликими групами ентузіастів, то сьогодні розробка масштабних проєктів вимагає залучення багатопрофільних команд і значних інвестицій. В той же час завдяки розвитку моделей цифрової дистрибуції та еволюції кросплатформних засобів розробки, незалежні автори та невеликі студії отримали можливість виводити власні програмні рішення на глобальний ринок без посередництва великих видавців.

У межах аналізу ринку мобільних додатків особливу увагу приділяють класифікації додатків за їх функціональною складністю, механіками взаємодії та вимогами до апаратних ресурсів. Тривалий час домінуючу позицію на ринку займав сегмент казуальних ігор, що характеризуються простим ігровим процесом, низькими вимогами до вмінь користувача та орієнтацією на короткі ігрові сесії.

Проте з підвищенням продуктивності мобільних процесорів та графічних прискорювачів вектор споживання змістився у бік більш складних, орієнтованих на сюжет та інтелектуальних жанрів, серед яких велике місце займають ігри-квести та логічні головоломки.

Жанр мобільного квесту є специфічним класом програмного забезпечення, ігровий процес якого зосереджений на вирішенні інтелектуальних завдань, дослідженні візуального простору та аналізі текстової чи графічної інформації. На відміну від динамічних аркад, де ключовим фактором успіху є швидкість реакції користувача, проєкти в жанрі квест базуються на когнітивних механіках взаємодії.

До базових механік квесту належать:

- механіка дослідження: взаємодія з об'єктами ігрового світу для виявлення підказок чи елементів задач;
- механіка тригерів та подій: активація нових станів ігрового світу після виконання користувачем певної послідовності дій;
- механіка менеджменту інвентарю: збір, комбінування знайдених предметів для їх подальшого використання у розв'язанні завдань;
- механіка діалогових систем: отримання інформації та розвиток сюжетної лінії через інтерактивну взаємодію з текстовими компонентами.

Аналіз статистичних даних розвитку мобільного сегмента свідчить про стабільне зростання інтересу аудиторії до продуктів, які вимагають активної ментальної залученості. Сучасні користувачі розглядають інтерактивні логічні застосунки як ефективний засіб для тренування когнітивних функцій та короткострокової релаксації. Динаміка завантажень таких додатків демонструє позитивний тренд, що підтверджує актуальність розробки нових рішень у цьому напрямку.

Створення конкурентоспроможного продукту приховує в собі низку інженерних викликів. Зайва спрощеність логічної структури може призвести до швидкої втрати інтересу з боку цільової аудиторії, тоді як надмірна складність відштовхне користувача.

Таким чином, дослідження предметної області підтверджує необхідність глибокого пророблення архітектури проєкту ще на етапі його планування.

1.1 Цілі розробки гри

Головною метою даного дипломного проєкту є проєктування, розробка та програмна реалізація мобільного ігрового застосунку в жанрі квест на базі рушія

Unity. На відміну від казуального сегмента, де основна увага приділяється монотонним механікам швидкого занурення та короткостроковим ігровим сесіям, розробка квесту передбачає створення цілісного інтерактивного продукту з глибоким логічним складником та сюжетним наповненням. Застосунок орієнтований на утримання уваги користувача через інтелектуальні виклики, дослідження ігрового простору та вирішення послідовних логічних ланцюжків.

Для досягнення поставленої мети в рамках інженерного проєктування необхідно реалізувати такі цільові завдання:

- створення інтуїтивно зрозумілого інтерфейсу (UI/UX): Оскільки взаємодія з мобільним застосунком відбувається безпосередньо через сенсорний екран смартфона, критично важливо розробити таку схему керування, яка б мінімізувала час на адаптацію користувача. Графічні елементи інтерфейсу, вікна меню та інтерактивні зони мають бути оптимізовані під різні роздільні здатності мобільних пристроїв;

- реалізація гнучкої архітектури ігрової логіки: Програмний комплекс має базуватися на об'єктно-орієнтованому підході засобами мови C#. Ключовою ціллю є побудова надійної системи тригерів та менеджерів станів, які забезпечують коректну обробку подій під час взаємодії гравця з предметами та локаціями;

- оптимізація використання апаратних ресурсів: Проєкт розробляється з використанням двовимірних спрайтів та асетів, що вимагає ретельного налаштування атласів текстур та алгоритмів очищення оперативної пам'яті для стабільного функціонування застосунку на пристроях із обмеженими технічними характеристиками.

1.2 Вибір платформ та мови програмування

У процесі проєктування мобільного застосунку першочерговим інженерним завданням є обґрунтування цільової архітектурної платформи та вибір адекватного

інструментарію розробки. Сучасний ринок програмного забезпечення розподілений між декількома основними екосистемами, включаючи стаціонарні персональні комп'ютери, стаціонарні та портативні ігрові консолі, а також мобільні пристрої, що функціонують під керуванням операційних систем Android та IOS. На відміну від дорогих стаціонарних проєктів, орієнтованих на розповсюдження на одній платформі, створення мобільного квесту вимагає фокусування на специфіці мобільних операційних систем, серед яких найбільш поширеною, відкритою та гнучкою є Android.

Вибір цієї системи як єдиної цільової платформи обумовлений її архітектурними особливостями, низьким порогом для тестування та можливістю прямої дистрибуції кінцевих пакетів інсталяції на цільові пристрої без жорстких обмежень, що властиві закритим платформам. Проте реалізація квесту на Android накладає специфічні вимоги до вибору мови програмування. Програмний стек повинен ефективно взаємодіяти з підсистемами Android, забезпечувати високу швидкість виконання інструкцій, мати надійні механізми керування пам'яттю та підтримувати сучасні норми об'єктно орієнтованого проєктування для оптимізації коду під різноманітні специфікації мобільних процесорів [4].

Для визначення оптимального технологічного стеку було зроблено порівняльний аналіз провідних мов програмування, що використовуються у сучасній індустрії розробки програмного забезпечення та ігрової інженерії (рисунки 1.1 – 1.4).

```
// Use a switch statement to do the math
switch (Console.ReadLine())
{
    case "a":
        Console.WriteLine($"Your result: {num1} + {num2} = " + (num1 + num2));
        break;
    case "s":
        Console.WriteLine($"Your result: {num1} - {num2} = " + (num1 - num2));
        break;
    case "m":
        Console.WriteLine($"Your result: {num1} * {num2} = " + (num1 * num2));
        break;
    case "d":
        // Ask the user to enter a non-zero divisor until they do so
        while (num2 == 0)
        {
            Console.WriteLine("Enter a non-zero divisor: ");
            num2 = Convert.ToInt32(Console.ReadLine());
        }
        Console.WriteLine($"Your result: {num1} / {num2} = " + (num1 / num2));
        break;
}

// Wait for the user to respond before closing
Console.Write("Press any key to close the Calculator console app...");
Console.ReadKey();
```

Рисунок 1.1 – Мова програмування C#

```
whatsapp_messaging.py ▶ Client
1  from twilio.rest import Client
2
3  def msg_mom_and_dad(event=None, context=None):
4
5      # get your sid and auth token from twilio
6      twilio_sid = 'AC84c9f1602d7fb6af4eda5b0c39a03b37'
7      auth_token = '4a2021b28f1aa606d9c6945d3c248ebd'
8
9      whatsapp_client = Client(twilio_sid, auth_token)
10
11     # keep adding contacts to this dict to send
12     # them the message
13     contact_directory = {'daddy': '+919624666836'}
14
15     for key, value in contact_directory.items():
16         msg_loved_ones = whatsapp_client.messages.create(
17             body = 'good morning {} !'.format(key),
18             from_ = 'whatsapp:+14155238886',
19             to='whatsapp:' + value,
20
21         )
22
23     print(msg_loved_ones.sid)
```

Рисунок 1.2 – Скрипт на мові Python

```

Heatmap.cpp x
24 #include "Heatmap.h"
23 #include <TileMap.hpp>
22 #include <SceneTree.hpp>
21 #include <Viewport.hpp>
20 #include <deque>
19 #include <TileSet.hpp>
18 #include <algorithm>
17
16 namespace godot {
15     inline float lerp(const float& a, const float& b, const float& t) {
14         return a + t * (b - a);
13     }
12
11     void Heatmap::_register_methods() {
10         //public
9         register_method("_ready", &Heatmap::_ready);
8         register_method("_draw", &Heatmap::_draw);
7         register_method("_process", &Heatmap::_process);
6         register_method("best_direction_for", &Heatmap::best_direction_for);
5         register_method("calculate_point_index", &Heatmap::calculate_point_index);
4         register_method("calculate_point_index_for_world_position",
3             &Heatmap::calculate_point_index_for_world_position);
2
1         //semi-private
0         register_method("_on_Events_player_moved", &Heatmap::on_Events_player_moved);
25
1         //properties
2         register_property<Heatmap, NodePath>("pathfinding_tilemap", &Heatmap::m_pathfinding_tilemap,
NodePath());
10k godot-platformer-2d/game/src/Native/Heatmap/Heatmap.cpp 25:0 Top7:35AM 0.97 LF UTF-8 C++//l master

```

Рисунок 1.3 – Скрипт в рушії Godot

```

#include <cstdio>

int main()
{
    int a = 0;
    int b = 0;
    std::scanf("%d %d", &a, &b);
    std::printf("%d %d\n", a, b);
}

```

Рисунок 1.4 – Скрипт на мові C++

Одним із найбільш ефективних інструментів для створення програм під ОС Android є мова програмування C# (Рисунок 1.1), яка виступає базовим засобом розробки в середовищі Unity. Завдяки технології компіляції IL2CPP (Intermediate Language to C++), яка інтегрована в Unity, вихідним код мови C# трансформується у високоефективні інструкції C++, що оптимізує швидкодію застосунку безпосередньо під архітектуру процесорів мобільних пристроїв на ОС Android. C# забезпечує сувору типізацію даних, автоматичне керування пам'яттю та має

розвинену екосистему стандартних бібліотек, що дозволяє будувати надійні системи менеджменту ігрових станів для квесту.

Як альтернативу розглядають мову програмування Python (Рисунок 1.2). Вона володіє низьким порогом входження та лаконічним синтаксисом, що зручно для швидкого прототипування або написання серверних скриптів взаємодії. Проте для розробки клієнтської частини ігор для Android Python є малоефективним через інтерпретований характер виконання коду та складність запуску інтерпретатора в архітектурі Android, що призводить до високих витрат ресурсів та зниження частоти кадрів.

Також у межах дослідження було проаналізовано спеціалізовані скриптові мови, такі як GDScript, що інтегрована в ігровий рушій Godot (рисунок 1.3). Ця мова оптимізована під архітектуру конкретного середовища та демонструє непогану продуктивність при розробці легковажних двовимірних проєктів. Однак порівняно з C#, екосистема GDScript має значно меншу кількість готових плагінів та інструментів для інтеграції зі специфічними сервісами ОС Android.

Для порівняння було розширено аналіз на низькорівневі мови, зокрема C++ (рисунок 1.4), яка традиційно вважається стандартом для розробки високонавантажених програм завдяки можливості прямого керування ресурсами пристрою. За допомогою Android NDK (Native Development Kit) розробник може писати код на C++ безпосередньо під цю операційну систему. Проте ручне керування пам'яттю, складний синтаксис та тривалий процес відлагодження значно збільшують час розробки й підвищують ризик виникнення критичних помилок, що є недоцільним при створенні додатку в межах даної роботи [5].

Таким чином, на основі проведеного техніко-економічного та архітектурного аналізу, для реалізації дипломного проєкту під операційну систему Android було обрано мову програмування C# у поєднанні з рушієм Unity Engine. Це компромісне рішення забезпечує найвищу швидкість написання коду та гарантує стабільні показники продуктивності продукту на більшості сучасних смартфонів.

1.3 Вибір двигунів для мобільних ігор

Важливим технологічним етапом проєктування мобільного застосунку є вибір рушія, який визначає архітектуру проєкту, швидкість обробки ігрових подій та ефективність використання апаратних ресурсів цільового пристрою. У сучасній індустрії розробки програмного забезпечення виділяють три провідні інструментальні системи, що застосовуються для створення додатків:

- Unity;
- Defold;
- Cocos Creator.

1.3.1 Рушій Unity Engine

Вибір середовища Unity як базового інструменту розробки для даного проєкту обумовлений низкою його суттєвих архітектурних та функціональних переваг. Першочерговим фактором є інтеграція мови програмування C#, яка, будучи високорівневою мовою із суворою типізацією, надає розробнику великий інструментарій для побудови об'єктно-орієнтованих систем. Наявність великої кількості вбудованих готових класів мінімізує потребу у написанні низкорівневого коду для взаємодії з компонентами мобільного пристрою, що значно прискорює процес проєктування логіки двовимірного квесту.

Іншою важливою характеристикою Unity є його розвинена архітектура, оптимізована під мобільний сегмент, зокрема під систему Android. Програмні модулі та скрипти, створені в межах цього середовища, трансформуються у ефективний нативний код завдяки компіляції IL2CPP, що дозволяє суттєво скоротити терміни розробки продукту.

1.3.2 Рушій Defold

Ігровий рушій Defold є спеціалізованим безкоштовним рішенням із відкритим кодом, яке з самого початку проєктувалося виключно для створення двовимірних мобільних та веб-застосунків. Головна його перевага – екстремальна легковажність та архітектура, орієнтована на максимальну продуктивність. Рушій

написаний на C++, а логіка гри описується за допомогою скриптової мови Lua. Завдяки цьому швидкість обробки спрайтів та споживання RAM демонструють кращі показники в індустрії.

Проте, для реалізації комплексного дипломного проєкту цей рушій має кілька суттєвих недоліків. Скриптова мова використовує динамічну типізацію, що підвищує ризик виникнення прихованих помилок при побудові складних логічних ланцюжків квесту. Крім того, екосистема рушія та його спільнота відносно невеликі. Кількість плагінів та інструментів в магазині значно обмежена, що змушує розробника проєктувати більшість систем з нуля [6].

1.3.3 Рушій Cocos Creator

Cocos Creator – це сучасний рушій з повноцінним графічним інтерфейсом, який займає лідируючі позиції на азіатському ринку мобільних ігор. Його архітектура оптимізована для створення двовимірних ігор, а як основний стек програмування використовуються мови JavaScript та TypeScript. Рушій має потужну систему роботи з компонентною моделлю, вбудовані інструменти для керування анімацією, елементами інтерфейсу та фізикою, що робить його зручним для розробки квестів та головоломок під мобільні операційні системи.

