Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“___” _____ 20 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти
на тему
“Програмне забезпечення системи керування емулятором комп’ютерної миші з використанням мікроконтролерів Joi-Con”

Виконав здобувач вищої освіти
IV курсу, групи _____
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Кот О.Р.
«____» _____ 20 р.

Керівник проекту
кандидат технічних наук, доцент
_____ Минайленко Р.М.
«____» _____ 20 р.
Рецензент _____
_____

м. Кропивницький

Міністерство освіти і науки України
Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*
Кафедра *Кібербезпеки та програмного забезпечення*
Освітній ступінь *бакалавр*
Спеціальність *123 Комп'ютерна інженерія*

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____О.А.Смірнов
« __ » _____ 20__ року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Кот Олександрі Романівні
(прізвище, ім'я, по батькові)

1. Тема роботи *Програмне забезпечення системи керування емулятором комп'ютерної миші з використанням мікроконтролерів Joi-Con*

керівник роботи *Минайленко Роман Миколайович, д-р техн. наук, доцент*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу *№131-02 від 01.04.2024 року*

2. Строк подання студентом роботи до захисту *04.06.2024 р.*

3. Мета та завдання кваліфікаційної бакалаврської роботи*: Метою розробки є програмне забезпечення для системи керування емулятором комп'ютерної миші з використанням контролерів Joy-Con*

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

1. Призначення та область використання.
2.Перегляд аналогічних існуючих систем.
3. Опис і обґрунтування проектних рішень.
4. Етапи програмування системи.
5. Впровадження системи в промислову експлуатацію.
6. Висновки

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

Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*

6. Дата видачі завдання «17» січня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної бакалаврської роботи	Строк виконання етапів кваліфікаційної бакалаврської роботи	Примітка
1.	Аналіз існуючих систем	10.03.2024 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2024 р.	
3.	Розробка моделі компонента	20.03.2024 р.	
4.	Розробка структур даних	25.03.2024 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.03.2024 р.	
6.	Програмування алгоритмів	10.04.2024 р.	
7.	Оформлення ПЗ	17.04.2024 р.	
8.	Попередній захист роботи	19.05.2024 р.	

Студент _____ _____
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Кот О. Р. Програмне забезпечення системи керування емулятором комп'ютерної миші з використанням мікроконтролерів Joi-Con. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2024.

У даній кваліфікаційній бакалаврській роботі розроблено програмне забезпечення, яке призначено для емуляції комп'ютерної миші з використанням мікроконтролерів Joi-Con.

Метою роботи є створення програмного забезпечення емулятора.

Результат роботи – програмна реалізація застосунку емулятору.

В процесі роботи над реалізацією емулятора виконано дослідження існуючих методів, алгоритмів та програмних засобів. Розроблено та реалізовано власне програмне забезпечення, здійснено опис всіх його компонентів.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

Програма може використовуватися на ПЕОМ архітектури IBM PC з ОС Windows 10/11.

Програму розроблено на мовах програмування C#.

Ключові слова: Joy-Con, контролер, емулятор, Unity, акселерометр, гіроскоп

ABSTRACT

Kot O. R. Software for a computer mouse emulator control system using Joi-Con microcontrollers. 123 Computer engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2024.

In this bachelor's thesis, software has been developed that is designed to emulate a computer mouse using Joi-Con microcontrollers.

The purpose of the work is to create emulator software.

The result of the work is the software implementation of the emulator application.

In the process of working on the implementation of the emulator, a study of existing methods, algorithms and software tools was performed. The proprietary software was developed and implemented, and all its components were described.

A convenient user interface was developed. Instructions for working with the software are given.

The program can be used on IBM PC architecture PCs with Windows 10/11.

The program is developed in C# programming language.

Keywords: Joy-Con, controller, emulator, Unity, accelerometer, gyroscope

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

PC-ABS – конструкційний матеріал для 3Д друку на основі ABS пластику з додаванням 30% PC

PCBA – Printed Circuit Board + Assembly, друкована плата

Amiibo – серія інтерактивних фігурок та карток, що випускається компанією Nintendo. Фігурки мають чіп і підтримують комунікацію з консолями при використанні спеціального зчитувача NFC міток

IDE – Інтегроване середовище розробки

NFC – Near-Field Communication, ехнологія бездротового зв'язку малого радіуса дії «за один дотик». Забезпечує обмін даними між пристроями на відстані близько 10 см

ІЧ-камера – це інфрачервоний пристрій, який створює зображення з інфрачервоного випромінювання

IMU – Інерційний вимірювальний пристрій

LRA – Лінійний резонансний привод

MCU – Multipoint Control Unit, Сервер багатосторонньої конференції апаратно-програмний пристрій, призначений для об'єднання аудіо- та відеоконференції в багатосторонньому режимі

АЦП – Аналого-цифровий перетворювач

ВСТУП

Сучасні технології надають широкі можливості для поліпшення якості життя людей, включаючи ті, хто має обмежені фізичні можливості. Однією з таких технологій є емуляція комп'ютерної миші за допомогою альтернативних пристроїв вводу, що може значно спростити процес взаємодії з комп'ютером для різних категорій користувачів. Зокрема, контролери Joy-Con від ігрової консолі Nintendo Switch володіють унікальними можливостями для створення інноваційних рішень у цій галузі завдяки своїй портативності, точності та універсальності.

Мета й завдання дослідження. Мета даного дипломного проекту — розробка програмного забезпечення (ПЗ) для системи керування емулятором комп’ютерної миші з використанням контролерів Joy-Con. Це рішення спрямоване на полегшення керування комп'ютерною мишею для всіх користувачів, особливо тих, хто має обмежені можливості. Основними завданнями цього проекту є забезпечення зручності, точності та надійності у використанні контролерів Joy-Con як альтернативного пристрою вводу. У процесі дослідження будуть розглянуті технічні та програмні аспекти розробки, включаючи алгоритми обробки даних з Joy-Con, методи передачі цих даних до комп’ютера та інтеграція з операційною системою.

Практична цінність отриманих результатів. Результати цієї кваліфікаційної роботи можуть стати кроком у напрямку створення доступніших технологій, що сприятимуть інтеграції людей з обмеженими можливостями в цифровий світ. Впровадження подібних рішень не лише покращить досвід користування даним пристроєм, але й надасть нові можливості для самореалізації та продуктивної діяльності людей зазначеної категорії.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Розроблене програмне забезпечення для системи керування емулятором комп’ютерної миші з використанням контролерів Joy-Con призначене для забезпечення альтернативного методу управління курсором на комп’ютері. Основні завдання та функції програмного забезпечення включають:

1) Емуляція рухів миші: Перетворення рухів контролерів Joy-Con у відповідні рухи курсора на екрані комп’ютера.

2) Емуляція кліків: Забезпечення можливості виконання натискань миші (лівий і правий кліки) за допомогою кнопок на Joy-Con.

3) Налаштування чутливості: Надання користувачам можливості налаштовувати чутливість та швидкість руху курсора відповідно до їхніх потреб та вподобань.

4) Підтримка налаштування кнопок: Налаштування кнопок на Joy-Con на клавіші клавіатури або простих команд, які можуть бути корисні для більш ефективного управління комп’ютером.

5) Інтерфейс користувача: Пропонування інтуїтивно зрозумілого та доступного інтерфейсу для налаштування та використання емулятора, зокрема для людей з обмеженими можливостями.

1.2 Область застосування

Система, розроблена в рамках цього проекту, призначена для широкого кола користувачів і організацій, що потребують альтернативних методів керування комп'ютерною мишею. Вона буде корисна як людям, які бажають спростити керування комп'ютером, так і людям з обмеженими можливостями, яким

традиційні методи можуть бути недоступні. Система поєднує в собі функціональність емулятора комп'ютерної миші та контролерів Joy-Con, забезпечуючи універсальність і простоту використання.

Сфери застосування:

а) Допоміжні технології для людей з обмеженими можливостями: для користувачів з фізичними обмеженнями, які не можуть ефективно користуватися стандартною комп’ютерною мишею; для осіб з травмами рук або іншими станами, що ускладнюють традиційне керування комп’ютером.

б) Освітні заклади: у школах та університетах для забезпечення доступу до комп’ютерів для всіх здобувачів освіти, включаючи тих, хто має особливі потреби, в якості навчального інструменту для вивчення альтернативних методів взаємодії з комп’ютерами.

в) Розважальні та ігрові додатки: у розважальних центрах і комп’ютерних клубах для забезпечення інноваційних методів керування іграми та іншими додатками; для геймерів, які хочуть спробувати нові способи взаємодії з іграми за допомогою Joy-Con.

г) Робочі місця та офіси: для працівників, які проводять багато часу за комп’ютером і потребують альтернативних методів керування для зменшення навантаження на руки та запобігання професійним захворюванням.

ґ) Сфера охорони здоров'я: для реабілітаційних програм, де використання Joy-Con може допомогти в розробці моторних навичок пацієнтів.

Це програмне забезпечення спрямоване на те, щоб зробити роботу з комп’ютером більш доступною та комфортною для широкого кола користувачів, забезпечуючи їм можливість ефективно взаємодіяти з сучасними цифровими технологіями.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми кваліфікаційної бакалаврської роботи

Для визначення вимог та функціоналу до програмного застосунку необхідно дослідити вже наявні аналоги. Було обрано чотири найпопулярніші емулятори комп'ютерної миші та клавіатури, що надають можливість керувати персональним комп'ютером за допомогою контролерів, після аналізу яких сформувались чіткі переваги і недоліки.

Steam

Steam - це сервіс цифрової дистрибуції відеоігор і магазин, розроблений компанією Valve Corporation. Функції клієнта Steam включають автоматизацію оновлення ігор, хмарне зберігання прогресу гри та функції спільноти, такі як прямий обмін повідомленнями, функції накладання в грі та віртуальний ринок колекційних предметів.

В Steam є вбудована підтримка контролера. Фактично, керування комп’ютером вже може бути активним. Steam має свою систему «Акордів», що мають форму певних комбінацій кнопок на контролері, які відповідають певним функціям на вашому ПК.

Наприклад, якщо використовувати контролер Xbox, ви можете утримувати кнопку Xbox і використовувати правий джойстик для переміщення миші. Правий курок відповідає за натискання лівої кнопки миші, лівий курок - правої кнопки миші. Якщо ви продовжуєте працювати Steam у фоновому режимі, це вийде миттєво, без жодних змін. Однак ви також можете ввімкнути повну підтримку контролера на робочому столі.

Цей метод має недоліки. Якщо запускається гра зі Steam, клієнт автоматично вимикає функцію управління миші, і ваш контролер діятиме як

контролер у грі. Але, якщо запустити гру не зі Steam, ваш контролер буде розпізнаватися як миша, а елементи керування працюватимуть неправильно.

Тому найкраще додавати ігри не зі Steam до вашої бібліотеки. Поки Steam виявляє гру та зміна керування працює належним чином, він вимкне функцію контролю миші.

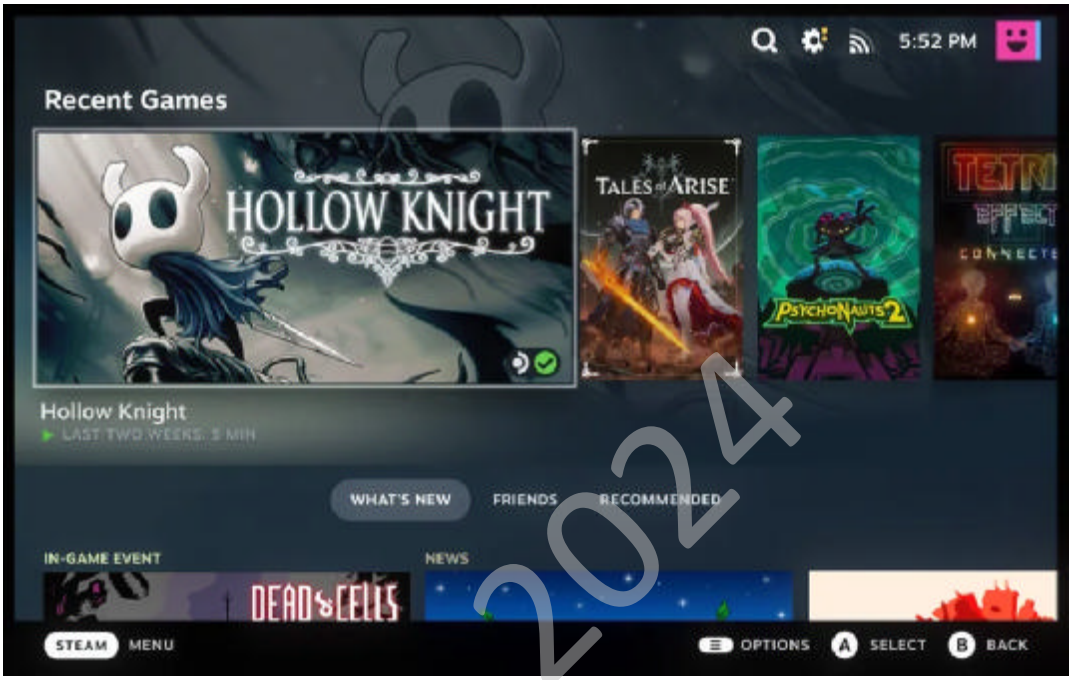

Рисунок 2.1 - Режим Big Picture у Steam

InputMapper

InputMapper 1.7 бере початкові цілі DS4Windows від DSDCS та попередніх версій InputMapper і розширює їх.

У поєднанні з розширеними макросами та можливостями налаштування, IM дає користувачам можливість не лише змінювати тип пристрою, але й спосіб перетворення та застосування входів.

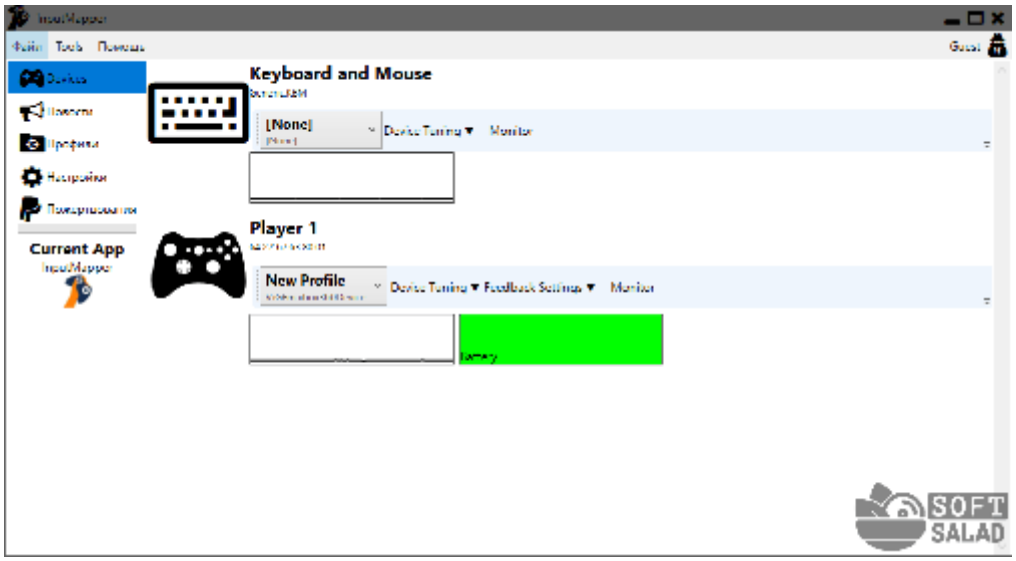

Рисунок 2.2 – Головне вікно InputMapper

Контролер DualShock від Sony є дещо унікальним, оскільки він взаємодіє з Windows не так, як контролер Xbox. Якщо ви хочете використовувати контролер PlayStation 4 як мишу, вам знадобиться окрема програма InputMapper. Коли ви під'єднаєте контролер DualShock, з'явиться запит на налаштування геймпада.

Є можливість створити профіль, який зіставляє контролер з його еквівалентами для Xbox 360 (щоб він працював в іграх, які не підтримують контролери PlayStation), або як миша і клавіатура (коли тачпад DualShock переміщує курсор, а натискання на нього викликає клацання мишею). Можна налаштувати відображення кнопок та багато інших параметрів у головному вікні програми.

Якщо контролер DualSense, то знадобиться емулятор контролера, наприклад, DS4Windows.

JoyToKey

JoyToKey - це повнофункціональний безкоштовний емулятор клавіатури, який дозволяє користувачам Windows керувати входами миші або клавіатури за допомогою кнопок і стіків своїх ігрових контролерів. Це особливо корисно для різноманітних ігор, розроблених без повної підтримки геймпаду, до яких відносяться численні веб-ігри, окремі ігрові програми, емулятори консолей та інші програми. За допомогою JoyToKey ви можете повністю налаштувати

функціонал вашого улюбленого геймпада і дозволити його кнопкам і стікам відповідати на натискання клавіш (одне або декілька) та/або рухи миші, при цьому цільова програма навіть не помітить різниці.

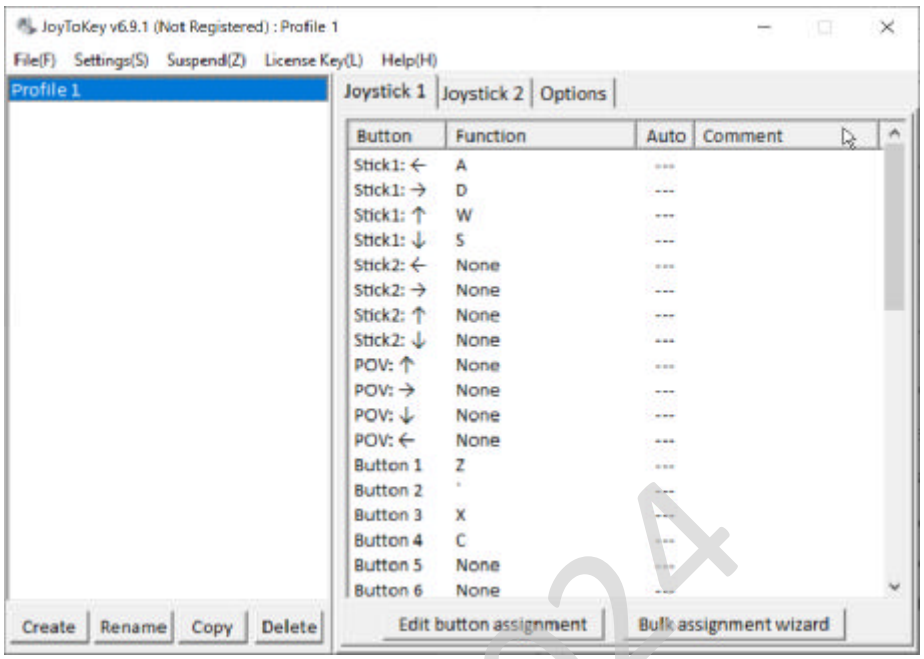

Рисунок 2.3 – Головне вікно JoyToKey

JoyToKey дозволяє вам керувати багатьма програмами (веб-іграми, браузерами, офісними програмами і навіть самою Windows) за допомогою вашого улюбленого джойстика.

Щоразу, коли ви натискаєте кнопки та стіки джойстика, JoyToKey емулює натискання клавіатури або введення миші на основі введення джойстика, так що цільова програма працює так, ніби ви використовуєте справжню клавіатуру або мишу. Ви можете створити кілька конфігураційних файлів для перемикання між різними призначеннями клавіш/миші. JoyToKey також підтримує автоматичну асоціацію з цільовими програмами, так що конфігураційний файл перемикається автоматично при зміні цільової програми.

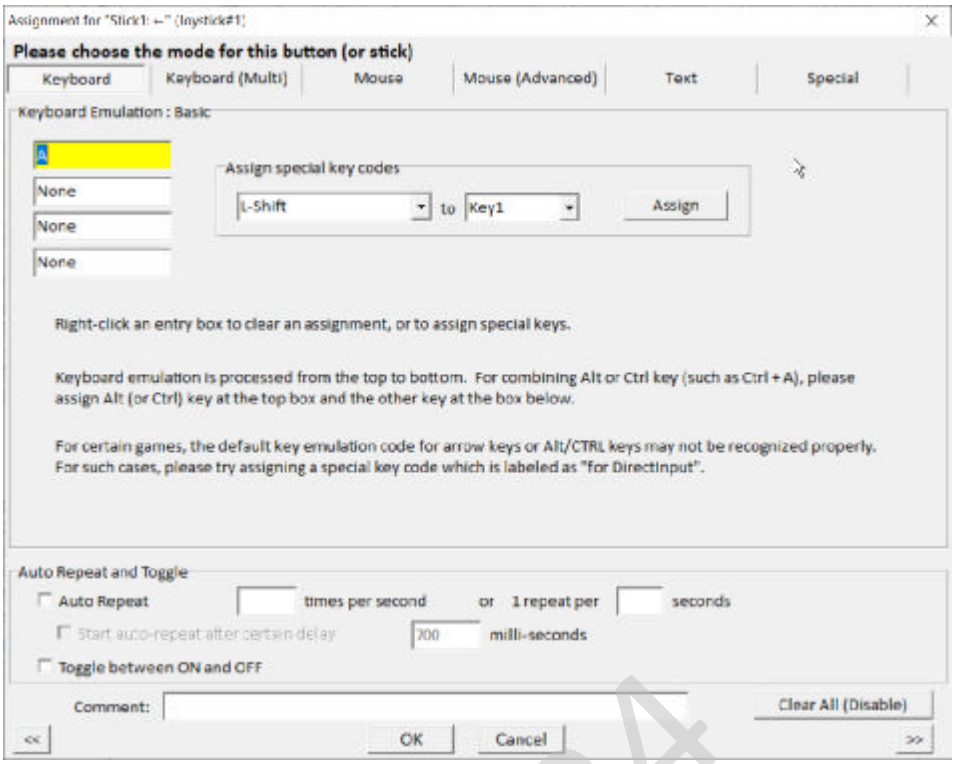

Рисунок 2.4 – Вікно налаштування кнопок у JoyToKey

JoyToKey - ідеальний помічник для фанатиків ігор, щоб зробити гру у відеоігри ще більш захоплюючою завдяки плавному відображенню введення з клавіатури та миші на джойстик та геймпади. Програма широко використовується для керування Windows-додатками, а не тільки веб-іграми за допомогою ігрового контролера.

Ще одна перевага JoyToKey - зручний інтерфейс з дуже простою навігацією та легким налаштуванням навіть для підлітків.

Окрім використання в ігрових програмах, застосунок також може допомогти звичайним користувачам, які хочуть підвищити свою загальну ефективність у різних неігрових програмах і пакетах програмного забезпечення для підвищення продуктивності. Це особливо актуально для користувачів, які хочуть спростити свій робочий процес і прив'язати різні часто використовувані комбінації клавіш і макроси до певної кнопки геймпада.

За допомогою JoyToKey ви можете створити спеціальні профілі для всіх ваших улюблених або найбільш використовуваних програм (таких як Photoshop

або веб-браузер) і автоматизувати певні завдання, призначивши їх на натискання кнопок на вашому ігровому контролері. JoyToKey можна використовувати безкоштовно за умовно-безкоштовною ліцензією (Shareware license), вона портативна і не має інсталятора/деінсталятора, а для її роботи потрібна лише наявність робочого геймпада. JoyToKey є безпечною і захищеною, протестованою на відсутність вірусів.

Якщо ви вважаєте JoyToKey корисною, ви можете придбати ліцензійний ключ у будь-який час з меню програми JoyToKey.

Controller Companion

Controller Companion - це найелегантніший спосіб керувати вашим комп'ютером за допомогою джойстика. Не потрібно вставати з дивана, коли ви використовуєте контролер з ПК. Керуйте мишею, клавіатурою та медіаплеєром. Більше не потрібно вставати, щоб запустити гру, в яку ви хотіли пограти за допомогою миші та клавіатури.

Рисунок 2.5 – Логотип Controller Companion

Придбавши програму та встановивши її через Steam, ви зможете використовувати лівий стік для переміщення миші, кнопку «А» для клацання на об'єктах, а якщо натиснути на лівий стік, ви навіть отримаєте зручну віртуальну клавіатуру, яку можна використовувати для швидкого набору тексту.

Controller Companion автоматично вимкнеться, коли виявить, що запущено повноекранний додаток, а це означає, що він має плавно перемикатися між

емуляцією миші та внутрішніми елементами керування автоматично. Controller Companion має низку простих у налаштуванні опцій у вікні налаштувань, як-от регулювання швидкості вказівника, керування мертвими зонами та створення профілів для конкретних програм. Хоча програма не працює з контролерами PlayStation (через те, як Windows їх розпізнає), вона має кнопку для налаштування емулятора контролера Xbox, який має працювати разом з геймпадами DualShock.

