

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

магістр

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ МЕТОДІВ
ТА ЗАСОБІВ РОЗРОБКИ ІГРОВОГО СЕРЕДОВИЩА
RESEARCH AND SOFTWARE IMPLEMENTATION OF METHODS
AND TOOLS FOR DEVELOPING A GAMING ENVIRONMENT

Виконав: студент 2 курсу, групи КНТ-114м

Спеціальності 121 Інженерія програмного

(код і найменування спеціальності)

забезпечення

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

СЛОБОДЯН М.В.

(ПРІЗВИЩЕ та ініціали)

Керівник ПАРХОМЕНКО А.В.

(ПРІЗВИЩЕ та ініціали)

Рецензент ГОЛУБ Т.В.

(ПРІЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Факультет ФКНТ
Кафедра програмних засобів
Ступінь вищої освіти магістр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
“ ” 2025 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

СЛОБОДЯНА Микити Володимировича

(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація методів та засобів розробки ігрового середовища. Research and Software Implementation of Methods and Tools for Developing a Gaming Environment

керівник проєкту (роботи) к.т.н., доцент, ПАРХОМЕНКО Анжеліка Володимирівна,

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

затверджені наказом закладу вищої освіти від “ 30 ” вересня 2025 року № 447

2. Строк подання студентом проєкту (роботи) 10 грудня 2025 року

3. Вихідні дані до проєкту (роботи) рекомендована література, завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Проєктування апаратно-програмного забезпечення мобільного застосунку. 3. Розробка та реалізація апаратно-програмного забезпечення. 4. Тестування та використання розробленого апаратно-програмного мобільного забезпечення.

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

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ПАРХОМЕНКО А.В., доцент		
Нормоконтроль	БЄЛОВА А.В., асистент		

7. Дата видачі завдання « 30 » вересня 2025 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання
2	Аналіз предметної області.	2-3 тижні	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	4 тиждень	Розділ 2
4	Розробка архітектури програми.	5-6 тижні	Розділ 3
5	Розробка програми.	7-8 тижні	Розділ 3
6	Тестування та експериментальне дослідження програмного забезпечення.	9 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	10-11 тижні	ПЗ, Додатки
8	Нормоконтроль та рецензування.	12 тиждень	
9	Захист роботи.	12 тиждень	

Студент

Підпис
(підпис)

Микита СЛОБОДЯН
(Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

(підпис)

Анжеліка ПАРХОМЕНКО
(Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
123 с., 14 табл., 38 рис., 2 дод., 37 джерел.

МАТЕРІАЛЬНИЙ ІНТЕРФЕЙС, СЕРЕДОВИЩЕ ВІРТУАЛЬНОЇ РЕАЛЬНОСТІ, ІГРОВИЙ РУШІЙ, ПРОГРАМУВАННЯ МОВОЮ C#, ІГРОВИЙ ЗАСТОСУНОК, ARDUINO, UNITY.

Об'єкт дослідження – процес взаємодії користувача з ігровим середовищем через матеріальний інтерфейс.

Предмет дослідження – методи та засоби організації матеріального інтерфейсу для взаємодії користувача з ігровим середовищем

Мета роботи – покращення користувацького досвіду з мобільним ігровим застосунком шляхом реалізації матеріального інтерфейсу.

Матеріали, методи та технічні засоби: структурне та об'єктно-орієнтоване програмування, C#, BlueUnity, Arduino Nano, акселерометр ADXL345, Bluetooth Serial Module HC-05, Unity, Unity Asset Store, мобільний пристрій OnePlus Nord CE 3 Lite, персональний комп'ютер під управлінням операційної системи Microsoft Windows 11.

Результати. Створено мобільний ігровий застосунок та матеріальний інтерфейс для взаємодії з ним.

Висновки. Розроблені рішення дозволяють покращити досвід користувача при грі на мобільному пристрої.

Галузь використання – ігрова індустрія.

ABSTRACT

Explanatory note to to the diploma qualifying work of the master:
123 pages, 14 tables, 38 figures, 2 appendices, 37 sources.

MATERIAL INTERFACE, VIRTUAL REALITY ENVIRONMENT,
GAME ENGINE, C# PROGRAMMING, GAME APPLICATION, ARDUINO,
UNITY.

The object of the study is the process of user interaction with the gaming environment through a material interface.

The subject of the study is methods and means for organizing a material interface for user interaction with the gaming environment.

The purpose of the work is to improve the user experience with a mobile gaming application by implementing a material interface.

Materials, methods and technical means: structured and object-oriented programming, C#, BlueUnity, Arduino Nano, ADXL345 accelerometer, Bluetooth Serial Module HC-05, Unity, Unity Asset Store, OnePlus Nord CE 3 Lite mobile device, personal computer running Microsoft Windows 11 operating system.

Results. A mobile gaming application and a material interface for interacting with it have been created.

Conclusions. The developed solutions allow improving the user experience when playing on a mobile device.

Industry of use – gaming industry.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок	8
Вступ	9
1 Аналіз предметної області	10
1.1 Огляд жанрів ігор	10
1.2 Огляд аналогів ігор жанру нескінченний раннер	12
1.3 Дослідження інструментів для створення ігрової графіки	15
1.4 Дослідження внутрішньоігрових магазинів.....	18
1.4.1 Різновиди внутрішньоігрових магазинів.....	19
1.5 Дослідження різновидів керування в іграх	21
1.6 Постановка завдань дипломної роботи	24
2 Проєктування апаратно-програмного забезпечення мобільного застосунку	26
2.1 Призначення та вимоги до розроблюваної системи	26
2.2 Вибір технологій для реалізації гри	29
2.2.1 Вибір стеку технологій для програмної реалізації гри.....	29
2.2.2 Вибір стеку технологій для програмної та апаратної реалізації матеріального інтерфейсу	32
2.3 Висновки до розділу 2.....	35
3 Розробка та реалізація апаратно-програмного забезпечення.....	36
3.1 Проєктування основних алгоритмів гри за допомогою діаграм	36
3.2 Проєктування ігрового контролера на базі Arduino	43
3.3 Розробка ігрового середовища.....	47
3.3.1 Створення стартової локації та генерація платформ	47
3.3.2 Створення персонажа та реалізація скрипта переміщення	52
3.3.3 Реалізація програшу гравця	55
3.4 Створення графічного інтерфейсу.....	56
3.5 Реалізація музичного супроводу	64
3.5 Висновки до розділу 3	65

4 Тестування та використання розробленого апаратно-програмного забезпечення	66
4.1 Тестування	66
4.1.1 З'єднання контролера з телефоном по Bluetooth	66
4.1.2 Розрив з'єднання з контролером	68
4.1.3 Робота меню паузи	70
4.1.4 Програш гравця. Обробка зіткнень із перешкодами	71
4.1.5 Перевірка підключення контролера до старту гри	73
4.1.6 Перевірка покупки товару	74
4.1.7 Перевірка вибору візуального стилю в меню інвентарю	76
4.2 Порадник для користувача.....	78
4.3 Порадник для програміста	79
4.4 Висновки до розділу 4	82
Висновки.....	83
Перелік джерел посилання	84
Додаток А Текст програми	87
Додаток Б Слайди презентації.....	115

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

AAA	– Triple-A;
COM	– Communication Port;
FPS	– First-Person Shooter;
F2P	– Free-To-Play;
HP	– High-Poly;
LP	– Low-Poly;
IDE	– Integrated Development Environment;
NPC	– Non-Player Character;
NURBS	– Non-uniform rational B-spline;
P2P	– Pay-To-Play;
QA	– Quality Assurance;
RB	– Programmable-Gain Amplifier;
RPG	– Role-Playing Games;
RTS	– Real-Time Strategy;
UI	– User Interface;
UX	– User Experience;
ПК	– персональний комп'ютер;
ПЗ	– програмне забезпечення.

ВСТУП

Ігрова індустрія розвивається неймовірними темпами. На сьогоднішній день існує більш ніж мільйон відеоігор та понад 30 різних жанрів. Найпопулярнішими жанрами ігор є action, RPG, пісочниці, симулятори і нескінченні раннери. Майже в кожній країні світу є своя компанія яка створює та випускає відеоігри. Україна також має свої компанії. Найвідоміша з них – 4A Games яка створила три гри серії Metro [1].

Для взаємодії з ігровим середовищем використовуються матеріальні інтерфейси. Це можуть бути клавіатура та миша, геймпад, ігрова гітара, барабани, джойстики та інші. Як і для комп'ютерів існують матеріальні інтерфейси для мобільних пристроїв. Якщо для взаємодії з комп'ютером матеріальний інтерфейс це необхідність, то для мобільних пристроїв вони використовуються для поліпшення користувацького досвіду.

Оскільки матеріальні інтерфейси мають досить велику ціну, а користувачі хочуть взаємодіяти з графічним інтерфейсом так само, як це відбувається у реальному житті, тобто не просто натискати на кнопки на екрані, вчені та розробники продовжують розробляти нові види матеріальних інтерфейсів.

Областю дослідження роботи є ігрові мобільні застосунки та матеріальні інтерфейси для взаємодії користувача з ігровим середовищем. Проблематика використання матеріальних інтерфейсів у мобільних іграх полягає у пошуку балансу між доступністю та технічною реалізацією. Тому тема роботи є актуальною.

Метою дипломної роботи є покращення користувацького досвіду з мобільним ігровим застосунком шляхом реалізації матеріального інтерфейсу.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд жанрів ігор

Як показало проведене дослідження ігрових жанрів, їх відмінностей та різновидів, ігри мають свої види, підвиди та жанри. Жанри ігор – це категорії, які використовуються для класифікації відеоігор за їх стилем, ігровою механікою та іншими характеристиками [2]. Використовуючи пошук за жанрами користувачі мають можливість легко знаходити ігри, які відповідають їхнім інтересам, бажанням та уподобанням. Зазвичай одна гра поєднує у собі елементи різних жанрів. Такі жанри називаються гібридними. Наприклад, гра може поєднувати елементи пригодницької гри, RPG, і стратегій, створюючи новий ігровий досвід. Загалом ігри поділяються на такі жанри [2]:

- екшен (Action);
- пригодницькі ігри (Adventure);
- рольові ігри (RPG);
- симулятори (Simulation);
- стратегії (Strategy);
- спортивні ігри (Sports);
- ігри жахів (Horror);
- нескінченні раннери (Endless Run).

Жанр «Екшен» відрізняється високою динамічністю, акцентує увагу на рефлексії та швидке прийняття рішень [2]. У іграх цього жанру значну частину ігрового часу займає знищення ворогів. Важливим аспектом жанру є наявність рівнів та перешкод, які користувачі повинні подолати, аби отримати певні досягнення та нагороди. Хоча акцент жанру більше сфокусований на діях, безліч ігор мають глибоку сюжетну лінію і продуманих персонажів. Найкращим доповненням до жанру є мультиплеєр, що дає можливість гравцям змагатися між собою або співпрацювати.

Рольові ігри – це подорож у світ фантазій та пригод, де гравець стає не

просто спостерігачем, а активним учасником захопливих подій. Цей жанр відкриває нескінченні можливості для самовираження, взаємодії з іншими персонажами та впливу на події у вигаданому світі [2]. Основними характерними елементами рольової гри є створення персонажа, його розвиток, відкритий світ, взаємодія з NPC.

Стратегічні ігри – жанр, що вимагає від гравця вміння планувати, аналізувати ситуацію та приймати обдумані рішення [2]. Цей жанр дає користувачеві можливість керувати ігровим світом використовуючи внутрішньоігрові механіки. Користувач постає у ролі державного діяча або внутрішньоігрового божества яке змінює світ або впливає на нього. Ігри цього жанру можна поділити на кілька підвидів, а саме покрокові стратегії, стратегії в реальному часі, глобальні стратегії.

Ігри жахів – це жанр який грає на почуттях користувачів, на їхніх страхах. Він може поєднувати в собі жанр екшену, оскільки всі події в грі відбуваються динамічно. Цьому жанру властиво створювати страшну атмосферу та психологічно тиснути на гравців для створення напруги та розкриття страхів гравців. Підвидами ігор жахів є психологічні хоррори, хоррор-квести, хоррор-виживачі.

Нескінченний раннер – це жанр ігор особливістю якого є ілюзія нескінченного рівня. Всі ігри такого жанру мають нескінченний рівень, який створюється процедурно або за допомогою рандомізації появи платформ. Зазвичай ігри такого жанру популярні на телефонах, оскільки вони не вимагають складних дій для взаємодії із собою. Прикладами таких ігор є Subway Surfers [3], Dinosaur Game [4], Sonic Dash [5].

Перелік жанрів та під жанрів наведено на рис. 1.1.

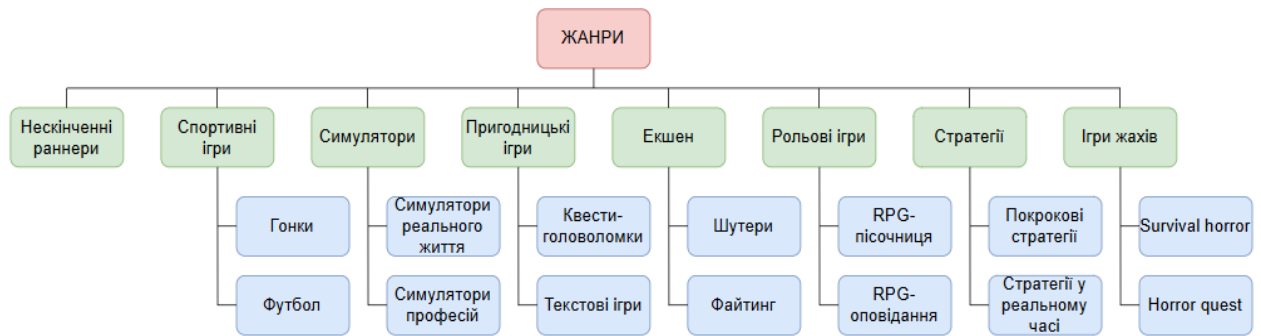


Рисунок 1.1 – Загальна таблиця жанрів та під жанрів

1.2 Огляд аналогів ігор жанру нескінченний раннер

Кожна гра жанру нескінченний раннер виділяється своїми особливостями та механіками. Для проведення огляду було обрано три гри, а саме Subway Surfers [3], Dinosaur Game [4], Sonic Dash [5].

Subway Surfers – це мобільна 3D гра в жанрі нескінченного раннера. Гравець постає в ролі молодого художника, який малює графіті на вагонах поїздів. Гра починається з моменту коли поліцейський помічає головного героя, після чого він починає тікати вздовж залізничних колій. Під час втечі гравець збирає монетки, так звану ігрову валюту, за яку згодом він може купувати нових героїв. Основне завдання гри набрати якнайбільше очок. Для керування головним героєм користувач використовує графічний інтерфейс. Процес створення перешкод включає їх появу за зоною видимості гравця для створення ілюзії нескінченності. Момент геймплею гри продемонстровано нижче (рис. 1.2) [3].

Dinosaur Game (Chrome Dino) – це браузерна 2D гра розроблена компанією Google і вбудована у веббраузер Google Chrome. Користувач керує піксельним динозавром переміщаючись ландшафтом. Ця гра працює в момент відсутності інтернету або ж в автономному режимі. Для запуску гри необхідно натиснути кнопку Space або ж натиснувши по екрану мобільного пристрою [4]. Сенс гри долати перешкоди у вигляді кактусів і птахів. Чим довше рухається гравець, тим швидше рухаються перешкоди і тим більше

очок отримує гравець. Момент геймплею гри продемонстровано нижче (рис.1.3) [4].



Рисунок 1.2 – Геймплей гри Subway Surfers [3]

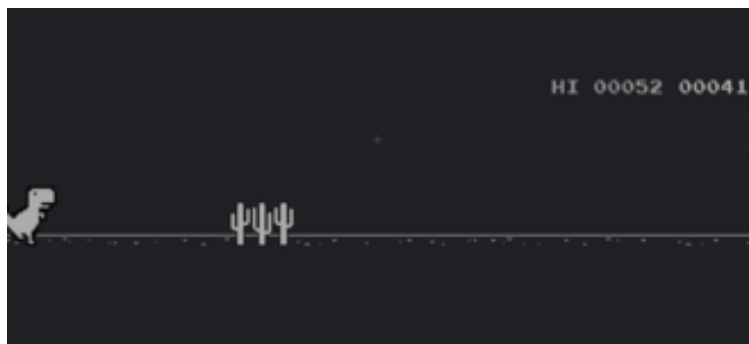


Рисунок 1.3 – Геймплей гри Dinosaur Game [4]

Sonic Dash – це мобільна 3D гра видана ігровою студією Sega. Основна мета гри - це уникати перешкоди і ворогів, збираючи золоті кільця. У грі присутній елемент змагання. Завдяки таблиці лідерів кожен гравець може побачити на якому він місці. Зароблені гравцем кільця дають можливість придбати покращення та додаткових персонажів. Для керування головним героєм користувач використовує графічний інтерфейс. Момент геймплею гри продемонстровано нижче (рис. 1.4) [5].

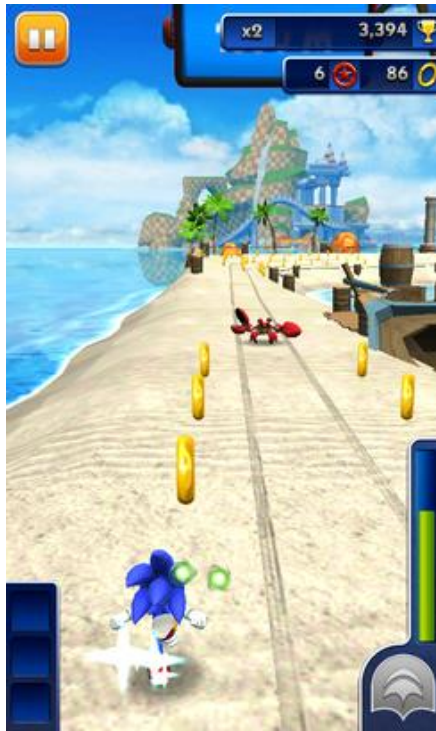


Рисунок 1.4 – Геймплей гри Sonic Dash [5]

Після огляду трьох ігор було складено порівняльну таблицю для аналізу їх особливостей та відмінностей (табл. 1.1) [3]-[5].

Таблиця 1.1 – Порівняльна таблиця ігор Subway Surfers, Dinosaur Game, Sonic Dash [3]-[5]

Критерії	Subway Surfers	Dinosaur Game	Sonic Dash
Платформи	Android, iOS	Браузер Chrome	Android, iOS
Сеттинг	Міська залізнична дорога	Пустеля	Фантастичний світ
Жанр	Нескінченний раннер	Нескінченний раннер	Нескінченний раннер
Режим гри	Одиночна	Одиночна	Одиночна
Керування	Пересування по екрану (вгору, вниз, ліворуч, праворуч)	Клавішами на клавіатурі або кліки на телефоні	Пересування по екрану, наявність додаткових кнопок

1.3 Дослідження інструментів для створення ігрової графіки

Під інструментами для створення ігрової графіки мається на увазі програмне забезпечення, яке має необхідний інструментарій, за допомогою якого можна створювати 2D спрайти та 3D моделі. На цей час існує багато програмних засобів, які значно спрощують процес створення ігрової графіки. Графіка в іграх зазвичай ділиться на 2D та 3D. Для створення 2D або 3D графіки використовуються різні інструменти. В цей час 2D та 3D графіка ділиться на кілька видів. 2D графіка може бути як растрової, так і векторної, а 3D графіка ділиться на Low-Poly та High-Poly. Нижче наведено приклади різновидів 2D та 3D графіки (рис. 1.5) [6]-[7].

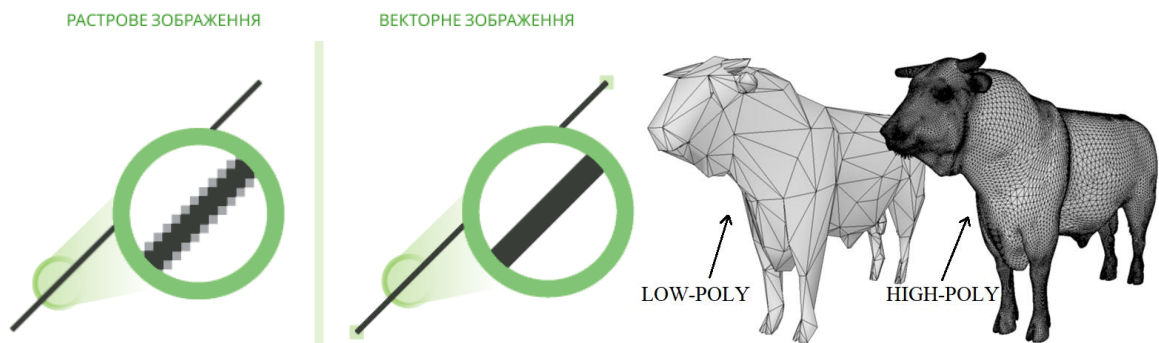


Рисунок 1.5 – Візуальні відмінності 2D та 3D видів графіки [6]-[7]

Для проведення дослідження було обрано 2D та 3D інструменти, а саме Adobe Photoshop, GIMP, Blender, Maya та 3DS Max. Ці інструменти були обрані через свою популярність серед розробників та функціональної складової.

Adobe Photoshop є одним з найпоширеніших інструментів для створення та малювання 2D-графіки. Його функціональні можливості включають широкий спектр інструментів для малювання, редагування, ретушування і створення композицій. Photoshop підтримує роботу із шарами, що дозволяє розробникам зберігати складні проекти у впорядкованому вигляді, а також створювати 2D анімації для різних персонажів, об'єктів та

предметів. Завдяки своїм розширеним можливостям Photoshop часто використовується для створення текстур, концепт-артів та інтерфейсів користувача [8]. Його великий інструментарій, гнучкість та можливість інтеграції з іншими продуктами робить його потужним інструментом для кожного розробника ігрової графіки. Завдяки постійним вдосконаленням Photoshop є актуальним вибором для професійної роботи у різних галузях, зокрема і створення ігрової графіки.

GIMP (GNU Image Manipulation Program) є безкоштовним та відкритим аналогом Adobe Photoshop. З його допомогою створюються різні графічні елементи, такі як текстури, спрайти, інтерфейси користувача і концепт-арт. Ця програма надає більшість функціональних можливостей, доступних у Photoshop, включаючи інструменти для малювання, редагування та роботи з шарами [9]. GIMP є популярна серед інді-розробників та невеликих студій завдяки своїй доступності та масштабованості з використанням численних плагінів. GIMP активно розвивається та підтримується спільнотою розробників та графічних дизайнерів і є актуальним інструментом професійного використання у створенні візуальних елементів для ігрової графіки.

Blender – це безкоштовна програма для 3D-моделювання, анімації, текстурювання та рендерингу. Вона має великий інструментарій для створення тривимірних моделей, скульптингу, текстуризації, анімації та симуляції фізики. Завдяки своїй багатofункціональності та відкритому коду, Blender використовується як незалежними розробниками, так і професійними студіями для створення як окремих елементів, так і повноцінних тривимірних сцен. Він підтримує багато форматів файлів, таких як FBX, OBJ, STL та інших, що полегшує обмін даними з іншими програмами. Також він має безліч безкоштовних аддонів і скриптів на Python, що дозволяють розширювати функціонал програми [10].

Maya та 3ds Max, розроблені компанією Autodesk, є двома досить

потужними програмами для 3D-моделювання, анімації та рендерингу. Вони використовуються в різних галузях, включаючи кіноіндустрію, архітектуру, візуалізацію та, зокрема, розробку ігрової графіки.

Maya пропонує розширені інструменти для полігонального, NURBS та subdivision моделювання, що дозволяє створювати високодеталізовані та складні моделі. Інструменти для скульптингу та ретопології забезпечують додаткову пружність у розробці органічних форм [11]. 3ds Max має потужні інструменти для полігонального та сплайнового моделювання, що дозволяє створювати як прості, так і високодеталізовані моделі. Інструменти для процедурного моделювання та модифікатори значно спрощують процес створення складних геометричних форм. Обидві програми використовуються для рендерингу високоякісних зображень та анімацій, які можуть використовуватись для створення трейлерів, кінематографічних вставок та маркетингових матеріалів. 3ds Max підтримує мови програмування MaxScript та Python, що дозволяє користувачам автоматизувати рутинні завдання, створювати власні інструменти та розширювати функціональність програми [12]. Інструменти для симуляції фізики та процедурного моделювання дозволяють створювати реалістичні ефекти, такі як вибухи, руйнування, рідини та тканини. Завдяки своїй популярності, Maya та 3ds Max мають велику спільноту користувачів та безліч навчальних матеріалів.

Для узагальнення всіх досліджуваних програм було складено порівняльну таблицю (табл. 1.2) [8]-[12].