Основним технологічним викликом при виборі цього рушія є специфіка виконання коду на базі його скриптових мов всередині ОС Android. Для роботи застосунку необхідна інтеграція віртуальної машини, що створює додаткове навантаження на процесор смартфона та збільшує час завантаження гри порівняно з Unity (IL2CPP). Також варто врахувати, що більша частина глибокої технічної документації, форумів підтримки та плагінів для цього рушія доступна виключно китайською мовою. Це створює серйозний мовний бар'єр при вирішенні нестандартних завдань або оптимізації рендерингу графіки, роблячи Unity більш надійним та прогнозованим вибором [7].

2 ПРОЄКТУВАННЯ ТА НАЛАШТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У межах розробки мобільного застосунку етап проєктування та налаштування програмного забезпечення є визначальним для забезпечення стабільності, оптимізації та сумісності продукту. Основним інструментальним середовищем для реалізації проєкту було обрано ігровий рушій Unity Engine. Дане програмне забезпечення інтегрує в собі комплекс підсистем, що дозволяють автоматизувати процеси взаємодії з операційною системою Android, графічними прискорювачами та модулями введення даних, звільняючи ресурси розробника для розробки безпосередньо архітектури гри та менеджменту ігрових станів.

Програмна архітектура Unity базується на компонентно-орієнтованому підході та парадигмі об'єктно-орієнтованого проєктування. Це дозволяє гнучко налаштовувати взаємодію між окремими ігровими об'єктами шляхом динамічного підключення модулів коду, написаних мовою C#. Важливою перевагою також є наявність гнучкого ліцензування для розробників-початківців та відкрита архітектура розширення редактора, що дає змогу адаптувати робочий простір середовища під специфічні завдання розробки.

Технічна документація рушія та розвинене прикладне програмне забезпечення (Unity API) надають повний контроль над життєвим циклом застосунку. Процес взаємодії розробника з екосистемою програмного забезпечення побудований на використанні базових архітектурних модулів, які структуровані за функціональним призначенням:

- ядро системи та базові класи (Core API): забезпечує життєвий цикл ігрових скриптів (методи Awake, Start, Update), обробку подій та керування потоками даних, що є основою для реалізації логіки квесту;
- підсистема двовимірної графіки Unity 2D: включає інструменти для роботи зі спрайтами (Sprite Renderer), побудови ігрових поверхонь (Tilemaps) та

оптимізації рендерингу (Sprite Atlas), що дозволяє мінімізувати кількість викликів отрисовки (Draw Calls) на мобільних графічних процесорах;

- менеджер графічного інтерфейсу користувача Unity UI / UI Toolkit: надає компоненти для побудови адаптивних екранних форм, систем меню, вікон діалогів та інвентарю, що автоматично масштабуються під різні роздільні здатності екранів Android-пристроїв;

- модуль компіляції та збірки Build Pipeline: забезпечує інтеграцію середовища з Android SDK та NDK, автоматизуючи процес генерації кінцевих пакетів розгортання.

Особливе місце в розробці займає налаштування взаємодії ігрових скриптів із внутрішньою архітектурою рушія через підсистему сценаріїв. При налаштуванні середовища розробки під ОС Android критично важливим є перехід від стандартного віртуального виконання Mono до передової технології компіляції IL2CPP. Це дозволяє трансформувати проміжний код мови C# у високоефективний нативний код C++ безпосередньо перед фінальною збіркою проєкту. Таке налаштування програмного забезпечення забезпечує суттєве підвищення швидкодії обчислювальних алгоритмів, оптимізує роботу з оперативною пам'яттю мобільного пристрою та дозволяє згенерувати 64-бітні архітектурні версії додатків (ARM64), що є обов'язковою вимогою для сучасного програмного забезпечення під керуванням Android.

2.1 Особливості використання Unity

Ефективна реалізація мобільного ігрового застосунку в середовищі Unity вимагає глибокого аналізу його внутрішніх підсистем та їх адаптації під обмежені апаратні ресурси мобільних пристроїв, що функціонують під керуванням ОС Android. На відміну від десктопних архітектур, проєктування для мобільних платформ накладає жорсткі ліміти на використання оперативної пам'яті та

обчислювальної потужності графічного процесора. Графічна підсистема Unity при розробці двовимірних проєктів орієнтована на роботу зі спрайтовими архітектурами, де ключовим компонентом виступає модуль Sprite Renderer.

Для забезпечення високої частоти кадрів та виключення затримок під час обробки графіки в грі, налаштування компонентів Unity передбачає обов'язкове групування текстур у спеціальні графічні атласи. Це дозволяє підсистемі рендерингу об'єднувати поодинокі об'єкти сцени в один загальний пакет для відправки на GPU. Таким чином, замість десятків окремих операцій обробки, мобільний процесор виконує лише одну, що мінімізує накладні витрати та запобігає перегріву мобільного пристрою. Менеджмент візуальних шарів реалізується через конфігурацію параметрів Sorting Layers та Order in Layer, що гарантує коректне відображення елементів заднього плану, ігрових об'єктів взаємодії та персонажів без залучення ресурсомісткого буфера глибини [8].

Керування станами гри та менеджмент ігрових ресурсів (скриптів, аудіофайлів, конфігурацій квесту) реалізується через оптимізовану роботу з потоками даних. Замість важких форматів аудіо без стиснення, підсистема Audio Source налаштовується на використання стиснутих форматів .ogg або .mp3, які при імпорті в Unity переводяться в режим Compressed In Memory або Streaming. Це запобігає надмірному виділенню оперативної пам'яті під звукові ефекти та музичний супровід, забезпечуючи стабільну роботу застосунку навіть на бюджетних мобільних пристроях з обмеженим об'ємом ОЗП.

Таким чином, системне налаштування підсистем рендерингу, фізичної взаємодії, інтерфейсу та аудіоменеджменту в Unity дозволяє створити оптимізовану архітектуру мобільної гри, що повністю відповідає технічним стандартам та вимогам операційної системи Android.

2.2 Скриптинг у Unity

Програмна реалізація ігрових механік, алгоритмів взаємодії та систем управління станами у середовищі Unity здійснюється за допомогою написання коду мовою C#. Архітектура сценаріїв проєкту базується на створенні модульних, слабо пов'язаних класів, що дозволяє забезпечити високу гнучкість та масштабованість системи мобільного додатку.

Базовим підходом при проєктуванні логіки є створення ієрархії класів, використання принципів наслідування та суворої типізації даних. Приклади програмної реалізації ключових компонентів (рисунок 2.1 – 2.2).

```

1  using UnityEngine;
2  using UnityEngine.SceneManagement;
3
4  public class Door : InteractiveObject
5  {
6      [SerializeField]
7      private float yPositionOffset = 1f;
8      [SerializeField]
9      private bool savePosition = false;
10     [SerializeField]
11     private string sceneName;
12
13     public override void action()
14     {
15         if (savePosition)
16         {
17             PlayerPrefs.SetInt(PlayerConsts.RESTART_POSITION_AVAILABLE, 1);
18             PlayerPrefs.SetFloat(PlayerConsts.X_RESTART_POSITION, gameObject.transform.position.x);
19             PlayerPrefs.SetFloat(PlayerConsts.Y_RESTART_POSITION, gameObject.transform.position.y - yPositionOffset);
20         }
21         SceneManager.LoadScene(sceneName);
22     }
23 }

```

Рисунок 2.1 – Приклад C# Script 1

```

re.cs  Door.cs  TabsController.cs
Assembly-CSharp
1  using UnityEngine;
2
3  public abstract class Fire : MonoBehaviour
4  {
5      public abstract void startAction(PlayerHealth playerHealth);
6      public abstract void stopAction(PlayerHealth playerHealth);
7  }
8
9

```

Рисунок 2.2 – Приклад C# Script 2

У межах розробки систем взаємодії з об'єктами (рисунок 2.1) було реалізовано клас Door, який наслідується від базового класу InteractiveObject. Це дозволяє застосувати принципи поліморфізму для створення універсальної системи тригерів на ігровій сцені. Важливою задачею для додатку є менеджмент ігрових станів та збереження прогресу користувача у внутрішній пам'яті смартфона. Для реалізації цієї логіки у методі action() використовується вбудована підсистема PlayerPrefs. При активації тригера (savePosition) програмний компонент записує у локальне сховище мобільного пристрою координати позиції об'єкта по осях X та Y, а також прапорець доступності кімнати через методи PlayerPrefs.SetInt та PlayerPrefs.SetFloat. Після фіксації стану менеджер сцен SceneManager.LoadScene виконує асинхронне завантаження наступної локації, що мінімізує затримки інтерфейсу на Android-пристроях [9].

Для оптимізації архітектури застосовано абстрагування логіки (рисунок 2.2). Створення абстрактного класу Fire, який наслідується від базового класу рушія MonoBehaviour, дозволяє задати єдиний програмний інтерфейс для групи однотипних ігрових перешкод або інтерактивних зон. Визначення абстрактних методів startAction та stopAction із передачею посилання на компонент стану здоров'я гравця (PlayerHealth) гарантує, що всі похідні класи будуть зобов'язані реалізувати власну логіку обробки шкоди або активації ефектів. Такий підхід забезпечує чистоту коду, виключає дублювання функціоналу та спрощує подальше розширення квестових механік.

Для обробки статичних наборів даних із заздалегідь відомим та незмінним об'ємом елементів використовуються одновимірні масиви. Приклад ініціалізації такого контейнера наведено на рисунку 2.3.

```
C#  
  
// Масив із 4 імен ворогів  
string[] enemies = { "Zombie", "Skeleton", "Orc", "Dragon" };  
  
// Доступ за індексом (нумерація починається з 0)  
string firstEnemy = enemies[0]; // "Zombie"
```

Рисунок 2.3 – Array

Для реалізації більш складних, гнучких підсистем, таких як інвентар персонажа, списки поточних квестових завдань або облік динамічних об'єктів, що з'являються на сцені під час проходження гри, використовуються динамічні списки типу `List<T>`. Приклад маніпулювання елементами динамічного контейнера наведено на рисунку 2.4.

```
// Створюємо порожній список цілих чисел
List<int> scores = new List<int>();

scores.Add(100); // Додали елемент
scores.Add(250); // Ще один
scores.Remove(100); // Видалили елемент "100"
```

Рисунок 2.4 – Список

В юніті скрипти взаємодіють з багатьма інструментами самого рушія. Наприклад, для об'єкта персонажа за допомогою скрипта та інструменту `input system actions` можна зв'язати анімації персонажа відносно дій гравця.

У прикладі нижче я підключив скрипт `player` до інструменту `ISA` (рисунок 2.5).

```
Ссылка 0
public void Move(InputAction.CallbackContext context)
{
    animator.SetBool("isMoving", true);

    if (context.canceled)
    {
        animator.SetBool("isMoving", false);
        animator.SetFloat("lastX", move.x);
        animator.SetFloat("lastY", move.y);
        lastMove = move;
    }

    move = context.ReadValue<Vector2>();
    animator.SetFloat("inputX", move.x);
    animator.SetFloat("inputY", move.y);
}
```

Рисунок 2.5 – Player з ISA

Ось приклад, де це може бути корисним. Припустимо, ми хочемо, щоб смуга здоров'я на екрані реагувала на зміни здоров'я анімацією, але ми хочемо, щоб користувальницький інтерфейс був відділений від гравця.

У скрипті `playerHealth.cs` (рисунок 2.6) ми підключаємо керування слайдерами здоров'я, які знаходяться на слої UI:

```
private Color mainHealthColor;
[SerializeField]
private float hungerUpdateSpeed = 0.05f;
[SerializeField]
private float healthUpdateSpeed = 0.05f;

private const float MAX_PLAYER_TEMP = 100f;
[SerializeField]
private float playerTemp = MAX_PLAYER_TEMP;
[SerializeField]
private float updateTempValue = 0f;

// Cooldown Unity Ссылка: 0
void Start()
{
    hungerFillImage = hungerSlider.fillRect.GetComponent<Image>();
    mainHungerColor = hungerFillImage.color;

    healthFillImage = healthSlider.fillRect.GetComponent<Image>();
    mainHealthColor = healthFillImage.color;
}

// Update is called once per frame
// Cooldown Unity Ссылка: 0
void Update()
{
    updatehunger();
    updatehealth();
    updatetemp();
}

Ссылка: 1
private void updatehunger()
{
    if (hungerSlider.value != 0)
    {
        hungerSlider.value = hungerSlider.value - (hungerUpdateSpeed * Time.deltaTime);
        ChangeHungerColor(hungerSlider, hungerFillImage, mainHungerColor);
    }
}

Ссылка: 1
private void updatehealth()
{
    if (hungerSlider.value == 0)
    {
        healthSlider.value = healthSlider.value - (healthUpdateSpeed * Time.deltaTime);
        ChangeHungerColor(healthSlider, healthFillImage, mainHealthColor);
    }
}
```

Рисунок 2.6 – Скрипт `playerHealth.cs`

Крім того, можна зв'язувати між собою не тільки ігровий об'єкт та UI а і два ігрових об'єкти. Наприклад, нам потрібно щоб гравець міг підібрати ягідку, яка росте на кущі. Для цього нам потрібно три ігрових об'єкти: кущ, ягідка та сам персонаж. Ми робимо три скрипти: перший для того, щоб персонаж, підібравши їжу відновив собі слайдер голоду, другий для куща щоб на ньому росли ягоди, а третій – для самої ягоди щоб вона відновлювала слайдер голоду. При цьому створювати для кожного їстівного об'єкту по скрипту немає сенсу: можна об'єднати всі їстівні об'єкти в групу «їстівні» і тоді кожен такий об'єкт буде набувати ті ж якості що й ягода. Для прикладу: (рисунок 2.7 – 2.9).

```

using UnityEngine;

Ссылка 1: Ссылка на ресурсы | Ссылка 0
public class Food : FoodObject
{
    [SerializeField]
    private float calories = 10;
    [SerializeField]
    private float growSpeed = 0.0003f;
    private Vector3 originScale;
    Ссылка 2
    public override void eat(PlayerHealth playerHealth)
    {
        playerHealth.addCalories(calories);
        Destroy(gameObject);
    }

    Ссылка 1: Ссылка на ресурсы | Ссылка 0
    void Start()
    {
        originScale = transform.localScale;
        transform.localScale = Vector3.zero;
    }

    Ссылка 1: Ссылка на ресурсы | Ссылка 0
    void Update()
    {
        grow();
    }

    Ссылка 1
    private void grow()
    {
        if (originScale.x > transform.localScale.x)
        {
            Vector3 newsScale = transform.localScale;
            newsScale.x += growSpeed * Time.deltaTime;
            newsScale.y += growSpeed * Time.deltaTime;
            calories += growSpeed * Time.deltaTime * 1000;
            transform.localScale = newsScale;
        }
    }
}

```

Рисунок 2.7 – Скрипт росту ягоды

```

1 using UnityEngine;
2
3 public class FruitSpawner : MonoBehaviour
4 {
5     [SerializeField]
6     private GameObject fruitPrefab;
7     private GameObject fruit;
8     [SerializeField]
9     private float fruitPosOffsetX = 0.3f;
10    [SerializeField]
11    private float fruitPosOffsetY = 0.3f;
12    private bool isFruitCreated = false;
13    void Start()
14    {
15    }
16
17    void Update()
18    {
19        if(fruit == null && !isFruitCreated)
20        {
21            Invoke("spawnFruit", 4);
22            isFruitCreated=true;
23        }
24    }
25
26    Ссылка 0
27    private void spawnFruit()
28    {
29        fruit = Instantiate(fruitPrefab);
30        Vector3 pos = transform.position;
31        pos.x += fruitPosOffsetX;
32        pos.y += fruitPosOffsetY;
33        fruit.transform.position = pos;
34        isFruitCreated = false;
35    }
}

```

Рисунок 2.8 – Скрипт куща

```

Ссылка 1
public void addCalories(float calories)
{
    hungerSlider.value = Mathf.Min(100, hungerSlider.value + calories);
}

```

Рисунок 2.9 – Скрипт персонажа з атрибутом calories який має ягода

2.3 Переваги Unity

Для розробки гри було обрано багатофункціональне середовище Unity, що займає провідні позиції у сфері проектування інтерактивного програмного забезпечення. Ключовою перевагою даної платформи є її архітектурна кросплатформність, яка дозволяє розробляти програмний продукт на базі єдиної кодової основи з подальшою компіляцією та оптимізацією під різні операційні системи, зокрема під мобільну платформу Android.