Керування мишею та клавіатурою на робочому столі. Коли режим Big-Picture не запущено, ви можете керувати мишею за допомогою лівого стіка, а прокручуванням - за допомогою правого. Клацніть лівий стік, і ви отримаєте клавіатурний ввід безпосередньо на робочому столі за допомогою клавіатурної спіралі в аркадному стилі.

Controller Companion вимикає бездротовий контролер, коли ви вимикаєте комп'ютер. Вам більше ніколи не доведеться виймати батарею, щоб вимкнути контролер, коли ви вимикаєте ПК (тільки для контролерів Xbox 360).

Дуже низьке завантаження процесора. Controller Companion працює у фоновому режимі і використовує якомога менше ресурсів. Він також вимкнеться, щойно буде запущено Big Picture.

Перемикання між дисплеями ПК і телевізора. Перемикайтеся між підключеними моніторами або екранами телевізора за допомогою контролера.

Використання контролера як пульта дистанційного керування. Керуйте музичним плеєром за допомогою контролера, використовуючи клавішу «Y» для відтворення/паузи та плечові кнопки для переходу до попередньої та наступної доріжки.

Налаштування. Ви можете змінити розташування кнопок на ваш розсуд.

Підтримка чат-панелі Xbox 360. Ви нарешті можете використовувати чатпад Xbox 360 на ПК. Спеціальний експериментальний драйвер дозволить вам використовувати чатпад, коли ви не граєте в ігри.

2.2 Обгрунтування вибору засобів для побудови системи та мови програмування

Розробка застосунку для емуляції комп'ютерної миші завдяки контролерам Joy-Con та обробка даних з них потребує використання ефективних та актуальних технологій. У цьому розділі розглядаються фреймворк який було обрано для реалізації проекту та контролери Joy-Con, а також обґрунтування їх вибору.

Unity

Рисунок 2.6 – Логотип Unity

Unity - це кросплатформенний ігровий рушій, розроблений компанією Unity Technologies, вперше анонсований і випущений у червні 2005 року на конференції Apple Worldwide Developers Conference як ігровий рушій для Mac OS X. З того часу рушій поступово розширювався для підтримки різноманітних настільних, мобільних, консольних платформ, платформ доповненої та віртуальної реальності. Він особливо популярний для розробки мобільних ігор для iOS та Android, вважається простим у використанні для розробників-початківців, а також популярний для розробки інді-ігор.

Unity був стандартним набором для розробки програмного забезпечення (SDK) для ігрової консолі Wii U від Nintendo, безкоштовна копія якого додавалася Nintendo до кожної ліцензії для розробників Wii U. Unity Technologies назвала це поєднання стороннього SDK «першим в індустрії».

Рушій можна використовувати для створення тривимірних (3D) і двовимірних (2D) ігор, а також інтерактивних симуляторів. Unity Software,

провідна платформа для розробки ігор, еволюціонувала далеко за межі свого зародження як ігрового рушія. Сьогодні це універсальний і потужний інструмент, що використовується розробниками по всьому світу для створення широкого спектру додатків, від імерсивних ігор до симуляторів, архітектурних візуалізацій і доповненої реальності. Рушій був прийнятий до використання в інших галузях, окрім відеоігор, таких як кіно, автомобільна промисловість, архітектура, інженерія та будівництво.

Основні можливості рушія:

а) Крос-платформна розробка. Unity дозволяє розробникам створювати додатки для широкого спектру платформ, включаючи мобільні пристрої (iOS, Android), настільні комп'ютери (Windows, macOS, Linux), консолі та нові технології, такі як VR та AR. Ця крос-платформенна сумісність є ключовим фактором популярності Unity.

б) Графіка та рендеринг. Рушій забезпечує високоякісну графіку та можливості рендерингу, підтримуючи просунуті візуальні ефекти та реалістичні симуляції.

в) Фізика та моделювання. Unity містить потужний фізичний рушій, який дозволяє розробникам створювати реалістичні рухи та взаємодії у своїх додатках. Це має вирішальне значення для створення реалістичних симуляцій та захопливого ігрового процесу.

г) Написання сценаріїв(скриптів). Unity підтримує декілька мов програмування, основною з яких є C#. Ця гнучкість дозволяє розробникам писати власний код та створювати складну поведінку для своїх проектів.

ґ) Unity Asset Store: Asset Store - це маркетплейс всередині Unity, де розробники можуть знаходити, продавати та обмінюватися ресурсами, інструментами та плагінами. Це не тільки прискорює розробку, але й сприяє створенню спільноти для співпраці.

Рисунок 2.7– Логотип Unity Asset Store

Застосування Unity Software:

а) Розробка ігор. Unity широко визнана як найкраща платформа для розробки ігор, з незліченною кількістю успішних ігор, створених за допомогою рушія. Його універсальність дозволяє розробникам створювати ігри для різних жанрів і платформ.

б) Віртуальна та доповнена реальність. Unity став найкращим вибором для розробки VR та AR. Його можливості дозволяють створювати імерсивний досвід, будь-то навчальні симулятори, освітні програми або розваги.

в) Архітектурна візуалізація. Потужні можливості рендерингу Unity роблять його чудовим інструментом для архітекторів та дизайнерів для візуалізації та презентації своїх проектів у віртуальному середовищі. Це має застосування у сфері нерухомості, містобудування та дизайну.

г) Симуляції та навчання. Unity використовується для створення реалістичних симуляцій у навчальних цілях. Сюди входять військові симуляції, медичні навчальні програми та промислові симуляції для покращення навчання у безпечному та контрольованому середовищі.

Unity Software перетворилася на багатогранну платформу для розробки, яка вийшла за межі ігрової індустрії. Її універсальність, потужні функції та підтримка

спільноти роблять її безцінним інструментом для розробників у різних галузях. Оскільки технології продовжують розвиватися, Unity, ймовірно, відіграватиме ключову роль у формуванні майбутнього інтерактивних та захоплюючих вражень.

Під час дискусій про створення ігор все частіше і частіше звучить назва рушія Unity. На сьогоднішній день це один з найкращих ігрових рушіїв у цій галузі. Unity використовується для створення більше половини всіх мобільних додатків, а для 1000 найпопулярніших додатків його частка становить 65%.

Перш ніж використовувати цей ігровий рушій, важливо вивчити всі аспекти Unity: його переваги та недоліки, щоб не розчарувати себе посеред розробки гри.

Переваги рушія Unity

Сумісність з C# та .NET. Рушій Unity використовує C# як основну мову сценаріїв. Цей ігровий рушій має велику спільноту, низький вхідний рівень та плавну криву навчання. Але в той же час він досить суворий і має багато потужних функцій для написання стабільного та ефективного коду. Все це робить технології Unity ідеальними як для початківців, так і для зрілих програмістів. Існує багато IDE для C# з чудовою підтримкою Unity, наприклад, Visual Studio, Visual Studio for Mac та Rider. Unity використовує кастомізовану версію середовища виконання Mono для запуску всіх ваших скриптів. Остання версія Unity надає програмістам доступ до функцій та інтерфейсів .NET Standard 2.1. Це означає, що багато бібліотек, написаних для звичайних .NET додатків, можна використовувати з Unity як стандартний .dll плагін. Планується підтримка більш сучасних версій .NET, що зробить ігровий рушій Unity більш інтегрованим в екосистему .NET. Розробка ігор на Unity дає вам перевагу програмування на C# як основній мові, і перевага полягає в тому, що це відносно низька точка входу і її досить легко засвоїти навіть програмісту-початківцю. Крива вивчення такої мови є відносно плавною порівняно з іншими мовами, а це означає, що ви будете регулярно заохочуватися до навчання та вдосконалення. Розробка ігор на Unity розроблена з урахуванням сумісності не лише з іншим програмним забезпеченням та середовищами виконання, але й з людьми.

Візуальне написання сценаріїв у Unity. Команда Unity докладає багато зусиль, щоб зробити його зручним для розробників ігор з низькими навичками програмування або взагалі без таких навичок. Однією з таких можливостей є візуальне написання сценаріїв. Ця функція дозволяє створювати логіку та сценарії без написання коду. Сценарії створюються шляхом комбінування вузлів. Одним з недоліків цього аспекту є гірша продуктивність у порівнянні з C#. Однак він постійно вдосконалюється, і вже розробляється новий рушій виконання візуальних скриптів, який зробить продуктивність візуальних скриптів майже ідентичною скриптам на C#.

Графіка Unity. Наразі рушій Unity має два варіанти графічного конвеєра: Вбудований та Скриптовий конвеєр рендерингу (Scriptable Render Pipeline, SPR). Вбудований конвеєр - це старий спосіб рендерингу Unity. У ньому немає нічого особливого, окрім того, що він існує вже давно.

З іншого боку, SRP є більш сучасною технологією. Її основні переваги - модульність, кастомізація та краща продуктивність. Налаштування циклу рендерингу здійснюється за допомогою скрипту на C#. Наразі існує два варіанти реалізації SRP за замовчуванням: Універсальний конвеєр рендерингу (URP) та Конвеєр рендерингу високої чіткості (HDRP).

URP орієнтований на високу продуктивність і переносимість на будь-який пристрій. Це найкращий варіант для відеоігор, які орієнтовані на невелике коло пристроїв або вимагають низького енергоспоживання.

HDRP орієнтований на високопродуктивні пристрої. Цей конвеєр рушія Unity забезпечує найкращу деталізацію та має підтримку дорогих графічних ефектів.

І URP, і HDRP підтримують широкий спектр інструментів постобробки рендерингу. Але Scriptable Render Pipeline не обмежується URP і HDRP. За потреби можна створювати власні конвеєри або модифікувати будь-який з існуючих конвеєрів під конкретні потреби. Додатковою перевагою SRP є графіка шейдерів і графіка візуальних ефектів. Shader Graph - це інструмент, який дозволяє

створювати шейдери без написання коду, просто комбінуючи вузли у візуальному середовищі з миттєвим попереднім переглядом. Visual Effect Graph - це інструмент для створення високоякісних високопродуктивних візуальних ефектів у подібному візуальному середовищі.

Різноманітність можливих проектів. Не лише агенції з розробки 3D ігор можуть отримати вигоду від використання платформи Unity для виробництва, але Unity також здатна створювати 2D ігри, фактично, це найпоширеніший інструмент розробки для 2D ігор для мобільних платформ, оскільки він забезпечує надійну можливість портування на інші ігрові магазини та платформи. Можливості крос-платформної розробки Unity роблять його ідеальним для створення мобільних ігор та ігор для Android, оскільки він дозволяє легко публікувати ваш контент. Маневреність рушія, безумовно, є однією з переваг Unity, завдяки якій він став таким популярним.

Вхід на платформу Unity є безкоштовним. У сучасній індустрії це твердження здається дивним, оскільки більшість програмного забезпечення є платним. Але Unity, принаймні безкоштовна версія, дозволяє новим розробникам спробувати те, що вона може запропонувати, і розвивати свої навички далі. Професійна версія, призначена для реальної розробки та публікації, є набагато надійнішою, але безкоштовна версія принаймні надає користувачам чудову відправну точку, з якої можна запустити творчі та технічні навички. У будь-якому випадку, краще вивчити загальні функції рушія, перш ніж вирішити, яка версія Unity більше підходить саме вам - персональна чи професійна.

Менеджер пакунків Unity та сховище ресурсів

Ігровий рушій Unity має багато інструментів для полегшення повторного використання ресурсів та коду. Два основних рішення для багаторазового використання - це магазин активів (Asset Store) та менеджер пакунків.

Asset Store - це платформа, де розробники можуть ділитися своїми ресурсами або бібліотеками. Вони можуть робити це безкоштовно або за певну ціну. Деякі ігри можна зробити з нуля до MVP, використовуючи лише ресурси з

Asset Store. Всі ресурси зі сховища активів можна скопіювати в проект і модифікувати під особисті потреби розробника гри.

Менеджер пакетів - це інший інструмент для обміну та повторного використання коду. Основна ідея керування пакунками полягає в тому, щоб підключити пакунок із зовнішнього джерела до вашого проекту, не вставляючи його безпосередньо у проект. Існує декілька джерел:

- Стандартний репозиторій Unity upm. Цей репозиторій містить деякі плагіни, створені рушієм Unity, але не включені до проекту за замовчуванням.
- Самостійне сховище upm. Компанії створюють власні репозиторії для власних інструментів, якими можна ділитися. Також існує безкоштовний репозиторій з відкритим вихідним кодом openupm, де багато розробників діляться своїми бібліотеками.
- Будь-який git-репозиторій, такий як GitHub або GitLab. Багато розробників ігор діляться своїми бібліотеками саме так, оскільки це найпростіший спосіб, який не вимагає багато церемоній.
- Спеціально запакований архів.

Всі ці інструменти є чудовими варіантами, щоб не вигадувати велосипед і прискорити розробку, використовуючи вже існуючі рішення.

Мінуси рушія Unity

Шаблони. Unity не має шаблонів для проектів, а це означає, що доведеться починати все з нуля. Це є однією з найпоширеніших проблем, з якою можуть зіткнутися розробники Unity. Рушій Unity не пропонує шаблонів для найважливіших ігрових функцій. У Unity немає зовнішніх бібліотек коду, що змушує вас створювати більше коду самостійно, порівняно з іншими платформами розробки. Бекенд Unity все ще підтримується за допомогою Mono, який є досить потужним, але все ж таки недосконалим, оскільки це призводить до повільної підтримки та оновлень.

Ліцензія. Unity - це безкоштовний рушій, який допоможе вашій грі заробити мільйони доларів. Але для того, щоб використовувати його повну

функціональність, вам потрібно придбати Unity Plus, преміум-версію рушія, яка коштує $15 на місяць. За додаткові інструменти також потрібно платити.

12 вересня 2023 року Unity оголосила про велике оновлення цінової моделі Unity. Компанія запровадила нову комісію для розробників. Розмір комісії напряму залежатиме від кількості встановлених ігор. Комісія почала діяти з 1 січня 2024 року.

Нова модель ціноутворення Uniy стосується не всіх категорій розробників. Умови для тих розробників, які користувалися безкоштовною підпискою, змінилися: Unity планує застосовувати комісію у розмірі $0,20 за кожне встановлення. Це стосується ігор з 200 000+ інсталяцій та доходом від $200 000+.

Ті, хто використовував тарифні моделі Pro та Enterprise Unity, зіткнуться з меншою комісією після 1 мільйона завантажень та доходу в $1+ млн.

Нова цінова модель Unity не поширюватиметься на демо-версії ігор, але застосовуватиметься до ігор з раннім доступом. Благодійні ігри також будуть звільнені від комісій.

Пам'ять. Великою проблемою є пам'ять. Unity має проблеми з пам'яттю, і під цим мається на увазі, що вміст Unity вимагає набагато більше пам'яті для запуску та розробки, що для користувачів слабких систем може призвести до проблем, коли проект стає складним. Ігри на Unity вимагають більше оперативної пам'яті і часто мають проблеми зі збирачем сміття. 16 ГБ оперативної пам'яті є задовільним обсягом для Unity.

Незважаючи на ці недоліки, переваги ігрового рушія Unity все ще роблять його найкращим вибором для розробників ігор.

Переваги Unity у порівнянні з рушієм Unreal Engine

Обидва рушії чудові, але мають різні цілі. Основні переваги Unity у порівнянні з Unreal:

- Простий у вивченні та використанні.
- Швидший час виконання ітерацій.
- Краща крос-платформенна інтеграція з Unity.

- Unity має чудову підтримку 2D ігор, в той час як Unreal Engine більше орієнтований на 3D.

- Unity має більш активну спільноту.

- Величезна кількість готових до використання ресурсів.

- Багато інструментів для повторного використання ресурсів та коду між проектами.

- Модульність та підтримка пакетів.

Інді-ігри, створені на Unity, з чудовими сюжетними лініями та геймплеєм, стали трендом останніх років. Вони захопили інтерес геймерів по всьому світу. Рушій Unreal Engine більше використовується для розробки AAA-ігор.

Можливості Unity та Unreal Engine дуже широкі. Щоб правильно обрати рушій для проекту, варто звернути увагу на те, яку мету потрібно досягти як розробник гри. Абсолютно все залежить від ваших цілей.

Розробка гри пов'язана з багатьма перешкодами, жодна з яких не є більшою, ніж вибір рушія для розробки гри. Unity для розробки ігор надає арсенал інструментів та ресурсів для швидкого та якісного створення гри.

Контролери Joy-Con

Рисунок 2.8 – Вигляд червоного та блакитного Joy-Con

Joy-Con - це основні ігрові контролери для ігрової консолі Nintendo Switch. Вони складаються з двох окремих блоків, кожен з яких містить аналоговий джойстик і набір кнопок. Їх можна використовувати, приєднавши до основного блоку консолі Nintendo Switch, або від'єднати і використовувати бездротовим способом, а саме: від'єднану пару Joy-Con можна використовувати одному гравцеві або розділити на двох як індивідуальні контролери.

Кожен контролер Nintendo Switch Joy-Con має повний набір кнопок і може діяти як окремий контролер. Це варте уваги інженерне рішення, оскільки для того, щоб контролери Joy-Con працювали ефективно в будь-який час, незалежно від того, з'єднані вони чи роз'єднані, конструкція вимагала сильного дублювання і повної розробки системи. Завдяки акселерометру та гіроскопічному датчику руху в обох контролерах, незалежне керування рухами вліво та вправо є простим та інтуїтивно зрозумілим.

Joy-Con - контролери, які роблять можливими нові види ігор для використання з Nintendo Switch. Універсальний Joy-Con пропонує гравцям безліч дивовижних нових способів розважитися. Два Joy-Con можна використовувати незалежно в кожній руці або разом як один ігровий контролер, якщо прикріпити їх до аксесуару «Joy-Con Grip». Їх також можна приєднати до основної консолі для використання в ручному режимі або поділитися з друзями, щоб насолоджуватися грою удвох у підтримуваних іграх. Кожен Joy-Con має повний набір кнопок і може діяти як окремий контролер, а також оснащений акселерометром і гіроскопом, що дає змогу незалежно керувати рухами вліво і вправо.

Joy-Con розповсюджуються парами, позначеними як «Joy-Con L» та «Joy-Con R» відповідно. Кожен з них має розміри 35,9 на 102 на 13,9 міліметрів, а Joy-Con L і R важать 49 грамів і 52,1 грама відповідно. Якщо вимірювати від верхньої частини аналогового джойстика до кінчика спускового гачка ZL/ZR, то гранична глибина становить 28,4 міліметра.

Joy-Con можна прикріпити до боків консолі Switch за допомогою рейок або від'єднати і використовувати бездротовим способом - або як пару, або розділити

між двома різними гравцями. До однієї консолі Switch можна під'єднати до 8 Joy-Con одночасно. За бажанням Joy-Con можна приєднати до аксесуару «Joy-Con Grip», з можливістю зарядки або без неї, який перетворює контролери на більш традиційний форм-фактор, схожий на геймпад.

Рисунок 2.9 - Joy-Con приєднані до Joy-Con Grip

Від'єднані від консолі, обидва контролери Joy-Con працюють автономно один від одного і зв'язуються з консоллю через Bluetooth. Передбачено кріплення для зап'ястного ремінця, які аналогічно встановлюються, надягаючи їх на рейки контролерів. Кріплення для ремінців мають заокруглену форму та рельєфні плечові кнопки для покращення ергономіки Joy-Con при індивідуальному використанні. Аксесуари для зап'ястя кріпляться до Joy-Con, коли їх використовують як пульт.

Ми бачимо лінії зварювання, також відомі як в'язані лінії або лінії зварювання, на чорних пластикових частинах корпусу біля кнопок і отворів для світлових труб. Лінії зварного шва виникають через те, що смола розтікається навколо стрижня, який використовується для формування отворів. Налаштування параметрів лиття може зменшити їхню видимість, але іноді їх неможливо повністю усунути через конструкцію деталі та форми.

Ось кілька особливостей, на які варто звернути увагу:

- Пружинне з'єднання між світловими трубками. Пружинне з'єднання між світловими трубками дозволяє деталі трохи згинатися під час збирання. Чотири окремі світлові трубки повинні вставлятися в невелике вікно, і якби не ця

пружинна конструкція, збірка була б шалено ускладнена, а нагромадження допусків призвело б до величезного головного болю.

Рисунок 2.10 - Пружинне з'єднання між світловими трубками

- Кнопки на браслеті. На кожному ремінці також є по дві кнопки - оскільки ремінець закриває кнопки SL і SR на Joy-Con, ці дві кнопки забезпечують наскрізне натискання кнопок. Дві легкі пружини стиснення забезпечують зворотну пружину.

Рисунок 2.11 - Дві легкі пружини під кнопками

- Металева рейка. П-подібна металева рейка з листового металу з'єднується з пазами на бічній стінці Joy-Con. Товщина металевої рейки становить 1,1 мм, а її стінки - 1,5 мм.

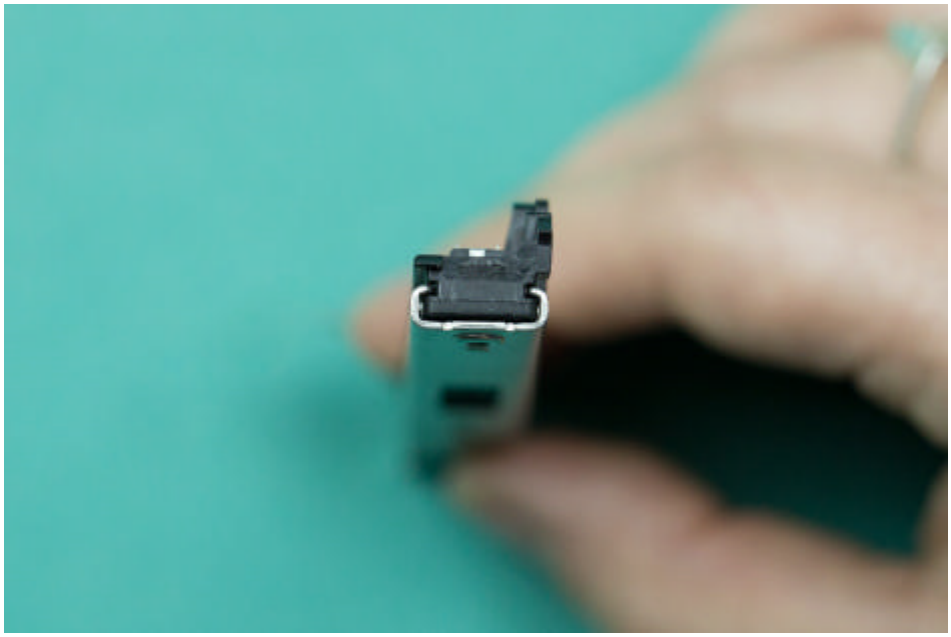

Рисунок 2.12 - Металева рейка

Рисунок 2.13 - Металева рейка у розібраному стані

- Фіксатор ремінця на зап'ясті. Там, де нейлонові ремінці входять в корпус ремінця, є невелика біла деталь. Це розумний замок для ремінця. Після того, як ви натиснете на нього, ремінець не зможе вийти, навіть якщо ви випадково натиснете кнопку розблокування.

Рисунок 2.14 - Фіксатор ремінця

Joy-Con і ремінці дуже добре поєднуються один з одним. Вони легко надягаються один на одного, а коли фіксуються, залишається лише ледь помітна слабина.

Joy-Con містять незнімні літій-іонні полімерні акумулятори на 3,7 вольта 525 мАг ємністю 1,9 ват-години; вони заряджаються при підключенні до консолі Switch, яка сама заряджається. Окремий аксесуар «зарядний руків'я» дозволяє заряджати контролери в конфігурації геймпада через USB-C. 16 червня 2017 року Nintendo випустила кріплення для Joy-Con з батарейками типу AA, яке надягається на Joy-Con так само, як і кріплення для наручного ремінця. Батарея в кожному з Joy-Con займає майже всю ширину контролера.

