МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВОДНОГО ГОСПОДАРСТВА ТА ПРИРОДОКОРИСТУВАННЯ

Навчально-науковий інститут кібернетики, інформаційних технологій та інженерії

Кафедра комп'ютерних наук та прикладної математики

"До захисту допущена"
Зав. кафедри комп'ютерних наук та
прикладної математики
д.т н., проф. Ю.В. Турбал
«____» _____ 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

Розробка мережевого рішення для ігрових додатків в середовищі Unity з інтеграцією унікального контролера на базі Arduino

Виконав: Осійчук Дмитро Ярославович _____
(прізвище, ім'я, по батькові) (підпис)

група ІПЗ-41

Керівник: к.т.н., доцент, доцент Остапчук О. П _____
(науковий ступінь, вчене звання, посада, прізвище, ініціали) (підпис)

Рівне – 2024

Національний університет водного господарства та природокористування

(повне найменування вищого навчального закладу)

Навчально-науковий інститут кібернетики, інформаційних технологій та інженерії

Кафедра **комп'ютерних наук та прикладної математики**
Освітньо-кваліфікаційний рівень **бакалавр**
Галузь знань **12 Інформаційні технології**
(шифр і назва)
Спеціальність **121 Інженерія програмного забезпечення**
(шифр і назва)
Спеціалізація –

«ЗАТВЕРДЖУЮ»
Завідувач кафедри
д.т н., проф. Ю.В. Турбал
"1" жовтня 2023 року

З А В Д А Н Н Я
НА КВАЛФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Осійчуку Дмитру Ярославовичу
(прізвище, ім'я, по батькові)

1. Тема роботи *"Розробка мережевого рішення для ігрових додатків в середовищі Unity з інтеграцією унікального контролера на базі Arduino"*

керівник роботи *Остапчук Оксана Петрівна, к.т.н., доцент, д кафедри комп'ютерних наук та прикладної математики.*

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

затверджені наказом по університету від *"22" квітня 2024 року С №-525.*

2. Термін подання роботи студентом *1 червня 2024 року.*

3. Вихідні дані до роботи: *технології, які необхідні для розробки мережевого рішення, апаратний мікроконтролер на базі Arduino та список основних завдань і вимог щодо основних функціональних можливостей.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) *розробити прототип мережевого рішення, зібрати унікальний контролер, налаштувати їх функціональні можливості. Провести тестування та оптимізацію.*

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

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Розділ 1	*доцент Остапчук О.П.*	*10.10.24*	*10.10.24*
Розділ 2	*доцент Остапчук О.П.*	*15.01.24*	*15.01.24*
Розділ 3	*доцент Остапчук О.П.*	*04.03.24*	*04.03.24*

7. Дата видачі завдання *01 жовтня 2023 р.*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	*Вивчення літератури за обраною тематикою*	*02.10.23 – 20.10.23*	*виконав*
2	*Розробка прототипу мережевого рішення*	*23.10.23 – 08.12.23*	*виконав*
3	*Розробка функціональних можливостей мережевого рішення*	*11.12.23 – 26.01.24*	*виконав*
4	*Розробка апаратного контролера*	*29.01.24 – 23.02.24*	*виконав*
5	*Створення гри з використанням рішення*	*26.02.24 – 18.03.24*	*виконав*
6	*Підключення контролера до гри*	*18.03.24 – 22.03.24*	*виконав*
8	*Загальні висновки до роботи*	*25.03.24 – 26.04.24*	*виконав*
9	*Підготовка звіту магістерської роботи*	*01.04.24 – 01.06.24*	*виконав*
10	*Підготовка мультимедійної презентації*	*03.06.24 –06.06.24*	*виконав*
11	*Підготовка до виступу*	*07.06.24 –10.06.24*	*виконав*

Студент _____ **Осійчук Д.Я.**
(підпис) (прізвище та ініціали)

Керівник роботи _____ **Остапчук О.П.**
(підпис) (прізвище та ініціали)

ЗМІСТ

РЕФЕРАТ

Кваліфікаційна робота: 47 с., 4 рисунки, 15 джерел.

Мета роботи: розробка мережевого рішення для ігрових додатків у середовищі Unity з інтеграцією унікального контролера на базі Arduino. Основні завдання включають в себе вивчення теоретичних аспектів ігрових двигунів та середовища Unity, архітектури та компонентів Unity. Розробка методології дослідження та експериментальне дослідження з подальшим аналізом отриманих результатів.

Об'єкт дослідження – мережеві рішення та апаратні контролери у мультиплеєрних ігрових додатках.

Предмет дослідження – механізми і технології, які використовуються для забезпечення мережевої взаємодії та контролю в мультиплеєрних іграх, а також способи інтеграції апаратних контролерів у ігровий процес.

Методи вивчення – мережеві протоколи, програмування апаратних систем та ігровий рушій Unity .

Проведено дослідження мережевих протоколів, мікроконтролерів та ігрового рушія Unity для розробки мультиплеєрних ігрових додатків. Проаналізовано функціональні можливості кожного компонента та їх можливі взаємодії в контексті ігрового середовища. Розроблено модель ігрового додатку, що базується на цих методах інтеграції, та проведено тестування її ефективності та функціональності в реальних умовах.

Ключові слова: МЕРЕЖЕВІ ПРОТОКОЛИ, МІКРОКОНТРОЛЕРИ, UNITY 3D, МУЛЬТИПЛЕЄРНІ ІГРИ, ВІДЕОІГРИ.

ВСТУП

Сучасна ігрова індустрія переживає епоху інтенсивного розвитку і конкуренції. З кожним роком користувачі стають все більш вибагливішими, і розробники змушені постійно вдосконалювати свої продукти, пропонуючи нові ідеї та технології [1]. У зв'язку з цим мережеві рішення для ігрових додатків постійно вдосконалюються, щоб забезпечити стабільну та ефективну взаємодію між гравцями. Метою цієї роботи є вивчення та розробка мережевого рішення для ігрових додатків в середовищі Unity. А для покращення взаємодії під час ігрової сесії було створено та використано унікальний контролер на базі Arduino.

Щоб спроєктувати і розробити таке рішення, необхідно володіти глибоким розумінням не тільки технічних аспектів програмування і мережевих протоколів, але і деталей ігрового процесу користувача. Дана робота має на меті створити цікаве і захоплююче ігрове середовище, яке було б не тільки функціональним, але й сприяло б активній взаємодії між гравцями та отриманню більшого задоволення від гри.

Робота над проєктом включає в себе аналіз новітніх мережевих технологій в ігровій індустрії, вивчення можливостей і обмежень середовища розробки Unity, а також вивчення можливостей створення унікального ігрового процесу з використанням апаратних контролерів на базі Arduino. Це досягнення не тільки допоможе поліпшити мережеві рішення існуючих ігор, але і посприяє розвитку нових технологій і підходів в області розробки ігор.

Незважаючи на важливість мережевих рішень у сучасних ігрових додатках, є деякі проблеми, які потребують уваги. Перш за все, це проблема гнучкості і складності розробки мережевої системи. Ігрові проєкти, як правило, вимагають великої кількості коду для ефективної реалізації мережі, що може призвести до збільшення розміру програми та зниження продуктивності.

Актуальність теми зумовлена необхідністю вирішення кількох ключових проблем. По-перше, розробка мережевої системи для ігрових проєктів є складною та потребує значної кількості коду, що може збільшити розмір

програми та знизити продуктивність. По-друге, вартість створення та обслуговування мережевої інфраструктури може бути високою, вимагаючи значних інвестицій в апаратне забезпечення, програмне забезпечення та інженерні рішення. Ця робота спрямована на вирішення зазначених проблем шляхом розробки ефективних та оптимізованих мережевих рішень, доступних широкому колу розробників з урахуванням їх фінансових можливостей.

Метою роботи є розробка мережевого рішення для ігрових додатків у середовищі Unity з інтеграцією унікального контролера на базі Arduino.

Завданням роботи є розробка мережевого плагіну для Unity з інтеграцією унікального контролера на базі Arduino, провести його тестування та аналіз ефективності.

Об’єкт дослідження – мережеві рішення та апаратні контролери у мультиплеєрних ігрових додатках.

Предмет дослідження – механізми і технології, які використовуються для забезпечення мережевої взаємодії та контролю в мультиплеєрних іграх, а також способи інтеграції апаратних контролерів у ігровий процес.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ОСНОВИ МЕРЕЖЕВИХ РІШЕНЬ ТА АПАРАТНИХ КОНТРОЛЕРІВ ДЛЯ ІГРОВИХ ДОДАТКІВ

1.1. Визначення та пояснення концепцій мережевих рішень в ігрових додатках

Мережеві рішення для ігрових додатків – це набір складних технологій, протоколів і алгоритмів, спрямованих на забезпечення взаємодії між користувачами за допомогою мережевого середовища. Даний розділ присвячений аналізу і розкриттю ключових концепцій і принципів, які лежать в основі мережевих рішень в сучасних ігрових додатках.