У контексті створення двовимірного квесту для мобільних пристроїв, Unity має низку суттєвих переваг порівняно з іншими спеціалізованими рушіями:

- висока ефективність 2D-підсистеми: На відміну від рушіїв, які адаптують тривимірний простір під псевдо-2D, Unity містить інтегрований інструментарій, орієнтований виключно на роботу з плоскою геометрією. Наявність вбудованих компонентів управління спрайтами, спеціалізованого фізичного рушія Physics 2D та інструментів анімації Animator дозволяє створювати складні механіки взаємодії квестових предметів із мінімальними затратами обчислювальних ресурсів процесора [10];

- автоматизована оптимізація графіки. Середовище розробки підтримує сучасні низькорівневі графічні API, такі як Vulkan та OpenGL ES, що є критично важливим для забезпечення високої продуктивності на ОС Android. Рушій

автоматично реалізує алгоритми динамічного та статичного пакетування геометрії, об'єднуючи однорідні графічні матеріали для передачі на графічний процесор. Окрім цього, вбудований механізм Frustum Culling автоматично виключає з процесу рендерингу ті об'єкти ігрової сцени, які в поточний момент часу знаходяться поза зоною видимості камери, що суттєво знижує навантаження на підсистему пам'яті смартфона;

- гнучкість проектування інтерфейсу користувача. Оскільки мобільний квест орієнтований на взаємодію користувача з графічними елементами (меню, інвентар, вікна діалогів), підсистема Unity UI надає повноцінний набір інструментів для динамічного масштабування контенту під різні типи екранів, співвідношення сторін та щільність пікселів (DPI) без втрати якості візуальних компонентів;

- інтеграція із сучасними середовищами проектування коду. Unity забезпечує нативну інтеграцію із модулем Visual Studio Code, що дозволяє виконувати розширене профілювання програмних скриптів, швидке налагодження безпосередньо на підключеному Android-пристрої, а також підтримує компонентний підхід, де кожен скрипт виступає як окремий модульний блок.

Низький поріг входження у поєднанні з потужними інструментами аналізу продуктивності Unity Profiler дозволяє відстежувати виділення оперативної пам'яті в реальному часі, фіксувати затримки рендерингу та оптимізувати роботу збирача сміття Garbage Collector. Це підтверджує доцільність використання Unity як базового інструментального засобу для успішної реалізації та розгортання проекту мобільної гри.

2.4 Архітектура програмного забезпечення гри

На діаграмі класів (рисунк 2.10) відображено об'єктно-орієнтовану архітектуру розробленого ігрового додатку та зв'язки між його компонентами.

Центральним елементом є клас Player, який взаємодіє з навколишнім середовищем на основі принципу інверсії залежностей та поліморфізму.

Замість прямої залежності від конкретних реалізацій, клас Player через тригери викликає методи абстрактних класів: InteractiveObject, FoodObject, Fire.

Конкретні класи такі як Door, Food, CampFire успадковують базову логіку та реалізують специфічну поведінку об'єктів.

Зв'язок із підсистемою здоров'я персонажа проходить через клас PlayerHealth, який оновлює показники здоров'я, голоду та температури в залежності від отриманих даних від Food та CampFire.

Для збереження стану гри та позиції рестарту класи Player та Door взаємодіють із класом констант.

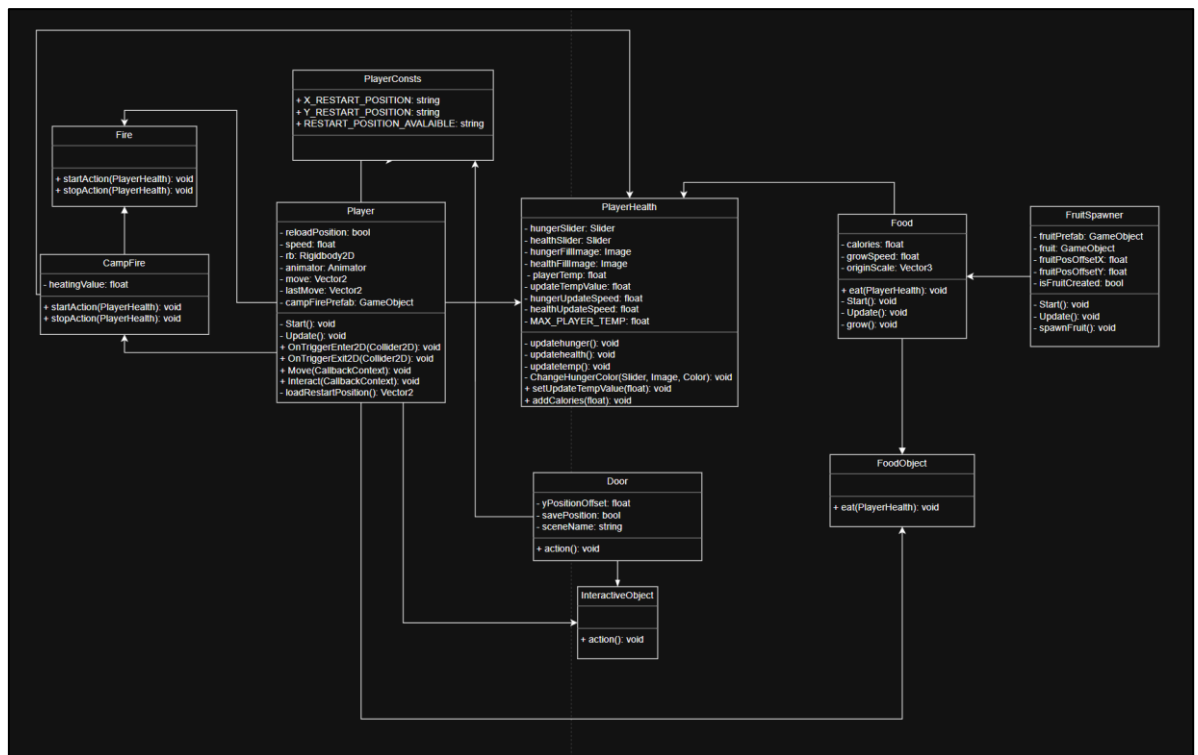


Рисунок 2.10 – Діаграма класів

3 РЕАЛІЗАЦІЯ ГРИ

3.1 Керування асетами та структура файлової системи додатка

Процес програмної реалізації мобільного 2D-квесту передбачає чітку структурування та інтеграцію ігрових ресурсів, які в інженерній практиці класифікуються як ігрові асети. Ігровий асет — це будь-який дискретний інформаційний елемент, компіляція якого у середовищі ігрового рушія дозволяє сформувати цілісне візуальне та функціональне середовище застосунку. До базових категорій активів, що інтегровані в межах розробки даного проєкту, належать двовимірні графіка, елементи інтерфейсу користувача, аудіофайли звукових ефектів та музичного супроводу, а також файли конфігурацій та керуючих сценаріїв.

Основне призначення використання ігрових асетів полягає в оптимізації часових витрат на розробку та зниженні накладних обчислювальних витрат при рендерингу контенту. Класифікація задіяних у проєкті цифрових активів включає:

- аудіоактиви: файли звукового супроводу, фонові музичні композиції та специфічні звукові ефекти взаємодії користувача з квестовими тригерами;
- графічні активи: статичні та анімовані зображення, що формують зовнішній вигляд персонажів, елементи навколишнього середовища, предмети інвентарю та деталі локацій;
- сюжетно-текстові ресурси: файли локалізації, сценарії діалогових систем, текстові масиви реплік та підказок для гравця;
- службові та системні активи: програмні модулі мовою C#, скомпіловані класи, шаблони об'єктів та структури метаданих.

Серед вбудованих системних активів Unity, які безпосередньо впливають на логіку функціонування застосунку, ключове значення мають анімаційні кліпи, параметри аудіомікшерів для керування амплітудно-частотними характеристиками звуку, конфігурації графічних інтерфейсів та префаби. Префаб є оптимізованим

шаблоном ігрового об'єкта, який містить набір налаштованих компонентів і може динамічно дублюватися на сцені під час виконання програми, мінімізуючи навантаження на оперативну пам'ять мобільного пристрою [11].

На початковому етапі розробки візуального стилю 2D-квесту було проведено аналіз відкритих репозиторіїв графічного контенту. Для формування базових елементів оточення використано ресурс stock.adobe.com, що є спеціалізованою базою даних цифрових активів із вільним типом ліцензування. Інтерфейс даної платформи представлено на рисунку 3.1.

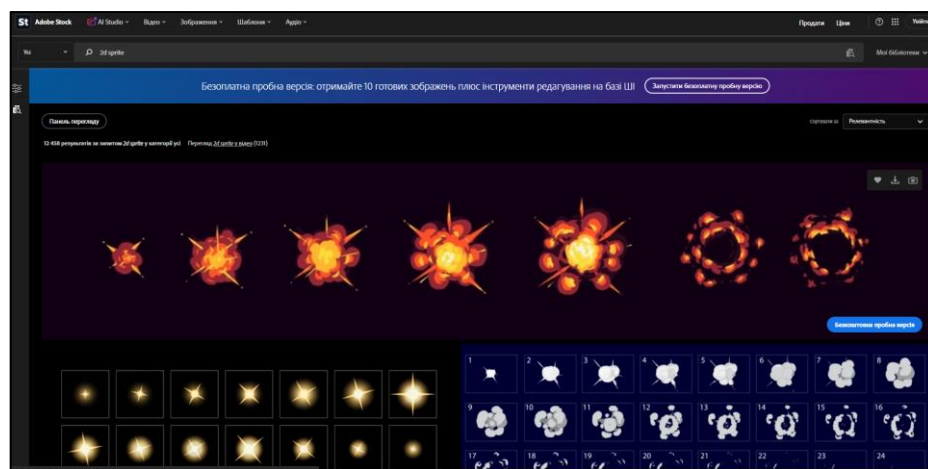


Рисунок 3.1 – Перший сайт

З метою деталізації персонажів та розширення набору об'єктів було додатково задіяно профільний вебресурс opengameart.org, який спеціалізується на постачанні оптимізованих двовимірних текстур та спрайт-листів для мобільних застосунків (рисунок 3.2).

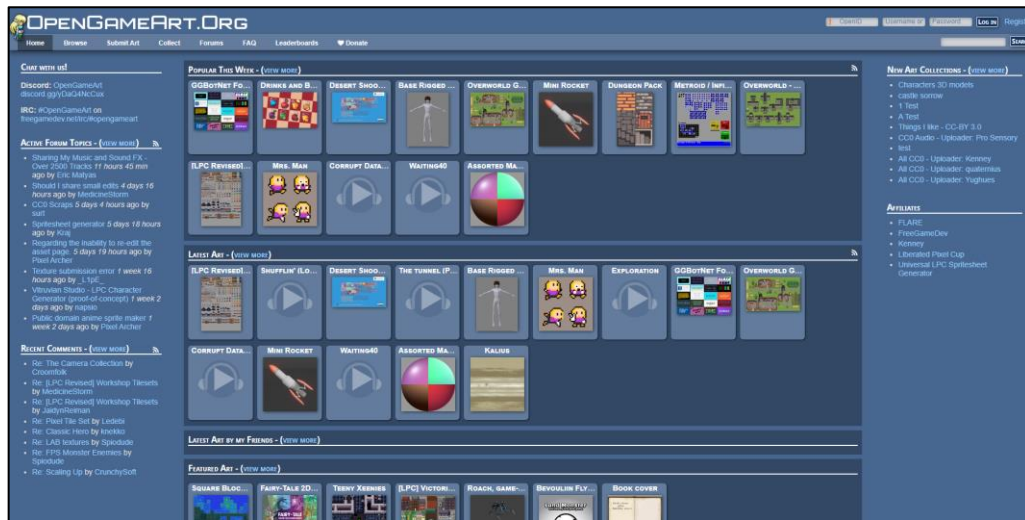


Рисунок 3.2 – Другий сайт

Для цього додатку було прийнято рішення використовувати виключно піксельну двовимірну графіку.

Спрайти у межах архітектури Unity виступають як плоскі графічні об'єкти, що використовують растрові текстури як базове джерело візуальних даних. Для того щоб використовувати їх у розробці додатку потрібно спочатку їх завантажити, потім розпакувати та загрузити у файлову систему Unity (рисунок 3.3 – 3.5).



