

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 проекту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РОЗРОБЛЕННЯ ТА ДОСЛІДЖЕННЯ АЛГОРИТМІВ ОБЧИСЛЕНЬ В
ІНТЕРАКТИВНІЙ КОМП'ЮТЕРНІЙ ГРІ
DEVELOPMENT AND RESEARCH OF COMPUTATION ALGORITHMS IN
AN INTERACTIVE COMPUTER GAME

Виконав(ла): студент(ка) 4 курсу, групи КНТ-212
Спеціальності 122 Комп'ютерні науки

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Комп'ютерні науки

ВЕРХОВЕЦЬКИЙ М.Є.

(ПРІЗВИЩЕ та ініціали)

Керівник ПАРХОМЕНКО А.В.

(ПРІЗВИЩЕ та ініціали)

Рецензент ПОЛЯКОВ М.О.

(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 бакалавр
 Спеціальність 122 Комп'ютерні науки
 (код і найменування)
 Освітня програма (спеціалізація) Комп'ютерні науки
 (назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
 “ ” 2026 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ВЕРХОВЕЦЬКОГО Микити Євгеновича

(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Розроблення та дослідження алгоритмів обчислень в інтерактивній комп'ютерній грі. Development and Research of Computational Algorithms in an Interactive Computer Game

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 “07” квітня 2026 року № 139

2. Строк подання студентом проєкту (роботи) 01 червня 2026 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 програмно-апаратної системи інтерактивної комп'ютерної гри. 3. Конструювання програмно-апаратної системи інтерактивної комп'ютерної гри. 4. Тестування системи та дослідження розроблених алгоритмічних рішень.

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

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ПАРХОМЕНКО А.П., доцент		
Нормоконтроль	ДЕЙНЕГА Л.Ю., ст. викладач		

7. Дата видачі завдання « 20 » квітня 2026 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.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка архітектури програми.	2 тиждень	Розділ 2
5	Розробка програми.	3-4 тижні	Розділ 3
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	ПЗ, Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

_____ Микита ВЕРХОВЕЦЬКИЙ
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)

_____ Анжеліка ПАРХОМЕНКО
(підпис) (Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 147 с., 16 табл., 22 рис., 2 дод., 34 джерела.

АЛГОРИТМИ ОБЧИСЛЕНЬ, ІНТЕРАКТИВНА КОМП'ЮТЕРНА ГРА, МАТЕРІАЛЬНИЙ ІНТЕРФЕЙС, СЕНСОРНЕ КЕРУВАННЯ, ІМУ-ДАНІ, ПРОЦЕДУРНА ГЕНЕРАЦІЯ ЛАБІРИНТУ, ТАКТИЛЬНИЙ ЗВОРОТНИЙ ЗВ'ЯЗОК.

Об'єкт дослідження – процес обчислень в інтерактивній комп'ютерній грі з матеріальним інтерфейсом.

Предмет дослідження – методи та алгоритми обчислень в інтерактивній комп'ютерній грі з матеріальним інтерфейсом.

Мета роботи – покращення продуктивності, стабільності та користувацького досвіду інтерактивної комп'ютерної гри на основі розроблення та дослідження удосконалених алгоритмів обчислень.

Матеріали, методи та технічні засоби: C++, Blueprint, Unreal Engine 5, Arduino IDE, Arduino Nano, MPU-6050, вібраційні модулі, COM-порт, експоненційне згладжування, комплементарна фільтрація, алгоритм обходу в глибину, функціональне тестування та порівняльне дослідження.

Результати. Розроблено вдосконалену програмно-апаратну систему інтерактивної гри з сенсорним керуванням і вібраційним зворотним зв'язком. Реалізовано алгоритми роботи з IMU-даними, імітації нахилу лабіринту, процедурної генерації лабіринту та просторового тактильного відгуку.

Висновки. Тестування підтвердило працездатність фінальної версії системи, а порівняльне дослідження показало зменшення затримки в роботі з IMU-даними, підвищення передбачуваності керування та покращення якості тактильного зворотного зв'язку.

Галузь використання – навчальні, демонстраційні та ігрові програмно-апаратні системи, дослідження у сфері людино-машинної взаємодії.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 147 pages, 16 tables, 22 figures, 2 appendixes, 34 sources.

COMPUTATIONAL ALGORITHMS, INTERACTIVE COMPUTER GAME, TANGIBLE INTERFACE, SENSOR CONTROL, IMU DATA, PROCEDURAL MAZE GENERATION, TACTILE FEEDBACK.

Object of study is computational processes in an interactive computer game with a tangible interface.

Subject of study is methods and computational algorithms in an interactive computer game with a tangible interface.

The purpose of the work is to improve the performance, stability and user experience of an interactive computer game based on the development and study of improved computational algorithms.

Materials, methods, and tools: C++, Blueprint, Unreal Engine 5, Arduino IDE, Arduino Nano, MPU-6050, vibration modules, COM port, exponential smoothing, complementary filtering, depth-first search algorithm, functional testing and comparative study.

Results. An improved hardware-software system of an interactive game with sensor control and vibration feedback has been developed. Algorithms for working with IMU data, maze tilt imitation, procedural maze generation and spatial tactile feedback have been implemented.

Conclusions. Testing confirmed the operability of the final version of the system, and the comparative study showed a reduction in delay when working with IMU data, an increase in control predictability and an improvement in the quality of tactile feedback.

The field of use is educational, demonstration and gaming hardware-software systems, as well as research in the field of human-machine interaction.

ЗМІСТ

	С.
Вступ.....	9
1 Аналіз предметної області.....	10
1.1 Комп'ютерні ігри з матеріальним інтерфейсом.....	10
1.2 Огляд особливостей реалізації комп'ютерних ігор з матеріальним інтерфейсом	12
1.3 Передавання, оброблення та інтерпретація сенсорних даних в інтерактивних іграх з матеріальним інтерфейсом.....	19
1.4 Аналіз попереднього варіанту розробленої системи.....	23
1.5 Постановка завдань дипломної роботи.....	26
2 Проєктування програмно-апаратної системи інтерактивної комп'ютерної гри	28
2.1 Призначення системи	28
2.2 Аналіз вимог до системи	29
2.3 Проєктування структури програмно-апаратної системи	32
2.4 Проєктування алгоритмічної логіки системи.....	40
2.5 Обґрунтування вибору інструментальних засобів і технологій.....	47
2.6 Висновки до розділу 2	54
3 Програмно-апаратна реалізація системи	56
3.1 Апаратна реалізація системи	56
3.2 Розроблення алгоритмів передавання, приймання та первинного оброблення сенсорних даних	59
3.3 Розроблення алгоритму оброблення та інтерпретації IMU-даних.....	68
3.4 Розроблення алгоритмів керування сценою для імітації нахилу лабіринту	74
3.5 Розроблення алгоритму процедурної генерації лабіринту	80
3.6 Розроблення алгоритму формування просторового тактильного зворотного зв'язку.....	87

3.7 Програмна реалізація компонентів системи та користувацької взаємодії	95
3.8 Висновки до розділу 3	104
4 Тестування системи та дослідження розроблених алгоритмічних рішень	106
4.1 Організація та методика дослідження.....	106
4.2 Функціональне тестування системи.....	108
4.3 Дослідження розроблених алгоритмічних рішень	110
4.4 Висновки до розділу 4	118
Висновки	120
Перелік джерел посилання	122
Додаток А Тексти програмних компонентів системи	126
Додаток Б Слайди презентації	140

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

COM	– Communication Port;
DFS	– Depth-First Search;
EMA	– Exponential Moving Average;
I2C	– Inter-Integrated Circuit;
IDE	– Integrated Development Environment;
IMU	– Inertial Measurement Unit;
PWM	– Pulse Width Modulation;
TUI	– Tangible User Interface;
UE5	– Unreal Engine 5;
UMG	– Unreal Motion Graphics;
USB	– Universal Serial Bus;
МК	– мікроконтролер;
ПЗ	– програмне забезпечення;
ПК	– персональний комп'ютер.

ВСТУП

Сучасні комп'ютерні ігри розвиваються як інтерактивні системи, у яких користувач безпосередньо впливає на цифрове середовище. Одним із напрямів такого розвитку є використання матеріальних інтерфейсів, де фізичні дії користувача стають частиною механізму керування грою. Поєднання фізичного контролера, сенсорного керування та зворотного зв'язку дає змогу зробити взаємодію з ігровим середовищем більш виразною [1]-[2].

Ефективність таких систем залежить не лише від конструкції контролера, а й від якості передавання, оброблення та інтерпретації сенсорних даних. IMU-сенсори дають змогу отримувати дані про рух і нахил пристрою, однак такі вимірювання можуть містити шум, дрейф і похибки. Тому для інтерактивної гри з матеріальним інтерфейсом важливими є алгоритми, що забезпечують стабільне керування, своєчасну реакцію сцени та коректне формування тактильного зворотного зв'язку [3]-[4].

Актуальність дипломної роботи зумовлена потребою в удосконаленні алгоритмічної частини інтерактивної комп'ютерної гри з матеріальним інтерфейсом. Попередній варіант системи підтвердив працездатність ідеї, однак виявив необхідність удосконалення роботи із сенсорними даними, керування сценою та формування тактильного зворотного зв'язку [5]-[6].

Мета роботи – покращення продуктивності, стабільності та користувацького досвіду інтерактивної комп'ютерної гри з матеріальним інтерфейсом на основі розроблення та дослідження удосконалених алгоритмів обчислень.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Комп'ютерні ігри з матеріальним інтерфейсом

Сучасні комп'ютерні ігри є не лише програмними продуктами для відтворення графіки, правил ігрового процесу й логіки взаємодії, а й інтерактивними системами, у яких спосіб керування безпосередньо впливає на залучення користувача. Для таких систем важливо, яким чином гравець вводить команди, як ці дії інтерпретуються програмним середовищем і який зворотний зв'язок він отримує під час гри. Одним із підходів до побудови такої взаємодії є використання матеріальних інтерфейсів користувача (Tangible User Interface, TUI), що поєднують фізичні об'єкти, зчитування дій користувача за допомогою сенсорів і цифрову реакцію ігрового середовища [2], [7].

Матеріальний інтерфейс користувача (TUI) – це тип інтерфейсу, у якому фізичні об'єкти пов'язуються з цифровими даними і використовуються як засоби подання або керування інформацією. У такому підході користувач взаємодіє не лише з віртуальними елементами на екрані, а й з реальними об'єктами, дії з якими інтерпретуються цифровою системою. Це дає змогу спиратися на природні навички взаємодії з фізичним світом: дотик, переміщення, нахил, обертання та просторову маніпуляцію [2].

На основі підходів, описаних у працях з TUI, основні ознаки такого інтерфейсу можна узагальнити так [2]:

- поєднання фізичних об'єктів із цифровими даними;
- використання матеріальних об'єктів як засобів введення або носіїв цифрового змісту;
- опора на дотик, рух і просторову маніпуляцію;
- поєднання фізичної дії з цифровим візуальним, звуковим або тактильним відгуком;
- можливість взаємодії, що не обмежується лише площиною екрана.

На відміну від графічного інтерфейсу користувача, де взаємодія переважно здійснюється через екранні елементи, мишу, клавіатуру або інші типові пристрої введення, TUI переносить частину дій користувача у фізичний простір. У такому підході керування пов'язане з маніпуляцією реальним об'єктом, а цифрова система розпізнає ці дії та формує відповідну реакцію. Це робить взаємодію більш безпосередньою, але водночас ускладнює реалізацію, оскільки потребує поєднання апаратних засобів, програмного оброблення дій користувача та механізмів зворотного зв'язку [1].

У комп'ютерних іграх матеріальні інтерфейси дають змогу наблизити керування до природних рухів користувача. Команди можуть формуватися не лише натисканням клавіш або кнопок, а й жестами, нахилами, переміщеннями чи іншими фізичними діями. Такий підхід може підвищувати інтуїтивність керування, посилювати залучення гравця та робити взаємодію з цифровим середовищем більш виразною [7].

Сенсорне керування є важливою складовою інтерактивних ігор з матеріальним інтерфейсом. У таких системах фізичні дії користувача зчитуються за допомогою сенсорів і перетворюються на керувальні сигнали для цифрового середовища. Для реалізації матеріальних інтерфейсів можуть використовуватися сенсори, мікроконтролери (МК), канали передавання даних і засоби зворотного зв'язку. Тому інтерактивну гру з матеріальним інтерфейсом доцільно розглядати як програмно-апаратну систему, у якій фізична взаємодія користувача є частиною механізму керування ігровим середовищем [1].

Ще одним важливим компонентом є тактильний, зокрема вібраційний, зворотний зв'язок. Він може доповнювати візуальну та звукову реакцію системи відчутним фізичним відгуком. Передавання інформації через дотик, тиск і вібрацію використовується для покращення сприйняття взаємодії з віртуальними об'єктами, зокрема під час оцінювання сили впливу,

розпізнавання характеру взаємодії та формування відчуття контакту з цифровим середовищем [8].

Отже, поєднання матеріального інтерфейсу користувача, сенсорного керування та тактильного зворотного зв'язку визначає специфіку предметної області цієї роботи. У таких системах важливо не лише зафіксувати фізичну дію користувача, а й коректно передати її до програмного середовища, інтерпретувати як параметр керування та сформуванню зрозумілу реакцію гри. Тому подальший аналіз доцільно зосередити на підходах до передавання, оброблення, інтерпретації сенсорних даних і формування зворотного зв'язку в інтерактивних програмно-апаратних ігрових системах.

1.2 Огляд особливостей реалізації комп'ютерних ігор з матеріальним інтерфейсом

1.2.1 Підходи до реалізації інтерактивних ігор з матеріальним інтерфейсом

У розвитку інтерактивних комп'ютерних ігор поступово сформувався запит на такі способи керування, які не обмежуються натисканням кнопок або використанням стандартного контролера. Це пов'язано з тим, що характер введення команд безпосередньо впливає на сприйняття гри: одна й та сама дія відчувається по-різному залежно від того, чи виконує користувач умовне натискання кнопки, чи здійснює рух, наближений до реальної дії. У дослідженнях наводяться приклади контролерів, у яких використовуються форма фізичного об'єкта, зчитування рухів і гіроскопічні засоби керування, а також показується, що збільшення тілесної участі може підвищувати залучення користувача та посилювати емоційне сприйняття ігрового процесу [9].

З погляду реалізації інтерактивних ігор з матеріальним інтерфейсом доцільно виокремити кілька підходів за тим, який механізм взаємодії є визначальним у системі: використання фізичного об'єкта як контролера,

зчитування дій користувача за допомогою сенсорів або формування фізичного відгуку від цифрового середовища. Такий поділ не є жорсткою класифікацією, оскільки в реальних програмно-апаратних системах ці підходи часто поєднуються, однак він дає змогу розрізнити основний акцент конкретного рішення. Узагальнену схему цих підходів наведено на рис. 1.1 [1]-[2].

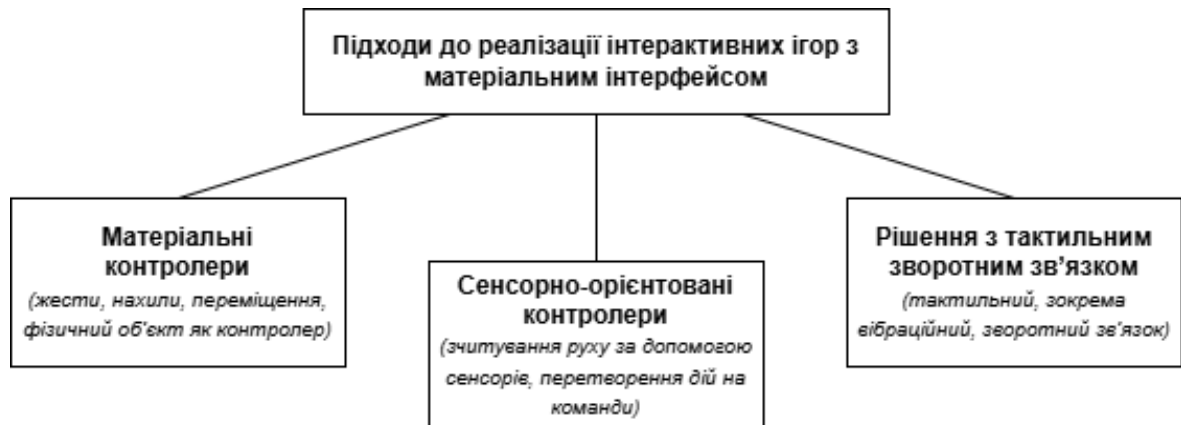


Рисунок 1.1 – Основні підходи до реалізації інтерактивних ігор з матеріальним інтерфейсом [1]-[2]

Матеріальні контролери є підходом, за якого фізичний об'єкт безпосередньо виступає засобом керування грою. У таких рішеннях важливими стають не тільки подані команди, а й сам спосіб їх виконання: нахил, поворот, переміщення, жест. Користувач взаємодіє не з абстрактною кнопкою на екрані, а з предметом, який має форму, вагу, положення в просторі та може бути частиною ігрової метафори. Саме тому матеріальний контролер дає змогу наблизити керування до звичної фізичної дії [7].

Прикладом реалізації матеріального контролера є ігровий пристрій, у якому фізичний об'єкт поєднує сенсорне зчитування рухів, передавання команд у цифрове середовище та фізичний відгук для користувача. Такі рішення показують, що матеріальний інтерфейс може бути не лише засобом керування, а й активним елементом ігрової взаємодії [7].

Сенсорно-орієнтовані контролери побудовані за іншим принципом. У них ключову роль відіграє не стільки сам фізичний носій, скільки механізм зчитування дій користувача. У гру тут вбудовується шар сенсорної інтерпретації: рухи, нахили, зміни положення чи інші фізичні дії реєструються за допомогою сенсорів, а потім перетворюються на керувальні сигнали. Це дає змогу будувати керування на природних рухах користувача, але водночас робить систему залежною від якості сенсорного зчитування, стабільності сигналу та точності подальшої інтерпретації. У технічному сенсі саме цей підхід найближче підводить до задач передавання та оброблення даних, які будуть розглядатися далі [1].

Близьким прикладом сенсорно-орієнтованого підходу є контролери, що підтримують або заохочують рухову взаємодію користувача з грою. У таких системах важливим є не лише подання команди, а й характер руху, який її супроводжує. Дослідження показують, що подібні контролери можуть підвищувати рівень залучення гравця та посилювати сприйняття ігрового досвіду. Рух у цьому випадку стає не супровідною дією, а частиною самого ігрового досвіду [10].

Рішення з тактильним зворотним зв'язком, зокрема вібраційним, становлять окремий підхід. Тут гра не лише приймає команди від користувача, а й повертає фізичний відгук у відповідь на події ігрового середовища. Найбільш поширеним варіантом є вібраційний сигнал, який доповнює візуальну та звукову реакцію системи і робить взаємодію більш відчутною. Для користувача це важливо не тільки на рівні ефекту присутності, а й на рівні сприйняття подій гри: тактильний сигнал може підсилювати момент контакту, попереджати про зміну стану системи або виділяти окрему ігрову подію серед інших [8].

Окремо варто згадати комплексні програмно-апаратні рішення, у яких матеріальний інтерфейс реалізується вже не одним контролером, а сукупністю кількох фізичних і цифрових компонентів. У таких системах можуть використовуватися сенсори, засоби розпізнавання фізичних об'єктів,

виконавчі елементи, фізичні ігрові об'єкти й програмна логіка, що координує їх роботу. Наведені в літературі приклади показують, що матеріальний інтерфейс у грі може бути реалізований як цілісна система, де фізичні елементи безпосередньо включені в ігрову механіку [11].

Переваги розглянутих підходів пов'язані насамперед із розширенням способів взаємодії користувача з грою. Залучення рухів тіла, жестів, фізичних дій і тактильного відгуку дає змогу:

- робити керування більш інтуїтивним;
- посилити залучення гравця;
- надати реакції системи більш виразного характеру;
- наблизити ігрову взаємодію до реальної дії [1], [9].

Узагальнену порівняльну характеристику підходів до реалізації інтерактивних ігор з матеріальним інтерфейсом наведено в табл. 1.1. Таблицю складено на основі аналізу підходів до TUI, тактильного зворотного зв'язку та прикладів ігрових систем із матеріальною й руховою взаємодією [1], [7]-[11].

Таблиця 1.1 – Порівняльна характеристика підходів до реалізації інтерактивних ігор з матеріальним інтерфейсом [1], [7]-[11]

Підхід	Матеріальні контролери	Сенсорно-орієнтовані контролери	Рішення з тактильним зворотним зв'язком
Спосіб взаємодії	Керування через фізичний об'єкт, жести, нахили, переміщення	Зчитування рухів, нахилів, змін положення та перетворення їх на команди	Фізичний відгук системи у відповідь на події гри
Технічна основа	Фізичний контролер, сенсори положення або руху, програмна інтерпретація дій	Сенсори, мікроконтролери, канали передавання даних, програмне оброблення сенсорних даних	Виконавчі елементи (актуатори), зокрема вібромотори, керувальна логіка, засоби синхронізації з ігровими подіями
Зворотний зв'язок	Візуальний, звуковий, за потреби тактильний	Переважно візуальний і звуковий, може поєднуватися з тактильним	Тактильний, зокрема вібраційний
Переваги	Природніше керування, краща інтуїтивність, тісніший зв'язок між дією користувача та реакцією гри	Дає змогу використовувати природні рухи як основу керування, розширює способи взаємодії з грою	Посилює сприйняття взаємодії, робить реакцію системи виразнішою, доповнює візуальний і звуковий канали
Обмеження	Складніша інтеграція апаратної та програмної частин, залежність від якості розпізнавання фізичних дій	Чутливість до нестабільності зчитування, залежність від точності сенсорних даних і способу їх інтерпретації	Потребує узгодження з логікою гри, ускладнює реалізацію, вимагає точного керування фізичним відгуком

Разом із цим такі підходи мають і низку обмежень. Порівняно зі стандартними засобами керування вони потребують складнішого поєднання апаратної та програмної частин, точнішого розпізнавання дій користувача, а також узгодження каналів сенсорного введення і зворотного зв'язку. Через це зростає залежність системи від якості технічної реалізації, стабільності зчитування фізичних дій і коректності їх інтерпретації в ігровому середовищі [1].

Інтерактивні ігри з матеріальним інтерфейсом потребують не лише вибору фізичного контролера або способу зворотного зв'язку. Визначальними для таких систем є також зчитування, передавання, оброблення та інтерпретація дій користувача, оскільки саме ці процеси впливають на точність, стабільність і своєчасність реакції ігрового середовища. Тому далі доцільно розглянути особливості роботи із сенсорними даними в інтерактивних програмно-апаратних системах [1], [7].

1.2.2 Аналіз аналогів системи

Для аналізу близьких аналогів доцільно розглядати не повні копії розроблюваної системи, а споріднені рішення, у яких фізичні дії користувача використовуються для керування цифровим середовищем, а матеріальний інтерфейс поєднується із сенсорним зчитуванням та тактильним або вібраційним зворотним зв'язком. Такий підхід дає змогу оцінити, які технічні та концептуальні рішення вже застосовуються в інтерактивних ігрових системах з матеріальним інтерфейсом, а також визначити їх сильні сторони й обмеження щодо поставленої в роботі задачі.

Одним із близьких аналогів є система, описана в роботі An Intuitive Tangible Game Controller. У ній запропоновано носимий ігровий контролер-рукавичку, який забезпечує керування цифровим середовищем за допомогою рухів руки та використовує вібраційний зворотний зв'язок для підтвердження подій у грі. Автори підкреслюють, що такий інтерфейс орієнтований на

інтуїтивність, невисоку вартість і підвищення залучення користувача до взаємодії з ігровим середовищем. Як приклад застосування розглянуто базовий авіаційний імітатор, у якому рухи руки безпосередньо впливають на поведінку віртуального об'єкта [7].

Це рішення є близьким до розроблюваної системи за кількома ознаками: використанням матеріального контролера, сенсорного зчитування рухів користувача та вібраційного зворотного зв'язку. Разом із тим воно орієнтоване на взаємодію через рукавичку та додатково використовує зовнішнє відстеження руху, тоді як у розроблюваній системі застосовано інший тип фізичного контролера та іншу ігрову задачу. Крім того, самі автори зазначають, що реалізований імітатор має демонстраційний характер і насамперед показує можливості інтерфейсу, а не завершене ігрове рішення [7].

Ще одним близьким аналогом є система HaptWarp: Soft Printable and Motion Sensible Game Controller. У цій роботі запропоновано м'який друкований контролер, обладнаний двома інерціальними вимірювальними модулями (Inertial Measurement Unit, IMU) та МК, який дає змогу зчитувати деформацію фізичного об'єкта й передавати відповідні дані у віртуальне середовище. Автори орієнтуються на інтуїтивну взаємодію у віртуальних сценаріях і підкреслюють, що фізична форма контролера дає змогу поєднати керування з тактильним відчуттям деформації об'єкта під час взаємодії з цифровим середовищем. У роботі також описано бездротове передавання даних і наведено приклади інтерактивних сценаріїв, у яких згинання та скручування контролера відповідають певним діям у віртуальному просторі [12].

Цей аналог є особливо близьким до теми дипломної роботи з технічного погляду, оскільки поєднує фізичний контролер, інерціальні сенсори, передавання даних і тактильний відгук. Водночас він зосереджений переважно на взаємодії у віртуальному середовищі та деформації самого контролера, тоді як у розроблюваній системі основна увага приділяється

керуванню інтерактивною комп'ютерною грою на основі нахилу фізичного пристрою, алгоритмічному обробленню сенсорних даних і подальшому використанню цих даних у логіці ігрового середовища [12].

Узагальнену порівняльну характеристику розглянутих аналогів наведено в табл. 1.2 [7], [12].

Таблиця 1.2 – Порівняння аналогів системи з матеріальним інтерфейсом [7], [12]

Характеристика	An Intuitive Tangible Game Controller	HaptWarp: Soft Printable and Motion Sensible Game Controller
Тип фізичного контролера	Носимий контролер у вигляді рукавички	М'який друкований контролер
Сенсорна основа	IMU, сенсори згину, зовнішнє відстеження Kinect	Два інерціальні вимірювальні модулі, МК
Спосіб взаємодії	Рухи руки та жести для керування віртуальним об'єктом	Згинання і скручування контролера для взаємодії з віртуальним середовищем
Зворотний зв'язок	Вібраційний	Тактильне відчуття через згинання та скручування м'якого контролера
Сильні сторони	Інтуїтивне керування, поєднання сенсорного введення і вібровідгуку	Поєднання фізичного контролера, IMU та передавання даних
Обмеження щодо поставленої задачі	Орієнтація на рукавичку, Kinect і авіаційний імітатор; обмежений демонстраційний характер гри	Орієнтація на сценарії віртуальної реальності, деформацію контролера та бездротове передавання; відсутність фокусу на алгоритмах керування ігровим об'єктом у лабіринті

Отже, розглянуті аналоги підтверджують, що поєднання матеріального інтерфейсу, сенсорного керування та фізичного зворотного зв'язку є перспективним напрямом розвитку інтерактивних ігрових систем. Вони демонструють можливість реалізації інтуїтивної взаємодії через фізичні дії користувача, однак не охоплюють повністю задачі, розв'язувані в цій роботі. Зокрема, у них не розглянуто комплексне поєднання алгоритмічного оброблення сенсорних даних, керування ігровим об'єктом у лабіринті та формування просторового вібраційного зворотного зв'язку. Це обґрунтовує доцільність подальшого проєктування та розроблення власної програмно-апаратної системи, орієнтованої саме на поставлену задачу.

1.3 Передавання, оброблення та інтерпретація сенсорних даних в інтерактивних іграх з матеріальним інтерфейсом

У системах з матеріальним інтерфейсом фізична дія користувача сама по собі ще не є керуванням грою. Щоб нахил, поворот або переміщення об'єкта вплинули на ігрове середовище, ці дії потрібно зафіксувати, передати в програмну частину та перетворити на цифрові сигнали. Через це якість взаємодії в таких іграх залежить не лише від конструкції контролера, а й від того, наскільки точно й швидко обробляються сенсорні дані [3]-[4].

Одним із найпоширеніших джерел таких даних є інерціальний вимірювальний модуль, який поєднує акселерометр і гіроскоп. Акселерометр реєструє питому силу, яка в умовах відносного спокою дає змогу оцінювати нахил відносно напрямку гравітації, а гіроскоп вимірює кутову швидкість. Такі дані можуть зчитуватися з високою частотою і використовуватися для оцінювання орієнтації об'єкта. Проблема в тому, що самі вимірювання не є ідеальними: на коротких інтервалах вони дають корисну інформацію для оцінювання руху, але з часом накопичуються похибки, і це погіршує результат. Саме тому інерціальні сенсори зазвичай не розглядають ізольовано, а поєднують з додатковими моделями або алгоритмами оброблення [3].

У практичній системі між сенсорним пристроєм і грою зазвичай є проміжна ланка. Це може бути МК або інший вбудований модуль, який читає дані з сенсорів, формує потік вимірювань і передає його до програмного середовища. Такий поділ потрібний не лише з технічних причин. Він дозволяє відокремити етап зчитування даних від етапу їх інтерпретації та спростити подальшу інтеграцію пристрою з ігровою системою [4].

Передавання сенсорних даних у реальному часі має свої обмеження. У дослідженні продуктивності IMU на різних апаратних платформах, зокрема Arduino-подібних, показано, що на якість роботи впливають не лише характеристики сенсорів, а й спосіб передавання даних. Бездротові

протоколи можуть вносити затримку, яку складно точно оцінити, тоді як дротове серійне передавання через інтерфейс Universal Serial Bus (USB) є стабільнішим, але також потребує контролю частоти зчитування та часових міток. У наведеному дослідженні для Arduino-подібних платформ використовувалося передавання через USB зі швидкістю 115200 бод, однак реальна частота оновлення даних не завжди збігалася із заданою [4].

Отже, у таких системах важливим є не лише саме зчитування сенсорних даних, а й уся послідовність їх проходження – від сенсорного пристрою до формування керувальних сигналів гри та зворотного зв'язку. Узагальнену схему цього процесу наведено на рис. 1.2 [3]-[4].



Рисунок 1.2 – Узагальнена схема процесу передавання та використання сенсорних даних в інтерактивній грі з матеріальним інтерфейсом [3]-[4]

Проблеми виникають і на рівні самих необроблених даних. Для IMU характерні шум, зміщення, накопичення похибки під час інтегрування та чутливість до зовнішніх впливів. Акселерометр дає корисну інформацію про нахил відносно напрямку гравітації, але в умовах вібрацій або сторонніх прискорень його дані стають нестабільними. Гіроскоп, навпаки, краще відстежує швидкі зміни, проте з часом його вимірювання схильні до дрейфу.

Через це пряме використання необроблених значень у системі керування є ненадійним [3], [13].

Окрема група проблем пов'язана з частотою зчитування, синхронізацією та часовими мітками. Для різних апаратних платформ можуть виникати нестабільність частоти зчитування, неузгодженість часової бази, а також пропущені або неточні дані. Якщо значення зчитуються швидше, ніж сенсор їх оновлює, система може повторно отримувати ті самі вимірювання. Якщо ж часові мітки неузгоджені, навіть коректні вимірювання втрачають частину цінності, оскільки ускладнюється відновлення руху в часі. Саме тому в таких системах важливе не лише зчитування даних, а й контроль їхньої часової структури [4].

Узагальнення основних проблем, які виникають під час роботи із сенсорними даними, а також можливих підходів до їх зменшення наведено в табл. 1.3. Таблицю складено на основі аналізу джерел, присвячених похибкам IMU, частоті зчитування, часовій синхронізації та алгоритмам фільтрації сенсорних даних [3]-[4], [13]-[15].