Одним з ключових аспектів є архітектура мережевих рішень. Вона включає в себе вибір між клієнт-серверною та peer-to-peer архітектурами, кожна з яких має свої переваги та недоліки. У клієнт-серверній архітектурі сервер відповідає за координацію гри та зберігання даних, тоді як клієнти лише отримують та відправляють дані. У peer-to-peer архітектурі всі пристрої взаємодіють без посередництва центрального сервера. Комбінована архітектура, як можна здогадатися з назви, поєднує переваги обох попередніх архітектур. Вона може використовувати централізовані сервери для надання основних послуг, але також може використовувати P2P механізми для покращення пропускної здатності та надійності мережі. Кожна з цих архітектур має свої переваги та недоліки і може бути вибрана в залежності від конкретних потреб та вимог до мережі [2].

У рамках дослідження було обрано клієнт-серверну архітектуру. Ця архітектура виявилася особливо цікавою та важливою у контексті даного дослідження за наступні переваги.

Перш за все, клієнт-серверна архітектура забезпечує централізовану точку доступу до ресурсів та послуг. Це дозволяє ефективно керувати ресурсами, контролювати доступ та забезпечувати безпеку мережі. Це було особливо

корисно при аналізі мережевих ігор, де необхідно забезпечити консистентність гри та уникнути можливих перешкод.

Наступною перевагою є те, що клієнт-серверна архітектура дозволяє забезпечити централізоване керування та підтримку. Оновлення, патчі та інші зміни можуть бути легко впроваджені на сервері, і всі клієнти автоматично отримають доступ до цих оновлень при підключенні. Це спрощує процес управління грою та дозволяє забезпечити єдність гри для всіх гравців.

Крім того, централізована архітектура сприяє покращенню безпеки мережі. Доступ до ресурсів та інформації може бути ефективно контрольований та обмежений на рівні сервера, що дозволяє уникнути можливих атак та зловживань.

Враховуючи ці переваги, клієнт-серверна архітектура виявилася найкращим вибором. Її централізований підхід до управління ресурсами та забезпечення безпеки дозволяє забезпечити ефективну та надійну мережу.

1.2. Порівняння та аналіз особливостей протоколів зв'язку UDP та TCP

Протоколи зв'язку є важливим компонентом мережевої системи, який дозволяє здійснювати обмін даними між пристроями у мережі. Протокол визначає правила та формати, за якими дані передаються, а також механізми контролю та забезпечення їхньої надійної доставки. Проведемо порівняння та аналіз двох основних протоколів зв'язку – UDP (User Datagram Protocol) та TCP (Transmission Control Protocol).

Що таке протокол зв'язку? Протокол зв'язку – це набір правил та процедур, які визначають, як пристрої у мережі обмінюються даними. Він визначає формати пакетів даних, порядок їхньої передачі, а також механізми перевірки цілісності та контролю потоку [3]. Кожен протокол має свої власні особливості та області застосування, що робить їх придатними для різних типів мереж та завдань.

UDP та TCP були обрані для подальшого аналізу через їх широке використання та значний вплив на мережеве програмування. Обидва протоколи представляють різні підходи до передачі даних та мають свої унікальні особливості, які варто докладніше розглянути.

UDP (User Datagram Protocol) – це простий та швидкий протокол передачі даних з неконтрольованим з'єднанням . Основні характеристики UDP включають:

1. Неконтрольована передача: UDP не встановлює з'єднання між відправником і отримувачем перед передачею даних, що призводить до більшої швидкості порівняно з TCP.
2. Відсутність гарантії доставки: UDP не має механізмів перевірки доставки або відновлення втрачених пакетів, тому може втратити деякі дані без попередження.
3. Низька накладна витрата: оскільки UDP не має складних механізмів контролю потоку та управління з'єднаннями, він має меншу накладну витрату порівняно з TCP.

TCP – це з'єднанковий протокол, що забезпечує надійну та послідовну передачу даних. Основні характеристики TCP включають:

1. Встановлення з'єднання: TCP перед передачею даних встановлює з'єднання між відправником і отримувачем, що гарантує послідовність та надійність передачі.
2. Контроль потоку: TCP має механізми контролю потоку, які регулюють швидкість передачі даних, щоб уникнути переповнення буфера отримувача.
3. Відновлення даних: у разі втрати пакетів TCP відновлює їх шляхом повторної передачі, що забезпечує надійність доставки.

Порівняння:

1. Надійність: UDP не гарантує надійну доставку даних, тоді як TCP забезпечує надійність за рахунок механізмів перевірки доставки та відновлення даних.
2. Швидкість: UDP є швидшим за TCP, оскільки не має накладних витрат, пов'язаних з встановленням з'єднання та контролем потоку.
3. Використання: UDP часто використовується в додатках, де важлива швидкість передачі даних, але втрата деяких даних не критична, наприклад, в потоковому відео. З іншого боку, TCP використовується в додатках, де критична надійність та послідовність передачі файлів.

1.3. Огляд існуючих мережевих рішень для багатокористувацьких ігор

Сучасні багатокористувацькі ігри вимагають ефективних і надійних мережевих рішень, щоб забезпечити стабільний ігровий процес і якісний досвід для гравців. Різні інструменти та платформи надають різноманітні можливості для розробників ігор, дозволяючи їм обирати найбільш підходящі рішення залежно від специфіки проєкту. Розглянемо деякі з найбільш популярних і широко використовуваних мережевих рішень для багатокористувацьких ігор.

1.3.1. Photon Unity Network (PUN)

Photon від Exit Games є однією з найпопулярніших мережевих платформ для Unity [4]. Вона надає широкий набір інструментів та сервісів для створення багатокористувацьких ігор різного типу. Photon дозволяє створювати різноманітні ігрові механіки, такі як синхронізація гравців, обмін даними, мережеві ефекти та інші. Однією з основних переваг Photon є швидкість розгортання та простота використання, що робить його популярним вибором серед розробників. Приклади використання:

1. VRChat – соціальна платформа, де користувачі можуть створювати власні аватари та світи. VRChat використовує Photon для забезпечення багатокористувацької взаємодії в реальному часі.

2. Among Us – популярна багатокористувацька гра, яка використовує Photon для забезпечення стабільної мережевої гри, що дозволяє гравцям приєднуватися та грати разом у режимі реального часу.

1.3.2. Mirror

Mirror – це інша мережева бібліотека для Unity, яка надає прості та потужні інструменти для створення багатокористувацьких ігор. Mirror базується на базових Unity Networking, але має вдосконалені можливості та підтримку [5]. Вона пропонує розробникам гнучкі інструменти для створення мережевих громадських ігор, а також має зручний API та детальну документацію, що робить її привабливим вибором для багатьох розробників. Приклади використання:

1. Rust – багатокористувацька гра на виживання, яка використовує Mirror для забезпечення стабільної взаємодії гравців у відкритому світі.
2. Eco – симуляційна гра, де гравці разом працюють над збереженням екологічного балансу у віртуальному світі. Mirror допомагає забезпечити надійну мережеву інфраструктуру для цього складного симуляційного середовища.

1.3.3. DarkRift

DarkRift – це ще одна потужна мережева бібліотека для Unity, яка відома своєю високою продуктивністю та масштабованістю [6]. Вона надає розробникам можливість створювати складні багатокористувацькі ігри з низькою затримкою та високою надійністю. Переваги:

1. Висока продуктивність: підходить для ігор з великим числом гравців.
2. Гнучкість: підтримує як UDP, так і TCP, що дозволяє оптимізувати мережеву взаємодію під конкретні вимоги гри.
3. Підтримка кластеризації: дозволяє розподіляти навантаження між декількома серверами для підвищення стабільності та масштабованості.

Найпоширенішим прикладом використання є MMORPG-проєкти. DarkRift часто використовується для створення масових багатокористувацьких онлайн-

ігор (MMORPG) завдяки своїй здатності обробляти велику кількість одночасних підключень.

1.3.4. UNet (Unity Multiplayer)

UNet – це попередній мережевий фреймворк Unity, який все ще використовується деякими розробниками, незважаючи на те, що він офіційно припинив свою підтримку [7]. UNet надає базові інструменти для створення мережевих ігор, але має деякі обмеження в порівнянні з більш сучасними рішеннями, такими як Photon або Mirror.

Переваги:

1. Інтеграція з Unity: проста у використанні для розробників, які вже знайомі з Unity.
2. Документація та приклади: широка база знань та прикладів використання.

Недоліки:

1. Обмежена масштабованість: не підходить для ігор з великою кількістю гравців.
2. Застаріла технологія: багато нових функцій та оптимізацій доступні лише на нових платформах.

1.3.5. Forge Networking

Forge Networking – це ще одна альтернатива для створення багатокористувацьких ігор у Unity [8]. Вона забезпечує гнучкість та простоту у використанні, дозволяючи розробникам швидко налаштовувати та керувати мережевими аспектами своїх ігор.

Переваги:

1. Гнучкий API: легкий у використанні для розробників різного рівня.
2. Підтримка P2P: дозволяє створювати ігри з peer-to-peer архітектурою.

Головним недоліком є менша популярність, з чого випливає і менша спільнота та обмежена підтримка порівняно з Photon або Mirror.

1.4. Роль Arduino в ігрових контролерах

Arduino є відкритою платформою для розробки електронних пристроїв, яка базується на простому мікроконтролері та інтуїтивно зрозумілому середовищі програмування. Ця платформа забезпечує широкий набір інструментів для розробників, дозволяючи створювати різноманітні інтерактивні пристрої, включаючи ігрові контролери [9].