Рисунок 3.3 – Скачування спрайтів

Имя	Тип	Сжатый размер	Защита па...
License free.txt	Текстовый документ	1 КБ	Нет
License premium.txt	Текстовый документ	1 КБ	Нет
O5GLN30.jpg	Файл "JPG"	633 КБ	Нет
O5GLN31.eps	Файл "EPS"	1 490 КБ	Нет

Рисунок 3.4 – Розпаковка спрайтів

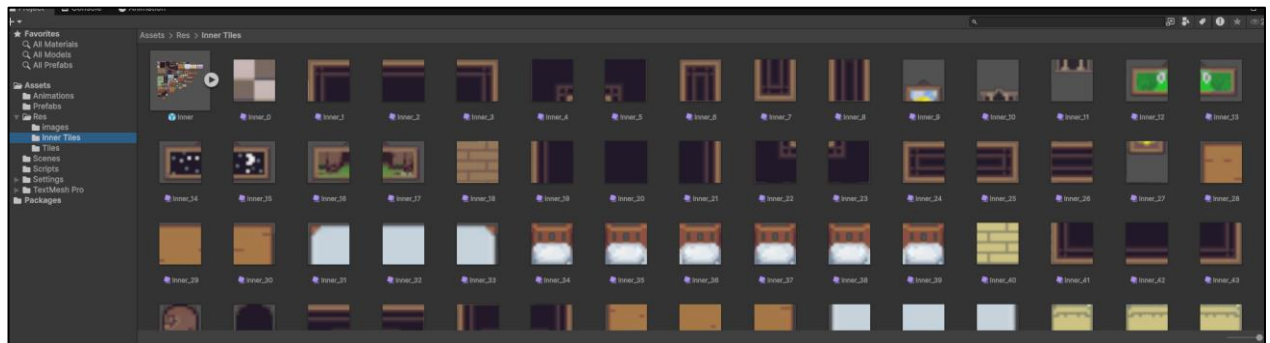


Рисунок 3.5 – Завантаження спрайтів в ФС

Вся файлова структура проєкту зберігається на локальному диску у вигляді ієрархічного дерева каталогів. Кожен асет при імпорті в Unity супроводжується автоматично згенерованим файлом метаданих із розширенням `.meta`. Цей файл містить унікальний глобальний ідентифікатор, який пов'язує актив із внутрішніми підсистемами рушія та іншими компонентами сцени. Середовище Unity підтримує інтеграцію із системами контролю версій, такими як Git або SVN, що забезпечує надійний контроль цілісності коду та ресурсів [12].

Важливою особливістю управління архітектурою файлів є необхідність проведення всіх операцій переміщення, видалення або перейменування активів виключно через внутрішню панель Project редактора Unity. Порухання цього правила призводить до десинхронізації метафайлів `.meta`, втрати зв'язків між об'єктами сцени та виникнення критичних помилок компіляції проєкту. Приклад коректної організації та візуального відображення асетів у файловій системі редактора Unity наведено на рисунку 3.6.

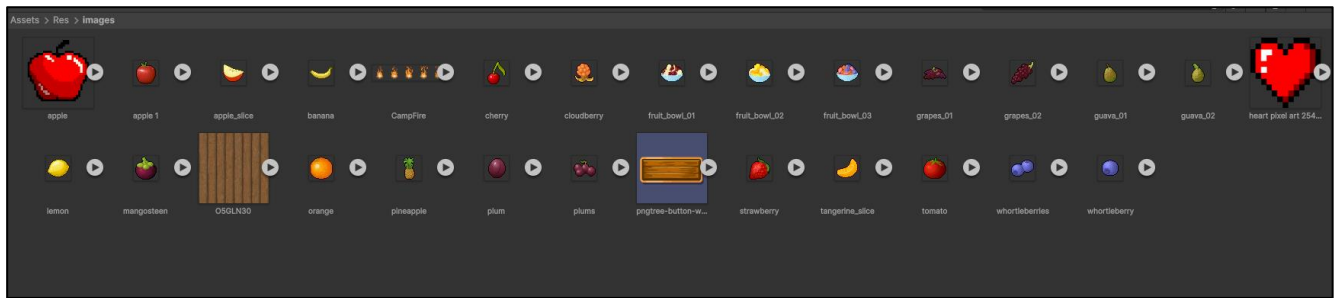


Рисунок 3.6 – Вигляд спрайтів в ФС

3.2 Створення Game Object

В основі розробки будь-якого візуального простору в середовищі Unity покладено компонентно-орієнтований підхід. Базовим будівельним блоком двовимірної гри виступає клас `GameObject`. Сам по собі екземпляр `GameObject` є первинною сутністю, яка за замовчуванням не має візуального відображення, фізичних властивостей чи логіки поведінки, а містить лише обов'язковий компонент трансформації координат `Transform`. Перетворення цієї сутності на функціональний ігровий елемент (персонажа, предмет чи об'єкт інтерфейсу) здійснюється шляхом динамічного або статичного прикріплення специфічних компонентів.

Ієрархічна структура Unity дозволяє створювати складні об'єкти за принципом відносин «батько-нащадок». Дочірні об'єкти успадковують просторові координати, кути повороту та масштабування від батьківського вузла, що дозволяє організовувати складні системи, наприклад, багаторівневі графічні інтерфейси користувача (UI). Концептуальну схему деревоподібної ієрархічної залежності об'єктів у просторі сцени представлено на рисунку 3.7.

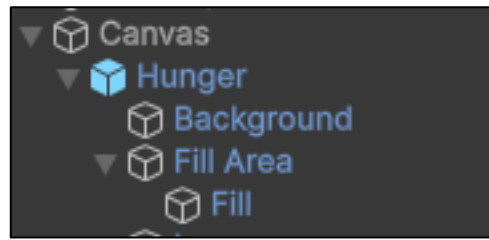


Рисунок 3.7 – Дерево ігрових об'єктів

Здатність групувати об'єкти та вибудовувати чітку ієрархію є потужним інструментом для декомпозиції інтерфейсу гри, систем меню та підменю, а також логічного розділення статичних елементів карти та динамічних тригерів [13].

Наступним логічним рівнем архітектури застосунку є поняття Сцени. Сцена є логічно завершеною підсистемою або рівнем гри, що об'єднує в собі групу ігрових об'єктів, організованих у вигляді ієрархічного дерева. Будь-який проєкт у Unity містить щонайменше одну головну сцену, яка призначається стартовою точкою запуску програмного продукту. Процес ініціалізації нової сцени в редакторі Unity через контекстне меню представлено на рисунку 3.8.

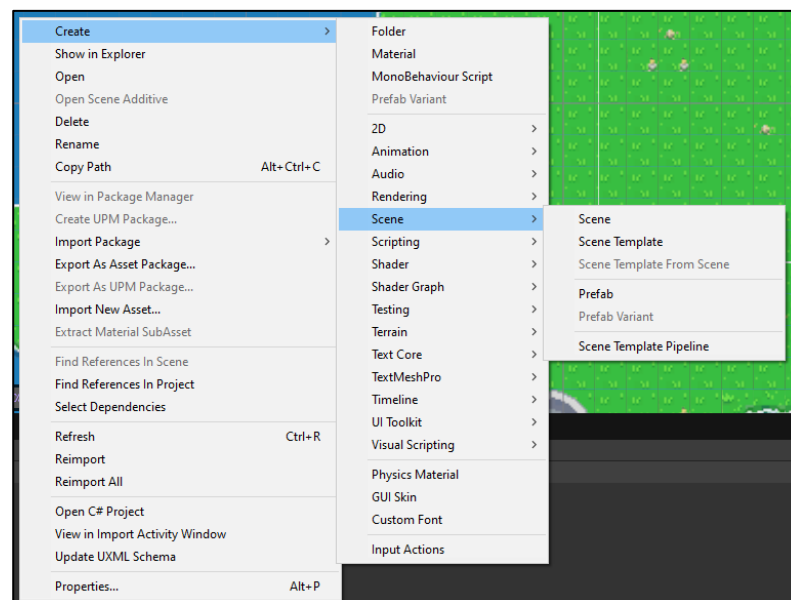


Рисунок 3.8 – Початок роботи

Надалі я створю потрібні мені сцени для моєї казуальної гри. Мені потрібні були кілька сцен: House 1 і основна (рисунок 3.9).

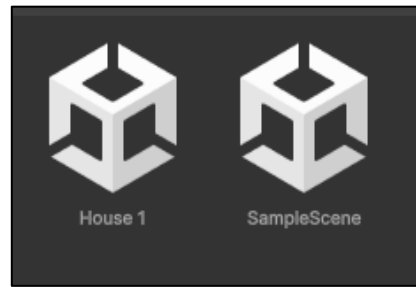


Рисунок 3.9 – Вікно з усіма сценами

Для організації взаємодії користувача з ігровим світом та реалізації механік квесту критично важливим є налаштування підсистеми двовимірної фізики (Physics 2D). Фізична поведінка об'єктів забезпечується інтеграцією двох базових компонентів:

- Rigidbody 2D: компонент, який ставить об'єкт під контроль фізичного рушія Unity. Він визначає такі параметри сутності, як маса, коефіцієнт тертя, вектори лінійної та кутової швидкості, а також ступінь впливу гравітаційного поля;
- Box Collider 2D / Polygon Collider 2D: компоненти, що описують геометричні межі об'єкта для прорахунку фізичних зіткнень у двовимірному просторі. Вони запобігають взаємному проникненню об'єктів та обмежують переміщення персонажа крізь тверді перешкоди (стіни, елементи декору).

Процес конфігурування параметрів компонента Box Collider 2D у панелі інспектора редактора відображено на рисунку 3.10.

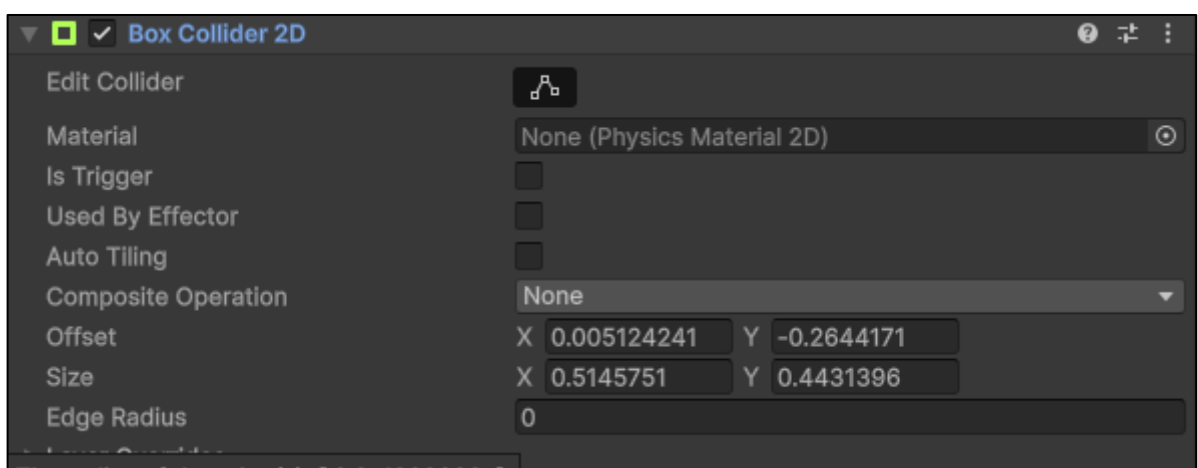


Рисунок 3.10 – Налаштування колайдери персонажа

Для точного збігу фізичних меж із візуальними контурами піксельного персонажа використовується ручне редагування сітки колайдера, що дозволяє оптимізувати обчислення сил протидії при контактах (рисунок 3.11).

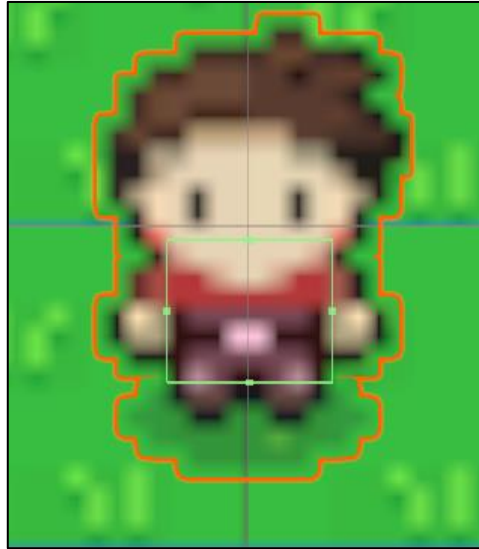


Рисунок 3.11 – Сам колайдер

Окремим функціональним режимом роботи компонентів зіткнення є активація прапорця `Is Trigger`. Переведення колайдера в режим тригера виключає його з фізичного прорахунку твердих тіл (ігрові об'єкти можуть безперешкодно проходити крізь нього), але активує генерацію програмних подій у коді. При входженні об'єкта з компонентом `Rigidbody 2D` у зону тригера, рушій викликає вбудовані методи обробки подій, зокрема `OnTriggerEnter2D()` та `OnTriggerExit2D()`.

Цей механізм є базовим для реалізації квестової логіки: він дозволяє фіксувати наближення гравця до дверей, предметів інвентарю або NPC, виводити діалогові вікна підказок та обмежувати вихід персонажа за межі ігрової зони.

Візуальне представлення розміщення зон колайдерів твердих тіл та зон функціональних тригерів на ігровій карті наведено на рисунку 3.12.



Рисунок 3.12 – Колізії та тригери

Для програмування специфічних часових затримок та циклічних подій (наприклад, таймерів перезапуску дій або затримок анімації) в Unity використовуються Співпрограми, які керуються методами `StartCoroutine()` та інструкцією очікування `WaitForSeconds`. Це дозволяє виконувати асинхронні операції в однопотоковому режимі без блокування основного графічного конвеєра гри [14].

Керування просторовим позиціонуванням у 2D-сценаріях повністю покладається на векторну структуру `Vector2` (координати X та Y), яка зчитується та модифікується через компонент `Transform` кожного окремого об'єкта `GameObject`. Таким чином забезпечується цілісність компонентної архітектури Unity, де кожен елемент виконує строго визначену функцію в межах загальної логіки проекту.

Для забезпечення модульності та гнучкого керування ігровим процесом усі поточні об'єкти структуровані у вікні ієрархії сцени. Це дозволяє розділити графічні компоненти карти, сутність головного персонажа, інтерактивні квестові тригери та елементи інтерфейсу користувача на автономні логічні групи. Загальний вигляд структури компонентів головної ігрової сцени в редакторі Unity представлено на рисунку 3.13.