Таблиця 1.2 – Узагальнена таблиця досліджуваних програм [8]-[12]

Критерії	Photoshop	GIMP	Blender	Maya	3ds Max
1	2	3	4	5	6
Розробник	Adobe	GNU community	Blender Foundation	Autodesk	Autodesk

Кінець таблиці 1.2

1	2	3	4	5	6
Призна-чення	Створення растрової графіки	Створення растрової графіки	3D-моделювання, анімування, скульптинг, рендеринг	3D-моделювання, рендеринг	3D-моделювання, архітектура, рендеринг
Підписка	Платна підписка	Безкоштовна, з відкритим вихідним кодом	Безкоштовна з відкритим вихідним кодом	Платна підписка або ліцензія	Платна підписка або ліцензія
Підтримка 3D	Обмежене створення 3D	Немає	Повний пакет 3D	Повний пакет 3D	Повний пакет 3D
Платформа	Windows, macOS	Windows, macOS, Linux	Windows, macOS, Linux	Windows, macOS, Linux	Тільки Windows

1.4 Дослідження внутрішньоігрових магазинів

Внутрішньоігровий магазин є одним із ключових елементів сучасної мобільної гри. На відміну від класичних консольних і ПК-ігор, мобільні проєкти здебільшого поширюються за моделлю free-to-play, при якій користувач отримує доступ до основного контенту безкоштовно. У таких умовах внутрішньоігровий магазин стає центральним елементом економічної моделі гри. Він виконує функцію сполучної ланки між ігровим процесом та системою монетизації, дозволяючи розробникам отримувати дохід без необхідності стягування плати за завантаження програми.

Основним завданням внутрішньоігрових магазинів є те, що користувачі можуть здійснювати покупки в додатках, щоб придбати віртуальні товари, такі як ігрова валюта, поліпшення або нові рівні та функції. Розробники використовують внутрішньоігрові покупки як стратегію монетизації для максимізації доходу від програми та збільшення часу, що проводиться гравцями у грі. Здійснюючи покупки, що покращують ігровий процес, користувачі з більшою ймовірністю продовжать грати і залишаться в грі надовго [13]-[14]. Крім фінансової функції, внутрішньоігровий магазин відіграє важливу роль у формуванні ігрового балансу, управлінні темпом проходження, створення системи довгострокової мотивації гравців.

Таким чином, внутрішньоігровий магазин слід розглядати не як додатковий елемент інтерфейсу, а як повноцінну ігрову підсистему.

1.4.1 Різновиди внутрішньоігрових магазинів

Для проведення дослідження внутрішньоігрових магазинів було обрано такі ігри, як Clash Royale [15], PUBG Mobile[16], World of Tanks [17], Grand Theft Auto 5 [18]. Перелік ігор складається з двох мобільних та двох комп'ютерних ігор. Це необхідно для розширення дослідницької бази.

Під час проведення дослідження було виявлено, що внутрішньоігрові магазини мають дві основні моделі придбання внутрішньоігрових товарів.

Перша модель полягає у придбанні товарів за допомогою внутрішньоігрової валюти, тобто, валюти яку гравець може заробити під час проходження рівнів, виконання завдань у грі, перемоги над іншими гравцям, тощо. Така валюта існує лише всередині певної гри та не може бути використана в реальному житті або інших іграх. Ця модель є рентабельною якщо сама гра не є безкоштовною. Вона надає можливість гравцям прогресувати у процесі проходження гри.

Друга модель базується на використанні реальних грошей для придбання внутрішньоігрових товарів. Зазвичай в іграх присутні декілька

різновидів валют. Вони поділяються на звичайну валюту, яку гравець може легко отримати виконуючи завдання, та донатну валюту, яку зазвичай необхідно придбати за реальні гроші. Але інколи розробники дають можливість гравцям отримати донатну валюту не вкладаючи реальні гроші в гру. Таким чином розробники заохочують гравців проводити більше часу у грі, що допомагає підтримувати популярність проєкту та заробляти гроші.

Порівняльний аналіз внутрішньоігрових магазинів обраних ігор продемонстровано у таблиці 1.3 [15]-[18].

Таблиця 1.3 – Порівняння внутрішньоігрових магазинів в іграх Clash Royale, PUBG Mobile, World of Tanks, Grand Theft Auto 5 [15]-[18]

Гра	Тип монетизації	Основні товари	Платіжна модель	Вплив на баланс у грі
1	2	3	4	5
Clash Royale	F2P з мікротранзакціями	Пакети карт з персонажами, донатна валюта, скрині	Друга модель	Пришвидшення прогресії гравця. Вплив - помірний
PUBG Mobile	F2P з мікротранзакціями	Скіни на зброю, донатна валюта, кейси, BattlePass	Друга модель	Можливість отримання косметичних оновлень. Вплив - відсутній
World of Tanks	F2P з мікротранзакціями	Техніка, донатна валюта	Друга модель	Пришвидшення прогресії гравця, можливість отримання додаткових бонусів. Вплив - високий

Кінець таблиці 1.3

1	2	3	4	5
Grand Theft Auto 5	P2P без мікротранзакцій	Техніка для пересування, зброя, нерухомість	Перша модель	Стимулює прогресію гравця. Вплив - помірний

Відповідно до таблиці можна зробити висновок, що всі внутрішньоігрові магазини у іграх є структурним елементом економіки гри й у деяких з них, їх значним джерелом доходу. Вони можуть допомагати в прогресії гравця, пропонувати різні косметичні предмети, але абсолютно по-різному впливають на ігровий баланс. На сьогоднішній день внутрішньоігрові магазини є не просто додатковою сторінкою в меню, а безпосередньо повноцінною механікою самої гри, наприклад, як це реалізовано в Grand Theft Auto V [18]. У цій грі є внутрішньоігрові сайти які розділені на різні категорії, наприклад автомобілі, літаки, катери, нерухомість. На кожному з цих сайтів гравець має можливість придбати собі потрібні йому товари. Цей приклад демонструє як внутрішньоігровий магазин вбудований у якості механіки самої гри.

1.5 Дослідження різновидів керування в іграх

Кожна гра потребує певного способу керування для взаємодії гравця з основним контентом, механіками та ігровим світом. На сьогоднішній день існує безліч різновидів керування. Основними з них є клавіатура та миша, геймпад, джойстики тощо. Спосіб керування визначає, як дії користувача (жести, натискання, рух) перетворюються на ігрові команди. Якість та тип керування безпосередньо впливають на зручність інтерфейсу користувача, рівень занурення гравця у гру, складність освоєння гри, яке ще називають порогом входження до гри, доступність для різних категорій користувачів

тощо. З розвитком ігрових платформ (ПК, консолі, мобільні пристрої) виникло безліч форм керування, кожна з яких має свої особливості та сфери застосування.

Найпопулярнішим типом керування у світі є клавіатура та миша. Це «золотий стандарт» для ігор, що вимагають високої точності та швидкого реагування, таких як шутери від першої особи (FPS) та стратегії в реальному часі (RTS). Клавіатура відповідає за рух, дії та гарячі клавіші, а миша – за прицілювання та керування камерою. Особливістю цього типу керування є висока точність, швидкий відгук та можливість встановлення великої кількості дій. [19] Прикладами ігор для яких клавіатура та миша є основним типом керування є Counter-Strike, PUBG, Path of Exile, Dota 2, World of Warcraft.

На другому місці за популярністю є геймпади. Спектр їх використання є ширшим ніж у клавіатури та миші, оскільки вони можуть під'єднуватися як до комп'ютерів так і до консолей [19]-[20]. Геймпади мають аналогові стіки завдяки яким користувач може рухатися у грі. Оскільки стіки працюють саме на аналоговому сигналі, це дає можливість користувачу більш точне та плавне керування, що особливо важливо у іграх на кшталт гонок, авіасимуляторів або іграх у яких важлива точність. Основною перевагою геймпадів над клавіатурою та мишею є зручність у використанні. Це пояснюється тим, що геймпад гравець може тримати у руках при цьому перебуваючи у будь-якій позі (сидячи, стоячи, лежачи), на відміну від клавіатури та миші яким потрібно фізичне місце (комп'ютерний стіл, килим для миші). Прикладами ігор для геймпадів є Forza Horizon 5, Snow Runner, Mortal Combat, FIFA, Red Dead Redemption 2.

Керування на мобільних пристроях налічує декілька різновидів, а саме сенсорне керування, керування жестами, керування нахилом (гіроскоп, акселерометр) та керування з допомогою контролерів [21]-[22]. Сенсорне керування є основним типом керування на мобільних пристроях. Він ґрунтується на звичайних дотиках до екрану. Основною перевагою такого

типу керування є те, що він доступний кожному користувачу та не потребує додаткового обладнання. Особливо сенсорне керування підходить для ігор-головоломок, карточних ігор тощо [21]. Керування жестами використовує свайпи, щипки, перетаскування, обертання та інші жести. Такий тип керування інтуїтивно зрозумілий для користувачів що знижує поріг входження до користування додатками та іграми. Зазвичай мобільні додатки та ігри поєднують декілька типів керування оскільки це допомагає розширити спектр команд [22]. Керування нахилом використовує вбудовані датчики мобільного пристрою для визначення нахилу та руху пристрою. Особливістю такого типу керування є те, що воно надає користувачу підвищене відчуття реалізму та більш точний контроль над ігровим світом. Він особливо підходить для складних ігрових механік в яких важлива точність керування, а саме гонки, авіасимулятори тощо [22]. Керування з використанням додаткових пристроїв, контролерів, дозволяє розробникам створювати більш складні та цікаві ігри, але значним мінусом такого підходу є те, що контролери необхідно придбати окремо, що звужує коло користувачів. Тому у таких іграх зазвичай є декілька типів керування щоб нівелювати відсутність фізичного контролера. Контролери надають високий відгук та велику точність керування що робить ігровий досвід набагато цікавішим.

Порівняльний аналіз усіх видів керування продемонстровано у таблиці 1.4 [19]-[22].

Таблиця 1.4 – Порівняльний аналіз видів керування [19]-[22]

Тип керування	Платформа	Точність керування	Зручність керування	Область застосування
1	2	3	4	5
Клавіатура та миша	ПК	Висока	Висока	Шутери, стратегії

Кінець таблиці 1.4

1	2	3	4	5
Геймпад	ПК, ігрові консолі	Дуже висока	Дуже висока	Гонки, симулятори
Сенсорне керування	Мобільний пристрій	Середня	Середня	Головоломки, карточні ігри
Керування жестами	Мобільний пристрій	Середня	Висока	Стратегії, раннери
Керування нахилом	Мобільний пристрій	Висока	Середня	Гонки, симулятори
Керування контролером	Мобільний пристрій	Дуже висока	Висока	Симулятори

Система керування є одним із ключових елементів ігрового дизайну і безпосередньо впливає на досвід користувача. Комп'ютерні ігри переважно використовують фізичні пристрої керування, які забезпечують високу точність та гнучкість. Мобільні ігри орієнтовані на сенсорні та жестові способи керування, що підвищує доступність, але накладає обмеження на складність механік. Під час розробки мобільної гри вибір системи управління повинен враховувати жанр гри, цільову аудиторію, тривалість ігрових сесій та фізичні обмеження мобільних пристроїв.

1.6 Постановка завдань дипломної роботи

Таким чином проведені дослідження показали що на сьогоднішній день існує безліч різних жанрів, найпопулярнішими з яких є екшен ігри, симулятори та нескінченні раннери. Під час проведення аналізу ігор у жанрі нескінченний раннер, було з'ясовано, що кожна гра має свої унікальні механіки, свій стиль, але всі вони керуються за допомогою графічного інтерфейсу.

Саме тому метою роботи є покращення користувацького досвіду з мобільним ігровим застосунком шляхом реалізації матеріального інтерфейсу.

Основними завданнями роботи є:

- дослідження сучасного стану розробки ігрових застосунків та огляд технологій реалізації;
- проєктування апаратно-програмного забезпечення (аналіз вимог, архітектура системи, обґрунтування обраних програмних та апаратних засобів розробки);
- розробка апаратно-програмного забезпечення (опис особливостей реалізації, схема алгоритму функціонування, опис реалізованого програмного та апаратного забезпечення);
- тестування розробленого апаратно-програмного забезпечення (сценарії та результати).

2 ПРОЄКТУВАННЯ АПАРАТНО-ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ МОБІЛЬНОГО ЗАСТОСУНКУ

Проектування апаратно-програмної частини складається з кількох основних етапів, кожен з яких є важливим для створення успішного та якісного продукту. Від самого етапу проектування залежить хід всього проєкту, тому розробники приділяють йому значну кількість часу, розмірковуючи над кожною деталлю. Саме проектування може тривати від декількох місяців, до року в залежності від складності та масштабу проєкту, розміру та досвіду команди, фінансування та використовуваних технологій. Коли проектування апаратно-програмної частини завершено, команда переходить до наступного етапу – її розробки.

2.1 Призначення та вимоги до розроблюваної системи

Опис призначення та загальної концепції гри.

Призначення розроблюваної системи полягає в тому, щоб надати користувачеві можливість керувати літаком за допомогою матеріального інтерфейсу. Це дозволить створити для гравця відчуття реального польоту та взаємодії з навколишнім світом.

Назва гри: «Canyon flight simulator».

Сетинг гри: симулятор польоту на літаку каньйоном у 3D.

Ідея гри: «Canyon flight simulator» – це захопливий симулятор польоту у межах Великого каньйону. Гравці постають у ролі пілота літака який прийняв участь у змаганні на виживання. Головне завдання подолати якомога більше перешкод, аби набрати більшу кількість очок. Джерелом натхнення для створення цієї гри були: захоплива нескінченна гра «Subway Surfers» та досить відома гра від компанії Microsoft «Microsoft Flight Simulator». Ці джерела вдихнули життя в саму ідею створення гри та стали основою для реалізації проєкту.

Жанр гри належить до нескінченного раннера-платформера та позиціонує себе як симулятор керування літаком. Враховуючи ці жанри можна зробити висновок, що гравець не просто спостерігає за перебігом подій у грі, він безпосередньо бере участь у керуванні літаком, досліджуючи світ та намагаючись подолати усі перешкоди. Основний концепт полягає саме у керуванні літаком, аби зацікавити гравця та залучити його до гри.

Сюжетна складова гри як така відсутня, оскільки гра є нескінченною, але є невелика передісторія. Гравець - це досвідчений пілот літака, який раніше працював на міжнародних рейсах. У свій вільний від польотів час він любив займатися екстремальними видами спорту та навіть брав участь у різних змаганнях. На одному з таких змагань він дізнався про екстремальні польоти у Каньйоні, у яких вирішив взяти участь. З цього моменту і починається етап гри.

Функціональні вимоги до системи.

Система керування польотом. Можливість керування польотом літака з використанням матеріального інтерфейсу для більшого занурення у гру.

Під'єднання матеріального інтерфейсу. З'єднання матеріального інтерфейсу з мобільним пристроєм за допомогою технології Bluetooth.

Інтерфейс. Графічний інтерфейс для взаємодії з внутрішніми подіями гри такими як підключення контролера, старт гри, пауза, перезапуск рівня та вихід.

Перешкоди. Наявність різноманітних перешкод для більшого інтересу та складності гри. Реалізація випадкової появи перешкод на шляху гравця для створення ілюзії нескінченного рівня.

Умови програшу. Реалізація програшу гравця при зіткненні з перешкодами.

Нефункціональні вимоги до системи.

Зручність використання. Наявність інтуїтивно зрозумілого інтерфейсу з підказками до використання чи підключення матеріального інтерфейсу.

Зображення. Візуальна складова має виглядати естетично та

відповідати загальній колірній гамі.

Адаптивність. Всі кнопки та текст інтерфейсу повинен бути адаптивним до різної роздільної здатності екранів.

Для аналізу дій користувача було створено діаграму прецедентів (рис. 2.1).

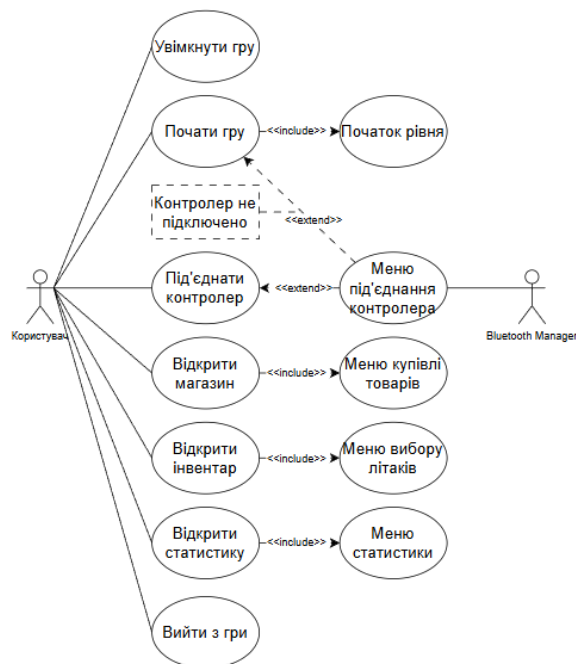


Рисунок 2.1 – Діаграма прецедентів

Для створення взаємодії матеріального інтерфейсу з ігровим середовищем була продумана архітектура розроблюваної системи (рис. 2.2).

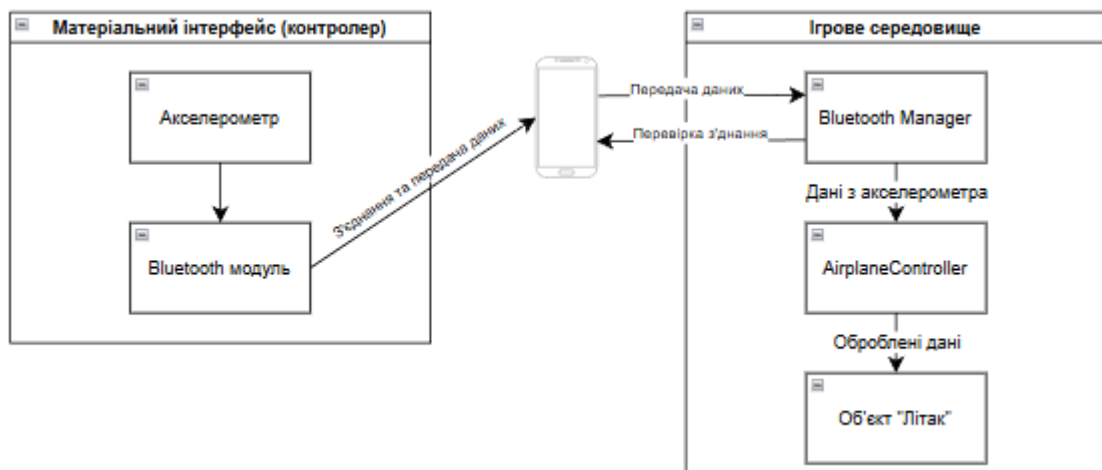


Рисунок 2.2 – Архітектура розроблюваної системи

2.2 Вибір технологій для реалізації гри

Вибір технологій є надзвичайно важливим етапом, оскільки він впливає на різні аспекти проєкту. Від цього залежить продуктивність проєкту, час розробки, вартість, масштабованість і майбутні оновлення. Тому для вибору технологій був проведений порівняльний аналіз за різними критеріями.

2.2.1 Вибір стеку технологій для програмної реалізації гри

Продуктивність та якість проєкту впливають на те, наскільки плавно та швидко він працюватиме на різних пристроях. Це передбачає вибір рушія (наприклад, Unity, Unreal Engine) і мови програмування (наприклад, C++, C#). Технології для реалізації графіки та ефектів визначають графічні можливості, ефекти, фізику, анімацію та інші візуальні характеристики проєкту.

Важливо також враховувати, на яких платформах буде інсталиватися гра: ПК, мобільні пристрої, VR тощо. Тому для порівняльного аналізу було обрано рушії Unity, Unreal Engine та Godot.

Unity – це один з найпопулярніших ігрових двигунів, який до вподоби великим компаніям. Але на сьогоднішній день він дуже сильно припав до душі саме інді-розробникам через його простоту та можливості які він пропонує. Він підтримує широкий спектр різних платформ, таких як ПК, консолі, мобільні пристрої та VR-гарнітура. Окрім вбудованих інструментів, які ідуть разом з двигуном, до нього існує багато різних доповнень, які можна встановити не виходячи з самого Unity, що робить його майже автономним та дає можливість не відвертати увагу від створення проєкту [23].

Unreal Engine – станом на сьогодні найновіша версія цього двигуна була розроблена у 2022 році компанією Epic Games та має назву Unreal

Engine 5. Цей двигун дивує своїми потужними графічними можливостями. В першу чергу Unreal Engine звісно використовується для створення ігор, а саме AAA проєктів. Але окрім його основного призначення, двигун також використовують у кіноіндустрії. Саме на Unreal Engine 4 переважно створювались задні фони для фільмів студії Marvel, в нових «Зоряних війнах» та у серіалі Мандалорець [24]. Сам двигун має можливість створювати як 2D, так і 3D проєкти, але все ж таки він більше налаштований для 3D.

Godot – це двигун з відкритим вихідним кодом, що робить його абсолютно безкоштовним для використання. Godot з'явився досить нещодавно у 2014 році, коли досяг своєї першої стабільної версії. Сутність двигуна досить проста – бути якомога простішим та самодостатнім середовищем розробки ігор. Саме тому він вважається одним з найлегших для вивчення двигунів розробки ігор. Його викладають дітям у школах та у комп'ютерних академіях аби зацікавити їх та залучити до ІТ сфери. Двигун підтримує 2D та 3D розробку ігор та має зрозумілу систему скриптів. Спільнота Godot активно допомагає розробникам розвивати двигун, тому він має регулярні оновлення [25].

Для узагальнення всіх даних було зроблено таблицю порівняльного аналізу (табл. 2.1) [23]-[25].

Таблиця 2.1 – Таблиця порівняльного аналізу ігрових рушіїв [23]-[25]

Критерії	Unity	Unreal Engine	Godot
1	2	3	4
Розробник	Unity Technologies	Epic Games	The Godot Foundation
Підписка	Безкоштовна для індивідуального використання	Безкоштовна до \$1 млн. прибутку	Безкоштовна

Кінець таблиці 2.1

1	2	3	4
Програмування	UnityScript, C#, Boo	C++, Blueprints	CDScripts, C++, C#, Visual Scripting
Призначення	Створення 2D та 3D ігор	Створення 3D ігор та анімаційних мультфільмів	Створення 2D та 3D ігор
Платформи	Windows, macOS, Linux	Windows, macOS, Linux	Windows, macOS, Linux
Marketplace	Unity Asset Store	Unreal Marketplace	Godot Asset Library

Крім можливостей ігрових рушіїв також важливим критерієм є знання та вміння роботи з ними розробника. Цей критерій був найважливішим при виборі ігрового рушія. Роблячи висновок з таблиці порівняльного аналізу та знань розробника, для реалізації гри був обраний ігровий рушій Unity, оскільки він відповідає вимогам проєкту та вмінням розробника.

Наступним етапом був вибір мови програмування. Оскільки Unity це рушій, який для програмування логіки ігор використовує мову C#, вибір був очевидним.

C# – це сучасна об'єктно-орієнтована мова програмування, розроблена компанією Microsoft як частина платформи .NET. Вона була створена у 2002 році і з того часу стала однією з найпопулярніших мов програмування в світі [26]. Саме ця мова програмування використовується як основний інструмент для створення складних ігрових механік та систем у рушії Unity.

2.2.2 Вибір стеку технологій для програмної та апаратної реалізації матеріального інтерфейсу

Вибір стеку технологій для розробки матеріального інтерфейсу був дещо складнішим, оскільки існує безліч різних аналогів мікроконтролерів. Тому був обраний мікроконтролер Arduino, оскільки він має прийнятну ціну, відкритий вихідний код і безліч матеріалів для навчання.

Arduino – апаратна обчислювальна платформа для конструювання, основними компонентами якої є плата мікроконтролера з елементами вводу/виводу та середовище розробки Processing/Wiring на мові програмування, що є спрощеною підмножиною C/C++ [27]. Arduino може використовуватися як для створення автономних інтерактивних об'єктів, так і підключатися до програмного забезпечення, яке виконується на комп'ютері.

Інформація про плату (рисунок друкованої плати, специфікації елементів, програмне забезпечення) знаходяться у відкритому доступі і можуть бути використані тими, хто воліє створювати плати власноруч.

Плата Arduino складається з мікроконтролера Atmel AVR, а також елементів обв'язки для програмування та інтеграції з іншими пристроями. На багатьох платах наявний лінійний стабілізатор напруги +5В або +3,3В. Тактування здійснюється на частоті 16 або 8 МГц кварцовим резонатором. У мікроконтролер записаний завантажувач (bootloader), тому зовнішній програматор не потрібен [27].

Для порівняльного аналізу було обрано три найпопулярніші плати Arduino, а саме Arduino Mega, Arduino Uno та Arduino Nano.