1.4.1. Основні характеристики Arduino

Апаратура

1. Мікроконтролери: Arduino використовує різні мікроконтролери, такі як ATmega328 (Arduino Uno), ATmega2560 (Arduino Mega), які забезпечують достатню обчислювальну потужність для більшості завдань.
2. Розширюваність: плати Arduino мають численні порти для підключення сенсорів, кнопок, дисплеїв та інших периферійних пристроїв, що робить їх ідеальними для створення кастомних ігрових контролерів.

Програмне забезпечення

1. IDE: Arduino IDE (інтегроване середовище розробки) є простим та інтуїтивно зрозумілим інструментом для написання коду та завантаження його на плату.
2. Бібліотеки: велика кількість бібліотек дозволяє легко інтегрувати різні компоненти і функції, такі як обробка введення з джойстика, керування світлодіодами, робота з дисплеями тощо.

1.4.2. Переваги використання Arduino в ігрових контролерах

1. Доступність та простота використання. Arduino забезпечує простий і доступний спосіб створення ігрових контролерів навіть для новачків. Інтуїтивно зрозуміле середовище програмування та наявність великої кількості навчальних матеріалів роблять Arduino популярним вибором серед аматорів та професіоналів.

2. Гнучкість та кастомізація. Завдяки відкритій архітектурі та підтримці великої кількості додаткових модулів і сенсорів, розробники можуть створювати унікальні та інноваційні ігрові контролери, які відповідають їхнім специфічним потребам. Наприклад, можна створити контролери з датчиками руху, гіроскопами, акселерометрами, а також додати зворотній зв’язок у вигляді вібрації або звукових сигналів.
3. Інтеграція з різними платформами. Arduino можна легко інтегрувати з різними ігровими платформами, включаючи ПК, консолі та мобільні пристрої. Це робить Arduino універсальним інструментом для розробки ігрових пристроїв, які можуть бути сумісними з широким спектром ігрових платформ та програмного забезпечення.
4. Спільнота та підтримка. Велика спільнота користувачів Arduino забезпечує широкий спектр підтримки та обміну знаннями. На форумах, блогах та відеоканалах можна знайти багато прикладів проєктів, які можна використовувати як основу для власних розробок.

1.4.3. Приклади використання Arduino в ігрових контролерах

1. Класичні геймпади. Багато ентузіастів використовують Arduino для створення кастомних геймпадів, які можна підключити до ПК або консолей. Такі геймпади можуть включати додаткові функції, такі як макроси, програмовані кнопки або спеціальні світлові ефекти.
2. VR-контролери. Arduino також використовується для створення контролерів для віртуальної реальності (VR), які можуть включати датчики руху та положення, що дозволяє забезпечити більш інтерактивний ігровий досвід.
3. Контролери для спеціалізованих ігор. Існують приклади використання Arduino для створення контролерів для специфічних ігор або жанрів. Наприклад, кермо для гоночних симуляторів з реалістичним зворотним зв’язком, або контролери для музичних ігор, які дозволяють грати на віртуальних інструментах.

Arduino є потужним інструментом для створення ігрових контролерів завдяки своїй доступності, гнучкості та широким можливостям кастомізації. Відкрита архітектура і велика спільнота підтримки роблять цю платформу ідеальним вибором для розробників, які прагнуть створювати унікальні та інноваційні ігрові пристрої. Таким чином, роль Arduino в ігрових контролерах полягає у наданні розробникам широких можливостей для творчості та інновацій, що дозволяє створювати унікальні і персоналізовані ігрові пристрої.

РОЗДІЛ 2
РОЗРОБКА МЕРЕЖЕВОЇ ГРИ З АПАРАТНИМ КОНТРОЛЕРОМ

2.1. Формулювання проблеми та методи дослідження

Проблема: розробка мережевої гри з апаратним контролером вимагає збалансованого поєднання високоякісного геймплею та ефективної роботи апаратного обладнання.

Методи дослідження:

1. Аналіз існуючих рішень: ретельний аналіз попередніх робіт у цій області, таких як ігрові контролери та мережеві ігри, щоб ідентифікувати найкращі практики та можливі обмеження.
2. Моделювання та симуляція: використання спеціального програмного забезпечення для моделювання та симуляції різних аспектів гри та взаємодії з контролером для виявлення можливих проблем або необхідних вдосконалень.
3. Експериментальне тестування з використанням користувацької аналітики: залучення активних гравців для тестування прототипів гри та контролера з метою збору відгуків, аналізу їхньої поведінки та уточнення вимог до проєкту.

2.2. Вимоги до обладнання та вибір Arduino моделі

Для успішної реалізації проєкту з розробки мережевого рішення для ігрових додатків в середовищі Unity з інтеграцією унікального контролера на базі Arduino необхідно врахувати вимоги до обладнання та обґрунтувати вибір відповідної моделі Arduino. В даному проєкті використовується Arduino Nano та датчик гіроскоп MPU6050. Необхідне обладнання:

Апаратний контролер Arduino Nano:

1. Обчислювальна потужність: Arduino Nano базується на мікроконтролері ATmega328P, який має достатню потужність для обробки сигналів від датчиків та керування ігровими механіками.

2. Кількість вхідно-вихідних портів: Nano має 14 цифрових вхідно-вихідних пінів (з яких 6 можуть використовуватися як ШІМ-канали), 8 аналогових входів, та один UART порт. Це дозволяє підключати достатню кількість периферійних пристроїв для більшості проєктів [10].
3. Розмір: компактний розмір Arduino Nano дозволяє легко інтегрувати його в різноманітні ігрові контролери, не займаючи багато місця.

Модуль MPU6050.

1. Гіроскоп та акселерометр: MPU6050 є інтегрованим датчиком, який включає в себе 3-осьовий гіроскоп та 3-осьовий акселерометр . Це дозволяє точно відстежувати рухи контролера в трьох вимірах.
2. Комунікація: датчик використовує шину I2C для зв'язку з Arduino, що дозволяє легко інтегрувати його в проєкт.

Обґрунтування вибору моделі Arduino Nano:

1. Компактність та гнучкість: Arduino Nano є дуже компактним і може бути легко інтегрований в будь-який ігровий контролер. Це особливо важливо для пристроїв, де простір є обмеженим.
2. Достатня потужність для більшості завдань: незважаючи на свій розмір, Arduino Nano має достатню обчислювальну потужність для обробки даних з датчиків та керування ігровими механіками. Це робить його відмінним вибором для більшості ігрових проєктів.
3. Широка підтримка та наявність бібліотек: Arduino Nano підтримується великою спільнотою розробників, що забезпечує наявність численних бібліотек та прикладів коду. Це значно спрощує процес розробки та інтеграції.
4. Економічна доцільність: Arduino Nano є економічно доступним рішенням, що дозволяє знизити вартість проєкту без втрати функціональності.

2.3. Вибір та інтеграція датчиків

Для успішної реалізації проєкту з використанням Arduino Nano та датчика гіроскоп MPU6050 необхідно детально розглянути процес підключення датчиків та модулів до апаратного контролера. Це забезпечить коректне зчитування даних та їх обробку для подальшої інтеграції з ігровим рушієм Unity.

Датчик гіроскоп MPU6050 є інтегрованим датчиком, який включає в себе 3-осьовий гіроскоп та 3-осьовий акселерометр [11]. Він дозволяє відстежувати обертання та прискорення по трьох осях, що є ключовим для створення інтерактивних ігрових контролерів.

З'єднання за допомогою I2C:

1. Підключення: MPU6050 використовує шину I2C для зв'язку з Arduino Nano. Це дозволяє легко підключати датчик до контролера, використовуючи всього два пін (SDA та SCL).
2. SDA (Data Line): підключається до аналогового піну A4 на Arduino Nano.
3. SCL (Clock Line): підключається до аналогового піну A5 на Arduino Nano.
4. Живлення: датчик потребує живлення 3.3V або 5V, що може бути забезпечено з відповідного виводу на Arduino Nano.

Програмна підтримка:

1. Бібліотеки: для роботи з MPU6050 можна використовувати готові бібліотеки, такі як "MPU6050" або "I2Cdevlib", які значно спрощують процес зчитування та обробки даних з датчика.
2. Ініціалізація: бібліотеки забезпечують простий спосіб ініціалізації та налаштування датчика для зчитування даних про обертання та прискорення.
3. Калібрування та обробка даних:
4. Калібрування: важливим етапом є калібрування датчика для забезпечення точної та стабільної роботи. Це включає усереднення показників, фільтрацію шумів та компенсацію зсувів.

5. Обробка даних: зібрані дані можуть бути оброблені безпосередньо на Arduino або передані на комп'ютер для подальшої обробки та візуалізації в середовищі Unity.

Правильне підключення датчиків та модулів до апаратного контролера є ключовим для забезпечення коректного зчитування та обробки даних. Використання Arduino Nano та датчика гіроскоп MPU6050 у поєднанні з ігровим рушієм Unity дозволяє створити інтерактивні ігрові контролери з високою точністю відстеження рухів. Використання готових бібліотек та простих алгоритмів обробки даних значно спрощує процес розробки та інтеграції таких контролерів у ігрові додатки

2.4. Засоби для розробки мережевого рішення

Розробка мережевих рішень для ігрових додатків передбачає використання різних інструментів та технологій для забезпечення надійного і ефективного обміну даними між клієнтами та сервером. Одним з основних інструментів для цього є сокети, які забезпечують низькорівневу комунікацію через мережу.