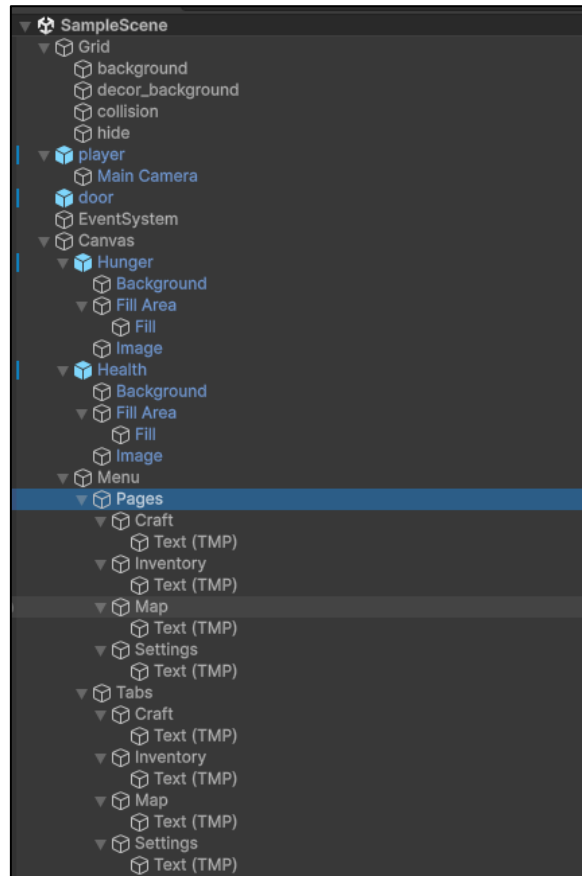


Рисунок 3.13 – Основна сцена з доп всіма GO

Аналіз представленої архітектури сцени (рисунок 3.13) дозволяє виділити кілька системних рівнів взаємодії:

- Grid: кореневий об'єкт підсистеми двовимірного оточення, що містить компоненти Tilemap та Tilemap Renderer. Усі графічні шари заднього плану (background), декорацій (decor_background) та фізичних обмежень карти (collision) інтегровані як дочірні елементи даного вузла, що гарантує суворий збіг просторових координат піксельної сітки;

- UI Canvas: спеціалізований компонент, який виступає базовим контейнером для розміщення всіх елементів графічного інтерфейсу. У межах розробки квесту на Canvas покладено функцію динамічного відображення життєво важливих параметрів персонажа (шкали Hunger та Health), а також навігаційного меню;
- компоненти Text (TextMeshPro): для виведення текстової інформації, квестових підказок та назв елементів інтерфейсу замість застарілого стандартного компонента UI Text задіяно сучасне інтегроване рішення TextMeshPro (TMP). Воно використовує розширені алгоритми генерації контурів шрифтів, що забезпечує ідеальну чіткість та згладжування символів при масштабуванні інтерфейсу під будь-яку роздільну здатність екрана Android-пристрою;
- модуль навігації: ієрархічна група об'єктів, що трансформує стандартні компоненти UI у функціональні інтерактивні блоки керування. Вона об'єднує елементи інтерфейсу в логічні вкладки: створення предметів (Craft), журнал та сховище (Inventory), інтерактивна мапа місцевості (Map) та системні параметри застосунку (Settings). Візуальну реалізацію кнопок керування головного інтерфейсу користувача представлено на рисунку 3.14.

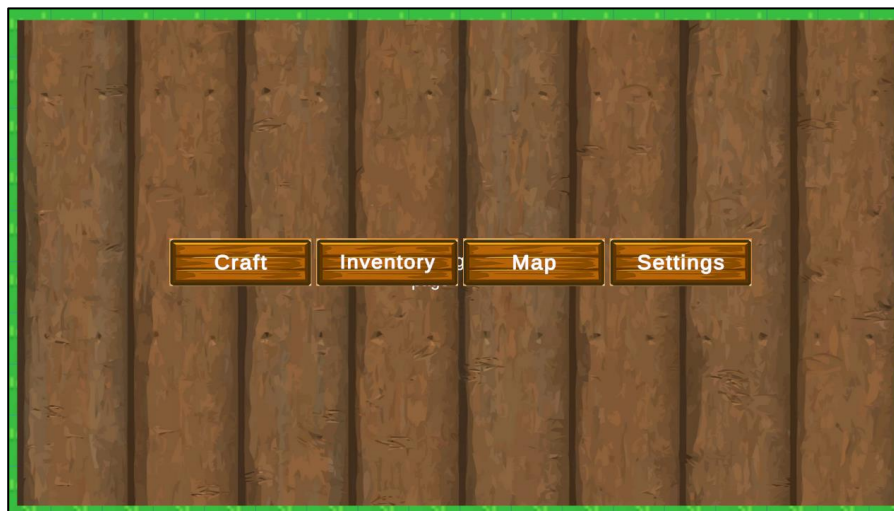


Рисунок 3.14 – Кнопки меню

Важливою задачею при проектуванні двовимірного візуального простору мобільного квесту є менеджмент порядку відрисовки спрайтів графічним процесором. Оскільки 2D-простір не має реальної глибини по осі Z, для

виключення артефактів перекриття об'єктів у Unity застосовується система шарів сортування Sorting Layers та індексів пріоритету Order in Layer.

Рушій дозволяє розробнику створювати необмежену кількість іменованих графічних шарів, де кожен шар обробляється відеоадаптером послідовно — від заднього плану до переднього. Всередині одного шару порядок рендерингу визначається числовим значенням індексу Order in Layer. Об'єкти з меншим індексом відрисовуються першими, формуючи фон, тоді як об'єкти з вищим значенням індексу накладаються поверх них, що є критично важливим для коректного відображення елементів HUD та рухомих персонажів [15].

Різні графічні шари та рівні сортування використовуються для розмежування планів сцени. Наприклад, один рівень виділяється під нижній статичний фон локації, другий — під інтерактивні елементи та меблі, третій — під рухомий спрайт персонажа, а четвертий шар налаштовується як перекриваючий, за якими персонаж може тимчасово ховатися від погляду користувача. Наочну демонстрацію розподілу елементів сцени за рівнями сортування та індексами шарів наведено на рисунку 3.15.

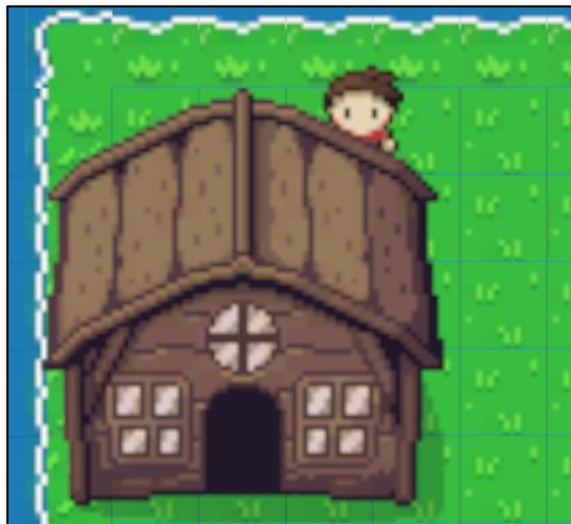


Рисунок 3.15 – Різниця в слоях

Але також можна розмістити декоративні об'єкти за персонажем (рисунок 3.16).

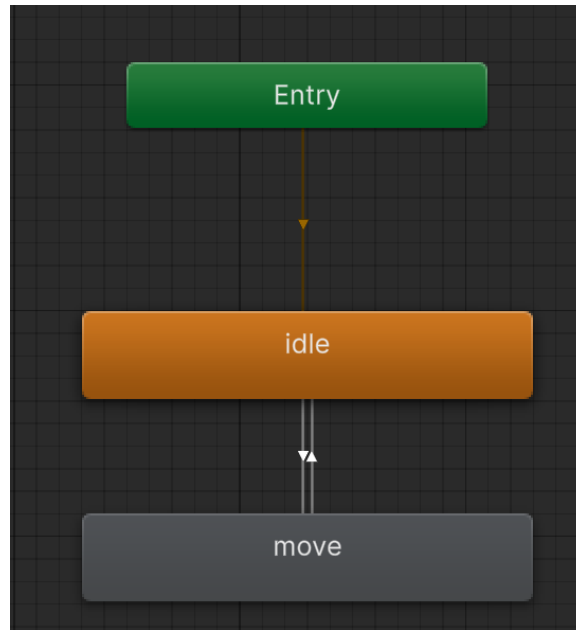


Рисунок 3.19 – Перехід станів анімації в залежності від активності гравця

3.3 Написання скрипта

Написання скрипту для CampFire. (рисунок 3.20). Він відповідає за обігрів персонажа, коли він не стоїть поруч – йому прохолоднішає.

```

1  using UnityEngine;
2
3  public class CampFire : Fire
4  {
5      [SerializeField]
6      private float heatingValue = 2f;
7
8      public override void startAction(PlayerHealth playerHealth)
9      {
10         playerHealth.setUpdateTempValue(heatingValue);
11     }
12
13     public override void stopAction(PlayerHealth playerHealth)
14     {
15         playerHealth.setUpdateTempValue(-1f);
16     }
17 }
18

```

Рисунок 3.20 – CampFire

Написання скрипту для Door (рисунок 3.21). В цьому скрипті реалізовано прохід персонажа у двері, які телепортують його між сценами, а саме те, що він з’являється не в стартовому місці після виходу з дому, а біля дверей.

```

1  using UnityEngine;
2  using UnityEngine.SceneManagement;
3
4  public class Door : InteractiveObject
5  {
6      [SerializeField]
7      private float yPositionOffset = 1f;
8      [SerializeField]
9      private bool savePosition = false;
10     [SerializeField]
11     private string sceneName;
12     public override void action()
13     {
14         if (savePosition)
15         {
16             PlayerPrefs.SetInt(PlayerConsts.RESTART_POSITION_AVAILABLE, 1);
17             PlayerPrefs.SetFloat(PlayerConsts.X_RESTART_POSITION, gameObject.transform.position.x);
18             PlayerPrefs.SetFloat(PlayerConsts.Y_RESTART_POSITION, gameObject.transform.position.y - yPositionOffset);
19         }
20         SceneManager.LoadScene(sceneName);
21     }
22 }
23

```

Рисунок 3.21 – Door скрипт

Написання скрипту для Food (рисунок 3.22). В ньому реалізовано ріст фрукту та те, що персонаж буде його їсти при взаємодії.

Написання скрипту для Player (рисунок 3.23 – 3.24). Тут реалізовано найбільше функцій: взаємодія персонажа з їжею, адже коли він їсть задіюється слайдер голоду та змінює своє значення, взаємодія з полум’ям (змінюється температура), взаємодія з дверима.

```

3  public class Food : FoodObject
4  {
5      [SerializeField]
6      private float calories = 10;
7      [SerializeField]
8      private float growSpeed = 0.0003f;
9      private Vector3 originScale;
10     public override void eat(PlayerHealth playerHealth)
11     {
12         playerHealth.addCalories(calories);
13         Destroy(gameObject);
14     }
15
16     @ Сообщение Unity Ссылка 0
17     void Start()
18     {
19         originScale = transform.localScale;
20         transform.localScale = Vector3.zero;
21     }
22
23     @ Сообщение Unity Ссылка 0
24     void Update()
25     {
26         grow();
27     }
28
29     @ Сообщение 1
30     private void grow()
31     {
32         if (originScale.x > transform.localScale.x)
33         {
34             Vector3 newscale = transform.localScale;
35             newscale.x += growSpeed * Time.deltaTime;
36             newscale.y += growSpeed * Time.deltaTime;
37             calories += growSpeed * Time.deltaTime * 1000;
38             transform.localScale = newscale;
39         }
40     }
41 }

```

Рисунок 3.22 – Food

```

1  using UnityEngine;
2  using UnityEngine.InputSystem;
3  using UnityEngine.SceneManagement;
4
5  @ Скрипт Unity (1 ссылка на ресурсы) Ссылка 0
6  public class Player : MonoBehaviour
7  {
8
9      [SerializeField]
10     private bool reloadPosition = false;
11     [SerializeField]
12     private float speed = 3f;
13     private Rigidbody2D rb;
14     [SerializeField]
15     private Animator animator;
16     private Vector2 move;
17     private Vector2 lastMove;
18
19
20
21
22
23
24     [SerializeField]
25     private GameObject campFirePrefab;
26
27     @ Сообщение Unity Ссылка 0
28     void Start()
29     {
30         rb = GetComponent<Rigidbody2D>();
31         int isPositionAvailable = PlayerPrefs.GetInt(PlayerConsts.RESTART_POSITION_AVAILABLE);
32         if (reloadPosition && isPositionAvailable == 1)
33         {
34             gameObject.transform.position = loadRestartPosition();
35         }
36     }
37
38     // Update is called once per frame
39     @ Сообщение Unity Ссылка 0
40     void Update()
41     {
42         rb.linearVelocity = move * speed;
43     }
44
45     @ Сообщение Unity Ссылка 0
46     public void OnTriggerEnter2D(Collider2D collision)
47     {
48         if (collision.gameObject.tag.Equals("interactive"))
49         {

```

Рисунок 3.23 – Player часть 1

```

        collision.gameObject.GetComponent<FoodObject>().eat(GetComponent<PlayerHealth>());
    }
    else if (collision.gameObject.tag.Equals("fire"))
    {
        collision.gameObject.GetComponent<Fire>().startAction(GetComponent<PlayerHealth>());
    }
}

// Collision 0
public void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.gameObject.tag.Equals("fire"))
    {
        collision.gameObject.GetComponent<Fire>().startAction(GetComponent<PlayerHealth>());
    }
}

// Collision 0
public void Move(InputAction.CallbackContext context)
{
    animator.SetBool("isMoving", true);

    if (context.canceled)
    {
        animator.SetBool("isMoving", false);
        animator.SetFloat("lastX", move.x);
        animator.SetFloat("lastY", move.y);
        lastMove = move;
    }

    move = context.ReadValue<Vector2>();
    animator.SetFloat("inputX", move.x);
    animator.SetFloat("inputY", move.y);
}

// Collision 0
public void Interact(InputAction.CallbackContext context)
{
    if (context.performed)
    {
        GameObject campFire = Instantiate(campFirePrefab);
        Vector2 campFirePos = transform.position;
        campFirePos.x += lastMove.x;
        campFirePos.y += lastMove.y;
        campFire.transform.position = campFirePos;
    }
}