Arduino Mega – це пристрій на основі мікроконтролера ATmega2560 (datasheet). До його складу входить все необхідне для зручної роботи з мікроконтролером: 54 цифрових входу/виходу (з яких 15 можуть використовуватися як ШІМ-виходи), 16 аналогових входів, 4 UART (апаратних приймачів для реалізації послідовних інтерфейсів), кварцовий резонатор на 16 МГц, скидання. Для початку роботи з пристроєм досить

просто подати живлення від AC/DC-адаптера або батарейки або підключити його до комп'ютера за допомогою USB-кабелю. Arduino Mega сумісний з більшістю плат розширення, розроблених для Arduino Duemilanove та Diecimila [28].

Arduino Uno – це пристрій на основі мікроконтролера ATmega328 (datasheet). До його складу входить все необхідне для зручної роботи з мікроконтролером: 14 цифрових входів/виходів (з них 6 можуть використовуватися як ШИМ-виходи), 6 аналогових входів, кварцовий резонатор на 16 МГц, роз'єм USB, роз'єм живлення, роз'єм для внутрішньосхемного програмування (ICSP) та кнопка скидання. Для початку роботи з пристроєм досить просто подати живлення від AC/DC-адаптера або батарейки, або підключити його до комп'ютера за допомогою кабелю USB [29].

Arduino Nano – це повнофункціональний мініатюрний пристрій на базі мікроконтролера ATmega328 (Arduino Nano 3.0) або ATmega168 (Arduino Nano 2.x), адаптований для використання з макетними платами. За функціональністю пристрій схожий на Arduino Duemilanove і відрізняється від нього розмірами, відсутністю роз'єму живлення, а також іншим типом (Mini-B) USB-кабелю. Arduino Nano розроблено та випускається фірмою Gravitech [30].

Проаналізувавши три види мікроконтролерів Arduino був обраний Arduino Nano через його компактність (табл. 2.2) [28]-[30].

Таблиця 2.2 – Порівняльний аналіз мікроконтролерів Arduino [28]-[30]

Критерії	Arduino Mega	Arduino Uno	Arduino Nano
1	2	3	4
Мікроконтролер	ATmega2560	ATmega328P	ATmega328P
Розмір	Велика (101.6 x 53.4 мм)	Середня (68.6 x 53.4 мм)	Мала (45 x 18 мм)
USB-інтерфейс	Type-B	Type-B	Mini-B

Кінець таблиці 2.2

1	2	3	4
Кількість цифрових пінів	54, з них 15 – PWM	14, з них 6 – PWM	14, з них 6 – PWM
Кількість аналогових пінів	16	6	8
Особливості	Велика кількість пінів. Використовується для великих проєктів	Універсальна плата. Має збалансовану кількість пінів до розміру	Дуже компактна. Підходить для проєктів, де розмір має значення

В якості середовища розробки компанія Arduino Software надає програмне забезпечення Arduino IDE. Саме воно і було обрано як основне середовище розробки через свої функціональні можливості і налаштованість до роботи з Arduino.

Інтегроване середовище розробки Arduino – або програмне забезпечення Arduino (IDE) – розроблене компанією Arduino Software та призначене для програмування плат Arduino. Програма містить текстовий редактор для написання коду, область повідомлень, текстову консоль, панель інструментів з кнопками для виконання поширених функцій та низку меню. Воно підключається до обладнання Arduino за допомогою кабелю для завантаження програм та взаємодії з ними [31]. Для програмування плат Arduino використовується мова програмування C++.

Середовище розробки Arduino поставляється разом із бібліотекою програм «Wiring».

Користувачам необхідно визначити лише дві функції для того, щоб створити програму, яка буде працювати за принципом циклічного виконання:

- `setup()`: функція виконується лише раз при старті програми і дозволяє задати початкові параметри;

– loop(): функція виконується періодично, доки плата не буде вимкнена.

2.3 Висновки до розділу 2

Під час проектування апаратно-програмного забезпечення мобільного застосунку були визначені вимоги до розроблюваної системи, створено діаграму прецедентів для аналізу дій користувача, спроектовано архітектуру взаємодії матеріального інтерфейсу з ігровим середовищем, також було проведено порівняльний аналіз ігрових рушіїв для порівняння їх можливостей.

Як ігровий рушій був обраний Unity, оскільки він відповідає вимогам проєкту та вмінням розробника.

Основною мовою програмування була обрана мова C#, оскільки вона використовується для програмування ігрової логіки в самій рушії Unity.

Для реалізації матеріального інтерфейсу були проаналізовані мікроконтролери Arduino, з яких був обраний мікроконтролер Arduino Nano через свою компактність. В якості середовища розробки було обрано Arduino IDE через свої функціональні можливості та пристосованість до роботи з мікроконтролерами Arduino.

3 РОЗРОБКА ТА РЕАЛІЗАЦІЯ АПАРАТНО-ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка та реалізація є центральною фазою при створенні апаратно-програмного забезпечення, під час якої фактичне створення гри та контролера здійснюється на основі концепцій, планів і прототипів, розроблених на попередніх фазах. Важливою частиною цього етапу є гнучкість і готовність до змін, оскільки в процесі розробки можуть виникати нові ідеї, проблеми та можливості покращити проєкт. Цей етап не вважається останнім, тому що після завершення реалізації починається етап QA-тестування. На цьому етапі відбувається пошук помилок та багів, які не були виявлені при розробці, перевірка основних механік та функцій гри, аналіз працездатності контролера, після чого обов'язково вносяться зміни для їх виправлення.

3.1 Проєктування основних алгоритмів гри за допомогою діаграм

Алгоритми являються ключовою складовою кожного проєкту. Вони використовуються абсолютно у кожній сфері роботи та в повсякденному житті.

Одним з найвідоміших алгоритмів є алгоритм приготування чаю. Він складається з дуже простих кроків, які можуть повторити навіть діти. Але це лише один з видів алгоритмів. Насправді існує два види алгоритмів:

- суворий алгоритм;
- не суворий алгоритм.

Сенс алгоритму полягає у суворому дотриманні всіх правил для виконання задачі. Це певний список, який описує порядок дій одна за одною. Програма яка слідує одному алгоритму повинна видавати один і той самий результат.

При розробці проєкту з'явилась необхідність у створенні алгоритму

дій гравця, які він буде виконувати одразу після відкриття гри, для аналізу, пошуку та виправлення всіх розбіжностей та додавання відсутніх кнопок взаємодії. На рисунках нижче (рис. 3.1 – 3.4) продемонстровано розроблений алгоритм взаємодії гравця з грою.

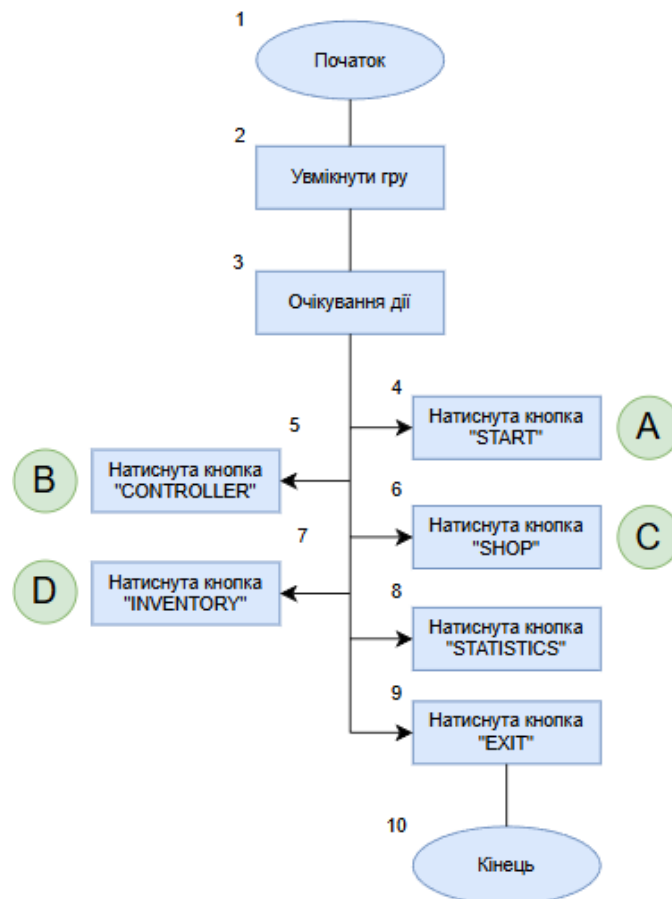


Рисунок 3.1 – Узагальнений алгоритм взаємодії гравця з грою

Блок 1. Початок алгоритму.

Блок 2. Користувач вмикає гру натискаючи на її іконку на головному екрані мобільного телефону.

Блок 3. Гра очікує взаємодії користувача.

Блок 4. При натисканні кнопки «START» починається виконання алгоритму А.

Блок 5. При натисканні кнопки «CONTROLLER» починається виконання алгоритму В.

Блок 6. При натисканні кнопки «SHOP» починається виконання алгоритму С.

Блок 7. При натисканні кнопки «INVENTORY» починається виконання алгоритму D.

Блок 8. При натисканні кнопки «STATISTICS» починається виконання алгоритму E.

Блок 9. При натисканні кнопки «EXIT» гра завершує свою роботу вимикаючи з'єднання з контролером.

Блок 10. Кінець алгоритму.

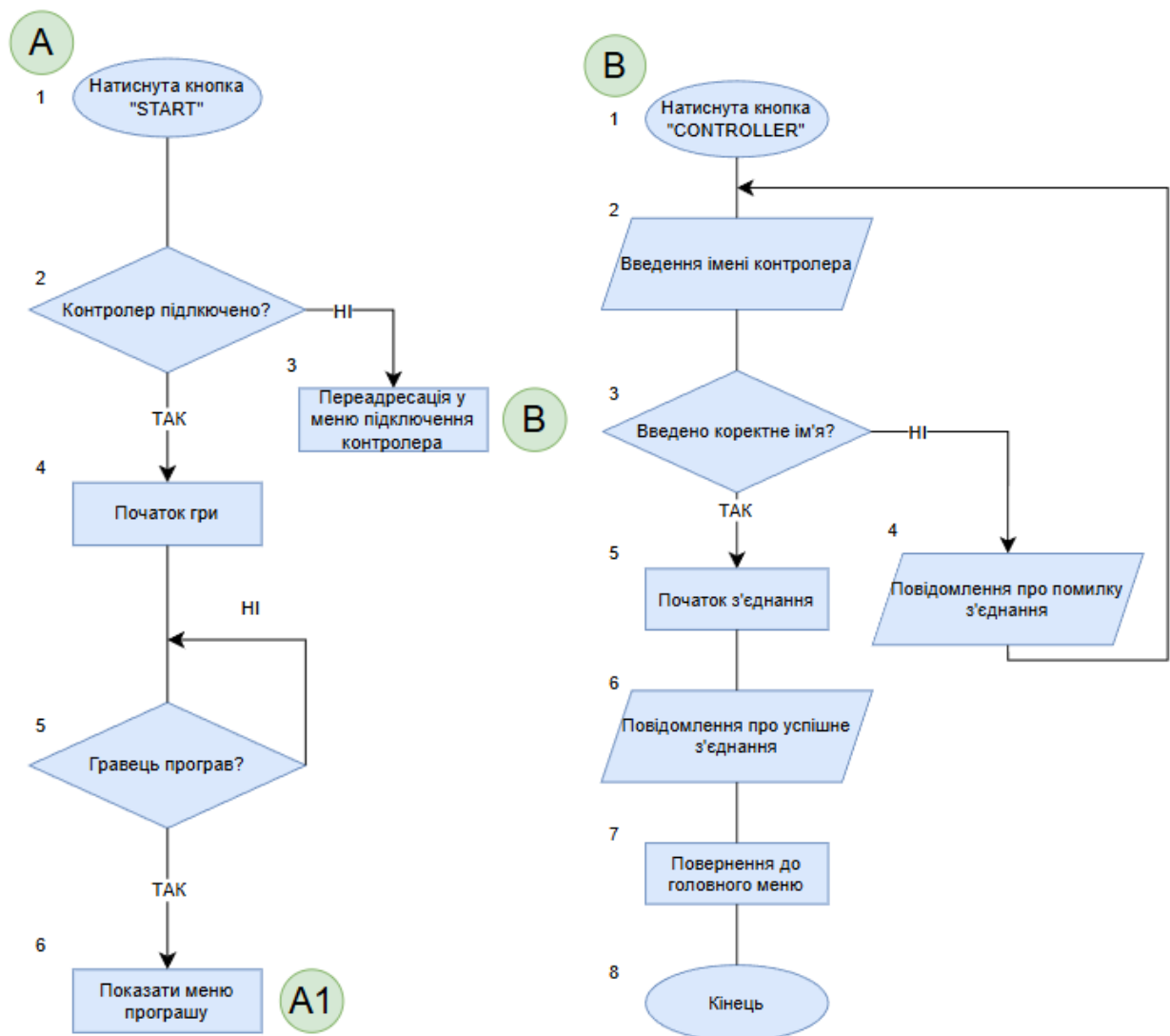


Рисунок 3.2 – Алгоритми початку гри (А) та підключення контролера (В)

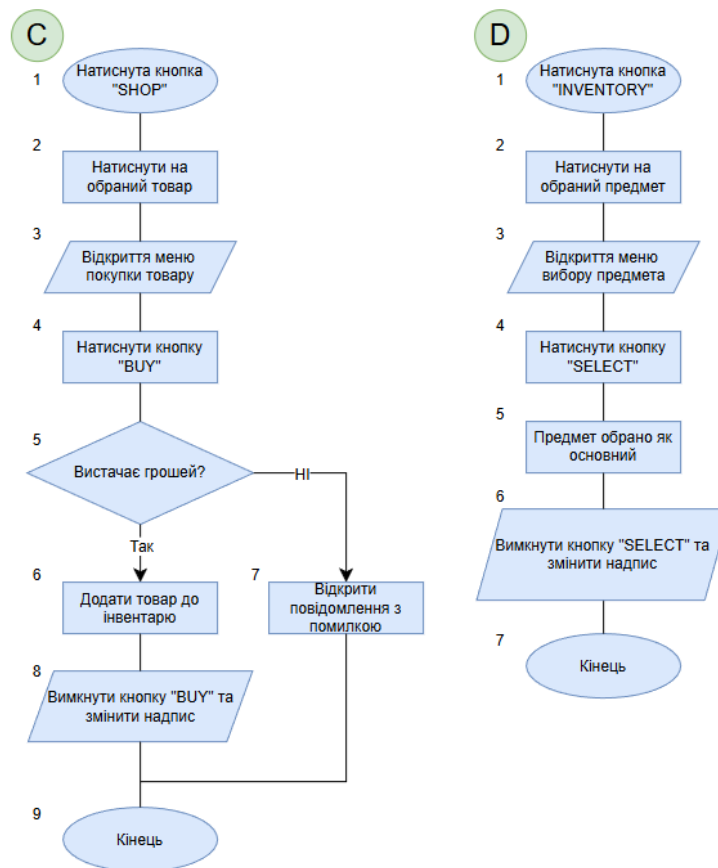


Рисунок 3.3 – Алгоритми покупки товарів (C) та вибору зовнішнього вигляду (D)

Опис алгоритму А наведено нижче.

Блок 1. Початок виконання алгоритму якщо кнопка «START» була натиснута.

Блок 2. Виконується перевірка підключення контролера.

Блок 3. Якщо контролер не був підключений, відкривається меню підключення контролера.

Блок 4. Якщо контролер був підключений, запускається основний рівень гри.

Блок 5. Перевірка програшу гравця. Якщо гравець не програв, гра продовжується.

Блок 6. Якщо гравець програв, гра припиняється, відкривається меню програшу та виконується алгоритм А1 (рис. 3.4).

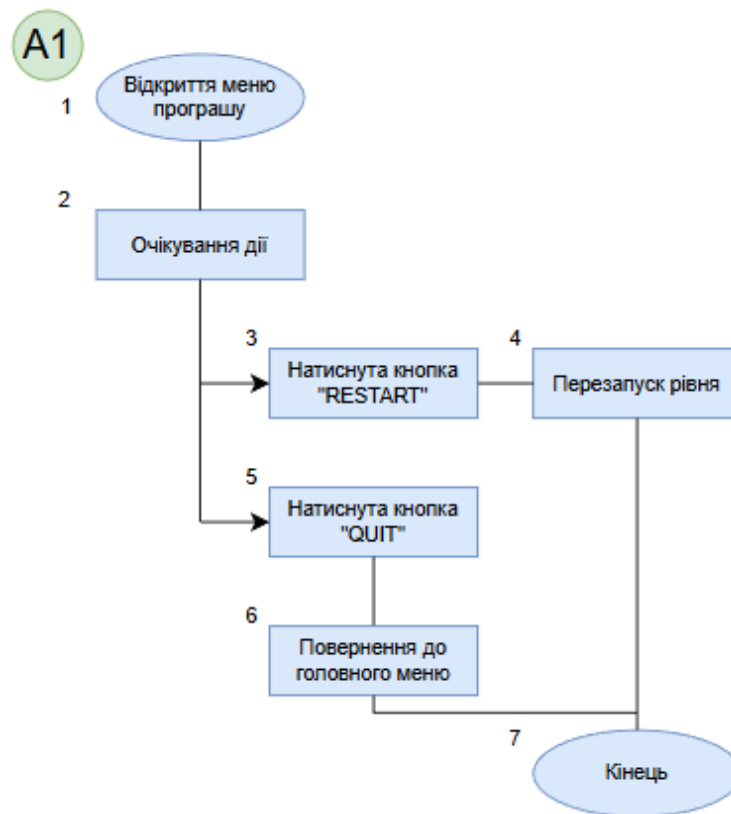


Рисунок 3.4 – Алгоритм А1 (взаємодія з меню програшу)

Опис алгоритму В наведено нижче.

Блок 1. Початок виконання алгоритму якщо кнопка «CONTROLLER» була натиснута.

Блок 2. Очікування введення назви контролера.

Блок 3. Виконується обробка введених даних.

Блок 4. Якщо назва контролера була введена неправильно або надіслано порожній рядок, виводиться повідомлення про помилку з'єднання і програма переходить в очікування введення імені контролера.

Блок 5. Якщо назву контролера було введено правильно, починається налаштування з'єднання.

Блок 6. Виводиться повідомлення про успішне з'єднання з контролером.

Блок 7. Повернення на головний екран гри.

Блок 8. Кінець алгоритму.

Опис алгоритму С наведено нижче.

Блок 1. Початок виконання алгоритму якщо кнопка «SHOP» була натиснута.

Блок 2. Натиснути на вподобаний товару.

Блок 3. Відкриття меню покупки товару.

Блок 4. Очікування покупки товару.

Блок 5. Перевірка кількості ігрових валютних одиниць.

Блок 6. Якщо грошей вистачає, додає товар до інвентарю.

Блок 7. Якщо грошей не вистачає, виводить відповідне повідомлення про нестачу грошей.

Блок 8. Змінити статус кнопки «BUY» на неактивний та змінює назву кнопки на «BOUGHT».

Блок 9. Кінець алгоритму.

Опис алгоритму D наведено нижче.

Блок 1. Початок виконання алгоритму якщо кнопка «INVENTORY» була натиснута.

Блок 2. Натиснути на вподобаний візуальний стиль.

Блок 3. Відкриття меню обраного візуального стилю.

Блок 4. Очікування натискання на кнопку «SELECT».

Блок 5. Змінити поточний візуальний стиль на обраний.

Блок 6. Змінити статус кнопки «SELECT» на неактивний та змінює назву кнопки на «SELECTED».

Блок 7. Кінець алгоритму.

Опис алгоритму A1 наведено нижче.

Блок 1. Початок виконання алгоритму. Відкриття меню програшу.

Блок 2. Очікування взаємодії гравця з меню.

Блок 3. Перевірка натискання кнопки «RESTART».

Блок 4. Перезавантаження рівня та вимкнення меню програшу.

Блок 5. Перевірка натискання кнопки «QUIT».

Блок 6. Зміна сцени. Повернення до головного меню.

Блок 7. Кінець алгоритму.

Алгоритм взаємодії гравця з грою демонструє кожен крок гравця, який він може зробити. Після увімкнення гри перед гравцем з'являється головне меню з вибором, почати проходження рівня, відкрити магазин, відкрити інвентар, відкрити меню статистики, відкрити меню під'єднання контролера та вийти з гри. Кнопка «Start» відповідає за початок гри, кнопка «Shop» за відкриття магазину, кнопка «Inventory» за відкриття інвентарю, кнопка «Statistics» за відкриття меню статистики, кнопка «Controller» за меню під'єднання контролера, кнопка «Exit» відповідно за вихід з гри. Після початку гри гравець починає проходити рівень, а гра очікує поки гравець дійде до кінця рівня, або не програє. В залежності від результату на екрані перед гравцем з'явиться привітання з успішним проходженням рівня з пропозицією перейти до наступного рівня, або повідомлення про програш з можливістю перезавантажити рівень.

Крім цього гравець має можливість посеред проходження рівня поставити гру на паузу. Алгоритм взаємодії з меню паузи продемонстровано на рис. 3.5.

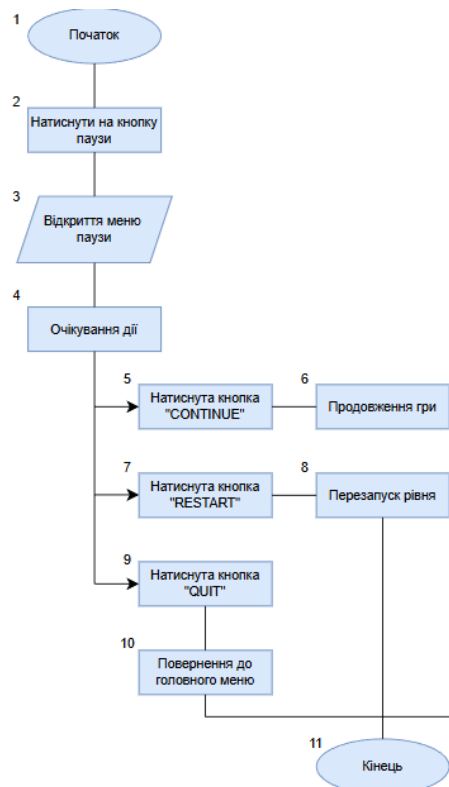


Рисунок 3.5 – Алгоритм взаємодії з меню паузи

Блок 1. Початок алгоритму.

Блок 2. Користувач натискає кнопку паузи.

Блок 3. Час у грі зупиняється. Перед гравцем з'являється меню паузи.

Блок 4. Очікування взаємодії гравця з меню паузи.

Блок 5. Перевірка натискання кнопки «CONTINUE».

Блок 6. Гра відновлює час. Меню паузи зникає. Гра продовжується.

Блок 7. Перевірка натискання кнопки «RESTART».

Блок 8. Перезавантаження рівня та вимкнення меню паузи.

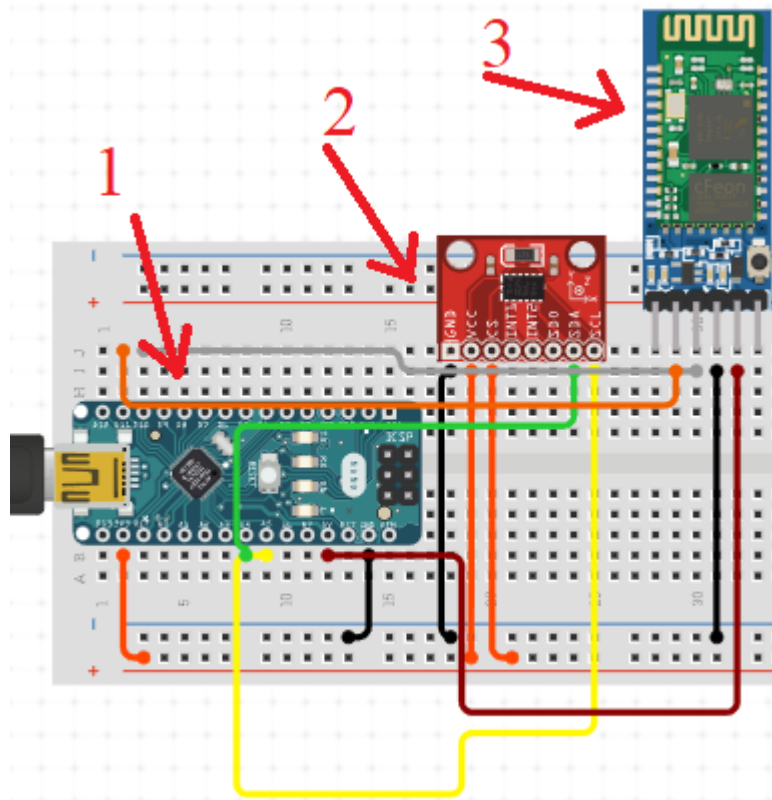
Блок 9. Перевірка натискання кнопки «QUIT».

Блок 10. Зміна сцени. Повернення до головного меню.

Блок 11. Кінець алгоритму.