Таблиця 1.3 – Типові проблеми сенсорних даних і підходи до їх зменшення [3]-[4], [13]-[15]

Проблема	Прояв у системі	Підхід до зменшення
Шум	Коливання вимірювань, нестабільність кутів, спотворення короточасних змін	Згладжування, низькочастотна фільтрація, поєднання даних акселерометра і гіроскопа
Дрейф	Поступове накопичення похибки, відхилення оцінки орієнтації з часом	Злиття сенсорних даних, використання комплементарного фільтра або фільтра Калмана, корекція за додатковими вимірюваннями
Нестабільність частоти зчитування	Нерівномірний потік даних, повторення значень, підвищення невизначеності під час оцінювання орієнтації	Узгодження частоти оброблення, контроль оновлення даних, інтерполяція або приведення до спільної часової шкали
Затримка / часовий зсув	Запізнення реакції системи, порушення синхронності між рухом користувача та цифровою відповіддю	Використання стабільнішого каналу передавання, контроль часових міток, зменшення затримки в каналі обміну даними
Похибка інтерпретації	Некоректне перетворення необроблених вимірювань у кути, нахили або інші керувальні сигнали	Калібрування, попереднє оброблення, фільтрація та використання моделей оцінювання орієнтації

Як видно з табл. 1.3, проблеми сенсорних даних пов'язані не лише з похибками самих сенсорів, а й із передаванням, синхронізацією та інтерпретацією вимірювань. Тому перед використанням таких даних у керуванні грою їх необхідно попередньо обробляти.

Після окреслення типових проблем сенсорних даних постає задача їх попереднього оброблення. На цьому етапі можуть застосовуватися калібрування, згладжування та фільтрація, оскільки необроблені значення з акселерометра і гіроскопа не завжди придатні для безпосереднього використання в системі керування. Одним із поширених підходів є комплементарний фільтр, який поєднує низькочастотну складову даних акселерометра з високочастотною складовою даних гіроскопа. Завдяки цьому короточасні зміни краще відстежуються за показниками гіроскопа, а довготривале зміщення коригується за допомогою даних акселерометра [13]-[14].

На практиці попереднє оброблення сенсорних даних не зводиться до одного кроку. Спочатку вимірювання приводять до робочого масштабу, далі згладжують випадкові коливання, а потім використовують у фільтрації або в алгоритмах оцінювання орієнтації. Це дає змогу зменшити вплив шуму, різких стрибків і накопичених похибок, узагальнених у табл. 1.3. Перед використанням дані гіроскопа та акселерометра мають бути обнулені й масштабовані, а результат роботи комплементарного фільтра залежить від коефіцієнта фільтрації та частоти вибірки [13].

Окреме місце посідає злиття сенсорних даних – поєднання вимірювань із кількох джерел для отримання стабільнішої оцінки стану системи. Для IMU це зазвичай означає спільне використання даних акселерометра і гіроскопа, а в ширшому випадку – також інших сенсорів. До поширених підходів такого злиття належать комплементарний фільтр і фільтр Калмана. Фільтр Калмана має ширші можливості моделювання, проте є складнішим у реалізації та потребує більших обчислювальних витрат. Комплементарний фільтр є простішим і легшим для впровадження, тому його

часто розглядають як практичний варіант для вбудованих систем і систем реального часу [13]-[15].

Після фільтрації та інтерпретації початковий потік сенсорних вимірювань перетворюється на форму, придатну для керування грою: кут нахилу, оцінену орієнтацію, напрямок зміни положення або іншу узагальнену команду. На цьому рівні сенсорні дані перестають бути просто потоком числових значень і стають основою взаємодії користувача з ігровим середовищем [3].

Передавання, оброблення та інтерпретація сенсорних даних безпосередньо впливають на точність, плавність і надійність керування в інтерактивних іграх з матеріальним інтерфейсом. Через це розроблення алгоритмічних рішень для таких систем є не допоміжною, а принциповою задачею, оскільки саме ці алгоритми визначають, наскільки коректно фізична дія користувача буде перетворена на реакцію ігрового середовища [3], [15].

1.4 Аналіз попереднього варіанту розробленої системи

Попередній варіант системи являв собою інтерактивний програмно-апаратний комплекс, у якому фізичний контролер на основі інерціального сенсора використовувався для керування ігровим процесом у віртуальному середовищі. Система була реалізована в середовищі Unreal Engine 5 (UE5), а обмін даними між апаратною та програмною частинами здійснювався через комунікаційний порт (Communication Port, COM). У межах цього варіанта було реалізовано керування грою за допомогою нахилу фізичного пристрою, базове оброблення сенсорних даних, вібраційний зворотний зв'язок і взаємодію з елементами ігрової сцени [6], [16].

Практична реалізація попередньої версії підтвердила працездатність загальної ідеї матеріального інтерфейсу для інтерактивної комп'ютерної гри. Користувач міг підключати пристрій, виконувати калібрування, керувати ігровим процесом через нахил контролера, отримувати вібраційний відгук

під час зіткнень, падіння або досягнення фінішу, а також використовувати додаткові функції, зокрема вибір розміру лабіринту й режими камери. Це засвідчило, що сама концепція поєднання фізичного контролера, сенсорного зчитування, віртуального середовища та тактильного відгуку є придатною для практичної реалізації [16].

У попередньому варіанті системи вже застосовувалися окремі алгоритмічні рішення для оброблення сенсорних даних. Зокрема, для підвищення стабільності керування використовувався комплементарний фільтр, а для зменшення ривків – експоненційне ковзне середнє (Exponential Moving Average, EMA). Крім того, було реалізовано двосторонній серійний обмін між апаратною та програмною частинами, а також базовий алгоритм вібраційного зворотного зв'язку [5]-[6].

На етапі попереднього дослідження також розглядався кватерніонний підхід до подання просторових обертань, який був доцільним для задачі точного обертання об'єктів у віртуальній сцені. Проте в подальшому вдосконаленні системи від цього підходу було відмовлено, оскільки нова архітектура не передбачає постійної зміни трансформацій геометрії лабіринту: його геометрія залишається статичною, а умови руху кульки формуються за допомогою штучного вектора сили [6].

Разом із тим аналіз попереднього варіанту показав, що така реалізація мала низку обмежень. Насамперед це стосувалося способу керування сценою, за якого обертання віртуального лабіринту в сцені створювало зайве навантаження на систему й ускладнювало узгодження руху сцени та кульки. Окремою проблемою була логіка зчитування даних із СОМ-порту: часткове опрацювання потоку вимірювань призводило до накопичення затримки, через що в керування могли потрапляти застарілі сенсорні дані. Додаткове згладжування керувального сигналу після фільтрації частково маскувало цю проблему, але не усувало її причину. Базовий вібраційний відгук також потребував подальшого уточнення. У попередній версії системи він уже використовувався як реакція на події зіткнення, однак просторовий характер

такого відгуку був недостатньо виразним: напрям взаємодії визначався менш коректно, проміжні напрямки між вібраційними модулями відчувалися слабо, а нелінійна відчутність вібраційних модулів не була достатньо врахована. Крім того, у попередній реалізації не було передбачено логіки пріоритету сильнішого сигналу, через що слабша команда могла послаблювати виразніший тактильний відгук [6].

Аналіз обмежень попереднього варіанту системи та напрямів її вдосконалення наведено в табл. 1.4. Таблицю складено на основі матеріалів попереднього варіанту системи [6], [16] та тез щодо подальшого алгоритмічного вдосконалення [5].

Таблиця 1.4 – Аналіз обмежень попереднього варіанту системи та напрямів удосконалення [5]-[6], [16]

Рішення в попередній версії	Виявлене обмеження	Напрямок удосконалення
Обертання віртуального лабіринту як спосіб керування сценою	Зайве навантаження на систему та складність узгодження руху сцени й кульки	Перехід до статичного лабіринту, штучного вектора сили та візуальної імітації нахилу
Зчитування даних із COM-порту частинами	Накопичення затримки та неповне використання потоку сенсорних даних	Пакетне зчитування всіх доступних рядків із буфера
Додаткове згладжування керувального сигналу після фільтрації	Маскування проблеми приймання даних і надмірне згладжування реакції системи	Покращення приймання даних і відмова від надлишкового згладжування після фільтрації
Базовий алгоритм вібраційного зворотного зв'язку	Недостатньо виразний просторовий характер відгуку, слабе відчуття проміжних напрямків і відсутність пріоритету сильнішого сигналу	Уточнення визначення напрямку взаємодії, нелінійне масштабування сигналу, просторовий розподіл між модулями та стабілізація відгуку

Отже, попередній варіант системи підтвердив працездатність ідеї інтерактивної комп'ютерної гри з матеріальним інтерфейсом, але виявив технічні та алгоритмічні обмеження, що впливали на продуктивність, стабільність керування та якість зворотного зв'язку. Це зумовило потребу у вдосконаленні архітектури системи, зокрема переході до статичного лабіринту, покращенні роботи із сенсорними даними, формуванні умов руху кульки за допомогою штучного вектора сили та розробленні просторового

тактильного зворотного зв'язку. Виявлені обмеження стали підставою для постановки завдань дипломної роботи.

1.5 Постановка завдань дипломної роботи

За результатами аналізу, виконаного в попередніх підрозділах, було встановлено, що інтерактивні ігри з матеріальним інтерфейсом є програмно-апаратними системами, які потребують узгодженої роботи фізичного контролера, сенсорного керування, програмної інтерпретації даних і зворотного зв'язку. Проведений аналіз показав, що якість таких систем визначається не лише типом фізичного контролера або наявністю зворотного зв'язку. Важливими є також стабільність приймання сенсорних даних, коректність їх оброблення, своєчасність формування керувальних сигналів і узгоджена робота апаратної та програмної частин [1], [3].

Аналіз близьких аналогів показав, що поєднання матеріального інтерфейсу, сенсорного керування та вібраційного зворотного зв'язку є перспективним напрямом розвитку ігрових систем. Водночас розглянуті рішення не повністю охоплюють задачі цієї роботи, зокрема в частині алгоритмічного оброблення сенсорних даних, керування сценою та формування просторового тактильного відгуку [7], [12].

Крім того, аналіз попереднього варіанту розробленої системи підтвердив працездатність загальної ідеї інтерактивної комп'ютерної гри з матеріальним інтерфейсом, але водночас виявив низку обмежень. До них належать зайве навантаження, пов'язане з постійною зміною трансформацій геометрії лабіринту в сцені, недостатньо ефективна логіка зчитування даних із СОМ-порту, надмірне згладжування керувального сигналу та потреба у точнішому формуванні вібраційного зворотного зв'язку, зокрема в частині визначення напрямку взаємодії, розподілу сигналу між вібраційними модулями, урахування нелінійної відчутності вібрації та стабільності

відтворення сигналу. Це показало доцільність удосконалення архітектури системи та розроблення нових алгоритмічних рішень [5]-[6].

Проблематика дипломної роботи полягає у забезпеченні продуктивного, плавного й надійного функціонування інтерактивної комп'ютерної гри з матеріальним інтерфейсом на основі коректного передавання, оброблення та інтерпретації сенсорних даних, а також формування просторово узгодженого вібраційного зворотного зв'язку. Для таких систем саме алгоритмічні рішення значною мірою визначають якість керування, стійкість роботи та загальний користувацький досвід.

З огляду на викладене, метою дипломної роботи є покращення продуктивності, стабільності та користувацького досвіду інтерактивної комп'ютерної гри з матеріальним інтерфейсом на основі розроблення та дослідження удосконалених алгоритмів обчислень.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- сформулювати вимоги до програмно-апаратної системи, розробити її структурну та функціональну схеми;
- розробити удосконалені алгоритми обчислень для реалізації інтерактивної комп'ютерної гри з матеріальним інтерфейсом;
- виконати програмну реалізацію інтерактивної комп'ютерної гри;
- провести функціональне тестування програмно-апаратної системи, порівняльне дослідження попередньої і вдосконаленої версій та оцінити отримані результати.

2 ПРОЄКТУВАННЯ ПРОГРАМНО-АПАРАТНОЇ СИСТЕМИ ІНТЕРАКТИВНОЇ КОМП'ЮТЕРНОЇ ГРИ

2.1 Призначення системи

Розроблювана система призначена для реалізації інтерактивної комп'ютерної гри з матеріальним інтерфейсом, у якій керування ігровим процесом здійснюється не традиційними засобами введення, а за допомогою фізичного пристрою з інерціальним сенсором. Система має забезпечувати зчитування дій користувача, передавання сенсорних даних до програмної частини, їх подальше оброблення та перетворення на керувальні сигнали гри, а також формування вібраційного зворотного зв'язку у відповідь на події ігрового середовища. Для цього використовується фізичний контролер на базі Arduino Nano та MPU-6050, а обмін даними з програмною частиною організовано через СОМ-порт.

Функціонально система орієнтована на проходження тривимірного лабіринту, у якому користувач за допомогою нахилу фізичного контролера керує імітованим нахилом лабіринту, що визначає умови руху кульки в ігровому середовищі. Перед початком роботи користувач підключає пристрій до комп'ютера, обирає відповідний СОМ-порт і виконує калібрування, після чого сенсорні дані використовуються для керування ігровим об'єктом у реальному часі. Під час гри система повинна забезпечувати не лише реакцію на нахил пристрою, а й підтримку основних подій ігрового процесу, зокрема зіткнень, падіння, досягнення фінішу, повторного запуску рівня та переходу між режимами роботи.

Окрім базового сценарію гри, система має забезпечувати допоміжні можливості, необхідні для повноцінного використання: відображення інтерфейсу користувача, вибір розміру лабіринту, перемикання режимів камери, калібрування сенсора, відкриття та закриття СОМ-порту, а також формування команд вібраційного зворотного зв'язку для фізичного пристрою. Таким чином, призначення системи полягає не лише у створенні

інтерактивної гри як програмного продукту, а й у реалізації цілісної програмно-апаратної взаємодії між користувачем, фізичним контролером і віртуальним ігровим середовищем.

2.2 Аналіз вимог до системи

2.2.1 Функціональні та нефункціональні вимоги

Вимоги до розроблюваної системи визначають, які можливості вона має забезпечувати під час взаємодії користувача з інтерактивною комп'ютерною грою, а також якими властивостями повинна характеризуватися її робота. З урахуванням призначення системи та результатів аналізу предметної області доцільно виділити функціональні та нефункціональні вимоги. Функціональні вимоги пов'язані з конкретними можливостями системи, тоді як нефункціональні визначають якість її роботи, зручність використання та стійкість функціонування.

До функціональних вимог системи належать:

- вибір доступного СОМ-порту для підключення фізичного контролера;
- відкриття та закриття з'єднання з фізичним пристроєм;
- калібрування інерціального сенсора перед початком гри;
- приймання потоку сенсорних даних від фізичного пристрою;
- первинний розбір вхідних повідомлень і передавання коректних значень до модуля оброблення та інтерпретації IMU-даних;
- інтерпретація нахилу фізичного контролера для формування параметрів імітованого нахилу лабіринту;
- формування умов руху кульки відповідно до параметрів імітованого нахилу лабіринту;
- процедурна генерація лабіринту відповідно до вибраного розміру;
- оброблення основних подій гри, зокрема зіткнень, падіння кульки та досягнення фінішу, а також підтримка повторного запуску рівня;

- формування команд вібраційного зворотного зв'язку для фізичного пристрою;
- перемикання режимів камери;
- робота головного меню, ігрового інтерфейсу, екрана калібрування та меню завершення гри;
- підтримка локалізованого інтерфейсу користувача.

До нефункціональних вимог системи належать:

- достатня продуктивність і швидкодія для приймання, оброблення та використання сенсорних даних у реальному часі без помітного запізнення реакції системи;
- недопущення накопичення затримки під час зчитування з СОМ-порту та зменшення ризику використання застарілих сенсорних даних;
- стабільність приймання та передавання даних між фізичним пристроєм і програмною частиною;
- плавність реакції системи та руху кульки без різких змін керувального впливу;
- надійність калібрування та коректність подальшого використання сенсорних вимірювань;
- стійкість роботи основних підсистем під час тривалого використання;
- зручність користування інтерфейсом і зрозумілість основних дій для користувача;
- коректне формування вібраційного зворотного зв'язку відповідно до подій ігрового середовища;
- гнучкість налаштування окремих параметрів системи без порушення її основної логіки, зокрема розміру лабіринту, режиму камери та мовних налаштувань.

Сформульовані вимоги охоплюють як необхідний набір функцій, які повинна реалізовувати система, так і основні характеристики її роботи. Саме

вони слугують основою для подальшого проєктування структури системи, вибору алгоритмічних рішень і програмної реалізації її компонентів.

2.2.2 Обмеження та умови функціонування

Розроблювана система призначена для роботи як програмно-апаратний комплекс, тому її використання можливе лише за наявності двох взаємопов'язаних частин: фізичного контролера та програмної частини гри на персональному комп'ютері (ПК). Фізичний пристрій повинен містити інерціальний сенсор, МК і засоби вібраційного зворотного зв'язку. У розроблюваній конфігурації для цього застосовується контролер на базі Arduino Nano, інерціальний модуль MPU-6050 і три вібротори, а обмін даними виконується через USB-інтерфейс із використанням COM-порту. Вимірювання мають надсилатися приблизно кожні 10 мс, тобто з частотою близько 100 Гц.

Важливою умовою функціонування системи є коректне підключення пристрою до комп'ютера через USB і наявність доступного COM-порту. Перед початком гри користувач повинен вибрати відповідний порт і виконати калібрування сенсора, оскільки без цього керування не можна вважати достатньо точним. Під час калібрування пристрій має бути зафіксований у визначеному положенні, а сама процедура повинна завершитися до початку активної взаємодії з ігровим середовищем. Тому калібрування передбачене як обов'язковий етап перед стартом гри.

Ще одне обмеження пов'язане з характером сенсорного керування. Якість роботи системи залежить від стабільності потоку даних, коректності їх приймання та подальшого оброблення в реальному часі. Різкі зовнішні впливи на контролер, порушення калібрування або збої під час передавання даних можуть зменшувати точність керування і погіршувати плавність реакції ігрового об'єкта. Крім того, система розрахована на використання одного фізичного контролера в межах одного ігрового сеансу, а зміна

параметрів гри виконується лише в межах передбачених налаштувань інтерфейсу.

Окремим архітектурним обмеженням є відмова від зміни трансформацій усієї геометрії лабіринту в ігровій сцені. Оскільки лабіринт складається з багатьох елементів, постійне оновлення їх положення та орієнтації створювало б додаткове навантаження на систему та ускладнювало б узгодження фізичної моделі руху кульки. Тому розроблювана система має передбачати статичну геометрію лабіринту, імітацію його нахилу та формування умов руху кульки на основі параметрів нахилу фізичного контролера.

Програмна частина системи також має власні умови функціонування. Вона повинна запускатися на ПК, який підтримує виконання гри та роботу з послідовним обміном даними. Через використання механізмів отримання списку доступних СОМ-портів система орієнтована на роботу в середовищі Windows. Це слід враховувати як практичне обмеження під час експлуатації та тестування.

Сформульовані умови та обмеження визначають межі, в яких система може використовуватися за призначенням. Їх урахування є необхідним під час подальшого проєктування, програмної реалізації та експериментальної перевірки роботи розроблюваної програмно-апаратної системи інтерактивної комп'ютерної гри.

2.3 Проєктування структури програмно-апаратної системи

На основі сформульованих функціональних і нефункціональних вимог, а також визначених умов функціонування, розроблювана система розглядається як програмно-апаратний комплекс, що поєднує фізичний контролер, канал обміну даними та програмну частину гри.

2.3.1 Склад системи та загальний принцип роботи

До складу розроблюваної системи входять три основні складові: фізичний пристрій керування, канал обміну даними та програмна частина гри. Фізичний пристрій містить інерціальний сенсор, МК і засоби вібраційного зворотного зв'язку. Його призначення полягає у зчитуванні рухів користувача, формуванні сенсорних даних, передаванні цих даних до програмної частини, а також у прийманні команд для керування вібротворами. У розроблюваній системі для цього використовується контролер на базі Arduino Nano та інерціального модуля MPU-6050, а вібраційний зворотний зв'язок забезпечується трьома вібротворами, якими можна керувати окремими командами.

Проміжною ланкою між фізичним пристроєм і програмною частиною є канал обміну даними, реалізований через COM-порт. Через нього здійснюється двосторонній послідовний обмін: від пристрою до гри передаються сенсорні дані у визначеному форматі, а у зворотному напрямі надсилаються команди калібрування та керування вібровідгуком. У програмній частині для цього передбачено модуль серійного обміну, який забезпечує відкриття COM-порту, приймання вхідних рядків, їх розбір, передавання команд до пристрою та отримання переліку доступних COM-портів системи.

Програмна частина гри реалізує приймання та оброблення сенсорних даних, ігрову логіку, процедурну генерацію лабіринту, імітацію нахилу лабіринту, формування умов руху кульки, формування команд вібраційного зворотного зв'язку та інтерфейс користувача. Для цього в ній передбачено модулі серійного обміну, інтерпретації IMU-вимірювань, процедурної генерації лабіринту, керування імітованим нахилом лабіринту, вібраційного зворотного зв'язку, а також модулі меню, калібрування, вибору COM-порту, вибору розміру лабіринту й перемикання режимів камери. Загальна структурна схема програмно-апаратної системи наведена на рис. 2.1.

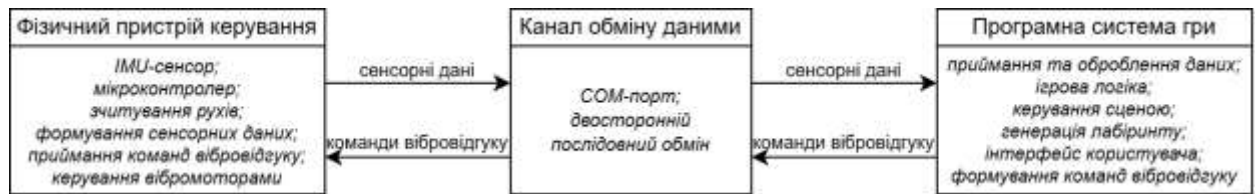


Рисунок 2.1 – Загальна структурна схема програмно-апаратної системи інтерактивної комп’ютерної гри

Загальний принцип роботи системи полягає в тому, що користувач виконує фізичні дії з контролером, після чого інерціальний сенсор фіксує зміни його положення та формує потік вимірювань. Ці дані передаються через COM-порт до програмної частини, де вони зчитуються, обробляються та перетворюються на параметри імітованого нахилу лабіринту й умови руху кульки в ігровому середовищі. У відповідь на події ігрового процесу, зокрема зіткнення, падіння або досягнення фінішу, програмна система формує команди вібраційного зворотного зв’язку, які зворотним каналом надходять до фізичного пристрою. Така організація забезпечує двосторонню взаємодію між користувачем, апаратною частиною та віртуальним середовищем.

2.3.2 Підсистеми та їх призначення

Після визначення загального складу системи доцільно виокремити кілька основних підсистем, кожна з яких виконує власну функцію в загальному циклі роботи гри. Таке розділення потрібне не лише для зручності опису, а й для подальшого проєктування взаємодії між компонентами, розподілу алгоритмічних задач і визначення меж відповідальності окремих модулів. У межах розроблюваної системи можна виділити апаратну підсистему керування, підсистему серійного обміну даними, підсистему оброблення та інтерпретації IMU-даних, ігрову підсистему та підсистему інтерфейсу користувача.

Апаратна підсистема керування забезпечує фізичну взаємодію користувача з ігровим середовищем. До її складу входять МК Arduino Nano,

інерціальний модуль MPU-6050 та три вібромотори. Ця підсистема відповідає за зчитування необроблених даних інерціального сенсора, виконання калібрування, приймання службових команд від програмної частини та керування вібраційним відгуком. Саме на цьому рівні фізичні дії користувача перетворюються на потік вимірювань, який передається до програмної частини гри, а у зворотному напрямі надходять команди для вібромоторів.

Підсистема серійного обміну даними виконує роль проміжної ланки між апаратною і програмною частинами. Вона відповідає за відкриття та закриття СОМ-порту, приймання рядків із потоку даних, їх первинний розбір, формування службових команд і отримання переліку доступних портів. У межах цієї підсистеми дані сенсора перетворюються з текстового формату у внутрішнє подання, придатне для подальшого оброблення, а також формуються команди калібрування і керування вібромоторами. Тому ця підсистема значною мірою визначає стійкість зв'язку між фізичним контролером і грою.

Підсистема оброблення та інтерпретації IMU-даних призначена для перетворення необроблених вимірювань акселерометра і гіроскопа на стабільні параметри керування. У ній виконуються нормалізація вимірювань, попереднє згладжування даних і оцінювання орієнтації на основі комплементарного фільтра. Результатом її роботи є набір кутових параметрів, які можуть використовуватися в логіці гри для формування параметрів імітованого нахилу лабіринту. Це окрема підсистема, оскільки вона відокремлює задачу приймання даних від задачі їх подальшого оброблення та змістовної інтерпретації.

Ігрова підсистема є центральною частиною програмної логіки системи. Вона відповідає за керування ігровим процесом, оброблення подій, формування умов руху кульки, керування параметрами імітованого нахилу лабіринту, процедурну генерацію лабіринту та формування команд вібраційного зворотного зв'язку. До цієї підсистеми належать модулі,

пов'язані з ігровою логікою в UE5: обчислення керувального впливу для формування умов руху кульки, процедурна генерація лабіринту, оброблення зіткнень, падіння та досягнення фінішу, а також керування режимами камери й допоміжними станами гри. Саме тут сенсорні дані набувають ігрового змісту і перетворюються на поведінку об'єктів у віртуальному середовищі.

Окремо виділяється підсистема інтерфейсу користувача. Вона забезпечує взаємодію користувача з програмною частиною гри до початку ігрового сеансу, під час гри та після її завершення. До неї належать головне меню, екран калібрування, ігровий інтерфейс, меню перемоги, елементи локалізації та допоміжні віджети для службової інформації. Через цю підсистему користувач виконує вибір СОМ-порту, змінює розмір лабіринту, перемикає режими камери, запускає гру або повертається до головного меню.

Таке виділення підсистем дає змогу розглядати розроблювану систему не як набір окремих функцій, а як узгоджену структуру взаємопов'язаних компонентів. У подальшому це полегшує опис потоків даних, алгоритмічної логіки та ролі окремих модулів у загальному функціонуванні гри. Взаємодію підсистем і проходження даних між ними розглянуто в наступному підрозділі.

2.3.3 Взаємодія підсистем і потоки даних

Після виокремлення основних підсистем доцільно розглянути, яким чином між ними відбувається обмін даними та як формується загальний цикл функціонування системи. Для розроблюваної програмно-апаратної системи важливою є не лише наявність окремих підсистем, а й узгоджена взаємодія між ними, оскільки саме вона визначає коректність керування ігровим процесом, своєчасність зворотного зв'язку та загальну стійкість роботи.

Початковою ланкою цього процесу є апаратна підсистема керування. Інерціальний сенсор фіксує зміни положення фізичного контролера, після чого МК формує потік вимірювань і передає його до комп'ютера через USB-

інтерфейс із використанням COM-порту. У розроблюваній конфігурації дані надсилаються у вигляді текстових рядків, що містять значення акселерометра та гіроскопа у форматі D:ax,ay,az,gx,gy,gz, а також можуть супроводжуватися службовими повідомленнями про стан пристрою або завершення калібрування. На цьому етапі апаратна підсистема виконує роль джерела сенсорних даних для всієї програмної логіки системи.

Далі потік даних надходить до підсистеми серійного обміну, де виконується приймання вхідних рядків, їх первинний розбір і виділення коректних значень акселерометра та гіроскопа. Після цього дані передаються до підсистеми оброблення та інтерпретації IMU-даних, у якій вони нормалізуються, згладжуються і перетворюються на стабільні параметри орієнтації. Саме на цьому етапі необроблені сенсорні значення набувають форми, придатної для використання в ігровій логіці. Отримані параметри далі використовуються ігровою підсистемою для керування параметрами імітованого нахилу лабіринту, формування умов руху кульки, оброблення подій гри та інших реакцій віртуального середовища.

Окремий потік взаємодії формується у зворотному напрямі. Після оброблення подій ігрового процесу, зокрема зіткнень, падіння або досягнення фінішу, ігрова підсистема генерує команди вібраційного зворотного зв'язку. Ці команди передаються через підсистему серійного обміну до апаратної підсистеми, де МК розподіляє їх між трьома вібромоторами. Таким чином забезпечується двостороння взаємодія: від користувача до гри передаються сенсорні дані, а від гри до користувача – сигнали вібраційного зворотного зв'язку. Підсистема інтерфейсу користувача при цьому не бере безпосередньої участі в перетворенні сенсорних даних, але забезпечує налаштування роботи системи, відображення її стану та доступ до основних сценаріїв взаємодії.

Схему взаємодії підсистем і проходження даних між ними наведено на рис. 2.2.

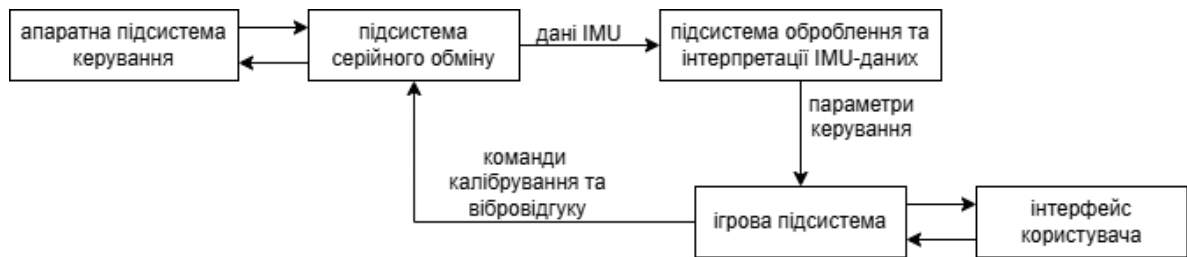


Рисунок 2.2 – Схема взаємодії підсистем і потоків даних у програмно-апаратній системі

Побудова такої взаємодії дає змогу відокремити зчитування, передавання, інтерпретацію і використання даних у грі, а також узгодити прямий і зворотний канали обміну між апаратною та програмною частинами. Це створює основу для подальшого проєктування користувацької взаємодії із системою.

2.3.4 Проєктування користувацької взаємодії

Під час проєктування програмно-апаратної системи важливо врахувати не лише її внутрішню структуру, а й логіку взаємодії користувача з основними функціями гри. У цій роботі така взаємодія розглядається як послідовність типових сценаріїв, які охоплюють підготовку системи до роботи, запуск ігрового процесу, керування під час гри та дії після її завершення. Це дає змогу визначити, які саме можливості повинні бути доступні користувачу на кожному етапі роботи із системою.

Початковий сценарій взаємодії пов'язаний із підготовкою системи до гри. Після запуску програми користувач переходить до головного меню, де має змогу обрати доступний COM-порт для підключення фізичного контролера, виконати калібрування інерціального сенсора та змінити розмір лабіринту. На цьому етапі користувач ще не взаємодіє безпосередньо з ігровим об'єктом, однак виконує дії, без яких повноцінне керування грою було б неможливим або неточним.

Після завершення підготовчого етапу користувач запускає гру і переходить до основного сценарію взаємодії. У межах цього сценарію користувач нахиляє фізичний контролер, а система інтерпретує цей нахил як керування нахилом лабіринту, що визначає рух кульки в ігровому середовищі. Під час гри користувач може взаємодіяти і з додатковими функціями системи: змінювати режим камери, ставити гру на паузу, перезапускати рівень або повертатися до головного меню. Тобто взаємодія із системою не зводиться до одного безперервного ігрового процесу, а включає також допоміжні керувальні дії.

Окремий сценарій пов'язаний із завершенням гри. Після досягнення фінішу система повинна відобразити екран перемоги, з якого користувач може або повернутися до головного меню, або розпочати нову гру. Такий підхід дає змогу логічно завершити ігровий сеанс і не розривати загальний цикл взаємодії користувача із системою.

Діаграму прецедентів, що відображає основні дії користувача під час роботи із системою, наведено на рис. 2.3.