Рисунок 2.15 - Батарея у Joy-Con

Обидва контролери містять аналоговий джойстик, що клацає, чотири лицьові кнопки, дві верхні кнопки, дві бічні кнопки, доступні при від'єднанні (які стають плечовими кнопками, якщо їх тримати горизонтально) і позначені як SL і SR, кнопку + або -, кнопку «Home». Joy-Con L містить кнопки керування напрямком руху, кнопку «-», верхні кнопки, позначені як L і ZL, а також кнопку скріншоту. В оновленні, випущеному 18 жовтня 2017 року, кнопка скріншоту також здатна записувати до 30 секунд ігрового процесу в деяких іграх, якщо утримувати її натиснутою протягом секунди. Joy-Con R містить кнопки A, B, X і Y, кнопку «+», верхні кнопки, позначені як R і ZR, і кнопку «Home». Крім того, Nintendo не використовувала традиційний спосіб зчитування кнопок «одна сторона витягнута вгору, інша до землі». Натомість вони використали конфігурацію клавіатури, де кнопки розташовані в рядках і стовпчиках. Для зчитування кнопок вони використовували сканер клавіатури, вбудований в BCM20734 з тактовою частотою 128 КГц.

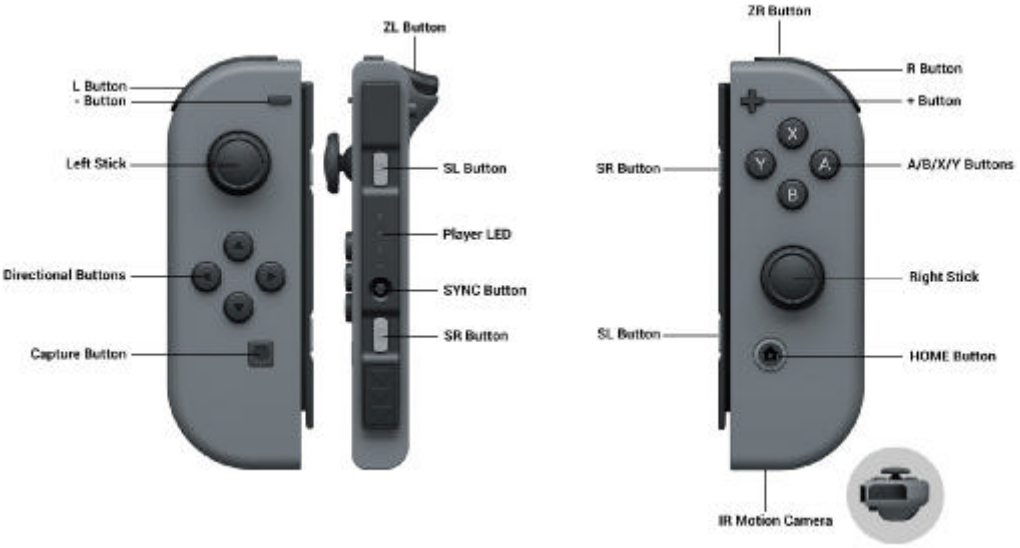

Рисунок 2.16 – Схема кнопок на Joy-Con

Корпус Joy-Con фактично складається з 3 частин:

- верхній корпус;
- нижній корпус;
- боковина.

Вони виготовлені з PC-ABS, а стінки досить жорсткі, як на товщину лише 1,1 мм у більшості місць. Сертифікати та інші маркування надруковані безпосередньо на пластику - зовнішня поверхня має гладеньке матове покриття, внутрішня поверхня - незавершена - на ній видно сліди від інструментів.

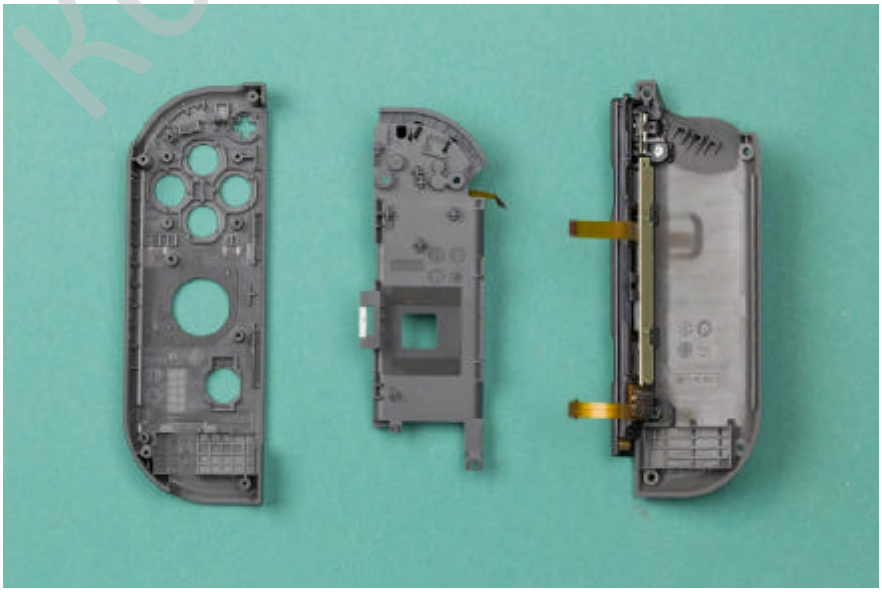

Рисунок 2.17 - Механічні корпуси Joy-Con

Верхній і нижній корпуси мають невеликі вирівнювальні елементи вздовж кромки замість звичних для нас застібок.

Рисунок 2.18 - Застібки на верхньому та нижньому корпусах Joy-Con

На боковій стінці розміщено ключові механізми для кріплення Joy-Con до консолі, руків'я та інших аксесуарів. Листовий метал обмежує підпружинений язичок, який відповідає за фіксацію Joy-Con на руків'ї та консолі. Гнучкий кабель відповідає за зв'язок між консоллю та Joy-Con.

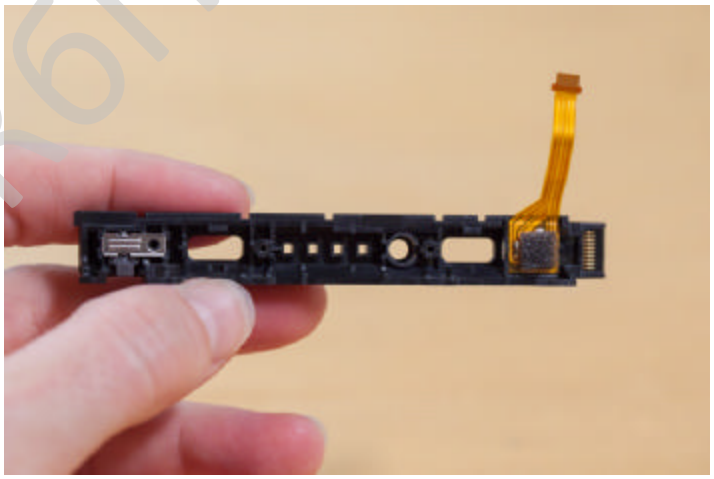

Рисунок 2.19 - Бокова стінка Joy-Con

Світлопроводи в бічній стінці досить витончені: чотири однакові світлопроводи вкладаються в маленький чорний пластиковий тримач, який

забезпечує маскування і розділення, а потім весь цей вузол вкладається в бічну стінку разом. Це набагато дешевше, ніж спільно відлиті світлові труби.

Рисунок 2.20 – Світлопроводи

Під кнопками SL, SR і синхронізації знаходяться невеликі металеві купольні перемикачі, які зазвичай використовуються нечасто.

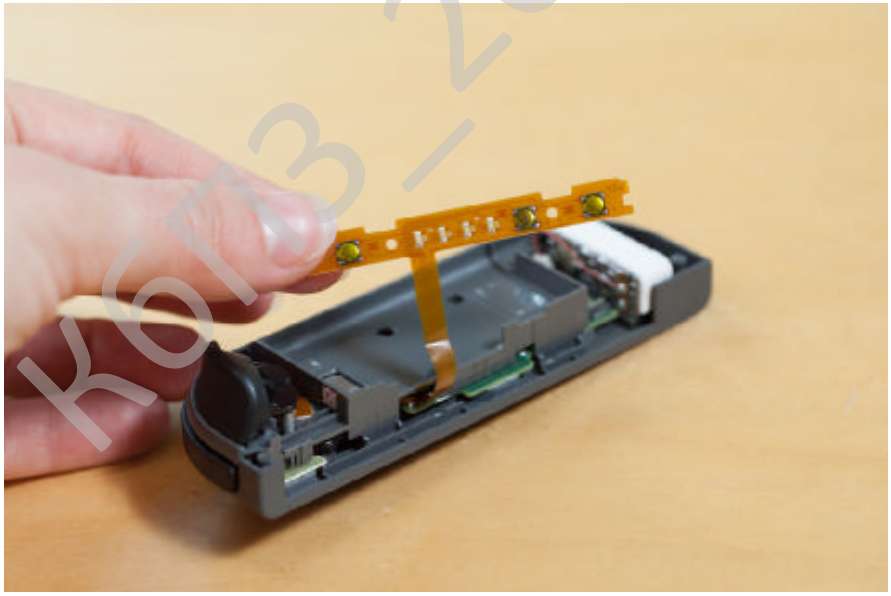

Рисунок 2.21 - Металеві купольні перемикачі

Тут ми маємо основну PCBA лівого Joy-Con. Між купольними перемикачами і жорсткими верхніми частинами кнопок залишився середній шар еластомеру. Проте, замість того, щоб виконувати провідну функцію, як у попередніх поколіннях, він слугує лише для повернення пружини.

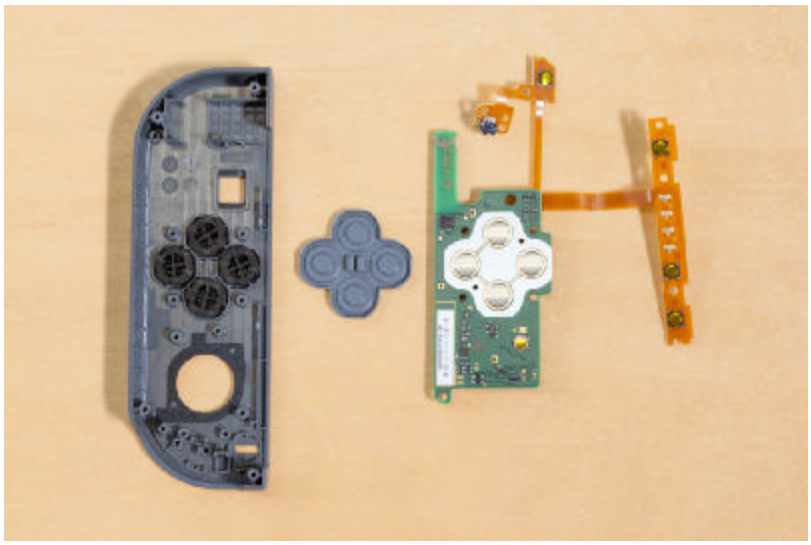

Рисунок 2.22 - Основна PCBA та верхня частина корпусу лівого Joy-Con

Спусковий гачок і плечовий перемикач тепер є просто кнопковими перемикачами. Кнопка спускового гачка використовує дві пружини стиснення для зворотного ходу, і таким чином, відстань ходу значно зменшилася порівняно зі старими контролерами.

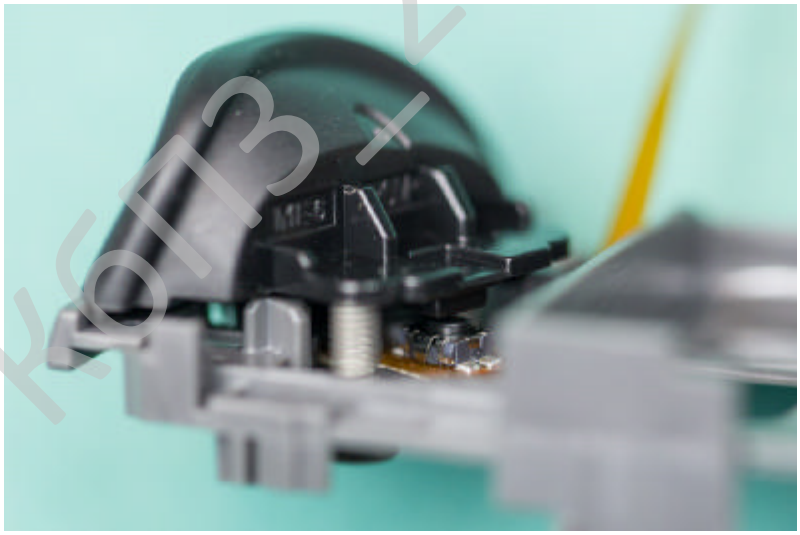

Рисунок 2.23 - Кнопкові перемикачами

Тепер перемикач-джойстик значно зменшився у вазі. Новий низькопрофільний модуль також більш захищений від пилу порівняно зі старим поколінням.

Рисунок 2.24 - Перемикач-джойстик

Кожен Joy-Con містить акселерометр і гіроскоп, які можна використовувати для відстеження руху. Ігри можуть підтримувати використання Joy-Con для вказівного керування, коли він від'єднаний, без сенсорної панелі. Правий Joy-Con (R) містить інфрачервоний датчик відстеження глибини, який може зчитувати об'єкти та рухи, що знаходяться перед ним. Правий Joy-Con (R) також містить зчитувач ближнього зв'язку для використання з Amiibo. Окрім того, що лівий та правий Joy-Con мають різні функціональні кнопки, основною відмінністю правого Joy-Con є доданий модуль ІЧ-камери та зчитувач/записувач NFC.

Рисунок 2.25 - Модуль ІЧ-камери

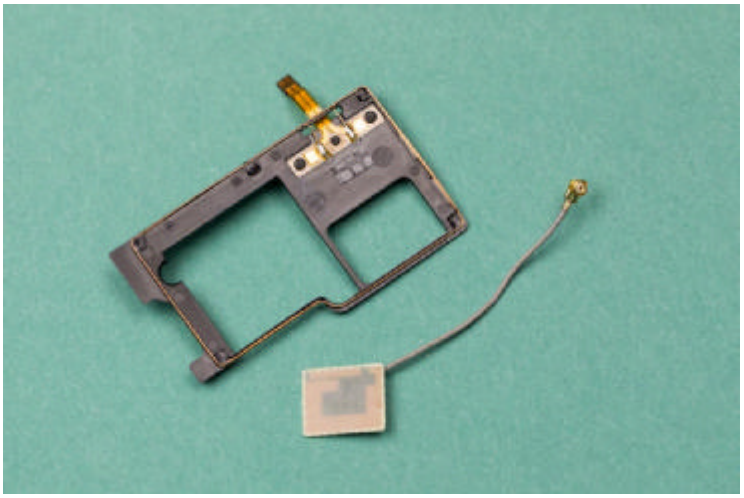

Рисунок 2.26 - Зчитувач/записувач NFC

Joy-Con містить механізм тактильного зворотного зв'язку, відомий як «HD Rumble», який був розроблений у партнерстві з Immersion Corporation. Nintendo заявила, що система може генерувати тонкий тактильний зворотний зв'язок, наприклад, відчуття окремих кубиків льоду та води у склянці. Завдяки Apple, яка є лідером у розробці технології Taptic Engine, все більше інвестувань вкладають в розробку високоточного дизайну з тактильним зворотним зв'язком.

Власна версія тактильного двигуна Nintendo у кожному Joy-Con називається «HD rumble». Хоча ми не відкривали його повністю, він виглядає як лінійний актуатор, але в кілька разів більший за той, що ми бачили в Apple Watch.

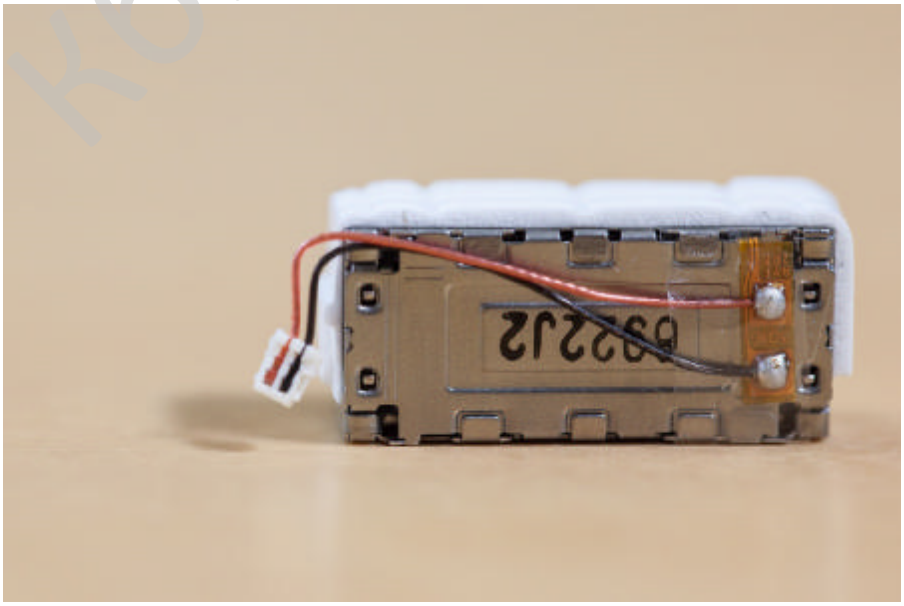

Рисунок 2.27 - Тактильний двигун «HD rumble»

Особливий інтерес викликає близькість IMU до тактильного контролера, який може бути складно розмістити через їхні конкуруючі потреби та необхідні драйвери. Joy-Con справляється з цим завданням завдяки наступним компонентам і вибору дизайну:

- IMU (TDK IM-20600) має вбудований 3-осьовий гіроскоп і 3-осьовий акселерометр для забезпечення точних позиційних і обертальних вимірювань.
- Кожен LRA (Альпійський реактор) дозволяє точно керувати програмним забезпеченням частотним сигналом, який створює вібрацію як по осі X, так і по осі Y.
- Застосувавши режекторний фільтр до механічно з'єднаних IMU та LRA, інженери змогли розумно відфільтрувати резонансні частоти LRA на 160 Гц та 320 Гц, щоб запобігти перевантаженню IMU щоразу, коли активується тактильний датчик.
- Щоб зменшити кількість мікросхем і обчислювальне навантаження, необхідне для MCU (і знизити загальне енергоспоживання контролера), вони додали інтегрований аналого-цифровий перетворювач (АЦП).

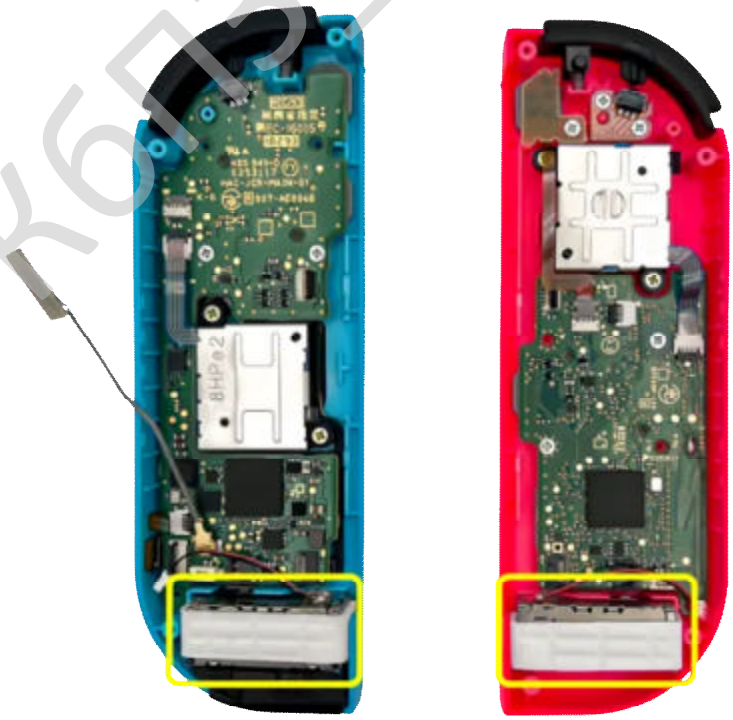

Рисунок 2.28 - IMU всередині Joy-Con

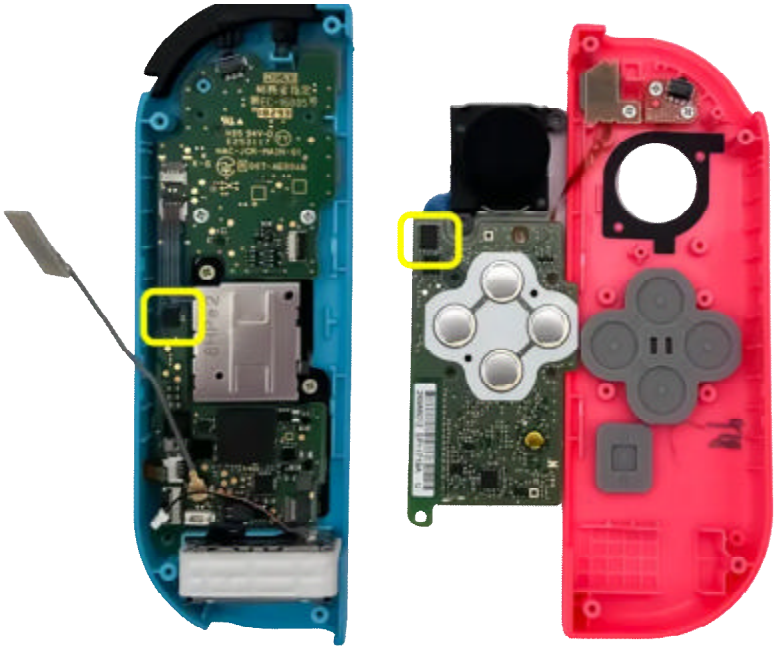

Рисунок 2.29 - Аналого-цифровий перетворювач всередині Joy-Con

Координація електричного та механічного дизайну

Контролери, як і вся побутова електроніка, регулярно тестуються на різних етапах життєвого циклу розробки продукту, щоб гарантувати, що вони витримають сотні мільйонів життєвих циклів. Контролер Joy-Con не став винятком. Успішний продаж Joy-Con по всьому світу свідчить про те, що він був чітко спроектований і протестований для досягнення рівня довговічності, який витримує великі масштаби використання.

Досвід показує, що для того, щоб будь-який продукт відповідав цим стандартам, необхідний високий рівень координації між інженерами-електриками та інженерами-механіками. Команди розробників повинні інтегрувати свої проекти за формою, розмірами та функціями, щоб:

- Досягти більшої продуктивності та надійності.
- Створювати менші та щільніші вузли.
- Приводити проводки/кабелів у відповідність до фізичної форми та обмежень конструкції.
- Ретельно вибирати компоненти, щоб зменшити витрати та спростити специфікації.

Усередині контролерів Joy-Con є чіткі докази таких скоординованих рішень, а саме:

- Використання гнучких друкованих плат (FPC) забезпечує вищу надійність завдяки відсутності роз'єму та прямому з'єднанню з однією стороною.

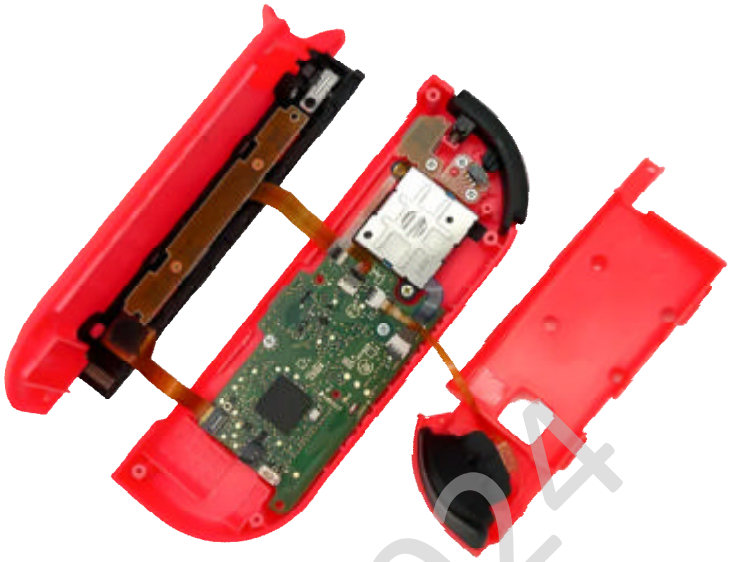

Рисунок 2.30 - Гнучка друкована плата всередині Joy-Con

- Використання заздалегідь визначеної форми зменшує потребу в прокладанні кабелів під час виробництва.