3.2 Проєктування ігрового контролера на базі Arduino

Одним з завдань дипломного проєкту було створення апаратної частини для взаємодії з ігровим середовищем. Саме тому було створено ігровий контролер на базі мікроконтролера Arduino. У грі контролер відіграє роль штурвалу літака. Для реалізації управління літаком, його рухом у сторони та нахилом передбачається використовувати акселерометр модулі ADXL345. Акселерометр це датчик, який вимірює прискорення об'єкта у просторі. Для підключення контролера до мобільного пристрою буде використовуватися модуль Bluetooth Serial Module HC-05, оскільки провідне підключення є незручним для більшості користувачів. Схема макету контролера (рис. 3.6) продемонстровано на малюнку нижче. Макет було розроблено на онлайн-ресурсі «Circuito.io» [32].



1 – Плата Arduino Nano; 2 – Акселерометр ADXL345; 3 – Bluetooth Serial Module HC – 05

Рисунок 3.6 – Макет контролера

Як тільки макет контролера був змодельований і перевірена його працездатність в онлайн-сервісі, настає етап відтворення його фізичної форми. Для початку було спаяно акселерометр та модуль Bluetooth з платою Arduino Nano (рис. 3.7).

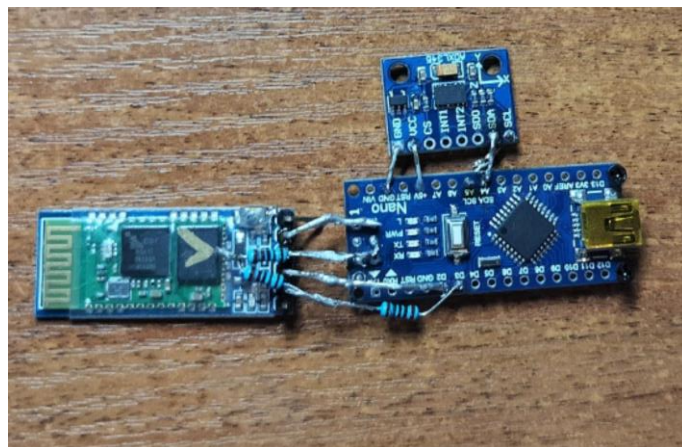


Рисунок 3.7 – Апаратна частина контролера

Також для апаратної частини контролера необхідна ручка до якої вона буде прикріплена (рис. 3.8). Це дозволить гравцеві комфортно керувати літаком у грі. Ручка була надрукована на 3D принтері.



Рисунок 3.8 – Зібрана модель контролера

Після створення та складання контролера настає етап програмування апаратної частини. Основне завдання апаратної частини передавати дані які отримує акселерометр по Bluetooth в рушій Unity. Схема зчитування даних з акселерометра та передача їх по Bluetooth представлена нижче (рис. 3.9).

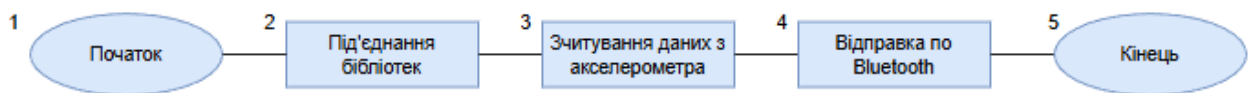


Рисунок 3.9 – Схема зчитування даних з акселерометра та їх передачі по Bluetooth

Блок 1. Початок алгоритму.

Блок 2. Підключення всіх необхідних бібліотек для роботи контролера.

Блок 3. Зчитування даних з акселерометра та запис їх у масив.

Блок 4. Відправка зчитаних даних через модуль Bluetooth.

Блок 5. Кінець алгоритму.

Наступний етап - це реалізація отримання даних у Unity. Оскільки в Unity немає вбудованих плагінів для обміну даними по блютузу, був використаний сторонній плагін який знаходиться у відкритому доступі [33]. Плагін був перероблений і оптимізований під потреби проєкту, оскільки в ньому було безліч зайвих дій, які можуть сильно навантажувати гру. Основна задача плагіна - це запит дозволів від системи Android про підключення блютузу, створення лінії зв'язку, отримання доступу про підключені пристрої та обмін даними. Структура плагіну представлено нижче (рис. 3.10).

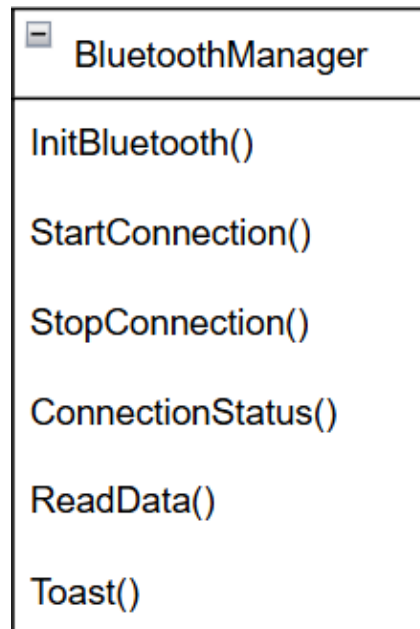


Рисунок 3.10 – Структура плагіну Blueunity

Як тільки з'єднання між пристроями налаштоване автоматично починається передача даних. Передані дані потрапляють у скрипт керування літаком. Щоб ці дані могли взаємодіяти з алгоритмами керування переміщенням і нахилом їх потрібно обробити. Для цього використовується функція парсингу (Додаток А.3), яка оброблює всі дані в коректний формат. Після чого вони передаються в функції алгоритмів.

3.3 Розробка ігрового середовища

Розробка ігрового середовища поділяється на конкретні етапи. Щоб правильно розподілити всі етапи, були проаналізовані стилі та підходи до розробки, на основі чого було складено план етапів розробки ігрового середовища (рис. 3.11).

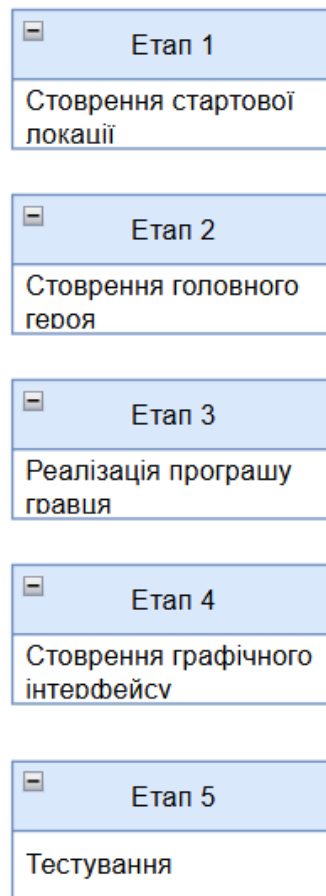


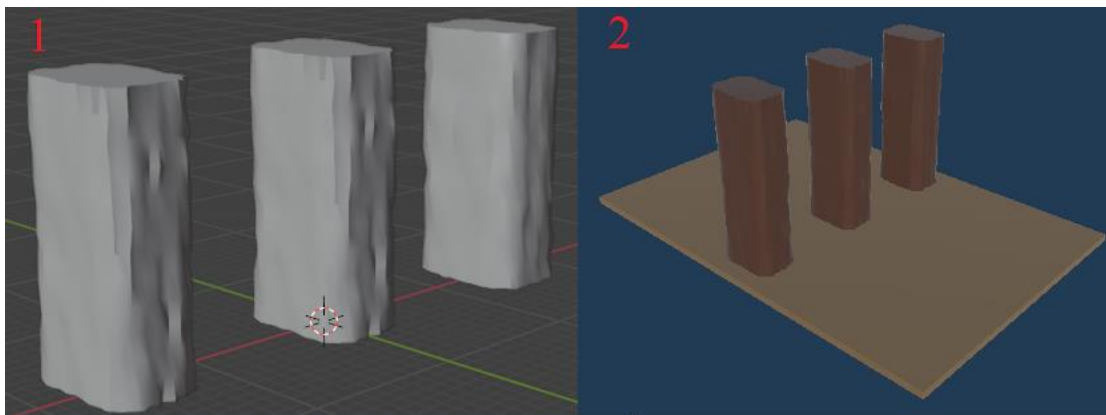
Рисунок 3.11 – Етапи розробки ігрового середовища

3.3.1 Створення стартової локації та генерація платформ

Створення базової локації це фундамент створення гри. З неї починається весь проєкт. Насамперед вона потрібна для того, щоб розробник міг протестувати початкові механіки переміщення персонажа та його взаємодію з навколишнім ігровим світом. Зазвичай базова локація створюється за допомогою простих, вбудованих в двигун фігур, або

заздалегідь підготовлених 3D-моделей. Після чого над нею починають працювати Level-дизайнери, та перетворюють на перший рівень гри, або на навчальний рівень.

Початком створення базової локації є додавання у двигун «Unity» спеціальних префабів з 3D-моделями. Префаб – збірні заготовки зі спрайтів або 3D-моделей, які мають відкориговані налаштування. Для створення базової локації було створено одну 3D модель у програмі для 3D моделювання Blender. Ця модель у грі буде представляти перешкоду яку гравець повинен буде облітати. Модель складається з трьох кам'яних стовпів, які стоять в ряд на рівній відстані один від одного. Після додавання в Unity стовпи були пофарбовані в темний коричневий колір, щоб створити видимість кам'яної споруди, а під ними була розташована платформа завдовжки 60 юнітів (рис. 3.12). Юніт (Англ. Unit) – одиниця виміру відстані або довжини в Unity.

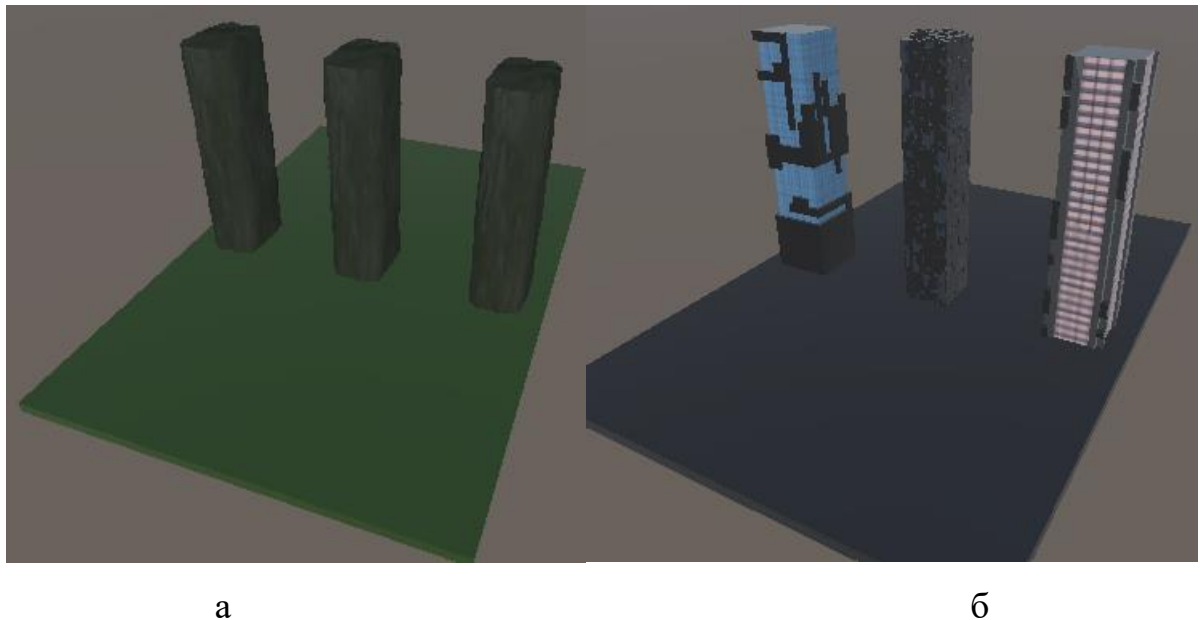


1 – Вигляд в середовищі моделювання Blender; 2 – Вигляд в двигуні Unity

Рисунок 3.12 – Модель з трьох кам'яних стовпів

Після експорту 3D моделі в Unity і додавання до неї всіх необхідних деталей, а саме кольору стовпів та нижньої платформи, було необхідним створити більше варіантів перешкод, щоб гра не виглядала одноманітно. Для цього було створено префаб готової моделі та продубльовано на сцені ще шість разів. Далі у кожному із шести продубльованих префабів було змінено кількість стовпців технікою звичайного комбінування. Комбінування

відбувається коли ми прибираємо один або два стовпи з платформи тим самим отримуючи нову перешкоду. Всього вийшло сім перешкод та одна пуста платформа. Цих перешкод цілком достатньо для того, щоб зробити гру різноманітною. Але все ж таки є одна проблема яка полягає у однотипності з боку візуальної складової. Усі перешкоди повторюються і протягом усієї гри мають однаковий стиль. З часом такий підхід починає набридати гравцю. Тому для вирішення цієї проблеми було додано два додаткових візуальних стилів перешкод (біоми), а саме джунглі та місто (рис. 3.13). Вони змінюються при досягненні певної відстані гравцем.



а – джунглі, б – місто

Рисунок 3.13 – Різновиди візуальних стилів

Після створення перешкод було додано дві стіни, які будуть потрібні для обмеження переміщення гравця, та одночасно візуалізують кам'яні скелі каньйону. Також було додано невидиму стелю над стінами. Вона потрібна для того, щоб гравець не міг вилетіти за межі локації і перелітати перешкоди. Хоча жанр гри нескінченний раннер-платформер, стартова локація повинна все одно бути присутньою. З цієї локації гравець починає свою гру (рис. 3.14).

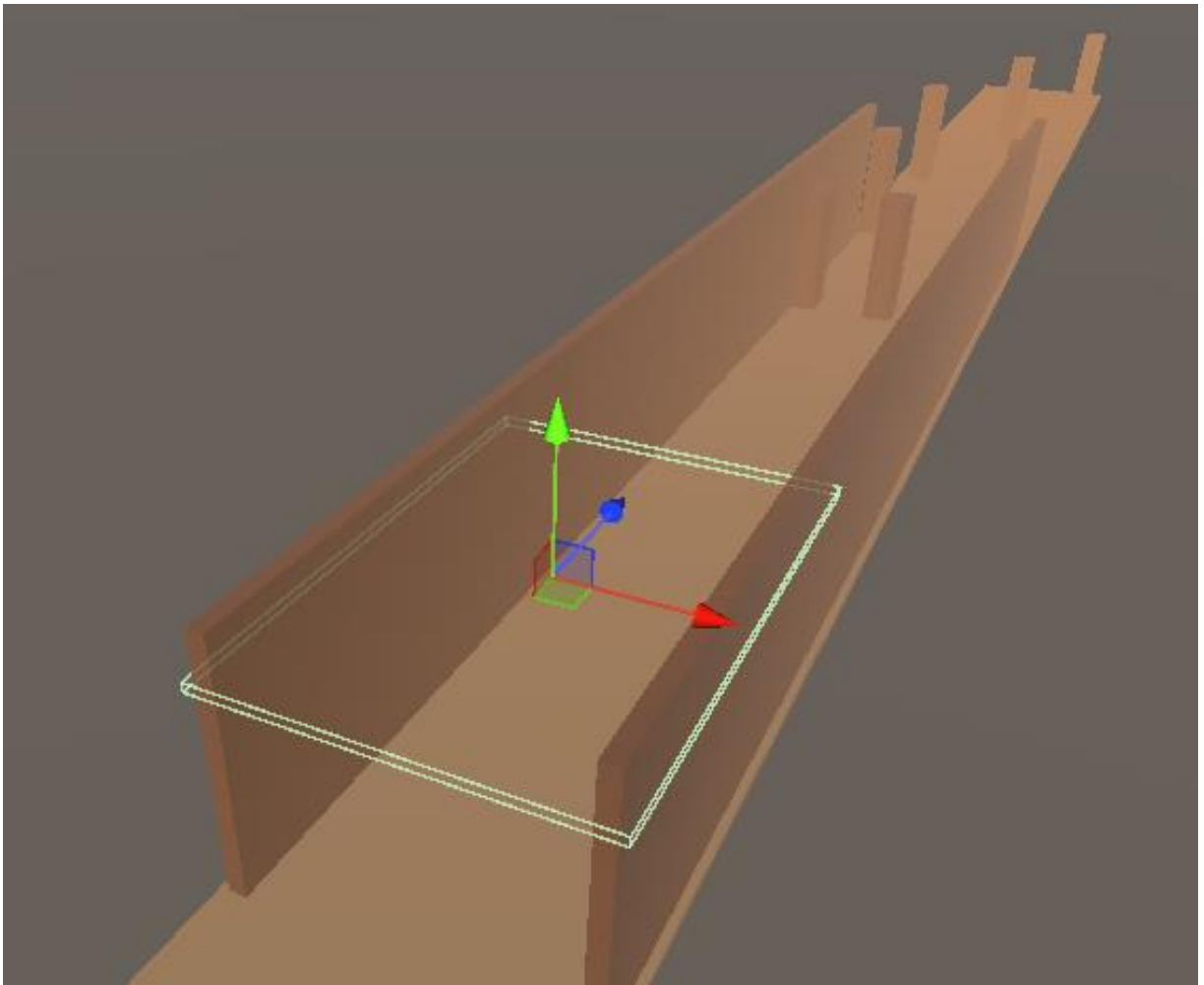


Рисунок 3.14 – Стартова локація, яка завжди присутня у рівні (Невидима стеля виділена зеленим контуром)

На кожен об'єкт, чи то платформа з перешкодами або стіни, був доданий «Box collider». Колайдер (Англ. Collider) - це компонент, який дає фізичні межі об'єктам для визначення зіткнень та взаємодій між ними. «Box collider» має форму прямокутного паралелепіпеда що дозволяє зробити межі для об'єктів кубоподібної форми.

Відтак на базовому рівні представлені стартові платформи з перешкодами, стіни та невидима стеля. Далі по ходу гри локація буде генеруватися автоматично.

Наступний етап – це генерація рівня, тобто створення нових платформ із перешкодами. Для цього на сцену, наприкінці стартової локації, було додано новий порожній об'єкт з «Box collider» (далі спавнер). Він

розміщується на відстані 30 юнітів від останньої платформи. Ця відстань розраховується поділом довжини платформи на два. Після налаштування спавнера до нього додається скрипт (Додаток А.1), який і відповідає за додавання нових платформ. Скрипт працює наступним чином – він має дві змінні, перша змінна приймає об'єкт типу лист у який передаються всі перешкоди, а друга приймає довжину платформи.

У функції `void Start()` у змінну `_obstacle` присвоюється випадкова перешкода. Потім у функції `void Spawn()` визначається позиція на сцені в якій з'явиться нова перешкода.

Для спавна кожної перешкоди має спрацьовувати тригер. Для цього до префабу кожної перешкоди був доданий порожній об'єкт на який був доданий «Box collider» зі значенням тригер. Тригер активується після того, як літак пролітає крізь колайдер. Коли літак активує тригер спрацьовує скрипт (Додаток А.10), який викликає функцію `void Spawn()`, після чого наприкінці локації з'являється нова платформа.

Наступним етапом було реалізувати пересування літака у світі. Було прийнято рішення не переміщати літак вперед, а навпаки переміщати платформи у бік камери, поки літак стоїть на місці. Таким чином створюється ілюзія того, що літак летить прямо. Це було зроблено для того, щоб не навантажувати систему прорахунків подальших координат, оскільки чим далі літак переміщається у просторі, тим більші координати потрібно прораховувати.

Після додавання алгоритму переміщення платформ була виявлена проблема. Об'єкти спавняються нескінченно, що дуже впливає на продуктивність гри. Тобто коли літак пролітає тригер, з'являється нова платформа, вона переміщається в бік камери, але об'єкт не зникає, а продовжує свій рух далі. Це було виправлено простим алгоритмом знищення платформ (Додаток А.2). Як тільки платформа виходить за межі камери вона видаляється. Таким чином кількість об'єктів на сцені є практично незмінною.

3.3.2 Створення персонажа та реалізація скрипта переміщення

Коли базова локація була побудована, прийшов час для додавання головного героя, а саме літака, на сцену та реалізація механіки його переміщення. Переміщення головного героя у просторі можна реалізувати кількома різними способами, залежно від потреб гри та типу контролю. У «Unity» є спеціальний компонент «Rigidbody», який наділяє об'єкт фізичними властивостями та дозволяє підкорятися законам фізики. Це надає можливість реалізувати реалістичне переміщення об'єкта з урахуванням сили гравітації та зіткнень з іншими об'єктами. «Rigidbody» налічує як мінімум три основних параметри, які мають вплив на об'єкти. «Body type» визначає тип фізичного тіла і впливає на те, як це тіло взаємодіє з фізичним середовищем. В «Body type» існує всього три основні типи фізичних тіл.

Dynamic. Використовується для об'єктів, які повинні рухатися та взаємодіяти з іншими фізичними об'єктами. Він залежить від законів сили, гравітації та імпульсу та має масу і здатність до зіткнень.

Kinematic. Застосовується для об'єктів, які повинні рухатися, але не повинні взаємодіяти фізично з іншими об'єктами. Він не підпорядковується гравітації або іншим фізичним силам, а рух об'єкта налаштовується вручну, зміщенням трансформації, але об'єкти можуть взаємодіяти з динамічними об'єктами, штовхаючи їх.

Static. Найчастіше використовується для об'єктів, які не повинні рухатись. Такі об'єкти дуже потрібні при створенні твердих перешкод, наприклад для обмеження області дії гравця або для елементів оточення, таких як стіни, платформи тощо.

Перед тим, як програмувати персонажа, його спочатку потрібно додати на сцену та налаштувати компоненти його префабу. Для створення літака було використано готовий префаб з паку «Simple Airplane Controller» [34] який є у вільному доступі в магазині «Unity Asset Store». Префаб літака складається з двох частин, а саме з пропелера та корпусу (рис. 3.15).

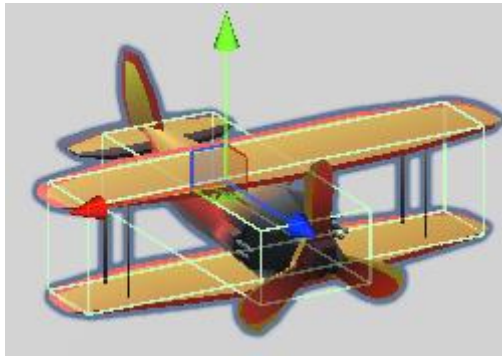


Рисунок 3.15 – Готова модель літака з набору «Simple Airplane Controller»

Для більшої реалістичності на пропелер був доданий скрипт (Додаток А.11) який обертає пропелер навколо своєї осі. Цей скрипт не несе в собі ніяких функціональних можливостей, він використовується тільки як косметична візуалізація.

Оскільки колайдер моделі оточував її повністю, довелося його переробити під формо самого літака. Якщо залишити колайдер, який повністю оточує модель, то реалістичність зіткнень може бути порушена, оскільки літак має специфічну форму. Тому для моделі літака було створено новий порожній об'єкт, на який був накладений «Box collider». Цей об'єкт відповідав за колайдер крил літака. Окрім колайдера на літак був доданий параметр `rigidbody`. Він потрібен для того, щоб літак міг фізично взаємодіяти з оточенням. Після того, як основні налаштування літака були виконані настав етап реалізації алгоритму руху (Додаток А.3). Було розроблено два типи керування, а саме керування за допомогою контролера та керування мобільним пристроєм. Під час керування за допомогою контролера алгоритм приймає дані, які передаються з акселерометра контролера. Якщо користувач не має контролера, то за замовчуванням буде використовуватися керування телефоном який приймає данні безпосередньо з вбудованих датчиків пристрою.

Щоб зробити переміщення літака більш реалістичним був також доданий алгоритм нахилу літака (Додаток А.3). Літак нахиляється в

залежності від напрямку руху. Це робить гру більш живою та наближеною до реальності.

Під час тестування польоту було помічено серйозний баг. Коли літак врізається в стіну або в перешкоду, він починає безконтрольно крутитися. Це пов'язано з компонентом `rigidbody` який прив'язаний до літака, оскільки цей компонент відповідає за фізику, він також керує інерцією через яку літак і починає безконтрольно крутитися при зіткненні. Щоб виправити цей баг, було розроблено алгоритм стабілізації польоту літака. Структура класу керування літаком наведена нижче (рис. 3.16).

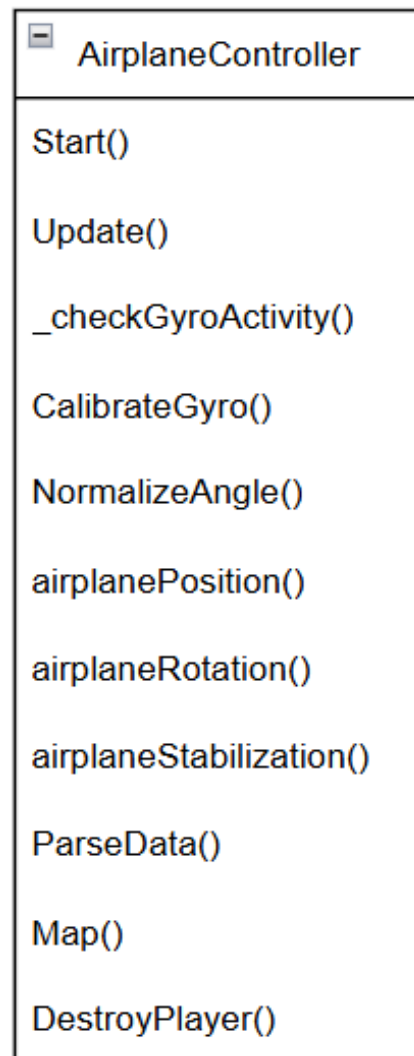


Рисунок 3.16 – Структура класу керування літаком

Момент геймплею продемонстровано нижче (рис. 3.17).