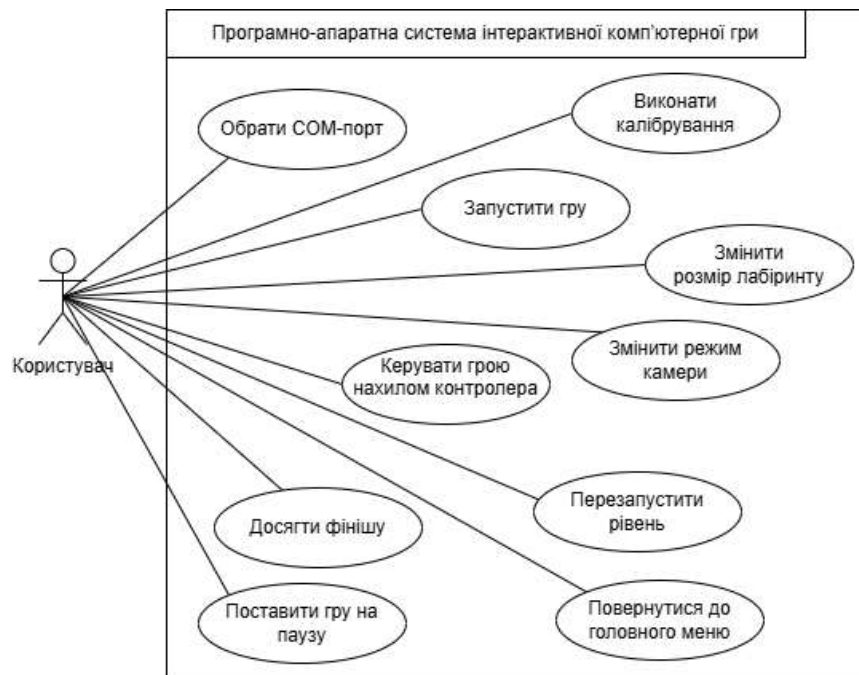


Рисунок 2.3 – Діаграма прецедентів взаємодії користувача із системою

Побудова таких сценаріїв взаємодії дає змогу ще на етапі проєктування узгодити логіку роботи інтерфейсу, порядок доступу до основних функцій та послідовність дій користувача під час використання системи. Надалі це спрощує проєктування інтерфейсної частини гри та забезпечує більш цілісну організацію взаємодії між користувачем, фізичним контролером і програмною системою.

2.4 Проєктування алгоритмічної логіки системи

2.4.1 Узагальнений алгоритмічний ланцюг функціонування системи

Для розроблюваної програмно-апаратної системи важливо не лише визначити окремі підсистеми, а й подати загальну послідовність алгоритмічних етапів, через які проходять дані під час роботи гри. Це дає змогу розглядати систему не як набір ізольованих функцій, а як цілісний ланцюг перетворень: від фізичної дії користувача до реакції ігрового середовища та формування вібраційного зворотного зв'язку.

Початковою ланкою цього ланцюга є фізичний контролер із сенсором і вібромоторами. Користувач нахиляє фізичний пристрій, після чого інерціальний сенсор фіксує відповідні зміни його положення. Далі відбувається зчитування і передавання сенсорних даних до програмної частини системи, де вони проходять етап приймання та первинної підготовки. На цьому рівні дані ще не можуть безпосередньо використовуватися в ігровій логіці, оскільки потребують подальшого оброблення, згладжування та інтерпретації.

Наступними етапами є згладжування та фільтрація, а також обчислення параметра керування. Саме тут необроблені сенсорні значення перетворюються на стабільні параметри, придатні для використання в системі керування грою. Отриманий керувальний параметр використовується для імітації нахилу лабіринту. На цій основі в ігровій системі формуються

умови руху кульки, що дає змогу реалізувати керування проходженням лабіринту через нахил фізичного контролера без фізичного обертання всієї геометрії сцени.

Окремим завершальним етапом є формування вібраційного зворотного зв'язку. Після зміни стану ігрового середовища система аналізує події ігрового процесу, зокрема зіткнення, падіння або досягнення фінішу, і за потреби генерує команди для вібромоторів фізичного контролера. У результаті формується замкнений цикл взаємодії, у якому фізичні дії користувача впливають на гру, а гра, своєю чергою, повертає користувачу тактильну реакцію.

Узагальнений алгоритмічний ланцюг функціонування системи наведено на рис. 2.4.



Рисунок 2.4 – Функціональна схема системи

Подання функціонування системи у вигляді такого ланцюга дає змогу перейти від загального опису її роботи до виокремлення основних алгоритмічних блоків. Саме вони надалі визначають логіку передавання, приймання та первинного оброблення сенсорних даних, оброблення та інтерпретації IMU-даних, керування сценою для імітації нахилу лабіринту, процедурної генерації лабіринту та формування просторового тактильного зворотного зв'язку.

2.4.2 Обґрунтування алгоритмічних блоків системи

Функціонування розроблюваної програмно-апаратної системи не зводиться до одного універсального алгоритму. Для забезпечення стійкого керування грою, коректного використання сенсорних даних і формування зворотного зв'язку в її структурі доцільно виділити кілька окремих алгоритмічних блоків. Кожен із них виконує власну задачу, але водночас пов'язаний з іншими блоками в межах єдиного циклу роботи системи.

Першим необхідним блоком є алгоритм передавання, приймання та первинного оброблення сенсорних даних. Його призначення полягає в тому, щоб забезпечити стійкий обмін між фізичним контролером і програмною частиною гри. На цьому рівні відбуваються зчитування вимірювань із сенсора, формування пакета даних, передавання його через послідовний інтерфейс, приймання в програмній частині, первинний розбір рядків і виділення коректних значень акселерометра та гіроскопа. Без такого блоку подальше оброблення IMU-даних було б неможливим, оскільки ігрова система не могла б отримувати узгоджений потік вимірювань у придатному для роботи форматі.

Другим алгоритмічним блоком є алгоритм оброблення та інтерпретації IMU-даних. Необроблені значення акселерометра і гіроскопа самі по собі ще не можуть використовуватися як параметри керування, тому їх необхідно нормалізувати, згладити та перетворити на стабільну оцінку орієнтації. Саме цей блок відокремлює фізичне зчитування даних від їх змістовного використання в ігровій логіці. Він є одним із ключових, оскільки від якості його роботи безпосередньо залежать плавність керування, точність реакції системи та стійкість поведінки ігрового середовища.

Окрему групу становлять алгоритми керування сценою для імітації нахилу лабіринту. У розроблюваній системі користувач сприймає взаємодію так, ніби нахиляє сам лабіринт, однак на рівні програмної логіки це реалізується через поєднання кількох дій: обчислення параметра керування,

узгодження параметрів ігрової сцени, оновлення положення камери та формування умов руху кульки на основі параметрів нахилу. Такий блок потрібний тому, що він поєднує результати інтерпретації сенсорних даних із візуальною та фізичною поведінкою ігрового середовища. Саме тут керувальні параметри набувають завершеного ігрового змісту.

Ще одним самостійним алгоритмічним блоком є алгоритм процедурної генерації лабіринту. Його виокремлення виправдане тим, що структура ігрового середовища в системі не є фіксованою, а формується під час роботи з урахуванням заданого розміру та логіки побудови проходів, стін, отворів і фінішної зони. Цей блок не бере участі в обробленні сенсорних даних безпосередньо, однак визначає простір, у межах якого реалізується керування грою. Через це він є повноцінною частиною алгоритмічної логіки системи, а не лише допоміжною процедурою створення рівня.

Завершальним важливим блоком є алгоритм формування просторового тактильного зворотного зв'язку. Його задача полягає в тому, щоб на основі подій ігрового процесу сформувати змістовну тактильну реакцію, яка повертається користувачу через фізичний пристрій. У цій системі вібраційний зворотний зв'язок пов'язаний не лише з фактом події, а й з її просторовими характеристиками, оскільки сила та напрямок вібрації розподіляються між кількома віброторами. Завдяки цьому тактильний канал перестає бути лише допоміжним ефектом і стає окремим елементом взаємодії користувача з грою.

Отже, у структурі розроблюваної системи доцільно виділити п'ять основних алгоритмічних блоків: алгоритм передавання, приймання та первинного оброблення сенсорних даних; алгоритм оброблення та інтерпретації IMU-даних; алгоритми керування сценою для імітації нахилу лабіринту; алгоритм процедурної генерації лабіринту; алгоритм формування просторового тактильного зворотного зв'язку. Таке виділення дає змогу перейти від загального алгоритмічного ланцюга до обґрунтування конкретних підходів, які доцільно використати в кожному з цих блоків.

2.4.3 Обґрунтування вибору підходів

Після виділення основних алгоритмічних блоків системи необхідно обґрунтувати вибір підходів, покладених в основу їх реалізації. У цій роботі вирішальними були не формальна складність алгоритмів і не прагнення використати найбільш складні з теоретичного погляду моделі, а відповідність конкретним умовам функціонування системи. Розроблювана гра працює в реальному часі, використовує сенсорне керування, двосторонній обмін даними та фізичний зворотний зв'язок, тому обрані підходи мали забезпечувати достатню точність, стабільність і швидкодію без надмірного ускладнення всієї програмно-апаратної системи.

Для блоку передавання, приймання та первинного оброблення сенсорних даних було обрано підхід із компактним текстовим форматом пакета та буферизованим прийманням рядків. У межах цієї системи таке рішення є виправданим, оскільки поєднує простоту реалізації з достатньою надійністю для роботи в реальному часі. На стороні фізичного контролера вимірювання надсилаються у фіксованому форматі, а в програмній частині розбираються та переводяться у внутрішнє подання, придатне для подальшого використання. Для цієї задачі не було потреби у складнішому двійковому протоколі, оскільки він ускладнив би налагодження системи й не забезпечив би суттєвої переваги за невеликого обсягу передаваних даних. Натомість критично важливими є своєчасність надходження вимірювань, відсутність накопичення застарілих значень у буфері та збереження придатної для керування частоти оновлення даних, що і визначило вибір компактного формату повідомлень і приймання доступних рядків із буфера [4].

Для блоку оброблення та інтерпретації IMU-даних було обрано поєднання експоненційного ковзного середнього та комплементарного фільтра. Таке рішення зумовлене характером самих інерціальних вимірювань. Необроблені IMU-дані містять шум, а покази акселерометра і

гіроскопа мають різні особливості: акселерометр дає корисну інформацію про орієнтацію відносно вектора тяжіння, але є чутливим до сторонніх прискорень і вібрацій, тоді як гіроскоп краще відстежує короточасні зміни, проте з часом накопичує похибку. Експоненційне ковзне середнє в цій схемі виконує функцію попереднього згладжування і зменшує дрібні коливання сигналу ще до етапу оцінювання орієнтації. Комплементарний фільтр дає змогу поєднати дані акселерометра і гіроскопа без надмірного обчислювального навантаження. У наукових джерелах такий підхід розглядається як практичний компроміс між якістю оцінювання орієнтації та простотою реалізації, а використання складніших методів, зокрема фільтра Калмана, не є обов'язковим для досягнення потрібної плавності й стійкості керування в межах розроблюваної системи [3], [14]-[15].

Для алгоритмів керування сценою було обрано підхід, за якого геометрія лабіринту залишається статичною і не змінює своїх трансформацій, а ефект нахилу імітується алгоритмічно. Це одне з ключових архітектурних рішень системи, оскільки лабіринт не є одним простим об'єктом, а складається з великої кількості елементів сцени. Як було показано під час аналізу попереднього варіанту системи, постійна зміна трансформацій геометрії лабіринту мала низку обмежень. Рушію доводилося постійно оновлювати трансформації багатьох об'єктів, обробляти їхню участь у фізичній моделі та узгоджувати це з рухом кульки. Такий підхід створював зайве обчислювальне навантаження, підвищував залежність поведінки системи від частоти кадрів і ускладнював стабільне керування. Тому в розроблюваній системі обрано інший підхід: геометрія лабіринту залишається статичною, рух кульки формується за допомогою штучного вектора сили, а візуальна імітація нахилу забезпечується узгодженою зміною положення камери та напрямку освітлення. Завдяки цьому система уникає великої кількості зайвих трансформацій, а керувальна логіка стає простішою, передбачуванішою і менш ресурсоємною [6].

Для процедурної генерації лабіринту було обрано алгоритм обходу в глибину (Depth-First Search, DFS) з рекурсивним проходженням клітин і випадковим перемішуванням напрямків. Такий вибір є виправданим не тому, що цей алгоритм формально перевершує всі інші можливі підходи, а тому, що він добре відповідає задачі системи. Алгоритм обходу в глибину є одним із поширених підходів до генерації лабіринтів і може застосовуватися для побудови випадкової зв'язної структури проходів. Для розроблюваної гри його перевага полягає у практичній придатності: він не потребує складних допоміжних структур, легко масштабується зі зміною розміру лабіринту і дозволяє послідовно формувати прохідний ігровий простір, на який потім можна накладати додаткові елементи. Для цієї системи цього достатньо, оскільки основною вимогою є стабільне створення коректного, варіативного ігрового середовища, а не використання найскладнішого з можливих генераторів лабіринтів [17].

Для блоку формування просторового тактильного зворотного зв'язку було обрано підхід, за якого вібрація залежить не лише від факту події, а й від її сили та напрямку. Простий варіант з однаковим сигналом для всіх моторів був би значно легшим у реалізації, однак у такому разі вібровідгук залишався б лише загальним підтвердженням події. У розроблюваній системі ставилося інше завдання: зробити так, щоб тактильний канал передавав користувачу додаткову інформацію про характер взаємодії з ігровим середовищем. Саме тому сила події визначається за параметрами зіткнення, далі нормалізується і масштабується, а напрямок розподіляється між трьома вібромоторами. У результаті користувач отримує не лише сигнал про факт події, а й додаткову інформацію про те, наскільки сильно куляка зіткнулася з перешкодою та з якого боку відбувся контакт. Це робить вібровідгук не декоративним ефектом, а окремим каналом передавання інформації про стан ігрового середовища. У дослідженнях, присвячених вібраційному зворотному зв'язку, показано, що такий канал може використовуватися не лише як загальне повідомлення про подію, а й як спосіб передавання напрямкових або

просторових ознак взаємодії, що узгоджується з логікою розроблюваної системи [18].

Таким чином, вибір підходів для основних алгоритмічних блоків системи визначався їх відповідністю конкретним задачам розроблюваної гри. Узагальнення вибраних підходів, причин їх застосування та очікуваного ефекту наведено в табл. 2.1. Таблицю сформовано на основі аналізу джерел [3]-[4], [14]-[15], [17]-[18], тез щодо алгоритмічного ланцюга системи [5] та матеріалів попереднього варіанту системи [6].

Таблиця 2.1 – Узагальнення вибору підходів для алгоритмічних блоків системи [3]-[6], [14]-[15], [17]-[18]

Алгоритмічний блок	Обраний підхід	Коротке обґрунтування
Передавання, приймання та первинне оброблення сенсорних даних	Компактний текстовий формат пакета та буферизоване приймання рядків	Забезпечує простоту обміну, своєчасне отримання вимірювань і зменшення затримки
Оброблення та інтерпретація IMU-даних	Експоненційне ковзне середнє та комплементарний фільтр	Дає змогу зменшити шум і отримати стійкі параметри керування без надмірного обчислювального навантаження
Керування сценою для імітації нахилу лабіринту	Статичний лабіринт, штучний вектор сили, керування камерою та освітленням	Зменшує кількість трансформацій об'єктів сцени та робить керування менш ресурсоємним і стабільнішим
Процедурна генерація лабіринту	DFS з рекурсивним проходженням клітин і випадковим перемішуванням напрямків	Дозволяє формувати зв'язний і варіативний лабіринт при відносно простій реалізації
Формування просторового тактильного зворотного зв'язку	Просторовий розподіл сигналу між трьома вібротворами	Дає змогу передавати інформацію про силу та напрямок контакту, а не лише факт події

2.5 Обґрунтування вибору інструментальних засобів і технологій

2.5.1 Засоби розроблення програмної частини

Для реалізації програмного забезпечення (ПЗ) системи необхідно було обрати такі засоби, які підтримують розроблення тривимірного ігрового середовища, роботу з фізичною моделлю, створення інтерфейсу користувача та інтеграцію з програмними модулями, що відповідають за приймання й оброблення даних від фізичного пристрою. Важливою вимогою була також можливість поєднувати візуальну логіку з програмним кодом, оскільки

частина системи пов'язана з ігровою поведінкою та взаємодією елементів сцени, а частина – з реалізацією службових модулів і обробленням даних.

Для обґрунтування вибору основного ігрового рушія було розглянуто кілька поширених засобів розроблення тривимірних інтерактивних застосунків: Unreal Engine 5, Unity та Godot. Порівняння виконано за критеріями, важливими для цієї роботи: підтримка тривимірної сцени й фізичної моделі, можливість поєднання візуальної логіки з програмним кодом, засоби створення інтерфейсу користувача, придатність до інтеграції з фізичним контролером і відповідність уже сформованій архітектурі системи. Узагальнення результатів порівняння наведено в табл. 2.2. Таблицю складено на основі документації Unreal Engine [19]-[21], Unity [22]-[24] та Godot [25]-[27].

Таблиця 2.2 – Порівняння ігрових рушіїв для реалізації програмної частини системи [19]-[27]

Критерій	Unreal Engine 5	Unity	Godot	Висновок для роботи
1	2	3	4	5
Підтримка тривимірної сцени та фізичної моделі	Має розвинені засоби роботи з 3D-сценою, фізикою, камерою, освітленням та ігровими об'єктами	Підтримує 3D-сцени та фізичні інтеграції	Підтримує 3D-сцени та базові засоби роботи з об'єктами сцени	Усі рушії придатні для 3D-гри, однак UE5 відповідає вже обраній архітектурі системи
Поєднання візуальної логіки та програмного коду	Підтримує Blueprint і C++, що дозволяє поєднувати візуальну і програмну реалізацію	Основна логіка реалізується через C#, візуальна логіка можлива через відповідні інструменти або додаткові засоби	Підтримує GDScript, C# і розширення через GDExtension	UE5 забезпечує зручне поєднання Blueprint для ігрової логіки та C++ для службових модулів
Засоби інтерфейсу користувача	Має UMG і Slate для створення меню, віджетів та інтерфейсних екранів	Має UI Toolkit та інші засоби побудови інтерфейсу	Має систему UI-вузлів на основі Control	Усі рушії мають засоби UI, але UMG/Slate безпосередньо узгоджуються з реалізацією в UE5
Інтеграція з фізичним контролером	Можлива через плагін серійного обміну та C++/Blueprint-логіку	Можлива через C#-код або сторонні засоби роботи з послідовним обміном	Можлива через додаткові розширення або зовнішні модулі	Для цієї роботи UE5 є доцільним, оскільки вже використовується плагін серійного обміну
Відповідність попередній версії системи	Попередній варіант системи вже був реалізований у UE5	Перехід потребував би перенесення ігрової сцени, логіки та інтеграційних рішень	Перехід потребував би повної перебудови сцени та програмної логіки	Збереження UE5 дозволяє зосередитися на вдосконаленні алгоритмів, а не на перенесенні системи в інший рушій

Кінець таблиці 2.2

1	2	3	4	5
Доцільність використання в цій роботі	Висока: рушій підтримує потрібну 3D-сцену, фізику, Blueprint, C++, UI та інтеграцію з COM-портом	Можлива, але потребувала б перенесення системи й адаптації реалізації	Можлива для менш складних або нових проєктів, але потребувала б перебудови реалізації	Для поставленої задачі UE5 є найбільш доцільним вибором

Порівняння показує, що всі розглянуті рушії можуть використовуватися для створення тривимірних інтерактивних застосунків, однак для цієї роботи найбільш доцільним є використання Unreal Engine 5. Це пов'язано не з абсолютною перевагою рушія над іншими засобами, а з відповідністю UE5 конкретним умовам розроблюваної системи: наявністю засобів для роботи з 3D-сценою та фізичною моделлю, підтримкою Blueprint і C++, інструментами UMG/Slate для побудови інтерфейсу, можливістю інтеграції серійного обміну та сумісністю з попереднім варіантом реалізації системи [19]-[27].

Як основне середовище розроблення ПЗ було обрано UE5. Цей рушій надає засоби для створення тривимірної сцени, роботи з ігровими об'єктами, камерою, освітленням і користувацьким інтерфейсом, а також підтримує спільне використання програмування мовою C++ і візуальної логіки Blueprint. Для розроблюваної системи це є важливим, оскільки вона поєднує віртуальний лабіринт, рух кульки, зміну режимів камери, оброблення подій сцени та інтерфейсні екрани налаштування, калібрування й завершення гри [19]-[21].

Підхід із поєднанням Blueprint і C++ було обрано свідомо. Blueprint доцільно використовувати для реалізації частини ігрової логіки, взаємодії об'єктів сцени, інтерфейсних переходів і налаштувань, оскільки він дає змогу швидко створювати й змінювати поведінку системи без постійного переписування програмного коду. C++, своєю чергою, доцільний для реалізації власних класів, службових модулів і частин системи, де важливими є точність, контроль над даними та тісніша інтеграція з внутрішніми

механізмами рушія. Таке поєднання дозволяє залишити більш гнучку логіку взаємодії у Blueprint, а технічні та повторно використовувані частини реалізувати програмно [19]-[20].

Для розроблення C++-частини системи використовується Visual Studio, оскільки вона підтримує роботу з проектами UE5, створення і редагування класів, перегляд службових журналів, налагодження коду та інтеграцію з інструментами рушія. Наявність спеціалізованих засобів для Unreal Engine спрощує розроблення власних програмних модулів і робить роботу з C++-частиною системи більш зручною та керованою [28].

Отже, вибір UE5, Blueprint, C++ і Visual Studio узгоджується з потребами розроблюваної системи. Ці засоби забезпечують створення тривимірного ігрового середовища, підтримку інтерфейсу користувача, реалізацію ігрової логіки та програмну реалізацію алгоритмічних компонентів, які буде детальніше розглянуто в наступному розділі.

2.5.2 Засоби розроблення апаратної частини

Для реалізації апаратної частини системи необхідно було обрати таку базу, яка забезпечує зчитування даних з інерціального сенсора, передавання їх до програмної частини гри, виконання калібрування та керування трьома вібромоторами в режимі, близькому до реального часу. При цьому важливими були не лише технічні характеристики окремих компонентів, а й їхня достатність для поставленої задачі, простота інтеграції між собою та сумісність із загальною логікою розроблюваної системи.

Як основу апаратної частини було обрано плату Arduino Nano. Такий вибір є виправданим, оскільки ця плата має достатню кількість цифрових і аналогових виводів, підтримує послідовний обмін через USB-інтерфейс із використанням USB–Serial-перетворювача і працює на частоті 16 МГц. Для розроблюваної системи цього достатньо, щоб зчитувати дані з інерціального сенсора, передавати їх до комп'ютера через послідовний інтерфейс і

формувати сигнали керування для трьох вібромоторів. Важливим є й те, що Arduino Nano має компактні розміри, тому її зручно використовувати у фізичному контролері, який повинен залишатися невеликим і придатним для ручного керування [29].

Для зчитування просторових параметрів руху було обрано інерціальний модуль MPU-6050. Це рішення прямо відповідає задачі системи, оскільки MPU-6050 належить до шестивісних інерціальних вимірювальних модулів і поєднує в одному пристрої акселерометр та гіроскоп. Сенсор використовує цифровий інтерфейс Inter-Integrated Circuit (I2C), а також підтримує кілька діапазонів вимірювання для гіроскопа й акселерометра. Для розроблюваної гри це є достатнім, оскільки система не потребує складнішої сенсорної конфігурації, а натомість потребує компактного й придатного до інтеграції джерела IMU-даних для керування нахилом [30].

Як середовище розроблення прошивки апаратної частини було обрано Arduino IDE (Integrated Development Environment) – інтегроване середовище розроблення, призначене для написання, перевірки та завантаження програм у плати Arduino. У межах цієї системи це середовище є достатнім для написання, налагодження та завантаження програми в МК, оскільки не виникає потреби в складнішому середовищі вбудованого розроблення. Функціональність апаратної частини зводиться до зчитування даних із сенсора, оброблення команд калібрування, формування потоку вимірювань і керування вібромоторами, тому використання Arduino IDE є виправданим як з погляду простоти, так і з погляду відповідності поставленій задачі [31].

Окремо слід зазначити, що вібромотори в цій системі не розглядаються як самостійний засіб розроблення, однак їх наявність впливає на вибір апаратної бази. Плата керування повинна забезпечувати формування окремих сигналів широтно-імпульсної модуляції (Pulse Width Modulation, PWM) для трьох вібромоторів, а отже мати достатню кількість придатних для цього виводів і підтримувати керування вібраційним відгуком без

ускладнення всієї схеми. У поєднанні з Arduino Nano та MPU-6050 така конфігурація виявляється достатньою для реалізації фізичного контролера розроблюваної гри.

Отже, вибір Arduino Nano, MPU-6050 та Arduino IDE узгоджується з вимогами до апаратної частини системи. Обрані засоби забезпечують зчитування IMU-даних, їх передавання до програмної частини, підтримку калібрування та керування вібраційним зворотним зв'язком, не створюючи надмірної складності на рівні апаратної реалізації.

2.5.3 Засоби інтеграції та підтримки користувацької взаємодії

Для розроблюваної системи було недостатньо лише засобів створення програмної та апаратної частин. Окремо потрібно було обрати інструменти, які забезпечують інтеграцію фізичного контролера з грою, побудову інтерфейсу користувача та підтримку локалізованої взаємодії. Саме ці засоби визначають, наскільки зручно користувач може підключити пристрій, виконати початкові налаштування і надалі працювати з системою без зайвих технічних дій.

Для організації зв'язку між програмною частиною гри та фізичним контролером використано сторонній Unreal Engine Serial Communication Plugin, розміщений у відкритому GitHub-репозиторії. Його застосування виправдане тим, що в межах UE5 потрібно забезпечити роботу з COM-портом, приймання даних від пристрою та передавання службових команд у зворотному напрямі. Використання готового засобу такого типу дозволяє інтегрувати послідовний обмін у середовище рушія без потреби самостійно створювати окрему низькорівневу підсистему для роботи з COM-портами. Для розроблюваної системи це є доцільним, оскільки взаємодія з фізичним пристроєм є обов'язковою частиною загальної логіки гри, а не зовнішнім допоміжним елементом [32].

Для побудови інтерфейсу користувача доцільно використати засоби Unreal Motion Graphics (UMG) і фреймворк Slate, що входять до складу Unreal Engine. Такий вибір є обґрунтованим, оскільки система містить не один екран, а кілька взаємопов'язаних інтерфейсних елементів: головне меню, екран калібрування, ігровий інтерфейс, екран завершення гри, а також засоби вибору COM-порту, розміру лабіринту й режиму камери. Використання вбудованих засобів рушія спрощує створення віджетів, керування переходами між екранами та узгодження інтерфейсної частини з ігровою логікою. Для цієї системи це важливо ще й тому, що користувач взаємодіє не лише з ігровою сценою, а й з меню налаштування та підготовки до гри [21].

Окремим важливим засобом підтримки користувацької взаємодії є локалізація інтерфейсу. У розроблюваній системі вона потрібна для того, щоб текстові елементи меню, повідомлення, підказки та службова інформація не були жорстко прив'язані до логіки окремих віджетів. Такий підхід спрощує зміну текстового наповнення, підтримку кількох мовних варіантів і подальше супроводження інтерфейсу. Для системи, у якій користувач проходить етапи підключення пристрою, калібрування, запуску гри та завершення сеансу, зрозумілість текстових повідомлень є не другорядною, а необхідною умовою зручного використання [33].

Отже, вибір засобів інтеграції та підтримки користувацької взаємодії узгоджується з потребами розроблюваної системи. Використання Serial Communication Plugin забезпечує зв'язок гри з фізичним пристроєм, засоби UMG і Slate дозволяють побудувати інтерфейс користувача, а механізми локалізації забезпечують зрозумілість текстових елементів інтерфейсу для користувача.

2.6 Висновки до розділу 2

У другому розділі сформовано проєктні основи вдосконаленої програмно-апаратної системи інтерактивної комп'ютерної гри з матеріальним інтерфейсом. На основі результатів аналізу, виконаного в першому розділі, було визначено вимоги до системи, окреслено обмеження її функціонування та уточнено напрям подальшого вдосконалення. Це дало змогу перейти від загальної постановки проблеми до цілеспрямованого проєктування структури системи, її функціональних складових і принципів роботи.

У розділі було визначено склад і загальну структуру системи, що охоплює фізичний контролер, програмну частину гри, канал обміну даними між ними, сенсорне керування, вібраційний зворотний зв'язок та інтерфейс користувача. Також було обґрунтовано поділ системи на функціональні підсистеми та показано їхню взаємодію в межах єдиного циклу роботи. У результаті сформовано цілісне уявлення про те, які частини повинні входити до системи та яку роль кожна з них виконує в забезпеченні ігрової взаємодії.

Окремим результатом другого розділу стало визначення й обґрунтування п'яти основних алгоритмічних блоків, що стали основою подальшої реалізації системи. До них належать алгоритм передавання, приймання та первинного оброблення сенсорних даних, алгоритм оброблення та інтерпретації IMU-даних, алгоритми керування сценою для імітації нахилу лабіринту, алгоритм процедурної генерації лабіринту та алгоритм формування просторового тактильного зворотного зв'язку. Саме ці блоки було визначено як такі, що забезпечують коректне проходження даних від фізичної дії користувача до реакції ігрового середовища та формування тактильного зворотного відгуку.

Важливим проєктним результатом стало обґрунтування підходу, за якого геометрія лабіринту залишається статичною, а його нахил імітується алгоритмічно. Такий підхід передбачає використання параметра керування,

штучного силового впливу на кульку, а також узгоджену роботу камери й освітлення для підтримки візуальної імітації нахилу. Це рішення стало одним із ключових у проєктуванні системи, оскільки дозволило відмовитися від постійної зміни трансформацій геометрії лабіринту та заклало основу для стабільнішої й менш ресурсоємної організації ігрової сцени.

Крім того, у другому розділі було обрано та обґрунтовано засоби розроблення, що відповідають вимогам системи. Для програмної частини виконано порівняння ігрових рушіїв Unreal Engine 5, Unity та Godot, за результатами якого визначено доцільність використання Unreal Engine 5 з урахуванням потреб розроблюваної системи та попередньої реалізації. Також обґрунтовано використання Blueprint, C++ та Visual Studio Tools for Unreal Engine, а для апаратної частини – Arduino Nano, MPU-6050 та Arduino IDE. Крім того, визначено доцільність використання Unreal Engine Serial Communication Plugin для послідовного обміну, а засобів UMG/Slate і механізмів локалізації – для побудови інтерфейсу користувача.

У результаті другого розділу сформовано завершену проєктну основу системи, що охоплює вимоги, структуру, функціональні підсистеми, алгоритмічні блоки та засоби реалізації. Саме ця основа стала підґрунтям для практичної реалізації системи в третьому розділі.

3 ПРОГРАМНО-АПАРАТНА РЕАЛІЗАЦІЯ СИСТЕМИ

У цьому розділі розглядається реалізація програмно-апаратної системи інтерактивної комп'ютерної гри, що поєднує фізичний контролер із сенсорним керуванням і програмну частину, створену в середовищі Unreal Engine 5. Якщо в попередньому розділі було подано проєктування системи, то далі увагу зосереджено саме на її практичній реалізації: складі фізичного контролера, способі його підключення до ПК, організації двостороннього обміну з грою та реалізованих програмних компонентах, що забезпечують функціонування системи в цілому.

3.1 Апаратна реалізація системи

Фізичний контролер реалізовано на базі плати Arduino Nano, інерціального модуля MPU-6050 та трьох вібраційних модулів з вбудованими вібротворами, розміщених у компактному корпусі-руків'ї. Така конфігурація дозволяє поєднати в одному пристрої зчитування IMU-вимірювань і відтворення просторового тактильного зворотного зв'язку. Arduino Nano забезпечує зчитування необроблених вимірювань акселерометра і гіроскопа, приймання службових команд від програмної частини та формування PWM-сигналів для керування вібраційними модулями. Підключення до ПК здійснюється через USB, що одночасно забезпечує живлення контролера і двосторонній обмін даними. Зовнішній вигляд реалізованого пристрою наведено на рис. 3.1.

Схему підключення компонентів фізичного контролера наведено на рис. 3.2. Інерціальний модуль MPU-6050 підключено до Arduino Nano через інтерфейс I2C, що використовується для обміну даними між МК і сенсором. Вібраційні модулі підключено до ліній живлення та окремих керувальних виводів Arduino Nano, на які подаються PWM-сигнали. У схемі також застосовано конденсатор, підключений до ліній живлення, що зменшує вплив

короткочасних просідань напруги під час спрацювання вібраційних модулів і підвищує стабільність роботи пристрою. У результаті апаратна частина забезпечує періодичне зчитування IMU-вимірювань, передавання їх до програмної частини та фізичне відтворення тактильного відгуку за командами, отриманими від гри.