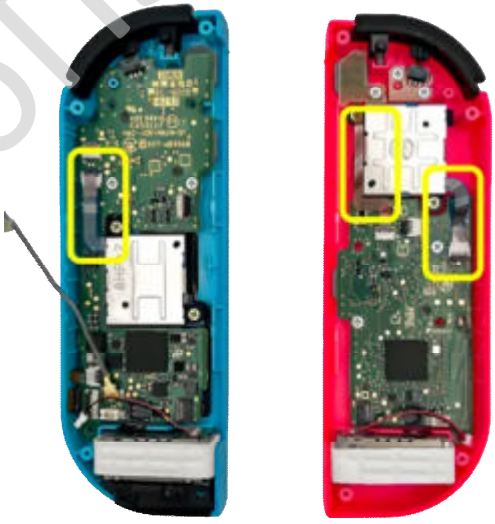

Рисунок 2.31 – Раціональне використання місця для прокладки кабелів всередині Joy-Con

- Використання коаксіального кабелю для переміщення антени NFC подалі від джойстика дозволяє генерувати та приймати сигнал NFC подалі від потенційних перешкод, спричинених металевим корпусом джойстика та мідними заливками друкованої плати. Використання в обох контролерах мікросхеми Broadcom BCM20734 з інтегрованим 2,4-гігагерцовим приймачем (Bluetooth) усуває необхідність у зовнішній антені для зв'язку між Joy-Con і консоллю Switch.

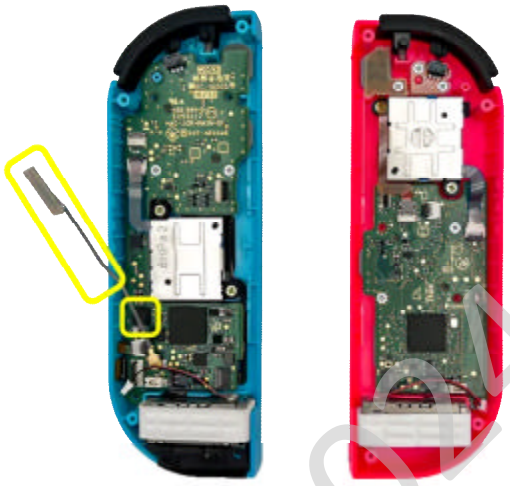

Рисунок 2.32 - Антена NFC і приймач Broadcom всередині Joy-Con

- Додавання процесора Cortex-M3 забезпечує пряме з'єднання та спеціальну обробку для контролера, світлодіодів, кнопок-джойстиків, IMU, NFC та мікросхеми обробки зображень.

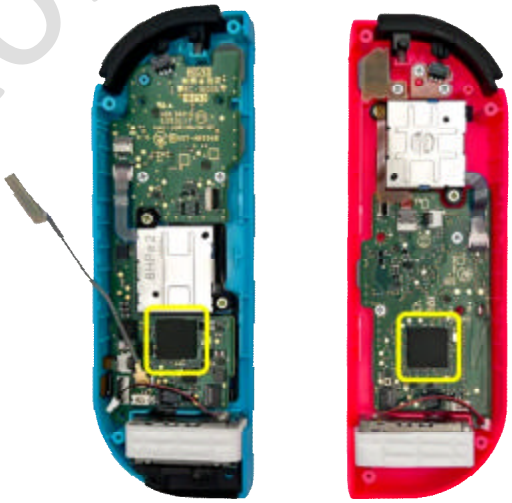

Рисунок 2.33 - Процесори Cortex-M3 всередині Joy-Con

- Використання спеціалізованої мікросхеми TI BQ24072 допомагає керувати продуктивністю літій-іонної (Li-Ion) батареї та дозволяє перемикатися між зовнішнім джерелом живлення та внутрішньою батареєю для зменшення кількості циклів заряду/розряду, що безпосередньо впливає на термін служби Li-Ion батареї. Налаштування цифрової шини ST LD39020 на лінійну напругу 1,8 В дозволяє мікроконтролеру знижувати напругу до 1,2 В для внутрішнього використання і для пам'яті.

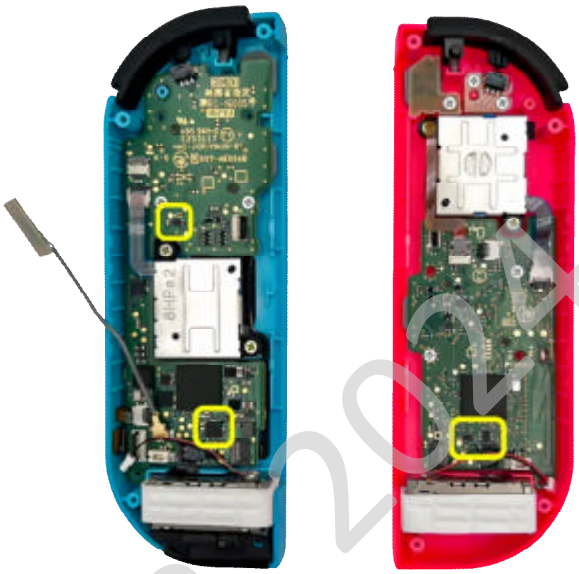

Рисунок 2.34 - Цифрової шини всередині Joy-Con

- Застосування малого форм-фактора для обмотки антени створює ефективне використання простору.

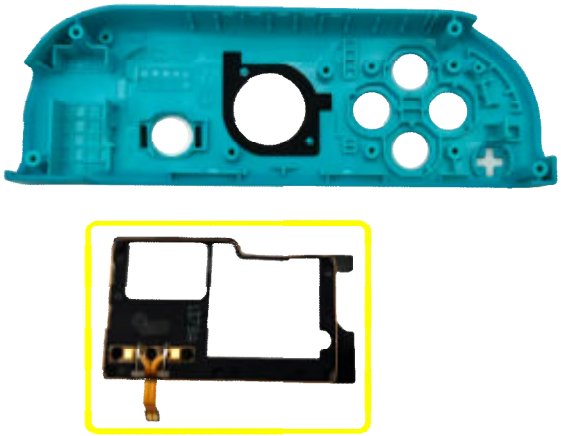

Рисунок 2.35– Ефективне використання простору всередині Joy-Con

- Дизайн маркування фізичних роз'ємів, що слугує подвійним цілям, забезпечує точну збірку під час виробництва та легку ідентифікацію користувачем.

Рисунок 2.36 - Маркування на Joy-Con та ремінця

- Додавання листового металу підсилює та зміцнює надзвичайно малі та крихкі пластикові деталі.

Рисунок 2.37 - Металеві складові всередині Joy-Con та ремінця

Продуманий дизайн і взаємодія між цими ключовими компонентами, безумовно, мали важливий вплив на продуктивність і зручність використання контролерів Joy-Con - це чудовий приклад правильної системної інтеграції. Невдовзі після публічного релізу було виявлено, що Joy-Con може підключатися до інших персональних комп'ютерів і мобільних пристроїв з підтримкою Bluetooth

і використовуватися з ними. Офіційна підтримка Joy-Con також була додана до iPhone та iPad в рамках оновлень iOS 16 та iPadOS 16 відповідно.

До комплекту консолі Nintendo Swtich входить один неоново-червоний лівий Joy-Con, один неоново-блакитний правий Joy-Con та два чорні ремінці Joy-Con. Ігри оживають завдяки простим у використанні елементам керування рухом та вібрації з поліпшеним звуком HD, вбудованим у кожну консоль Joy-Con. Залежно від гри, ви можете використовувати по одному Joy-Con у кожній руці - або навіть віддати другий другу.

2.3 Розгорнута постановка завдання

Програмне забезпечення, розробку якого планується зробити в даній роботі, є базою для керування емулятором комп'ютерної миші з використанням контролерів Joy-Con. Розробка даного програмного забезпечення передбачає його використання, зокрема людям з обмеженими можливостями, ефективне керування комп'ютером, використовуючи Joy-Con, а також налаштування параметрів емулятора відповідно до потреб користувача.

Основні компоненти системи:

- Модуль приймання та обробки даних з контролерів Joy-Con. Приймання даних з контролерів Joy-Con: нахил, рух, швидкість переміщення, натискання кнопок та обробка їх для програмного застосунку.
- Модуль налаштування взаємодії контролерів Joy-Con із програмним забезпеченням. Налаштування як саме застосунок реагує на контролери Joy-Con та корегування за необхідності.
- Модуль взаємодії між системою та програмним забезпеченням. Мета роботи полягає в керуванні комп’ютерної миші, що включає використання функцій та констант з Windows API (Application Programming Interface) для успішної реалізації.

- Модуль налаштування контролерів Joy-Con. Налаштування кнопок контролерів на кнопки комп’ютерної миші, клавіатури або простих команд.

- Модуль керування комп’ютерною мишою за допомогою контролерів Joy-Con. Конвертація отриманих даних з контролерів Joy-Con у рух комп’ютерної миші на екрані.

Враховуючи зазначене вище, план кроків реалізації мети цієї дипломної роботи наступний:

а) Розробити загальну архітектуру ПЗ, що включає компоненти для зчитування даних з Joy-Con, обробки цих даних та емуляції миші.

б) Спроектувати інтерфейс користувача для налаштування параметрів емулятора. Реалізувати модулі для підключення Joy-Con до комп'ютера та зчитування даних з його датчиків.

в) Розробити алгоритми для обробки даних з акселерометрів та гіроскопів Joy-Con і перетворення їх у рухи курсора.

г) Забезпечити можливість емуляції натискань миші за допомогою кнопок Joy-Con. Використати Unity для створення інтерфейсу користувача та забезпечення візуального зворотного зв'язку.

г) Інтегрувати програмні модулі в середовище Unity для забезпечення узгодженої роботи системи.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Для реалізації роботи було обрано фреймворк Unity. Unity — це потужний багатоплатформний ігровий рушій та інтегроване середовище розробки (IDE), що дозволяє створювати 2D та 3D ігри, симуляції та інтерактивні додатки для різних платформ, включаючи ПК, консолі, мобільні пристрої та віртуальну реальність.

Основні компоненти Unity:

1. Редактор Unity: графічний інтерфейс, що надає інструменти для створення, редагування та тестування ігор і додатків. Включає в себе сцени, компоненти, об'єкти та ресурси.

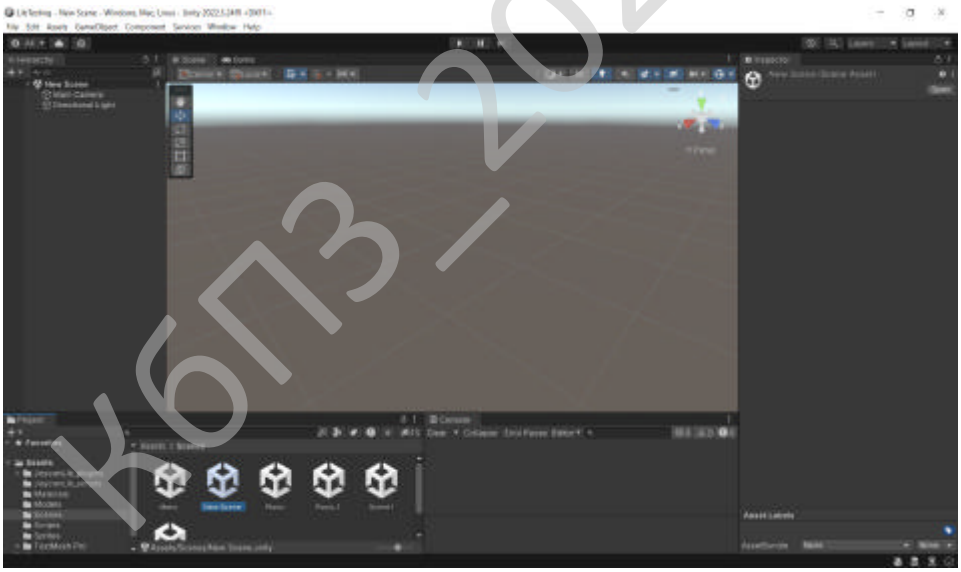

Рисунок 3.1 – Редактор Unity

2. Сцена (Scene): середовище, де розміщуються всі ігрові об'єкти. Кожна сцена може містити різні рівні, місії чи екрани гри.

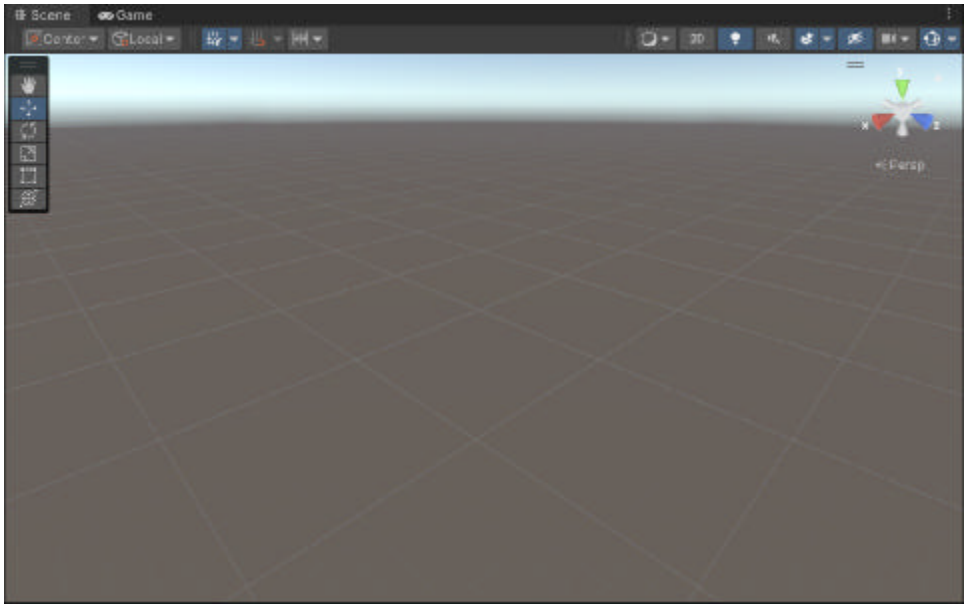

Рисунок 3.1 – Сцена у Unity

3. Ігрові об'єкти (GameObjects): основні будівельні блоки в Unity, які можуть представляти будь-який об'єкт у грі, від персонажів і реквізиту до камер і світла.

4. Компоненти (Components): модулі, які додають функціональність до ігрових об'єктів. Кожен об'єкт може мати безліч компонентів, таких як фізика, рендеринг, скрипти, звуки та інші.

5. Скрипти: коди, написані на C#, які додають поведінку до ігрових об'єктів. Скрипти використовуються для управління ігровою логікою, взаємодії з користувачем, фізики та інших аспектів гри.

6. Фізичний рушій: Unity включає вбудовану фізику для 2D і 3D, що дозволяє створювати реалістичні симуляції з використанням фізичних законів.

7. Рендеринг: потужний механізм рендерингу, який підтримує сучасні графічні ефекти, такі як тіні, освітлення, постобробка, шейдери та інші.

8. Інтерфейс користувача (UI): інструменти для створення та налаштування користувацького інтерфейсу, включаючи меню, кнопки, текстові поля та інші елементи.

Загальна ідея програми. Основна мета створити програму емулятор комп'ютерної миші та продемонструвати можливості контролерів Joy-Con.

Контролери будуть виступати як заміна комп’ютерної миші на більш інтуїтивно зрозумілим маніпулятором для керування комп’ютером.

Інтерфейс користувача. Інтерфейс інтуїтивно зрозумілий. При запуску застосунку з'являється меню, яке складається з 4 кнопок, які ведуть до самого функціоналу застосунку. Перша переносить користувача в сам емулятор, що надає можливість керувати комп’ютерною мишею через контролери Joy-Con. Дві наступні кнопки ведуть до демонстраційних програм-ігор. Вони присутні для демонстрації можливостей контролерів Joy-Con. Остання кнопка відповідає за налаштування кнопок Joy-Con на розсуд користувача.

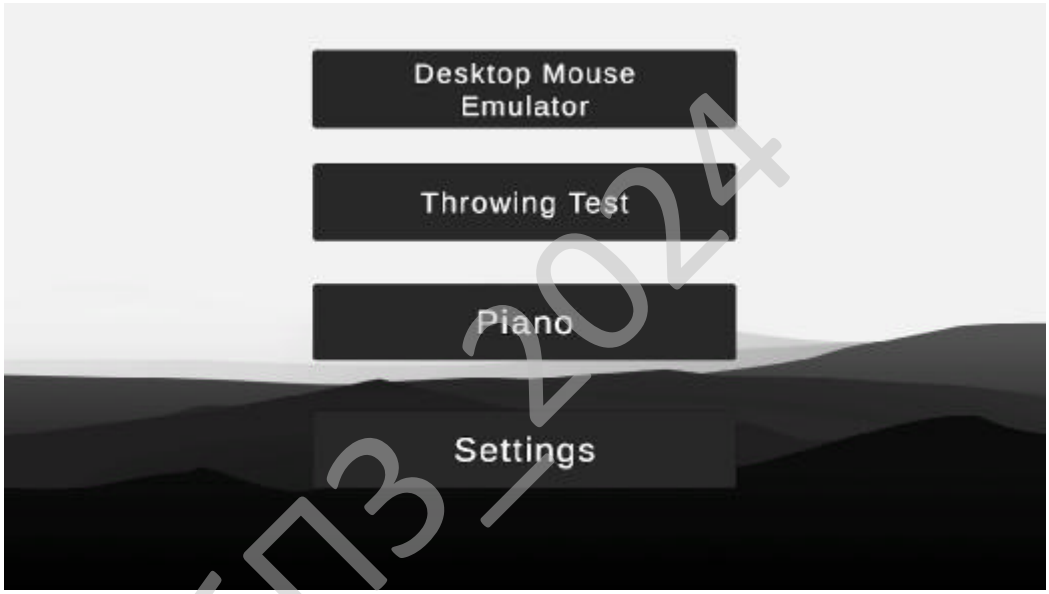

Рисунок 3.3 – Інтерфейс меню програми

Введення даних. Основні дані входять з контролерів Joy-Con: кнопок, нахилу та руху. Кожен лівий і правий контролер використовує власний процесор, пам'ять, літій-іонну батарею, мікросхему керування батареєю, стік, блок інерційних вимірювань (IMU), лінійний резонансний актуатор (LRA) і тактильний драйвер (HD Rumble). Блок інерційних вимірювань має вбудований 3-осьовий гіроскоп і 3-осьовий акселерометр для забезпечення точних позиційних і обертальних вимірювань. Кожен лінійний резонансний актуатор дозволяє точно керувати програмним забезпеченням частотного сигналу, який створює вібрацію як по осі X, так і по осі Y. Решта даних надходить через взаємодію із програмним

забезпеченням через комп'ютерну мишу та клавіатуру у деяких випадках. Сам по собі Unity не має можливості працювати з контролерами Joy-Con. Тому було використано бібліотеку JoyconLib яка дає таку можливість. Бібліотека оголошує прості функції для керування контролерами та дозволяє дуже легко впроваджувати функціонал контролерів Joy-Con. Нижче приведені функції бібліотеки, які приймають натиск кнопок на контролері.

```
public bool GetButtonDown(Button b)
    {
        return buttons_down[(int)b];
    }
public bool GetButton(Button b)
    {
        return buttons[(int)b];
    }
public bool GetButtonUp(Button b)
    {
        return buttons_up[(int)b];
    }
```

У приведених функціях використовується тип `Button`, який знаходиться у тій же бібліотеці JoyconLib. У бібліотеці оголошені всі кнопки, присутні на контролерах Joy-Con. Кнопки `DPAD_DOWN`, `DPAD_RIGHT`, `DPAD_LEFT` та `DPAD_UP` окрім кнопок «Вниз», «Вправо», «Вліво», «Угору» на лівому Joy-Con також відповідають за кнопки «B», «Y», «A» та «X» на правому Joy-Con відповідно.

```
public enum Button : int
    {
        DPAD_DOWN = 0,
        DPAD_RIGHT = 1,
        DPAD_LEFT = 2,
        DPAD_UP = 3,
        SL = 4,
        SR = 5,
        MINUS = 6,
```

```
        HOME = 7,
        PLUS = 8,
        CAPTURE = 9,
        STICK = 10,
        SHOULDER_1 = 11,
        SHOULDER_2 = 12
    };
```

`buttons_down`, `buttons` та `buttons_up` стани натиснутої, затиснутої та віджатої кнопки на контролері Joy-Con. Оголошені наступним чином:

```
private bool[] buttons_down = new bool[13];
private bool[] buttons_up = new bool[13];
private bool[] buttons = new bool[13];
```

При підключенні до комп'ютера завдяки Bluetooth емулятор зчитує рух, нахил, швидкість переміщення та конвертує це все у керування комп'ютерної миші на екрані комп'ютера. Для зчитування руху, нахилу та швидкості переміщення було написано наступний код.

```
private Int16[] acc_r = { 0, 0, 0 };
private Vector3 acc_g;

private Int16[] gyr_r = { 0, 0, 0 };
private Int16[] gyr_neutral = { 0, 0, 0 };
private Vector3 gyr_g;



private float err;
public Vector3 i_b, j_b, k_b, k_acc;
private Vector3 d_theta;
private Vector3 i_b_;
private Vector3 w_a, w_g;
private Quaternion vec;

private int ProcessIMU(byte[] report_buf)
{
```

```
if (!imu_enabled | state < state_.IMU_DATA_OK)
      return -1;

if (report_buf[0] != 0x30) return -1; // no gyro data
int dt = (report_buf[1] - timestamp);
if (report_buf[1] < timestamp) dt += 0x100;

for (int n = 0; n < 3; ++n)
{
      ExtractIMUValues(report_buf, n);


      float dt_sec = 0.005f * dt;
      sum[0] += gyr_g.x * dt_sec;
      sum[1] += gyr_g.y * dt_sec;
      sum[2] += gyr_g.z * dt_sec;

      if (isLeft)
      {
            gyr_g.y *= -1;
            gyr_g.z *= -1;
            acc_g.y *= -1;
            acc_g.z *= -1;
      }

      if (first_imu_packet)
      {
            i_b = new Vector3(1, 0, 0);
            j_b = new Vector3(0, 1, 0);
            k_b = new Vector3(0, 0, 1);
            first_imu_packet = false;
      }
      else
      {
            k_acc = -Vector3.Normalize(acc_g);
            w_a = Vector3.Cross(k_b, k_acc);
            w_g = -gyr_g * dt_sec;
            d_theta = (filterweight * w_a + w_g) / (1f + filterweight);
            k_b += Vector3.Cross(d_theta, k_b);
            i_b += Vector3.Cross(d_theta, i_b);
            j_b += Vector3.Cross(d_theta, j_b);
            //Correction, ensure new axes are orthogonal
```

```
                    err = Vector3.Dot(i_b, j_b) * 0.5f;
                    i_b_ = Vector3.Normalize(i_b - err * j_b);
                    j_b = Vector3.Normalize(j_b - err * i_b);
                    i_b = i_b_;
                    k_b = Vector3.Cross(i_b, j_b);
                }
                dt = 1;
            }
        timestamp = report_buf[1] + 2;
        return 0;
    }
```

Vector3 є структурою, яка представляє вектор або точку в тривимірному просторі (3D). Це один із найважливіших типів даних у Unity, особливо для роботи з позиціями, напрямками, масштабами та іншими векторами в тривимірному просторі.

Vector3 має три складові: x, y і z, які представляють координати в просторі.

```
public struct Vector3 {
    public float x;
    public float y;
    public float z;
}
```

Функція ProcessIMU розшифровується як процесс інерційного вимірювального пристрою, що збирає вхідні дані з контролерів Joy-Con та оброблює їх.

Основна мета - це керування комп’ютерною мишею. Декілька функцій, що наведені нижче, дозволяють цього досягнути.

Функція переміщення комп’ютерної миші на екрані:

```
if(joycon.isLeft == true)Mouse.current.WarpCursorPosition(new Vector2(transform.position.x * -10, transform.position.z * -10));
```

`joycon` - клас у бібліотеці, що створює об’єкт і дозволяє використовувати функції з бібліотеки JoyconLib.

`isLeft` - константа типу bool, яка відповідає лівий контролер Joy-Con чи ні.

`Mouse` - звертання до комп’ютерної миші.

`WarpCursorPosition` – функція, яка надає можливість змінити позицію миші. На десктопних платформах можете переміщувати курсор миші за допомогою коду. При цьому переміщується власне системний курсор миші, а не лише внутрішньо збережена у Unity позиція миші.

`transform` - об’єкт, до якого прив’язане керування.

`position` - позиція у 3D просторі.

`x` та `y` - лінії координат у 3D просторі.