Рисунок 3.17 – Момент геймплею

3.3.3 Реалізація програшу гравця

Рівень гри повинен мати логічне завершення, логічний кінець. Фініш – це об’єкт, який позначає логічне завершення рівня, та є відправною точкою до наступного рівня. Фінішем може бути абсолютно будь-який предмет, чи то двері, прапор, будинок, край карти або просто порожня область в кінці рівня. Кожне досягнення фінішу, це не просто кінець рівня, це можливість отримати нову інформацію про персонажів, світ гри або відкриття сюжетних таємниць. Перед переходом на новий рівень після завершення попереднього, гравець може отримати яку-небудь нагороду, або ресурси, які йому вдалося знайти за час проходження рівня. Також кінець рівня може розблокувати нові рівні, локації чи здібності головного героя, які він зможе застосовувати у подальших рівнях.

Оскільки це нескінченний раннер-платформер у ньому немає як такого фінішу, але є програш. Програш у таких іграх можна вважати своєрідним фінішом у грі, оскільки після цього відбувається перезапуск і весь рівень змінюється.

Програш у грі працює наступним чином. Кожна перешкода має колайдер із певним тегом. Коли колайдер літака стикається з колайдером перешкоди, завдяки компоненту `rigidbody`, у літака вмикається гравітація після чого він починає падати вниз. Як тільки літак досягає підлоги гра зупиняється і відкривається меню програшу. Модель літака має скрипт який реагує на колізію літака з перешкодою (Додаток А.6).

3.4 Створення графічного інтерфейсу

Наявність головного меню, меню паузи та повідомлень про виграш і програш є ключовими елементами інтерфейсу користувача. Вони забезпечують зручність, організованість і задоволення від гри.

Головне меню це як обкладинка книги. Перше, що бачить гравець, коли вперше завантажує гру, це головне меню. Дивлячись на нього, можна одразу зрозуміти який вигляд має і сама гра. Тому головне меню повинно виглядати якомога красивіше та привабливіше.

Головними критеріями якості головного меню є:

- зручність використання;
- інтуїтивно зрозумілий інтерфейс;
- гарне оформлення;
- наявність тільки важливих компонентів.

Якщо дотримуватись цих критеріїв, тоді головне меню буде виглядати дуже привабливо та технічно правильно, що додасть пару балів до оцінки якості меню користувачем.

Етап розробки головного меню починається зі створення нової сцени до якої одразу додається компонент «Canvas».

Створення нової сцени необхідно для того, щоб відділити головне меню від основного контенту гри. У головному меню не потрібні усі елементи та префаби гри, тому нова, чиста сцена є першочерговим етапом створення головного меню.

Одразу після створення «Canvas», до нього був доданий компонент «Panel», на якому і будуть розміщені усі кнопки. До компонента «Panel» можна додати задній фон, який буде відображатись одразу після входу в гру. Задній фон – це звичайна картинка. Для створення всіх меню був використаний ассет «2D casual UI HD» [35]. Як тільки був створений задній фон меню, необхідно створити кнопки, з якими користувач зможе взаємодіяти.

Кнопкам обов'язково необхідно налаштувати текстове поле, змінивши його розмір, колір та шрифт. Також до кнопок було додано фон, щоб вони виглядали красиво та естетично (рис. 3.18). Задній фон головного меню було згенеровано за допомогою штучного інтелекту Chat GPT [36].



Рисунок 3.18 – Вигляд головного меню гри очима гравця

Щоб створити алгоритм, спочатку необхідно додати скрипт до компонента «Canvas». Алгоритм роботи кнопок дуже простий. При натисканні кнопки старт, для користувача завантажиться перший рівень та почнеться гра. У кожній сцені є свій порядок, тому при написанні програми

потрібно вказати номер сцени, яку необхідно завантажити. Порядок сцен корегується у налаштуваннях проєкту (рис. 3.19).

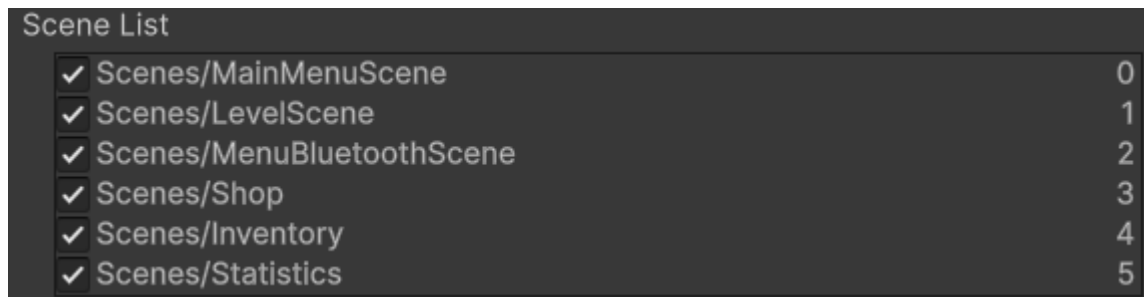


Рисунок 3.19 – Порядок завантаження сцен

Після корегування списку, можна переходити до створення алгоритму. Структура скрипта керування головним меню наведено нижче (рис. 3.20).



Рисунок 3.20 – Структура скрипта, який відповідає за натискання кнопок

Наступним було створено меню магазину (рис. 3.21). У самому меню присутнє поле з кількістю внутрішньої грової валюти, кнопка повернення у головне меню та таблиця з горизонтальною прокруткою у якій знаходяться усі товари для покупки. Кожен товар має свою ціну та картинку, аби користувач міг візуально оцінити зовнішній стиль товару.



Рисунок 3.21 – Меню магазину

При натисканні на картку товару з'являється меню покупки візуального стилю (рис. 3.22). У меню покупки товару відображається картинка обраного товару, його опис, та кнопка «BUY». Після натискання на кнопку «BUY» відбувається одна з двох дій. Якщо гравцю вистачає ігрової валюти, тоді покупка буде виконана успішно, візуальний стиль буде додано до інвентарю, кнопка «BUY» перестане бути активною, а назва кнопки зміниться на «BOUGHT». В іншому випадку якщо гравцю не вистачає валюти, тоді на екрані з'явиться повідомлення для гравця про нестачу коштів на балансі. Усі товари, які були придбані, мають свій унікальний номер який потрібен для ідентифікації та запам'ятовування придбаних товарів.



Рисунок 3.22 – Меню покупки візуального стилю

Після покупки візуального стилю, він потрапляє до інвентарю (рис. 3.23). Інвентар має певну схожість з меню магазину оскільки вони використовують схожі механіки. У меню інвентарю також присутнє поле з балансом гравця, кнопкою повернення до головного меню та картки придбаних стилів. При натисканні на картку відкривається меню у якому гравець може побачити опис стилю, його картинку, назву та кнопку для вибору (рис. 3.24). Якщо гравець натисне на кнопку «SELECT», спрацює алгоритм який відповідає за зміну візуального стилю безпосередньо на сцені ігрового рівня. Після вибору кнопка перестане бути активною, а назва зміниться на selected. Візуальний стиль, який був обраний до цього, знову буде доступний до вибору.



Рисунок 3.23 – Меню інвентарю



Рисунок 3.24 – Меню вибору візуального стилю

Наступним було створено меню статистики (рис. 3.25). Воно несе тільки інформативну функцію для відстеження досягнень гравця за весь час гри. Оскільки більшості людей подобається обмінюватися своїми досягненнями, це меню додає у гру змагального елементу. Таким чином вони проводять більше часу у грі щоб встановити свій власний новий рекорд, або щоб побити рекорд свого друга. Саме меню містить два поля з назвою та числовим значенням для відображення рекорду.

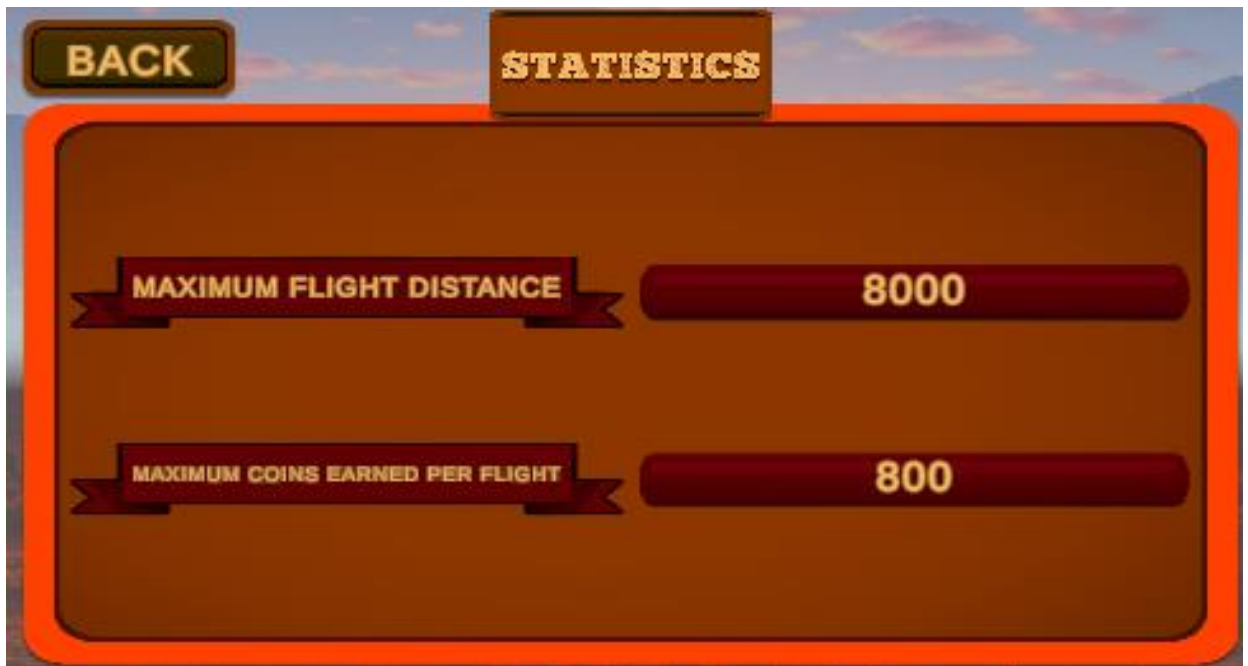


Рисунок 3.25 – Меню статистики

Далі було створено меню підключення контролера (рис. 3.26). В цьому меню присутнє поле для введення імені пристрою, щоб встановити з'єднання з контролером. Також присутня кнопка «Connect» після натискання на яку відбувається підключення. Як тільки підключення завершиться, знизу з'явиться спливаюче вікно, яке підтвердить підключення. В лівому верхньому куті присутня кнопка «Back» які повертає користувача головне меню.



Рисунок 3.26 – Меню підключення контролера

Наступним за рахунком було створено меню паузи. Воно знаходиться безпосередньо на рівнях гри, і відповідає за можливість зупинки гри під час проходження. Оскільки меню не належить до частини геймплею, у компоненті «Canvas» параметр «Renderer Mode» встановлено як «Screen Space – Overlay». Це потрібно для того, щоб у меню паузи була прив’язка до глобальних координат, а не до координат гри та воно завжди відображалось на екрані гравця.

Після переходу з головного меню до рівня гри, гравець одразу бачить локацію та кнопку паузи зліва зверху. Поруч з кнопкою паузи знаходяться інформативні поля з іконкою ліворуч від них. Ці поля необхідні для відображення відстані яку пролетів гравець та кількості зароблених грошей (рис. 3.27). Одразу після натискання на кнопку гра зупиняється, а всі об’єкти завмирають. Перед гравцем з’являється меню паузи у якому є три кнопки. Кнопка «Continue», «Restart» та «Quit». Кнопка «Continue» повертає користувача до гри знімаючи паузу та розморожуючи усі об’єкти. «Restart», за допомогою коду, перезавантажує рівень починаючи його з початку. Кнопка «Quit» повертає гравця у головне меню, де гравець може вийти з гри або почати нову гру (рис. 3.28).



Рисунок 3.27 – Локація рівня, кнопка паузи та інформативні поля



Рисунок 3.28 – Кнопка паузи зліва зверху та меню паузи з кнопками

Після програшу у грі перед гравцем з'являється меню програшу. Меню паузи та програшу дуже схожі між собою, в них є кнопки для взаємодії гравця з грою, але в меню програшу є червоний надпис, який і повідомляє гравця про програш (рис. 3.29).



Рисунок 3.29 – Меню програшу гравця з надписом та двома кнопками

3.5 Реалізація музичного супроводу

Музика та анімація відіграють важливу роль у створенні захопливого ігрового досвіду. Вони забезпечують емоційний зв'язок, занурення в ігровий світ і візуальне задоволення.

Завдяки музичному супроводу у грі створюється певна атмосфера, що відповідає подіям у грі. Гарний саундтрек може повністю змінити враження про гру, та захопити гравців ще більше. За допомогою музики можна керувати динамічністю подій у грі. Чим швидше музика, тим динамічніше відбуваються події у грі, або навпаки спокійна музика спонукає гравця до розслаблення.

При додаванні музичного супроводу використовувався сайт з безкоштовними мелодіями для ігор «OpenGameArt.org» [37]. Щоб зробити музичний супровід, до кожного об'єкту, який повинен мати звук, був доданий компонент «Audio Source». Цей компонент відповідає за налаштування музичної доріжки, зміну гучності, зацикленні тощо. Після чого використання звуку додається до скрипта, який відповідає за певні дії.

До кнопок був доданий короткий звуковий сигнал, аби гравець міг не тільки візуально побачити натискання кнопки, а ще й почути. У всіх меню та на рівні гри був доданий музичний супровід для емоціонального впливу на

гравця. Для цього на сцену був доданий окремий порожній об'єкт до якого було додано компонент «Audio Source» з налаштуваннями (рис. 3.30).

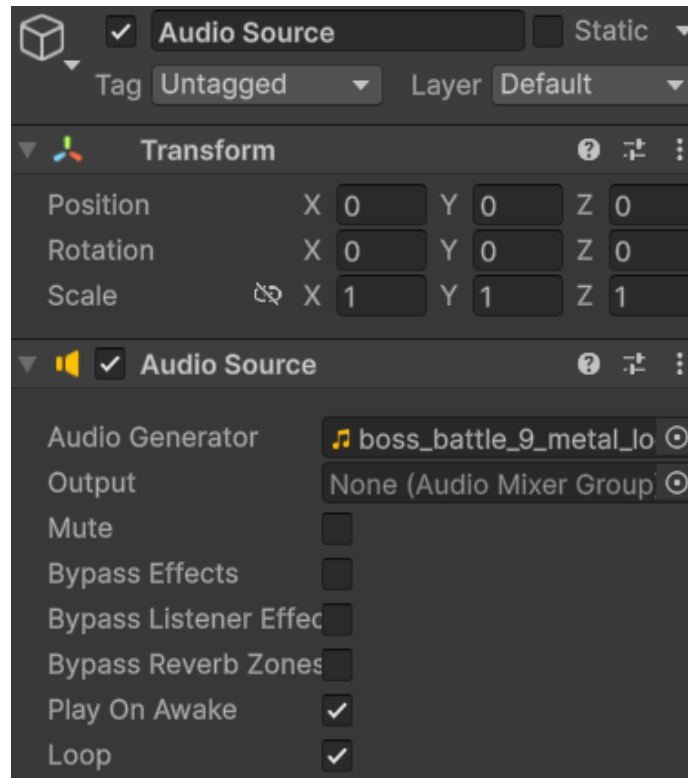


Рисунок 3.30 – Об'єкт Audio Source з компонентом «Audio Source»

3.5 Висновки до розділу 3

Під час реалізації проєкту було розроблено алгоритми взаємодії гравця з грою, спроектовано, зібрано та запрограмовано матеріальний інтерфейс на базі Arduino з використанням модуля Bluetooth та акселерометра. Далі були сплановані етапи розробки ігрового середовища. Було створено стартову локацію, розроблено перешкоди, додано та запрограмовано головного персонажа (літак), реалізовано програш гравця. Останнім було додано графічний інтерфейс із усіма необхідними меню для зручної взаємодії гравця з ігровим середовищем.

4 ТЕСТУВАННЯ ТА ВИКОРИСТАННЯ РОЗРОБЛЕНОГО АПАРАТНО-ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Тестування

Для перевірки працездатності та виявлення багів та недоліків було прийнято рішення провести тестування розробленого програмного забезпечення за п'ятьма сценаріями.

Сценарій 1. З'єднання контролера з телефоном по Bluetooth.

Сценарій 2. Розрив з'єднання з контролером.

Сценарій 3. Робота меню паузи.

Сценарій 4. Програш гравця. Обробка зіткнень із перешкодами.

Сценарій 5. Перевірка підключення контролера до старту гри.

Сценарій 6. Перевірка покупки товару.

Сценарій 7. Перевірка вибору візуального стилю в меню інвентарю.

4.1.1 З'єднання контролера з телефоном по Bluetooth

Мета тесту: успішне з'єднання мобільного пристрою з контролером.

Вхідні дані: назва контролера, яка відображається у вікні Bluetooth мобільного пристрою. Коректні дані для тестування: 00:18:E4:35:0F:00. Некоректні дані для тестування: 00:18:E4:35:0F:0X.

План тестування наведено нижче.

Крок 1. Тестувальник відкриває застосунок на мобільному пристрої з версією Android 8.1 або новіше.

Крок 2. На головному екрані додатку необхідно натиснути велику коричневу кнопку з написом Controller (на англ.) / Контролер (на укр.).

Крок 3. У вікні, що відкрилося, є поле для введення даних, та кнопка для підтвердження відправлення введених даних з написом Connect (на англ.) / З'єднати (на укр.). Поле призначено для введення назви контролера. У поле тестувальник повинен ввести назву контролера, яка вказана у вхідних даних.

Після введення та перевірки введених даних необхідно натиснути на кнопку коричневого кольору з написом Connect (на англ.) / З'єднати (на укр.), яка розташована під полем для вводу даних, тим самим підтвердивши відправлення форми.

Крок 4. Якщо дані були введені правильно, тестувальник повинен побачити повідомлення внизу екрана про успішне з'єднання з текстом «Connection Status: Connected» (на англ.) / «Статус з'єднання: З'єднано» (на укр.). Якщо після відправлення форми з даними для з'єднання, застосунок виводить повідомлення про помилку з текстом «Connection Status: Device not found» (на англ.) / «Статус з'єднання: пристрій не знайдено» (на укр.), необхідно повторити пункт 3 ще раз. Якщо дані були введені правильно, але після відправлення форми застосунок все одно виводить помилку з'єднання, необхідно перевірити вікно Bluetooth на наявність контролера у категорії «Сполучені пристрої», інакше вказати про помилку в звіті про результати тестування. Якщо дані були введені правильно, але після відправлення форми застосунок не виводить жодного повідомлення, необхідно вказати це в звіті про результати тестування. Якщо після відправлення форми з порожніми полями для введення даних, застосунок виводить повідомлення про помилку авторизації з текстом «Connection Status: Device not found» (на англ.) / «Статус з'єднання: пристрій не знайдено» (на укр.), необхідно вказати це в звіті про результати тестування.

Крок 5. Створення звіту результатів тестування.

Умови проходження тестування: застосунок виконує поставлене завдання згідно з метою та планом тестування, а саме успішне з'єднання контролера з мобільним пристроєм після введення правильної назви контролера яка зазначені у вхідних даних.

Результати тестування наведено в табл. 4.1.

Таблиця 4.1 – Результати тестування сценарію 1

Тест	Результат
Введення коректних даних які вказані у вхідних даних.	Після введення коректних даних внизу екрана з'являється повідомлення про успішне з'єднання.
Введення некоректних даних, які вказані у вхідних даних.	Після введення некоректних даних внизу екрана з'являється повідомлення про помилку з'єднання.
Відправлення порожнього поля.	Після надсилання порожнього поля внизу екрана з'являється повідомлення про помилку з'єднання.

За результатами тестів можна зробити висновок, що програма коректно реагує на помилки, які можуть допустити користувачі. Вона активно взаємодіє з користувачем попереджаючи про проблеми або успіх з'єднання.

4.1.2 Розрив з'єднання з контролером

Мета тесту: успішний розрив з'єднання мобільного пристрою з контролером.

Вхідні дані: назва контролера, яка відображається у вікні Bluetooth мобільного пристрою. Коректні дані для тестування: 00:18:E4:35:0F:00.

План тестування наведено нижче.

Крок 1. Тестувальник відкриває застосунок на мобільному пристрої з версією Android 8.1 або новіше.

Крок 2. На головному екрані додатку необхідно натиснути велику коричневу кнопку з написом Controller (на англ.) / Контролер (на укр.).

Крок 3. У вікні, що відкрилося, є поле для введення даних, та кнопка для підтвердження відправлення введених даних з написом Connect (на англ.) / З'єднати (на укр.). Поле призначено для введення назви контролера. У поле тестувальник повинен ввести назву контролера, яка вказана у вхідних даних. Після введення та перевірки введених даних необхідно натиснути на кнопку коричневого кольору з написом Connect (на англ.) / З'єднати (на укр.), яка розташована під полем для вводу даних, тим самим підтвердивши відправлення форми. Якщо дані були введені правильно, тестувальник повинен побачити повідомлення внизу екрана про успішне з'єднання з текстом «Connection Status: Connected» (на англ.) / «Статус з'єднання: З'єднано» (на укр.).

Крок 4. Після натискання на кнопку «Connect» та відображення повідомлення про успішне з'єднання слід вийти з програми. На модулі Bluetooth є світлодіод який блимає червоним кольором. Якщо після виходу з програми світлодіод блимає швидко без пауз, тоді контролер був відключений і з'єднання розірвано. Якщо після виходу з програми світлодіод на модулі Bluetooth блимає по два рази з паузами приблизно в одну дві секунди, то зв'язок між пристроєм і контролером не розірвані. У цьому випадку необхідно вказати про це у звіті про результати тестування.

Крок 5. Створення звіту результатів тестування.

Умови проходження тестування: при виході з програми зв'язок між мобільним пристроєм і контролером повинні бути розірвані, що підтверджує швидко миготливий світлодіод.

Результати тестування наведено в табл. 4.2.

Таблиця 4.2 – Результати тестування сценарію 2

Тест	Результат
Перевірка розриву з'єднання між мобільним пристроєм та контролером.	Після виходу з програми світлодіод на модулі Bluetooth блимає швидко і без пауз, що підтверджує розрив з'єднання між пристроями.

За результатами тестів можна зробити висновок, що з'єднання між мобільним пристроєм і контролером розривається автоматично після виходу з програми та закінчення сесії.

4.1.3 Робота меню паузи

Мета тесту: перевірка роботи меню паузи під час ігрової сесії.

Вхідні дані: увімкнене програмне забезпечення. Коректні дані для тестування: 00:18:E4:35:0F:00.

План тестування наведено нижче.

Крок 1. Тестувальник відкриває застосунок на мобільному пристрої з версією Android 8.1 або новіше.

Крок 2. На головному екрані додатку необхідно натиснути велику коричневу кнопку з написом Controller (на англ.) / Контролер (на укр.) та у поле для введення даних ввести назву контролера. Після чого натиснути кнопку «Connect». Повернутися у головне меню натиснувши кнопку «Back». Почати гру натиснувши кнопку «Start».

Крок 3. Після старту гри у лівому верхньому кутку знаходиться коричнева кнопка паузи. Тестувальник повинен натиснути кнопку паузи. Як тільки кнопка була натиснута має відкритися меню паузи, а гра на задньому фоні має бути зупинена. У вікні паузи є три кнопки. Кнопка «Continue» (на англ.) / «Продовжити» (на укр.) вимикає меню паузи та розморожує гру. Кнопка «Restart» (на англ.) / «Перезапуск» (на укр.) вимикає меню паузи та

перезапускає рівень спочатку. Кнопка «Quit» (на англ.) / «Вийти» повертає у головне меню. Якщо якась із кнопок не виконує описані вище дії про це необхідно вказати у результатах тестування.

Крок 4. Створення звіту результатів тестування.

Умови проходження тестування: після натискання на кнопку паузи має відкритися меню паузи з трьома кнопками, а гра на задньому плані має бути зупинена. Кожна кнопка виконує свою пряму функцію.

Результати тестування наведено в табл. 4.3.