Рисунок 3.1 – Зовнішній вигляд фізичного контролера інтерактивної системи

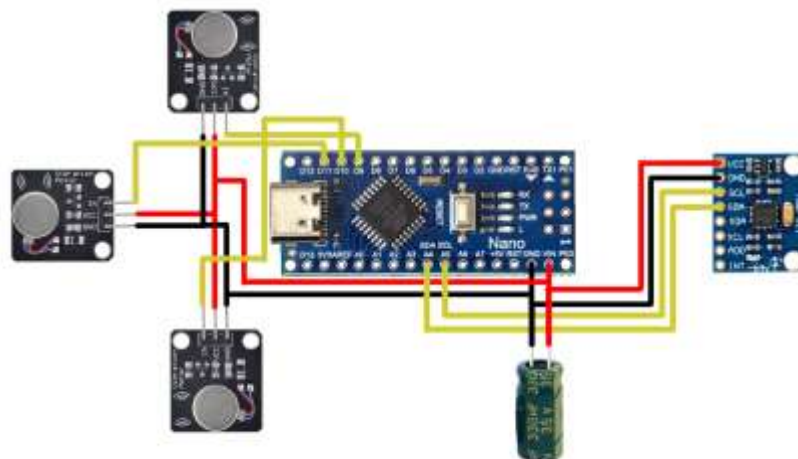


Рисунок 3.2 – Схема підключення компонентів фізичного контролера

Після підключення контролера до ПК обмін із грою здійснюється через COM-порт. У реалізованій системі послідовний обмін між фізичним

контролером і грою організовано за допомогою стороннього Unreal Engine Serial Communication Plugin. У програмній частині з ним взаємодіє блюпринт SerialReader, а розбір отриманих рядків, формування службових команд і отримання списку доступних COM-портів винесено в C++-клас SerialCommandProcessor. У межах такої взаємодії реалізовано передавання пакетів сенсорних даних, службових повідомлень, повідомлень про помилки, команду калібрування та команд керування вібраційними модулями. Детальніше формати повідомлень і логіку їх оброблення буде розглянуто в підрозділі 3.2.

Програмну частину системи реалізовано поєднанням Blueprint і C++. Технічні та службові частини, для яких важливими є чітка структура даних і окремі процедури оброблення, винесено в C++-класи, тоді як значну частину логіки взаємодії об'єктів сцени, станів гри та інтерфейсу користувача реалізовано засобами Blueprint. Для оброблення та інтерпретації IMU-даних створено C++-клас SensorDataInterpreter. Роботу із серійним обміном у програмній частині забезпечує блюпринт SerialReader, а розбір повідомлень і формування службових команд виконує C++-клас SerialCommandProcessor. Процедурну генерацію лабіринту реалізовано в окремому C++-класі MazeGeneratorActor, фізичну поведінку кульки – у блюпринті Ball, а загальну координату гри, інтерфейсу, калібрування, режимів камери та вібраційних подій – у блюпринті GameManager. Окремо реалізовано набір інтерфейсних віджетів, що забезпечують головне меню, екран калібрування, ігровий інтерфейс та меню завершення гри.

Отже, реалізована система поєднує фізичний контролер для зчитування IMU-вимірювань і відтворення просторового тактильного зворотного зв'язку, двосторонній послідовний обмін із програмною частиною та набір програмних компонентів, кожен з яких виконує визначену роль у загальному циклі роботи гри. Така структура створює основу для подальшого детального розгляду алгоритму передавання, приймання та первинного оброблення сенсорних даних, алгоритму оброблення та

інтерпретації IMU-даних, алгоритмів керування сценою для імітації нахилу лабіринту, алгоритму процедурної генерації лабіринту та алгоритму формування просторового тактильного зворотного зв'язку.

3.2 Розроблення алгоритмів передавання, приймання та первинного оброблення сенсорних даних

Алгоритми передавання, приймання та первинного оброблення сенсорних даних є першим етапом загального циклу функціонування розроблюваної системи. Вони забезпечують зв'язок між фізичним контролером і програмною частиною гри та створюють основу для подальшого алгоритму оброблення та інтерпретації IMU-даних. На цьому етапі ще не виконується визначення параметрів нахилу чи формування керувального впливу на сцену. Основне завдання цих алгоритмів полягає у своєчасному зчитуванні IMU-вимірювань, їх передаванні до програмної частини, прийманні через СОМ-порт, первинному розборі повідомлень і формуванні перевірених структурованих даних, придатних для подальшого використання.

На стороні фізичного контролера алгоритм починається з ініціалізації плати Arduino Nano, інерціального модуля MPU-6050, серійного порту та керувальних виводів для вібраційних модулів. Після цього контролер переходить у циклічний режим роботи, в межах якого одночасно виконує дві групи дій: опрацьовує команди, що надходять від програмної частини, і періодично формує пакет сенсорних даних. Якщо від гри надходить команда калібрування, контролер запускає процедуру накопичення вибірки вимірювань і визначення офсетів. Якщо надходить команда керування вібраційним модулем, контролер інтерпретує її параметри та формує відповідний PWM-сигнал на потрібному керувальному виводі. У разі відсутності службових команд основний цикл роботи не переривається, а контролер продовжує зчитування IMU-вимірювань.

Після отримання поточних значень акселерометра та гіроскопа до відповідних вимірювальних компонентів застосовуються офсети, визначені на етапі калібрування. Це дає змогу зменшити систематичне зміщення вимірювань ще до передавання в програмну частину. Далі формується текстовий пакет сенсорних даних у форматі D:ax,ay,az,gx,gy,gz, де перші три значення відповідають вимірюванням акселерометра, а наступні три – вимірюванням гіроскопа. Такий формат є достатньо простим для налагодження, не потребує складного двійкового протоколу і водночас забезпечує чітке розділення сенсорних даних і службових повідомлень. Передавання пакетів виконується з періодом близько 10 мс, тобто приблизно з частотою 100 Гц, що є достатнім для інтерактивного керування грою в реальному часі. Фрагменти реалізації цього етапу на стороні контролера наведено в додатку А.1.

Алгоритми передавання, приймання та первинного оброблення сенсорних даних охоплюють три взаємопов'язані частини: роботу фізичного контролера, серійний обмін між апаратною та програмною частинами та первинне оброблення повідомлень у програмній частині гри. Оскільки ці частини виконуються на різних рівнях системи, їх подано окремими схемами. Схему алгоритму роботи фізичного контролера наведено на рис. 3.3.

Блок 1. Початок. На цьому етапі запускається алгоритм роботи фізичного контролера.

Блок 2. Ініціалізація. Виконується підготовка Arduino Nano, інерціального модуля MPU-6050, послідовного інтерфейсу Serial та керувальних виводів до подальшої роботи.

Блок 3. Перевірка команди. МК перевіряє, чи надійшла від програмної частини службова команда, що потребує оброблення.

Блок 4. Обробка команди. Якщо команда отримана, виконується її інтерпретація, зокрема команда калібрування або команда керування вібраційним модулем.



Рисунок 3.3 – Схема алгоритму роботи фізичного контролера

Блок 5. Зчитування ІМУ. Апаратна частина отримує поточні значення акселерометра та гіроскопа з інерціального модуля.

Блок 6. Застосування офсетів. До отриманих вимірювань застосовуються калібрувальні зміщення, що дає змогу зменшити вплив початкового відхилення сенсора.

Блок 7. Формування пакета. З підготовлених значень формується текстовий пакет сенсорних даних у форматі D:ax,ay,az,gx,gy,gz.

Блок 8. Передавання пакета. Сформований пакет надсилається через послідовний інтерфейс до програмної частини гри.

Блок 9. Перевірка продовження роботи. Алгоритм визначає, чи має фізичний контролер перейти до наступної ітерації роботи. У фактичній прошивці повторення забезпечується основним циклом МК, а припинення роботи можливе лише після вимкнення живлення або перезапуску пристрою.

Блок 10. Кінець. Завершення алгоритму відповідає припиненню роботи фізичного контролера у випадку вимкнення живлення або перезапуску.

Сформований пакет передається до програмної частини через COM-порт, який використовується як канал двостороннього серійного обміну. Схему цього етапу наведено на рис. 3.4.



Рисунок 3.4 – Схема алгоритму серійного обміну між апаратною та програмною частинами

Блок 1. Початок. На цьому етапі починається передавання окремого пакета між апаратною та програмною частинами системи.

Блок 2. Передавання пакета. Пакет сенсорних даних проходить через канал серійного обміну від фізичного контролера до ПК.

Блок 3. Приймання пакета. Програмна частина отримує переданий пакет і робить його доступним для подальшого зчитування.

Блок 4. Кінець. Передавання окремого пакета через канал серійного обміну вважається завершеним.

На стороні програмної частини приймання рядків організовано за допомогою стороннього Unreal Engine Serial Communication Plugin, з яким взаємодіє бібліотека SerialReader. Його роль полягає не в інтерпретації змісту повідомлення, а в зчитуванні доступних рядків із буфера COM-порту та передаванні їх далі в логіку первинного оброблення. Схему алгоритму приймання та первинного оброблення повідомлень у програмній частині наведено на рис. 3.5.

Блок 1. Початок. На цьому етапі запускається логіка приймання та первинного розбору повідомлень у програмній частині гри.

Блок 2. Зчитування рядків. Компонент SerialReader отримує наявні рядки з буфера послідовного обміну, щоб не обмежуватися лише одним повідомленням за цикл.

Блок 3. Передавання рядка. Отриманий рядок передається до C++-класу SerialCommandProcessor, який відповідає за класифікацію повідомлення та виділення корисних даних.

Блок 4. Перевірка повідомлення D:. Якщо рядок починається з префікса D:, він розглядається як пакет IMU-вимірювань.

Блок 5. Формування структури сенсорних даних. У разі коректного пакета з рядка виділяються значення акселерометра та гіроскопа, після чого формується внутрішня структура сенсорних даних.

Блок 6. Передавання валідних даних. Перевірені та структуровані IMU-дані передаються на етап оброблення та інтерпретації.

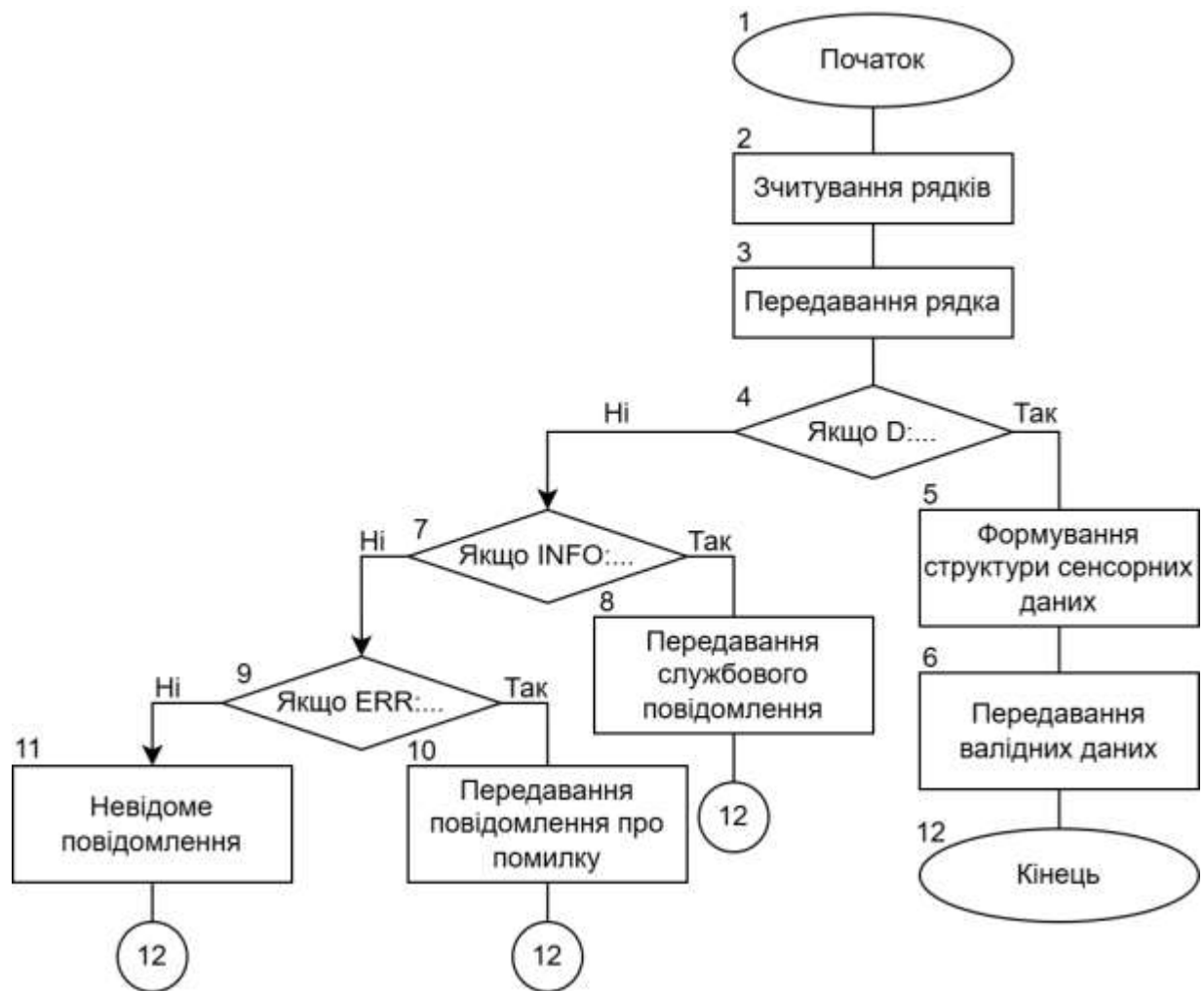


Рисунок 3.5 – Схема алгоритму приймання та первинного оброблення сенсорних даних у програмній частині

Блок 7. Перевірка повідомлення INFO:. Якщо повідомлення не є пакетом сенсорних даних, перевіряється наявність префікса INFO:.

Блок 8. Передавання службового повідомлення. У разі виявлення службового повідомлення воно передається до відповідної логіки програмної частини для подальшого використання або відображення.

Блок 9. Перевірка повідомлення ERR:. Якщо рядок не належить до службових повідомлень, виконується перевірка префікса ERR:.

Блок 10. Передавання повідомлення про помилку. У разі виявлення повідомлення про помилку воно передається до програмної частини для оброблення або інформування користувача.

Блок 11. Оброблення невідомого повідомлення. Якщо рядок не відповідає жодному з очікуваних форматів, він класифікується як невідоме повідомлення і не використовується як сенсорні дані.

Блок 12. Кінець. Після класифікації або передавання даних поточний цикл оброблення повідомлення завершується.

Важливою особливістю реалізованого алгоритму є те, що приймання виконується не як опрацювання одного рядка за один ігровий кадр, а шляхом зчитування поточного вмісту буфера з подальшим послідовним розбором отриманих рядків. Щоб така процедура не блокувала виконання ігрового кадру в разі збою або надмірного накопичення повідомлень, кількість рядків, що обробляються за один цикл, обмежується встановленим лімітом. Це рішення є принципово важливим для інтерактивної системи з сенсорним керуванням, оскільки дозволяє використовувати IMU-вимірювання, ближчі до поточного стану контролера, і зменшує ризик накопичення затримки під час тривалого сеансу гри. У порівнянні з попереднім варіантом системи така логіка приймання стала більш стійкою до нерівномірного надходження даних і краще узгоджується з вимогами реального часу.

Після приймання рядки передаються в C++-клас `SerialCommandProcessor`, який виконує їх первинний розбір. Саме на цьому етапі відбувається класифікація повідомлень за типом і перевірка того, чи є конкретний рядок пакетом сенсорних даних, службовим інформаційним повідомленням або повідомленням про помилку. Якщо рядок починається з префікса `D:`, він інтерпретується як пакет сенсорних даних. У такому разі виконується розбиття рядка на окремі числові значення, перевіряється їхня кількість, після чого формується структура `FSerialSensorData`, що містить поля акселерометра, гіроскопа та ознаку коректності пакета. Якщо рядок починається з префікса `INFO:`, він обробляється як службове повідомлення, зокрема може повідомляти про завершення калібрування. Якщо ж використовується префікс `ERR:`, повідомлення передається в програмну

логіку як сигнал про помилку або некоректний стан пристрою. Реалізацію цього етапу наведено в додатках А.2–А.3.

Таким чином, первинне оброблення вхідних повідомлень охоплює не лише розбір пакета сенсорних даних, а й опрацювання допоміжних службових рядків, що забезпечують повноцінний цикл взаємодії між грою і фізичним контролером. Саме завдяки цьому на наступний алгоритмічний етап передаються не довільні рядки з СОМ-порту, а вже структуровані й перевірені дані, придатні для подальшого оброблення та інтерпретації.

Окрему роль у межах цього алгоритму відіграє калібрування. З боку гри воно ініціюється передаванням короткої службової команди CAL, яка надходить до фізичного контролера через той самий СОМ-порт. Після її отримання контролер переходить у режим накопичення вибірки вимірювань, обчислює середні значення для акселерометра і гіроскопа та формує офсети, що використовуються в подальшому циклі зчитування. Після завершення цієї процедури контролер надсилає службове інформаційне повідомлення, яке дає програмній частині змогу оновити стан системи та перейти до нормального режиму керування. Отже, калібрування в межах алгоритму передавання, приймання та первинного оброблення сенсорних даних розглядається не як окремий ізольований процес, а як складова загальної логіки обміну між фізичним контролером і грою.

Крім команди калібрування, через серійний канал передаються і команди керування вібраційними модулями. Вони формуються програмною частиною після виникнення відповідних подій у грі, а далі надходять до контролера у вигляді коротких службових рядків. На стороні Arduino Nano ці команди розбираються, після чого контролер формує PWM-сигнали для вибраних вібраційних модулів. Хоча логіка визначення сили та просторового розподілу тактильного відгуку належить до окремого алгоритму, уже на цьому етапі важливо, що канал серійного обміну використовується не лише для передавання ІМУ-вимірювань від пристрою до гри, а й для надсилання керувальних команд у зворотному напрямі.

Формати повідомлень і команд, реалізованих у системі серійного обміну, наведено в табл. 3.1.

Таблиця 3.1 – Формати повідомлень і команд серійного обміну в системі

Формат повідомлення	Напрямок передавання	Призначення
D:ax,ay,az,gx,gy,gz	Від фізичного контролера до гри	Передавання пакета сенсорних даних акселерометра і гіроскопа
INFO:...	Від фізичного контролера до гри	Передавання службового інформаційного повідомлення, зокрема про завершення калібрування
ERR:...	Від фізичного контролера до гри	Передавання повідомлення про помилку або некоректний стан
CAL	Від гри до фізичного контролера	Запуск процедури калібрування
M:<motor>:<value>	Від гри до фізичного контролера	Керування вибраним вібраційним модулем, де motor – номер модуля, value – рівень PWM-сигналу

Узагальнюючи, алгоритми передавання, приймання та первинного оброблення сенсорних даних забезпечують початковий цикл взаємодії між фізичним контролером і програмною частиною гри. На стороні Arduino Nano вони охоплюють зчитування IMU-вимірювань, застосування офсетів, формування пакета сенсорних даних і опрацювання службових команд. На стороні гри вони включають приймання рядків через COM-порт у блюпринті SerialReader, передавання їх до C++-класу SerialCommandProcessor, класифікацію повідомлень і формування валідної структури сенсорних даних. У результаті наступний етап системи отримує перевірені та структуровані IMU-вимірювання, що краще відповідають поточному стану фізичного контролера, на основі яких далі виконується алгоритм оброблення та інтерпретації IMU-даних.

3.3 Розроблення алгоритму оброблення та інтерпретації IMU-даних

Після алгоритму передавання, приймання та первинного оброблення сенсорних даних програмна частина системи отримує коректні, перевірені та структуровані IMU-вимірювання. Проте ці дані ще не можна безпосередньо використовувати в алгоритмах керування сценою для імітації нахилу лабіринту, оскільки вони містять шум, потребують масштабування, згладжування, узгодження осей і поєднання оцінок, отриманих від акселерометра та гіроскопа. Тому в системі реалізовано окремий алгоритм оброблення та інтерпретації IMU-даних, призначення якого полягає у формуванні стабільних параметрів нахилу на основі вхідних IMU-вимірювань [3].

На вхід алгоритму надходить структура IMU-даних із числовими значеннями акселерометра і гіроскопа. Перед основним оцінюванням орієнтації ці значення перетворюються у векторне подання, узгоджуються з координатною системою гри та масштабуються відповідно до діапазонів вимірювання сенсора. Необхідність узгодження осей пов'язана з тим, що фізична орієнтація MPU-6050 у контролері не збігається безпосередньо з координатною системою UE5. Тому під час перетворення вхідних значень у вектори враховується відповідність осей сенсора і осей, які далі використовуються в логіці гри.

Масштабування виконується відповідно до налаштувань діапазонів вимірювання MPU-6050, використаних у реалізації системи. Сам модуль підтримує кілька діапазонів вимірювання для акселерометра та гіроскопа [30]. У цій реалізації застосовано коефіцієнти $S_a = 1/16384$ для акселерометра та $S_g = 1/131$ для гіроскопа. Масштабування вхідних вимірювань виконується за формулою:

$$a_i = a_{raw,i} \times S_a, \omega_i = \omega_{raw,i} \times S_g,$$

де a_i – масштабоване значення прискорення за відповідною віссю;

$a_{raw,i}$ – необроблене значення акселерометра;

ω_i – кутова швидкість за відповідною віссю;

$\omega_{raw,i}$ – необроблене значення гіроскопа;

$S_a = 1/16384$ – масштабний коефіцієнт акселерометра;

$S_g = 1/131$ – масштабний коефіцієнт гіроскопа.

У реалізації ці перетворення виконуються в C++-класі `SensorDataInterpreter`, де дані акселерометра переводяться у вектор прискорення, а дані гіроскопа – у вектор кутових швидкостей. Цей етап не визначає нахил сам по собі, але приводить IMU-вимірювання до узгодженого формату, придатного для подальшої фільтрації та поєднання. Реалізацію масштабування наведено в додатках А.4–А.5.

Оскільки необроблені IMU-вимірювання містять дрібні коливання і короткочасні сплески, перед основним оцінюванням орієнтації в алгоритмі застосовується попереднє згладжування методом експоненційного ковзного середнього. Його використання дає змогу зменшити вплив високочастотних коливань без істотного ускладнення обчислювальної частини алгоритму. ЕМА-згладжування не є фінальним способом визначення нахилу, а виконує допоміжну функцію підготовки сигналу до наступного етапу. Узагальнено таке згладжування описується формулою:

$$x_t = \alpha_{EMA} \times x_{new} + (1 - \alpha_{EMA}) \times x_{t-1},$$

де x_t – згладжене значення на поточному кроці;

x_{new} – нове вхідне значення;

x_{t-1} – попереднє згладжене значення;

α_{EMA} – коефіцієнт експоненційного ковзного середнього.

У реалізації алгоритму використано різні коефіцієнти ЕМА для даних акселерометра і гіроскопа: $\alpha_{acc} = 0,15$ для акселерометра та $\alpha_{gyro} = 0,10$ для гіроскопа. Значення цих коефіцієнтів підбрано з урахуванням компромісу між зменшенням дрібних коливань сигналу та збереженням прийнятної швидкості реакції системи.

Після попереднього згладжування алгоритм переходить до оцінювання орієнтації. Акселерометр дає змогу визначити нахил відносно вектора тяжіння, але його оцінка чутлива до сторонніх прискорень і вібрацій. Гіроскоп, навпаки, добре відстежує швидкі зміни орієнтації, однак під час інтегрування кутової швидкості з часом може накопичувати похибку. Саме тому для формування стабільних параметрів нахилу використовується поєднання обох джерел інформації. Такий підхід відповідає загальній логіці злиття сенсорних даних для IMU-сенсорів, де дані акселерометра і гіроскопа доповнюють одне одного [3], [14].

У реалізації спочатку обчислюються кути Pitch і Roll за даними акселерометра, після чого виконується інтегрування кутової швидкості гіроскопа за відповідними осями. Далі ці оцінки поєднуються за допомогою комплементарного фільтра. Комплементарний фільтр дає змогу об'єднати інтегровану гіроскопічну складову з оцінкою кута за акселерометром, зберігаючи швидку реакцію на зміну положення контролера і водночас зменшуючи довготривале накопичення похибки [14]-[15]. У загальному вигляді така операція може бути подана формулою:

$$\theta_t = \beta(\theta_{t-1} + \omega\Delta t) + (1 - \beta)\theta_{acc},$$

де θ_t – результуюче значення кута нахилу на поточному кроці;

θ_{t-1} – попереднє значення кута;

ω – кутова швидкість за даними гіроскопа;

Δt – проміжок часу між оновленнями;

θ_{acc} – кут нахилу, обчислений за даними акселерометра;

β – коефіцієнт комплементарного фільтра.

У реалізації використовується значення $\beta=0,96$, що надає більшої ваги гіроскопічній складовій, але водночас поступово коригує результат за даними акселерометра. Завдяки цьому вдається отримати параметри нахилу, які достатньо швидко реагують на рух користувача і при цьому не втрачають довготривалої стійкості.

Після фільтрації сформовані значення Pitch і Roll обмежуються допустимими межами, щоб надмірний нахил контролера не створював нестабільної реакції в наступному алгоритмі. У реалізованій сцені для цього використано параметр $\text{MaxTiltAngle} = 40^\circ$, який задає максимальне значення нахилу для подальшого використання в логіці керування. У результаті алгоритм формує стабільні параметри нахилу, які передаються далі в алгоритми керування сценою для імітації нахилу лабіринту.

Схему алгоритму оброблення та інтерпретації IMU-даних наведено на рис. 3.6.

Блок 1. Початок виконання алгоритму. На цьому етапі запускається алгоритм оброблення та інтерпретації IMU-даних, отриманих від попереднього алгоритмічного блоку.

Блок 2. Отримання структури IMU-даних. Алгоритм приймає перевірені та структуровані значення акселерометра й гіроскопа, сформовані після первинного оброблення вхідних повідомлень.

Блок 3. Перетворення значень акселерометра та гіроскопа у вектори. Отримані числові значення групуються у векторні представлення, що спрощує подальше опрацювання вимірювань.

Блок 4. Узгодження осей із координатною системою гри. На цьому етапі виконується приведення напрямів вимірювань до координатної системи, у якій працює ігрова логіка.

Блок 5. Масштабування вимірювань. Значення акселерометра та гіроскопа приводяться до масштабу, придатного для подальших обчислень і використання в системі керування.

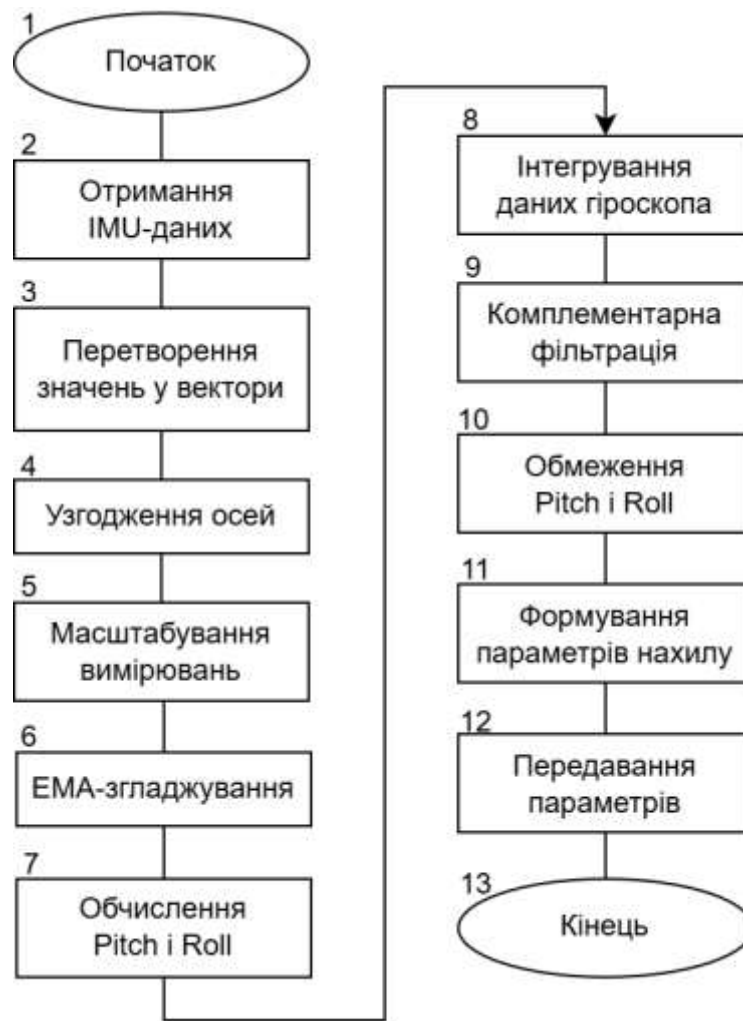


Рисунок 3.6 – Схема алгоритму оброблення та інтерпретації IMU-даних

Блок 6. Застосування ЕМА-згладжування. До підготовлених вимірювань застосовується експоненційне ковзне середнє, що дає змогу зменшити вплив короточасних коливань і шуму.

Блок 7. Обчислення кутів Pitch і Roll за акселерометром. На основі згладжених значень акселерометра визначаються оцінки кутів нахилу відносно відповідних осей.

Блок 8. Інтегрування кутової швидкості гіроскопа. Дані гіроскопа використовуються для врахування швидких змін орієнтації контролера під час руху.

Блок 9. Поєднання оцінок комплементарним фільтром. Оцінки, отримані за акселерометром і гіроскопом, об'єднуються для формування більш стійкого результату, що поєднує переваги обох типів вимірювань.

Блок 10. Обмеження Pitch і Roll допустимими межами. Обчислені кути нахилу обмежуються встановленими межами, щоб уникнути надмірних або некоректних значень керувального параметра.

Блок 11. Формування стабільних параметрів нахилу. Після фільтрації та обмеження формуються підготовлені параметри нахилу, які можуть бути використані в ігровій логіці.

Блок 12. Передавання параметрів в алгоритми керування сценою. Сформовані параметри нахилу передаються до наступного алгоритмічного блоку, який відповідає за імітацію нахилу лабіринту та формування умов руху кульки.

Блок 13. Завершення виконання алгоритму. Після передавання параметрів оброблення поточного набору IMU-даних завершується.

Основні параметри алгоритму оброблення та інтерпретації IMU-даних наведено в табл. 3.2.

Таблиця 3.2 – Основні параметри алгоритму оброблення та інтерпретації IMU-даних

Параметр	Позначення / значення	Призначення
Масштабний коефіцієнт акселерометра	$S_a = 1/16384$	Перетворення необроблених значень акселерометра у масштабовані значення
Масштабний коефіцієнт гіроскопа	$S_g = 1/131$	Перетворення значень гіроскопа у кутову швидкість
Коефіцієнт ЕМА для акселерометра	$\alpha_{acc} = 0,15$	Попереднє згладжування вимірювань акселерометра
Коефіцієнт ЕМА для гіроскопа	$\alpha_{gyro} = 0,10$	Попереднє згладжування вимірювань гіроскопа
Коефіцієнт комплементарного фільтра	$\beta = 0,96$	Визначає вагу гіроскопічної складової під час поєднання з оцінкою акселерометра
Крок часу	Δt	Інтервал між оновленнями, що використовується для інтегрування кутової швидкості
Обмеження кута нахилу	$MaxTiltAngle = 40^\circ$	Обмеження надмірних значень Pitch і Roll

Уся логіка цього алгоритму реалізована в C++-класі SensorDataInterpreter. У ньому виконуються перетворення вхідних значень, узгодження осей із координатною системою гри, масштабування, ЕМА-

згладжування, оцінювання орієнтації за допомогою комплементарного фільтра, обмеження значень Pitch і Roll та формування кінцевих параметрів нахилу. Реалізацію C++-класу `SensorDataInterpreter` наведено в додатках А.4–А.5.

Отже, алгоритм оброблення та інтерпретації IMU-даних перетворює коректні, перевірені та структуровані IMU-вимірювання на стабільні параметри нахилу, придатні для подальшого використання в ігровій логіці. Саме ці параметри далі передаються в алгоритми керування сценою для імітації нахилу лабіринту, де на їх основі формується керувальний вплив на віртуальне середовище.

3.4 Розроблення алгоритмів керування сценою для імітації нахилу лабіринту

Алгоритми керування сценою для імітації нахилу лабіринту є одним із ключових алгоритмічних блоків розроблюваної системи. Саме в них реалізовано головне вдосконалення фінальної версії гри: відмову від підходу, за якого зміна нахилу відтворювалася через постійне оновлення трансформацій геометрії лабіринту. У новій реалізації геометрія лабіринту залишається статичною, а сам нахил імітується алгоритмічно. Це дає змогу формувати умови руху кульки через формування та зміну напрямку штучного вектора сили, підтримувати узгоджене візуальне сприйняття сцени та уникати зайвого обчислювального навантаження, пов'язаного з постійною зміною великої кількості об'єктів сцени.