Сокети - це інтерфейси для обміну даними між програмами, що працюють на різних комп'ютерах у мережі. Вони підтримують різні протоколи зв'язку, такі як TCP (Transmission Control Protocol) та UDP (User Datagram Protocol) [12].

Основні концепції сокетів:

1. IP-адреса та порт: кожен сокет пов'язаний з IP-адресою та портом, що дозволяє ідентифікувати конкретне з'єднання в мережі.
2. Сокет API: більшість мов програмування мають власні бібліотеки для роботи з сокетами. Наприклад, у C# це *System.Net.Sockets*, у Python - *socket*, а в Java - *java.net.Socket*.
3. Протоколи:
    3.1. TCP: забезпечує надійне, орієнтоване на з'єднання з'єднання. Гарантує доставку пакетів даних у правильному порядку.

3.2. UDP: забезпечує менш надійне, але більш швидке з'єднання. Використовується для додатків, де швидкість важливіша за надійність (наприклад, для потокового відео або аудіо).

Етапи розробки мережевого рішення за допомогою сокетів:

1. Ініціалізація сокетів:
   1.1. Встановлення необхідних параметрів (IP-адреса, порт, протокол).
   1.2. Створення сокетів за допомогою відповідних бібліотек.
2. Встановлення з'єднання:
   2.1. Для TCP: Створення сервера, що прослуховує вхідні з'єднання, та клієнта, що ініціює з'єднання з сервером.
   2.2. Для UDP: Обмін даними без встановлення постійного з'єднання.
3. Передача даних:
   3.1. Для TCP: створення сервера, що прослуховує вхідні з'єднання, та клієнта, що ініціює з'єднання з сервером.
   3.2. Для UDP: обмін даними без встановлення постійного з'єднання.
4. Обробка даних:
   4.1. Використання відповідних методів для відправлення та отримання даних через сокети (наприклад, send та receive для TCP або sendto та recvfrom для UDP).
5. Закриття з'єднання:
   5.1. Перетворення отриманих даних у формат, придатний для використання в ігровому додатку.
   5.2. Обробка можливих помилок та відновлення з'єднання при необхідності.
   5.3. Закриття сокетів після завершення передачі даних.

Приклад використання сокетів в Unity

Для демонстрації використання сокетів розглянемо приклад простого клієнт-серверного застосування на основі TCP в Unity. Серверний додаток

створює сокет, прослуховує вхідні з'єднання та обробляє отримані дані. Клієнтський додаток, з іншого боку, ініціює з'єднання із сервером та відправляє дані.

Сервер:

1. Сервер створює об'єкт TcpListener, який прослуховує вказаний порт.
2. Після прийому з'єднання сервер зчитує дані з потоку і виводить їх.

Клієнт в Unity:

1. Клієнт створює об'єкт TcpClient і з'єднується із сервером.
2. Після з'єднання клієнт відправляє дані на сервер і закриває з'єднання.

Використання UDP в Unity.

Для додатків, де важлива швидкість передачі даних, можна використовувати протокол UDP. Серверний додаток прослуховує порт і приймає вхідні пакети даних, а клієнт відправляє дані на сервер без встановлення з'єднання.

Сервер:

1. Сервер створює об'єкт UdpClient, який прослуховує вказаний порт.
2. Сервер приймає пакети даних і виводить їх.

Клієнт в Unity:

1. Клієнт створює об'єкт UdpClient і відправляє дані на вказану IP-адресу і порт.

Використання сокетів для розробки мережевого рішення дозволяє забезпечити надійну та ефективну комунікацію між клієнтами та сервером в ігрових додатках. Вибір протоколу (TCP або UDP) залежить від конкретних вимог до швидкості та надійності передачі даних. Завдяки підтримці сокетів у різних мовах програмування та середовищах розробки, їх інтеграція в ігрові додатки є відносно простою та ефективною.

2.5. Вимоги до мережевого рішення

Розробка мережевих рішень для ігрових додатків вимагає врахування ряду технічних та функціональних вимог. Ці вимоги забезпечують стабільну, ефективну та масштабовану взаємодію між клієнтами та сервером, а також сприяють високій якості ігрового досвіду для користувачів.

Технічні вимоги:

1. Продуктивність:
   1.1. Низька затримка (Latency): важливо забезпечити мінімальні затримки при передачі даних між клієнтом і сервером. Це особливо критично для ігор реального часу, де затримки можуть впливати на ігровий процес.
   1.2. Висока пропускна здатність (Throughput): мережеве рішення повинно підтримувати високу пропускну здатність для обробки великого обсягу даних, що передаються між клієнтами та сервером.
2. Надійність та стійкість:
   2.1. Відмовостійкість (Fault Tolerance): система повинна бути стійкою до відмов компонентів, з можливістю автоматичного відновлення та мінімальними втратами даних.
   2.2. Стійкість до помилок (Error Handling): важливо забезпечити ефективну обробку помилок, таких як втрати пакетів або збої в з'єднанні.
3. Масштабованість:
   3.1. Підтримка великої кількості користувачів (Scalability): мережеве рішення повинно мати можливість масштабування для обробки великої кількості одночасних користувачів без втрати продуктивності.
   3.2. Гнучка архітектура: мережеве рішення повинно мати модульну архітектуру, яка дозволяє легко додавати нові функції та розширювати існуючі.
4. Безпека:
   4.1. Захист даних (Data Security): забезпечення шифрування даних, що передаються між клієнтами та сервером, для захисту від несанкціонованого доступу.

4.2. Аутентифікація та авторизація (Authentication and Authorization): впровадження механізмів аутентифікації та авторизації для захисту доступу до гри та її ресурсів.

Функціональні вимоги:

1. Сумісність:

1.1. Підтримка різних платформ: мережеве рішення повинно бути сумісним з різними операційними системами та платформами, такими як Windows, macOS, Linux, iOS, та Android.

1.2. Інтеграція з ігровим рушієм Unity: забезпечення легкої інтеграції з Unity для реалізації мультиплеєрного ігрового процесу.

2. Гнучкість налаштувань:

2.1. Налаштування параметрів з'єднання: можливість налаштування параметрів з'єднання, таких як IP-адреси, порти, протоколи (TCP/UDP), та інші.

2.2. Підтримка кастомних сценаріїв: можливість створення та використання кастомних сценаріїв для різних типів ігрових взаємодій.

3. Логування та моніторинг:

3.1. Система логування: впровадження системи логування для відстеження подій та помилок у мережевій взаємодії.

3.2. Моніторинг продуктивності: забезпечення інструментів для моніторингу продуктивності мережевого рішення в реальному часі.

4. Зворотний зв'язок для користувачів:

4.1. Системи сповіщень: впровадження систем сповіщень для інформування користувачів про статус з'єднання, помилки або інші важливі події.

4.2. Інтерфейс користувача для налаштувань: наявність зручного інтерфейсу для налаштувань мережевих параметрів користувачем.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1. Розробка ігрової частини гри

У сучасній галузі розробки ігор, використання передових технологій та інноваційних концепцій, геймплей відіграє важливу роль у створенні захоплюючого ігрового досвіду для гравців. Актуальною та перспективою темою є використання дронів у віртуальних іграх. Далі ми розглянемо процес розробки ігрової частини онлайн гри, яка базується на використанні дронів як основного засобу переміщення для гравців.

3.1.1. Ігровий UI

Ігровий UI складається з декількох основних елементів, які надають гравцю необхідну інформацію під час гри. При вході гравець бачить головне меню (рис. 3.1), де зображено:

1. Нікнейм гравця: в цьому полі гравець вводить свій ігровий нікнейм. Це ім'я буде використане для ідентифікації гравця у таблиці лідерів.
2. Кнопка «Play»: приєднує гравця до ігрового сервера.
3. Кнопка «Quit»: закриває гру.