За результатами тестів можна зробити висновок, що програма коректно обробляє події кнопки паузи та меню паузи. Кожна кнопка виконує вказані у плані тестування дії.

Таблиця 4.3 – Результати тестування сценарію 3

Тест	Результат
Натискання на кнопку паузи.	Відкривається меню паузи з трьома кнопками. Гра на задньому плані зупиняється.
Натискання на кнопку «Continue».	Меню паузи вимикається, а гра знімається з паузи.
Натискання на кнопку «Restart».	Меню паузи вимикається, а рівень гри перезапускається спочатку.
Натискання на кнопку «Quit».	Гра повертається до головного меню

4.1.4 Програш гравця. Обробка зіткнень із перешкодами

Мета тесту: обробка події програшу гравця при зіткненні з перешкодами.

Вхідні дані: назва контролера, яка відображається у вікні Bluetooth мобільного пристрою. Коректні дані для тестування: 00:18:E4:35:0F:00. Некоректні дані для тестування: 00:18:E4:35:0F:0X.

План тестування наведено нижче.

Крок 1. Тестувальник відкриває застосунок на мобільному пристрої з версією Android 8.1 або новіше та виконує дії щодо підключення контролера до мобільного пристрою.

Крок 2. Після старту гри тестувальник направляє літак у бік перешкоди.

Крок 3. Після зіткнення з перешкодою літак повинен почати падати вниз. Як тільки він стикається із землею відкривається меню програшу, а гра на задньому фоні зупиняється. Меню програшу виділяє характерний напис «You Lose» (на англ.) / «Ви програли» (на укр.).

Крок 4. Якщо після зіткнення з перешкодою літак продовжує летіти далі і не починає знижуватися про це варто зазначити в результатах тестування. Якщо після зіткнення з перешкодою літак починає падати вниз, але при зіткненні з землею не викликається меню програшу, тоді необхідно вказати про це в результатах тестування.

Крок 5. Створення звіту результатів тестування.

Умови проходження тестування: після зіткнення з перешкодою літак повинен почати падати вниз після чого доторкнутися до землі. При зіткненні викликається меню програшу, а гра на задньому фоні зупиняється.

Результати тестування наведено в табл. 4.4.

Таблиця 4.4 – Результати тестування сценарію 4

Тест	Результат
Зіткнення з перешкодою.	Після зіткнення з перешкодою літак починає падати донизу.
Зіткнення із землею.	Після зіткнення із землею викликається меню програшу, а гра на задньому фоні зупиняється.

За результатами тестів можна зробити висновок, що програма коректно реагує на зіткнення з перешкодами і обробляє умови програшу. При програші викликається меню програшу в якому гравець може перезапустити рівень або вийти в головне меню.

4.1.5 Перевірка підключення контролера до старту гри

Мета тесту: переадресація в меню підключення контролера у випадку, якщо контролер не був підключений до мобільного пристрою.

Вхідні дані: назва контролера, яка відображається у вікні Bluetooth мобільного пристрою. Коректні дані для тестування: 00:18:E4:35:0F:00.

План тестування наведено нижче.

Крок 1. Тестувальник відкриває застосунок на мобільному пристрої з версією Android 8.1 або новіше.

Крок 2. На головному екрані додатку необхідно натиснути велику коричневу кнопку з написом Start (на англ.) / Початок (на укр.).

Крок 3. У випадку, якщо контролер не був підключений до мобільного пристрою, має відкритися вікно підключення до контролера. Після введення у форму вхідних даних та натискання на кнопку «Connect», необхідно повернутися до головного меню за допомогою кнопки «Back». Після цього знову натиснути на кнопку «Start».

Крок 4. При першому заході в гру контролер не буде підключений. Якщо при першому заході в гру була натиснута кнопка «Start» і гра відкрила стартовий рівень, потрібно вказати про це у результатах тестування. Якщо при першому заході в гру була натиснута кнопка «Start» після чого відкрилося меню підключення контролера, були введені всі вхідні дані, але при повторному натисканні на кнопку «Start» гра знову відкриває меню підключення контролера, необхідно перевірити ще раз введені дані. Якщо дані введені правильно але поведінка гри не змінилась, потрібно вказати про це у результатах тестування.

Крок 5. Створення звіту результатів тестування.

Умови проходження тестування: застосунок виконує поставлене завдання згідно з метою та планом тестування, а саме успішна переадресація в меню контролера після натискання на кнопку «Start» при першому заході в гру, якщо контролер не було підключено. Результати тестування наведено в табл. 4.5.

Таблиця 4.5 – Результати тестування сценарію 5

Тест	Результат
Переадресація в меню підключення контролера при натисканні на кнопку «Start», якщо контролер не був підключений.	Кнопка «Start» виконує дію переадресації у меню підключення контролера, якщо він не був підключений.
Запуск ігрового рівня після натискання на кнопку «Start», якщо контролер був підключений.	Кнопка «Start» виконує дію запуску ігрового рівня після підключення контролера.

За результатами тестів можна зробити висновок, що програма коректно оброблює помилку не підключеного контролера. При першому заході в гру відразу після натискання на кнопку «Start» гравець потрапляє в меню підключення контролера. Після введення даних він повертається в головне меню, після чого повторно натискає кнопку «Start», що запускає ігровий рівень у разі успіху підключення контролера.

4.1.6 Перевірка покупки товару

Мета тесту: перевірити працездатність алгоритму придбання товару, а саме його збереження до інвентарю та зміна статусу на «придбаний».

Вхідні дані: увімкнене програмне забезпечення.

План тестування наведено нижче.

Крок 1. Тестувальник відкриває застосунок на мобільному пристрої з версією Android 8.1 або новіше.

Крок 2. На головному екрані додатку необхідно натиснути велику коричневу кнопку з написом Shop (на англ.) / Магазин (на укр.).

Крок 3. У меню магазину необхідно натиснути на картку товару, після чого повинно відкритися меню товару у якому присутнє зображення обраного товару, його ціна та опис.

Крок 4. У відкритому меню товару треба натиснути на кнопку Buy (на англ.) / Купити (на укр.). Якщо після натискання на кнопку не відбулося візуальних змін, а саме кнопка не змінила статус на неактивний та назва кнопки не змінилася на Purchased (на англ.) / Придбано (на укр.), потрібно вказати про це в результатах тестування.

Крок 5. Вийти з меню магазину до головного меню натиснувши кнопку Back (на англ.) / Повернутися (на укр.). У головному меню натиснути кнопку Inventory (на англ.) / Інвентар (на укр.). Після відкриття меню інвентарю повинен з'явитися щойно придбаний товар. Якщо у меню інвентарю немає товару, який був придбаний, необхідно вказати про це у результатах тестування.

Крок 6. Створення звіту результатів тестування.

Умови проходження тестування: застосунок виконує поставлене завдання згідно з метою та планом тестування, а саме успішна покупка товару, зміна назви кнопки Buy (на англ.) / Купити (на укр.) на Purchased (на англ.) / Придбано (на укр.), зміна статусу кнопки на неактивний, у меню інвентарю повинен відображатися придбаний товар. Результати тестування наведено в табл. 4.6.

Таблиця 4.6 – Результати тестування сценарію 6

Тест	Результат
Відкриття меню товару після натискання на картку товару.	Кнопка товару активна, меню товару відображається коректно.
Придбання товару після натискання на кнопку Buy.	Стан кнопки Buy змінюється на неактивний, назва кнопки змінюється на Purchased.
Відображення придбаного товару в меню інвентарю.	Придбаний товар відображається в меню інвентарю і готовий до можливих подальших маніпуляцій.

За результатами тестів можна зробити висновок, що програма коректно оброблює етап придбання товару. Поведінка усіх кнопок спрацьовує вірно, товар відображається в меню інвентарю.

4.1.7 Перевірка вибору візуального стилю в меню інвентарю

Мета тесту: перевірити працездатність алгоритму вибору візуального стилю, зміну стану кнопок та відображення обраного стилю на рівні гри.

Вхідні дані: увімкнене програмне забезпечення, придбаний візуальний стиль.

План тестування наведено нижче.

Крок 1. Тестувальник відкриває застосунок на мобільному пристрої з версією Android 8.1 або новіше.

Крок 2. На головному екрані додатку необхідно натиснути велику коричневу кнопку з написом Inventory (на англ.) / Інвентар (на укр.).

Крок 3. У меню інвентарю необхідно натиснути на картку візуального стилю, після чого повинно відкритися його меню у якому присутнє зображення обраного товару, його назва та опис.

Крок 4. У відкритому меню візуального стилю треба натиснути на кнопку Select (на англ.) / Обрати (на укр.). Якщо після натискання на кнопку не відбулося візуальних змін, а саме кнопка не змінила статус на неактивний

та назва кнопки не змінилася на Selected (на англ.) / Обрано (на укр.), потрібно вказати про це в результатах тестування.

Крок 5. Вийти з меню магазину до головного меню натиснувши кнопку Back (на англ.) / Повернутися (на укр.). У головному меню натиснути кнопку Start (на англ.) / Початок (на укр.). Після відкриття рівня гри на сцені повинен з'явитися щойно обраний візуальний стиль. Якщо на рівні гри обраний візуальний стиль не відображається, необхідно вказати про це у результатах тестування.

Крок 6. Створення звіту результатів тестування.

Умови проходження тестування: застосунок виконує поставлене завдання згідно з метою та планом тестування, а саме успішне відображення обраного візуального стилю на рівні гри, зміна назви кнопки Select (на англ.) / Обрати (на укр.) на Selected (на англ.) / Обрано (на укр.), зміна статусу кнопки на неактивний

Результати тестування наведено в табл. 4.7.

Таблиця 4.7 – Результати тестування сценарію 7

Тест	Результат
Відкриття меню візуального стилю після натискання на картку стилю.	Кнопка товару активна, меню візуального стилю відображається коректно.
Вибір візуального стилю після натискання на кнопку Select.	Стан кнопки Select змінюється на неактивний, назва кнопки – Selected.
Відображення обраного візуального стилю на рівні гри.	Обраний візуальний стиль відображається на рівні гри.

За результатами тестів можна зробити висновок, що програма коректно оброблює етап вибору візуального стилю. Поведінка усіх кнопок спрацьовує вірно, товар відображається на рівні гри.

4.2 Порадник для користувача

Підключення контролера та запуск гри.

Крок 1. Увімкніть контролер підключивши його за допомогою дроту до мережі.

Крок 2. На мобільному пристрої увімкніть Bluetooth, почніть пошук і знайдіть контролер. Приклад ім'я контролера – A1:B2:C3:4D:E5:58. Почніть з'єднання. Якщо з'єднання успішне, контролер з'явиться в категорії «Сполучені пристрої».

Крок 3. Знайдіть логотип гри на робочому столі мобільного пристрою, натисніть і дочекайтеся її запуску.

Крок 4. Коли програма відкриється на головному екрані буде три кнопки, кнопка «START», кнопка «SHOP», кнопка «INVENTORY», кнопка «STATISTICS», кнопка «CONTROLLER» і кнопка «EXIT». Спочатку необхідно підключити контролер. Для цього натисніть на кнопку «CONTROLLER» і дочекайтеся поки відкриється меню підключення контролера.

Крок 5. У полі для введення введіть ім'я контролера, яке вказано в налаштуваннях Bluetooth у категорії «Сполучені пристрої». Після чого натисніть кнопку «Connect». Якщо з'єднання успішне, знизу екрана з'явиться надпис «Connection Status: connected».

Крок 6. Зліва вгорі натисніть кнопку «BACK» для повернення до головного меню гри.

Крок 7. Візьміть контролер у зручну для вас руку і тримайте його перед собою миготливою лампочкою догори.

Крок 8. У головному меню натисніть кнопку «START» для початку гри. Перед вами відкриється ігровий рівень з літаком та локацією після чого ви можете нахиляти контролер таким чином керуючи літаком.

Для виходу з гри використовуйте стандартні кнопки або жести мобільного пристрою. Також з гри можна вийти наступним чином.

Якщо ви знаходитесь у головному меню, тоді натисніть на кнопку «EXIT», після чого гра вимкнеться, а з'єднання з контролером буде розірвано.

Якщо ви знаходитесь безпосередньо на рівні гри, тоді натисніть зліва вгорі кнопку паузи. Перед вами з'явиться меню паузи. Натисніть кнопку «QUIT» для виходу в головне меню. Після чого повторіть дії описані вище.

4.3 Порадник для програміста

Інструменти для розробки ігрового середовища.

Для розробки ігрового середовища використовується ігровий рушій Unity версії 2022 року. Мова програмування C#, оскільки це основна мова у рушії Unity.

Інструменти для розробки інтерфейсу та ігрових об'єктів.

Для створення ігрового інтерфейсу було використано префаб з Unity Asset Store «2D Casual UI HD» [25]. Але за необхідності зміни або оновлення можна використовувати програмні інструменти, такі як Photoshop, Illustrator або інші аналоги. Розширення файлів повинно відповідати формату .jpg.

Для створення ігрової локації та перешкод використовувався інструмент для 3D моделювання Blender. Літак «Simple Airplane Controller» був доданий з Unity Asset Store як безкоштовний префаб. Розширення файлів повинно відповідати формату .fbx. Для створення нових ігрових об'єктів можна використовувати усі доступні інструменти для 3D моделювання, які можуть зберігати об'єкти в форматі .fbx. Для накладання текстур можна використовувати інструмент Substance 3D Painter, або будь який інший аналог.

Інструменти для розробки матеріального інтерфейсу (контролера).

Для розробки матеріального інтерфейсу використовувався мікроконтролер Arduino Nano, акселерометр ADXL345, Bluetooth Serial Module HC-05 та надрукована на 3D-принтері ручка. Для програмування

матеріального інтерфейсу використовувалося середовище розробки Arduino IDE.

Структуру проєкту наведено на рис. 4.1.

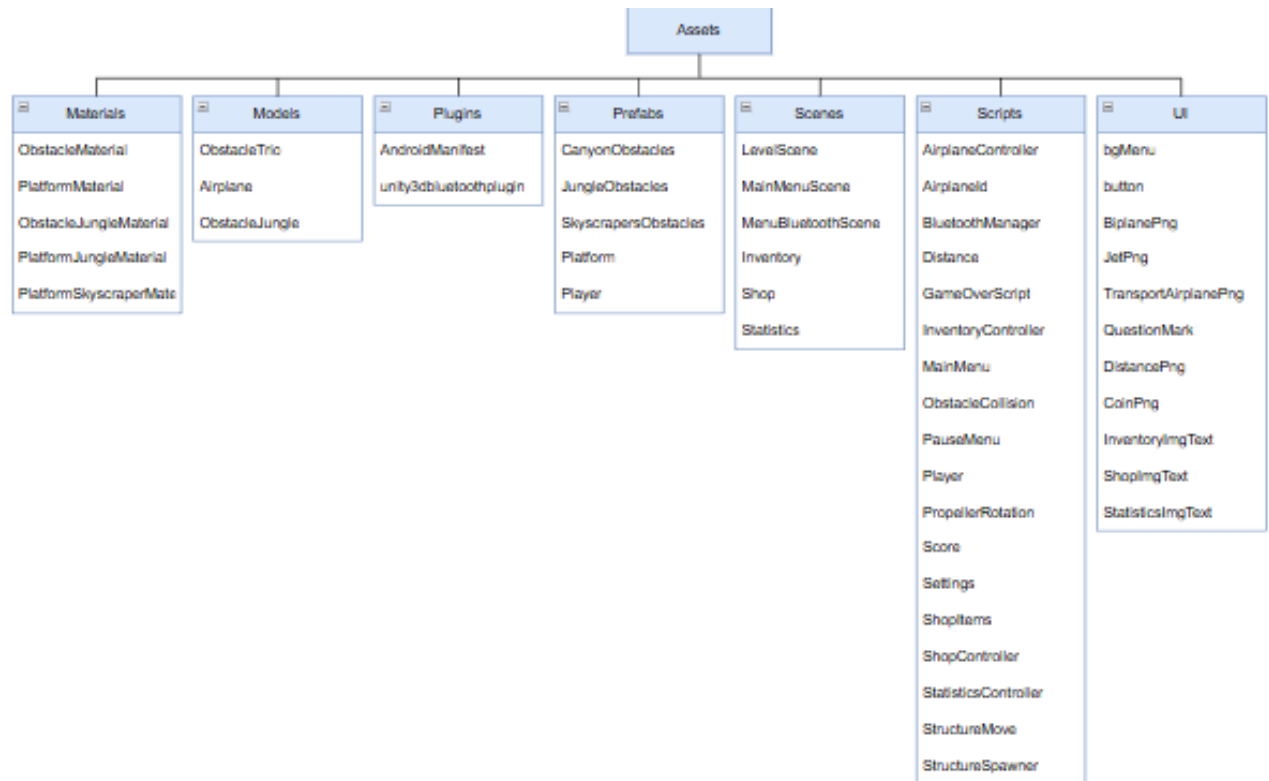


Рисунок 4.1 – Структура організації проєкту

Директорія Materials. Містить в собі усі файли пов'язані з матеріалами об'єктів сцени.

Директорія Models. Містить усі готові 3D об'єкти без накладених матеріалів та текстур.

Директорія Plugins. Містить завантажений сторонній плагін unity3dblueetoothplugin для можливості взаємодії рушія Unity з мобільним пристроєм через Bluetooth та файл AndroidManifest.

Директорія Prefabs. Містить налаштовані префаби об'єктів такі як перешкоди та літак з доданими до них скриптами та матеріалами.

Директорія Scenes. Містить різні сцени гри, а саме сцену рівня, сцену з головним меню та сцену з меню під'єднання контролера.

Директорія Scripts. Містить усі скрипти гри.

Директорія UI. Містить зображення кнопок, заднього фону та загалом інтерфейсу.

Правила іменування директорій та файлів.

Назви усіх директорій та файлів пишуться обов'язково з великої літери. Назва повинна описувати призначення директорії або файлу. При створенні файлів скриптів необхідно в кінці назви обов'язково поставити крапку та написати формат файлу, а саме <назва файлу>.cs. При найменуванні файлу не допускається використання спеціальних символів та кирилиці.

Структура скриптів продемонстрована в табл. 4.8.

Таблиця 4.8 – Структура скриптів з переліком наявних в них функцій

Скрипт	Перелік функцій
1	2
AirplaneController	Start(), Update(), _checkGyroActivity(), CalibrateGyro(), NormalizeAngle(), airplaneRotation(), airplanePosition(), airplaneStabilization(), ParseData(), Map(), DestroyPlayer().
BluetoothManager	InitBluetooth(), StartConnection(), StopConnection(), ConnectionStatus(), ReadData(), Toast().
Distance	OnTriggerEnter(), AddDistance().
GameOverScript	RestartHandler(), ReturnMainMenuHandler().
InventoryController	Start(), checkBoughtItems(), checkSelectedItem(), SelectItem(), ResetItemsInfo(), openProductMenu(), closeProductMenu().
MainMenu	Start(), StartGame(), ShopMenu(), InventoryMenu(), StatisticsMenu(), BluetoothMenu(), BackMainMenu(), QuitHandler().
ObstacleCollision	Srart(), OnCollisionEnter().

Кінець таблиці 4.8

1	2
PauseMenu	Start(), ContinueHandler(), PauseMenuHandler(), RestartMenuHandler(), ReturnMainMenuHandler().
Player	OnTriggerEnter().
PropellerRotation	Update().
Score	OnTriggerEnter(), AddScore().
Settings	Start(), Awake(), SpawnSelectedAirplane(), GetProductInfo(), GetSpecificProductInfo(), SetScore(), DeveloperAddScore(), DeveloperResetProducts().
ShopController	Start(), buyAirplane(), getInfoAbtProduct(), openPurchaseMenu(), closePurchaseMenu(), closePurchaseErrorMenu().
ShopItems	Awake().
StatisticsController	Start(), GetMaxDistance(), GetMaxScore(), ResetStatistics().
StructureMove	Update(), FixedUpdate(), Move(), DestroyStructure().
StructureSpawner	Start(), ChooseBiome(), Spawn(), SpawnObstacle().

4.4 Висновки до розділу 4

Для перевірки працездатності розробленого програмного забезпечення було проведено тестування за п'ятьма сценаріями. Тестування показало що апаратно-програмне забезпечення виконує поставлені перед ним завдання. Також було додано посібник користувача та посібник розробника. Посібник користувача це інструкція до використання апаратно-програмного забезпечення, завдяки якій користувач зможе швидко у ньому розібратися. Посібник розробника надає йому всю необхідну інформацію стосовно структури проєкту та можливості його редагування або оновлення.

ВИСНОВКИ

Під час виконання дипломної роботи було проаналізовано предметну область, а саме проведено дослідження ігрових жанрів, проаналізовано аналоги жанру нескінченний раннер та досліджені інструменти для створення ігрової графіки. Спроектовано та розроблено матеріальний інтерфейс для взаємодії з ігровим середовищем, розроблено ігрове середовище та графічний інтерфейс.

Матеріальний інтерфейс був розроблений з використанням мікроконтролера Arduino, датчика акселерометр ADXL345 та Bluetooth модуля HC-05. Матеріальний інтерфейс було запрограмовано для передачі даних, які отримує акселерометр, через Bluetooth модуль в ігрове середовище. Для налаштування обробки отриманих даних у Unity був завантажений та переналаштований сторонній безкоштовний плагін blueunity.

Ігрове середовище було розроблено з використанням ігрового рушія Unity. Під час проектування ігрового середовища були визначені основні вимоги, жанр та сеттинг гри. Розробка об'єктів та перешкод для ігрового рівня відбувалася з використанням інструменту для 3D моделювання Blender. Також було додано головного героя (літак), яким користувач буде керувати.

Графічний інтерфейс був створений згідно з вимогами та алгоритмом взаємодії гравця з грою. Він налічує чотири ігрових меню, а саме головне меню, вікно з'єднання з контролером, меню паузи та меню програшу.

Після завершення розробки мобільного забезпечення було проведено тестування за сценаріями. Результати тестування продемонстрували готовність системи до роботи.

У майбутньому планується розширяти мобільне забезпечення, додаючи до нього нові механіки та функції.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Game development studio 4A Games [Electronic resource]. – Access mode: <https://www.4a-games.com.mt/>.
2. Ігрові жанри [Електрон. ресурс]. – Режим доступу: <https://training.qatestlab.com/blog/technical-articles/games-genres/>.
3. Wikipedia Subway Surfers [Electronic resource]. – Access mode: https://en.wikipedia.org/wiki/Subway_Surfers.
4. Wikipedia Dinosaur Game [Electronic resource]. – Access mode: https://en.wikipedia.org/wiki/Dinosaur_Game.
5. Wikipedia Sonic Dash [Electronic resource]. – Access mode: https://en.wikipedia.org/wiki/Sonic_Dash.
6. Растрова та векторна графіка [Електрон. ресурс]. – Режим доступу: <https://armedsoft.com/ua/blog/rastrova-ta-vektorna-grafika>.
7. From cad data to Low-Poly models for interactive applications [Electronic resource]. – Access mode: <https://www.lightshape.net/en/blog/article/from-cad-data-to-low-poly-models-for-interactive-applications>.
8. Wikipedia Adobe Photoshop [Electronic resource]. – Access mode: https://uk.wikipedia.org/wiki/Adobe_Photoshop.
9. About GIMP [Electronic resource]. – Access mode: <https://www.gimp.org/about/introduction.html>.
10. The software Blender [Electronic resource]. – Access mode: <https://www.blender.org/about/>.
11. What is 3ds Max [Electronic resource]. – Access mode: <https://www.vanas.ca/en/blog/what-is-autodesk-maya>.
12. What is 3ds Max [Electronic resource]. – Access mode: <https://www.autodesk.com/eu/products/3ds-max/overview>.
13. In-game purchase [Electronic resource]. – Access mode: <https://unity.com/glossary/in-game-purchase>.

14. In-Game Shops [Electronic resource]. – Access mode:
https://www.gamigion.com/evolution-of-in%E2%80%91game-shops-in-mobile-f2p-games/?utm_source=chatgpt.com.
15. Clash Royal [Electronic resource]. – Access mode:
<https://store.supercell.com/clashroyale>.
16. PUBG Mobile [Electronic resource]. – Access mode:
<https://www.pubgmobile.com/en-US/home.shtml>.
17. World Of Tanks [Electronic resource]. – Access mode:
<https://wargaming.net/en/games/wot>.
18. Grand Theft Auto 5 [Electronic resource]. – Access mode:
<https://www.rockstargames.com/gta-v>.
19. Game Controllers [Electronic resource]. – Access mode:
https://en.wikipedia.org/wiki/Game_controller.
20. Різновиди керування [Електрон. ресурс]. – Режим доступу:
<https://pressa.rv.ua/news/dzhojstyky-ta-manipulyatory-dlya-kompyuternih-igor-shho-varto-znaty-pro-vybir/>.
21. Ways of Playing on Mobile [Electronic resource]. – Access mode:
<https://mobilefreetoplay.com/control-mechanics/#:~:text=Accelerometer%20control,can%20still%20play%20the%20game>.
22. Most Common Input Types [Electronic resource]. – Access mode:
<https://learn.hypehype.com/working-with-logic/player-controls>.
23. What is Unity [Electronic resource]. – Access mode:
<https://gamedevacademy.org/what-is-unity/>.
24. Unreal Engine 4 [Electronic resource]. – Access mode:
<https://blog.artcraft.net.ua/unreal-engine-4-dlya-kino>.
25. Godot Engine [Electronic resource]. – Access mode:
<https://github.com/godotengine/godot>.
26. C++ [Electronic resource]. – Access mode:
<https://uk.wikipedia.org/wiki/C%2B%2B>.