```

Рисунок 3.24 – Player частина 2

Написання скрипту для PlayerHealth (рисунок 3.25). Він відповідає за взаємодію голоду і здоров'я, слайдери цих показників на інтерфейсі користувача, їх зміну.

```

35 void Update()
36 {
37     updateHunger();
38     updateHealth();
39     updateTemp();
40 }
41
42 // Collision 1
43 private void updateHunger()
44 {
45     if (hungerSlider.value != 0)
46     {
47         hungerSlider.value = hungerSlider.value - (hungerUpdateSpeed * Time.deltaTime);
48         ChangeHungerColor(hungerSlider, hungerFillImage, mainHungerColor);
49     }
50 }
51
52 // Collision 1
53 private void updateHealth()
54 {
55     if (hungerSlider.value == 0)
56     {
57         healthSlider.value = healthSlider.value - (healthUpdateSpeed * Time.deltaTime);
58         ChangeHungerColor(healthSlider, healthFillImage, mainHealthColor);
59     }
60 }
61
62 // Collision 1
63 private void updateTemp()
64 {
65     if (updateTempValue > 0 && playerTemp < MAX_PLAYER_TEMP || updateTempValue < 0 && playerTemp > 0)
66     {
67         playerTemp += updateTempValue * Time.deltaTime;
68         playerTemp = Mathf.Clamp(playerTemp, 0, MAX_PLAYER_TEMP);
69     }
70 }
71
72 // Collision 2
73 public void setUpdateTempValue (float value)
74 {
75     updateTempValue = value;
76 }
77
78 // Collision 2
79 private void ChangeHungerColor(Slider slider, Image image, Color mainColor)
80 {
81     if (slider.value < 15 && image.color != Color.red)
82     {
83         image.color = Color.red;
84     } else if (slider.value > 15 && image.color != mainColor)
85     {
86         image.color = mainColor;
87     }
88 }

```

Рисунок 3.25 – PlayerHealth

Написання скрипту для Fire (рисунок 3.26). Він відповідає за створення абстрактного класу Fire, який взаємодіє з здоров'ям гравця. Його можна використовувати як тег для GO для передавання всіх параметрів цього класу.

```

1  using UnityEngine;
2
3  Скрипт Unity | Ссылка: 3
4  public abstract class Fire : MonoBehaviour
5  {
6      Ссылка: 2
7      public abstract void startAction(PlayerHealth playerHealth);
8      Ссылка: 2
9      public abstract void stopAction(PlayerHealth playerHealth);
10 }

```

Рисунок 3.26 – Fire

Написання скрипту для Food Object (рисунок 3.27). Він як і Fire створює абстрактний клас Food Object при взаємодії з яким працює функція eat і змінюються параметри голоду.

```

1  using UnityEngine;
2
3  Скрипт Unity | Ссылка: 2
4  public abstract class FoodObject : MonoBehaviour
5  {
6      Ссылка: 2
7      public abstract void eat(PlayerHealth playerHealth);
8  }
9

```

Рисунок 3.27 – Food Object

Написання скрипту для Fruit Spawner (рисунок 3.28). Цей скрипт відповідає за те, щоб префаб ягоди ріс на куці. Перевіряє, чи створена вже ягода, якщо ні або її з'їли запускає ріст нової.

```

© Скрипт Unity (1 ссылка на ресурсы) | Ссылка: 0
public class FruitSpawner : MonoBehaviour
{
    [SerializeField]
    private GameObject fruitPrefab;
    private GameObject fruit;
    [SerializeField]
    private float fruitPosOffsetX = 0.3f;
    [SerializeField]
    private float fruitPosOffsetY = 0.3f;
    private bool isFruitCreated = false;
    © Сообщение Unity | Ссылка: 0
    void Start()
    {
    }
    © Сообщение Unity | Ссылка: 0
    void Update()
    {
        if(fruit == null && !isFruitCreated)
        {
            Invoke("spawnFruit", 4);
            isFruitCreated=true;
        }
    }
    Ссылка: 0
    private void spawnFruit()
    {
        fruit = Instantiate(fruitPrefab);
        Vector3 pos = transform.position;
        pos.x += fruitPosOffsetX;
        pos.y += fruitPosOffsetY;
        fruit.transform.position = pos;
        isFruitCreated = false;
    }
}

```

Рисунок 3.28 – Fruit Spawner

Написання скрипту для Interactive object (рисунок 3.29). Цей скрипт визначає абстрактний клас інтерактивного об'єкту, та те що до об'єктів з цим тегом треба виконувати функцію action.

```

using UnityEngine;

© Скрипт Unity | Ссылка: 2
public abstract class InteractiveObject : MonoBehaviour
{
    Ссылка: 2
    public abstract void action();
}

```

Рисунок 3.29 – Interactive object

Написання скрипту для Menu controller(рисунок 3.30). Відповідає за меню, робить так щоб за замовчуванням воно було закрите та відкривалося при натисканні кнопки

```

using UnityEngine;
using UnityEngine.InputSystem;

Скрипт Unity (1 ссылка на ресурсы) | Ссылок: 0
public class MenuController : MonoBehaviour
{
    Сообщение Unity | Ссылок: 0
    void Start()
    {
        gameObject.SetActive(false);
    }
    Ссылок: 0
    public void OnMenuClick(InputAction.CallbackContext context)
    {
        if (context.performed)
        {
            gameObject.SetActive(!gameObject.activeSelf);
        }
    }
}

```

Рисунок 3.30 – Menu controller

Написання скрипту для Player consts (рисунок 3.31). Цей скрипт містить у собі константи місцезнаходження гравця на мапі, вони використовуються для переміщення гравця між сценами.

```

1 using UnityEngine;
2
3 public class PlayerConsts
4 {
5     public const string X_RESTART_POSITION = "x_restart_position";
6     public const string Y_RESTART_POSITION = "y_restart_position";
7     public const string RESTART_POSITION_AVALAIBLE = "restart_position_avaible";
8 }
9

```

Рисунок 3.31 – Player Consts

Написання скрипту для Tabs controller (рисунок 3.32). Тут реалізована зміна кольору кнопок меню при натисканні. Якщо кнопки не чіпати вони нормального кольору, якщо натиснути на будь-яку то вона змінить колір на більш темний.

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

[Script]
public class TabsController : MonoBehaviour
{
    [SerializeField]
    public GameObject [] tabs;
    private int LastTabIndex = 0;
    [SerializeField]
    public GameObject[] pages;
    private int LastPageIndex = 0;

    void Start()
    {
        for (int i = 0; i < tabs.Length; i++)
        {
            if (i > 0)
            {
                tabs[i].GetComponent<Image>().color = Color.gray;
                pages[i].SetActive(false);
            }
        }
    }

    public void NextPage(int index)
    {
        if (index != LastTabIndex)
        {
            tabs[index].GetComponent<Image>().color = Color.white;
            tabs[LastTabIndex].GetComponent<Image>().color = Color.gray;
            LastTabIndex = index;

            pages[index].SetActive(true);
            pages[LastTabIndex].SetActive(false);
            LastPageIndex = index;
        }
    }
}
```

Рисунок 3.32 – Tabs controller

3.4 Апаратне забезпечення системи

Важливим етапом проєктування мобільного програмного забезпечення є комплексний аналіз цільової архітектури пристроїв та формування специфікацій системних вимог. Оскільки розроблюваний ігровий застосунок базується на кросплатформному рушії Unity та орієнтований на розгортання в екосистемі Android, конфігурація апаратних ресурсів безпосередньо впливає на стабільність обробки логічних скриптів, швидкість рендерингу графічних кадрів та ефективність менеджменту оперативної пам'яті.

Враховуючи специфіку двовимірної графіки Pixel Art та оптимізовану компонентну структуру сцени, гра здатна функціонувати на широкому спектрі мобільних пристроїв — від бюджетного сегмента до флагманських рішень. Для забезпечення стабільного показника частоти кадрів без критичного перегріву процесора було проведено профілювання та встановлено граничні апаратні та системні параметри. Складові мінімальної та рекомендованої конфігурацій цільового обладнання систематизовано й наведено у таблиці 3.1.

Таблиця 3.1 – Мінімальні вимоги

№ п/п	Компонент	Мінімальні вимоги	Рекомендовані вимоги
1	Версія ОС	Android 5.1	Android 10
2	Процесор	ARMv7 або ARM64	Сучасний 8 ядерний процесор (ARM 64)
3	Графіка	Підтримка OpenGL ES 2.0 3.0	Підтримка Vulkan або OpenGL ES 3.1+
4	Оперативна пам'ять	2 ГБ	4ГБ+
5	Вільне місце	200МБ	1ГБ

4 ТЕСТУВАННЯ РОБОТИ СИСТЕМИ

4.1 Тестування гри

Проведення етапу тестування розробленого мобільного застосунку є невід'ємною частиною життєвого циклу розробки програмного забезпечення, що дозволяє оцінити відповідність реалізованого функціоналу вихідним технічним вимогам.

В ході тестування перевірялись всі можливі варіанти взаємодії гравця з оточуючим світом, предметами, тригерами, колізіями, перевірялась правильність налаштування порядку шарів спрайтів та керування самим персонажем.

результати базових сценаріїв тестування систематизовано нижче у таблиці 4.1.

Таблиця. 4.1 – Тестування гри

№ п/п	1	2	3	4	5
	Призначення	Дії	Очікуваний результат	Отриманий результат	підсумок
1	Перевірка керування персонажем	Натиснути W,A,S,D,F,M почергово	Персонаж рухається згідно натиснутої клавіші, відкривається мапа, ставиться багаття	Персонаж рухається згідно натиснутої клавіші, відкривається мапа, ставиться багаття	Пройдено

Продовження таблиці 4.1

№ п/п	1	2	3	4	5
2	Перевірка відповідності анімацій	Натискаючи кнопки переміщення дивитись за програвасною анімацією	Анімація програвється згідно напрямку руху	Анімація програвється згідно напрямку руху	пройдено
3	Перевірка взаємодії гравця з їжею	Перемістити персонажа до їжі	Іжа зникає, слайдер голоду заповнюється	Іжа зникає, слайдер голоду заповнюється	пройдено
4	Перевірка взаємодії гравця з дверями	Пройти персонажем через двері	Персонаж переміщується на іншу сцену та навпаки при взаємодії з дверями в новій сцені	Персонаж переміщується на іншу сцену та навпаки при взаємодії з дверями в новій сцені	пройдено
5	Перевірка правильної роботи куща з ягодами	Спостерігати за кущем	На кущі з'являється ягідка, при взаємодії гравця з нею — зникає, через деякий час з'являється	На кущі з'являється ягідка, при взаємодії гравця з нею — зникає, через деякий час з'являється	пройдено

Продовження таблиці 4.1

№ п/п	1	2	3	4	5
6	Перевірка взаємодії гравця з багаттям	Перемістити персонажа в область дії багаття	Показник температури росте, при віддаленні - падає	Показник температури росте, при віддаленні - падає	пройдено

За результатами тестування було підтверджено повну функціональність додатку.

4.2 Результати роботи

Результатом виконання дипломного проєкту є повністю працездатний та оптимізований ігровий додаток у жанрі 2D-квесту з механіками виживання, розроблений на базі ігрового двигуна Юніті.

У ході роботи було реалізовано:

- систему керування персонажем;
- динамічну зміну життєвих показників голоду, температури, здоров'я;
- тригерну систему взаємодії з об'єктами (багаття, їжа, двері);
- механізм автоматичного створення ігрових ресурсів на мапі;
- систему збереження позиції гравця та прогресу між сесіями.

Створена гра має гнучку архітектуру, що дозволяє легко додавати нові квести, локації, їжу без зміни ядра коду. Інтерфейс користувача оптимізований під різні дисплеї мобільних пристроїв на базі андроїд. Нище наведено скріншоти ігрового додатка, що демонструють фінальний вигляд інтерфейсу та основний геймплей. (рисунок 4.1 – 4.7).

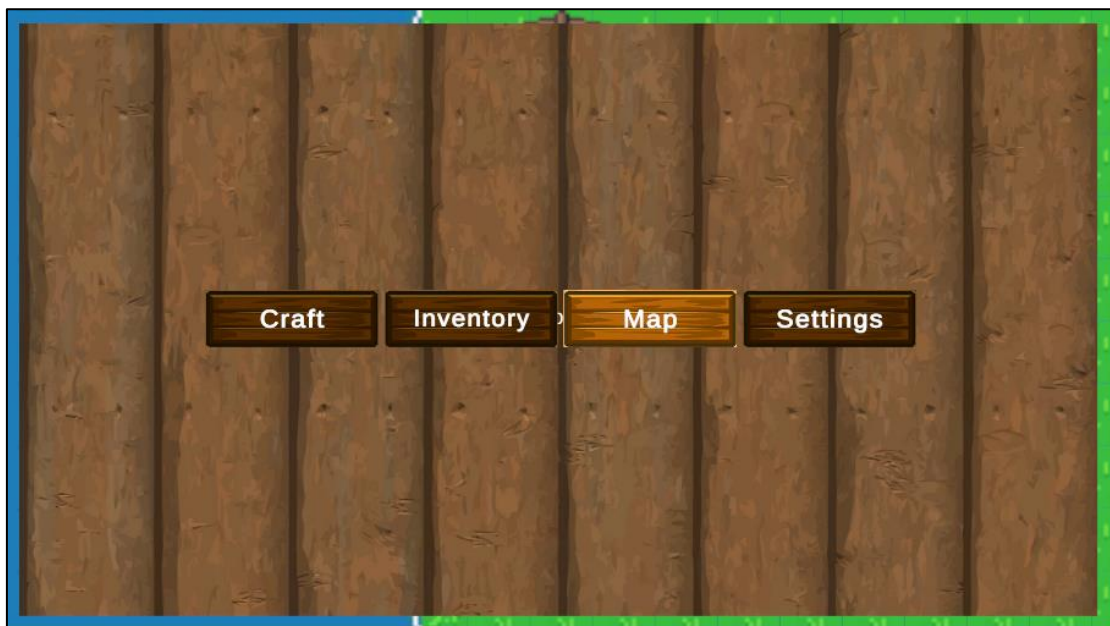


Рисунок 4.1 – Вигляд головного меню з підсвіткою кнопок



Рисунок 4.2 – Розміщення вогнища персонажем та обігрів

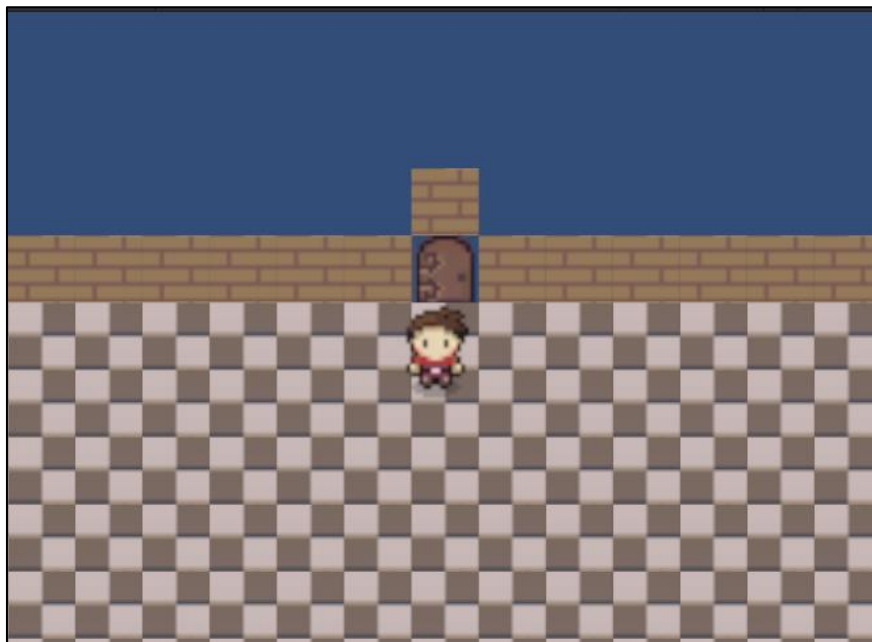


Рисунок 4.3 – Переміщення персонажа між сценами

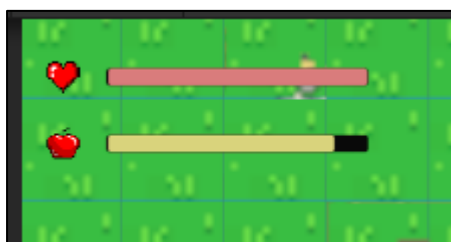


Рисунок 4.4 – Інтерфейс користувача з динамічними слайдерами



Рисунок 4.5 – Поява ягоди на ягідному куці



Рисунок 4.6 – В залежності від часу росту змінюється розмір та калорійність
ягоди



Рисунок 4.7 – Персонаж з'їв ягоду та відновив шкалу голоду

ВИСНОВКИ

У дипломній роботі виконано теоретичне обґрунтування, проєктування та програмну реалізацію працездатного прототипу мобільного ігрового застосунку в жанрі 2D-квесту на базі сучасного рушія Unity.

За результатами виконання поставлених завдань отримано такі основні висновки та науково-практичні результати:

Проведено аналіз технологічних рішень у сфері розробки мобільних ігор та обґрунтовано вибір компонентно-орієнтованого середовища Unity. Визначено, що інтегрована підтримка мови програмування C# забезпечує максимальну гнучкість при побудові логічних систем квесту, високу швидкість обробки подій та сумісність із цільовою операційною системою Android.

Розроблено модульну архітектуру сцени, в якій за допомогою інструментів Tilemap оптимізовано структуру двовимірного простору. Реалізовано ієрархічну систему графічного інтерфейсу користувача на базі компонента Canvas із інтеграцією технології TextMeshPro, що гарантує збереження чіткості текстових квестових підказок при масштабуванні на екранах мобільних пристроїв із різною роздільною здатністю.

Реалізовано ігрові механіки взаємодії, включаючи алгоритми динамічного прорахуна життєвих показників персонажа, а також логіку функціональних тригерів для фіксації контактів із квестовими об'єктами, дверима та зонами відпочинку.

Проведено тестування гри. Тестування підтвердило стабільність основних ігрових циклів, навігації та інтерфейсу.

Створений програмний продукт повністю відповідає критеріям технічного завдання і має практичну цінність як самостійний ігровий проєкт, а також як база для дослідження принципів оптимізації 2D-графіки на мобільних платформах.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1 ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура та правила оформлювання. На заміну ДСТУ3008-95 ; чинний від 2017-07-01. Вид. офіц. Київ : УкрНДНЦ, 2016. 32 с.

2 ДСТУ 8302:2015. Інформація і документація. Бібліографічне посилання. Загальні положення та правила складання. Чинний від 2016-07-01. Вид. офіц. Київ : УкрНДНЦ, 2016. 30 с.

3 Нові бланки НУ «Запорізька політехніка». *Запорізька політехніка*. URL: <https://zp.edu.ua/novi-blanki-zntu/> (дата звернення: 15.05.2026).

4 Документація Unity. URL: <https://docs.unity3d.com/Manual/index.html> (дата звернення: 15.05.2026).

5 Сайт з асетами для гри. URL: <https://opengameart.org/> (дата звернення: 15.05.2026).

6 Code Monkey URL: <https://unitycodemonkey.com/kitchenchaoscourse.php> (дата звернення: 15.05.2026).

7 15 найкращих 2д платформерів на андроїд URL: <https://vistrya.com.ua/statti/top-15-best-2d-platformers/> (дата звернення: 15.05.2026).

8 Основи юніті URL: <https://itproger.com/ua/course/unity/5> (дата звернення: 15.05.2026).

9 A History of the Unity Game Engine URL: http://www.daelab.cn/wp-content/uploads/2023/09/A_History_of_the_Unity_Game_Engine.pdf (дата звернення: 15.05.2026).

10 Unity Engine Dissection: Improvement Points in Comparison between Unity Engine and Open- Source Engines URL: <http://www.apjcriweb.org/content/vo110no11/33.pdf> (дата звернення: 15.05.2026).

11 Unity Game Development Engine: A Technical Survey URL: https://www.researchgate.net/profile/Faizan-Hussain-2/publication/348917348_Unity_Game_Development_Engine_A_Technical_Survey/links/60167daf45851517ef2b2c0d/

Unity-Game-Development-Engine-A-Technical-Survey.pdf (дата звернення: 15.05.2026).

12 The robot engine — Making the unity 3D game engine work for HRI URL: <https://ieeexplore.ieee.org/abstract/document/7333561> (дата звернення: 15.05.2026).

13 United We Stand: Platforms, Tools and Innovation With the Unity Game Engine URL: <https://journals.sagepub.com/doi/full/10.1177/2056305119880177> (дата звернення: 15.05.2026).

14 Unity game engine in visualization, simulation and modelling URL: <https://trepo.tuni.fi/bitstream/handle/10024/133433/NieminenTopi.pdf> (дата звернення: 15.05.2026).

15 Gaming Engines: Unity, Unreal, and Interactive 3D Spaces URL: <https://www.tandfonline.com/doi/pdf/10.1080/24751448.2021.1967068> (дата звернення: 15.05.2026).

ДОДАТОК А

ЛІСТИНГИ СКРИПТІВ

Лістинг А.1 – Скрипт CampFire.cs