Після алгоритму оброблення та інтерпретації IMU-даних система отримує стабільні параметри нахилу, які надходять до блоку `GameManager`. У практичній реалізації саме тут формується параметр керування `Virtual Tilt Rotation`, який виконує роль узагальненого параметра імітованого нахилу лабіринту. Назва функції `RotateMaze` збереглася з попередньої логіки проєкту, однак у фінальній реалізації вона не виконує

фізичного обертання геометрії лабіринту. Її призначення полягає в отриманні стабільних параметрів нахилу з `SensorDataInterpreter`, записі їх у `Virtual Tilt Rotation` та подальшій координації фізичного й візуального відтворення імітованого нахилу. У реалізації бюпринта `GameManager` для цього передбачено змінну `Virtual Tilt Rotation`, функцію `RotateMaze`, а також окрему логіку керування камерою та освітленням.

Оновлення параметрів імітації нахилу виконується лише під час активного ігрового стану. У бюпринті `GameManager` це контролюється через перевірку прапорця `PausedGame`, тому під час паузи або службових станів гри параметри імітованого нахилу не повинні змінювати поведінку сцени. Такий підхід дозволяє уникнути ситуацій, коли кулька, камера чи освітлення продовжують реагувати на входні IMU-вимірювання в той момент, коли ігровий процес зупинено. Це робить загальну поведінку системи передбачуванішою і краще узгоджує логіку керування з інтерфейсними станами гри.

Після формування `Virtual Tilt Rotation` система послідовно використовує цей параметр для оновлення фізичної та візуальної складових сцени: обчислення штучного вектора сили для кульки, коригування параметрів камери та зміни орієнтації джерела світла. Схему алгоритму наведено на рис. 3.7.

Блок 1. Початок. На цьому етапі запускається логіка керування сценою, що використовує результати оброблення та інтерпретації IMU-даних.

Блок 2. Отримання параметрів нахилу. Програмна частина отримує стабільні параметри нахилу, сформовані попереднім алгоритмом у модулі `SensorDataInterpreter`.



Рисунок 3.7 – Схема алгоритму керування сценою для імітації нахилу лабіринту

Блок 3. Формування Virtual Tilt Rotation. На основі отриманих параметрів нахилу створюється керувальний параметр імітованого нахилу лабіринту.

Блок 4. Обчислення штучного вектора сили. На основі Virtual Tilt Rotation визначається напрямок сили, яка має впливати на кульку відповідно до імітованого нахилу лабіринту.

Блок 5. Передавання вектора сили до Ball. Обчислені параметри керувального впливу передаються до логіки кульки, яка відповідає за її подальший рух у сцені.

Блок 6. Застосування сили до кульки. До кульки застосовується штучний силовий вплив, що формує її рух без потреби змінювати трансформації геометрії лабіринту.

Блок 7. Визначення режиму камери. Система визначає активний режим камери, відповідно до якого має оновлюватися її положення та орієнтація.

Блок 8. Обчислення положення й обертання камери. На основі активного режиму камери та параметра Virtual Tilt Rotation визначаються цільові параметри камери в ігровій сцені.

Блок 9. Інтерполяція руху камери. Зміна положення та обертання камери виконується плавно, щоб візуальна реакція сцени не виглядала різкою або нестабільною.

Блок 10. Формування обертання джерела світла. Для підтримки візуального ефекту нахилу обчислюється зміна орієнтації джерела освітлення відповідно до стану імітованого нахилу.

Блок 11. Поєднання з базовою орієнтацією світла. Обчислене обертання узгоджується з початковою орієнтацією джерела світла, щоб зберегти коректне освітлення сцени.

Блок 12. Оновлення Directional Light. Після обчислення нових параметрів виконується оновлення напрямку джерела освітлення в ігровій сцені.

Блок 13. Кінець. Після оновлення фізичної, камерної та освітлювальної складових поточний цикл керування сценою завершується.

У схемі ці дії подано як послідовні етапи одного циклу оновлення сцени. Далі окремо розглянуто фізичну, камерну та освітлювальну складові цього алгоритму.

Перший напрямок пов'язаний із формуванням фізичного впливу на кульку. На основі Virtual Tilt Rotation у GameManager обчислюється напрямок штучного вектора сили, який імітує дію сили тяжіння в умовах нахиленого лабіринту. Для цього базовий напрямок «вниз» перетворюється відповідно до поточного параметра керування, після чого разом із величиною Gravity Strength передається до блюпринта Ball. Таким чином, у фінальній реалізації зміна поведінки кульки досягається не зміною геометрії лабіринту,

має підтримуватися не лише фізичною реакцією кульки, а й візуальним сприйняттям сцени. Для цього в GameManager реалізовано окрему логіку керування камерою. У системі підтримується кілька режимів камери, але в усіх випадках її робота узгоджується з параметром керування Virtual Tilt Rotation. На його основі коригуються положення та орієнтація камери, після чого перехід до потрібного стану виконується плавно. Завдяки цьому камера не просто спостерігає за сценою, а бере участь у відтворенні ілюзії нахилу лабіринту.

Третій напрямок пов'язаний з освітленням. У GameManager зберігається базова орієнтація основного джерела світла, після чого до неї додається поточний параметр Virtual Tilt Rotation. У результаті орієнтація Directional Light оновлюється відповідно до стану імітованого нахилу. Це важливо не як декоративний ефект, а як спосіб узгодити візуальне сприйняття сцени з фізичною поведінкою кульки. Якщо кулька рухається так, ніби лабіринт нахилено, але світло й камера не узгоджуються зі станом імітованого нахилу, у користувача виникає відчуття розриву між логікою руху й виглядом сцени. Узгоджене оновлення освітлення допомагає уникнути такого розриву і підтримує цілісність сприйняття ігрового середовища. На фрагменті RotateMaze, наведеному на рис. 3.8, видно, що орієнтація Directional Light оновлюється в межах того самого циклу, в якому формується параметр Virtual Tilt Rotation і передається керувальний вплив до Ball.

Основна частина цього алгоритмічного блоку реалізована у блюпринтах GameManager і Ball. GameManager формує параметр керування Virtual Tilt Rotation, координує логіку камери й освітлення та передає напрямок штучного вектора сили до Ball. Блюпринт Ball, у свою чергу, забезпечує прикладення цього керувального впливу до фізичного тіла кульки. Такий розподіл ролей дозволяє відокремити загальну координацію імітації нахилу від безпосередньої фізичної реакції ігрового об'єкта.

Отже, алгоритми керування сценою для імітації нахилу лабіринту реалізують нову архітектурну схему взаємодії в системі. Геометрія лабіринту залишається статичною, а сам нахил імітується через параметр керування Virtual Tilt Rotation, штучний вектор сили, оновлення камери та узгоджену зміну орієнтації джерела світла. Такий підхід дозволяє відмовитися від постійної зміни трансформацій геометрії лабіринту, зменшити кількість зайвих перетворень у сцені та зробити поведінку системи стабільнішою й передбачуванішою.

3.5 Розроблення алгоритму процедурної генерації лабіринту

Алгоритм процедурної генерації лабіринту призначений для формування ігрового середовища перед початком проходження рівня. Його роль у системі не обмежується лише побудовою випадкової структури проходів. У межах цього алгоритму формується повна логічна модель лабіринту, яка далі перетворюється на геометрію сцени, доповнюється стартовою зоною, виходом і отворами та підготовлюється до використання в грі. Основою генерації проходів у реалізації є алгоритм обходу в глибину, який використовується для побудови лабіринтів методом рекурсивного повернення і дає змогу отримувати варіативну зв'язну структуру проходів [17].

У C++-класі MazeGeneratorActor лабіринт спочатку подається не як набір мешів, а як двовимірний масив MazeGrid, який виконує роль проміжного логічного подання. Кожна клітина цього масиву має певний стан, що визначає її призначення в структурі лабіринту. У реалізації використано чотири основні стани: прохід або підлога, стіна, отвір і вихід. Таке подання є зручним, оскільки дозволяє спочатку сформувати внутрішню логіку лабіринту, а вже потім, на її основі, побудувати геометрію сцени. Основні стани клітин внутрішньої структури лабіринту наведено в табл. 3.3.

Таблиця 3.3 – Основні стани клітин внутрішньої структури лабіринту

Стан клітини	Позначення	Призначення в логічній моделі	Геометричне відображення
Прохід / підлога	0	Клітина, доступна для руху кульки	Екземпляр підлоги
Стіна	1	Перешкода, що формує структуру лабіринту	Екземпляр стіни та підлоги під нею
Отвір	2	Небезпечна зона, потрапляння в яку призводить до падіння кульки	Екземпляр отвору
Вихід	3	Клітина завершення проходження лабіринту	Підлога та фінішний актор із колізійним об'ємом

Користувач задає розмір лабіринту у вигляді параметра n , після чого у внутрішньому поданні формується сітка розміром $2n + 1$. Такий підхід забезпечує непарну розмірність сітки, наявність центральної клітини та правильне чергування стін і проходів. У реалізованій системі розмір лабіринту обмежується діапазоном від 5 до 15. Після визначення розміру формується масив MazeGrid, який спочатку повністю заповнюється стінами. Це створює початкову основу, на якій далі виконується генерація проходів.

Побудова проходів виконується методом обходу в глибину. Як стартова точка використовується центральна клітина лабіринту, що надалі також є основою для стартової області користувача. На кожному кроці алгоритм визначає сусідні невідвідані клітини, випадково переміщує напрями обходу, обирає наступну клітину та відкриває прохід між нею і поточною позицією. Після цього процедура повторюється рекурсивно, поки не буде вичерпано всі можливі маршрути. Використання випадкового перемішування напрямків дозволяє формувати різні конфігурації лабіринту навіть за однакового розміру, що і забезпечує варіативність рівнів [17].

Проте у реалізації алгоритм не завершується на етапі побудови проходів. Після базової DFS-генерації виконується додаткове доопрацювання структури лабіринту, пов'язане з особливостями ігрового процесу. Першим таким етапом є очищення стартової зони розміром 3×3 у центрі лабіринту. Це робиться для того, щоб початок гри не був надто стисненим або випадково невдалим для користувача. Очищення центральної області полегшує початкову орієнтацію, а також частково порушує строго деревоподібну

структуру, отриману після DFS, завдяки чому навколо стартової позиції з'являється більше локальних варіантів руху.

Наступним етапом є формування виходу з лабіринту. У реалізації вихід визначається на зовнішній межі лабіринту, але не довільно, а з урахуванням того, щоб внутрішня сусідня клітина вже належала до структури проходів. Завдяки цьому вихід пов'язується з уже згенерованою системою маршрутів і не утворює ізольованої області, недоступної для проходження. Після логічного позначення клітини виходу в масиві MazeGrid під час побудови сцени в цій позиції додатково розміщується окремий фінішний актор із колізійним об'ємом.

Після визначення виходу до структури лабіринту додаються отвори. Вони не є частиною базової DFS-генерації, а накладаються на вже побудовану структуру як додатковий ігровий шар. У реалізації клітина-стіна може бути перетворена на отвір приблизно з імовірністю 5%, а клітина-прохід – приблизно з імовірністю 0,5%. Таке співвідношення обрано з урахуванням того, що проходи безпосередньо формують маршрут гравця, тому поява отворів у них має бути обмеженою, щоб не робити проходження надмірно випадковим або ускладненим. Натомість перетворення частини стінових клітин на отвори додає небезпечні зони й візуальну варіативність без істотного порушення прохідності рівня. Завдяки цьому отвори виступають як елементи ризику, що ускладнюють проходження лабіринту, але не розглядаються як гарантоване повне блокування маршруту. Розміри отворів і кульки підібрані так, щоб вони не створювали суцільного блокування проходу, але створювали реальну небезпеку падіння у разі неточного керування.

Після завершення логічної генерації MazeGrid перетворюється на геометрію сцени. Для повторюваних елементів рівня в MazeGeneratorActor використано компоненти типу InstanceStaticMeshComponent, що дозволяє розміщувати велику кількість однотипних екземплярів підлоги, стін і отворів без ручного створення кожного об'єкта окремо. На етапі ініціалізації в

BeginPlay для відповідних інстансерів задаються меші підлоги, стін і отворів, після чого в процесі генерації відбувається розміщення екземплярів відповідно до станів клітин. Для всіх клітин, що не є отворами, створюються екземпляри підлоги. Для клітин зі станом стіни додаються екземпляри стін, а для клітин зі станом отвору – екземпляри отворів. Клітина виходу додатково доповнюється окремим фінішним актором із колізійним об'ємом.

Схему алгоритму процедурної генерації лабіринту, що включає логічну генерацію та побудову геометрії сцени, наведено на рис. 3.9.



Рисунок 3.9 – Схема алгоритму процедурної генерації лабіринту

Блок 1. Початок виконання алгоритму. На цьому етапі запускається процедура формування нового лабіринту для подальшого використання в ігровій сцені.

Блок 2. Очищення попередніх екземплярів лабіринту. Перед створенням нового рівня з ігрової сцени видаляються або скидаються елементи попередньо згенерованого лабіринту, щоб уникнути накладання старої та нової геометрії.

Блок 3. Задання розміру лабіринту. Алгоритм отримує параметр розміру лабіринту, який визначає кількість логічних клітин і впливає на подальше формування структури рівня.

Блок 4. Формування сітки розміром $2n + 1$. На основі вибраного розміру створюється внутрішня сіткова структура, у якій враховуються як клітини проходів, так і проміжні елементи між ними.

Блок 5. Заповнення сітки стінами. Початковий стан сітки формується як заповнений стінами простір, у якому проходи будуть створюватися на наступних етапах алгоритму.

Блок 6. Вибір центральної стартової клітини. Для початку генерації визначається стартова клітина, з якої розпочинається побудова проходів лабіринту.

Блок 7. Генерація проходів методом DFS. Алгоритм обходу в глибину використовується для послідовного формування зв'язної структури проходів між клітинами лабіринту.

Блок 8. Очищення стартової зони 3×3 . Після формування основних проходів у центральній частині лабіринту очищується стартова зона, щоб забезпечити коректне початкове розміщення кульки та зручний початок проходження рівня.

Блок 9. Визначення виходу на зовнішній межі. Алгоритм обирає місце виходу з лабіринту на зовнішній межі сформованої структури.

Блок 10. Додавання отворів до структури лабіринту. До згенерованої структури додаються отвори, які виконують роль додаткових ігрових перешкод під час проходження рівня.

Блок 11. Побудова геометрії за станами клітин. Логічна структура лабіринту перетворюється на геометрію ігрової сцени: відповідно до станів клітин створюються підлога, стіни, отвори та інші елементи рівня.

Блок 12. Додавання проміжних стінок між суміжними стіновими клітинами. На цьому етапі уточнюється геометрія стін, щоб забезпечити цілісне відображення меж лабіринту та коректне поєднання сусідніх стінових елементів.

Блок 13. Розміщення фінішного актора у клітинці виходу. У визначеній області виходу розміщується фінішний об'єкт, який використовується системою для фіксації завершення проходження рівня.

Блок 14. Завершення виконання алгоритму. Після побудови геометрії та розміщення фінішного об'єкта згенерований лабіринт вважається готовим до використання в ігровому процесі.

Окремим важливим етапом побудови геометрії є додавання проміжних стінок між сусідніми стіновими клітинами. Необхідність цього пов'язана з тим, що розмір клітини в логічній моделі становить 100 одиниць, тоді як сама модель стіни є тоншою. Якщо розміщувати стіни лише в центрах клітин зі станом стіни, вони виглядали б як окремі ізольовані стовпчики, а не як суцільні лабіринтні перегородки. Тому після основного розміщення стін додатково виконується побудова проміжних стінок між сусідніми клітинами-стінами. Це рішення не є єдино можливим, однак у межах цієї реалізації воно дозволяє отримати візуально цілісну структуру лабіринту без ускладнення базової логіки генерації.

Приклад згенерованого лабіринту, побудованого за результатами роботи описаного алгоритму, наведено на рис. 3.10.

На рис. 3.10 показано підсумковий вигляд лабіринту після генерації проходів, формування стартової зони, додавання виходу, отворів і побудови геометрії сцени. Це ілюструє, що результатом роботи алгоритму є не лише внутрішня логічна структура, а готовий до використання ігровий рівень.



Рисунок 3.10 – Приклад згенерованого лабіринту в ігровій сцені

Уся описана логіка реалізована в C++-класі `MazeGeneratorActor`. Саме в ньому поєднано створення сіткової моделі лабіринту, DFS-генерацію проходів, очищення стартової зони, формування виходу, додавання отворів, побудову геометрії сцени та розміщення фінішного актора. Реалізацію C++-класу `MazeGeneratorActor` наведено в додатках А.6–А.7.

Отже, алгоритм процедурної генерації лабіринту в розроблюваній системі охоплює не лише побудову випадкової структури проходів методом обходу в глибину, а й повний цикл підготовки ігрового середовища до використання. Сіткова модель виступає проміжним логічним поданням, на основі якого формуються проходи, стартова зона, вихід, отвори та геометрія рівня. У результаті система отримує не просто абстрактний лабіринт, а готовий до проходження ігровий рівень, який надалі використовується під час роботи алгоритмів керування сценою та формування просторового тактильного зворотного зв'язку.

3.6 Розроблення алгоритму формування просторового тактильного зворотного зв'язку

Просторовий тактильний зворотний зв'язок у розроблюваній системі виконує не лише допоміжну або декоративну функцію. Його призначення полягає в тому, щоб передавати користувачу додаткову інформацію про події ігрового середовища через фізичний контролер. На відміну від простого підходу, за якого будь-яке зіткнення викликало б однаковий вібраційний сигнал, у цій реалізації тактильний відгук залежить і від сили зіткнення, і від його напрямку. Такий підхід дозволяє зробити матеріальний інтерфейс інформативнішим, оскільки вібраційний зворотний зв'язок може використовуватися не лише для повідомлення про сам факт події, а й для передавання напрямкових або просторових ознак взаємодії [18].

Основна логіка цього алгоритму реалізована у блюпринті Ball і запускається під час події On Component Hit, тобто в момент зіткнення кульки з елементами сцени. На першому етапі визначається інтенсивність зіткнення. Для цього використовується вектор Normal Impulse, а числова оцінка удару обчислюється за його довжиною. Після цього виконується порогова перевірка: якщо сила зіткнення є надто малою, тактильний відгук не формується. Такий поріг потрібний для відсікання дрібних і малоінформативних контактів, які в іншому випадку створювали б зайвий шум у вібраційних модулях і погіршували б сприйняття справді важливих подій.

Після проходження порогової перевірки формується базова інтенсивність тактильного сигналу. У реалізації величина імпульсу зіткнення спочатку нормалізується відносно заданого масштабного значення, далі обмежується в діапазоні від 0 до 1, після чого до неї застосовується нелінійне степеневе перетворення з показником 0,35. Таке масштабування підібрано експериментально. Воно дозволяє підсилити відчутність слабких і середніх

зіткнень, але не робить сильні удари непропорційно різкими. У загальному вигляді базову інтенсивність сигналу можна подати формулою:

$$B = (\text{clamp}(\frac{\|I\|}{I_{max}}, 0, 1))^{0,35},$$

де I – вектор імпульсу зіткнення;

I_{max} – масштабний коефіцієнт для нормалізації сили зіткнення;

B – базова інтенсивність тактильного сигналу.

Проте для просторового тактильного зворотного зв'язку важливо враховувати не лише силу, а й напрямок зіткнення. У реалізації для цього використовується вектор Impact Normal зі структури Hit Result, який задає нормаль контакту в точці зіткнення та використовується як напрямкова ознака події. Після нормалізації він переводиться у двовимірне подання для подальшого порівняння з напрямковими векторами вібраційних модулів. Далі цей напрямок зіткнення порівнюється з трьома напрямковими векторами, що відповідають просторовому розташуванню трьох вібраційних модулів у фізичному контролері. Оскільки модулі розміщені навколо руків'я у трьох основних напрямках, для їх подання використано одиничні вектори у двовимірній площині, рознесені між собою на 120° . Верхньому модулю відповідає напрямок $V_{M3} = (0; 1)$, а двом нижнім боковим модулям – напрямки $V_{M1} = (0,866; -0,5)$ та $V_{M2} = (-0,866; -0,5)$.

Схему цих напрямкових векторів наведено на рис. 3.11. Червоний вектор d на цьому рисунку показує приклад поточного напрямку зіткнення; під час роботи системи він визначається окремо для кожної події контакту. Розподіл інтенсивності між модулями залежить від близькості цього вектора до напрямкових векторів відповідних вібраційних модулів.

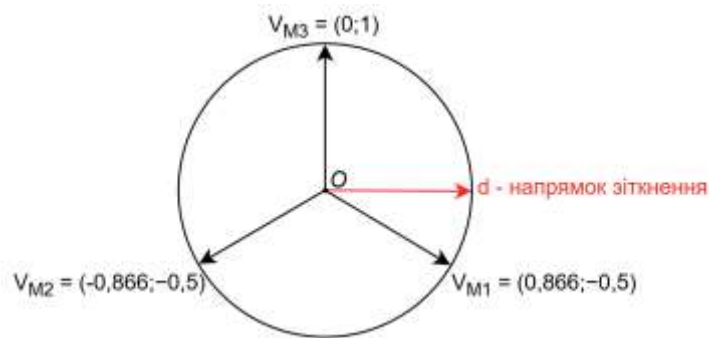


Рисунок 3.11 – Схема напрямкових векторів вібраційних модулів для просторового розподілу сигналу

Близькість напрямку зіткнення до напрямку кожного модуля в реалізації визначається через скалярний добуток. Отримане значення обмежується в діапазоні $[0;1]$, після чого до нього застосовується квадратний корінь. Це нелінійне підсилення потрібне для того, щоб слабші напрямкові складові не зникали повністю, а переходи між модулями були м'якшими й краще відчувалися користувачем. Вибір такого перетворення та його параметрів виконано експериментально під час налагодження тактильного відгуку. Напрямову вагу для i -го модуля можна подати формулою:

$$w_i = \sqrt{\text{clamp}(d \cdot V_{Mi}, 0, 1)},$$

де d – нормалізований напрямок зіткнення;

V_{Mi} – напрямковий вектор i -го вібраційного модуля;

w_i – напрямкова вага відповідного модуля.

Важливо, що інтенсивність розподіляється між модулями не шляхом жорсткого вибору одного активного модуля для кожного напрямку, а плавно, відповідно до близькості напрямку зіткнення до напрямкових векторів модулів. Чим ближчий напрямок зіткнення до напрямкового вектора певного модуля, тим більшою стає інтенсивність цього модуля. Завдяки цьому тактильний сигнал сприймається не як грубе перемикання між трьома окремими точками, а як більш безперервний просторовий відгук.

Після визначення базової інтенсивності та напрямкових ваг для кожного модуля формуються PWM-рівні. Для цього базова інтенсивність множиться на відповідну напрямкову вагу, після чого результат масштабується до діапазону PWM-сигналу і переводиться в ціле число. У загальному вигляді це можна подати формулою:

$$P_i = \text{trunc}(255 \cdot B \cdot w_i),$$

де P_i – PWM-рівень для i -го вібраційного модуля;

B – базова інтенсивність сигналу;

w_i – напрямкова вага відповідного модуля.

У реалізації використано такі основні експериментально підібрані параметри: поріг спрацювання за величиною імпульсу становить 300, масштабне значення для нормалізації інтенсивності зіткнення – 1500, а показник степеневого перетворення базової інтенсивності – 0,35. Після обчислення базової інтенсивності та напрямкової ваги результат масштабується до діапазону PWM-рівня від 0 до 255, який використовується для керування вібраційними модулями.

Таким чином, підсумковий сигнал для кожного модуля формується з урахуванням двох складових: сили зіткнення та просторової близькості напрямку зіткнення до напрямкового вектора цього модуля. Узагальнений алгоритм формування просторового тактильного зворотного зв'язку наведено на рис. 3.12.

Блок 1. Початок виконання алгоритму. На цьому етапі запускається логіка формування тактильного зворотного зв'язку на основі подій ігрового середовища.

Блок 2. Отримання події зіткнення кульки. Алгоритм приймає інформацію про зіткнення кульки з елементом сцени, яка використовується для подальшого визначення параметрів вібраційного відгуку.

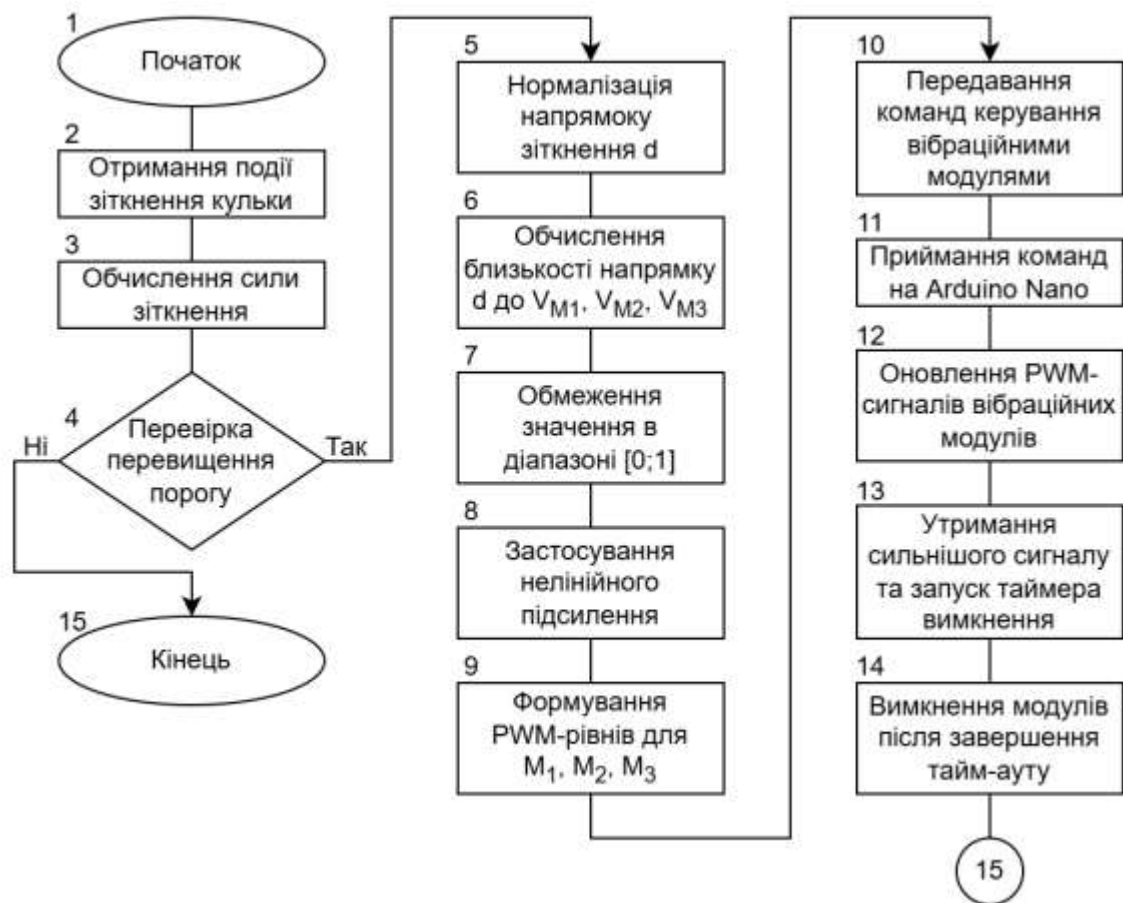


Рисунок 3.12 – Схема алгоритму формування просторового тактильного зворотного зв'язку

Блок 3. Обчислення сили зіткнення. На основі параметрів події визначається інтенсивність взаємодії кульки з перешкодою, що надалі впливає на силу тактильного сигналу.

Блок 4. Перевірка перевищення порогу. Система перевіряє, чи є сила зіткнення достатньою для формування вібраційного відгуку. Якщо значення не перевищує встановлений поріг, алгоритм переходить до завершення без подавання сигналу.

Блок 5. Нормалізація напрямку зіткнення d . Якщо сила зіткнення є достатньою, визначається та нормалізується напрямок взаємодії, який використовується для просторового розподілу сигналу між вібраційними модулями.

Блок 6. Обчислення близькості напрямку d до VM1, VM2, VM3. Для кожного вібраційного модуля визначається, наскільки напрямок зіткнення відповідає його просторовому розташуванню.

Блок 7. Обмеження значення в діапазоні $[0;1]$. Обчислені значення приводяться до допустимого діапазону, щоб уникнути некоректних рівнів інтенсивності сигналу.

Блок 8. Застосування нелінійного підсилення. До отриманих значень застосовується нелінійне масштабування, що дає змогу зробити проміжні напрямки взаємодії більш відчутними для користувача.

Блок 9. Формування PWM-рівнів для M1, M2, M3. На основі сили та напрямку зіткнення обчислюються рівні PWM-сигналів для трьох вібраційних модулів.

Блок 10. Передавання команд керування вібраційними модулями. Сформовані команди надсилаються з програмної частини гри до фізичного контролера через канал серійного обміну.

Блок 11. Приймання команд на Arduino Nano. МК отримує команди керування вібраційними модулями та виконує їх подальше опрацювання на стороні фізичного пристрою.

Блок 12. Оновлення PWM-сигналів вібраційних модулів. Для відповідних вібромоторів оновлюються рівні керувальних PWM-сигналів, що визначають інтенсивність вібрації.

Блок 13. Утримання сильнішого сигналу та запуск таймера. Якщо для модуля вже діє сильніший сигнал, слабший не перезаписує його одразу. Одночасно запускається або оновлюється таймер, який визначає тривалість відтворення відгуку.

Блок 14. Вимкнення модулів після завершення тайм-ауту. Після завершення встановленого часу вібраційні модулі вимикаються, що запобігає неконтрольованому продовженню сигналу.

Блок 15. Завершення виконання алгоритму. Після формування, передавання та завершення відтворення тактильного відгуку поточний цикл алгоритму завершується.

Фрагмент реалізації цього алгоритму у блюпринті Ball наведено на рис. 3.13. На ньому показано подію On Component Hit, визначення сили зіткнення через Vector Length, порогову перевірку, нормалізацію напрямку, обчислення близькості до напрямкових векторів через скалярний добуток, який у Blueprint реалізовано вузлом dot, обмеження значень, застосування квадратного кореня, формування PWM-рівнів та виклики функції Call Vibrate Motor для трьох модулів.

Після формування PWM-рівнів відповідні значення передаються на фізичний контролер у вигляді службових команд M:<motor>:<value>. Загальний механізм серійного обміну між грою і контролером уже було розглянуто в підрозділі 3.2, тому в межах цього підрозділу важливим є не сам канал обміну, а зміст команд, які надсилаються для відтворення просторового тактильного сигналу. Для кожного з трьох модулів формується окрема команда керування, після чого вона надходить на Arduino Nano.