27. Wikipedia Arduino [Electronic resource]. – Access mode: <https://uk.wikipedia.org/wiki/Arduino> .
28. Arduino Mega 2560 [Електрон. ресурс]. – Режим доступу: <https://doc.arduino.ua/ru/hardware/Mega2560>.
29. Arduino Uno [Електрон. ресурс]. – Режим доступу: <https://doc.arduino.ua/ru/hardware/Uno>.
30. Arduino Nano [Електрон. ресурс]. – Режим доступу: <https://doc.arduino.ua/ru/hardware/Nano>.
31. Інтегроване середовище розробки Arduino [Electronic resource]. – Access mode: <https://docs.arduino.cc/software/ide-v1/tutorials/arduino-ide-v1-basics>.
32. Circuito.io [Electronic resource]. – Access mode: <https://www.circuito.io/>.
33. Plugin BlueUnity [Electronic resource]. – Access mode: <https://github.com/bentalebahmed/BlueUnity>.
34. Simple Airplane Controller [Electronic resource]. – Access mode: <https://assetstore.unity.com/packages/tools/game-toolkits/simple-airplane-controller-228575>.
35. 2D Casual UI HD [Electronic resource]. – Access mode: <https://assetstore.unity.com/packages/2d/gui/icons/2d-casual-ui-hd-82080>.
36. Chat GPT [Electronic resource]. – Access mode: <https://chatgpt.com/>.
37. OpenGameArt [Electronic resource]. – Access mode: <https://opengameart.org/>.

ДОДАТОК А
Текст програми

A.1 Код файла «StructureSpawner.cs»

```

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

public class StructureSpawner : MonoBehaviour
{
    [SerializeField] private List<GameObject> _obstacles;
    [SerializeField] private List<GameObject> _obstaclesJungle;
    [SerializeField] private List<GameObject> _obstaclesSkyscrapers;
    [SerializeField] private float _lengthPlatformObstacles = 60f;
    private GameObject _obstacle;
    private GameObject _obstacleJungle;
    private GameObject _obstacleSkyscrapers;
    private int randomBiome;
    private Vector3 position;

    private GameObject _currentObstacle;
    void Start()
    {
        SpawnObstacle(_obstacles);
    }

    public void ChooseBiome(int countSpawnTrigger)
    {
        if (countSpawnTrigger >= 500)
        {
            randomBiome = Random.Range(0, 3);
        }
    }

    public void Spawn()
    {
        switch (randomBiome)
        {
            case 0:
                SpawnObstacle(_obstacles);
                Debug.Log("Desert");
                break;
            case 1:

```

```

        SpawnObstacle(_obstaclesJungle);
        Debug.Log("Jungle");
        break;
    case 2:
        SpawnObstacle(_obstaclesSkyscrapers);
        Debug.Log("Skyscrapers");
        break;
    }
}

private void SpawnObstacle(List<GameObject> obstacles)
{
    Vector3 position = Vector3.zero;
    if (_currentObstacle != null)
    {
        position = new Vector3(0,0,
            _currentObstacle.transform.position.z + _lengthPlatformObstacles
        );
    }
    else
    {
        position = transform.position;
    }

    _currentObstacle = Instantiate(
        obstacles[Random.Range(0, obstacles.Count)],
        position,
        Quaternion.identity
    );
}
}

```

A.2 Код файла «StructureMove.cs»

```

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

public class StructureMove : MonoBehaviour
{
    private Rigidbody rb;

```

```

[SerializeField] private float _structureSpeed = 10f;
void Update()
{
    DestroyStructure();
}
void FixedUpdate()
{
    Move();
}
private void Move()
{
    transform.Translate(-transform.forward * _structureSpeed * 4 * Time.fixedDeltaTime);
}
private void DestroyStructure()
{
    if (transform.position.z < -30f) {
        Destroy(gameObject);
    }
}
}

```

A.3 Код файла «AirplaneController.cs»

```

using System.Numerics;
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Vector3 = UnityEngine.Vector3;
using Quaternion = UnityEngine.Quaternion;
using System.IO.Ports;
using System.Globalization;
using UnityEngine.UI;

public class AirplainController : MonoBehaviour
{
    [SerializeField] private GameObject gameOverCanvas;
    private float _speedRotation = 0.05f;
    private float _speedPositionController = 10f;
    private float _speedPositionMobile = 0.4f;
    private float[] _maxRotationZ = { -0.5f, 0.5f };
}

```

```

private float[] _maxRotationX = { -0.5f, 0.5f };
private float[] _readyDataController;
private float calibrationX = 0f;
private float calibrationY = 0f;
private Gyroscope _gyro;
private Rigidbody rb;
public Text _receivedData;
void Start()
{
    rb = GetComponent<Rigidbody>();
    _checkGyroActivity();
    CalibrateGyro();
}
void Update()
{
    _readyDataController = ParseData();

    airplaneRotation(_readyDataController);

    airplanePosition(_readyDataController);

    airplaneStabilization();

    DestroyPlayer();
}

void _checkGyroActivity()
{
    if (SystemInfo.supportsGyroscope)
    {
        _gyro = Input.gyro;
        _gyro.enabled = true;
    }
}

void CalibrateGyro()
{
    Quaternion deviceRot = _gyro.attitude;
    var rot = Quaternion.Euler(90f, 0f, 0f) *
        new Quaternion(-deviceRot.x, -deviceRot.y, deviceRot.z, deviceRot.w);

    calibrationX = NormalizeAngle(rot.eulerAngles.x);

```

```

    calibrationY = NormalizeAngle(rot.eulerAngles.z);
    Debug.Log($"Calibration X: {calibrationX}, Y: {calibrationY}");
}

private float NormalizeAngle(float angle)
{
    if (angle > 180f) angle -= 360f;
    return angle;
}

void airplaneRotation(float[] readyData, bool _mobileControl = true)
{
    if (!_mobileControl)
    {
        var rotationAirplane = transform.rotation;

        float mapX = Map(readyData[0], -10f, 10f, -0.5f, 0.5f);
        float mapY = Map(readyData[1], -10f, 10f, -0.5f, 0.5f);

        transform.Rotate(mapX * _speedRotation, 0f, mapY * _speedRotation);
    } else
    {
        Quaternion deviseRotation = _gyro.attitude;
        var rotationAirplane = Quaternion.Euler(90f, 0f, 0f) * (new Quaternion(-deviseRotation.x, -deviseRotation.y,
        deviseRotation.z, deviseRotation.w));

        float tiltX =
            NormalizeAngle(rotationAirplane.eulerAngles.x) - calibrationX;
        float tiltY =
            NormalizeAngle(rotationAirplane.eulerAngles.z) - calibrationY;

        transform.Rotate(tiltX * _speedRotation, 0f, tiltY * _speedRotation);
    }
}

void airplanePosition(float[] readyData, bool _mobileControl = true)
{
    if (!_mobileControl)
    {
        float[] parsedData = readyData;
        if (parsedData != null && parsedData.Length == 2)
        {
            float mapX = Map(parsedData[0], -10f, 10f, -2f, 2f);

```

```

        float mapY = Map(parsedData[1], -10f, 10f, -2f, 2f);
        Vector3 movement = new Vector3(-mapY, -mapX, 0.0f);
        rb.AddForce(movement * _speedPositionController);
    }
} else
{
    Quaternion deviseRotation = _gyro.attitude;
    var rotationAirplane = Quaternion.Euler(90f, 0f, 0f) * (new Quaternion(-deviseRotation.x, -deviseRotation.y,
deviseRotation.z, deviseRotation.w));

    float tiltX =
        NormalizeAngle(rotationAirplane.eulerAngles.x) - calibrationX;
    float tiltY =
        NormalizeAngle(rotationAirplane.eulerAngles.z) - calibrationY;

    Vector3 movement = new Vector3(-tiltY, -tiltX, 0.0f);
    rb.AddForce(movement * _speedPositionMobile);
}
}
void airplaneStabilization()
{
    rb.angularVelocity = Vector3.Lerp(rb.angularVelocity, Vector3.zero, Time.deltaTime * 2f);

    Quaternion targetRotation = Quaternion.Euler(0f, 0f, 0f);
    transform.rotation = Quaternion.Lerp(transform.rotation, targetRotation, Time.deltaTime * 1.5f);
}
float[] ParseData()
{
    string data = _receivedData.text;
    try
    {
        string[] parts = data.Split(',');
        if (parts.Length != 2) return null;

        float x = float.Parse(parts[0].Trim(), CultureInfo.InvariantCulture);
        float y = float.Parse(parts[1].Trim(), CultureInfo.InvariantCulture);

        return new float[] { x, y };
    }
    catch
    {
        return null;
    }
}

```

```

    }
}
public static float Map(float value, float fromLow, float fromHigh, float toLow, float toHigh)
{
    return (value - fromLow) * (toHigh - toLow) / (fromHigh - fromLow) + toLow;
}
private void DestroyPlayer()
{
    if (transform.position.z < -12f)
    {
        Destroy(gameObject);
        gameOverCanvas.SetActive(true);
        Time.timeScale = 0;
    }
}
}

```

A.4 Код файла «Controller.ino»

```

#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_ADXL345_U.h>
#include <SoftwareSerial.h>

SoftwareSerial BTSerial(2, 3);
Adafruit_ADXL345_Unified accel = Adafruit_ADXL345_Unified();
float accelerationDataXY[] = {0, 0};
void setup() {
    BTSerial.begin(9600);
    accel.begin();
}
void loop() {
    sensors_event_t event;
    accel.getEvent(&event);
    accelerationDataXY[0] = event.acceleration.x;
    accelerationDataXY[1] = event.acceleration.y;
    BTSerial.print(accelerationDataXY[0], 2);
    BTSerial.print(",");
    BTSerial.println(accelerationDataXY[1], 2);
    delay(20);
}

```

A.5 Код файла «BluetoothManager.cs»

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine.Android;
using UnityEngine;
using UnityEngine.UI;

public class BluetoothManager : MonoBehaviour
{
    public Text deviceAdd;
    public Text receivedData;
    public static bool isConnected;

    private static AndroidJavaClass unity3dblueetoothplugin;
    private static AndroidJavaObject BluetoothConnector;
    // Start is called before the first frame update
    void Start()
    {
        InitBluetooth();
        isConnected = false;
    }

    // creating an instance of the bluetooth class from the plugin
    public void InitBluetooth()
    {
        if (Application.platform != RuntimePlatform.Android)
            return;

        // Check BT and location permissions
        if (!Permission.HasUserAuthorizedPermission(Permission.CoarseLocation)
            || !Permission.HasUserAuthorizedPermission(Permission.FineLocation)
            || !Permission.HasUserAuthorizedPermission("android.permission.BLUETOOTH_ADMIN")
            || !Permission.HasUserAuthorizedPermission("android.permission.BLUETOOTH")
            || !Permission.HasUserAuthorizedPermission("android.permission.BLUETOOTH_SCAN")
            || !Permission.HasUserAuthorizedPermission("android.permission.BLUETOOTH_ADVERTISE")
            || !Permission.HasUserAuthorizedPermission("android.permission.BLUETOOTH_CONNECT"))
        {

            Permission.RequestUserPermissions(new string[] {
                Permission.CoarseLocation,

```

```

        Permission.FineLocation,
        "android.permission.BLUETOOTH_ADMIN",
        "android.permission.BLUETOOTH",
        "android.permission.BLUETOOTH_SCAN",
        "android.permission.BLUETOOTH_ADVERTISE",
        "android.permission.BLUETOOTH_CONNECT"
    });

}

unity3dblueoothplugin = new AndroidJavaClass("com.example.unity3dblueoothplugin.BluetoothConnector");
BluetoothConnector = unity3dblueoothplugin.CallStatic<AndroidJavaObject>("getInstance");
}

// Start BT connect using device MAC address "deviceAdd"
public void StartConnection()
{
    if (Application.platform != RuntimePlatform.Android)
        return;
    BluetoothConnector.CallStatic("StartConnection",deviceAdd.text.ToString().ToUpper());
}

// Stop BT connection
public void StopConnection()
{
    if (Application.platform != RuntimePlatform.Android)
        return;
    if (isConnected)
        BluetoothConnector.CallStatic("StopConnection");
}

// This function will be called by Java class to update BT connection status,
public void ConnectionStatus(string status)
{
    Toast("Connection Status: " + status);
    isConnected = status == "connected";
    // _connectionStatus.text = isConnected.ToString();
}

// This function will be called by Java class whenever BT data is received,
public void ReadData(string data)
{
    Debug.Log("BT Stream: " + data);
    receivedData.text = data;
}

// Function to display an Android Toast message

```

```

public void Toast(string data)
{
    if (Application.platform != RuntimePlatform.Android)
        return;
    BluetoothConnector.CallStatic("Toast", data);
}
}

```

A.6 Код файла «ObstacleCollision.cs»

```

using System;
using System.Collections;
using System.Collections.Generic;
using NUnit.Framework.Constraints;
using TMPro;
using UnityEngine;
using UnityEngine.SceneManagement;

public class ObstacleCollision : MonoBehaviour
{
    [SerializeField] private GameObject gameOverCanvas;
    public TextMeshProUGUI finalScoreText;
    public TextMeshProUGUI finalDistanceText;
    public AudioSource audioSource;
    public AudioClip crashSound;
    private Rigidbody rb;
    void Start()
    {
        rb = GetComponent<Rigidbody>();
    }
    void OnCollisionEnter(Collision collision)
    {
        if (collision.gameObject.tag == "Obstacle")
        {
            audioSource.PlayOneShot(crashSound);
            rb.useGravity = true;
        }
        if (collision.gameObject.tag == "Floor")
        {
            Settings.Instance.score = PlayerPrefs.GetFloat("PlayerScore", 0f);

```

```

Settings.Instance.score += float.Parse(finalScoreText.text);
PlayerPrefs.SetFloat("PlayerScore", Settings.Instance.score);
PlayerPrefs.Save();
Debug.Log("Final Score: " + Settings.Instance.score);

    if (float.Parse(finalScoreText.text) > PlayerPrefs.GetFloat("MaxPlayerScore", 0))
    {
        PlayerPrefs.SetFloat("MaxPlayerScore", float.Parse(finalScoreText.text));
        PlayerPrefs.Save();
    }
    if (float.Parse(finalDistanceText.text) > PlayerPrefs.GetFloat("MaxPlayerDistance", 0))
    {
        PlayerPrefs.SetFloat("MaxPlayerDistance", float.Parse(finalDistanceText.text));
        PlayerPrefs.Save();
    }

    gameOverCanvas.SetActive(true);
    Time.timeScale = 0;
}
}
}

```

A.7 Код файла «GameOverScript.cs»

```

using System.Collections;
using System.Collections.Generic;
using Unity.VisualScripting;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;

public class GameOverScript : MonoBehaviour
{
    private AudioSource audioSource;
    private AudioClip buttonClickSound;
    void Start()
    {
        audioSource = Settings.Instance.audioSource;
        buttonClickSound = Settings.Instance.buttonClickSound;
    }
}

```

```

public void RestartHandler()
{

    audioSource.PlayOneShot(buttonClickSound);
    Scene scene = SceneManager.GetActiveScene();
    SceneManager.LoadScene(scene.name);
    Time.timeScale = 1f;
}
public void ReturnMainMenuHandler()
{

    audioSource.PlayOneShot(buttonClickSound);
    Time.timeScale = 1f;
    SceneManager.LoadScene(0);
}
}

```

A.8 Код файла «MainMenu.cs»

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;

public class MainMenu : MonoBehaviour
{
    private AudioSource audioSource;
    private AudioClip buttonClickSound;
    public string _connectionStatus;

    void Start()
    {
        audioSource = Settings.Instance.audioSource;
        buttonClickSound = Settings.Instance.buttonClickSound;
    }
    public void StartGame()
    {
        audioSource.PlayOneShot(buttonClickSound);
        SceneManager.LoadScene(1);
    }
}

```

```

public void ShopMenu()
{
    audioSource.PlayOneShot(buttonClickSound);
    SceneManager.LoadScene(3);
}
public void InventoryMenu()
{
    audioSource.PlayOneShot(buttonClickSound);
    SceneManager.LoadScene(4);
}
public void StatisticsMenu()
{
    audioSource.PlayOneShot(buttonClickSound);
    SceneManager.LoadScene(5);
}
public void BluetoothMenu()
{
    audioSource.PlayOneShot(buttonClickSound);
    SceneManager.LoadScene(2);
}
public void BackMainMenu()
{
    audioSource.PlayOneShot(buttonClickSound);
    SceneManager.LoadScene(0);
}
public void QuitHandler()
{
    audioSource.PlayOneShot(buttonClickSound);
    Application.Quit();
}
}

```

A.9 Код файла «PauseMenu.cs»

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class PauseMenu : MonoBehaviour
{

```

```

private AudioSource audioSource;
private AudioClip buttonClickSound;
[SerializeField] GameObject pauseCanvas;

void Start()
{
    audioSource = Settings.Instance.audioSource;
    buttonClickSound = Settings.Instance.buttonClickSound;
}
public void ContinueHandler()
{
    audioSource.PlayOneShot(buttonClickSound);
    pauseCanvas.SetActive(false);
    Time.timeScale = 1f;
}

public void PauseMenuHandler()
{
    audioSource.PlayOneShot(buttonClickSound);
    pauseCanvas.SetActive(true);
    Time.timeScale = 0f;
}
public void RestartMenuHandler()
{
    audioSource.PlayOneShot(buttonClickSound);
    Scene scene = SceneManager.GetActiveScene();
    SceneManager.LoadScene(scene.name);
    Time.timeScale = 1f;
}

public void ReturnMainMenuHandler()
{
    audioSource.PlayOneShot(buttonClickSound);
    Time.timeScale = 1f;
    SceneManager.LoadScene(0);
}
}

```

A.10 Код файла «Player.cs»

```

using System.Collections;

```

```

using System.Collections.Generic;
using UnityEngine;

public class Player : MonoBehaviour
{
    [SerializeField] private StructureSpawner _structureSpawner;
    private Rigidbody lastRb;
    public int countSpawnTrigger = 0;

    void OnTriggerEnter(Collider other)
    {
        Rigidbody rb = other.attachedRigidbody;
        if (rb != null && rb != lastRb)
        {
            lastRb = rb;
            if (countSpawnTrigger >= 500)
            {
                _structureSpawner.ChooseBiome(countSpawnTrigger);
                countSpawnTrigger = 0;
            } else
            {
                countSpawnTrigger += 100;
            }
            Debug.Log(countSpawnTrigger);
            _structureSpawner.Spawn();
            Debug.Log(other.name);
        }
        else
        {
            return;
        }
    }
}

```

A.11 Код файла «PropellerRotation.cs»

```

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

public class PropellerRotation : MonoBehaviour
{

```

```

private float _speedPropeller = 10f;
// Update is called once per frame
void Update()
{
    transform.Rotate(0, 0, 100 * _speedPropeller * Time.deltaTime);
}
}

```

A.12 Код файла «Distance.cs»

```

using TMPro;
using UnityEngine;

public class Distance : MonoBehaviour
{
    [SerializeField] TextMeshProUGUI _distanceText;
    private float _distance = 0f;
    private Rigidbody lastRb;
    void OnTriggerEnter(Collider other)
    {
        Rigidbody rb = other.attachedRigidbody;
        if (rb != null && rb != lastRb)
        {
            lastRb = rb;
            AddDistance();
        }
        else
        {
            return;
        }
    }

    void AddDistance()
    {
        _distance += 100f;
        _distanceText.text = _distance.ToString();
    }
}

```

A.13 Код файла «InventoryController.cs»

```

using System;
using TMPro;
using UnityEngine;
using UnityEngine.UI;

public class InventoryController : MonoBehaviour
{
    private AudioSource audioSource;
    private AudioClip buttonClickSound;
    public GameObject productMenuBtn;
    public GameObject productMenuImg;
    public GameObject productMenuName;
    public GameObject productMenuDescription;
    public GameObject productMenuBtnSelect;
    private string defaultAirplane = "BiplaneAirplane";
    public string itemId;
    public string selectedItem;

    void Start()
    {
        audioSource = Settings.Instance.audioSource;
        buttonClickSound = Settings.Instance.buttonClickSound;
        PlayerPrefs.SetInt(defaultAirplane, 1);
        PlayerPrefs.Save();
        checkSelectedItem();
        checkBoughtItems();
        LogItemsInfo();
    }

    void checkBoughtItems()
    {
        foreach (GameObject product in Settings.Instance.products)
        {
            if (PlayerPrefs.GetInt(product.GetComponent<ShopItems>().itemId, 0) == 1)
            {
                product.SetActive(true);
            } else
            {
                product.SetActive(false);
            }
        }
    }
}

```

```

    }
}
}

```

```

public void checkSelectedItem()
{
    foreach (GameObject product in Settings.Instance.products) {
        Debug.Log($"(Inventory)Selected   name:  {product.GetComponent<ShopItems>().selectedItem},   Value:
{PlayerPrefs.GetInt(product.GetComponent<ShopItems>().selectedItem, 0)}");
    }
}

```

```

public void SelectItem()
{
    audioSource.PlayOneShot(buttonClickSound);
    foreach (GameObject product in Settings.Instance.products)
    {
        PlayerPrefs.SetInt(product.GetComponent<ShopItems>().selectedItem, 0);
        PlayerPrefs.Save();
    }

    PlayerPrefs.SetInt(selectedItem, 1);
    PlayerPrefs.Save();
    productMenuBtnSelect.GetComponentInChildren<TextMeshProUGUI>().text = "selected";
    productMenuBtnSelect.GetComponent<Button>().interactable = false;
}

```

```

private void ResetItemsInfo()
{
    foreach (GameObject product in Settings.Instance.products)
    {
        PlayerPrefs.SetInt(product.GetComponent<ShopItems>().selectedItem, 0);
        PlayerPrefs.Save();
    }
}

```

```

public void openProductMenu(GameObject objectToHandle)
{
    audioSource.PlayOneShot(buttonClickSound);
    itemId = objectToHandle.GetComponent<ShopItems>().itemId;
    selectedItem = objectToHandle.GetComponent<ShopItems>().selectedItem;

    checkSelectedItem(objectToHandle);
}

```

```

productMenuImg.GetComponent<UnityEngine.UI.Image>().sprite =
    objectToHandle.transform.Find("ProductImg")
    .GetComponent<UnityEngine.UI.Image>()
    .sprite;

productMenuName.GetComponent<TextMeshProUGUI>().text =
    objectToHandle
    .transform.Find("NameBG")
    .transform.Find("Name")
    .GetComponent<TextMeshProUGUI>()
    .text;

switch (itemId)
{
    case "BiplaneAirplane":
        productMenuDescription.GetComponent<TextMeshProUGUI>().text =
Settings.Instance.airplaneDescriptions[0];
        break;
    case "TransportAirplane":
        productMenuDescription.GetComponent<TextMeshProUGUI>().text =
Settings.Instance.airplaneDescriptions[1];
        break;
    case "JetAirplane":
        productMenuDescription.GetComponent<TextMeshProUGUI>().text =
Settings.Instance.airplaneDescriptions[2];
        break;
    case "ComingSoon":
        productMenuDescription.GetComponent<TextMeshProUGUI>().text =
Settings.Instance.airplaneDescriptions[3];
        break;
}
productMenuBtn.SetActive(true);
}

private void checkSelectedItem (GameObject item)
{
    if (PlayerPrefs.GetInt(item.GetComponent<ShopItems>().selectedItem, 0) == 1)
    {
        productMenuBtnSelect.GetComponentInChildren<TextMeshProUGUI>().text = "selected";
        productMenuBtnSelect.GetComponent<Button>().interactable = false;
    }
}

```

```

else
{
    productMenuBtnSelect.GetComponentInChildren<TextMeshProUGUI>().text = "select";
    productMenuBtnSelect.GetComponent<Button>().interactable = true;
}
}

public void closeProductMenu()
{
    audioSource.PlayOneShot(buttonClickSound);
    productMenuBtn.SetActive(false);
}
}

```

A.14 Код файла «Score.cs»

```

using TMPro;
using UnityEngine;

public class Score : MonoBehaviour
{
    [SerializeField] TextMeshProUGUI _scoreText;
    private float _score = 0f;
    private Rigidbody lastRb;
    void OnTriggerEnter(Collider other)
    {
        Rigidbody rb = other.attachedRigidbody;
        if (rb != null && rb != lastRb)
        {
            lastRb = rb;
            AddScore();
        }
        else
        {
            return;
        }
    }
    void AddScore()
    {
        _score += 10f;
        _scoreText.text = _score.ToString();
    }
}

```