Функція емуляції кліків миші:

```
if(joycon.GetButtonUp(Joycon.Button.DPAD_RIGHT)){
      DoMouseClick(2);
}
if(joycon.GetButton(Joycon.Button.DPAD_RIGHT)){
      DoMouseClick(3);
}

if(joycon.GetButtonUp(Joycon.Button.DPAD_LEFT)){
      DoMouseClick(0);
}
if(joycon.GetButton(Joycon.Button.DPAD_LEFT)){
      DoMouseClick(1);
}


public void DoMouseClick(int button){
      uint X = (uint)Input.mousePosition.x;
      uint Y = (uint)Input.mousePosition.y;
      switch(button){
            case 0:
                mouse_event(MOUSEEVENTF_LEFTDOWN | MOUSEEVENTF_LEFTUP, X, Y,
0, 0);
                break;
```

```
                case 1:
                    mouse_event(MOUSEEVENTF_LEFTDOWN, X, Y, 0, 0);
                    break;

                case 2:
                    mouse_event(MOUSEEVENTF_RIGHTDOWN | MOUSEEVENTF_RIGHTUP, X,
Y, 0, 0);
                    break;

                case 3:
                    mouse_event(MOUSEEVENTF_RIGHTDOWN, X, Y, 0, 0);
                    break;
            }
        }
```

Для того щоб код спрацював коректно, застосовується функція та константи з Windows API(Application Programming Interface). Windows API — це набір функцій, класів, структур та констант, які надає операційна система Windows для розробників програмного забезпечення. Він дозволяє створювати програми, які можуть взаємодіяти з компонентами операційної системи Windows. Windows API надає інтерфейси для роботи з апаратними ресурсами комп'ютера, файлами, мережами, графічним інтерфейсом користувача та іншими системними функціями.

Windows API - це основний інтерфейс прикладного програмування, який дозволяє комп'ютерній програмі отримати доступ до функцій операційної системи Microsoft Windows, в якій вона працює.

```
        [DllImport("user32.dll",CharSet=CharSet.Auto,
CallingConvention=CallingConvention.StdCall)]
            public static extern void mouse_event(uint dwFlags, uint dx, uint dy,
uint cButtons, uint dwExtraInfo);
            private const int MOUSEEVENTF_LEFTDOWN = 0x02;
            private const int MOUSEEVENTF_LEFTUP = 0x04;
            private const int MOUSEEVENTF_RIGHTDOWN = 0x08;
            private const int MOUSEEVENTF_RIGHTUP = 0x10;
```

Функція `mouse_event` викликає емуляцію заданого кліку миші. Константи `MOUSEEVENTF_LEFTDOWN`, `MOUSEEVENTF_LEFTUP`, `MOUSEEVENTF_RIGHTDOWN` та `MOUSEEVENTF_RIGHTUP` відповідають за натиснуту і відпущену ліву кнопку мишу та натиснуту і відпущену праву кнопку мишу відповідно.

При запуску емулятора вид наступний.

Рисунок 3.4 – Вид емулятора

Керування комп'ютером відбувається за допомогою «рук», до одної з яких прив'язаний курсор миші. Прозорість застосунку необхідна для взаємодії з системою. Для досягання прозорості та взаємодії з елементами системи було використано декілька функцій з Windows API. За прозорість застосунку відповідає скрипт `TransparentWindow`.

```
using System;
using System.Runtime.InteropServices;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class TransparentWindow : MonoBehaviour{
```

```
            [DllImport("user32.dll")]
            public static extern int MessageBox(IntPtr hWnd, string text, string
caption, uint type);

            [DllImport("user32.dll")]
            public static extern IntPtr GetActiveWindow();

            [DllImport("user32.dll")]
            private static extern int SetWindowLong(IntPtr hWnd, int nIndex, uint
dwNewLong);

            [DllImport("user32.dll", SetLastError = true)]
            static extern bool SetWindowPos(IntPtr hWnd, IntPtr hWndInsertAfter, int
X, int Y, int cx, int cy, uint uFlags);

            private struct MARGINS{
                public int cxLeftWidth;
                public int cxRightWidth;
                public int cyTopHeight;
                public int cyBottomHeight;
            }

            [DllImport("Dwmapi.dll")]
            private static extern uint DwmExtendFrameIntoClientArea(IntPtr hWnd, ref
MARGINS margins);

            const int GWL_EXSTYLE = -20;

            const uint WS_EX_LAYERED = 0x00080000;
            const uint WS_EX_TRANSPARENT = 0x00000020;

            static readonly IntPtr HWND_TOPMOST = new IntPtr(-1);

            private void Start(){

        #if !UNITY_EDITOR
                IntPtr hWnd = GetActiveWindow();

                MARGINS margins = new MARGINS {cxLeftWidth = -1};
                DwmExtendFrameIntoClientArea(hWnd, ref margins);
```

```
            SetWindowLong(hWnd, GWL_EXSTYLE, WS_EX_LAYERED | WS_EX_TRANSPARENT);

            SetWindowPos(hWnd, HWND_TOPMOST, 0, 0, 0, 0, 0);
    #endif
        }
    }
```

3.2 Розробка структурної схеми

В даному розділі буде наведено структурну схему системи програмного забезпечення. На ній зображено внутрішню будову та взаємозв'язки між основними компонентами системи. Ця схема допомагає зрозуміти, як організована структура системи емулятора.

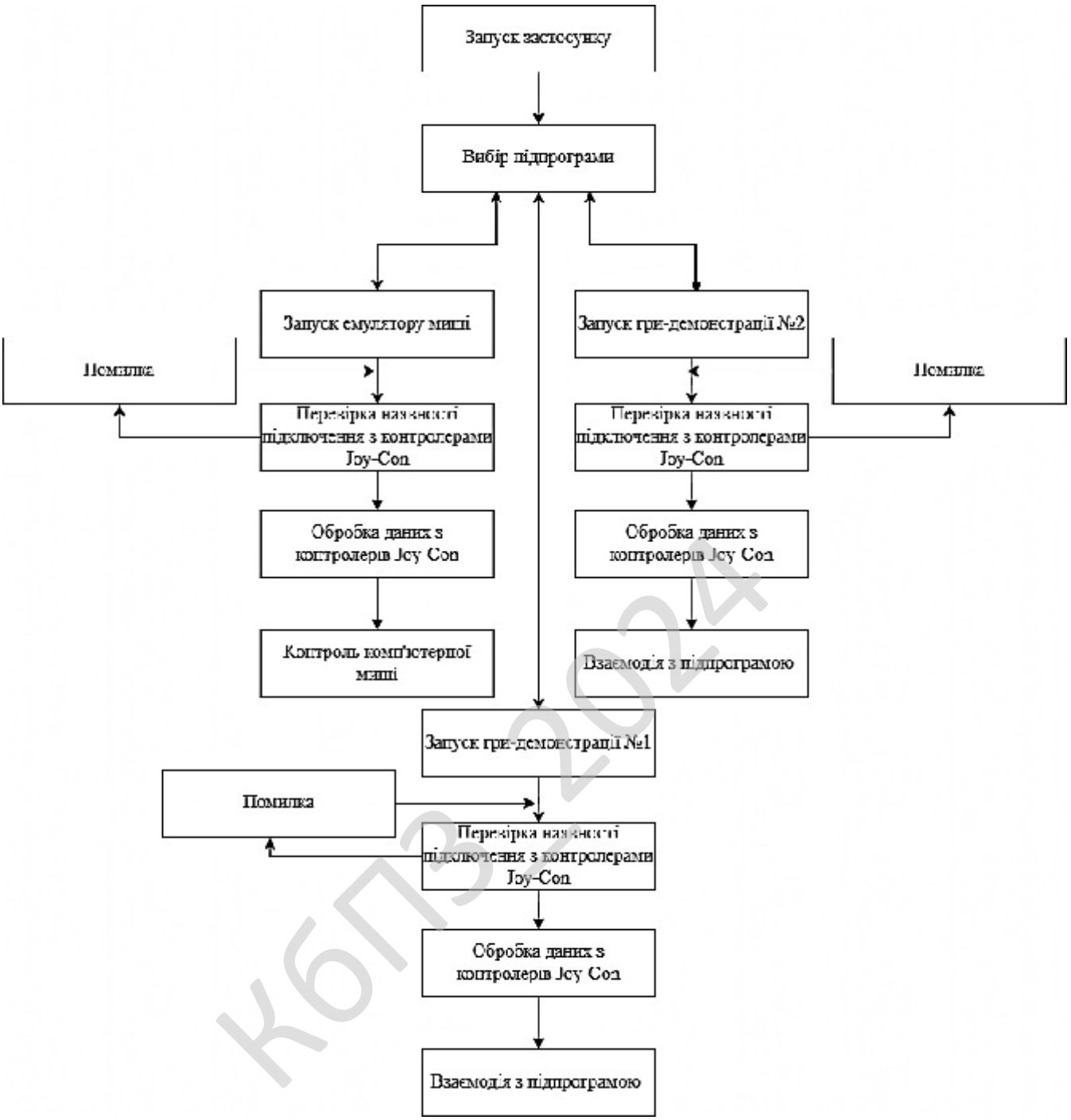

Рисунок 3.5 – Структурна схема застосунку

3.3 Розробка функціональної схеми

В даному розділі було наведено функціональну схему системи. Вона показує функціональні блоки програмного забезпечення. На відміну від структурної схеми, що демонструє внутрішню будову приладу, функціональна схема відображає функції та зв'язки між різними частинами застосунку.

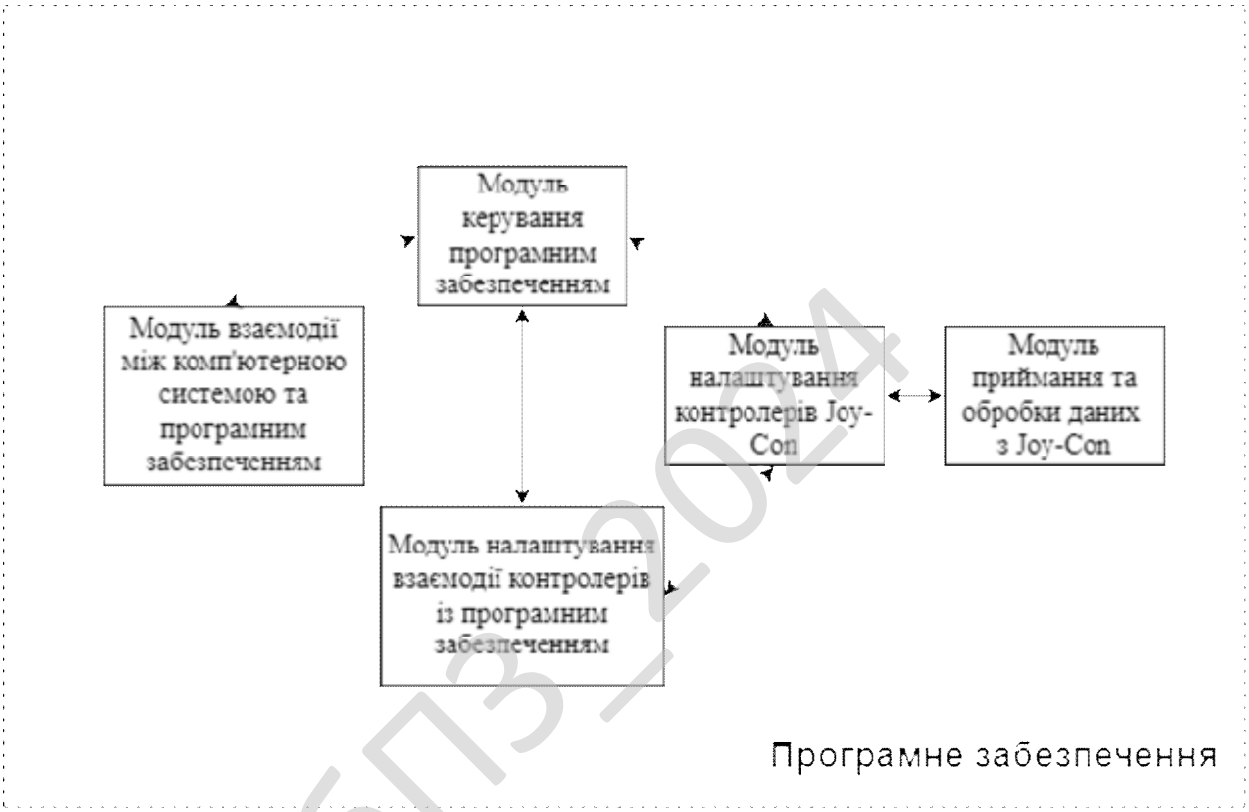

Рисунок 3.6 – Функціональна схема системи

Ця функціональна схема застосунку демонструє роль кожного компоненту впродовж всієї роботи системи. Розглядаються основні принципи взаємодії між її компонентами, що спрямовані на забезпечення їхньої ефективної роботи та надійності, підкреслюється важливість їхньої внутрішньої взаємодії. Також у цьому розділі висвітлені аспекти, які стосуються обміну даними між компонентами системи, обробки та аналізу даних та запитів, отриманих від контролерів Joy-Con.

3.4 Розробка діаграми процесів

Схема містить компоненти, які описують взаємодію існуючих процесів при використанні створеного програмного забезпечення. Кожен з цих компонентів має зв’язки з іншими, що і було відображено на схемі нижче.

Кожен компонент схеми відображає функції, що виконує програмне забезпечення при взаємодії з ним. Використання цієї діаграми спрощує розуміння взаємодії користувача і програмного забезпечення та зв’язків компонентів застосунку.

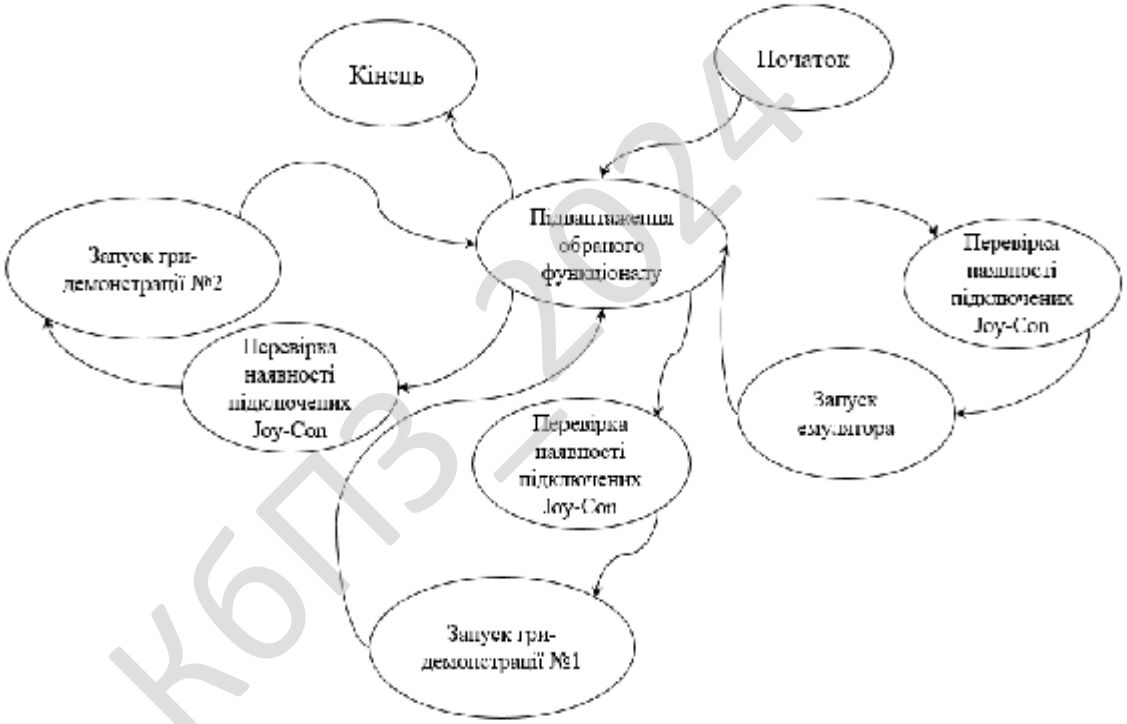

Рисунок 3.7 – Діаграма процесів застосунку

4 РЕАЛІЗАЦІЯ ПРОЕКТУ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ПРАВИЛЬНІСТЬ ПРОЕКТНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

Блок-схема схема демонструє основні компоненти роботи програмного застосунку, а саме: виведення головного вікна ПЗ, налаштування програмного забезпечення, аналіз та обчислення даних з контролерів Joy-Con, основні принципи взаємодії між компонентами.

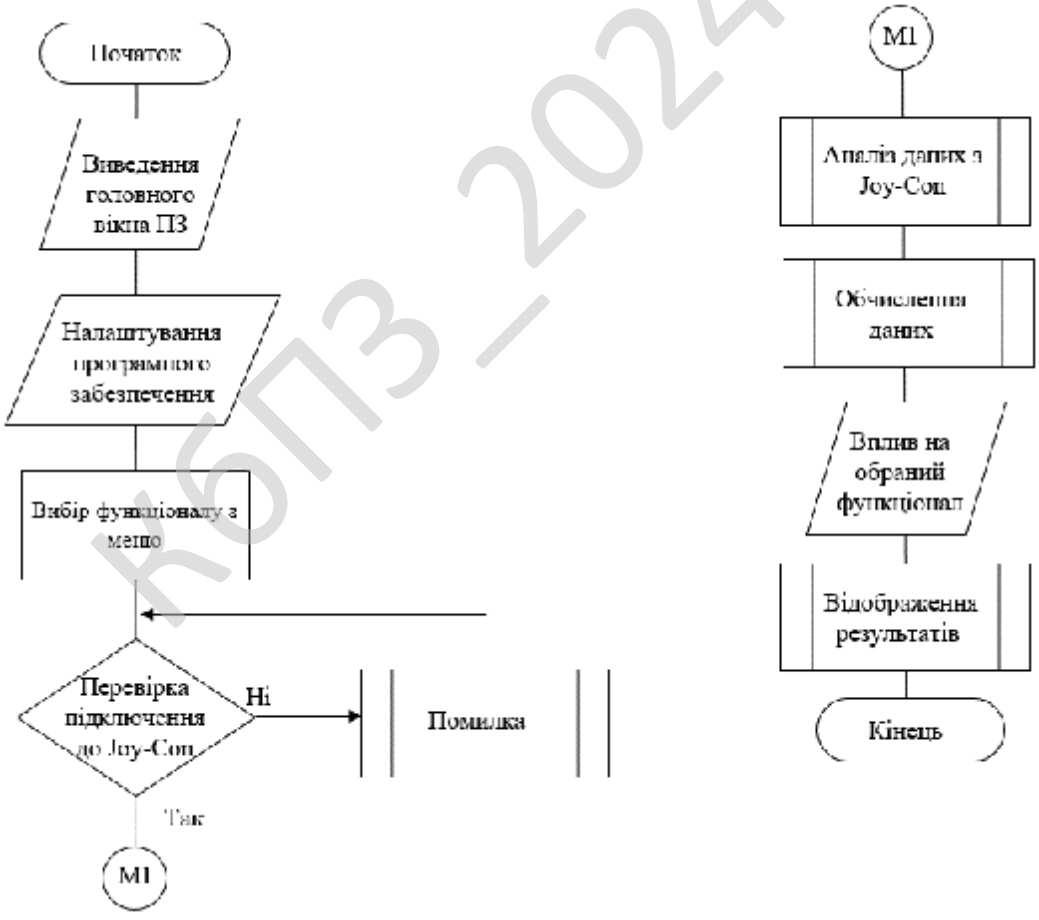

Рисунок 4.1 – Блок-схема роботи основної програми

Блок-схема роботи підпрограми демонструє головні принципи роботи підпрограми, а саме: налаштування програмного забезпечення, завантаження функціоналу та запуск роботи емулятора, отримання та обчислення даних з контролерів Joy-Con, переміщення курсору миші на основі отриманих даних, принципи взаємодії та взаємозв'язку компонентів.

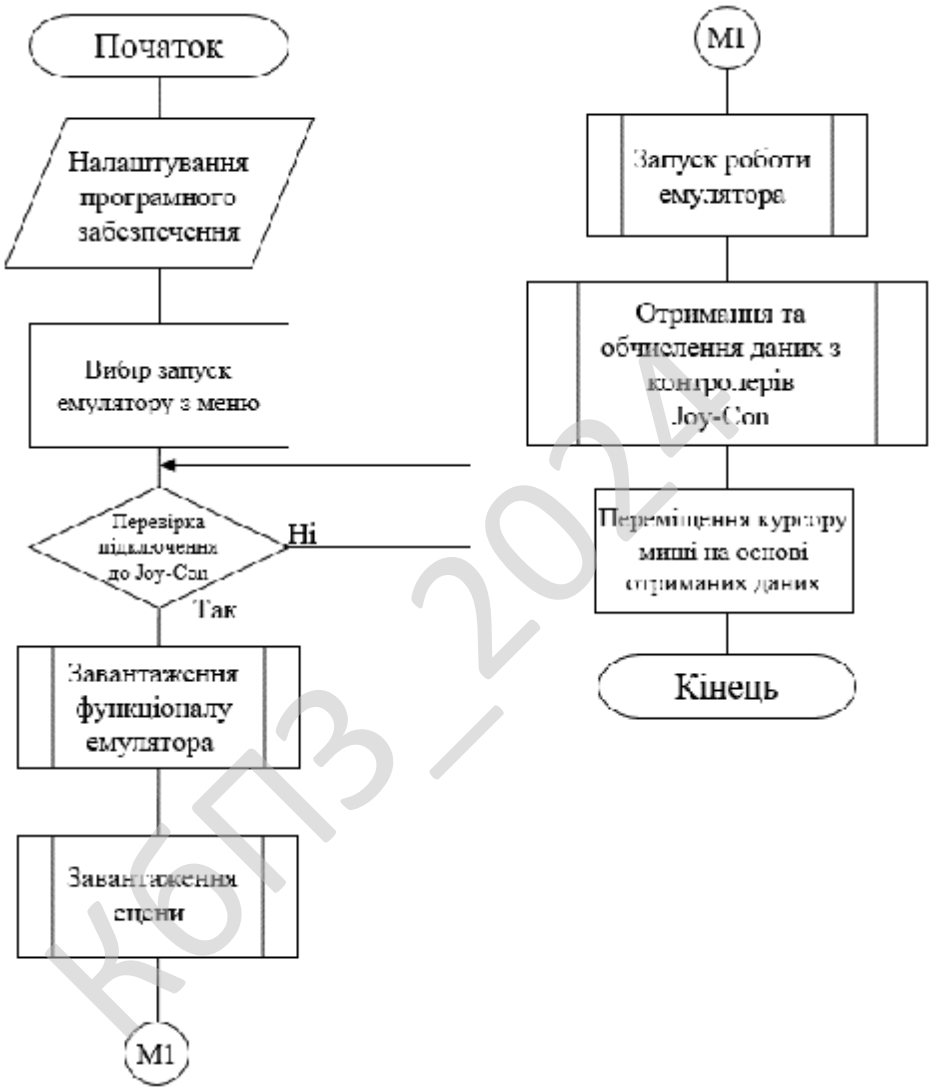

Рисунок 4.2 – Блок-схема роботи підпрограми

Для реалізації поставленної мети було використано фреймворк Unity і мову програмування C#. Для підключення та отримання даних вводу та руху с Joy-Con була використана бібліотека JoyconLib.