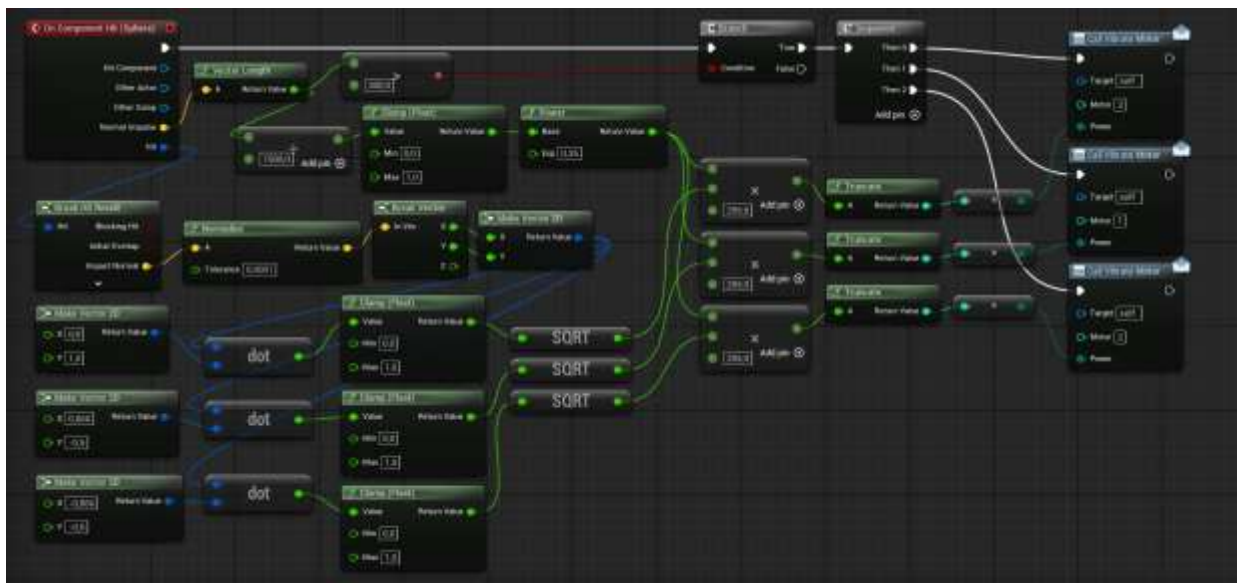


Рисунок 3.13 – Фрагмент реалізації формування тактильного відгуку у блюпринті Ball

На стороні Arduino Nano реалізовано допоміжну логіку відтворення сигналу. Після приймання команди контролер визначає номер модуля та PWM-рівень, який потрібно встановити. Далі значення перетворюється на PWM-сигнал для відповідного вібраційного модуля. У прошивці стандартна тривалість відгуку для команди керування становить 10 циклів роботи контролера, що за періоду циклу близько 10 мс відповідає приблизно 100 мс. Важливим удосконаленням фінальної реалізації є логіка пріоритету сильнішого сигналу. Якщо новий PWM-рівень є меншим за вже активне значення для відповідного модуля, він не замінює поточний сильніший сигнал. Якщо ж новий рівень не менший, PWM-сигнал оновлюється, а таймер відгуку для цього модуля запускається заново. Це підвищує виразність тактильного відгуку й не дозволяє слабким короточасним зіткненням миттєво перекривати сильніші сигнали.

Крім того, у прошивці реалізовано тайм-аут автоматичного вимкнення. Для кожного модуля використовується окремий лічильник часу, який зменшується в циклі роботи контролера. Якщо нових команд для модуля більше не надходить, після завершення тайм-ауту він вимикається автоматично. Така логіка запобігає «залипанню» вібраційного сигналу й забезпечує коректне завершення відгуку навіть тоді, коли подія в грі вже закінчилася. Реалізацію цієї частини алгоритму наведено в додатку А.1.

Отже, алгоритм формування просторового тактильного зворотного зв'язку перетворює подію зіткнення в грі на фізично відчутний сигнал, який залежить як від сили, так і від напрямку взаємодії. У блюпринті Ball виконується оцінювання сили зіткнення, визначення напрямку, просторовий розподіл сигналу між трьома вібраційними модулями та формування PWM-рівнів, а на стороні Arduino Nano – приймання команд, відтворення сигналу, реалізація пріоритету сильнішого сигналу та автоматичне вимкнення після завершення тайм-ауту. Завдяки цьому тактильний канал у розроблюваній системі виступає не декоративним доповненням, а повноцінним інформаційним компонентом взаємодії користувача з ігровим середовищем.

3.7 Програмна реалізація компонентів системи та користувацької взаємодії

3.7.1 Реалізація основних програмних компонентів системи

Програмна частина розроблюваної системи реалізована як сукупність взаємопов'язаних програмних компонентів, кожен із яких виконує окрему роль у загальному циклі роботи гри. До її складу входять як C++-класи, що забезпечують службові та обчислювальні частини реалізації, так і блюпринти, у яких зосереджено значну частину логіки ігрової взаємодії. Такий поділ дає змогу розмежувати приймання й інтерпретацію даних, генерацію ігрового простору, фізичну поведінку об'єктів і координацію загального стану системи. Основні програмні компоненти системи та їх призначення наведено в табл. 3.4.

Таблиця 3.4 – Основні програмні компоненти системи та їх призначення

Програмний компонент	Тип	Призначення
SerialReader	блюпринт	Зчитування доступних рядків із COM-порту в програмній частині
SerialCommandProcessor	C++-клас	Розбір повідомлень, формування службових команд, отримання списку COM-портів
SensorDataInterpreter	C++-клас	Оброблення та інтерпретація IMU-даних
MazeGeneratorActor	C++-клас	Процедурна генерація лабіринту та побудова його геометрії
MazeGenerator	блюпринт / актор сцени	Сценове представлення генератора лабіринту, прив'язка параметрів і базових елементів лабіринту
Ball	блюпринт	Фізична поведінка кульки, оброблення зіткнень, формування подій тактильного відгуку
GameManager	блюпринт	Координація станів гри, керування камерою, калібруванням, інтерфейсом користувача та взаємодією компонентів

Компоненти SerialReader, SerialCommandProcessor і SensorDataInterpreter утворюють послідовний ланцюг приймання, розбору та інтерпретації даних, що надходять від фізичного контролера. SerialReader забезпечує роботу з вхідними рядками у програмній частині, SerialCommandProcessor виконує первинний розбір повідомлень і формує

службові команди, а `SensorDataInterpreter` перетворює IMU-вимірювання на стабільні параметри нахилу, які надалі використовуються в ігровій логіці. Завдяки такому поділу в системі відокремлено етапи приймання даних, їх первинного оброблення та подальшої змістовної інтерпретації.

Іншу групу становлять компоненти, що забезпечують безпосередню роботу ігрового середовища. C++-клас `MazeGeneratorActor` відповідає за алгоритмічну частину генерації лабіринту та побудову його геометрії, а блюпринт `MazeGenerator` використовується як сценове представлення цього генератора в ігровому рівні. Блюпринт `Ball` реалізує фізичну поведінку кульки та її реакцію на події середовища, а блюпринт `GameManager` координує керування сценою, калібрування, вибір режимів камери, запуск і завершення ігрового процесу. У сукупності ці програмні компоненти формують основу реалізації системи на рівні ігрової логіки.

Окрім програмних компонентів, реалізація гри спирається на набір базових ігрових елементів, які використовуються для побудови сцени та організації взаємодії кульки з лабіринтом. Основні такі елементи наведено на рис. 3.14.

На рис. 3.14 показано базові елементи, з яких формується ігровий простір лабіринту. Кулька є основним рухомим об'єктом сцени, підлога та стіни формують простір лабіринту, а отвори виступають небезпечними зонами, взаємодія з якими призводить до падіння. Саме поєднання цих елементів забезпечує реалізацію проходження лабіринту, зіткнень і подій, що далі використовуються в інших алгоритмічних блоках системи.

Таким чином, програмна частина системи складається з набору спеціалізованих компонентів, які відповідають за приймання даних, їх інтерпретацію, генерацію ігрового простору, фізичну поведінку кульки та координацію загального циклу гри. У поєднанні з базовими ігровими елементами ці компоненти формують програмну основу реалізованої системи.

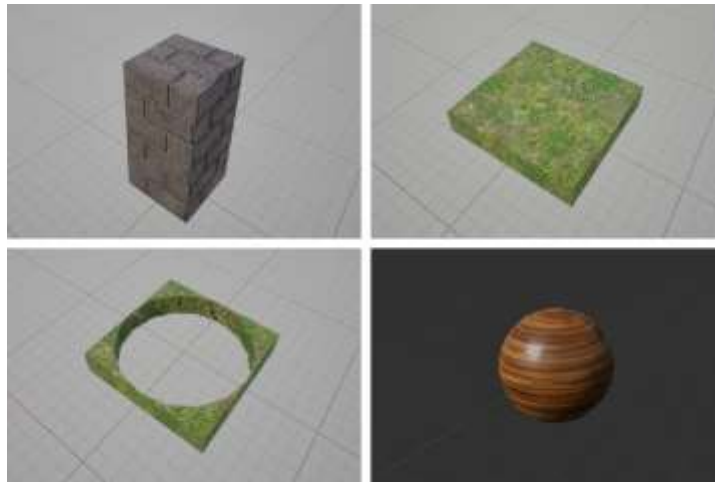


Рисунок 3.14 – Основні ігрові елементи, використані в реалізації системи

3.7.2 Реалізація взаємодії компонентів та керування станами системи

Після опису основних програмних компонентів важливо показати, як вони взаємодіють між собою в межах реалізованої системи. Якщо в попередніх підрозділах було окремо описано алгоритми передавання й інтерпретації даних, керування сценою, генерації лабіринту та формування просторового тактильного зворотного зв'язку, то в цьому пункті увагу зосереджено саме на їх практичному поєднанні. Взаємодія компонентів у системі побудована так, щоб окремі спеціалізовані частини реалізації виконували власні функції, а їх узгоджену роботу забезпечував центральний координувальний компонент.

Після ініціалізації рівня система отримує доступ до основних програмних компонентів, які надалі беруть участь у роботі гри. На цьому етапі готуються початкові параметри, що визначають подальший перебіг ігрового процесу: розмір лабіринту, вибраний СОМ-порт, режим камери та службові прапорці стану. Активна ігрова взаємодія не починається відразу після запуску, оскільки перед цим користувач повинен виконати підготовчі дії, насамперед обрати СОМ-порт і завершити калібрування фізичного контролера. Таким чином, початковий етап роботи системи є не просто

запуском сцени, а підготовкою компонентів до узгодженого функціонування в межах єдиного ігрового циклу.

Під час активного сенсорного керування взаємодія компонентів реалізується як послідовний ланцюг перетворення вхідних даних у керувальний вплив на ігрове середовище. Блюпринт `SerialReader` зчитує доступні рядки з COM-порту, C++-клас `SerialCommandProcessor` виконує їх первинний розбір, а C++-клас `SensorDataInterpreter` формує стабільні параметри нахилу. Далі ці параметри використовуються блюпринтом `GameManager` для координації роботи сцени. На їх основі оновлюється параметр імітованого нахилу, формується керувальний вплив для кульки, а також узгоджуються камера й освітлення. Отже, сенсорні дані не надходять безпосередньо в окремий ігровий об'єкт, а проходять послідовний шлях через кілька спеціалізованих компонентів, кожен з яких виконує власну частину загальної логіки.

Окремий цикл взаємодії компонентів пов'язаний зі створенням і оновленням лабіринту. Блюпринт `GameManager` ініціює створення або повторне формування рівня відповідно до вибраних параметрів, насамперед до розміру лабіринту. Алгоритмічна частина генерації та побудови геометрії реалізована в C++-класі `MazeGeneratorActor`, тоді як блюпринт `MazeGenerator` виступає сценовим представленням генератора в ігровому рівні та пов'язує його з іншими елементами сцени. Після завершення генерації оновлюється сам ігровий простір, у межах якого надалі працюють кулька, камера, події зіткнення та фінішна область. Завдяки цьому генерація лабіринту в системі є не ізольованою процедурою, а частиною загального циклу керування рівнем.

Важливу роль у взаємодії компонентів відіграють події ігрового процесу. Блюпринт `Ball` виступає не лише фізичним об'єктом сцени, а й джерелом подій, що мають значення для всієї системи. До таких подій належать зіткнення кульки, падіння в отвори та досягнення фінішної області. На основі цих подій система або формує просторовий тактильний зворотний зв'язок, або змінює стан проходження рівня, або ініціює реакцію, пов'язану з

перезапуском чи завершенням гри. При цьому блюпринт GameManager не підміняє собою спеціалізовані компоненти, а виконує координувальну роль: він узгоджує реакцію системи на події, оновлює відповідні стани та забезпечує правильний перехід між етапами ігрового процесу.

Реалізація керування станами системи побудована так, щоб у різних режимах роботи були активними різні компоненти й різні сценарії взаємодії. У стані головного меню система очікує налаштувань з боку користувача. На етапі підключення контролера та вибору COM-порту основна увага зосереджується на доступності фізичного пристрою та підготовці серійного обміну. Під час калібрування пріоритетною стає взаємодія з контролером і службовими командами, тоді як основна ігрова логіка ще не використовується в повному обсязі. В активному ігровому стані одночасно працюють сенсорне керування, фізика кульки, керування сценою, генерація подій і тактильний зворотний зв'язок. У стані паузи сцена не реагує на IMU-дані так само, як під час активної гри, а після завершення проходження система переходить до відповідного екрану результату і змінює подальший сценарій взаємодії з користувачем. Отже, керування станами в системі реалізоване не як набір окремих несумісних режимів, а як узгоджена логіка переходів між етапами роботи гри.

У такій організації блюпринт GameManager виконує роль центрального координувального компонента. Він не реалізує самостійно всі спеціалізовані частини системи, але саме через нього узгоджується робота сенсорного керування, генерації рівня, камери, освітлення, подій гри та переходів між інтерфейсними станами. Водночас окремі компоненти зберігають власну спеціалізацію: SerialReader, SerialCommandProcessor і SensorDataInterpreter відповідають за вхідні дані, MazeGeneratorActor – за побудову рівня, Ball – за фізичну поведінку кульки та події середовища. Такий розподіл дозволяє відокремити спеціалізовані частини реалізації від загальної координації та робить структуру системи зрозумілішою.

Таким чином, взаємодія компонентів у системі побудована як узгоджений цикл приймання даних, оновлення стану гри, керування сценою, оброблення подій і взаємодії з користувачем. Окрему роль у цьому циклі відіграє інтерфейс користувача, реалізацію якого розглянуто далі.

3.7.3 Реалізація інтерфейсу користувача, локалізації та допоміжних засобів взаємодії

Інтерфейс користувача в розроблюваній системі є складовою програмної реалізації, яка забезпечує зв'язок між користувачем, грою та фізичним контролером. Саме через інтерфейс виконуються підготовчі дії перед початком гри, вибір параметрів роботи системи, калібрування контролера, супровід ігрового процесу та завершення сеансу взаємодії. Отже, інтерфейс користувача в цій системі виконує не лише роль засобу відображення інформації, а й роль організатора повного циклу взаємодії користувача з програмно-апаратною грою.

Реалізація інтерфейсу побудована на наборі окремих інтерфейсних віджетів, кожен із яких відповідає певному етапу користувацького сценарію. Початковою точкою взаємодії є головне меню, через яке користувач обирає доступний СОМ-порт, задає розмір лабіринту, переходить до калібрування та запускає гру. Таким чином, саме головне меню поєднує в собі функції стартового екрана, засобу початкового налаштування системи та точки входу в подальший ігровий процес.

Окремим етапом користувацької взаємодії є екран калібрування. Його призначення полягає в тому, щоб повідомити користувача про необхідність підготовки фізичного контролера, відобразити перебіг калібрування та повідомити про його завершення. У межах цього екрана користувач отримує вказівки щодо правильного положення пристрою, а сама система переходить від стану підготовки до режиму, у якому дані фізичного контролера можуть використовуватися для сенсорного керування. Завдяки цьому калібрування

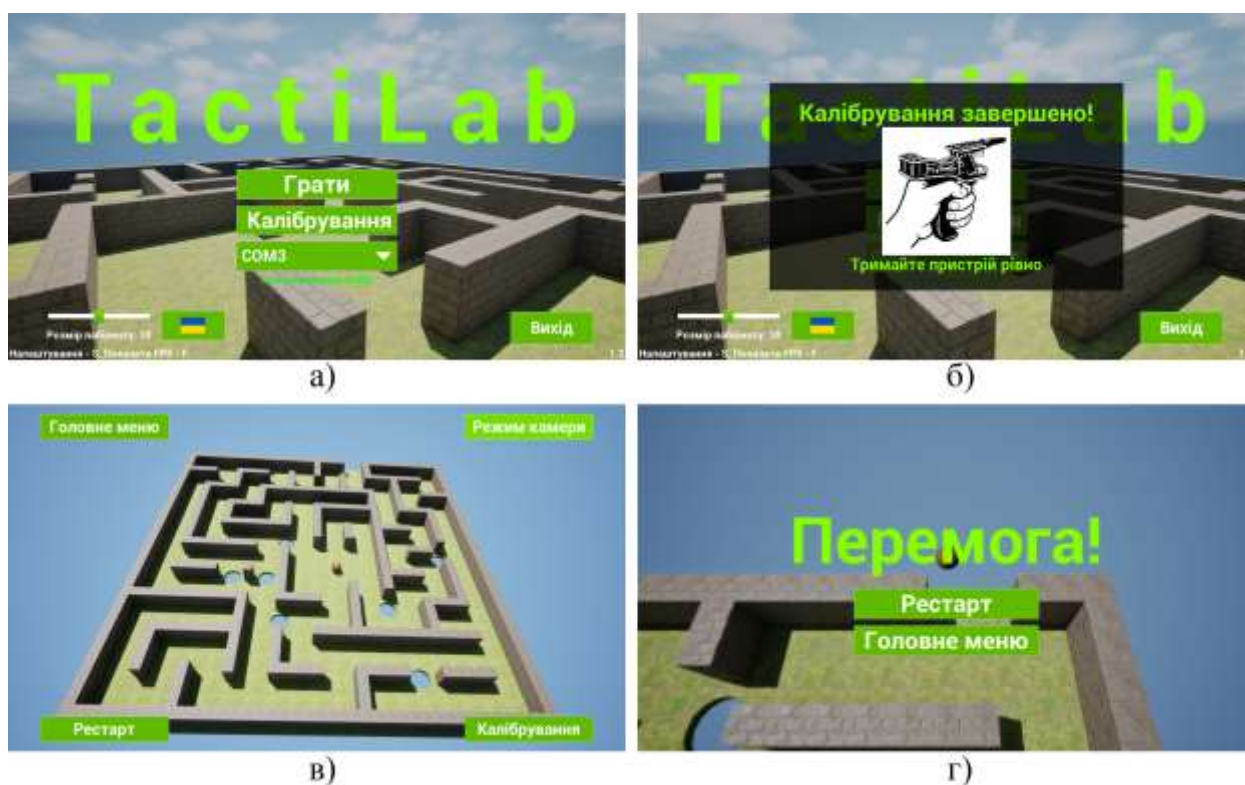
подається не як прихована службова операція, а як зрозумілий етап взаємодії із системою.

Під час активного проходження лабіринту використовується ігровий інтерфейс, який супроводжує основний ігровий процес. Через нього відображаються допоміжні елементи взаємодії, зокрема керування режимом камери, перезапуск рівня, повернення до головного меню та службова інформація, потрібна користувачеві під час гри. Після завершення проходження система переходить до окремого екрана завершення гри, який повідомляє користувача про успішне досягнення фінішу та надає можливість або повторно запустити рівень, або повернутися до головного меню. Таким чином, основні екрани інтерфейсу відповідають природній послідовності роботи користувача із системою: підготовка, калібрування, гра та завершення сеансу.

Окрім основних екранів, через інтерфейс реалізовано низку допоміжних дій, які безпосередньо впливають на зручність користування системою. До них належать вибір СОМ-порту для підключення фізичного контролера, вибір розміру лабіринту, запуск калібрування, перемикання режимів камери, а також дії, пов'язані з перезапуском або поверненням до початкового стану взаємодії. Усі ці можливості інтегровані в інтерфейс так, щоб користувач міг керувати системою без звернення до зовнішніх засобів або службових налаштувань поза межами гри.

Основні екрани інтерфейсу користувача, що відповідають ключовим етапам взаємодії із системою, наведено на рис. 3.15.

Наведені екрани охоплюють повний сценарій роботи користувача із системою: підготовку до запуску гри, калібрування фізичного контролера, активне проходження рівня та завершення ігрового сеансу. Завдяки такому поділу кожен етап взаємодії має окреме інтерфейсне подання, а користувач може послідовно виконати всі необхідні дії без звернення до зовнішніх службових засобів.



а – головне меню; б – екран калібрування; в – ігровий інтерфейс; г – екран завершення гри

Рисунок 3.15 – Основні екрани інтерфейсу користувача системи

Окремо в системі реалізовано локалізацію текстових елементів інтерфейсу. Вона охоплює головне меню, службові повідомлення, підказки та основні текстові написи, що супроводжують користувача під час роботи із системою. Локалізація не змінює алгоритмічну або ігрову логіку, проте підвищує зрозумілість взаємодії, оскільки дозволяє відображати інтерфейсні елементи відповідно до вибраної мови.

Крім того, у системі передбачено графічні налаштування, які дозволяють адаптувати візуальне відображення гри до потреб користувача. Через відповідний екран можна змінювати параметри якості зображення та окремі додаткові параметри відображення. Такі налаштування не належать до основної ігрової механіки, однак відіграють важливу роль як допоміжний

засіб взаємодії, оскільки дозволяють узгодити вигляд гри з уподобаннями користувача та можливостями комп'ютера.

Приклад реалізації локалізації та графічних налаштувань у системі наведено на рис. 3.16.



а – англomовний варіант головного меню; б – екран графічних налаштувань

Рисунок 3.16 – Приклад локалізації та графічних налаштувань системи

Англomовний варіант головного меню демонструє роботу локалізації текстових елементів інтерфейсу, а екран графічних налаштувань показує можливість адаптації візуального відображення гри до потреб користувача та можливостей комп'ютера. Ці засоби не змінюють основну ігрову механіку, однак підвищують зручність використання системи.

Таким чином, реалізація інтерфейсу користувача, локалізації та допоміжних засобів взаємодії доповнює алгоритмічну й програмну реалізацію системи та забезпечує завершений цикл роботи користувача з інтерактивною комп'ютерною грою. У сукупності з описаними раніше алгоритмічними блоками та програмними компонентами це формує цілісну реалізацію розроблюваної програмно-апаратної системи.

3.8 Висновки до розділу 3

У третьому розділі було розглянуто практичну реалізацію розроблюваної програмно-апаратної системи інтерактивної комп'ютерної гри. У межах цього розділу показано, яким чином поєднано фізичний контролер на базі Arduino Nano, інерціальний модуль MPU-6050, вібраційні модулі, серійний обмін через COM-порт і програмну частину, реалізовану в Unreal Engine 5. У результаті сформовано цілісну структуру реалізації, у якій фізична та програмна частини працюють узгоджено.

У розділі було реалізовано п'ять основних алгоритмічних блоків системи. Алгоритм передавання, приймання та первинного оброблення сенсорних даних забезпечує обмін між фізичним контролером і грою та формує коректні, перевірені й структуровані IMU-вимірювання. Алгоритм оброблення та інтерпретації IMU-даних перетворює ці вимірювання на стабільні параметри нахилу. Алгоритми керування сценою для імітації нахилу лабіринту забезпечують формування керувального впливу без постійної зміни трансформацій усієї геометрії лабіринту. Алгоритм процедурної генерації лабіринту формує логічну структуру рівня та перетворює її на готовий ігровий рівень. Алгоритм формування просторового тактильного зворотного зв'язку забезпечує формування вібраційного сигналу, що залежить від сили та напрямку взаємодії.

Практичним результатом реалізації цих алгоритмів стало створення системи, у якій дані фізичного контролера приймаються, інтерпретуються та використовуються для сенсорного керування грою в реальному часі. Нахил лабіринту в системі імітується алгоритмічно, а не через постійну зміну трансформацій усієї його геометрії. Лабіринт генерується процедурно та перетворюється на готовий до використання ігровий рівень. Тактильний зворотний зв'язок передає користувачу не лише факт зіткнення, а й додаткові просторові ознаки взаємодії через розподіл сигналу між трьома вібраційними модулями.

У розділі також узагальнено програмну реалізацію основних компонентів системи. Компоненти `SerialReader`, `SerialCommandProcessor` і `SensorDataInterpreter` забезпечують приймання вхідних повідомлень, їх первинний розбір та інтерпретацію IMU-даних. `MazeGeneratorActor` і `MazeGenerator` відповідають за алгоритмічну генерацію лабіринту та його представлення в ігровій сцені. `Ball` реалізує фізичну поведінку кульки та події взаємодії із середовищем, а `GameManager` координує загальний цикл роботи гри, узгоджуючи роботу спеціалізованих компонентів і керування станами системи. Окремо було описано реалізацію інтерфейсу користувача, локалізації та допоміжних налаштувань, які забезпечують повний цикл взаємодії користувача із системою: від вибору параметрів і підключення контролера до калібрування, проходження рівня та завершення гри.

Таким чином, у третьому розділі було сформовано цілісну програмно-апаратну реалізацію інтерактивної комп'ютерної гри, у якій фізичний контролер, алгоритмічні блоки, програмні компоненти, ігрова сцена, просторовий тактильний зворотний зв'язок та інтерфейс користувача працюють узгоджено. Це створює основу для подальшого оцінювання роботи системи, аналізу результатів її функціонування та перевірки прийнятих рішень у наступному розділі.

4 ТЕСТУВАННЯ СИСТЕМИ ТА ДОСЛІДЖЕННЯ РОЗРОБЛЕНИХ АЛГОРИТМІЧНИХ РІШЕНЬ

4.1 Організація та методика дослідження

Оскільки дипломна робота спрямована не лише на розроблення, а й на дослідження алгоритмів обчислень в інтерактивній комп'ютерній грі, у цьому розділі виконано функціональне тестування системи та дослідження основних алгоритмічних рішень. Такий підхід дає змогу не лише перевірити працездатність фінальної версії розроблюваної програмно-апаратної системи, а й оцінити, яким чином запропоновані алгоритмічні рішення вплинули на її поведінку, стабільність роботи та якість взаємодії користувача з грою.

У межах цього розділу попередній варіант системи, проаналізований у першому розділі, використовується як базова версія для порівняння. Вдосконалена версія системи розглядається як результат реалізації алгоритмів і програмних компонентів, описаних у третьому розділі. Для тих алгоритмічних блоків, що мали попередній варіант реалізації, дослідження виконується шляхом зіставлення характеристик попередньої та вдосконаленої версій. Це дає змогу оцінити не лише сам факт наявності нового рішення, а й практичний результат його впровадження. Водночас для алгоритму процедурної генерації лабіринту, який не має прямого попереднього аналога для коректного порівняння, виконується перевірка коректності формування структури рівня та працездатності згенерованого ігрового рівня.

Дослідження в четвертому розділі охоплює всі основні алгоритмічні блоки, визначені та обґрунтовані в другому розділі й реалізовані в третьому. До них належать алгоритм передавання, приймання та первинного оброблення сенсорних даних, алгоритм оброблення та інтерпретації IMU-даних, алгоритми керування сценою для імітації нахилу лабіринту, алгоритм процедурної генерації лабіринту та алгоритм формування просторового тактильного зворотного зв'язку. При цьому перші два алгоритмічні блоки в дослідженні розглядаються разом як єдиний алгоритмічний ланцюг роботи з

IMU-даними, оскільки в практичній системі вони послідовно забезпечують приймання, підготовку та інтерпретацію сенсорних вимірювань.

Поряд із дослідженням алгоритмічних рішень у розділі виконується функціональне тестування фінальної версії системи. Його призначення полягає у перевірці того, що розроблена програмно-апаратна система працює як цілісний комплекс і коректно підтримує основні сценарії взаємодії користувача з грою та фізичним контролером. Таким чином, у розділі поєднуються два взаємодоповнювальні напрями оцінювання: перевірка завершеної працездатності системи та аналіз ефекту від упроваджених алгоритмічних рішень.

Умови та критерії проведення тестування й дослідження алгоритмічних рішень наведено в таблиці 4.1.

Таблиця 4.1 – Умови та критерії проведення тестування й дослідження алгоритмічних рішень

Параметр / умова	Значення / спосіб перевірки
Об'єкт тестування	Фінальна програмно-апаратна система інтерактивної комп'ютерної гри
Базова версія для порівняння	Попередній варіант системи
Вдосконалена версія	Система після реалізації алгоритмів і програмних компонентів, описаних у розділі 3
Апаратна частина	Arduino Nano, MPU-6050, вібраційні модулі
Канал обміну	USB-з'єднання з використанням COM-порту
Середовище виконання	ПК із програмною частиною, розробленою в Unreal Engine 5
Критерії функціонального тестування	Відповідність фактичної поведінки системи очікуваній у ключових сценаріях роботи
Критерії дослідження алгоритмів	Своєчасність реакції, стабільність керування, коректність генерації рівня, інформативність тактильного відгуку

Отже, у четвертому розділі дослідження побудовано так, щоб оцінити як працездатність завершеної системи, так і результати вдосконалення її основних алгоритмічних блоків. Така організація дає змогу послідовно перейти спочатку до функціонального тестування системи в цілому, а далі – до аналізу окремих алгоритмічних рішень, що визначають її поведінку та якість взаємодії з користувачем.

4.2 Функціональне тестування системи

Функціональне тестування виконано з метою перевірки того, що розроблена програмно-апаратна система коректно працює як цілісний комплекс і підтримує основні сценарії взаємодії користувача з грою та фізичним контролером. На відміну від подальших підрозділів, у яких досліджуються окремі алгоритмічні рішення, у цьому пункті увагу зосереджено на загальній працездатності фінальної версії системи, тобто на перевірці того, чи правильно відбуваються запуск, підключення, калібрування, проходження рівня, оброблення подій гри та завершення сеансу взаємодії.

Під час функціонального тестування перевірялися ключові сценарії, що охоплюють повний цикл роботи системи: від запуску гри й підключення фізичного контролера до генерації лабіринту, проходження рівня, тактильного відгуку, завершення гри та тривалої роботи без критичних збоїв. Результати функціонального тестування системи наведено в табл. 4.2.

Таблиця 4.2 – Результати функціонального тестування системи

№	Сценарій тестування	Очікуваний результат	Фактичний результат	Статус
1	2	3	4	5
1	Запуск гри та відображення головного меню	Після запуску коректно відображається головне меню з основними елементами керування	Головне меню відображається коректно, основні елементи інтерфейсу доступні	Успішно
2	Вибір СОМ-порту та підключення фізичного контролера	Користувач може обрати доступний СОМ-порт і підключити контролер до системи	Список доступних СОМ-портів відображається, підключення контролера виконується коректно	Успішно
3	Калібрування контролера	Після запуску калібрування команда передається на фізичний контролер, а після завершення процедура підтверджується в інтерфейсі	Калібрування виконується, користувач отримує підтвердження про його завершення	Успішно
4	Приймання IMU-даних і реакція системи на нахил	Система приймає IMU-дані та реагує на зміну нахилу контролера під час гри	Приймання IMU-даних відбувається стабільно, реакція на нахил відповідає зміні положення контролера	Успішно

Кінець таблиці 4.2

1	2	3	4	5
5	Генерація лабіринту та побудова його геометрії	Після запуску гри формується лабіринт обраного розміру з коректною геометрією сцени	Лабіринт генерується та коректно відображається в сцені	Успішно
6	Повторна генерація або перезапуск рівня	Система коректно виконує повторне формування рівня без порушення роботи гри	Перезапуск рівня та повторна генерація виконуються коректно	Успішно
7	Керування кулькою під час проходження лабіринту	Кулька реагує на сенсорне керування та рухається відповідно до стану імітованого нахилу	Керування кулькою працює коректно, рух відповідає нахилу контролера	Успішно
8	Зіткнення кульки зі стінами та формування тактильного відгуку	Під час зіткнення формується тактильний сигнал, що передається на фізичний контролер	Тактильний відгук формується та відтворюється під час зіткнень	Успішно
9	Падіння кульки в отвір	Під час потрапляння кульки в отвір система формує вібраційний відгук і повертає кульку в стартове положення без повторної генерації рівня	Подія падіння визначається коректно, вібраційний відгук формується, кулька повертається в стартове положення	Успішно
10	Досягнення фінішу та перехід до екрана завершення	Після досягнення фінішу система завершує рівень і відображає екран завершення	Фініш визначається коректно, екран завершення відображається	Успішно
11	Перемикання режимів камери, локалізація та графічні налаштування	Допоміжні параметри доступні через інтерфейс і змінюють відповідні елементи системи	Режими камери перемикаються, мова інтерфейсу змінюється, графічні налаштування доступні для користувача	Успішно
12	Робота системи протягом ігрового сеансу без критичних збоїв	Під час послідовного виконання основних сценаріїв система зберігає працездатність без критичних помилок	Критичних збоїв під час тестового ігрового сеансу не виявлено	Успішно

Наведені результати показують, що фінальна версія системи коректно підтримує основні сценарії роботи користувача з інтерактивною комп'ютерною грою та фізичним контролером. Функціональне тестування підтвердило працездатність системи як цілісного програмно-апаратного комплексу. Далі доцільно перейти до дослідження окремих алгоритмічних рішень, покладених в основу її роботи.