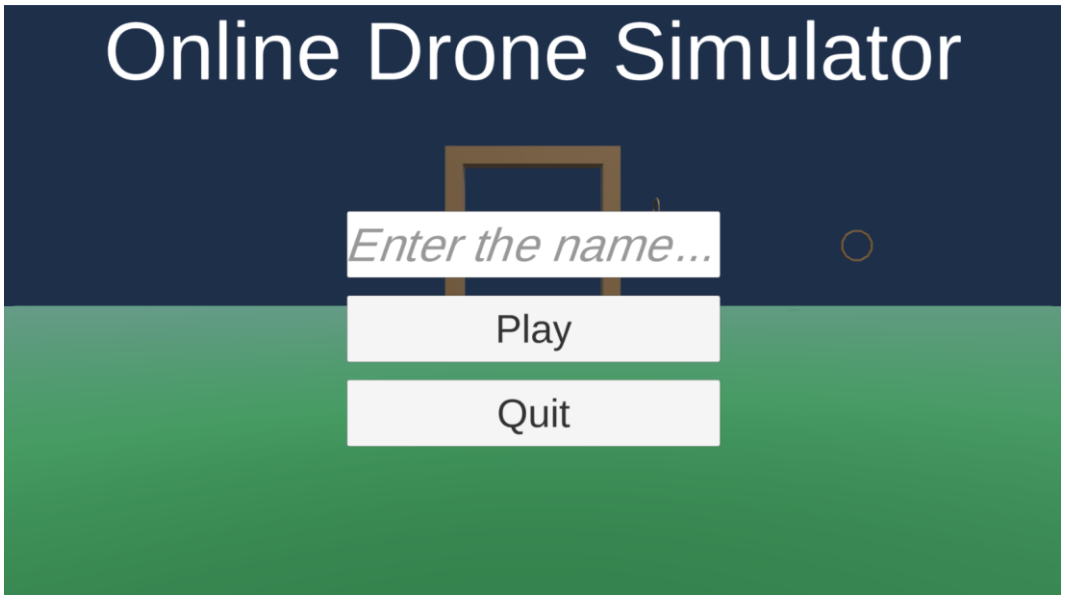

Рис 3.1 Зображення головного меню

Після успішного з’єднання з сервером, гравець бачить свій дрон, та ігровий HUD (Head-Up Display) (рис. 3.2), де зображено:

1. Таблицю лідерів: список з гравців, які пройшли весь трек, та час за який вони це зробили. Під таблицею знаходиться ім’я гравця, та його найкращий час.
2. Таймер: час який постійно збільшується поки гравець проходить трек.
3. Кнопка паузи: відкриває паузу, з якої гравець може вийти з гри.

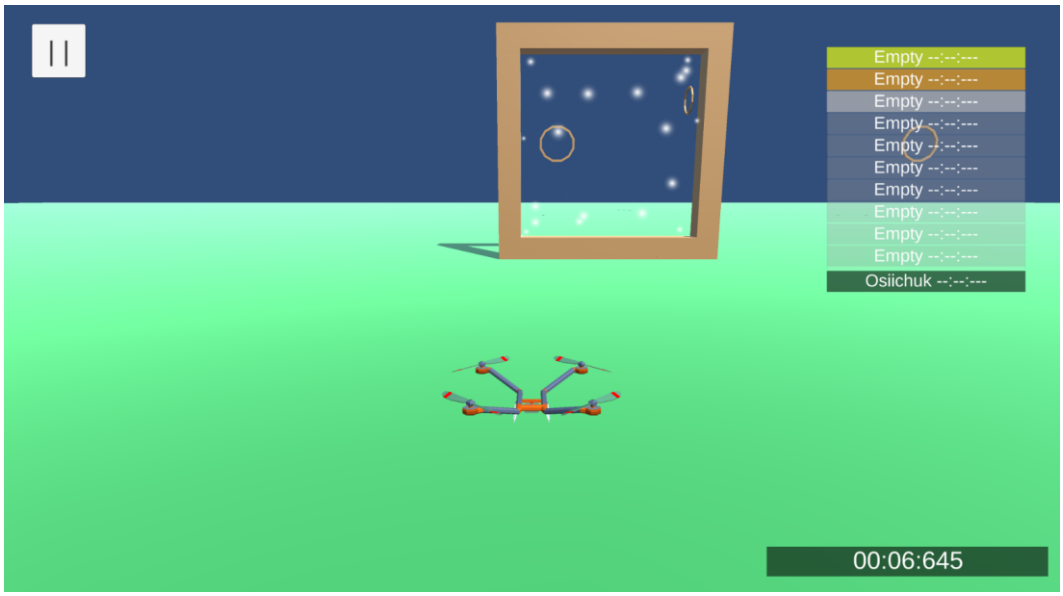

Рис 3.2 Зображення вікна гри, після приєднання до сервера

Ігровий UI відіграє важливу роль у забезпеченні позитивного ігрового досвіду для гравців. Він не лише надає необхідну інформацію для гравця, але й допомагає стимулювати його до досягнення кращих результатів та підтримує конкурентну атмосферу гри.

3.1.2. Ігрове Оточення

Розглянемо ігрове оточення гри, яке включає в себе використання LowPoly 3D моделей дерев та різних каменів (рис 3.3) [13]. Метою цих елементів гри є створення іммерсивного і атмосферного середовища, яке допоможе гравцям відчути себе частиною ігрового світу та поглибити їх імагінацію під час гри. Використання LowPoly моделей дозволяє нам досягти балансу між візуальною привабливістю та продуктивністю гри.

Рис 3.3 Зображення ігрового треку та зовнішнього оточення

Важливість ігрового оточення полягає у створенні атмосферного та іммерсивного середовища, яке допомагає гравцям відчути себе частиною ігрового світу. Правильно підібрані та реалістично відтворені моделі дозволяють нам досягти цієї мети, надаючи гравцям приємне та захоплююче сприйняття гри.

3.1.3. Контролер дрона

Опис роботи та руху дрона: контролер дрона забезпечує його стабільний політ та керування шляхом застосування фізичних сил та моментів, що дозволяють коригувати положення та орієнтацію дрона у просторі. Основні функції контролера включають корекцію кута нахилу, підтримку спрямованості вперед, а також управління підйомною силою та швидкістю обертання моторів.

Корекція кута нахилу: кут нахилу дрона коригується за допомогою функції *UpCorrect()*, яка застосовує відповідний момент сили для вирівнювання вектора "вгору" дрона з бажаним напрямком.

```
void UpCorrect() {
   Vector3 upVect = Vector3.up;
   if (yawObj != null && isCamLock){
upVect += Flat(yawObj.transform.right).normalized *
      playerMovement.inputDir.x * upCorrectInput;
upVect += Flat(yawObj.transform.forward).normalized *
      playerMovement.inputDir.y * upCorrectInput;
```

```
        } else {
            upVect += Vector3.right * playerMovement.inputDir.x;
            upVect += Vector3.forward * playerMovement.inputDir.y;
        }
        Vector3 cross = Vector3.Cross(transform.up, upVect);
        float ang = Vector3.Angle(transform.up, upVect);
        if (ang < -upCorrectMax) ang = -upCorrectMax;
        if (ang > upCorrectMax) ang = upCorrectMax;
        myBody.AddTorque(cross.normalized * ang * upCorrectMul,
          ForceMode.Acceleration);
    }
```

Функція *UpCorrect()* розраховує кутову різницю між поточним вектором "вгору" дрона та бажаним напрямком (з урахуванням введення користувача), і застосовує відповідний момент сили для корекції положення.

Підтримка руху вперед

Для підтримки руху дрона вперед використовується функція *YawCorrect()*. Ця функція забезпечує, щоб передній вектор дрона завжди залишався вирівняним з напрямком, заданим об'єктом *yawObj*.

```
    void YawCorrect() {
        if (yawObj == null) return;
        Vector3 objFw = Flat(yawObj.transform.forward);
        Vector3 selfFw = Flat(transform.forward);
        Vector3 cross = Vector3.Cross(selfFw, objFw);
        float ang = Vector3.Angle(selfFw, objFw);
        if (ang < -yawOfsMax) ang = -yawOfsMax;
        if (ang > yawOfsMax) ang = yawOfsMax;
    myBody.AddTorque(cross.normalized * ang * yawOfsMul,
ForceMode.Acceleration);
    }
```

Функція *YawCorrect()* розраховує різницю між поточним напрямком вперед дрона та бажаним напрямком, і застосовує момент сили для корекції орієнтації.

Управління підйомною силою та швидкістю обертання моторів

Підйомна сила та швидкість обертання моторів керуються функцією *UpdateThrust()*. Ця функція розраховує необхідну підйомну силу та швидкість обертання на основі введення користувача та застосовує їх до дрона.

```
void UpdateThrust() {
    float upThrust = 0;
    if (playerMovement.inputs[1]) upThrust -= 1;
    if (playerMovement.inputs[0]) upThrust += 1;
    if (upThrust > 1) upThrust = 1;
    if (upThrust < -1) upThrust = -1;
    float spin = spinBase;
    spin += upThrust * (upThrust < 0 ?  spinBase : spinFast);
    float delta = spinDelta * Time.deltaTime;
    foreach (Spinner spinner in mySpinner)
    {
        if (spinner.speed < spin - delta)
            spinner.speed += delta;
        else if (spinner.speed > spin + delta)
            spinner.speed -= delta;
        else
            spinner.speed = spin;
    }
    float thrust = thrustBase + upThrust * thrustOffset;
    if (upThrust < 0)
        thrust = thrustBase * (1.0f + upThrust);
    myBody.AddForce(transform.up * thrust, ForceMode.Acceleration);
}
```

Функція *UpdateThrust()* керує підйомною силою дрона шляхом зміни обертів моторів, що дозволяє дрону підніматися або опускатися відповідно до введення користувача. Швидкість обертання моторів також коригується для забезпечення стабільного польоту.

3.2. Проєктування мережевого плагіну для Unity

Мережеві плагіни відіграють важливу роль у забезпеченні комунікації між різними програмними компонентами в сучасних розподілених системах. Вони дозволяють додаткам взаємодіяти через мережу, використовуючи стандартизовані протоколи передачі даних.

Мета цього проєкту полягає в розробці мережевого плагіну, здатного забезпечувати надійну та ефективну передачу даних між клієнтськими та серверними додатками. Особливу увагу було приділено аспектам реалізації, які

включають ініціалізацію сокетів, управління з'єднаннями, передачу даних та обробку помилок.