JoyconLib створює свій клас Joycon для роботи з контролерами Joy-Con у Unity.

```
using System.Threading;
using UnityEngine;

public class Joycon
{
    public enum DebugType : int
    {
        NONE,
        ALL,
        COMMS,
        THREADING,
        IMU,
        RUMBLE,
    };
    public DebugType debug_type = DebugType.NONE;
    public bool isLeft;
    public enum state_ : uint
    {
        NOT_ATTACHED,
```

Рисунок 4.3 – Оголошення класу Joycon

Бібліотека дозволяє перевіряти підключення контролерів до комп’ютера.

```
if (ptr == IntPtr.Zero)
{
    ptr = HIDapi.hid_enumerate(vendor_id_, 0x0);
    if (ptr == IntPtr.Zero)
    {
        HIDapi.hid_free_enumeration(ptr);
        Debug.Log ("No Joy Cons found!");
        Scene scene = SceneManager.GetActiveScene();
        if(scene.name == "Scene1"){
            TextForNotification.text = "No Joy-Cons found!";
            Button.gameObject.SetActive(true);
            Panel.gameObject.SetActive(true);
            GameObject cylinder = GameObject.Find("Cylinder");
            cylinder.SetActive(false);
        }

    }
}
```

Рисунок 4.4 – Фрагмент коду, що відповідає за перевірку підключення Joy-Con

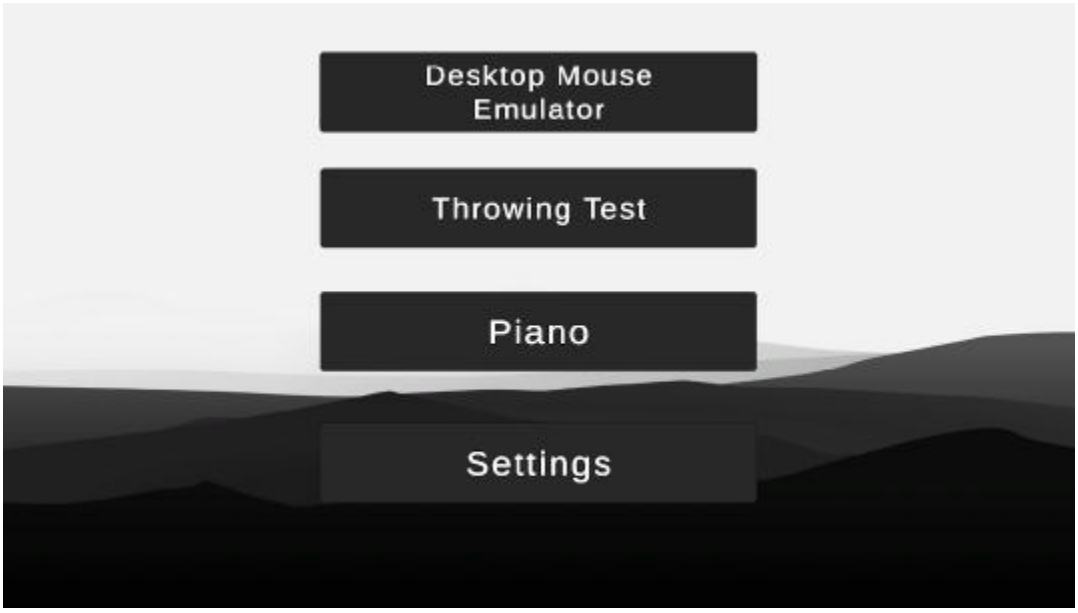

Рисунок 4.5 – Меню застосунку

Завдяки скрипту `TransparentWindow.cs` було можливо добитися «прозорості» застосунку.

```
using System;
using System.Runtime.InteropServices;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class TransparentWindow : MonoBehaviour{

    [DllImport("user32.dll")]
    public static extern int MessageBox(IntPtr hWnd, string text, string caption, uint type);

    [DllImport("user32.dll")]
    public static extern IntPtr GetActiveWindow();

    [DllImport("user32.dll")]
    private static extern int SetWindowLong(IntPtr hWnd, int nIndex, uint dwNewLong);

    [DllImport("user32.dll", SetLastError = true)]
    static extern bool SetWindowPos(IntPtr hWnd, IntPtr hWndInsertAfter, int X, int Y, int cx, int cy, uint uFlags);

    private struct MARGINS{
        public int cxLeftWidth;
        public int cxRightWidth;
        public int cyTopHeight;
        public int cyBottomHeight;
    }

    [DllImport("Dwmapi.dll")]
    private static extern uint DwmExtendFrameIntoClientArea(IntPtr hWnd, ref MARGINS margins);

    const int GWL_EXSTYLE = -20;

    const uint WS_EX_LAYERED = 0x00080000;
    const uint WS_EX_TRANSPARENT = 0x00000020;

    static readonly IntPtr HWND_TOPMOST = new IntPtr(-1);

    private void Start(){

#if !UNITY_EDITOR
        IntPtr hWnd = GetActiveWindow();

        MARGINS margins = new MARGINS {cxLeftWidth = -1};
        DwmExtendFrameIntoClientArea(hWnd, ref margins);

        SetWindowLong(hWnd, GWL_EXSTYLE, WS_EX_LAYERED | WS_EX_TRANSPARENT);

        SetWindowPos(hWnd, HWND_TOPMOST, 0, 0, 0, 0, 0);
#endif
    }


}
```


Рисунок 4.6 - TransparentWindow.cs

Рисунок 4.7 – Вигляд роботи емулятора у режимі Debug

Емуляція натискань кнопки миші була відтворена за допомогою наступної функції у скрипті JoyconController.cs.

```
[DllImport("user32.dll",CharSet=CharSet.Auto, CallingConvention=CallingConvention.StdCall)]
public static extern void mouse_event(uint dwFlags, uint dx, uint dy, uint cButtons, uint dwExtraInfo);
//Mouse actions
private const int MOUSEEVENTF_LEFTDOWN = 0x02;
private const int MOUSEEVENTF_LEFTUP = 0x04;
private const int MOUSEEVENTF_RIGHTDOWN = 0x08;
private const int MOUSEEVENTF_RIGHTUP = 0x10;
```

Рисунок 4.8 – Оголошення використання функції з Windows API у скрипті JoyconController.cs

```
public void DoMouseClick(int button){
    //Call the imported function with the cursor's current position
    uint X = (uint)Input.mousePosition.x;
    uint Y = (uint)Input.mousePosition.y;
    switch(button){
        case 0:
            mouse_event(MOUSEEVENTF_LEFTDOWN | MOUSEEVENTF_LEFTUP, X, Y, 0, 0);
            TextForColors.text = "LMB";
            break;

        case 1:
            mouse_event(MOUSEEVENTF_LEFTDOWN, X, Y, 0, 0);
            TextForColors.text = "LMB_Hold";
            break;

        case 2:
            mouse_event(MOUSEEVENTF_RIGHTDOWN | MOUSEEVENTF_RIGHTUP, X, Y, 0, 0);
            TextForColors.text = "RMB";
            break;

        case 3:
            mouse_event(MOUSEEVENTF_RIGHTDOWN, X, Y, 0, 0);
            TextForColors.text = "RMB_Hold";
            break;
    }

}
```

Рисунок 4.9 – Оголошення функції натискання кнопок миші за допомогою кнопок контролерів Joy-Con у скрипті JoyconController.cs

4.2 Захист розробленого програмного забезпечення

Розробка програмного забезпечення вимагає уваги до питань безпеки та захисту від несанкціонованого доступу і можливих зловмисних дій. У цьому розділі описані основні заходи та підходи, використані для забезпечення захисту розробленого програмного забезпечення.

1. Шифрування даних

Для забезпечення конфіденційності даних, які передаються між контролерами Joy-Con та емулятором комп’ютерної миші, використовується шифрування. Це запобігає перехопленню та розшифруванню інформації

зловмисниками. Всі дані, що зберігаються на локальному пристрої користувача, також шифруються для додаткового захисту.

2. Захист від несанкціонованого доступу

Для запобігання несанкціонованому доступу до налаштувань та функцій емулятора, використовується багаторівнева система перевірки прав доступу. Кожен користувач має певний рівень доступу, який визначає, які дії він може виконувати в програмі.

3. Верифікація та перевірка контролерів

Для забезпечення правильного функціонування та захисту від підробки контролерів Joy-Con, програмне забезпечення перевіряє автентичність підключених контролерів. Це дозволяє виявити спроби використання непідтримуваних або підроблених пристроїв.

4. Тестування на безпеку

Розроблене програмне забезпечення пройшло ретельне тестування на безпеку, включаючи тестування на проникнення та аналіз коду. Це дозволило виявити та усунути потенційні вразливості до випуску продукту.

Отже, впровадження вищезазначених заходів забезпечило високий рівень захисту розробленого програмного забезпечення. Це дозволяє користувачам безпечно та надійно використовувати систему керування емулятором комп’ютерної миші з використанням контролерів Joy-Con.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Для впровадження в експлуатацію застосунку потрібно:

1) Завантажити застосунок.

Рисунок 5.1 – Структура директорії проекту

2) Мати та підключити контролери Joy-Con до комп’ютера через Bluetooth.

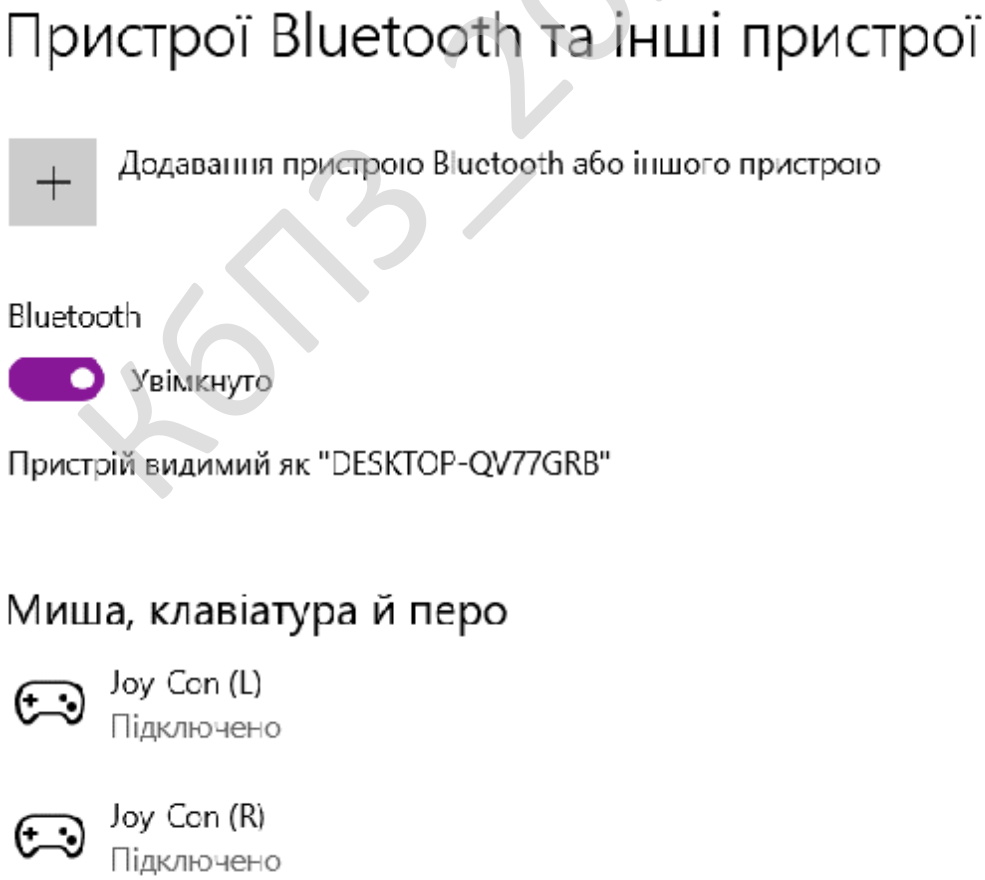

Рисунок 5.2 – Вікно Bluetooth і інших пристроїв

3) Запустити застосунок.

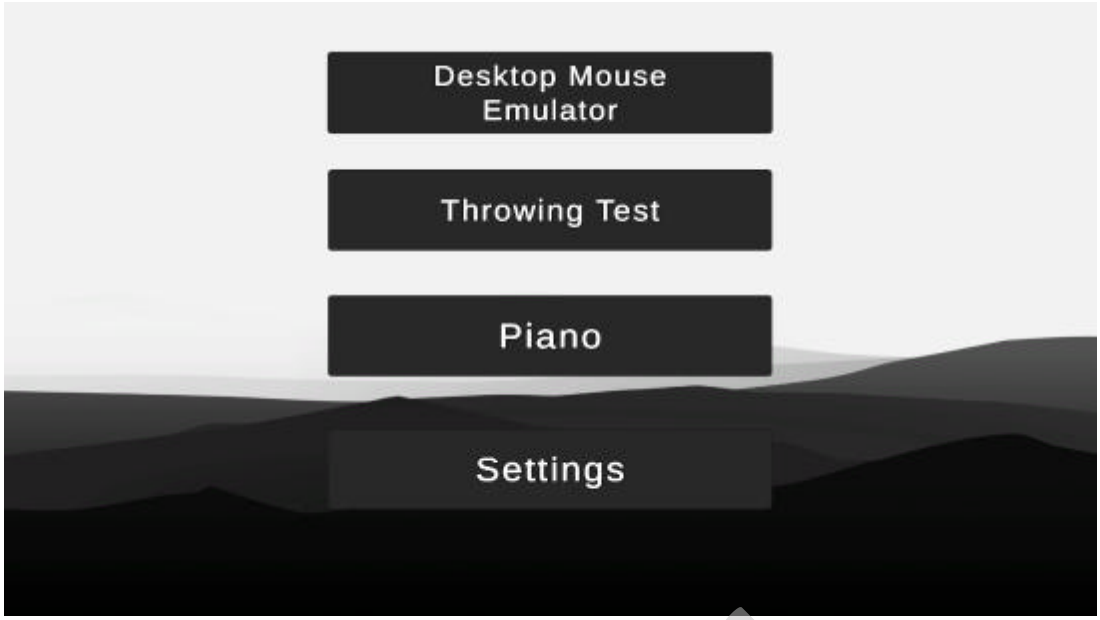

Рисунок 5.3 – Меню застосунку

4) Налаштувати застосунок за потреби.

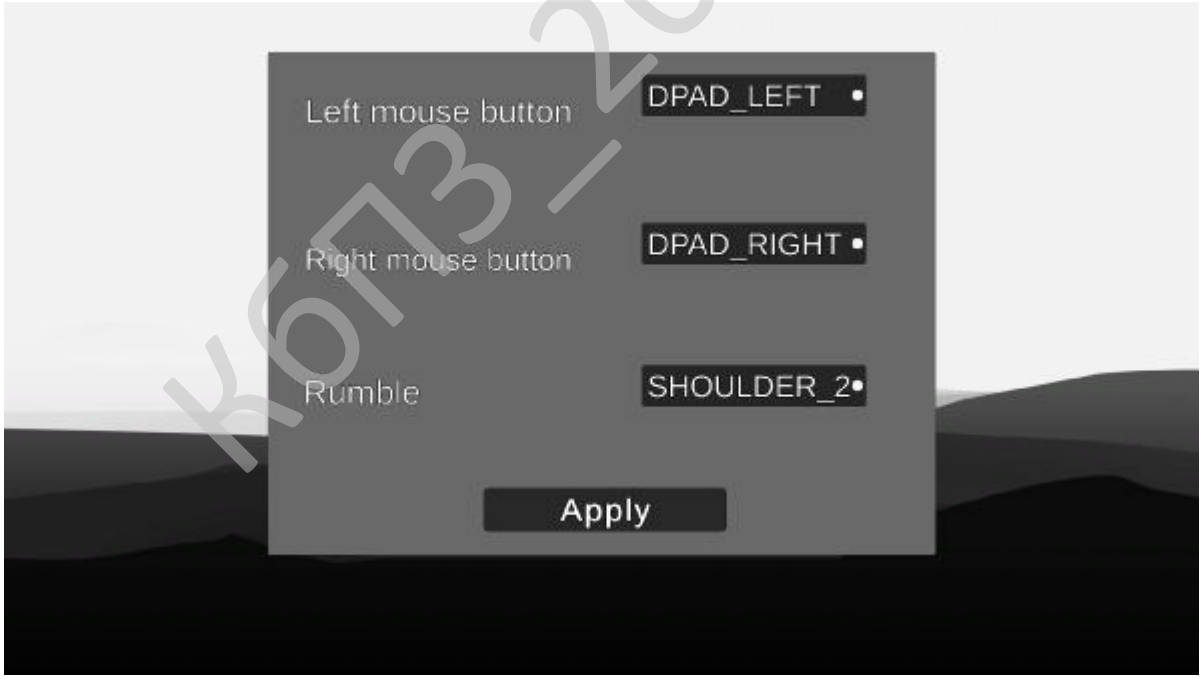

Рисунок 5.4 – Меню налаштувань

5) Застосунок готовий до роботи.

Слідування цим простим інструкціям має бути достатньо для успішного впровадження у промислову експлуатацію.

6 ОСНОВНІ ВИСНОВКИ

Під час реалізації поставленого завдання, я проаналізувала аналоги у рішенні схожої задачі, на яких були побудовані та працюють готові рішення, описані у розділі 2, мною були детально розібрано та вивчено шаблони, проаналізовано використання технологій.

При написанні цієї роботи я успішно розробила застосунок-емулятор на платформі Unity. Реалізація проекту дозволила мені поглибити знання в галузі розробки на Unity, скриптами на C#, роботи з 3D-моделями та ознайомитися більш широко з роботою контролерів Joy-Con, а саме: акселерометр, гіроскоп, «HDRumble» та іншим.

Система, розроблена в рамках цього проекту, призначена для широкого кола користувачів і організацій, що потребують альтернативних методів керування комп'ютерною мишею. Вона буде корисна як людям, які бажають спростити керування комп'ютером, так і людям з обмеженими можливостями, яким традиційні методи можуть бути недоступні. Система поєднує в собі функціональність емулятора комп'ютерної миші та контролерів Joy-Con, забезпечуючи універсальність і простоту використання.

Це рішення спрямоване на полегшення керування комп'ютерною мишею для всіх користувачів, особливо тих, хто має обмежені можливості. Основними завданнями цього проекту є забезпечення зручності, точності та надійності у використанні контролерів Joy-Con як альтернативного пристрою вводу. У процесі дослідження були розглянуті технічні та програмні аспекти розробки, включаючи алгоритми обробки даних з Joy-Con, методи передачі цих даних до комп’ютера та інтеграція з операційною системою.

Реалізація застосунку дозволила мені краще зрозуміти структуру та принципи роботи фреймворку Unity, який спрощує процес розробки застосунків та ігор і підвищує їхню швидкодію.

У результаті дипломної роботи я здобула практичні навички розробки застосунків на Unity та вивчила основні принципи роботи з контролерами Joy-Con.

Ці принципи функціонування програмного забезпечення та шаблони проектування дозволяють організувати гнучкий процес розробки застосунку та будуть використані мною у подальшому.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. JoyconLib [Електронний ресурс]. – Режим доступу: https://github.com/Looking-Glass/JoyconLib

2. JoyConUnity [Електронний ресурс]. – Режим доступу: https://github.com/vksokolov/JoyConUnity

3. Introduction to collision detection in Unity [Електронний ресурс]. – Режим доступу: https://www.educative.io/answers/introduction-to-collision-detection-in-unity

4. Nintendo Switch reverse engineering [Електронний ресурс]. – Режим доступу: https://github.com/dekuNukem/Nintendo_Switch_Reverse_Engineering

5. BetterJoy [Електронний ресурс]. – Режим доступу: https://github.com/Davidobot/BetterJoy

6. Unity Documentation. Mouse support [Електронний ресурс]. – Режим доступу: https://docs.unity3d.com/Packages/com.unity.inputsystem@1.0/manual/Mouse.html

7. Unity Documentation. Cursor [Електронний ресурс]. – Режим доступу: https://docs.unity3d.com/ScriptReference/Cursor.html

8. Unity. Move and send mouse click signals [Електронний ресурс]. – Режим доступу: https://discussions.unity.com/t/how-can-i-control-the-mouse-move-and-send-mouse-click-signals-via-the-keyboard/85297/2

9. Unity Documentation. Mouse support. Cursor warping [Електронний ресурс]. – Режим доступу: https://docs.unity3d.com/Packages/com.unity.inputsystem@1.0/manual/Mouse.html#:~:text=To%20move%20the%20cursor%20to,screen%20coordinates%2C%20just%20like%20Mouse

10. Simulate Mouse Click [Електронний ресурс]. – Режим доступу: https://stackoverflow.com/questions/2416748/how-do-you-simulate-mouse-click-in-c

11. How to simulate a mouse click [Електронний ресурс]. – Режим доступу: https://www.gamedev.net/forums/topic/321029-how-to-simulate-a-mouse-click-in-c/

12. Simulate a mouse [Електронний ресурс]. – Режим доступу: https://discussions.unity.com/t/simulate-a-mouse-click/97796/3

13. Unity Documentation. Class Mouse [Електронний ресурс]. – Режим доступу: https://docs.unity3d.com/Packages/com.unity.inputsystem@1.0/api/UnityEngine.InputSystem.Mouse.html

14. Unity Documentation. GUILayout [Електронний ресурс]. – Режим доступу: https://docs.unity3d.com/ScriptReference/GUILayout.html

15. Joy-Con [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Joy-Con

16. Nintendo Switch [Електронний ресурс]. – Режим доступу: https://www.nintendo.com/us/switch/

17. Inside the Nintendo Switch Joy-Con | Engineering Teardown [Електронний ресурс]. – Режим доступу: https://www.andrews-cooper.com/tech-talks/ac-teardowns-inside-the-nintendo-switch-joy-con/#:~:text=Each%20left%20and%20right%20controller,LRA)%2C%20and%20haptic%20driver.&text=However%2C%20only%20the%20right%20controller,and%20infrared%20(IR)%20camera.

18. Як працює Лінійний Резонансний Привід (LRA)? [Електронний ресурс]. – Режим доступу: https://ru.ineed-motors.com/news/how-does-the-linear-resonant-actuator-lra-wo-44526105.html

19. PC Gaming On Your TV? How to Turn Your Gamepad Into a Computer Mouse [Електронний ресурс]. – Режим доступу: https://www.pcmag.com/how-to/turn-your-game-controller-into-a-computer-mouse

20. Nintendo Switch Teardown [Електронний ресурс]. – Режим доступу: https://www.fictiv.com/teardowns/nintendo-switch-teardown

21. Toggle Game Objects On/Off [Електронний ресурс]. – Режим доступу: https://forum.unity.com/threads/toggle-game-objects-on-off-script.881257/

22. Detect When A Button is Being Pressed & Held On - EventType [Електронний ресурс]. – Режим доступу: https://forum.unity.com/threads/how-do-i-detect-when-a-button-is-being-pressed-held-on-eventtype.352368/

23. Unity Documentation. Button [Електронний ресурс]. – Режим доступу: https://docs.unity3d.com/ru/2018.4/ScriptReference/UI.Button.html

24. Unity Documentation. Input [Електронний ресурс]. – Режим доступу: https://docs.unity3d.com/ScriptReference/Input.html

25. Unity Documentation. KeyCode [Електронний ресурс]. – Режим доступу: https://docs.unity3d.com/ScriptReference/KeyCode.html

26. Unity Documentation. Work with sprites [Електронний ресурс]. – Режим доступу: https://docs.unity3d.com/Manual/Sprites.html

27. Unity. Button On Click - Реагуємо на клік! [Електронний ресурс]. – Режим доступу: https://geekstand.top/development/unity-button-on-click-reagiruem-na-klik/

28. Making a Text file when In-Game [Електронний ресурс]. – Режим доступу: https://forum.unity.com/threads/help-making-a-text-file-when-in-game.848125/

29. Extract just the numbers from a string [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/answers/questions/472735/extract-just-the-numbers-from-a-string

30. File.cs [Електронний ресурс]. – Режим доступу: https://github.com/dotnet/runtime/blob/1d1bf92fcf43aa6981804dc53c5174445069c9e4/src/libraries/System.Private.CoreLib/src/System/IO/File.cs#L706C13-L712C10

31. How to find and replace text in a file [Електронний ресурс]. – Режим доступу: https://stackoverflow.com/questions/13509532/how-to-find-and-replace-text-in-a-file

32. Add, Remove, Replace String In C# [Електронний ресурс]. – Режим доступу: https://www.c-sharpcorner.com/UploadFile/mahesh/add-remove-replace-strings-in-C-Sharp/#:~:text=Interpolation%20in%20C%23.-,Remove%20String%20In%20C%23,first%20character%20in%20the%20string

33. Unity Documentation. Protecting Content [Електронний ресурс]. – Режим доступу: https://docs.unity3d.com/540/Documentation/Manual/protectingcontent.html

34. Unity Blog. Security. [Електронний ресурс]. – Режим доступу: https://blog.unity.com/topic/security

35. Nintendo Switch [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Nintendo_Switch

36. Unity Documentation. Transform.position [Електронний ресурс]. – Режим доступу: https://docs.unity3d.com/ScriptReference/Transform-position.html

37. 3D Object follow mouse [Електронний ресурс]. – Режим доступу: https://forum.unity.com/threads/3d-object-follow-mouse.899123/

38. I made a JOYCON RUMBLE ORCHESTRA with Game Builder Garage! [Електронний ресурс]. – Режим доступу: https://www.youtube.com/watch?v=vbPye12knFs&list=PL3HrepT7jYaD5UzIbCdTofF5RtyIrDwBq&index=36

39. DualSense Haptic Feedback vs. Joy-Con HD Rumble [Електронний ресурс]. – Режим доступу: https://www.youtube.com/watch?v=GDT9MAKjxAk&list=PL3HrepT7jYaD5UzIbCdTofF5RtyIrDwBq&index=23

40. You can now play music through your Joy-Cons [Електронний ресурс]. – Режим доступу: https://www.youtube.com/watch?v=xSPQ3t_6YII&list=PL3HrepT7jYaD5UzIbCdTofF5RtyIrDwBq&index=21

41. Transparent Unity App! (Overlay, Assistant, Particles) [Електронний ресурс]. – Режим доступу: https://www.youtube.com/watch?v=RqgsGaMPZTw&list=PL3HrepT7jYaD5UzIbCdTofF5RtyIrDwBq&index=12

42. Unity C#. Transform Translate [Електронний ресурс]. – Режим доступу: https://www.youtube.com/watch?v=gYRReYDYaZs&list=PL3HrepT7jYaD5UzIbCdTofF5RtyIrDwBq&index=10

43. Creating Transparent App with Unity [Електронний ресурс]. – Режим доступу: https://www.youtube.com/watch?v=d688FLvaq9w&list=PL3HrepT7jYaD5UzIbCdTofF5RtyIrDwBq&index=11

44. Nintendo Switch Joycon Teardown [Електронний ресурс]. – Режим доступу: https://www.youtube.com/watch?v=m63sRY_1eOw&list=PL3HrepT7jYaD5UzIbCdTofF5RtyIrDwBq&index=9

45. 如何把Joy-Con用進Unity [Електронний ресурс]. – Режим доступу: https://www.youtube.com/watch?v=E0mVtqDkrfU&list=PL3HrepT7jYaD5UzIbCdTofF5RtyIrDwBq&index=5

46. Nintendo Switch Joycon Controller Detailed [Електронний ресурс]. – Режим доступу: https://www.youtube.com/watch?v=vnPiHru4IcM&list=PL3HrepT7jYaD5UzIbCdTofF5RtyIrDwBq&index=2

47. Nintendo Switch Joycon Fix and Teardown [Електронний ресурс]. – Режим доступу: https://www.youtube.com/watch?v=lFRRS-nRZuU&list=PL3HrepT7jYaD5UzIbCdTofF5RtyIrDwBq&index=3

48. Unity Documentation. WaitForSeconds Constructor [Електронний ресурс]. – Режим доступу: https://docs.unity3d.com/ScriptReference/WaitForSeconds-ctor.html

49. How to make the script wait/sleep in unity [Електронний ресурс]. – Режим доступу: https://stackoverflow.com/questions/30056471/how-to-make-the-script-wait-sleep-in-a-simple-way-in-unity

50. Unity top-down gyro aim assist [Електронний ресурс]. – Режим доступу: https://www.youtube.com/watch?v=g28VPoKKirc

51. Unity Documentation. Supported Input Devices [Електронний ресурс]. – Режим доступу: https://docs.unity3d.com/Packages/com.unity.inputsystem@1.1/manual/SupportedDevices.html

52. Use Joycon Motion Controls [Електронний ресурс]. – Режим доступу: https://forum.unity.com/threads/use-joycon-motion-controls.1423623/

53. Nintendo Switch Joy-Con support [Електронний ресурс]. – Режим доступу: https://forum.unity.com/threads/nintendo-switch-joy-con-support.805251/

54. Joy-Con support for Windows and macOS [Електронний ресурс]. – Режим доступу: https://forum.unity.com/threads/any-word-on-joy-con-support-for-windows-and-macos.1206673/

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку програмного забезпечення емулятора комп’ютерної миші.