4.3 Дослідження розроблених алгоритмічних рішень

У цьому підрозділі розглянуто результати дослідження основних алгоритмічних рішень, реалізованих у системі. На відміну від функціонального тестування, яке підтверджує працездатність системи загалом, подальший аналіз спрямований на оцінювання результату впровадження окремих алгоритмічних блоків. Для рішень, що мали попередній варіант реалізації, виконано порівняння попередньої та вдосконаленої версій системи. Для алгоритму процедурної генерації лабіринту, який не має прямого попереднього аналога, перевірено коректність формування рівня та придатність згенерованої геометрії до проходження.

4.3.1 Дослідження алгоритмічного ланцюга роботи з IMU-даними

Алгоритмічний ланцюг роботи з IMU-даними є одним із базових для всієї системи, оскільки саме через нього сенсорні вимірювання фізичного контролера перетворюються на параметри, придатні для подальшого керування ігровою сценою. До цього ланцюга належать алгоритм передавання, приймання та первинного оброблення сенсорних даних, а також алгоритм оброблення та інтерпретації IMU-даних. Саме якість їх спільної роботи визначає, наскільки своєчасно система реагує на зміну нахилу контролера, наскільки плавним є керування та чи не виникає відчуття запізнення під час взаємодії з грою.

Для оцінювання результату вдосконалення цього ланцюга виконано порівняння попередньої та вдосконаленої версій системи за критеріями, що безпосередньо впливають на керованість і стійкість роботи. Порівняння роботи з IMU-даними в попередній та вдосконаленій версіях системи наведено в табл. 4.3.

Таблиця 4.3 – Порівняння роботи з IMU-даними в попередній та вдосконаленій версіях системи

Критерій	Попередня версія	Вдосконалена версія	Результат удосконалення
Принцип зчитування з СОМ-порту	Оброблення вхідного потоку не передбачало послідовного зчитування всіх доступних рядків, що могло призводити до накопичення даних у буфері	Виконується послідовне зчитування доступних рядків із подальшим первинним розбором повідомлень	Зменшено ризик використання застарілих даних у керуванні
Ризик накопичення затримки	Підвищений під час тривалої роботи через менш ефективну організацію зчитування вхідного потоку	Знижений завдяки зчитуванню доступних рядків і послідовному розбору повідомлень	Реакція системи на зміну нахилу стала своєчаснішою
Ризик використання застарілих IMU-даних	У керування могли потрапляти значення, що вже не відповідали поточному стану контролера	У керування могли потрапляти значення, що вже не відповідали поточному стану контролера	У керування могли потрапляти значення, що вже не відповідали поточному стану контролера
Плавність реакції на зміну нахилу	Для зменшення нестабільності використовувалося сильніше згладжування, що послаблювало швидкість реакції	Реакція формується на основі даних, що краще відповідають поточному стану контролера, і помірного згладжування	Досягнуто кращого балансу між плавністю та швидкістю реакції
Стійкість керування під час ігрового сеансу	Стабільність могла знижуватися через накопичення затримки та надмірне згладжування	Керування зберігає передбачувану реакцію протягом тестового ігрового сеансу	Підвищено передбачуваність поведінки системи
Потреба в надмірному згладжуванні	Вища, оскільки згладжування частково компенсувало проблеми приймання даних	Зменшена завдяки кращій організації приймання та інтерпретації IMU-даних	Забезпечено кращий баланс між чутливістю та стабільністю керування

Наведене порівняння показує, що вдосконалення алгоритмічного ланцюга роботи з IMU-даними дало змогу зменшити ризик накопичення затримки та використання застарілих сенсорних вимірювань, а також зробити реакцію системи на нахил контролера більш узгодженою з поточним станом фізичного пристрою. Важливим результатом стало також зменшення потреби в надмірному згладжуванні, яке в попередній версії частково компенсувало недоліки приймання та опрацювання даних, але водночас послаблювало швидкість відгуку системи.

Отже, порівняння попередньої та вдосконаленої версій системи показало, що зміни в алгоритмах приймання, первинного оброблення та інтерпретації IMU-даних забезпечили кращий баланс між плавністю й

чутливістю керування. Це дає змогу розглядати вдосконалений ланцюг роботи з IMU-даними як один із чинників підвищення передбачуваності поведінки системи та якості взаємодії користувача з грою.

4.3.2 Дослідження алгоритмів керування сценою для імітації нахилу лабіринту

Алгоритми керування сценою для імітації нахилу лабіринту належать до ключових рішень, що визначають загальну поведінку системи під час гри. Саме вони забезпечують зв'язок між параметрами нахилу, отриманими після оброблення IMU-даних, та реакцією ігрової сцени. У попередній версії системи відтворення нахилу було пов'язане з постійною зміною трансформацій геометрії лабіринту. У вдосконаленій версії від цього підходу відмовлено: геометрія лабіринту залишається статичною, а нахил імітується алгоритмічно через параметр керування, штучний силовий вплив на кульку, а також узгоджену роботу камери й освітлення.

Для оцінювання результату такого вдосконалення виконано порівняння попередньої та вдосконаленої версій системи за критеріями, що характеризують не лише спосіб відтворення нахилу, а й його вплив на структуру сцени, рух кульки та узгодженість фізичної й візуальної складових. Порівняння підходів до керування сценою в попередній та вдосконаленій версіях системи наведено в табл. 4.4.

Наведене порівняння показує, що головним результатом удосконалення став перехід від підходу, заснованого на постійній зміні трансформацій геометрії лабіринту, до підходу зі статичною геометрією та алгоритмічною імітацією нахилу. Це дало змогу усунути потребу в постійних перетвореннях основної геометрії сцени під час гри та перенести керування в площину окремого силового впливу на кульку й візуальної імітації стану лабіринту.

Таблиця 4.4 – Порівняння підходів до керування сценою в попередній та вдосконаленій версіях системи

Критерій	Попередня версія	Вдосконалена версія	Результат удосконалення
Спосіб відтворення нахилу	Нахил відтворювався через постійну зміну трансформацій геометрії лабіринту	Нахил імітується алгоритмічно через параметр керування, штучний силовий вплив на кульку, камеру та освітлення	Змінено принцип керування сценою на більш передбачуваний і структурований
Оновлення трансформацій геометрії лабіринту	Для відтворення нахилу потрібно було постійно змінювати трансформації геометрії лабіринту	Геометрія лабіринту залишається статичною під час ігрового процесу	Усунено потребу в постійному оновленні трансформацій геометрії лабіринту
Кількість перетворень у сцені	У сцені виконувалися додаткові перетворення, пов'язані з відтворенням нахилу лабіринту	Трансформації основної геометрії рівня не змінюються, а керування переноситься на силовий вплив і візуальну імітацію	Зменшено кількість зайвих перетворень, необхідних для роботи сцени
Стабільність фізичної поведінки кульки	Поведінка кульки залежала від постійної зміни трансформацій елементів сцени	Рух кульки формується через окремий керувальний силовий вплив	Покращено передбачуваність руху кульки під час проходження лабіринту
Візуальна імітація нахилу	Візуальне сприйняття нахилу забезпечувалося переважно зміною геометрії сцени	Візуальне сприйняття нахилу підтримується камерою, освітленням і параметром імітованого нахилу	Збережено ефект нахилу без постійної зміни геометрії лабіринту
Узгодженість фізичної та візуальної складових	Фізична поведінка кульки й візуальне відтворення нахилу були тісно пов'язані зі зміною трансформацій сцени	Фізичний вплив на кульку, камера й освітлення узгоджуються через спільний параметр імітованого нахилу	Забезпечено цілісніше узгодження руху кульки та візуального сприйняття нахилу

Важливим наслідком такого переходу стало підвищення передбачуваності фізичної поведінки кульки. У вдосконаленій версії її рух не залежить від постійного оновлення трансформацій геометрії лабіринту, а формується через окремий керувальний вплив. Це робить реакцію системи більш упорядкованою та краще узгоджує сенсорне керування з фактичним рухом кульки під час проходження рівня.

Окремо слід відзначити, що відмова від постійної зміни трансформацій геометрії лабіринту не призвела до втрати візуального ефекту нахилу. У вдосконаленій версії це сприйняття підтримується через камеру, освітлення та спільний параметр імітованого нахилу. Завдяки цьому

досягнуто ціліснішого узгодження між тим, як система поводить себе фізично, і тим, як цей стан сприймається користувачем візуально.

Отже, порівняння попередньої та вдосконаленої версій системи показало доцільність переходу до алгоритмічної імітації нахилу лабіринту за статичної геометрії сцени. Такий підхід зменшив кількість зайвих перетворень у сцені, підвищив передбачуваність руху кульки та забезпечив узгодженість фізичної й візуальної складових керування. Це дає змогу розглядати вдосконалення алгоритмів керування сценою як один із ключових результатів оновлення всієї системи.

4.3.3 Перевірка алгоритму процедурної генерації лабіринту

Алгоритм процедурної генерації лабіринту є окремим алгоритмічним блоком системи, результат роботи якого безпосередньо визначає структуру ігрового рівня. На відміну від попередніх підпунктів, у цьому випадку не виконується порівняння попередньої та вдосконаленої версій системи, оскільки алгоритм процедурної генерації лабіринту не має прямого попереднього аналога для коректного зіставлення. Тому для нього виконано перевірку коректності формування структури рівня та придатності згенерованого рівня до використання в грі.

Перевірка охоплює роботу алгоритму для різних розмірів лабіринту, формування додаткових елементів рівня, побудову геометрії сцени, повторну генерацію та придатність отриманого результату до практичного використання в ігровій системі. У межах перевірки параметр n позначає заданий користувачем розмір лабіринту, на основі якого у внутрішньому поданні формується сітка рівня. Результати перевірки алгоритму процедурної генерації лабіринту наведено в табл. 4.5.

Таблиця 4.5 – Результати перевірки алгоритму процедурної генерації

лабіринту

Параметр / сценарій	Очікуваний результат	Фактичний результат	Висновок
Генерація лабіринту мінімального розміру $n=5$	Алгоритм формує лабіринт мінімального допустимого розміру без порушення структури	Лабіринт мінімального розміру створюється коректно, основні елементи рівня формуються	Алгоритм працездатний для мінімального розміру
Генерація лабіринту середнього розміру $n=10$	Алгоритм формує коректну структуру лабіринту середнього розміру	Генерація виконується без порушення структури рівня, елементи лабіринту розміщуються коректно	Алгоритм працездатний для типового розміру рівня
Генерація лабіринту максимального розміру $n=15$	Алгоритм формує лабіринт максимального допустимого розміру	Лабіринт максимального розміру створюється, геометрія сцени будується коректно, критичних збоїв не виявлено	Алгоритм зберігає працездатність у межах визначеного діапазону розмірів
Формування стартової зони, виходу та отворів	Після базової генерації до структури рівня додаються стартова зона, вихід і отвори	Стартова зона очищується, вихід формується на межі лабіринту, отвори додаються до структури рівня	Додаткові елементи рівня формуються коректно
Побудова підлоги, стін, отворів і фінішної області	Логічна структура лабіринту перетворюється на відповідну геометрію сцени	Підлога, стіни, отвори та фінішна область відображаються відповідно до згенерованої структури	Алгоритм забезпечує коректну побудову геометрії рівня
Повторна генерація рівня	Повторний запуск генерації формує нову конфігурацію лабіринту	У тестових запусках повторна генерація створює відмінні конфігурації проходів і розміщення елементів рівня	Забезпечено варіативність згенерованих рівнів
Придатність згенерованого рівня до проходження	Згенерований рівень може бути використаний для ігрового проходження	Згенеровані лабіринти мають цілісну структуру, коректну стартову область, вихід і геометрію для руху кульки	Результат генерації придатний до практичного використання в системі

Наведені результати показують, що алгоритм процедурної генерації лабіринту коректно працює в межах визначеного діапазону розмірів і забезпечує повний цикл формування рівня: від створення внутрішньої структури до побудови геометрії сцени. Перевірка також підтвердила, що після базової генерації правильно формуються додаткові елементи, зокрема стартова зона, вихід і отвори, а їх наявність узгоджується із загальною структурою рівня.

Окремо важливо, що повторний запуск алгоритму дає відмінні конфігурації лабіринту, тобто забезпечує варіативність ігрових рівнів. Це означає, що генерація не зводиться до однотипного відтворення однієї й тієї

самої структури, а дозволяє створювати різні конфігурації проходів і розташування елементів сцени в межах заданих параметрів.

Отже, перевірка показала, що алгоритм процедурної генерації лабіринту є працездатним і коректно формує варіативний і придатний до використання ігровий рівень. Це дає змогу розглядати його як повноцінний алгоритмічний блок системи, результат роботи якого може безпосередньо використовуватися в інтерактивній комп'ютерній грі.

4.3.4 Дослідження алгоритму формування просторового тактильного зворотного зв'язку

Алгоритм формування просторового тактильного зворотного зв'язку є важливою складовою розроблюваної системи, оскільки саме він забезпечує передавання користувачеві додаткової інформації про події ігрового середовища через фізичний контролер. У попередній версії системи вібраційний відгук уже використовувався як реакція на події зіткнення, однак його просторовий характер був недостатньо виразним. Це було пов'язано з менш коректним визначенням напряму взаємодії, слабким відчуттям проміжних напрямків між вібраційними модулями, недостатнім урахуванням нелінійної відчутності вібрації та відсутністю логіки пріоритету сильнішого сигналу. У вдосконаленій версії системи тактильний відгук формується з урахуванням сили та напряму взаємодії, просторово розподіляється між трьома вібраційними модулями та стабільніше відтворюється фізичним контролером.

Для оцінювання результату вдосконалення обрано критерії, що характеризують не лише сам принцип формування сигналу, а й його просторову виразність, стійкість відтворення та значення для користувацької взаємодії. Порівняння тактильного зворотного зв'язку в попередній та вдосконаленій версіях системи наведено в табл. 4.6.

Таблиця 4.6 – Порівняння тактильного зворотного зв'язку в попередній та вдосконаленій версіях системи

Критерій	Попередня версія	Вдосконалена версія	Результат удосконалення
Врахування сили зіткнення	Інтенсивність відгуку мала спрощений характер і не забезпечувала достатньо виразного розрізнення слабких та сильних зіткнень	Інтенсивність сигналу формується з урахуванням сили зіткнення та нелінійного масштабування	Тактильний відгук краще відповідає характеру взаємодії
Визначення напрямку зіткнення	Напрямок взаємодії визначався менш коректно, оскільки для розподілу сигналу використовувалися координатні значення події, а не нормалізований напрямок удару	Напрямок взаємодії визначається на основі напрямкової інформації зіткнення та використовується для просторового розподілу сигналу	Підвищено відповідність між напрямком події та відчуттям тактильного відгуку
Розподіл сигналу між вібраційними модулями	Просторовий розподіл між модулями був недостатньо виразним, особливо для напрямків між окремими вібраційними модулями	Інтенсивність сигналу розподіляється між трьома вібраційними модулями відповідно до напрямку взаємодії	Тактильний відгук став просторово диференційованим
Урахування нелінійної відчутності вібрації	Нелінійна відчутність вібраційних модулів не була достатньо врахована, через що слабкі або проміжні складові могли майже не відчуватися	Для базової інтенсивності та напрямкових складових використано нелінійне перетворення, підібране експериментально	Підвищено відчутність слабких і проміжних складових сигналу
Стабільність тактильного сигналу	Короткочасні зміни команд могли призводити до менш виразного або нестійкого відгуку	На стороні контролера реалізовано стабілізацію сигналу та пріоритет сильнішого сигналу	Підвищено виразність і стійкість тактильного відгуку
Перебивання сильнішого сигналу слабшим	Слабший сигнал міг одразу замінювати сильніший	Слабший сигнал не перезаписує поточний сильніший сигнал для відповідного модуля	Зменшено небажане ослаблення відгуку під час послідовних подій
Інформативність для користувача	Сигнал переважно повідомляв про факт події, але недостатньо чітко передавав її просторовий характер	Сигнал передає не лише факт події, а й силу та просторові ознаки взаємодії	Підвищено інформативність матеріального інтерфейсу

Наведене порівняння показує, що основним результатом удосконалення стало підвищення просторової виразності тактильного відгуку. Якщо в попередній версії сигнал уже використовувався як реакція на зіткнення, але недостатньо чітко передавав напрямковий характер події, то у вдосконаленій версії відгук формується з урахуванням сили зіткнення, напрямку взаємодії та розподіляється між трьома вібраційними модулями. Окреме нелінійне масштабування дало змогу зробити слабкі й проміжні

складові сигналу відчутнішими, що особливо важливо для напрямків між окремими модулями.

Важливим наслідком удосконалення стало також підвищення стійкості відтворення сигналу. Логіка пріоритету сильнішого сигналу на стороні контролера зменшує небажане ослаблення відгуку під час послідовних подій, коли слабша команда могла б перекрити сильнішу. Завдяки цьому тактильний відгук став не лише інформативнішим, а й виразнішим у практичному використанні.

Отже, порівняння попередньої та вдосконаленої версій системи показало, що алгоритм формування просторового тактильного зворотного зв'язку підвищив просторову виразність, стабільність та інформативність матеріального інтерфейсу. Удосконалений відгук передає користувачеві не лише факт події, а й додаткові ознаки сили та напрямку взаємодії, що робить фізичний контролер активнішою частиною взаємодії з ігровим середовищем.

4.4 Висновки до розділу 4

У четвертому розділі було проведено функціональне тестування фінальної версії розроблюваної програмно-апаратної системи та дослідження основних алгоритмічних рішень, покладених в основу її роботи. Такий підхід дав змогу оцінити систему з двох взаємопов'язаних позицій: як цілісний інтерактивний комплекс і як сукупність окремих алгоритмічних блоків, що визначають її поведінку, стійкість і якість взаємодії з користувачем.

Функціональне тестування підтвердило, що фінальна версія системи коректно працює як єдиний програмно-апаратний комплекс. Було перевірено основні сценарії взаємодії користувача з грою та фізичним контролером, зокрема запуск системи, підключення контролера, калібрування, приймання IMU-даних, генерацію лабіринту, проходження рівня, формування тактильного відгуку, оброблення падіння кульки в отвір, завершення гри та використання допоміжних налаштувань. Результати тестування показали, що

система підтримує повний цикл роботи без критичних збоїв у межах перевірених сценаріїв.

Дослідження алгоритмічного ланцюга роботи з ІМУ-даними та алгоритмів керування сценою показало, що впроваджені рішення забезпечили практично значущі покращення. Порівняння показало зменшення затримки в ланцюгу приймання та оброблення ІМУ-вимірювань, зниження ризику використання застарілих даних і досягнення кращого балансу між плавністю й чутливістю керування. Також підтверджено доцільність переходу від постійної зміни трансформацій геометрії лабіринту до статичної геометрії та алгоритмічної імітації нахилу. Це дозволило зменшити кількість зайвих перетворень у сцені, підвищити передбачуваність руху кульки та краще узгодити фізичну й візуальну складові взаємодії.

Перевірка алгоритму процедурної генерації лабіринту показала, що він коректно формує лабіринти різних розмірів, забезпечує варіативність конфігурацій і дає результат, придатний до використання в грі. Дослідження алгоритму формування просторового тактильного зворотного зв'язку, своєю чергою, засвідчило, що вдосконалений тактильний відгук став інформативнішим, оскільки враховує силу та напрям взаємодії, а також стабільніше відтворюється фізичним контролером. У сукупності це підтверджує, що вдосконалення алгоритмічних рішень вплинуло не лише на внутрішню логіку системи, а й на якість користувацької взаємодії.

Таким чином, результати четвертого розділу підтвердили працездатність фінальної версії системи та доцільність упроваджених алгоритмічних рішень у межах визначених критеріїв тестування й дослідження. Отримані результати створюють основу для формулювання загальних висновків дипломної роботи.

ВИСНОВКИ

У дипломній роботі вирішено задачу покращення продуктивності, стабільності та користувацького досвіду інтерактивної комп'ютерної гри з матеріальним інтерфейсом на основі розроблення та дослідження алгоритмів функціонування системи. Розглянуто програмно-апаратну інтерактивну гру, у якій фізичний контролер на основі інерціального сенсора забезпечує сенсорне керування, а вібраційний зворотний зв'язок доповнює взаємодію з цифровим середовищем.

У процесі виконання роботи проаналізовано предметну область інтерактивних ігор з матеріальним інтерфейсом, розглянуто близькі аналоги та попередній варіант розробленої системи. На основі проведеного аналізу сформульовано функціональні та нефункціональні вимоги до вдосконаленої системи, визначено її загальну структуру та функціональні підсистеми, а також обґрунтовано основні алгоритмічні блоки, що забезпечують приймання та інтерпретацію сенсорних даних, керування сценою, генерацію ігрового простору і формування тактильного відгуку.

У роботі розроблено алгоритми передавання, приймання та первинного оброблення сенсорних даних, алгоритм оброблення та інтерпретації IMU-даних, алгоритми керування сценою для імітації нахилу лабіринту, алгоритм процедурної генерації лабіринту та алгоритм формування просторового тактильного зворотного зв'язку. Сукупність цих алгоритмічних рішень дала змогу побудувати узгоджену систему, у якій сенсорні вимірювання перетворюються на стабільний керувальний вплив, сцена реагує без надлишкових перетворень геометрії, лабіринт формується процедурно, а події доповнюються змістовним тактильним відгуком.

Реалізовано програмно-апаратну систему на базі Unreal Engine 5, Arduino Nano, MPU-6050 та вібраційних модулів. У програмній частині створено компоненти для приймання й первинного розбору даних, інтерпретації IMU-вимірювань, процедурної генерації лабіринту, керування

сценою, оброблення подій гри та забезпечення інтерфейсу користувача. У результаті сформовано завершену інтерактивну систему, що поєднує фізичний контролер, сенсорне керування, тактильний зворотний зв'язок, ігрову сцену, локалізацію та допоміжні засоби користувацької взаємодії.

Проведене функціональне тестування підтвердило працездатність фінальної версії системи як цілісного програмно-апаратного комплексу. Порівняльне дослідження в межах визначених критеріїв показало зменшення затримки в роботі з IMU-даними, зниження ризику використання застарілих вимірювань, підвищення передбачуваності керування та доцільність переходу до алгоритмічної імітації нахилу лабіринту за статичної геометрії сцени. Також перевірено коректність процедурної генерації лабіринту для різних розмірів і придатність згенерованих рівнів до використання в грі. Дослідження тактильного каналу взаємодії показало підвищення якості тактильного зворотного зв'язку, зокрема в частині врахування сили та напрямку події, просторового розподілу сигналу між вібраційними модулями та стабільності його відтворення.

Отримані результати мають практичну цінність і можуть бути використані в навчальних, демонстраційних та ігрових програмно-апаратних системах, а також у дослідженнях людино-машинної взаємодії. Запропоновані алгоритмічні рішення можуть бути корисними під час створення інтерактивних систем із сенсорним керуванням, процедурно сформованим ігровим простором та тактильним супроводом подій. Окремі результати, пов'язані з використанням ігрових підходів у навчальних системах, відображено в публікації [34]. Результати дослідження було представлено на Щорічній НПК викладачів, науковців, молодих учених, аспірантів та студентів НУ «Запорізька політехніка» «Тиждень науки-2026» (13-17 квітня 2026 року) [6]. Тези за результатами роботи також були представлені на I Міжнародній науково-практичній конференції «Комп'ютерні ігри і мультимедіа, як інноваційний підхід до комунікації – 2026» (21–22 травня 2026 року) [5].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Krestanova A., Cerny M., Augustynek M. Review: Development and Technical Design of Tangible User Interfaces in Wide-Field Areas of Application. *Sensors*. 2021. Vol. 21, no. 13. Article 4258.
2. Shaer O., Hornecker E. Tangible User Interfaces: Past, Present, and Future Directions. *Foundations and Trends in Human-Computer Interaction*. 2010. Vol. 3, no. 1–2. P. 1–137.
3. Kok M., Hol J. D., Schön T. B. Using Inertial Sensors for Position and Orientation Estimation. *Foundations and Trends in Signal Processing*. 2017. Vol. 11, no. 1–2. P. 1–153.
4. Franček P., Jambrošić K., Horvat M., Planinec V. The Performance of Inertial Measurement Unit Sensors on Various Hardware Platforms for Binaural Head-Tracking Applications. *Sensors*. 2023. Vol. 23, no. 2. Article 872.
5. Верховецький М.Є., Туленков А.В., Пархоменко А.В. Розроблення та дослідження алгоритмів обчислень в інтерактивній комп'ютерній грі. *Комп'ютерні ігри та мультимедіа як інноваційний підхід до комунікації – 2026: матеріали I Міжнародної науково-практичної конференції. (21–22 травня 2026 р.)*. Одеса: ОНТУ, 2026. У друці.
6. Верховецький М. Є., Пархоменко А.В. Алгоритми опрацювання сенсорних даних при взаємодії з ігровим середовищем через матеріальний інтерфейс. *Тиждень науки-2026: збірник тез щорічної науково-практичної конференції викладачів, науковців, молодих учених, аспірантів та студентів (Запоріжжя, 13–17 квітня 2026 року)*. Запоріжжя: НУ «Запорізька політехніка», 2026. У друці.
7. Foottit J., Brown D., Marks S., Connor A. M. An Intuitive Tangible Game Controller. *Proceedings of the 2014 Conference on Interactive Entertainment: proceedings, 2–3 December 2014, Newcastle, Australia / New York: ACM, 2014*. Article 4.
8. See A. R., Choco J. A. G., Chandramohan K. Touch, Texture and Haptic

Feedback: A Review on How We Feel the World around Us. *Applied Sciences*. 2022. Vol. 12, no. 9. Article 4686.

9. Bianchi-Berthouze N., Kim W. W., Patel D. Does Body Movement Engage You More in Digital Game Play? And Why? *Affective Computing and Intelligent Interaction: Lecture Notes in Computer Science* / Eds. A. C. R. Paiva, R. Prada, R. W. Picard. Berlin: Springer, 2007. Vol. 4738. P. 102–113.

10. Lindley S. E., Le Couteur J., Berthouze N. L. Stirring up Experience through Movement in Game Play: Effects on Engagement and Social Behaviour. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: proceedings*, 5–10 April 2008, Florence, Italy / New York: ACM, 2008. P. 511–514.

11. Jung B., Schrader A., Carlson D. V. Tangible Interfaces for Pervasive Gaming. *Proceedings of DiGRA 2005 Conference: Changing Views: Worlds in Play: proceedings*, 16–20 June 2005, Vancouver, Canada. 2005. P. 261–265.

12. Lee J. T., Rajapakse R. P. C. J., Miyata K. HaptWarp: Soft Printable and Motion Sensible Game Controller. *Proceedings of the 2021 International Conference on Artificial Life and Robotics*. 2021. Vol. 26. P. 732–735.

13. Gui P., Tang L., Mukhopadhyay S. MEMS Based IMU for Tilting Measurement: Comparison of Complementary and Kalman Filter Based Data Fusion. *Proceedings of the 2015 10th IEEE Conference on Industrial Electronics and Applications (ICIEA 2015): proceedings*, 15–17 June 2015, Auckland, New Zealand / Piscataway: IEEE, 2015. P. 2004–2009.

14. Öz M. N., Budak S., Kurnaz E., Durdu A. Orientation Determination in IMU Sensor with Complementary Filter. *Turkish Journal of Forecasting*. 2022. Vol. 6, no. 1. P. 34–39.

15. Narkhede P., Poddar S., Walambe R., Ghinea G., Kotecha K. Cascaded Complementary Filter Architecture for Sensor Fusion in Attitude Estimation. *Sensors*. 2021. Vol. 21, no. 6. Article 1937.

16. Демонстрація попереднього варіанту програмно-апаратної системи інтерактивної гри TactiLab: відео. *YouTube*. URL:

<https://www.youtube.com/watch?v=c-rX5qZtM7M> (дата звернення: 11.05.2026).

17. Gabrovšek P. Analysis of Maze Generating Algorithms. *IPSI Transactions on Internet Research*. 2019. Vol. 15, no. 1. P. 23–30.

18. Pascher M., Franzen T., Kronhardt K., Gruenefeld U., Schneegass S., Gerken J. HaptiX: Vibrotactile Haptic Feedback for Communication of 3D Directional Cues. *Extended Abstracts of the 2023 CHI Conference on Human Factors in Computing Systems: proceedings*, 23–28 April 2023, Hamburg, Germany / New York: ACM, 2023. Article no. 154. P. 1–7.

19. Programming with C++ in Unreal Engine. Unreal Engine Documentation. *Epic Games*. URL: <https://dev.epicgames.com/documentation/unreal-engine/programming-with-cplusplus-in-unreal-engine> (date of access: 11.05.2026).

20. Blueprints Visual Scripting in Unreal Engine. Unreal Engine Documentation. *Epic Games*. URL: <https://dev.epicgames.com/documentation/unreal-engine/blueprints-visual-scripting-in-unreal-engine> (date of access: 11.05.2026).

21. Creating User Interfaces with UMG and Slate in Unreal Engine. Unreal Engine Documentation. *Epic Games*. URL: <https://dev.epicgames.com/documentation/unreal-engine/creating-user-interfaces-with-umg-and-slate-in-unreal-engine> (date of access: 11.05.2026).

22. Unity Manual. *Unity Documentation*. URL: <https://docs.unity3d.com/6000.4/Documentation/Manual/UnityManual.html> (date of access: 11.05.2026).

23. Built-in 3D Physics. *Unity Documentation*. URL: <https://docs.unity3d.com/6000.4/Documentation/Manual/PhysicsOverview.html> (date of access: 11.05.2026).

24. UI Toolkit. *Unity Documentation*. URL: <https://docs.unity3d.com/6000.4/Documentation/Manual/UIElements.html> (date of access: 11.05.2026).

25. Node3D. *Godot Engine Documentation*. URL:

https://docs.godotengine.org/en/stable/classes/class_node3d.html (date of access: 11.05.2026).

26. Control. *Godot Engine Documentation*. URL: https://docs.godotengine.org/en/stable/classes/class_control.html (date of access: 11.05.2026).

27. The GDExtension System. *Godot Engine Documentation*. URL: <https://docs.godotengine.org/en/stable/tutorials/scripting/gdextension/index.html> (date of access: 11.05.2026).

28. Visual Studio Tools for Unreal Engine overview. *Microsoft Learn*. URL: <https://learn.microsoft.com/en-us/visualstudio/gamedev/unreal/get-started/vs-tools-unreal-overview> (date of access: 11.05.2026).

29. Arduino Nano (V2.3): user manual. *Arduino*. URL: <https://www.arduino.cc/en/uploads/Main/ArduinoNanoManual23.pdf> (date of access: 11.05.2026).

30. MPU-6050. *TDK InvenSense*. URL: https://product.tdk.com/en/search/sensor/motion-inertial/imu/info?part_no=MPU-6050 (date of access: 11.05.2026).

31. Arduino IDE 2. *Arduino Documentation*. URL: <https://docs.arduino.cc/software/ide-v2> (date of access: 11.05.2026).

32. Unreal Engine Serial Communication Plugin. *GitHub*. URL: https://github.com/videofeedback/Unreal_Engine_SerialCOM_Plugin (date of access: 11.05.2026).

33. Localizing Content in Unreal Engine. *Unreal Engine Documentation*. *Epic Games*. URL: <https://dev.epicgames.com/documentation/unreal-engine/localizing-content-in-unreal-engine> (date of access: 11.05.2026).

34. Parkhomenko A., Tulenkov A., Verkhovetskyi M., Wolff C., Parkhomenko I., Gladkova O., Parkhomenko A. Gamification for Learning at the Intersection of Subject Areas. *Innovation via Collaborative Learning in Engineering Education*. / Auer, M.E., Toth, P. (eds) ICL 2025. Lecture Notes in Networks and Systems, vol 1847. Springer, Cham P. 1-8.