```
using UnityEngine;

public class CampFire : Fire
{
    [SerializeField]
    private float heatingValue = 2f;

    public override void startAction(PlayerHealth playerHealth)
    {
        playerHealth.setUpdateTempValue(heatingValue);
    }

    public override void stopAction(PlayerHealth playerHealth)
    {
        playerHealth.setUpdateTempValue(-1f);
    }
}
```

Лістинг А.2 – Скрипт Door.cs

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class Door : InteractiveObject
{
    [SerializeField]
    private float yPositionOffset = 1f;
    [SerializeField]
    private bool savePosition = false;
    [SerializeField]
    private string sceneName;
    public override void action()
    {
        if (savePosition)
        {
            PlayerPrefs.SetInt(PlayerConsts.RESTART_POSITION_AVAILABLE, 1);
            PlayerPrefs.SetFloat(PlayerConsts.X_RESTART_POSITION,
                gameObject.transform.position.x);
            PlayerPrefs.SetFloat(PlayerConsts.Y_RESTART_POSITION,
                gameObject.transform.position.y - yPositionOffset);
        }
        SceneManager.LoadScene(sceneName);
    }
}
```

```
}
```

Лістинг А.3 – Скрипт Fire.cs

```
using UnityEngine;

public abstract class Fire : MonoBehaviour
{
    public abstract void startAction(PlayerHealth playerHealth);
    public abstract void stopAction(PlayerHealth playerHealth);
}
```

Лістинг А.4 – Скрипт Food.cs

```
using UnityEngine;

public class Food : FoodObject
{
    [SerializeField]
    private float calories = 10;
    [SerializeField]
    private float growSpeed = 0.0003f;
    private Vector3 originScale;
    public override void eat(PlayerHealth playerHealth)
    {
        playerHealth.addCalories(calories);
        Destroy(gameObject);
    }

    void Start()
    {
        originScale = transform.localScale;
        transform.localScale = Vector3.zero;
    }

    void Update()
    {
        grow();
    }

    private void grow()
    {
        if (originScale.x > transform.localScale.x)
        {
            Vector3 newsScale = transform.localScale;
            newsScale.x += growSpeed * Time.deltaTime;
            newsScale.y += growSpeed * Time.deltaTime;
            calories += growSpeed * Time.deltaTime * 1000;
            transform.localScale = newsScale;
        }
    }
}
```

```
}
```

Лістинг А.5 – Скрипт FoodObject.cs

```
using UnityEngine;

public abstract class FoodObject : MonoBehaviour
{
    public abstract void eat(PlayerHealth playerHealth);
}
```

Лістинг А.6 – Скрипт FruitSpawner.cs

```
using UnityEngine;

public class FruitSpawner : MonoBehaviour
{
    [SerializeField]
    private GameObject fruitPrefab;
    private GameObject fruit;
    [SerializeField]
    private float fruitPosOffsetX = 0.3f;
    [SerializeField]
    private float fruitPosOffsetY = 0.3f;
    private bool isFruitCreated = false;
    void Start()
    {

    }

    void Update()
    {
        if(fruit == null && !isFruitCreated)
        {
            Invoke("spawnFruit", 4);
            isFruitCreated=true;
        }

    }
    private void spawnFruit()
    {
        fruit = Instantiate(fruitPrefab);
        Vector3 pos = transform.position;
        pos.x += fruitPosOffsetX;
        pos.y += fruitPosOffsetY;
        fruit.transform.position = pos;
        isFruitCreated = false;
    }
}
```

Лістинг А.7 – Скрипт FruitSpawner.cs

```
using UnityEngine;

public abstract class InteractiveObject : MonoBehaviour
{
    public abstract void action();
}
```

Лістинг А.8 – Скрипт MenuController.cs

```
using UnityEngine;
using UnityEngine.InputSystem;

public class MenuController : MonoBehaviour
{
    void Start()
    {
        gameObject.SetActive(false);
    }
    public void OnMenuClick(InputAction.CallbackContext context)
    {
        if (context.performed)
        {
            gameObject.SetActive(!gameObject.activeSelf);
        }
    }
}
```

Лістинг А.9 – Скрипт Player.cs

```
using UnityEngine;
using UnityEngine.InputSystem;
using UnityEngine.SceneManagement;

public class Player : MonoBehaviour
{
    [SerializeField]
    private bool reloadPosition = false;
    [SerializeField]
    private float speed = 3f;
    private Rigidbody2D rb;
    [SerializeField]
    private Animator animator;
    private Vector2 move;
    private Vector2 lastMove;
```

```

[SerializeField]
private GameObject campFirePrefab;

void Start()
{
    rb=GetComponent<Rigidbody2D>();
    int isPositionAvalaible =
PlayerPrefs.GetInt(PlayerConsts.RESTART_POSITION_AVALAIBLE);
    if (reloadPosition&& isPositionAvalaible == 1)
    {
        gameObject.transform.position = loadRestartPosition();
    }
}

// Update is called once per frame
void Update()
{
    rb.linearVelocity = move * speed;
}

public void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.gameObject.tag.Equals("interactive"))
    {
collision.gameObject.GetComponent<InteractiveObject>().action();
    }else if(collision.gameObject.tag.Equals("food"))
    {
collision.gameObject.GetComponent<FoodObject>().eat(GetComponent<Pla
yerHealth>());
    }
    else if (collision.gameObject.tag.Equals("fire"))
    {
collision.gameObject.GetComponent<Fire>().startAction(GetComponent<P
layerHealth>());
    }
}

public void OnTriggerExit2D(Collider2D collision)
{
    if (collision.gameObject.tag.Equals("fire"))
    {

```

```

collision.gameObject.GetComponent<Fire>().stopAction(GetComponent<Pl
ayerHealth>());
    }
}

```

```

public void Move(InputAction.CallbackContext context)
{
    animator.SetBool("isMoving", true);

    if (context.canceled)
    {
        animator.SetBool("isMoving", false);
        animator.SetFloat("lastX", move.x);
        animator.SetFloat("lastY", move.y);
        lastMove = move;
    }
}

```

```

    move = context.ReadValue<Vector2>();
    animator.SetFloat("inputX", move.x);
    animator.SetFloat("inputY", move.y);
}

```

```

public void Interact(InputAction.CallbackContext context)
{
    if (context.performed)
    {
        GameObject campFire = Instantiate(campFirePrefab);
        Vector2 campFirePos = transform.position;
        campFirePos.x += lastMove.x;
        campFirePos.y += lastMove.y;
        campFire.transform.position = campFirePos;
    }
}

```

```

private Vector2 loadRestartPosition()
{
    return new Vector2(

        PlayerPrefs.GetFloat(PlayerConsts.X_RESTART_POSITION),
        PlayerPrefs.GetFloat(PlayerConsts.Y_RESTART_POSITION)
    );
}
}

```

Лістинг А.10 – Скрипт PlayerConsts.cs

```
using UnityEngine;

public class PlayerConsts
{
    public const string X_RESTART_POSITION = "x_restart_position";
    public const string Y_RESTART_POSITION = "y_restart_position";
    public const string RESTART_POSITION_AVALAIBLE =
"restart_position_avaaible";
}
```