2 Підстава для розробки

Підставою для розробки служить завдання на кваліфікаційну бакалаврську роботу №131-02 від 01.04.2024 року, видане на кафедрі кібербезпеки та програмного забезпечення.

3 Мета та призначення розробки

Метою кваліфікаційної бакалаврської роботи є розробка програмного забезпечення емулятора комп’ютерної миші використовуючи контролери Joy-Con.

4 Джерела розробки

Джерелом цієї кваліфікаційної бакалаврської роботи є стосовна до теми література і існуючі аналоги.

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

– розробка програмної частин системи, а також розробка взаємодії системи з ОС та з користувачем;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– програмне забезпечення емулятора комп’ютерної миші;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинно мати обмежень на версію драйверів та операційної системи.

5.4 Вимоги до архітектури

Компонент, що розробляється повинен використовувати системні засоби та апаратні засоби, що на даному етапі розвитку обчислювальної техніки найбільше поширені.

5.5 Вимоги до надійності

Програмні модулі написані по всім правилам, які стосуються стандартних викликів процедур, функцій, методів і форм, визначених технічною документацією на середовище розробки.

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 19-20 град. по Цельсію;

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на ПЕОМ архітектури IBM PC, працювати в ОС Windows 10/11 і з сумісними з цією платформою пристроями і прикладним програмним забезпеченням.

5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows 10/11.

5.8.1 Обладнання

Комп’ютер Intel Core i5/16 ГБ /1 Tb/ NVIDIA GeForce GTX 1650 4GB або сумісні з ним.

5.8.2 Мова програмування

Програму розроблено на мовах програмування C#.

5.8.3 Вхідні дані

Дані руху, нахилу, швидкості руху з контролерів Joy-Con.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у виді опису структури даних, схем та опису алгоритму, а також текстів вихідних модулів програмного забезпечення згідно ЄСПД.

7 Перелік документів, що розробляються

– Структурна схема системи.	1 аркуш
– Функціональна схема системи.	1 аркуш
– Діаграма процесів.	1 аркуш
– Блок-схема алгоритму роботи програми.	2 аркуші
– Пояснювальна записка.	74 аркуші

8 Етапи розробки

8.1 Збір і обробка інформації по темі кваліфікаційної бакалаврської роботи. Постановка задачі на виконання кваліфікаційної бакалаврської роботи (складання ТЗ).

8.2 Проведення досліджень або експериментальних робіт для уточнення основних положень кваліфікаційної бакалаврської роботи.

8.3 Розробка функціональних схем, блок схем алгоритмів роботи програмного забезпечення.

8.4 Побудова схем взаємодії даних.

8.5 Створення прототипу ПЗ.

8.6 Віднаходження ПЗ, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

9 Порядок контролю та приймання

9.1 Подання кваліфікаційної бакалаврської роботи на попередній захист 19.05.2024 р.

9.2 Подання кваліфікаційної бакалаврської роботи на захист 04.06.2024 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи

за першим (бакалаврським) рівнем вищої освіти

_____ Минайленко Р. М.

Програмне забезпечення системи керування емулятором комп’ютерної миші з використанням мікроконтролерів Joi-Con

Лістинг програми

Код документу 12

Носій: CD/DVD-диск / USB-накопичувач

Загальна кількість аркушів: 22

Літера: РП

Кропивницький – 2024 року

// Joycon.cs – Бібліотека для роботи з Joy-Con

```
using System.Collections;
using System.Collections.Generic;
using System.Runtime.InteropServices;
using System;

using System.Threading;
using UnityEngine;

public class Joycon
{
    public enum DebugType : int
    {
        NONE,
        ALL,
        COMMS,
        THREADING,
        IMU,
        RUMBLE,
    };
    public DebugType debug_type = DebugType.NONE;
    public bool isLeft;
    public enum state_ : uint
    {
        NOT_ATTACHED,
        DROPPED,
        NO_JOYCONS,
        ATTACHED,
        INPUT_MODE_0x30,
        IMU_DATA_OK,
    };
    public state_ state;
    public enum Button : int
    {
        DPAD_DOWN = 0,
        DPAD_RIGHT = 1,
        DPAD_LEFT = 2,
        DPAD_UP = 3,
        SL = 4,
        SR = 5,
        MINUS = 6,
        HOME = 7,
        PLUS = 8,
        CAPTURE = 9,
        STICK = 10,
        SHOULDER_1 = 11,
        SHOULDER_2 = 12
    };
    private bool[] buttons_down = new bool[13];
    private bool[] buttons_up = new bool[13];
    private bool[] buttons = new bool[13];
    private bool[] down_ = new bool[13];

    private float[] stick = { 0, 0 };

    private IntPtr handle;

    byte[] default_buf = { 0x0, 0x1, 0x40, 0x40, 0x0, 0x1, 0x40, 0x40 };

    private byte[] stick_raw = { 0, 0, 0 };
    private UInt16[] stick_cal = { 0, 0, 0, 0, 0, 0 };
    private UInt16 deadzone;
    private UInt16[] stick_precal = { 0, 0 };

    private bool stop_polling = false;
```

```
    private int timestamp;
    private bool first_imu_packet = true;
    private bool imu_enabled = false;
    private Int16[] acc_r = { 0, 0, 0 };
    private Vector3 acc_g;

    private Int16[] gyr_r = { 0, 0, 0 };
    private Int16[] gyr_neutral = { 0, 0, 0 };
    private Vector3 gyr_g;
    private bool do_localize;
    private float filterweight;
    private const uint report_len = 49;
    private struct Report
    {
        byte[] r;
        System.DateTime t;
        public Report(byte[] report, System.DateTime time)
        {
            r = report;
            t = time;
        }
        public System.DateTime GetTime()
        {
            return t;
        }
        public void CopyBuffer(byte[] b)
        {
            for (int i = 0; i < report_len; ++i)
            {
                b[i] = r[i];
            }
        }
    };
    private struct Rumble
    {
        private float h_f, amp, l_f;
        public float t;
        public bool timed_rumble;

        public void set_vals(float low_freq, float high_freq, float amplitude,
int time = 0)
        {
            h_f = high_freq;
            amp = amplitude;
            l_f = low_freq;
            timed_rumble = false;
            t = 0;
            if (time != 0)
            {
                t = time / 1000f;
                timed_rumble = true;
            }
        }
        public Rumble(float low_freq, float high_freq, float amplitude, int time
= 0)
        {
            h_f = high_freq;
            amp = amplitude;
            l_f = low_freq;
            timed_rumble = false;
            t = 0;
            if (time != 0)
            {
                t = time / 1000f;
                timed_rumble = true;
            }
        }
        private float clamp(float x, float min, float max)
        {
```

```
            if (x < min) return min;
            if (x > max) return max;
            return x;
        }
        public byte[] GetData()
        {
            byte[] rumble_data = new byte[8];
            if (amp == 0.0f){
                rumble_data[0] = 0x0;
                rumble_data[1] = 0x1;
                rumble_data[2] = 0x40;
                rumble_data[3] = 0x40;
            }else{
                l_f = clamp(l_f, 40.875885f, 626.286133f);
                amp = clamp(amp, 0.0f, 1.0f);
                h_f = clamp(h_f, 81.75177f, 1252.572266f);
                UInt16 hf = (UInt16)((Mathf.Round(32f * Mathf.Log(h_f * 0.1f,
2)) - 0x60) * 4);
                byte lf = (byte)(Mathf.Round(32f * Mathf.Log(l_f * 0.1f, 2)) -
0x40);
                byte hf_amp;
                if (amp == 0) hf_amp = 0;
                else if (amp < 0.117) hf_amp = (byte)(((Mathf.Log(amp * 1000, 2)
* 32) - 0x60) / (5 - Mathf.Pow(amp, 2)) - 1);
                else if (amp < 0.23) hf_amp = (byte)(((Mathf.Log(amp * 1000, 2)
* 32) - 0x60) - 0x5c);
                else hf_amp = (byte)((((Mathf.Log(amp * 1000, 2) * 32) - 0x60) *
2) - 0xf6);

                UInt16 lf_amp = (UInt16)(Mathf.Round(hf_amp) * .5);
                byte parity = (byte)(lf_amp % 2);
                if (parity > 0)
                {
                    --lf_amp;
                }
                lf_amp = (UInt16)(lf_amp >> 1);
                lf_amp += 0x40;
                if (parity > 0) lf_amp |= 0x8000;
                rumble_data = new byte[8];
                rumble_data[0] = (byte)(hf & 0xff);
                rumble_data[1] = (byte)((hf >> 8) & 0xff);
                rumble_data[2] = lf;
                rumble_data[1] += hf_amp;
                rumble_data[2] += (byte)((lf_amp >> 8) & 0xff);
                rumble_data[3] += (byte)(lf_amp & 0xff);
            }
            for (int i = 0; i < 4; ++i)
            {
                rumble_data[4 + i] = rumble_data[i];
            }
            return rumble_data;
        }
    }
    private Queue<Report> reports = new Queue<Report>();
    private Rumble rumble_obj;

    private byte global_count = 0;
    private string debug_str;

      public Joycon(IntPtr handle_, bool imu, bool localize, float alpha, bool
left)
    {
            handle = handle_;
            imu_enabled = imu;
            do_localize = localize;
            rumble_obj = new Rumble(160, 320, 0);
            filterweight = alpha;
            isLeft = left;
    }
```

```
    public void DebugPrint(String s, DebugType d)
    {
        if (debug_type == DebugType.NONE) return;
        if (d == DebugType.ALL || d == debug_type || debug_type ==
DebugType.ALL)
        {
            Debug.Log(s);
        }
    }
    public bool GetButtonDown(Button b)
    {
        return buttons_down[(int)b];
    }
    public bool GetButton(Button b)
    {
        return buttons[(int)b];
    }
    public bool GetButtonUp(Button b)
    {
        return buttons_up[(int)b];
    }
    public float[] GetStick()
    {
        return stick;
    }
    public Vector3 GetGyro()
    {
        return gyr_g;
    }
    public Vector3 GetAccel()
    {
        return acc_g;
    }
    public Quaternion GetVector()
    {
        Vector3 v1 = new Vector3(j_b.x, i_b.x, k_b.x);
        Vector3 v2 = -(new Vector3(j_b.z, i_b.z, k_b.z));
        if (v2 != Vector3.zero){
                return Quaternion.LookRotation(v1, v2);
        }else{
            return Quaternion.identity;
        }
    }
      public int Attach(byte leds_ = 0x0)
    {
        state = state_.ATTACHED;
        byte[] a = { 0x0 };
        Subcommand(0x3, new byte[] { 0x3f }, 1, false);
        a[0] = 0x1;
        dump_calibration_data();
        a[0] = 0x01;
        Subcommand(0x1, a, 1);
        a[0] = 0x02;
        Subcommand(0x1, a, 1);
        a[0] = 0x03;
        Subcommand(0x1, a, 1);
        a[0] = leds_;
        Subcommand(0x30, a, 1);
        Subcommand(0x40, new byte[] { (imu_enabled ? (byte)0x1 : (byte)0x0) },
1, true);
        Subcommand(0x3, new byte[] { 0x30 }, 1, true);
        Subcommand(0x48, new byte[] { 0x1 }, 1, true);
        DebugPrint("Done with init.", DebugType.COMMS);
        return 0;
    }
    public void SetFilterCoeff(float a)
    {
        filterweight = a;
    }
```

```
    public void Detach()
    {
        stop_polling = true;
        PrintArray(max, format: "Max {0:S}", d: DebugType.IMU);
        PrintArray(sum, format: "Sum {0:S}", d: DebugType.IMU);
        if (state > state_.NO_JOYCONS)
        {
            Subcommand(0x30, new byte[] { 0x0 }, 1);
            Subcommand(0x40, new byte[] { 0x0 }, 1);
            Subcommand(0x48, new byte[] { 0x0 }, 1);
            Subcommand(0x3, new byte[] { 0x3f }, 1);
        }
        if (state > state_.DROPPED)
        {
            HIDapi.hid_close(handle);
        }
        state = state_.NOT_ATTACHED;
    }
    private byte ts_en;
    private byte ts_de;
    private System.DateTime ts_prev;
    private int ReceiveRaw()
    {
        if (handle == IntPtr.Zero) return -2;
        HIDapi.hid_set_nonblocking(handle, 0);
        byte[] raw_buf = new byte[report_len];
        int ret = HIDapi.hid_read(handle, raw_buf, new UIntPtr(report_len));
        if (ret > 0)
        {
            lock (reports)
            {
                reports.Enqueue(new Report(raw_buf, System.DateTime.Now));
            }
            if (ts_en == raw_buf[1])
            {
                DebugPrint(string.Format("Duplicate timestamp enqueued. TS:
{0:X2}", ts_en), DebugType.THREADING);
            }
            ts_en = raw_buf[1];
            DebugPrint(string.Format("Enqueue. Bytes read: {0:D}. Timestamp:
{1:X2}", ret, raw_buf[1]), DebugType.THREADING);
        }
        return ret;
    }
    private Thread PollThreadObj;
    private void Poll()
    {
        int attempts = 0;
        while (!stop_polling & state > state_.NO_JOYCONS)
        {
            SendRumble(rumble_obj.GetData());
            int a = ReceiveRaw();
            a = ReceiveRaw();
            if (a > 0)
            {
                state = state_.IMU_DATA_OK;
                attempts = 0;
            }
            else if (attempts > 1000)
            {
                state = state_.DROPPED;
                DebugPrint("Connection lost. Is the Joy-Con connected?",
DebugType.ALL);
                break;
            }
            else
            {
                DebugPrint("Pause 5ms", DebugType.THREADING);
                Thread.Sleep((Int32)5);
```

```
            }
            ++attempts;
        }
        DebugPrint("End poll loop.", DebugType.THREADING);
    }
    float[] max = { 0, 0, 0 };
    float[] sum = { 0, 0, 0 };
    public void Update()
    {
        if (state > state_.NO_JOYCONS)
        {
            byte[] report_buf = new byte[report_len];
            while (reports.Count > 0)
            {
                Report rep;
                lock (reports)
                {
                    rep = reports.Dequeue();
                    rep.CopyBuffer(report_buf);
                }
                if (imu_enabled)
                {
                    if (do_localize)
                    {
                        ProcessIMU(report_buf);
                    }
                    else
                    {
                        ExtractIMUValues(report_buf, 0);
                    }
                }
                if (ts_de == report_buf[1])
                {
                    DebugPrint(string.Format("Duplicate timestamp dequeued. TS:
{0:X2}", ts_de), DebugType.THREADING);
                }
                ts_de = report_buf[1];
                ts_prev = rep.GetTime();
            }
            ProcessButtonsAndStick(report_buf);
                  if (rumble_obj.timed_rumble) {
                    if (rumble_obj.t < 0) {
                        rumble_obj.set_vals (160, 320, 0, 0);
                    } else {
                        rumble_obj.t -= Time.deltaTime;
                    }
                  }
        }
    }
    private int ProcessButtonsAndStick(byte[] report_buf)
    {
        if (report_buf[0] == 0x00) return -1;

        stick_raw[0] = report_buf[6 + (isLeft ? 0 : 3)];
        stick_raw[1] = report_buf[7 + (isLeft ? 0 : 3)];
        stick_raw[2] = report_buf[8 + (isLeft ? 0 : 3)];

        stick_precal[0] = (UInt16)(stick_raw[0] | ((stick_raw[1] & 0xf) << 8));
        stick_precal[1] = (UInt16)((stick_raw[1] >> 4) | (stick_raw[2] << 4));
        stick = CenterSticks(stick_precal);
        lock (buttons)
        {
            lock (down_)
            {
                for (int i = 0; i < buttons.Length; ++i)
                {
                    down_[i] = buttons[i];
                }
            }
```

```
            buttons[(int)Button.DPAD_DOWN] = (report_buf[3 + (isLeft ? 2 : 0)] &
(isLeft ? 0x01 : 0x04)) != 0;
            buttons[(int)Button.DPAD_RIGHT] = (report_buf[3 + (isLeft ? 2 : 0)]
& (isLeft ? 0x04 : 0x08)) != 0;
            buttons[(int)Button.DPAD_UP] = (report_buf[3 + (isLeft ? 2 : 0)] &
(isLeft ? 0x02 : 0x02)) != 0;
            buttons[(int)Button.DPAD_LEFT] = (report_buf[3 + (isLeft ? 2 : 0)] &
(isLeft ? 0x08 : 0x01)) != 0;
            buttons[(int)Button.HOME] = ((report_buf[4] & 0x10) != 0);
            buttons[(int)Button.MINUS] = ((report_buf[4] & 0x01) != 0);
            buttons[(int)Button.PLUS] = ((report_buf[4] & 0x02) != 0);
            buttons[(int)Button.STICK] = ((report_buf[4] & (isLeft ? 0x08 :
0x04)) != 0);
            buttons[(int)Button.SHOULDER_1] = (report_buf[3 + (isLeft ? 2 : 0)]
& 0x40) != 0;
            buttons[(int)Button.SHOULDER_2] = (report_buf[3 + (isLeft ? 2 : 0)]
& 0x80) != 0;
            buttons[(int)Button.SR] = (report_buf[3 + (isLeft ? 2 : 0)] & 0x10)
!= 0;
            buttons[(int)Button.SL] = (report_buf[3 + (isLeft ? 2 : 0)] & 0x20)
!= 0;
            lock (buttons_up)
            {
                lock (buttons_down)
                {
                    for (int i = 0; i < buttons.Length; ++i)
                    {
                        buttons_up[i] = (down_[i] & !buttons[i]);
                        buttons_down[i] = (!down_[i] & buttons[i]);
                    }
                }
            }
        }
        return 0;
    }
    private void ExtractIMUValues(byte[] report_buf, int n = 0)
    {
        gyr_r[0] = (Int16)(report_buf[19 + n * 12] | ((report_buf[20 + n * 12]
<< 8) & 0xff00));
        gyr_r[1] = (Int16)(report_buf[21 + n * 12] | ((report_buf[22 + n * 12]
<< 8) & 0xff00));
        gyr_r[2] = (Int16)(report_buf[23 + n * 12] | ((report_buf[24 + n * 12]
<< 8) & 0xff00));
        acc_r[0] = (Int16)(report_buf[13 + n * 12] | ((report_buf[14 + n * 12]
<< 8) & 0xff00));
        acc_r[1] = (Int16)(report_buf[15 + n * 12] | ((report_buf[16 + n * 12]
<< 8) & 0xff00));
        acc_r[2] = (Int16)(report_buf[17 + n * 12] | ((report_buf[18 + n * 12]
<< 8) & 0xff00));
        for (int i = 0; i < 3; ++i)
        {
            acc_g[i] = acc_r[i] * 0.00025f;
            gyr_g[i] = (gyr_r[i] - gyr_neutral[i]) * 0.00122187695f;
            if (Math.Abs(acc_g[i]) > Math.Abs(max[i]))
                max[i] = acc_g[i];
        }
    }
      private float err;
      public Vector3 i_b, j_b, k_b, k_acc;
      private Vector3 d_theta;
      private Vector3 i_b_;
      private Vector3 w_a, w_g;
       private Quaternion vec;

      private int ProcessIMU(byte[] report_buf)
      {
        if (!imu_enabled | state < state_.IMU_DATA_OK)
            return -1;
```

```
    if (report_buf[0] != 0x30) return -1;

    int dt = (report_buf[1] - timestamp);
    if (report_buf[1] < timestamp) dt += 0x100;

    for (int n = 0; n < 3; ++n)
    {
        ExtractIMUValues(report_buf, n);

              float dt_sec = 0.005f * dt;
        sum[0] += gyr_g.x * dt_sec;
        sum[1] += gyr_g.y * dt_sec;
        sum[2] += gyr_g.z * dt_sec;

        if (isLeft)
        {
            gyr_g.y *= -1;
            gyr_g.z *= -1;
            acc_g.y *= -1;
            acc_g.z *= -1;
        }

        if (first_imu_packet)
        {
            i_b = new Vector3(1, 0, 0);
            j_b = new Vector3(0, 1, 0);
            k_b = new Vector3(0, 0, 1);
            first_imu_packet = false;
        }
        else
        {
            k_acc = -Vector3.Normalize(acc_g);
            w_a = Vector3.Cross(k_b, k_acc);
            w_g = -gyr_g * dt_sec;
            d_theta = (filterweight * w_a + w_g) / (1f + filterweight);
            k_b += Vector3.Cross(d_theta, k_b);
            i_b += Vector3.Cross(d_theta, i_b);
            j_b += Vector3.Cross(d_theta, j_b);

            err = Vector3.Dot(i_b, j_b) * 0.5f;
            i_b_ = Vector3.Normalize(i_b - err * j_b);
            j_b = Vector3.Normalize(j_b - err * i_b);
            i_b = i_b_;
            k_b = Vector3.Cross(i_b, j_b);
        }
        dt = 1;
    }
    timestamp = report_buf[1] + 2;
    return 0;
}
public void Begin()
{
    if (PollThreadObj == null)
    {
        PollThreadObj = new Thread(new ThreadStart(Poll));
        PollThreadObj.Start();
    }
}
public void Recenter()
{
    first_imu_packet = true;
}
private float[] CenterSticks(UInt16[] vals)
{

    float[] s = { 0, 0 };
    for (uint i = 0; i < 2; ++i)
    {
        float diff = vals[i] - stick_cal[2 + i];
```

```
            if (Math.Abs(diff) < deadzone) vals[i] = 0;
            else if (diff > 0) // if axis is above center
            {
                s[i] = diff / stick_cal[i];
            }
            else
            {
                s[i] = diff / stick_cal[4 + i];
            }
        }
        return s;
    }
    public void SetRumble(float low_freq, float high_freq, float amp, int time =
0)
    {
        if (state <= Joycon.state_.ATTACHED) return;
            if (rumble_obj.timed_rumble == false || rumble_obj.t < 0)
        {
            rumble_obj = new Rumble(low_freq, high_freq, amp, time);
        }
    }
    private void SendRumble(byte[] buf)
    {
        byte[] buf_ = new byte[report_len];
        buf_[0] = 0x10;
        buf_[1] = global_count;
        if (global_count == 0xf) global_count = 0;
        else ++global_count;
        Array.Copy(buf, 0, buf_, 2, 8);
        PrintArray(buf_, DebugType.RUMBLE, format: "Rumble data sent: {0:S}");
        HIDapi.hid_write(handle, buf_, new UIntPtr(report_len));
    }
    private byte[] Subcommand(byte sc, byte[] buf, uint len, bool print = true)
    {
        byte[] buf_ = new byte[report_len];
        byte[] response = new byte[report_len];
        Array.Copy(default_buf, 0, buf_, 2, 8);
        Array.Copy(buf, 0, buf_, 11, len);
        buf_[10] = sc;
        buf_[1] = global_count;
        buf_[0] = 0x1;
        if (global_count == 0xf) global_count = 0;
        else ++global_count;
        if (print) { PrintArray(buf_, DebugType.COMMS, len, 11, "Subcommand 0x"
+ string.Format("{0:X2}", sc) + " sent. Data: 0x{0:S}"); };
        HIDapi.hid_write(handle, buf_, new UIntPtr(len + 11));
        int res = HIDapi.hid_read_timeout(handle, response, new
UIntPtr(report_len), 50);
        if (res < 1) DebugPrint("No response.", DebugType.COMMS);
        else if (print) { PrintArray(response, DebugType.COMMS, report_len - 1,
1, "Response ID 0x" + string.Format("{0:X2}", response[0]) + ". Data: 0x{0:S}");
}
        return response;
    }
    private void dump_calibration_data()
    {
        byte[] buf_ = ReadSPI(0x80, (isLeft ? (byte)0x12 : (byte)0x1d), 9); //
get user calibration data if possible
        bool found = false;
        for (int i = 0; i < 9; ++i)
        {
            if (buf_[i] != 0xff)
            {
                Debug.Log("Using user stick calibration data.");
                found = true;
                break;
            }
        }
        if (!found)
```

```
        {
            Debug.Log("Using factory stick calibration data.");
            buf_ = ReadSPI(0x60, (isLeft ? (byte)0x3d : (byte)0x46), 9);
}
        stick_cal[isLeft ? 0 : 2] = (UInt16)((buf_[1] << 8) & 0xF00 | buf_[0]);
        stick_cal[isLeft ? 1 : 3] = (UInt16)((buf_[2] << 4) | (buf_[1] >> 4));
        stick_cal[isLeft ? 2 : 4] = (UInt16)((buf_[4] << 8) & 0xF00 | buf_[3]);
        stick_cal[isLeft ? 3 : 5] = (UInt16)((buf_[5] << 4) | (buf_[4] >> 4));
        stick_cal[isLeft ? 4 : 0] = (UInt16)((buf_[7] << 8) & 0xF00 | buf_[6]);
        stick_cal[isLeft ? 5 : 1] = (UInt16)((buf_[8] << 4) | (buf_[7] >> 4));

        PrintArray(stick_cal, len: 6, start: 0, format: "Stick calibration data:
{0:S}");

        buf_ = ReadSPI(0x60, (isLeft ? (byte)0x86 : (byte)0x98), 16);
        deadzone = (UInt16)((buf_[4] << 8) & 0xF00 | buf_[3]);

        buf_ = ReadSPI(0x80, 0x34, 10);
        gyr_neutral[0] = (Int16)(buf_[0] | ((buf_[1] << 8) & 0xff00));
        gyr_neutral[1] = (Int16)(buf_[2] | ((buf_[3] << 8) & 0xff00));
        gyr_neutral[2] = (Int16)(buf_[4] | ((buf_[5] << 8) & 0xff00));
        PrintArray(gyr_neutral, len: 3, d: DebugType.IMU, format: "User gyro
neutral position: {0:S}");

        if (gyr_neutral[0] + gyr_neutral[1] + gyr_neutral[2] == -3 ||
Math.Abs(gyr_neutral[0]) > 100 || Math.Abs(gyr_neutral[1]) > 100 ||
Math.Abs(gyr_neutral[2]) > 100)
        {
            buf_ = ReadSPI(0x60, 0x29, 10);
            gyr_neutral[0] = (Int16)(buf_[3] | ((buf_[4] << 8) & 0xff00));
            gyr_neutral[1] = (Int16)(buf_[5] | ((buf_[6] << 8) & 0xff00));
            gyr_neutral[2] = (Int16)(buf_[7] | ((buf_[8] << 8) & 0xff00));
            PrintArray(gyr_neutral, len: 3, d: DebugType.IMU, format: "Factory
gyro neutral position: {0:S}");
        }
    }
    private byte[] ReadSPI(byte addr1, byte addr2, uint len, bool print = false)
    {
        byte[] buf = { addr2, addr1, 0x00, 0x00, (byte)len };
        byte[] read_buf = new byte[len];
        byte[] buf_ = new byte[len + 20];

        for (int i = 0; i < 100; ++i)
        {
            buf_ = Subcommand(0x10, buf, 5, false);
            if (buf_[15] == addr2 && buf_[16] == addr1)
            {
                break;
            }
        }
        Array.Copy(buf_, 20, read_buf, 0, len);
        if (print) PrintArray(read_buf, DebugType.COMMS, len);
        return read_buf;
    }
    private void PrintArray<T>(T[] arr, DebugType d = DebugType.NONE, uint len =
0, uint start = 0, string format = "{0:S}")
    {
        if (d != debug_type && debug_type != DebugType.ALL) return;
        if (len == 0) len = (uint)arr.Length;
        string tostr = "";
        for (int i = 0; i < len; ++i)
        {
            tostr += string.Format((arr[0] is byte) ? "{0:X2} " : ((arr[0] is
float) ? "{0:F} " : "{0:D} "), arr[i + start]);
        }
        DebugPrint(string.Format(format, tostr), d);
    }
}
```