3.2.1. Основи клієнт-серверної взаємодії

Клієнт-серверна архітектура є основою для багатьох мережевих додатків. У цій моделі клієнти звертаються до сервера для отримання ресурсів або виконання певних завдань. Сервер обробляє ці запити та повертає відповідні результати. Така модель забезпечує централізоване управління та розподіл ресурсів, що дозволяє ефективно масштабувати систему.

Переваги клієнт-серверної архітектури включають централізацію контролю, що спрощує управління системою та забезпечення безпеки, а також можливість легкого масштабування шляхом додавання нових клієнтів або серверів. Недоліки включають можливі проблеми з продуктивністю у випадку перевантаження сервера та залежність клієнтів від доступності сервера.

У контексті цього проєкту, клієнт-серверна архітектура забезпечить чіткий розподіл обов'язків між клієнтським додатком, який генерує запити, та серверним додатком, який обробляє ці запити. Це дозволить створити гнучку та масштабовану систему для передачі даних.

3.2.2. Протоколи TCP та UDP

TCP (Transmission Control Protocol) та UDP (User Datagram Protocol) є двома основними протоколами транспортного рівня, що використовуються для передачі даних у мережах. Кожен з них має свої унікальні особливості та сфери застосування.

TCP є надійним протоколом, що забезпечує гарантовану доставку даних. Він використовує механізми встановлення з'єднання, контролю помилок та управління потоком, що робить його ідеальним для додатків, де важлива точність передачі даних, наприклад, для веб-сайтів, електронної пошти та передачі файлів. TCP реалізований у нашому проєкті за допомогою класів *TcpClient* та *TcpListener*.

UDP, навпаки, є ненадійним протоколом, що не гарантує доставку даних, але забезпечує високу швидкість передачі. Він підходить для додатків, де важлива швидкість, а не точність, таких як потокове передавання відео, голосові дзвінки та онлайн-ігри. Реалізація UDP у нашому проєкті здійснюється за допомогою класу *UdpClient*.

3.2.3. Реалізація сокетів

Сокети є основним інструментом для реалізації мережевих додатків. Вони забезпечують інтерфейс для відправлення та отримання даних через мережу. У цьому розділі ми розглянемо процес ініціалізації, налаштування та прив'язки сокетів для протоколів TCP та UDP.

Ініціалізація сокетів

Ініціалізація сокетів включає створення об'єктів сокетів та їх налаштування. Для TCP-сокетів ми використовуємо наступний код:

```
TcpClient client = new TcpClient();
client.Connect(ipAddress, port);
```

Для UDP-сокетів процес ініціалізації виглядає наступним чином:

```
UdpClient udpClient = new UdpClient();
udpClient.Connect(ipAddress, port);
```

Ці методи створюють сокети, які можуть використовуватися для відправлення та отримання даних.

Прив'язка сокетів

Прив'язка сокетів до конкретних IP-адрес та портів є важливим кроком у налаштуванні мережевого з'єднання. Для TCP це виглядає так:

```
TcpListener listener = new TcpListener(ipAddress, port);
listener.Start();
```

Для UDP-прив'язки використовується наступний код:

```
udpClient.Client.Bind(new IPEndPoint(ipAddress, port));
```

Прив'язка дозволяє сокетам приймати вхідні з'єднання або дані з певних адрес та портів.

Управління з'єднанням

Для TCP управління з'єднанням включає встановлення з'єднання, його підтримку та завершення. Наприклад, встановлення з'єднання:

```
TcpClient client = new TcpClient();
client.Connect(ipAddress, port);
```

Завершення з'єднання здійснюється за допомогою методу *Close()*. UDP-сокети не вимагають встановлення з'єднання, оскільки вони працюють за принципом безз'єдної передачі даних.

3.2.4. Передача даних

Передача даних є ключовою функцією мережевого плагіну. Розглянемо процеси відправлення та прийому даних для протоколів TCP та UDP, а також обробку помилок.

Передача даних через TCP

TCP забезпечує надійну передачу даних з контролем помилок. Для відправлення даних використовується метод *Write*:

```
NetworkStream stream = client.GetStream();
byte[] data = Encoding.UTF8.GetBytes("Hello, Server!");
stream.Write(data, 0, data.Length);
```

Прийом даних здійснюється за допомогою методу *Read*:

```
byte[] receivedData = new byte[256];
int bytes = stream.Read(receivedData, 0, receivedData.Length);
string responseData = Encoding.UTF8.GetString(receivedData, 0,
bytes);
```

Такі методи забезпечують надійну та послідовну передачу даних між клієнтом та сервером.

Передача даних через UDP

UDP забезпечує швидку передачу даних без гарантії доставки. Для відправлення даних використовується метод *Send*:

```
byte[] data = Encoding.UTF8.GetBytes("Hello, Server!");
udpClient.Send(data, data.Length);
```

Прийом даних здійснюється за допомогою методу *Receive*:

```
IPEndPoint remoteEndPoint = new IPEndPoint(IPAddress.Any, 0);
byte[] receivedData = udpClient.Receive(ref remoteEndPoint);
string responseData = Encoding.UTF8.GetString(receivedData);
```

Хоча UDP не гарантує доставку даних, він забезпечує високу швидкість передачі, що важливо для реальних додатків.

Обробка помилок

Обробка помилок є важливою частиною будь-якого мережевого додатку. Основні помилки, з якими можна зіткнутися, включають помилки з'єднання, тайм-аути та перевантаження. Для обробки помилок використовуються блоки *try-catch*:

```
try {
    client.Connect(ipAddress, port);
} catch (SocketException ex) {
    Console.WriteLine($"SocketException: {ex.Message}");
}
```

Такі методи дозволяють ефективно обробляти помилки та забезпечувати стабільну роботу додатку.

3.2.5. Реалізація клієнтської частини

Клас *Client* відповідає за реалізацію клієнтської логіки, включаючи встановлення з'єднання, обробку повідомлень та подій, пов'язаних із мережею.

Опис класу Client

Клас *Client* є похідним від базового класу *Peer* і надає функціонал для взаємодії з сервером. Основні можливості класу включають встановлення та

підтримку з'єднання, обробку повідомлень, а також управління подіями, такими як підключення та відключення клієнтів.

Конструктор та ініціалізація

Клас *Client* має два конструктори, що забезпечують початкову настройку:

1. *Client(IClient transport, string logName = "CLIENT")* - використовується для ініціалізації з визначеним транспортним протоколом для обміну даними.
2. *Client(string logName = "CLIENT")* - використовує вбудований транспортний протокол UDP для обміну даними.

Встановлення з'єднання

Метод *Connect* відповідає за спробу встановлення з'єднання з сервером за вказаною адресою. Він приймає наступні параметри:

1. *hostAddress* - адреса хоста для підключення.
2. *maxConnectionAttempts* - максимальна кількість спроб підключення.
3. *messageHandlerGroupId* - ідентифікатор групи методів обробки повідомлень.
4. *message* - дані, що відправляються серверу при спробі підключення.
5. *useMessageHandlers* - визначає, чи використовувати вбудовану систему обробки повідомлень.
6. Метод повертає *true*, якщо спроба підключення буде здійснена, і *false* у разі помилки.

Події класу Client

Клас *Client* визначає кілька подій, які викликаються у різних ситуаціях:

1. *Connected* - викликається при успішному встановленні з'єднання.
2. *ConnectionFailed* - викликається при невдалій спробі встановлення з'єднання.
3. *MessageReceived* - викликається при отриманні повідомлення.
4. *Disconnected* - викликається при відключенні від сервера.
5. *ClientConnected* - викликається при підключенні іншого клієнта.
6. *ClientDisconnected* - викликається при відключенні іншого клієнта.

Обробка повідомлень

Клас *Client* використовує делегати та словники для обробки отриманих повідомлень. Метод *CreateMessageHandlersDictionary* створює словник методів обробки повідомлень на основі ідентифікатора групи методів обробки повідомлень.

Метод *Handle* визначає логіку обробки отриманих повідомлень залежно від їх заголовка. Повідомлення можуть бути як користувацькими, так і внутрішніми.

Управління з'єднанням

Методи *Disconnect* та *LocalDisconnect* забезпечують відключення клієнта від сервера з відповідною обробкою причин відключення. Метод *Heartbeat* відповідає за підтримку активного з'єднання, надсилаючи "серцебиття" серверу.

3.2.6. Реалізація серверної частини

Клас *Server* відповідає за обробку підключень клієнтів, обмін повідомленнями та управління з'єднаннями в мережевій архітектурі. Він успадковує функціональність від класу *Peer* та використовує механізм подій для оповіщення про різні стани та події, такі як підключення, відключення клієнтів і отримання повідомлень.

Основні властивості та методи

Події:

1. *ClientConnected*: виникає при підключенні клієнта до сервера.
2. *ConnectionFailed*: виникає при невдалій спробі встановлення з'єднання.
3. *MessageReceived*: виникає при отриманні повідомлення.
4. *ClientDisconnected*: виникає при відключенні клієнта від сервера.

Властивості:

1. *IsRunning*: вказує, чи працює сервер.
2. *Port*: повертає локальний порт, на якому працює сервер.
3. *TimeoutTime*: встановлює час очікування для майбутніх з'єднань і оновлює час очікування всіх підключених клієнтів.