Лістинг А.11 – Скрипт PlayerHealth.cs

```
using System;
using UnityEngine;
using UnityEngine.UI;
public class PlayerHealth : MonoBehaviour
{
    [SerializeField]
    private Slider healthSlider;
    [SerializeField]
    private Slider hungerSlider;
    private Image hungerFillImage;
    private Color mainHungerColor;
    private Image healthFillImage;
    private Color mainHealthColor;
    [SerializeField]
    private float hungerUpdateSpeed = 0.05f;
    [SerializeField]
    private float healthUpdateSpeed = 0.05f;

    private const float MAX_PLAYER_TEMP = 100f;
    [SerializeField]
    private float playerTemp = MAX_PLAYER_TEMP;
    [SerializeField]
    private float updateTempValue = 0f;

    void Start()
    {
        hungerFillImage =
hungerSlider.fillRect.GetComponent<Image>();
        mainHungerColor = hungerFillImage.color;

        healthFillImage =
healthSlider.fillRect.GetComponent<Image>();
        mainHealthColor = healthFillImage.color;
    }

    // Update is called once per frame
    void Update()
    {
        updatehunger();
    }
}
```

```

        updatehealth();
        updatetemp();
    }

    private void updatehunger()
    {
        if (hungerSlider.value != 0)
        {
            hungerSlider.value = hungerSlider.value -
(hungerUpdateSpeed * Time.deltaTime);
            ChangeHungerColor(hungerSlider, hungerFillImage,
mainHungerColor);
        }
    }

    private void updatehealth()
    {
        if (hungerSlider.value == 0)
        {
            healthSlider.value = healthSlider.value -
(healthUpdateSpeed * Time.deltaTime);
            ChangeHungerColor(healthSlider, healthFillImage,
mainHealthColor);
        }
    }

    private void updatetemp()
    {
        if (updateTempValue > 0 && playerTemp < MAX_PLAYER_TEMP ||
updateTempValue < 0 && playerTemp > 0)
        {
            playerTemp += updateTempValue * Time.deltaTime;
            playerTemp = Math.Clamp(playerTemp, 0, MAX_PLAYER_TEMP);
        }
    }

    public void setUpdateTempValue (float value)
    {
        updateTempValue = value;
    }

    private void ChangeHungerColor(Slider slider , Image image,
Color mainColor)
    {
        if (slider.value <15 && image.color != Color.red)
        {
            image.color = Color.red;
        } else if(slider.value > 15 && image.color != mainColor)
        {
            image.color = mainColor;
        }
    }

```

```

    }

    public void addCalories(float calories)
    {
        hungerSlider.value = Mathf.Min(100, hungerSlider.value +
calories);
    }
}

```

Лістинг А.12 – Скрипт TabsController.cs

```

using System;
using UnityEngine;
using UnityEngine.UI;

public class TabsController : MonoBehaviour
{
    [SerializeField]
    public GameObject [] tabs;
    private int LastTabIndex = 0;
    [SerializeField]
    public GameObject[] pages;
    private int LastPageIndex = 0;
    void Start()
    {
        for (int i = 0; i < tabs.Length; i++)
        {
            if (i>0)
            {
                tabs[i].GetComponent<Image>().color = Color.gray;
                pages[i].SetActive(false);
            }
        }
    }
    public void NextPage(int index)
    {
        if (index != LastTabIndex)
        {
            tabs[index].GetComponent<Image>().color = Color.white;
            tabs[LastTabIndex].GetComponent<Image>().color =
Color.gray;
            LastTabIndex = index;

            pages[index].SetActive(true);
            pages[LastTabIndex].SetActive(false);
            LastPageIndex = index;
        }
    }
}

```

ДОДАТОК Б

СЛАЙДИ ПРЕЗЕНТАЦІЇ

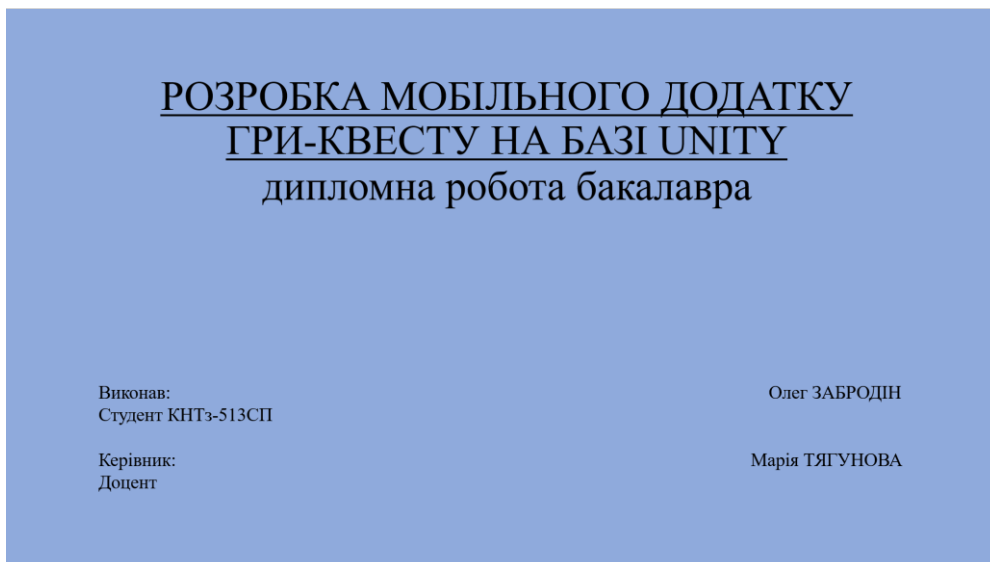


Рисунок Б.1 – Слайд 1

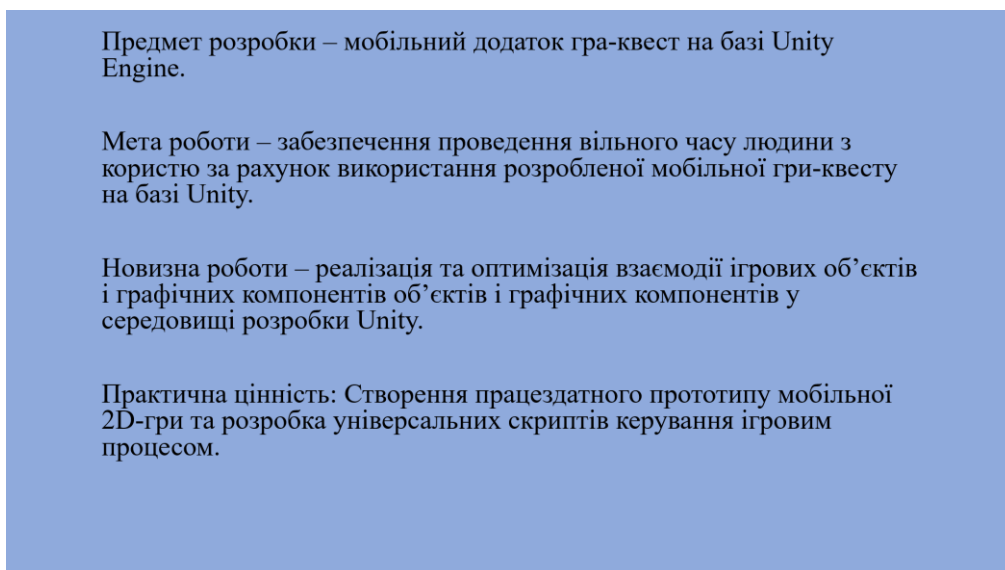


Рисунок Б.2 – Слайд 2

АНАЛІЗ ІСНУЮЧИХ АНАЛОГІВ

Критерій порівняння	Forager	Soul Knight	Розроблений проєкт
Жанровий фокус	Виявлення, крафт, економіка	Динамічний екшен, шутер	Квест з елементами виживання
Стиль та складність графіки	Деталізований Pixel Art (багатошарові анімації)	Складний Pixel Art (масивні спефекти, частки)	Мінімалістичний 2D Pixel Art (оптимізовані спрайти)
Розмір додатку (МБ)	Великий (150+ МБ)	Великий (300+ МБ)	Мінімальний (50 МБ)
Монетизація	Платна гра	Нав'язлива реклама	Безкоштовна, без реклами
Архітектура коду	Закрита (Комерційна)	Закрита (Комерційна)	Відкрита (Модульна)

Аналоги: Орієнтовані на тривалу комерційну гру, мають велику вагу файлів та висувають серйозні вимоги до процесора й оперативної пам'яті через велику кількість ефектів.

Мій проєкт: Сфокусований на чистому геймплеї без реклами, має мінімальну вагу і оптимізований для стабільної роботи навіть на застарілих смартфонах.

Рисунок Б.3 – Слайд 3

ТЕХНІЧНІ ВИМОГИ ДО ФУНКЦІОНАЛУ ДОДАТКУ

Функціональні вимоги:

- Керування персонажем: реалізація базового переміщення персонажа у двовимірному просторі.
- Взаємодія з об'єктами: обробка подій зіткнення та активація тригерів при взаємодії з оточенням.
- Переміщення між сценами: коректне завантаження та вивантаження ігрових сцен із збереженням логіки гри.

Нефункціональні вимоги:

- Стабільна робота на операційній системі Android.
- Плавна робота графіки (мінімум 60 FPS)
- Мінімальний розмір інсталяційного файлу.

Рисунок Б.4 – Слайд 4

ОБГРУНТУВАННЯ ВИБОРУ ІГРОВОГО РУШІЯ ТА МОВИ РОЗРОБКИ

Ігровий рушій Unity:

- Наявність системи *Tilemap* для оптимізованого проектування піксельних рівнів із виглядом зверху.
- Гнучкі інструменти для збірки та стиснення текстур під платформу Android.
- Низьке споживання системних ресурсів при роботі з двовимірними спрайтами.



Мова програмування C#:

- Зручне написання модульних скриптів для керування персонажем та об'єктами.
- Висока швидкість обробки логіки взаємодії та подій у грі.



Рисунок Б.5 – Слайд 5

АЛГОРИТМ РОБОТИ

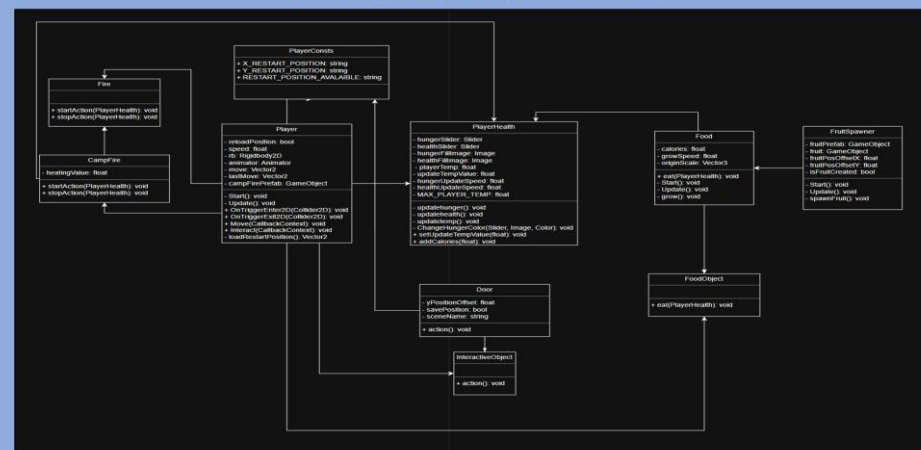


Рисунок Б.6 – Слайд 6



Рисунок Б.7 – Слайд 7

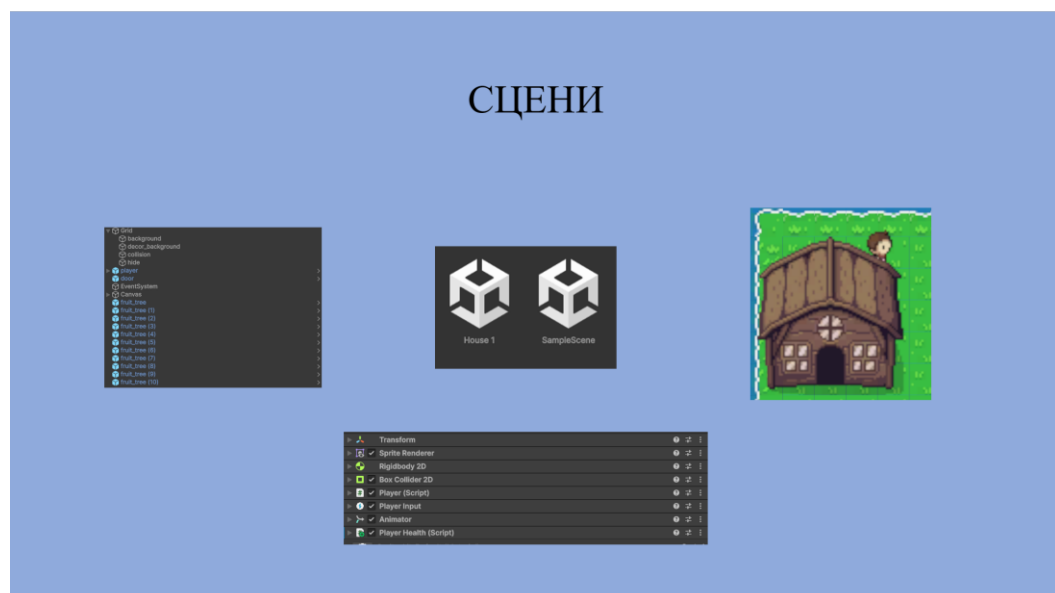


Рисунок Б.8 – Слайд 8

ІНТЕРФЕЙС КОРИСТУВАЧА



Рисунок Б.9 – Слайд 9

ВИСНОВКИ

В ході виконання дипломного проекту було отримано такі результати:

- Проаналізовано існуючі аналоги мобільних 2D-ігор та обґрунтовано вибір інструментів розробки (Unity та мова C#).
- Розроблено та оптимізовано графічні компоненти гри (піксельні 2D-спрайти) для забезпечення низьких системних вимог додатка.
- Реалізовано базовий функціонал ігрового процесу: керування персонажем у перспективі Top-Down, систему взаємодії з об'єктами та логіку переміщення між сценами.
- Створено працездатний прототип мобільної гри під операційну систему Android із розміром інсталяційного файлу до 50 МБ.
- Підтверджено стабільну продуктивність додатка (60 FPS) на мобільних пристроях з обмеженими апаратними ресурсами.

Рисунок Б.10 – Слайд 10

РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ ГРИ-КВЕСТУ НА БАЗІ UNITY

Доповідь закінчено. Дякую за увагу!

Виконав:
Студент КНТз-513СП

Олег ЗАБРОДІН

Керівник:
Доцент

Марія ТЯГУНОВА

Рисунок Б.11 – Слайд 11