```

    }
}

```

A.15 Код файла «Settings.cs»

```

using System;
using System.Collections.Generic;
using System.Linq;
using TMPro;
using Unity.VectorGraphics;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;

public class Settings : MonoBehaviour
{
    public static Settings Instance;
    public float score = 0f;
    public GameObject btnBuy;
    public GameObject btnSelect;
    public TextMeshProUGUI scoreText;
    public List<GameObject> products;
    public List<GameObject> airplanes;
    public List<string> airplaneDescriptions = new List<string>
    {
        "A classic biplane with excellent maneuverability and a vintage design.",
        "A sturdy cargo plane capable of carrying heavy loads over long distances.",
        "A high-performance jet with advanced avionics and supersonic capabilities.",
        "This item is under development."
    };
    public string selectedAirplane;
    public AudioSource audioSource;
    public AudioClip buttonClickSound;

    private void Start()
    {
        score = 200000f;
        PlayerPrefs.SetFloat("PlayerScore", score);
        PlayerPrefs.Save();

        if (SceneManager.GetActiveScene().name != "LevelScene")

```

```

    {
        SetScore();
    }
    else
    {
        SpawnSelectedAirplane();
    }

    if (SceneManager.GetActiveScene().name == "Shop")
    {
        GetProductsInfo();
    }
}

private void Awake()
{
    Instance = this;
}

public void SpawnSelectedAirplane()
{
    foreach (GameObject airplane in airplanes)
    {
        if (PlayerPrefs.GetInt(airplane.GetComponent<AirplanesId>().airplaneId, 0) == 1)
        {
            airplane.SetActive(true);
            Debug.Log($"Selected Airplane: {airplane.name}");
        }
        else
        {
            airplane.SetActive(false);
        }
    }
}

#region SettingsShop
public void GetProductsInfo()
{
    foreach (GameObject product in products)
    {
        GetSpecificProductInfo(product);
    }
}

```

```

    }

    public void GetSpecificProductInfo(GameObject product)
    {
        if (PlayerPrefs.GetInt(product.GetComponent<ShopItems>().itemId, 0) == 1)
        {
            btnBuy.GetComponentInChildren<TextMeshProUGUI>().text = "purchased";
            product.GetComponentInChildren<TextMeshProUGUI>().text = "purchased";
            btnBuy.GetComponent<Button>().interactable = false;
        }
        else if (product.GetComponent<ShopItems>().itemId == "ComingSoon")
        {
            btnBuy.GetComponentInChildren<TextMeshProUGUI>().text = "coming soon";
            btnBuy.GetComponent<Button>().interactable = false;
        }
        else
        {
            btnBuy.GetComponentInChildren<TextMeshProUGUI>().text = "buy";
            btnBuy.GetComponent<Button>().interactable = true;
        }
    }

    public void SetScore()
    {
        scoreText.text = PlayerPrefs.GetFloat("PlayerScore", 0f).ToString();
    }
#endregion SettingsShop

#region Developer Methods
public void DeveloperAddScore()
{
    score += 100000f;
    PlayerPrefs.SetFloat("PlayerScore", score);
    PlayerPrefs.Save();
    SetScore();
}

public void DeveloperResetProducts()
{
    foreach (GameObject product in products)
    {
        PlayerPrefs.SetInt(product.GetComponent<ShopItems>().itemId, 0);
    }
}

```

```

        PlayerPrefs.Save();
    }
    GetProductsInfo();
}

public static implicit operator Settings(PauseMenu v)
{
    throw new NotImplementedException();
}
#endregion Developer Methods
}

```

A.16 Код файла «ShopController.cs»

```

using UnityEngine;
using UnityEngine.UIElements;
using TMPro;
using Unity.VisualScripting;

public class ShopController : MonoBehaviour
{
    private AudioSource audioSource;
    private AudioClip buttonClickSound;
    public GameObject btnBuy;
    public GameObject purchaseMenu;
    public GameObject purchaseErrorMessage;
    public GameObject purchaseMenuImg;
    public GameObject purchaseMenuPrice;
    public GameObject purchaseMenuDescription;
    private GameObject btnProduct;
    public string itemId;

    void Start()
    {
        audioSource = Settings.Instance.audioSource;
        buttonClickSound = Settings.Instance.buttonClickSound;
    }
    public void buyAirplane()
    {
        audioSource.PlayOneShot(buttonClickSound);
        try

```

```

{
    if (PlayerPrefs.GetFloat("PlayerScore", 0) >=
float.Parse(purchaseMenuPrice.GetComponent<TextMeshProUGUI>().text.Replace(" ", "")))
    {
        PlayerPrefs.SetFloat("PlayerScore", PlayerPrefs.GetFloat("PlayerScore", 0) -
float.Parse(purchaseMenuPrice.GetComponent<TextMeshProUGUI>().text.Replace(" ", "")));
        PlayerPrefs.Save();
        Settings.Instance.SetScore();

        PlayerPrefs.SetInt(itemId, 1);
        PlayerPrefs.Save();
        Settings.Instance.GetSpecificProductInfo(btnProduct);
    } else
    {
        purchaseErrorMenu.SetActive(true);
    }
}
catch (System.Exception e)
{
    Debug.Log("Exception: " + e.Message);
}
}

public void getInfoAbtProduct(GameObject objectToHandle)
{
    btnProduct = objectToHandle;
    itemId = btnProduct.GetComponent<ShopItems>().itemId;
}

public void openPurchaseMenu()
{
    audioSource.PlayOneShot(buttonClickSound);
    Settings.Instance.GetSpecificProductInfo(btnProduct);
    purchaseMenuImg.GetComponent<UnityEngine.UI.Image>().sprite =
        btnProduct.transform.Find("ProductImg")
        .GetComponent<UnityEngine.UI.Image>()
        .sprite;
    purchaseMenuPrice.GetComponent<TextMeshProUGUI>().text =
        btnProduct
        .transform.Find("PriceBG")
        .transform.Find("Price")
        .GetComponent<TextMeshProUGUI>()

```

```

        .text;
switch (itemId)
{
    case "TransportAirplane":
        purchaseMenuDescription.GetComponent<TextMeshProUGUI>().text
Settings.Instance.airplaneDescriptions[1];
        break;
    case "JetAirplane":
        purchaseMenuDescription.GetComponent<TextMeshProUGUI>().text
Settings.Instance.airplaneDescriptions[2];
        break;
    case "ComingSoon":
        purchaseMenuDescription.GetComponent<TextMeshProUGUI>().text
Settings.Instance.airplaneDescriptions[3];
        break;
}
purchaseMenu.SetActive(true);
}
public void closePurchaseMenu()
{
    audioSource.PlayOneShot(buttonClickSound);
    closePurchaseErrorMenu();
    purchaseMenu.SetActive(false);
}
public void closePurchaseErrorMenu()
{
    audioSource.PlayOneShot(buttonClickSound);
    purchaseErrorMenu.SetActive(false);
}
}

```

A.17 Код файла «ShopItems.cs»

```

using System.Collections.Generic;
using UnityEngine;

public class ShopItems : MonoBehaviour
{
    public static ShopItems Instance;
    public string itemId;
    public string selectedItem;

```

```

private void Awake()
{
    Instance = this;
}
}

```

A.18 Код файла «StatisticsController»

```

using System.Reflection.Emit;
using TMPro;
using UnityEngine;

public class StatisticsController : MonoBehaviour
{
    public TextMeshProUGUI textMaxDist;
    public TextMeshProUGUI textMaxScore;

    void Start()
    {
        GetMaxDistance();
        GetMaxScore();
    }

    public void GetMaxDistance()
    {
        textMaxDist.text = PlayerPrefs.GetFloat("MaxPlayerDistance", 0).ToString();
    }

    public void GetMaxScore()
    {
        textMaxScore.text = PlayerPrefs.GetFloat("MaxPlayerScore", 0).ToString();
    }

    private void ResetStatistics()
    {
        PlayerPrefs.SetFloat("MaxPlayerDistance", 0);
        PlayerPrefs.SetFloat("MaxPlayerScore", 0);
        PlayerPrefs.Save();
    }
}

```

ДОДАТОК Б
Слайди презентації

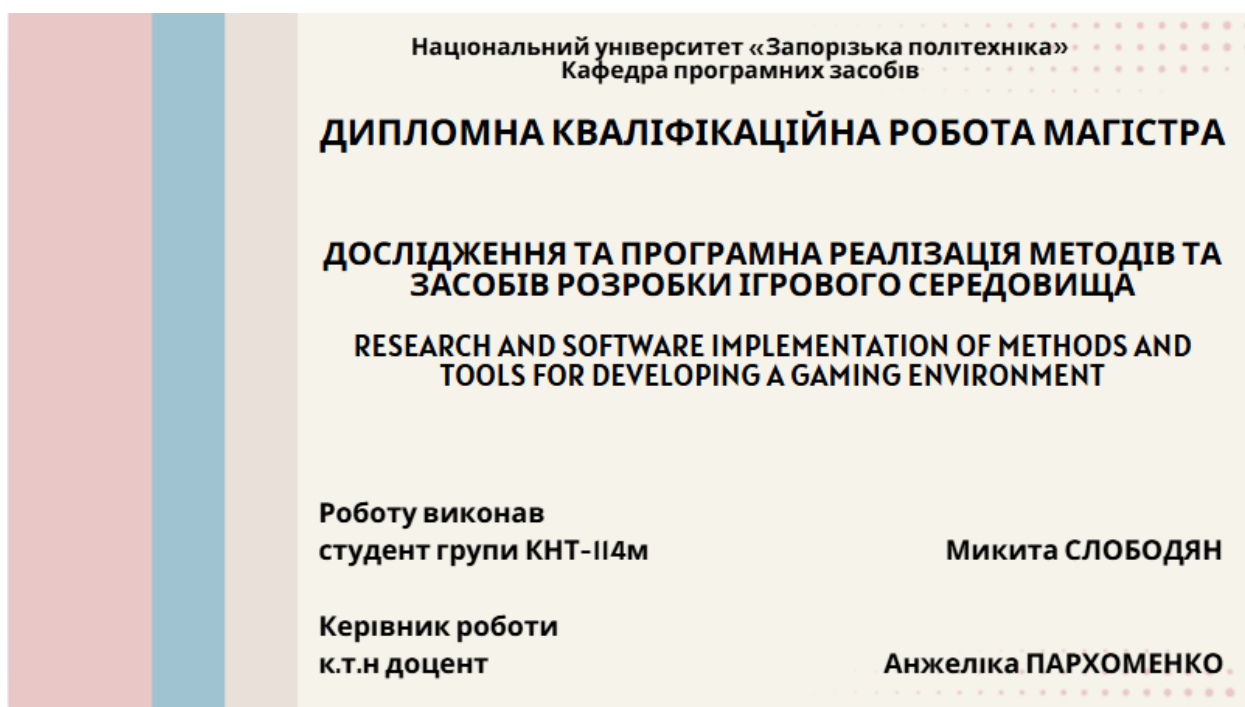


Рисунок Б.1 – Слайд 1

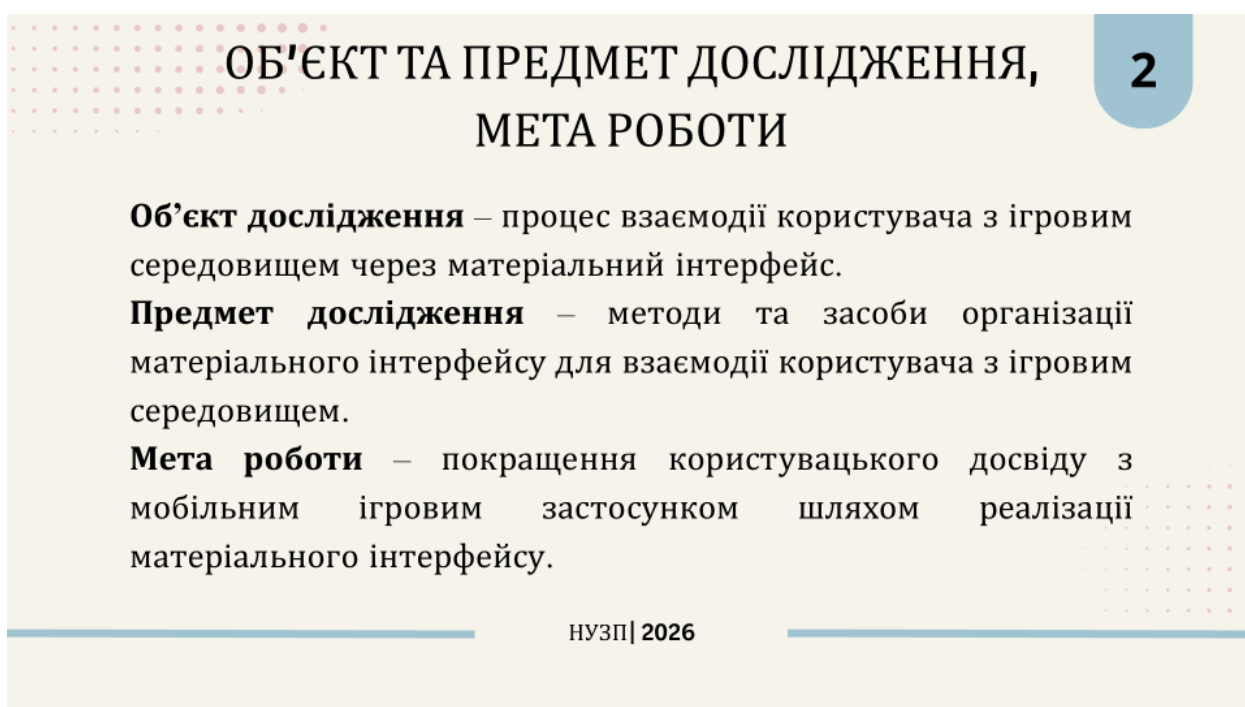


Рисунок Б.2 – Слайд 2

АКТУАЛЬНІСТЬ ТЕМИ

3

Актуальність цієї теми обумовлена стрімким зростанням ринку мобільних ігор, збільшенням числа користувачів мобільних пристроїв та їх бажанням мати більш точні та реалістичні методи керування ігровим середовищем. Сучасні смартфони мають високу продуктивність, що дозволяє реалізовувати складні ігрові механіки і тривимірну графіку. Розробка мобільної гри з використанням Unity дозволяє вивчити сучасні підходи до проектування, програмування та оптимізації інтерактивних додатків, а також отримати практичні навички, потрібні в індустрії розробки ігор та мобільних додатків.

НУЗП| 2026

Рисунок Б.3 – Слайд 3

ОГЛЯД ІГРОВИХ ЖАНРІВ

4



Загальна таблиця жанрів та під жанрів

НУЗП| 2026

Рисунок Б.4 – Слайд 4

АНАЛОГИ ІГР ЖАНРУ "НЕСКІНЧЕННИЙ РАННЕР"

5

Критерії	Subway Surfers	Dinosaur Game	Sonic Dash
Платформи	Android, iOS	Браузер Chrome	Android, iOS
Сеттинг	Міська залізнична дорога	Пустеля	Фантастичний світ
Жанр	Нескінченний раннер	Нескінченний раннер	Нескінченний раннер
Режим гри	Одиночна	Одиночна	Одиночна
Керування	Пересування по екрану (вгору, вниз, ліворуч, праворуч)	Клавішами на клавіатурі або кліки на телефоні	Пересування по екрану, наявність додаткових кнопок

НУЗП | 2026

Рисунок Б.5 – Слайд 5

ДОСЛІДЖЕННЯ ІНСТРУМЕНТІВ ДЛЯ СТВОРЕННЯ ІГРОВОЇ ГРАФІКИ

6

Критерії	Photoshop	GIMP	Blender	Maya	3ds Max
Розробник	Adobe	GNU community	Blender Foundation	Autodesk	Autodesk
Призначення	Створення растрової графіки	Створення растрової графіки	3D-моделювання, анімування, скульптинг, рендеринг	3D-моделювання, рендеринг	3D-моделювання, архітектура, рендеринг
Підписка	Платна підписка	Безкоштовна, з відкритим вихідним кодом	Безкоштовна, з відкритим вихідним кодом	Платна підписка або ліцензія	Платна підписка або ліцензія
Підтримка 3D	Обмежене створення 3D	Немає	Повний пакет 3D	Повний пакет 3D	Повний пакет 3D
Платформа	Windows, macOS	Windows, macOS, Linux	Windows, macOS, Linux	Windows, macOS, Linux	Тільки Windows

НУЗП | 2026

Рисунок Б.6 – Слайд 6

ДОСЛІДЖЕННЯ ВНУТРІШНЬОІГРОВИХ МАГАЗИНІВ

7

Гра	Тип монетизації	Основні товари	Платіжна модель	Вплив на баланс у грі
Clash Royale	F2P з мікротранзакціями	Пакети карт з персонажами, донатна валюта, скрині	Друга модель	Пришвидшення прогресії гравця. Вплив - помірний
PUBG Mobile	F2P з мікротранзакціями	Скіни на зброю, донатна валюта, кейси, BattlePass	Друга модель	Можливість отримання косметичних оновлень. Вплив - відсутній
World of Tanks	F2P з мікротранзакціями	Техніка, донатна валюта	Друга модель	Пришвидшення прогресії гравця, можливість отримання додаткових бонусів. Вплив - високий
Grand Theft Auto 5	P2P без мікротранзакцій	Техніка для пересування, зброя, нерухомість	Перша модель	Стимулює прогресію гравця. Вплив - помірний

НУЗП | 2026

Рисунок Б.7– Слайд 7

ДОСЛІДЖЕННЯ РІЗНОВИДІВ КЕРУВАННЯ В ІГРАХ

8

Тип керування	Платформа	Точність керування	Зручність керування	Область застосування
Клавіатура та миша	ПК	Висока	Висока	Шутери, стратегії
Геймпад	ПК, ігрові консолі	Дуже висока	Дуже висока	Гонки, симулятори
Сенсорне керування	Мобільний пристрій	Середня	Середня	Головоломки, карточні ігри
Керування жестами	Мобільний пристрій	Середня	Висока	Стратегії, раннери
Керування нахилом	Мобільний пристрій	Висока	Середня	Гонки, симулятори
Керування контролером	Мобільний пристрій	Дуже висока	Висока	Симулятори

НУЗП | 2026

Рисунок Б.8 – Слайд 8

ОГЛЯД ІГРОВИХ РУШІЇВ

9

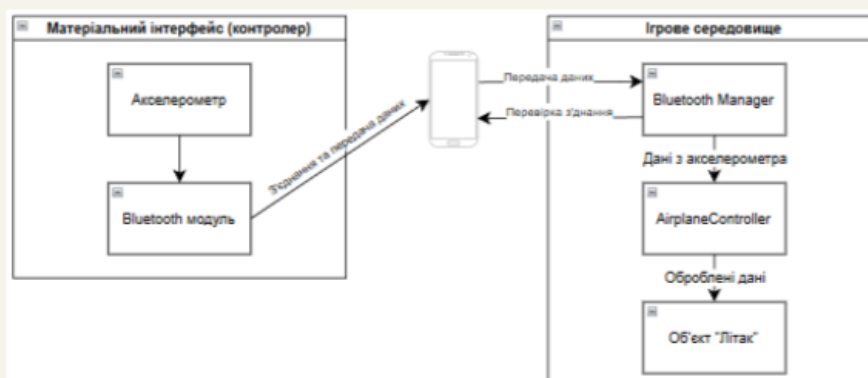
Критерії	Unity	Unreal Engine	Godot
Розробник	Unity Technologies	Epic Games	The Godot Foundation
Підписка	Безкоштовна для індивідуального використання	Безкоштовна до \$1 млн. прибутку	Безкоштовна
Програмування	UnityScript, C#, Boo	C++, Blueprints	C#Scripts, C++, C#, Visual Scripting
Призначення	Створення 2D та 3D ігор	Створення 3D ігор та анімаційних мультфільмів	Створення 2D та 3D ігор
Платформи	Windows, macOS, Linux	Windows, macOS, Linux	Windows, macOS, Linux
Marketplace	Unity Asset Store	Unreal Marketplace	Godot Asset Library

НУЗП| 2026

Рисунок Б.9 – Слайд 9

АРХІТЕКТУРА РОЗРОБЛЮВАНОЇ СИСТЕМИ

10



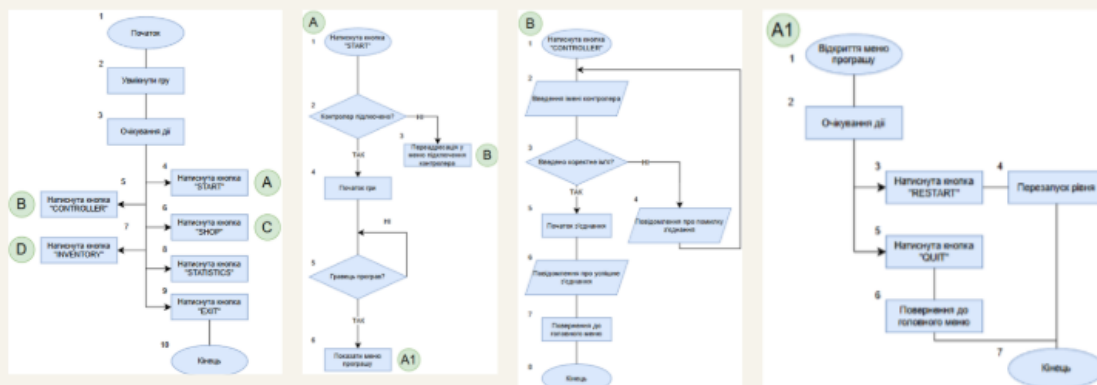
Система складається з матеріального інтерфейсу та ігрового середовища встановленого на мобільний пристрій.

НУЗП| 2026

Рисунок Б.10 – Слайд 10

ВЗАЄМОДІЯ ГРАВЦЯ З ІГРОВИМ ЗАСТОСУНКОМ

11



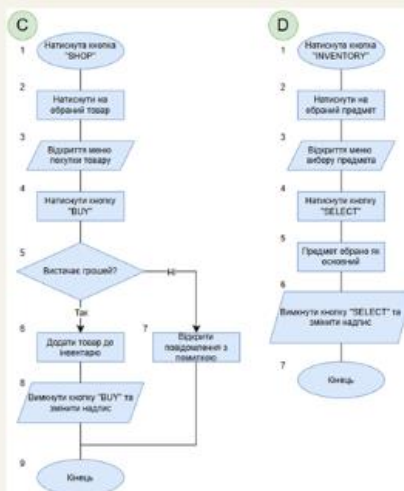
Повний алгоритм взаємодії гравця з ігровим середовищем.

НУЗП | 2026

Рисунок Б.11 – Слайд 11

ВЗАЄМОДІЯ ГРАВЦЯ З ІГРОВИМ ЗАСТОСУНКОМ

12

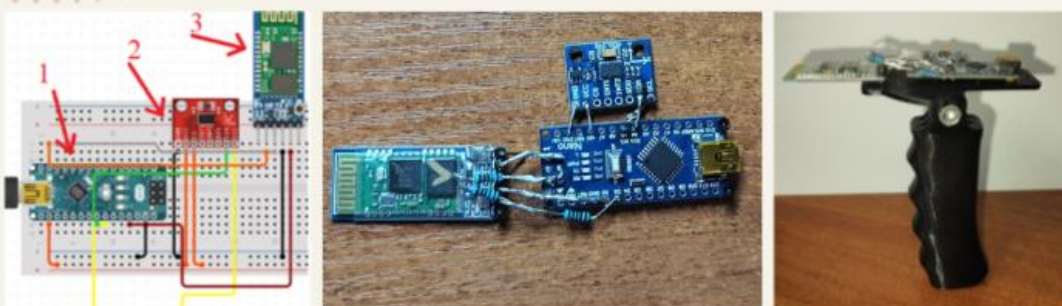


НУЗП | 2026

Рисунок Б.12 – Слайд 12

СТВОРЕННЯ КОНТРОЛЕРА

13



Контролер складається з акселерометра, модуля Bluetooth hc-05, мікроконтролера Arduino Nano та ручки надрукованої на 3D-принтері. Макет апаратної частини контролера було спроектовано на сайті circuitio.io.

НУЗП | 2026

Рисунок Б.13 – Слайд 13

ГЕЙМПЛЕЙ

14



НУЗП | 2026

Рисунок Б.14 – Слайд 14

ВИСНОВКИ

15

Під час виконання дипломної роботи було проаналізовано предметну область, а саме проведено дослідження ігрових жанрів, проаналізовано аналоги жанру нескінченний раннер, досліджено різновиди внутрішньоігрових магазинів, різновидів керування та досліджені інструменти для створення ігрової графіки. Спроековано та розроблено матеріальний інтерфейс для взаємодії з ігровим середовищем, розроблено ігрове середовище та графічний інтерфейс.

НУЗП| 2026

Рисунок Б.15 – Слайд 15