4. *MaxClientCount*: максимальна кількість одночасних підключень.
5. *ClientCount*: кількість підключених клієнтів.
6. *Clients*: масив усіх підключених клієнтів.

Методи:

1. *Start(ushort port, ushort maxClientCount, byte messageHandlerGroupId = 0, bool useMessageHandlers = true)*: запускає сервер на вказаному порту з заданою максимальною кількістю клієнтів. Може використовувати групи обробників повідомлень.
2. *Stop()*: зупиняє сервер.
3. *Send(Message message, ushort toClient, bool shouldRelease = true)*: відправляє повідомлення вказаному клієнту.
4. *SendToAll(Message message, bool shouldRelease = true)*: відправляє повідомлення всім підключеним клієнтам.
5. *DisconnectClient(ushort id, Message message = null)*: відключає клієнта за його *ID*.

Внутрішні механізми

Підключення клієнтів: метод *HandleConnectionAttempt* обробляє спроби підключення клієнтів. При успішному підключенні метод *AcceptConnection* додає клієнта до словника *clients* та відправляє повідомлення про вітання. Якщо підключення не вдалося, метод *Reject* відхиляє з'єднання з відповідним повідомленням.

Обробка повідомлень: метод *Handle* обробляє різні типи повідомлень, включаючи користувацькі та внутрішні повідомлення сервера. Повідомлення можуть бути надіслані всім клієнтам, або ж конкретному клієнту.

Відключення клієнтів: метод *LocalDisconnect* забезпечує локальне відключення клієнта, видаляючи його зі списку підключених клієнтів і вивільняючи його ID для повторного використання.

Події та їх обробка: методи *OnClientConnected*, *OnConnectionFailed*, *OnMessageReceived* та *OnClientDisconnected* забезпечують обробку відповідних подій, викликаючи зареєстровані обробники подій.

3.3. Інтеграція мережевого рішення

Інтеграція мережевого рішення є критичним етапом у розробці багатокористувацьких ігор, оскільки забезпечує зв'язок між клієнтськими додатками та сервером. Це дозволяє гравцям взаємодіяти один з одним у реальному часі, синхронізуючи їхні дії та положення у віртуальному середовищі. В даному розділі буде розглянуто інтеграцію мережевого рішення в Unity для гри про дрони, зокрема процес реалізації на стороні клієнта. Гра потребує надсилання інформації про кути нахилу дрона та стан кнопок прискорення на сервер, а також отримання від сервера даних про поточне положення та нахил гравця. Цей процес включає налаштування мережевого менеджера, обробку підключення/відключення, синхронізацію даних та інтерполяцію положення дронів.

3.3.1. Реалізація на стороні клієнта

На стороні клієнта важливою складовою є обробка введень користувача та передача їх на сервер. Клієнт також отримує дані від сервера для оновлення стану гри, зокрема, позицій дронів, і здійснює інтерполяцію для плавного відображення рухів. Перелік основних класів та методів:

Клас *NetworkManager* – відповідає за встановлення та підтримку з'єднання з сервером.

1. Метод *Connect*: встановлює з'єднання з сервером, використовуючи IP-адресу та порт, зазначені в конфігурації. При успішному з'єднанні викликається метод DidConnect, який сигналізує про успішне підключення.
2. Метод *FixedUpdate*: виконує оновлення стану клієнта та інкрементує серверний такт, забезпечуючи синхронізацію з сервером.
3. Метод *Sync*: обробляє повідомлення синхронізації від сервера, встановлюючи поточний серверний такт.

Клас *PlayerController* – відповідає за обробку введень користувача. Основні методи включають:

1. Метод *Update*: обробляє введення від користувача, включаючи рух дрона (введення по осях X та Y) та натискання кнопок прискорення і респавну.
2. Метод *SendInput*: формує та відправляє повідомлення на сервер з інформацією про введення користувача.
3. Метод *SendRespawn*: відправляє на сервер запит на респавн дрона.

Після отримання від сервера інформації про нову позицію дрона, клієнт виконує інтерполяцію для плавного переходу між старою та новою позиціями. Це реалізовано за допомогою класу Interpolator, який здійснює оновлення позицій дрона на основі отриманих даних.

3.3.2. Реалізація на стороні сервера

Серверна сторона відповідає за створення та управління дронами, обробку введень від клієнтів та синхронізацію стану гри між усіма підключеними клієнтами. Перелік основних класів та методів:

Клас *NetworkManager* – відповідає за встановлення та підтримку з'єднання з клієнтами.

1. Метод *Start*: ініціалізує сервер, встановлюючи максимальну кількість клієнтів і порт для з'єднання. Після запуску сервер очікує підключення клієнтів.
2. Метод FixedUpdate: виконує регулярні оновлення стану сервера, інкрементує серверний такт та відправляє синхронізаційні повідомлення клієнтам.
3. Метод SendSync: відправляє поточний стан гри всім підключеним клієнтам для забезпечення синхронізації.

Клас *Player* – відповідає за управління дронами гравців. Основні методи включають:

1. Метод *Spawn*: створює нового дрона для підключеного гравця та відправляє відповідне повідомлення всім клієнтам.

2. Метод *SendSpawned*: відправляє повідомлення про спавн дрона всім клієнтам.
3. Метод *Input*: обробляє введення від клієнтів та оновлює стан дронів відповідно до отриманих даних.

Клас *GameLogic* – відповідає за загальне управління ігровим процесом, включаючи обробку підключень та відключень гравців, а також зміни в стані гри. Основні методи включають:

1. Метод *PlayerCountChanged*: реагує на зміну кількості гравців у грі, адаптуючи стан гри відповідно.
2. Метод *GetSpawnPos*: визначає позицію для спавну нового дрона, забезпечуючи рівномірний розподіл дронів на ігровому полі.
3. Метод *OnApplicationQuit*: зупиняє сервер при завершенні роботи додатку, забезпечуючи коректне завершення ігрової сесії.

Інтеграція мережевого ріщення в гру про дрони на платформі Unity є складним і багатогранним процесом, що включає реалізацію різноманітних функцій як на стороні клієнта, так і на стороні сервера. Клієнт відповідає за обробку введень користувача, інтерполяцію позицій та регулярне оновлення стану, тоді як сервер забезпечує створення та управління дронами, обробку введень від клієнтів та синхронізацію стану гри між усіма підключеними клієнтами. Спільна робота цих компонентів забезпечує коректний та реалістичний ігровий процес, що мінімізує затримки та забезпечує плавність рухів дронів.

3.4. Збірка та підключення контролера до ігрового середовища

Для успішного підключення Arduino до ігрового середовища та отримання даних з гіроскопу та контролера моторів, було виконано низку підготовчих і реалізаційних заходів. Перш за все, було обрано відповідний модуль Arduino, який має достатню кількість цифрових та аналогових входів/виходів для забезпечення необхідної функціональності. До нього було підключено гіроскоп,

який дозволяє отримувати дані про просторову орієнтацію, та контролери моторів, які визначають напрямок їх обертання.

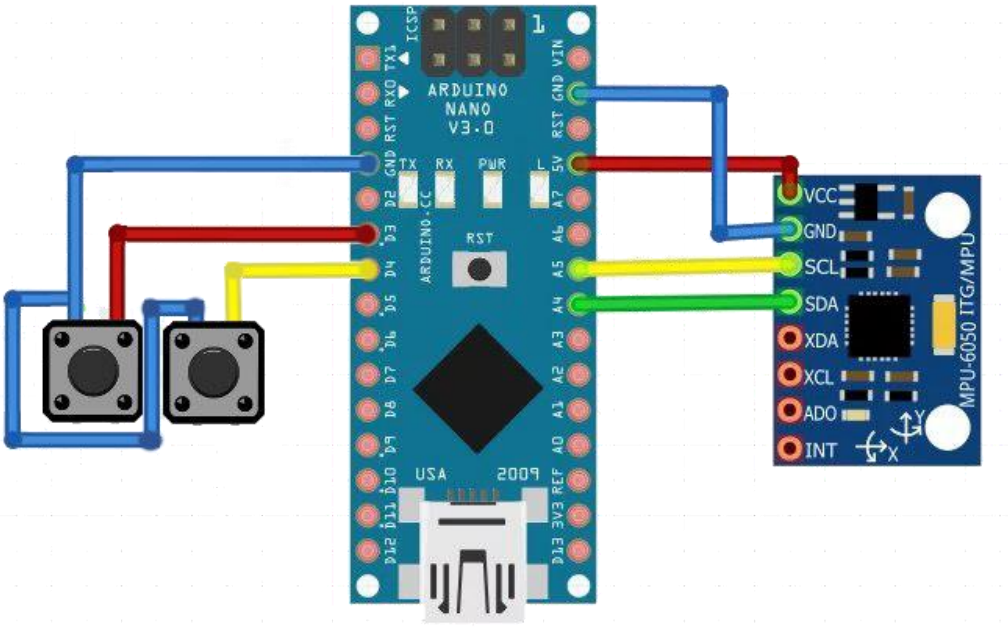

Рис. 3.4 Схема підключення MPU6050 та кнопок до Arduino Nano

Підключення компонентів до Arduino здійснювалося відповідно до зображеної схеми (див. рис. 3.4). Гіроскоп підключено до аналогових входів контролера, що дозволяє зчитувати дані про кутові швидкості по трьох осях. Контролери моторів підключені до цифрових виходів, що дозволяє керувати напрямком і швидкістю обертання моторів.