ДОДАТОК А**Тексти програмних компонентів системи**

A.1 Текст програми мікроконтролера Arduino Nano

```
#include <Wire.h>
#include <MPU6050.h>

MPU6050 mpu;

const int motor1Pin = 9; //>
const int motor2Pin = 10; //<
const int motor3Pin = 11; //^

int16_t ax_offset = 0, ay_offset = 0;
int16_t gx_offset = 0, gy_offset = 0, gz_offset = 0;

const byte BUFFER_SIZE = 16;
char inputBuffer[BUFFER_SIZE];
byte bufferIndex = 0;

byte timeMotor1 = 0;
byte timeMotor2 = 0;
byte timeMotor3 = 0;

byte powerMotor1 = 0;
byte powerMotor2 = 0;
byte powerMotor3 = 0;

void setup() {
    Serial.begin(115200);

    while (!Serial) {}

    Serial.println("INFO:Start_initialization");
    Serial.println("INFO:controller_version:1");
    Serial.println("INFO:firmware_version:4");

    Wire.begin();
    mpu.initialize();

    if (!mpu.testConnection()) {
        Serial.println("ERR:MPU6050_connection_failed");
        while (1);
    } else {
        Serial.println("INFO:MPU6050_connection");
    }

    pinMode(motor1Pin, OUTPUT);
    pinMode(motor2Pin, OUTPUT);
    pinMode(motor3Pin, OUTPUT);

    Serial.println("INFO:ready");

    startMotor();
}

void loop() {
    readSerialCommands();

    int16_t ax, ay, az;
    int16_t gx, gy, gz;
    mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);

    ax -= ax_offset;
    ay -= ay_offset;
```

```

    gx -= gx_offset;
    gy -= gy_offset;
    gz -= gz_offset;

    Serial.print("D:");
    Serial.print(ax); Serial.print(",");
    Serial.print(ay); Serial.print(",");
    Serial.print(az); Serial.print(",");
    Serial.print(gx); Serial.print(",");
    Serial.print(gy); Serial.print(",");
    Serial.println(gz);

    delay(10); // 25 = 40Hz, 10 - 100Hz
    motorSafety();
}

void readSerialCommands() {
    while (Serial.available()) {
        char c = Serial.read();

        if (c == '\n') {
            inputBuffer[bufferIndex] = '\0';
            handleCommand(inputBuffer);
            bufferIndex = 0;
        } else {
            if (bufferIndex < BUFFER_SIZE - 1) {
                inputBuffer[bufferIndex++] = c;
            } else {
                // Переполнения буфера - скидывания
                bufferIndex = 0;
            }
        }
    }
}

void stopMotor(uint8_t pin, byte& currentPower, byte& currentTime) {
    analogWrite(pin, 0);
    currentPower = 0;
    currentTime = 0;
}

void applyMotor(uint8_t pin, byte& currentPower, byte& currentTime, uint8_t
pwm, byte time) {
    if (pwm >= currentPower) {
        analogWrite(pin, pwm);
        currentPower = pwm;
        currentTime = time;
    }
}

// Motor duration is measured in loop ticks.
// Loop runs at ~100Hz (10ms per tick).
void controlMotor(uint8_t motor, uint8_t pwm, byte time = 10) { // default
duration: 10 ticks (~100ms at 100Hz)
    switch (motor) {
        case 1: applyMotor(motor1Pin, powerMotor1, timeMotor1, pwm, time); break;
        case 2: applyMotor(motor2Pin, powerMotor2, timeMotor2, pwm, time); break;
        case 3: applyMotor(motor3Pin, powerMotor3, timeMotor3, pwm, time); break;
        default: Serial.println("ERR:invalid_motor");
    }
}

void handleCommand(char* command) {

```

```

    if (strncmp(command, "M:", 2) == 0) {
        int m, v;
        if (sscanf(command, "M:%d:%d", &m, &v) == 2 && m >= 1 && m <= 3 && v >= 0
&& v <= 255) {
            controlMotor((uint8_t)m, (uint8_t)v);
        } else {
            Serial.println("ERR:invalid_command_format");
        }
    } else if (strcmp(command, "CAL") == 0) {
        calibrateSensors();
        Serial.println("INFO:calibration_done");
    } else {
        Serial.println("ERR:unknown_command");
        Serial.println(command);
    }
}

void motorSafety(){
    if(timeMotor1 > 0) {timeMotor1--; if(timeMotor1 == 0) stopMotor(motor1Pin,
powerMotor1, timeMotor1);}
    if(timeMotor2 > 0) {timeMotor2--; if(timeMotor2 == 0) stopMotor(motor2Pin,
powerMotor2, timeMotor2);}
    if(timeMotor3 > 0) {timeMotor3--; if(timeMotor3 == 0) stopMotor(motor3Pin,
powerMotor3, timeMotor3);}
}

void startMotor(){
    controlMotor(1,255,20); // 20 ticks (~200ms at 100Hz)
    controlMotor(2,255,20); // 20 ticks (~200ms at 100Hz)
    controlMotor(3,255,20); // 20 ticks (~200ms at 100Hz)
}

void calibrateSensors() {
    long sum_ax = 0, sum_ay = 0;
    long sum_gx = 0, sum_gy = 0, sum_gz = 0;
    const int samples = 200;

    for (int i = 0; i < samples; i++) {
        int16_t ax, ay, az;
        int16_t gx, gy, gz;
        mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);

        sum_ax += ax;
        sum_ay += ay;

        sum_gx += gx;
        sum_gy += gy;
        sum_gz += gz;

        delay(5);
    }

    ax_offset = sum_ax / samples;
    ay_offset = sum_ay / samples;

    gx_offset = sum_gx / samples;
    gy_offset = sum_gy / samples;
    gz_offset = sum_gz / samples;
}

```

A.2 C++-клас SerialCommandProcessor (.h)

```
#pragma once

#include "UObject/Object.h"
#include "CoreMinimal.h"
#include "UObject/NoExportTypes.h"
#include "SerialCommandProcessor.generated.h"

UENUM(BlueprintType)
enum class ESerialCommandType : uint8{
    Unknown,
    Data,
    Info,
    Error,
    CalibrationDone
};

USTRUCT(BlueprintType)
struct FSerialSensorData{
    GENERATED_BODY()

    UPROPERTY(BlueprintReadOnly)
    FVector Acc;

    UPROPERTY(BlueprintReadOnly)
    FVector Gyro;

    UPROPERTY(BlueprintReadOnly)
    bool bValid = false;
};

/**
 * Головний клас, який обробляє вхідні та вихідні команди серійного порту
 */
UCLASS(Blueprintable)
class TACTILAB_API USerialCommandProcessor : public UObject{
    GENERATED_BODY()

public:
    UFUNCTION(BlueprintCallable)
    ESerialCommandType ParseLine(const FString& Line, FSerialSensorData&
OutData);

    UFUNCTION(BlueprintCallable)
    FString MakeMotorCommand(int32 MotorId, uint8 Power);

    UFUNCTION(BlueprintCallable)
    FString MakeCalibrationCommand();

    UFUNCTION(BlueprintCallable, Category = "Serial")
    TArray<FString> GetAvailableCOMPorts();
};
```

A.3 C++-клас SerialCommandProcessor (.cpp)

```
#include "SerialCommandProcessor.h"
#include "Windows/AllowWindowsPlatformTypes.h"
#include <windows.h>
#include "Windows/HideWindowsPlatformTypes.h"
```

```

ESerialCommandType USerialCommandProcessor::ParseLine(const FString& Line,
FSerialSensorData& OutData){
    if (Line.StartsWith("D:")) {
        TArray<FString> Parts;
        Line.Mid(2).ParseIntoArray(Parts, TEXT(","), true);
        if (Parts.Num() == 6){
            OutData.Acc.X = FString::Atoi(*Parts[0]);
            OutData.Acc.Y = FString::Atoi(*Parts[1]);
            OutData.Acc.Z = FString::Atoi(*Parts[2]);

            OutData.Gyro.X = FString::Atoi(*Parts[3]);
            OutData.Gyro.Y = FString::Atoi(*Parts[4]);
            OutData.Gyro.Z = FString::Atoi(*Parts[5]);

            OutData.bValid = true;

            return ESerialCommandType::Data;
        }
    }
    else if (Line.StartsWith("INFO:calibration_done")){
        return ESerialCommandType::CalibrationDone;
    }
    else if (Line.StartsWith("INFO:")){
        return ESerialCommandType::Info;
    }
    else if (Line.StartsWith("ERR:")){
        return ESerialCommandType::Error;
    }

    return ESerialCommandType::Unknown;
}

FString USerialCommandProcessor::MakeMotorCommand(int32 MotorId, uint8
Power){
    return FString::Printf(TEXT("M:%d:%d"), MotorId, Power);
}

FString USerialCommandProcessor::MakeCalibrationCommand(){
    return TEXT("CAL"); // \n
}

TArray<FString> USerialCommandProcessor::GetAvailableCOMPorts(){
    TArray<FString> PortList;

    HKEY hKey;
    if (RegOpenKeyExA(HKEY_LOCAL_MACHINE, "HARDWARE\\DEVICEMAP\\SERIALCOMM",
0, KEY_READ, &hKey) == ERROR_SUCCESS){
        char valueName[256];
        BYTE data[256];
        DWORD valueNameSize, dataSize, valueType;
        DWORD index = 0;

        while (true){
            valueNameSize = sizeof(valueName);
            dataSize = sizeof(data);
            if (RegEnumValueA(hKey, index, valueName, &valueNameSize,
nullptr, &valueType, data, &dataSize) != ERROR_SUCCESS)
                break;

            if (valueType == REG_SZ){
                FString PortName =
FString(ANSI_TO_TCHAR(reinterpret_cast<const char*>(data)));
                if (PortName.StartsWith("COM")){
                    PortList.Add(PortName);
                }
            }
        }
    }
}

```

```

        }
    }
    ++index;
}
RegCloseKey(hKey);
}

return PortList;
}

```

A.4 C++-клас SensorDataInterpreter (.h)

```

#pragma once

#include "CoreMinimal.h"
#include "UObject/Object.h"
#include "SerialCommandProcessor.h"
#include "SensorDataInterpreter.generated.h"

UCLASS(Blueprintable)
class TACTILAB_API USensorDataInterpreter : public UObject{
    GENERATED_BODY()

private:
    FVector SmoothVectorEMA(const FVector& In, FVector& State, float Alpha);
    FRotator ApplyComplementaryFilter(const FVector& SmoothedAccel, const
FVector& SmoothedGyro, float DeltaTime);

    FVector AccelState = FVector::ZeroVector;
    FVector GyroState = FVector::ZeroVector;
    float GyroAnglePitch = 0.0f;
    float GyroAngleRoll = 0.0f;
    FRotator LastValidRotation = FRotator::ZeroRotator;

public:
    /** Перетворює Асс в нормалізований вектор (g) */
    UFUNCTION(BlueprintCallable)
    FVector ConvertAccelToVector(const FSerialSensorData& Data);

    /** Перетворює Gyro в нормалізований вектор (deg/s) */
    UFUNCTION(BlueprintCallable)
    FVector ConvertGyroToVector(const FSerialSensorData& Data);

    /** Повертає згладжений ротатор */
    UFUNCTION(BlueprintCallable)
    FRotator GetStableRotation(const FSerialSensorData& Data, float
DeltaTime);

    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Control")
    float MaxTiltAngle = 40.f;

};

```

A.5 C++-клас SensorDataInterpreter (.cpp)

```

#include "SensorDataInterpreter.h"

FVector USensorDataInterpreter::ConvertAccelToVector(const FSerialSensorData&
Data){
    constexpr float AccelScale = 1.0f / 16384.0f; // +-2g
    return FVector(

```

```

        static_cast<float>(Data.Acc.Y) * AccelScale,
        static_cast<float>(Data.Acc.X) * AccelScale,
        static_cast<float>(Data.Acc.Z) * AccelScale
    );
}

FVector USensorDataInterpreter::ConvertGyroToVector(const FSerialSensorData&
Data){
    constexpr float GyroScale = 1.0f / 131.0f; // +/-250 dps
    return FVector(
        static_cast<float>(Data.Gyro.X) * GyroScale,
        static_cast<float>(Data.Gyro.Y) * GyroScale,
        static_cast<float>(Data.Gyro.Z) * GyroScale
    );
}

FVector USensorDataInterpreter::SmoothVectorEMA(const FVector& In, FVector&
State, float Alpha){
    State = Alpha * In + (1.0f - Alpha) * State;
    return State;
}

FRotator USensorDataInterpreter::ApplyComplementaryFilter(const FVector&
SmoothedAccel, const FVector& SmoothedGyro, float DeltaTime){
    constexpr float Alpha = 0.96f;

    // Кути з акселерометра (відносно гравітації)
    float AccelPitch = FMath::RadiansToDegrees(FMath::Atan2(SmoothedAccel.X,
FMath::Sqrt(FMath::Square(SmoothedAccel.Y)
+
FMath::Square(SmoothedAccel.Z))));
    float AccelRoll = FMath::RadiansToDegrees(FMath::Atan2(-SmoothedAccel.Y,
SmoothedAccel.Z));

    // Інтерполяція гіроскопа
    GyroAnglePitch += SmoothedGyro.X * DeltaTime;
    GyroAngleRoll += SmoothedGyro.Y * DeltaTime;

    // Комплементарне злиття
    float Pitch = Alpha * GyroAnglePitch + (1.0f - Alpha) * AccelPitch;
    float Roll = Alpha * GyroAngleRoll + (1.0f - Alpha) * AccelRoll;

    // Оновлення
    GyroAnglePitch = Pitch;
    GyroAngleRoll = Roll;

    return FRotator(Pitch, 0.0f, Roll);
}

FRotator USensorDataInterpreter::GetStableRotation(const FSerialSensorData&
Data, float DeltaTime){
    if (Data.bValid) {
        FVector RawAccel = ConvertAccelToVector(Data);
        FVector RawGyro = ConvertGyroToVector(Data);

        FVector SmoothedAccel = SmoothVectorEMA(RawAccel, AccelState, 0.15f);
        FVector SmoothedGyro = SmoothVectorEMA(RawGyro, GyroState, 0.1f);

        FRotator TargetRotation = ApplyComplementaryFilter(SmoothedAccel,
SmoothedGyro, DeltaTime);

        LastValidRotation = TargetRotation;
    }
}

```

```

        LastValidRotation.Pitch    =    FMath::Clamp(LastValidRotation.Pitch,    -
MaxTiltAngle, MaxTiltAngle);
        LastValidRotation.Roll    =    FMath::Clamp(LastValidRotation.Roll,    -
MaxTiltAngle, MaxTiltAngle);

    return LastValidRotation;
}

```

A.6 C++-клас MazeGeneratorActor (.h)

```

#pragma once

#include "CoreMinimal.h"
#include "GameFramework/Actor.h"
#include "MazeGeneratorActor.generated.h"

UCLASS()
class TACTILAB_API AMazeGeneratorActor : public AActor{
    GENERATED_BODY()

public:
    AMazeGeneratorActor();

private:
    TArray<TArray<int32>>> MazeGrid;
    FRandomStream RandomStream;

    UPROPERTY()
    UInstancedStaticMeshComponent* BlockInstancer;

    UPROPERTY()
    UInstancedStaticMeshComponent* FloorInstancer;

    UPROPERTY()
    UInstancedStaticMeshComponent* HoleInstancer;

    UPROPERTY()
    AActor* FinishActorInstance;

protected:
    virtual void BeginPlay() override;

public:
    // Blueprint callable функція генерації
    UFUNCTION(BlueprintCallable, Category = "Maze")
    void Generate(int32 CustomSeed = -1);

    // Допоміжні внутрішні методи
    void ClearMaze();
    void InitMazeGrid();
    void GeneratePaths();
    bool IsValidCell(int32 X, int32 Y);
    void VisitCell(int32 X, int32 Y);
    void PlaceMazeMeshes();
    void AddWallConnections();

    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Maze")
    int32 MazeSize = 10; // Мінімум 5, максимум 15

    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Maze")
    int32 Seed = -1; // -1 = random

    UPROPERTY(EditAnywhere, Category = "Maze")

```

```

    UStaticMesh* BlockMesh;

    UPROPERTY(EditAnywhere, Category = "Maze")
    UStaticMesh* FloorMesh;

    UPROPERTY(EditAnywhere, Category = "Maze")
    UStaticMesh* HoleMesh;

    UPROPERTY(EditAnywhere, Category = "Maze")
    TSubclassOf<AActor> FinishActorClass;
};

```

A.7 C++-класс MazeGeneratorActor (.cpp)

```

#include "MazeGeneratorActor.h"
#include "Components/InstancedStaticMeshComponent.h"
#include "Kismet/KismetMathLibrary.h"
#include "Engine/World.h"

AMazeGeneratorActor::AMazeGeneratorActor() {
    PrimaryActorTick.bCanEverTick = false;

    RootComponent = CreateDefaultSubobject<USceneComponent>(TEXT("Root"));

    BlockInstancer =
    CreateDefaultSubobject<UInstancedStaticMeshComponent>(TEXT("BlockInstancer"));
    ;
    BlockInstancer->SetupAttachment(RootComponent);

    FloorInstancer =
    CreateDefaultSubobject<UInstancedStaticMeshComponent>(TEXT("FloorInstancer"));
    ;
    FloorInstancer->SetupAttachment(RootComponent);

    HoleInstancer =
    CreateDefaultSubobject<UInstancedStaticMeshComponent>(TEXT("HoleInstancer"));
    HoleInstancer->SetupAttachment(RootComponent);
}

void AMazeGeneratorActor::BeginPlay() {
    Super::BeginPlay();

    BlockInstancer->SetStaticMesh(BlockMesh);
    FloorInstancer->SetStaticMesh(FloorMesh);
    HoleInstancer->SetStaticMesh(HoleMesh);
}

void AMazeGeneratorActor::Generate(int32 CustomSeed) {
    Seed = (CustomSeed == -1) ? FMath::Rand() : CustomSeed;
    UE_LOG(LogTemp, Warning, TEXT("MazeGenerator: Using seed %d"), Seed);

    RandomStream.Initialize(Seed);

    ClearMaze();
    InitMazeGrid();
    GeneratePaths();
    PlaceMazeMeshes();
    AddWallConnections();
}

void AMazeGeneratorActor::ClearMaze() {
    BlockInstancer->ClearInstances();
    FloorInstancer->ClearInstances();
}

```

```

HoleInstancer->ClearInstances();

if (FinishActorInstance && FinishActorInstance->IsValidLowLevel()){
    FinishActorInstance->Destroy();
    FinishActorInstance = nullptr;
}

}

void AMazeGeneratorActor::InitMazeGrid(){
    MazeSize = FMath::Clamp(MazeSize, 5, 15);
    const int32 GridSize = MazeSize * 2 + 1;

    MazeGrid.SetNum(GridSize);
    for (int32 i = 0; i < GridSize; i++){
        MazeGrid[i].SetNum(GridSize);
        for (int32 j = 0; j < GridSize; j++){
            if (i == 0 || j == 0 || i == GridSize - 1 || j == GridSize -
1){
                MazeGrid[i][j] = 1;
            }
            else{
                MazeGrid[i][j] = 0;
            }
        }
    }
}

// Перевірка: чи координати в межах лабіринту
bool AMazeGeneratorActor::IsValidCell(int32 X, int32 Y){
    int32 GridSize = MazeSize * 2 + 1;
    return X > 0 && Y > 0 && X < GridSize - 1 && Y < GridSize - 1;
}

// Рекурсивний DFS
void AMazeGeneratorActor::VisitCell(int32 X, int32 Y){
    MazeGrid[X][Y] = 0; // поточна клітинка – прохід

    TArray<FIntPoint> Directions = {
        FIntPoint(2, 0),    // Вправо
        FIntPoint(-2, 0),   // Вліво
        FIntPoint(0, 2),    // Вгору
        FIntPoint(0, -2)    // Вниз
    };

    // Перемішуємо напрямки
    for (int32 i = 0; i < Directions.Num(); ++i) {
        int32 RandIndex = RandomStream.RandRange(i, Directions.Num() - 1);
        Directions.Swap(i, RandIndex);
    }

    // DFS
    for (FIntPoint Dir : Directions) {
        int32 NX = X + Dir.X;
        int32 NY = Y + Dir.Y;

        if (IsValidCell(NX, NY) && MazeGrid[NX][NY] == 1) {
            MazeGrid[X + Dir.X / 2][Y + Dir.Y / 2] = 0; // Пропізаємо
стіну між
            VisitCell(NX, NY);
        }
    }
}

void AMazeGeneratorActor::GeneratePaths(){

```

```

int32 GridSize = MazeSize * 2 + 1;

// Спочатку все – стіни
MazeGrid.SetNum(GridSize);
for (int32 x = 0; x < GridSize; ++x) {
    MazeGrid[x].SetNum(GridSize);
    for (int32 y = 0; y < GridSize; ++y) {
        MazeGrid[x][y] = 1;
    }
}

// Стартуємо з центру
int32 Center = (GridSize / 2) | 1;
VisitCell(Center, Center);

// Очищення зони 3 на 3 в центрі
for (int32 dx = -1; dx <= 1; dx++){
    for (int32 dy = -1; dy <= 1; dy++){
        MazeGrid[Center + dx][Center + dy] = 0;
    }
}

// Встановлення виходу
for (int32 attempt = 0; attempt < 100; ++attempt) {
    int32 x, y;

    if (RandomStream.RandRange(0, 1) == 1) {
        x = (GridSize - 1) * RandomStream.RandRange(0, 1);
        y = RandomStream.RandRange(1, GridSize - 2);
    }
    else {
        x = RandomStream.RandRange(1, GridSize - 2);
        y = (GridSize - 1) * RandomStream.RandRange(0, 1);
    }

    int32 nx = (x == 0) ? 1 : (x == GridSize - 1) ? GridSize - 2 : x;
    int32 ny = (y == 0) ? 1 : (y == GridSize - 1) ? GridSize - 2 : y;

    if (MazeGrid[nx][ny] == 0) {
        MazeGrid[x][y] = 3;
        break;
    }
}

// Створення отворів замість стін з вірогідністю 5% і замість підлоги з
шансом 0.5%
for (int32 x = 1; x < GridSize - 2; ++x) {
    for (int32 y = 1; y < GridSize - 2; ++y) {
        if (MazeGrid[x][y] == 1 && RandomStream.RandRange(0, 19) ==
0) {
            MazeGrid[x][y] = 2; // Отвір з шансом 5%
        }
        if (MazeGrid[x][y] == 0 && RandomStream.RandRange(0, 200) ==
0) {
            MazeGrid[x][y] = 2; // Отвір з шансом 0.5%
        }
    }
}

void AMazeGeneratorActor::PlaceMazeMeshes() {
    const int32 GridSize = MazeGrid.Num();
    const float TileSize = 100.f;

```

```

    const FVector Origin = GetActorLocation() - FVector((GridSize / 2) *
    TileSize, (GridSize / 2) * TileSize, 0);

    for (int32 x = 0; x < GridSize; x++){
        for (int32 y = 0; y < GridSize; y++){
            const FVector Position = Origin + FVector(x * TileSize, y *
    TileSize, 0);

            switch (MazeGrid[x][y]){
            case 0: // Порожньо
                if (FloorMesh){
                    FTransform TF(Position + FVector(0, 0, -20));
                    FloorInstancer->AddInstance(TF);
                }
                break;

            case 1: // Стіна (Блок)
                if (BlockMesh && FloorMesh){
                    FTransform FloorTF(Position + FVector(0, 0, -
    20));

                    FloorInstancer->AddInstance(FloorTF);

                    FTransform BlockTF(Position);
                    BlockInstancer->AddInstance(BlockTF);
                }
                break;

            case 2: // Отвір
                if (HoleMesh){
                    FTransform HoleTF(Position + FVector(0, 0, -
    10));

                    HoleInstancer->AddInstance(HoleTF);
                }
                break;

            case 3: // Вихід
                if (FloorMesh){
                    FTransform TF(Position + FVector(0, 0, -20));
                    FloorInstancer->AddInstance(TF);
                }
                if (FinishActorClass) {
                    FRotator OriginalRotation = GetActorRotation();
                    SetActorRotation(FRotator::ZeroRotator);
                    FActorSpawnParameters Params;
                    Params.Owner = this;
                    FVector SpawnLocation = Position + FVector(0, 0,
    50);

                    FinishActorInstance = GetWorld()-
    >SpawnActor<AActor>(FinishActorClass, SpawnLocation, FRotator::ZeroRotator,
    Params);

                    if (FinishActorInstance) {
                        FinishActorInstance->AttachToActor(this,
    FAttachmentTransformRules::KeepWorldTransform);
                        FinishActorInstance->Tags.Add("Finish");
                    }
                    SetActorRotation(OriginalRotation);
                }
                break;
            }
        }
    }
}

void AMazeGeneratorActor::AddWallConnections() {

```

```

const int32 GridSize = MazeGrid.Num();
const float TileSize = 100.f;
const FVector Origin = GetActorLocation() - FVector((GridSize / 2) *
TileSize, (GridSize / 2) * TileSize, 0);

for (int32 x = 0; x < GridSize; x++) {
    for (int32 y = 0; y < GridSize; y++) {
        if (MazeGrid[x][y] != 1)
            continue;

        if (x + 1 < GridSize && MazeGrid[x + 1][y] == 1) {
            FVector Mid = Origin + FVector((x + 0.5f) * TileSize,
y * TileSize, 0);
            BlockInstancer->AddInstance(FTransform(Mid));
        }

        if (y + 1 < GridSize && MazeGrid[x][y + 1] == 1) {
            FVector Mid = Origin + FVector(x * TileSize, (y +
0.5f) * TileSize, 0);
            BlockInstancer->AddInstance(FTransform(Mid));
        }
    }
}

```

ДОДАТОК Б
Слайди презентації



Рисунок Б.1 – Слайд 1



Рисунок Б.2 – Слайд 2

Проблематика попереднього варіанту системи

3

Таблиця 1 – Аналіз обмежень попереднього варіанту системи та напрямів удосконалення.

Рішення в попередній версії	Виявлене обмеження	Напрямок удосконалення
Обертання віртуального лабіринту	Зайве навантаження та складність узгодження руху сцени й кульки	Статичний лабіринт, штучний вектор сили та візуальна імітація нахилу
Зчитування СОМ-даних частинами	Накопичення затримки та неповне використання потоку даних	Пакетне зчитування всіх доступних рядків із буфера
Додаткове згладжування після фільтрації	Маскування проблеми приймання даних і надмірне згладжування реакції	Покращення приймання даних і відмова від надмірного згладжування
Базовий вібраційний зворотний зв'язок	Недостатньо виразний просторовий характер відгуку	Уточнення напрямку взаємодії, просторовий розподіл між модулями та стабілізація відгуку

Рисунок Б.3 – Слайд 3

Структура системи та потоки даних

4



Загальна структура системи



Взаємодія підсистем і потоки даних

Рисунок Б.4 – Слайд 4

Функціональна схема та вибір алгоритмічних підходів

5

Таблиця 2 – Узагальнення вибору підходів для алгоритмічних блоків

Алгоритмічний блок	Обраний підхід	Обґрунтування
Передавання, приймання та первинне оброблення сенсорних даних	Текстовий пакет і буферизоване приймання рядків	Простий обмін, своєчасне приймання вимірювань, зменшення затримки
Оброблення та інтерпретація IMU-даних	EMA та комплементарний фільтр	Зменшення шуму та стійкі параметри керування без надмірного навантаження
Імітація нахилу лабіринту	Статичний лабіринт, штучний вектор сили, камера й освітлення	Менше трансформацій сцени, стабільніше й менш ресурсоємне керування
Процедурна генерація лабіринту	DFS і випадкове перемішування напрямків	Зв'язний і варіативний лабіринт при простій реалізації
Просторовий тактильний зворотний зв'язок	Розподіл сигналу між трьома вібротворами	Передавання сили й напрямку контакту, а не лише факту події



Функціональна схема системи

Рисунок Б.5 – Слайд 5

Обґрунтування вибору ігрового рушія

6

Таблиця 3 – Порівняння ігрових рушіїв для реалізації програмної частини системи

Критерій	Unreal Engine 5	Unity	Godot	Висновок
3D-сцена та фізична модель	Розвинені засоби 3D, фізики, камери й освітлення	Підтримує 3D і фізику	Підтримує 3D і базову фізику	Усі придатні, UE5 відповідає архітектурі системи
Візуальна логіка та код	Blueprint + C++	C#, візуальна логіка через додаткові засоби	GDScript, C#, GDExtension	UE5 зручний для поєднання Blueprint і C++
Інтерфейс користувача	UMG / Slate	UI Toolkit	Control-вузли	Усі мають UI, UE5 узгоджується з реалізацією системи
Інтеграція з фізичним контролером	Плагін серійного обміну + C++/Blueprint	C# або сторонні засоби serial-обміну	Розширення або зовнішні модулі	UE5 уже має використану інтеграцію з COM-портом
Відповідність попередньому варіанту системи	Система вже реалізована в UE5	Потрібне перенесення сцени й логіки	Потрібна повна перебудова	UE5 дозволяє зосередитися на алгоритмах, а не на перенесенні системи

Рисунок Б.6 – Слайд 6



Рисунок Б.7 – Слайд 7



Рисунок Б.8 – Слайд 8



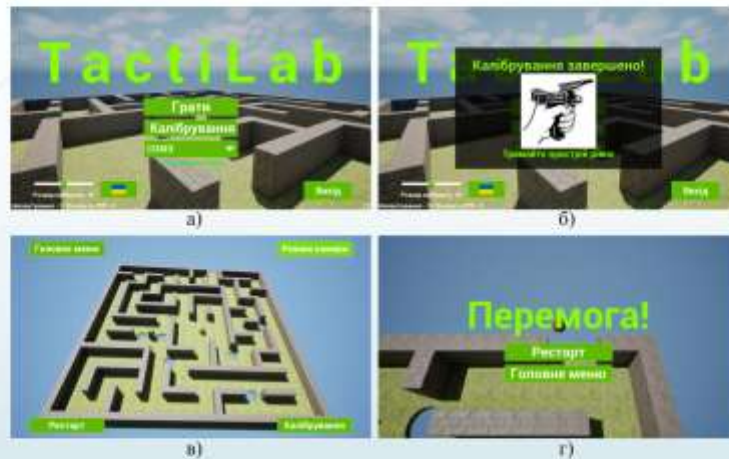
Рисунок Б.9 – Слайд 9



Рисунок Б.10 – Слайд 10

11

Реалізація користувацької взаємодії



а – головне меню; б – екран калібрування;
в – ігровий інтерфейс; г – екран завершення гри

Основні екрани інтерфейсу користувача системи

Рисунок Б.11 – Слайд 11

12

Дослідження розроблених алгоритмічних рішень

Таблиця 4 – Узагальнення результатів дослідження розроблених алгоритмічних рішень

Алгоритмічне рішення	Попередній варіант / перевірка	Вдосконалена версія / результат
Робота з IMU-даними	Часткове зчитування COM-потoku, ризик накопичення затримки та використання застарілих даних	Буферизоване зчитування доступних рядків, своєчасніше приймання даних і кращий баланс між плавністю та швидкістю реакції
Імітація нахилу лабіринту	Нахил відтворювався через постійну зміну трансформацій геометрії лабіринту	Геометрія залишається статичною; нахил імітується через штучний силовий вплив, камеру та освітлення
Процедурна генерація лабіринту	Перевірялась генерація для розмірів $n=5$, $n=10$, $n=15$, стартової зони, виходу, отворів і повторної генерації	Алгоритм формує зв'язні та варіативні лабіринти в заданому діапазоні розмірів; результат придатний до проходження
Просторовий тактильний зворотний зв'язок	Базовий відгук переважно повідомляв факт події та недостатньо передавав силу й напрямок взаємодії	Сигнал розподіляється між трьома модулями з урахуванням сили, напрямку, нелінійної відчутності та пріоритету сильнішого сигналу

Рисунок Б.12 – Слайд 12

13

Висновки та апробація результатів

- ❖ Сформульовано вимоги до програмно-апаратної системи, визначено її структуру, підсистеми та функціональну схему.
- ❖ Розроблено алгоритми передавання, приймання та первинного оброблення сенсорних даних, оброблення та інтерпретації IMU-даних, імітації нахилу лабіринту, процедурної генерації та просторового тактильного зворотного зв'язку.
- ❖ Реалізовано інтерактивну комп'ютерну гру з матеріальним інтерфейсом на базі Unreal Engine 5, Arduino Nano, MPU-6050 та вібраційних модулів.
- ❖ Проведене тестування підтвердило працездатність системи, а порівняльне дослідження показало підвищення стабільності керування, зменшення ризику використання застарілих IMU-даних і покращення якості тактильного зворотного зв'язку.
- ❖ Отримані результати можуть бути використані в навчальних, демонстраційних та ігрових програмно-апаратних системах.

Апробація результатів:

- ❖ Щорічна НПК викладачів, науковців, молодих учених, аспірантів та студентів НУ «Запорізька політехніка» «Тиждень науки-2026», 13–17 квітня 2026 року.
- ❖ І Міжнародна науково-практична конференція «Комп'ютерні ігри і мультимедіа, як інноваційний підхід до комунікації – 2026», 21–22 травня 2026 року.

Рисунок Б.13 – Слайд 13