Для забезпечення безперервного зчитування та передачі даних з Arduino до ігрового середовища було розроблено відповідне програмне забезпечення. На стороні Arduino був написаний скетч, який зчитує дані з гіроскопу та контролерів моторів, обробляє їх та відправляє на комп'ютер через послідовний порт.

На стороні ігрового середовища було розроблено скрипт, який слухає всі доступні COM порти і зчитує дані, отримані від Arduino [14]. Цей скрипт обробляє отримані дані та інтерпретує їх як інпут гравця. Дані про орієнтацію гравця, отримані з гіроскопу, використовуються для визначення напрямку руху в грі, а дані про напрямок обертання моторів - для регулювання швидкості та

напрямку руху дрона. Отримані дані передаються на сервер для подальшої обробки та синхронізації між клієнтами в мультиплеєрному режимі.

3.5. Інструкція користувача

Опишемо процес запуску сервера, підключення до нього клієнта та обмін повідомленнями між клієнтом і сервером. Для прикладу розглянуто відправку клієнтом повідомлення "Hello World" та отримання у відповідь повідомлення "Hello" від сервера.

Запуск сервера

1. Ініціалізація сервера:
   1.1. Створіть екземпляр сервера *Server*.
   1.2. Запустіть сервер на певному порту (у нашому випадку порт 7777) з максимальною кількістю підключень (10).
   1.3. Призначте обробники подій (*ClientConnected*, *ClientDisconnected*, *MessageReceived*) для обробки відповідних подій сервера.
2. Оновлення сервера:
   2.1. Викликайте метод *Update()* сервера у *FixedUpdate()* для обробки подій і оновлення стану сервера.
3. Обробка отриманого повідомлення:
   3.1. У методі *OnMessageReceived* перевірте отримане повідомлення.
   3.2. Відправте відповідь *("Hello")* назад до клієнта, якщо отримане повідомлення має вміст *"Hello World"*.
4. Зупинка сервера:
   4.1. Викликайте метод *Stop()* у *OnApplicationQuit()* для коректного закриття сервера при вимкненні додатка.

Підключення клієнта до сервера

1. Ініціалізація клієнта та підключення:
   1.1. Створіть екземпляр клієнта Client.

1.2. Призначте обробники подій (Connected, ConnectionFailed, MessageReceived, Disconnected) для обробки відповідних подій клієнта.
1.3. Підключіть клієнта до сервера за допомогою методу Connect() з вказанням IP-адреси (127.0.0.1) і порту (7777).

2. Оновлення клієнта:
   2.1. Викликайте метод *Update()* клієнта у *FixedUpdate()* для обробки подій і оновлення стану клієнта.
3. Відправка повідомлення *"Hello World"*:
   3.1. У методі *OnConnected*, який викликається після успішного підключення клієнта до сервера, відправляйте повідомлення *"Hello World"* за допомогою методу *Send()*.
4. Обробка отриманих повідомлень від сервера:
   4.1. У методі *OnMessageReceived* клієнта отримуйте та обробляйте отримане повідомлення від сервера, виводячи його в консоль Unity (Debug.Log).
5. Зупинка клієнта:
   5.1. Викликайте метод *Disconnect()* у *OnApplicationQuit()* для закриття з'єднання клієнта при вимкненні додатка.

Ця інструкція надає базовий шаблон для створення мережевої взаємодії між клієнтом і сервером у мережевих іграх на Unity. Використовуючи цей підхід, можна забезпечити відправлення та отримання повідомлень між клієнтом і сервером, що є основою для створення багатокористувацького ігрового процесу. Завантажити готове рішення можна за посиланням на GitHub [15].

ВИСНОВКИ

В даній роботі було здійснено комплексне дослідження розробки мережевого рішення для ігрових додатків у середовищі Unity з інтеграцією унікального контролера на базі Arduino. Проведено вивчення теоретичних аспектів, аналіз існуючих мережевих технологій та здійснено розробку практичного прототипу, що дозволяє оцінити ефективність запропонованих рішень у реальних умовах.

По-перше, було проаналізовано сучасні мережеві технології для багатокористувацьких ігор, включаючи такі рішення як Photon Unity Network, Mirror Networking, та інші. Кожне з цих рішень має свої переваги та недоліки, які були детально розглянуті у роботі. Це дозволило визначити найбільш відповідні технології для конкретних завдань проєкту, таких як забезпечення стабільної ігрової взаємодії та ефективної передачі даних між клієнтами та сервером.

По-друге, було досліджено можливості ігрового рушія Unity для реалізації мережевих рішень та інтеграції апаратних контролерів. Було розроблено методику створення та тестування мережевого додатку, що включає в себе як програмні, так і апаратні компоненти. Особливу увагу приділено розробці функціональних можливостей та забезпеченню надійної взаємодії між ігровими об’єктами.

По-третє, проведено розробку прототипу мережевого рішення та інтеграцію контролера на базі Arduino у ігровий процес. Це дозволило оцінити практичну придатність запропонованих рішень, їх гнучкість та можливості масштабування. Важливим аспектом було забезпечення користувацької зручності та надійності контролера, що досягається за рахунок використання сучасних методів програмування та протоколів обміну даними.

У підсумку, виконана робота демонструє можливість успішної інтеграції мережевих рішень та апаратних контролерів у сучасні ігрові додатки. Крім того, проведена робота відкриває широкі перспективи для подальших досліджень, які можуть бути спрямовані на оптимізацію існуючих рішень, розширення їх функціональних можливостей та адаптацію до нових технологічних вимог. Це

відкриває широкі перспективи для розвитку ігрової індустрії та створення інноваційних продуктів, що відповідають сучасним запитам користувачів. Таким чином, робота не тільки досягає своїх цілей, але й закладає міцний фундамент для майбутніх інноваційних проектів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Прогнози ігрової індустрії на 2024 рік DeanBeat - Oksim. Oksim - IT допомога. URL: https://www.oksim.ua/2024/01/02/prognozi-igrovoï-industriï-na-2024-rik-deanbeat/ (дата звернення: 10.05.2024).
2. Клієнт-серверна архітектура. Онлайн-курси від компанії QATestLab Головна сторінка. URL: https://training.qatestlab.com/blog/technical-articles/client-server-architecture/ (дата звернення: 10.05.2024).
3. Протоколи TCP та UDP - пояснення простою мовою. DevZone. URL: https://devzone.org.ua/post/protokoly-tcp-ta-udp-poiasnennia-prostoiu-movoiu (дата звернення: 13.05.2024).
4. Photon Unity Networking for Unity Multiplayer Games PUN2. Multiplayer Game Development Made Easy Photon Engine. URL: https://www.photonengine.com/pun (дата звернення: 18.05.2024).
5. Mirror Networking – Open Source Networking for Unity. Mirror Networking – Open Source Networking for Unity. URL: https://mirror-networking.com/ (дата звернення: 18.05.2024).
6. DarkRift Networking 2 Network Unity Asset Store. Unity Asset Store - The Best Assets for Game Making. URL: https://assetstore.unity.com/packages/tools/network/darkrift-networking-2-95309 (дата звернення: 18.05.2024).
7. Announcing UNET – new multiplayer technology Unity Blog. Unity Blog. URL: https://blog.unity.com/engine-platform/announcing-unet-multiplayer-technology (дата звернення: 18.05.2024).
8. Forge Networking Remastered Network Unity Asset Store. Unity Asset Store - The Best Assets for Game Making. URL: https://assetstore.unity.com/packages/tools/network/forge-networking-remastered-38344 (дата звернення: 18.05.2024).
9. Що таке Arduino?. Hardware libre. URL: https://www.hwlibre.com/uk/що-таке-ардуїно/ (дата звернення: 16.06.2024).

10. Arduino Nano: все, що вам потрібно знати про цю плату розробки. Hardware libre. URL: https://www.hwlibre.com/uk/arduino-nano/ (дата звернення: 20.05.2024).
11. MPU-6050 - гіроскоп - акселерометр - Avislab - сайт для палких паяльників. Електроніка, схеми, плати, статті - Avislab - сайт для палких паяльників. URL: https://blog.avislab.com/mpu-6050/ (дата звернення: 20.05.2024).
12. Клас Socket krypton.com.ua. З нами програмування вчити легше. URL: https://krypton.com.ua/rozdil-1-osnovy-roboty-z-merezhamy-v-c-ta-net/klas-socket/ (дата звернення: 16.06.2024).
13. Survival Kit · Kenney. Home · Kenney. URL: https://kenney.nl/assets/survival-kit (дата звернення: 20.05.2024).
14. What is COM port? - Comprehensive guide for 2024. Serial over Ethernet. URL: https://www.serial-over-ethernet.com/serial-to-ethernet-guide/what-is-com-port/ (дата звернення: 26.05.2024).
15. GitHub - Mad1San/OnlineDroneSimulator. GitHub. URL: https://github.com/Mad1San/OnlineDroneSimulator (дата звернення: 01.06.2024).