// JoyconManager.cs – Файл менеджу підключених Joy-Con

```
using System.Collections;
using System.Collections.Generic;
using System.Runtime.InteropServices;
using UnityEngine.UI;
using TMPro;
using UnityEngine;
using System;
using UnityEngine.InputSystem;
using UnityEngine.SceneManagement;
public class JoyconManager: MonoBehaviour
{
    public enum JoyconType { left, right };

    public bool EnableIMU = true;
    public bool EnableLocalize = true;
    public TMP_Text TextForNotification;
    public Button Button;
    public GameObject Panel;

    private const ushort vendor_id = 0x57e;
    private const ushort vendor_id_ = 0x057e;
    private const ushort product_l = 0x2006;
    private const ushort product_r = 0x2007;

    public List<Joycon> j;
    static JoyconManager instance;
    Joycon leftJoycon, rightJoycon;

    public static JoyconManager Instance
    {
        get { return instance; }
    }

    public Joycon GetJoycon(JoyconType type){
        if (type == JoyconType.right && rightJoycon != null)
            return rightJoycon;

        if (type == JoyconType.left && leftJoycon != null)
            return leftJoycon;

        return null;
    }

    void Awake()
    {
        if (instance != null) Destroy(gameObject);
        instance = this;
            int i = 0;

            j = new List<Joycon>();
            bool isLeft = false;
            HIDapi.hid_init();

            IntPtr ptr = HIDapi.hid_enumerate(vendor_id, 0x0);
            IntPtr top_ptr = ptr;


            if (ptr == IntPtr.Zero)
            {
                  ptr = HIDapi.hid_enumerate(vendor_id_, 0x0);
                  if (ptr == IntPtr.Zero)
                  {
                        HIDapi.hid_free_enumeration(ptr);
                        Debug.Log ("No Joy-Cons found!");
                Scene scene = SceneManager.GetActiveScene();
                if(scene.name == "Scene1"){
                    TextForNotification.text = "No Joy-Cons found!";
```

```
                    Button.gameObject.SetActive(true);
                    Panel.gameObject.SetActive(true);
                    GameObject cylinder = GameObject.Find("Cylinder");
                    cylinder.SetActive(false);
                }

                  }
            }
            hid_device_info enumerate;
            while (ptr != IntPtr.Zero) {
                  enumerate = (hid_device_info)Marshal.PtrToStructure (ptr,
typeof(hid_device_info));

                        if (enumerate.product_id == product_l ||
enumerate.product_id == product_r) {
                              if (enumerate.product_id == product_l) {
                                    isLeft = true;
                                    Debug.Log ("Left Joy-Con connected.");
                              } else if (enumerate.product_id == product_r) {
                                    isLeft = false;
                                    Debug.Log ("Right Joy-Con connected.");
                              } else {
                                    Debug.Log ("Non Joy-Con input device
skipped.");
                              }
                              IntPtr handle = HIDapi.hid_open_path
(enumerate.path);
                              HIDapi.hid_set_nonblocking (handle, 1);
                Joycon joycon = new Joycon(handle, EnableIMU, EnableLocalize &
EnableIMU, 0.05f, isLeft);
                j.Add (joycon);

                if (isLeft)
                    leftJoycon = joycon;
                else
                    rightJoycon = joycon;

                              ++i;
                        }
                        ptr = enumerate.next;
                  }
            HIDapi.hid_free_enumeration (top_ptr);
    }
    void Start()
    {
            for (int i = 0; i < j.Count; ++i)
            {
                  Joycon jc = j [i];
                  byte LEDs = 0x0;
                  LEDs |= (byte)(0x1 << i);
                  jc.Attach (leds_: LEDs);
                  jc.Begin ();
            }
    }
    void Update()
    {
            for (int i = 0; i < j.Count; ++i)
            {
                  j[i].Update();
            }
    }
    void OnApplicationQuit()
    {
            for (int i = 0; i < j.Count; ++i)
            {
                  j[i].Detach ();
            }
    }
}
```

// JoyconController.cs – Файл налаштування та функції для контролерів Joy-Con

```
using System.Collections;
using System.Collections.Generic;
using System;
using System.Runtime.InteropServices;
using UnityEngine;
using UnityEngine.UI;
using TMPro;
using UnityEngine.SceneManagement;
using UnityEngine.InputSystem;

public class JoyconController : MonoBehaviour{

    public JoyconManager.JoyconType joyconType;
    public float[] stick;
    public Vector3 gyro = Vector3.zero;
    public float gyroMagnitude;
    public Vector3 accel = Vector3.zero;
    public float accelMagnitude;
    public Quaternion orientation;
    public Vector3 rotation;
    public Joycon joycon;
    public TMP_Text TextForColors;
    public TMP_Text TextForZamah;
    public AudioClip defaultSound;
    public AudioSource audioSource;
    public int lowFrequencyRumble, highFrequencyRumble;
    Vector3 rotationOffset = new Vector3(0, 180, 0);
    Vector3 mousePosition;
    private bool HelpGuiOn = false;
    private bool FreeMouse = true;
    private Scene scene;
    public int octava = 8;

    [DllImport("user32.dll",CharSet=CharSet.Auto,
CallingConvention=CallingConvention.StdCall)]
    public static extern void mouse_event(uint dwFlags, uint dx, uint dy, uint
cButtons, uint dwExtraInfo);

    private const int MOUSEEVENTF_LEFTDOWN = 0x02;
    private const int MOUSEEVENTF_LEFTUP = 0x04;
    private const int MOUSEEVENTF_RIGHTDOWN = 0x08;
    private const int MOUSEEVENTF_RIGHTUP = 0x10;

    JoyconManager joyconManager;
    List<Joycon> joycons;

    void Start (){
        joyconManager = JoyconManager.Instance;
        joycon = joyconManager.GetJoycon(joyconType);

        joycons = JoyconManager.Instance.j;

        if (joycon == null){
            Destroy(gameObject);
        }
        Mouse.current.WarpCursorPosition(new Vector2(900,500));
        scene = SceneManager.GetActiveScene();
      }

        void Update (){
        if (joycon != null){
            if(scene.name == "Scene1"){

                if (joycon.GetButtonDown(Joycon.Button.SHOULDER_2)){
                        Debug.Log ("Shoulder button 2 pressed");
```

```
                        Debug.Log(string.Format("Stick x: {0:N} Stick y:
{1:N}",joycon.GetStick()[0],joycon.GetStick()[1]));

                        joycon.Recenter ();
                }

                if (joycon.GetButtonUp (Joycon.Button.SHOULDER_1)){
                        joycon.SetRumble (0, 0, 0);
                        Debug.Log ("Shoulder button 1 released");
                }

                if (joycon.GetButton (Joycon.Button.SHOULDER_1)){
                        joycon.SetRumble (160, 320, 0.6f);
                        Debug.Log ("Shoulder button 1 held");
                }

                if (joycon.GetButton(Joycon.Button.DPAD_DOWN)){
                        Debug.Log ("Rumble");

                  StartCoroutine(HDRumble());

                }

                if (joycon.GetButtonDown (Joycon.Button.STICK)){
                        audioSource.PlayOneShot(defaultSound);
                }

                if(joycon.GetButtonUp(Joycon.Button.DPAD_RIGHT)){
                  DoMouseClick(2);
                }
                if(joycon.GetButton(Joycon.Button.DPAD_RIGHT)){
                  DoMouseClick(3);
                }

                if(joycon.GetButtonUp(Joycon.Button.DPAD_LEFT)){
                  DoMouseClick(0);
                }
                if(joycon.GetButton(Joycon.Button.DPAD_LEFT)){
                  DoMouseClick(1);
                }

                bool Rotation = false;

                if(lowFrequencyRumble!=0 && highFrequencyRumble!=0){
                        if (joycon.GetButtonDown (Joycon.Button.SR)){
                                Rotation = true;
                                Debug.Log (string.Format("SR On {0:N}",Rotation));
                         joycon.SetRumble (lowFrequencyRumble,
highFrequencyRumble, 0.6f, 200);

                        }
                        if (joycon.GetButtonUp (Joycon.Button.SR)){
                                Rotation = false;
                                Debug.Log (string.Format("SR Off
{0:N}",Rotation));
                         joycon.SetRumble (0, 0, 0);
                        }
                }

                  stick = joycon.GetStick();

                  float moveSpeed = 10;
                  transform.Translate(new Vector3(stick[0] * -1, 0, stick[1]) *
moveSpeed * Time.deltaTime);


                  switch(joycon.isLeft){
                        case true:
                        if (joycon.GetButton(Joycon.Button.DPAD_UP)){
```

```
                              gameObject.GetComponent<Renderer>().material.color
= Color.magenta;
                              TextForColors.text = "Pressed DPad Up button";
                        }else if(joycon.GetButton(Joycon.Button.DPAD_DOWN)){
                              gameObject.GetComponent<Renderer>().material.color
= Color.cyan;
                              TextForColors.text = "Pressed DPad Down button";
                        }else{
                              gameObject.GetComponent<Renderer>().material.color
= Color.blue;
                        }
                        break;

                        case false:
                        if (joycon.GetButton(Joycon.Button.DPAD_UP)){
                              gameObject.GetComponent<Renderer>().material.color
= Color.magenta;
                              TextForColors.text = "Pressed DPad Up button";
                        }else if(joycon.GetButton(Joycon.Button.DPAD_DOWN)){
                              gameObject.GetComponent<Renderer>().material.color
= Color.cyan;
                              TextForColors.text = "Pressed DPad Down button";
                        }else{
                              gameObject.GetComponent<Renderer>().material.color
= Color.red;
                        }
                        break;
                  }

                  if (joycon.GetButton(Joycon.Button.MINUS))transform.position =
new Vector3(7, 0, 5);
                  if (joycon.GetButton(Joycon.Button.PLUS))transform.position =
new Vector3(4, 0, 5);

                  switch (gameObject.transform.rotation.x){
                        case > 0:
                              TextForZamah.text = "ZAMAH";
                              break;

                        case < 0:
                              TextForZamah.text = "STOIM NA MESTE";
                              break;
                  }

                  gyro = joycon.GetGyro();
                  gyroMagnitude = gyro.magnitude;


                 transform.Translate(new Vector3(-gyro.z, 0, gyro.y) *
gyroMagnitude * Time.deltaTime, Space.World);


                  accel = joycon.GetAccel();
                  accelMagnitude = accel.magnitude;


                  if(joycon.isLeft == true && !FreeMouse){
                      Mouse.current.WarpCursorPosition(new
Vector2(transform.position.x * -10, transform.position.z * -10));
                      transform.Translate(new Vector3(-gyro.z/100, 0, gyro.y/100)
* gyroMagnitude/10 * Time.deltaTime, Space.World);

                  }

                  if (joycon.GetButton(Joycon.Button.DPAD_UP)){
                      if(FreeMouse){
                          FreeMouse = !FreeMouse;
                      }else{
                          FreeMouse = true;
```

```
                    }
                }

                if(joycon.isLeft == true)mousePosition = Input.mousePosition;
if(joycon.isLeft == true && FreeMouse)transform.position =
Camera.main.ScreenToWorldPoint(new Vector3(Input.mousePosition.x,
Input.mousePosition.y, Camera.main.transform.position.z));

                  orientation = joycon.GetVector();
                  gameObject.transform.rotation = orientation;
                  orientation = new Quaternion(orientation.x, orientation.z,
orientation.y, orientation.w);
                  gameObject.transform.localRotation =
Quaternion.Inverse(orientation);
            }else if(scene.name == "Piano_1"){
                if (joycon.GetButtonDown(Joycon.Button.SHOULDER_1)){
                        Debug.Log ("Not in Scene1, in scene Piano_1)");
                }
            }
        }
    }

      protected void OnGUI(){
            if(Input.GetKey(KeyCode.Keypad0)&&scene.name == "Scene1") HelpGuiOn
= true;
            if(HelpGuiOn){
                  GUI.skin.label.fontSize = Screen.width / 40;

                  GUILayout.Label("Orientation: " + Screen.orientation);
                  GUILayout.Label("Left Joy-Con:");
                  if(joycon.isLeft ==
      true){GUILayout.Label("input.gyro.attitude" +
      joycon.GetGyro());}else{GUILayout.Label("");}
                  GUILayout.Label("Right Joy-Con:");
                  if(joycon.isLeft == false)GUILayout.Label("input.gyro.attitude
      Right: " + joycon.GetGyro());else{GUILayout.Label("");}
                  GUILayout.Label("Screen width/font: " + Screen.width + " : " +
      GUI.skin.label.fontSize);
                  if(joycon.isLeft == true)GUILayout.Label("Mouse position:" +
      mousePosition);else{GUILayout.Label("");}
                  if(joycon.isLeft == true)GUILayout.Label("Hand position:" +
      transform.position);else{GUILayout.Label("");}
             }

    }

      private void OnStickPressed(){
            if (defaultSound != null && !audioSource.isPlaying)
            {
                  audioSource.PlayOneShot(defaultSound);
            }
      }

      public void DoMouseClick(int button){
      uint X = (uint)Input.mousePosition.x;
      uint Y = (uint)Input.mousePosition.y;
      switch(button){
            case 0:
                mouse_event(MOUSEEVENTF_LEFTDOWN | MOUSEEVENTF_LEFTUP, X, Y, 0,
0);
                TextForColors.text = "LMB";
                break;

            case 1:
                mouse_event(MOUSEEVENTF_LEFTDOWN, X, Y, 0, 0);
                TextForColors.text = "LMB_Hold";
                break;

            case 2:
```

```
                mouse_event(MOUSEEVENTF_RIGHTDOWN | MOUSEEVENTF_RIGHTUP, X, Y,
0, 0);
                TextForColors.text = "RMB";
                break;

            case 3:
                mouse_event(MOUSEEVENTF_RIGHTDOWN, X, Y, 0, 0);
                TextForColors.text = "RMB_Hold";
                break;
        }
    }

    IEnumerator HDRumble(){
        for(int i=0;i<4;i++){
            joycon.SetRumble (0, 392, 0.6f, 1350);
            yield return new WaitForSeconds(1.35f);
            joycon.SetRumble (0, 293.66f, 0.6f, 3350);
            yield return new WaitForSeconds(3.35f);
            joycon.SetRumble (0, 329.63f, 0.6f, 1000);
            yield return new WaitForSeconds(1);
            joycon.SetRumble (0, 261.63f, 0.6f, 1000);
            yield return new WaitForSeconds(1);
        }
    }

    public void CreateRumble(){
        if(scene.name == "Piano_1"){
            Button[] buttons = FindObjectsOfType<Button>();
            foreach (Button button in buttons){
                Debug.Log(button.name);
                button.onClick.AddListener(() => SetRumblePiano(button));
            }

        }
    }

    void SetRumblePiano(Button button){
        Debug.Log("Натиснута кнопка: " + button.name);
        switch(button.name){
            case "ButtonA":
                joycon.SetRumble (16.35f*octava, 0, 0.6f, 200);
            break;
            case "ButtonS":
                joycon.SetRumble (18.35f*octava, 0, 0.6f, 200);
            break;
            case "ButtonD":
                joycon.SetRumble (20.60f*octava, 0, 0.6f, 200);
            break;
            case "ButtonF":
                joycon.SetRumble (21.83f*octava, 0, 0.6f, 200);
            break;
            case "ButtonJ":
                joycon.SetRumble (24.50f*octava, 0, 0.6f, 200);
            break;
            case "ButtonK":
                joycon.SetRumble (27.50f*octava, 0, 0.6f, 200);
            break;
            case "ButtonL":
                joycon.SetRumble (30.87f*octava, 0, 0.6f, 200);
            break;
        }
    }

}
```

// LevelManager.cs – Контролер переключення сцен

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

public class LevelManager : MonoBehaviour
{
    void Start(){
        string path = Application.dataPath + "/Button Settings.txt";
        if (!File.Exists(path)) {
            File.WriteAllText(path, "LMB: " + "2" + "\n" + "RMB: " + "1" + "\n"
+ "HDRumble: " + "12" + "\n");
        }else{}
    }
    void Update(){

    }
    public void ChangeScene(string SceneName){
        SceneManager.LoadScene(SceneName);
    }
}
```

// TransparentWindow.cs – Файл конфігурації прозорості вікна застосунка під час емулятора

```
using System;
using System.Runtime.InteropServices;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class TransparentWindow : MonoBehaviour{

    [DllImport("user32.dll")]
    public static extern int MessageBox(IntPtr hWnd, string text, string
caption, uint type);

    [DllImport("user32.dll")]
    public static extern IntPtr GetActiveWindow();

    [DllImport("user32.dll")]
    private static extern int SetWindowLong(IntPtr hWnd, int nIndex, uint
dwNewLong);

    [DllImport("user32.dll", SetLastError = true)]
    static extern bool SetWindowPos(IntPtr hWnd, IntPtr hWndInsertAfter, int X,
int Y, int cx, int cy, uint uFlags);

    private struct MARGINS{
    public int cxLeftWidth;
    public int cxRightWidth;
    public int cyTopHeight;
    public int cyBottomHeight;
    }

    [DllImport("Dwmapi.dll")]
    private static extern uint DwmExtendFrameIntoClientArea(IntPtr hWnd, ref
MARGINS margins);

    const int GWL_EXSTYLE = -20;

    const uint WS_EX_LAYERED = 0x00080000;
    const uint WS_EX_TRANSPARENT = 0x00000020;

    static readonly IntPtr HWND_TOPMOST = new IntPtr(-1);
    private void Start(){
#if !UNITY_EDITOR
        IntPtr hWnd = GetActiveWindow();
        MARGINS margins = new MARGINS {cxLeftWidth = -1};
        DwmExtendFrameIntoClientArea(hWnd, ref margins);
        SetWindowLong(hWnd, GWL_EXSTYLE, WS_EX_LAYERED | WS_EX_TRANSPARENT);
        SetWindowPos(hWnd, HWND_TOPMOST, 0, 0, 0, 0, 0);
#endif
    }
}
```

// CreateSettingFile.cs – Файл налаштування для емулятора

```
using System.Collections;
using System.Linq;
using System.Collections.Generic;
using UnityEngine;
using System.IO;
using UnityEditor;
using UnityEngine.UI;
using TMPro;

public class CreateSettingFile : MonoBehaviour{

    [SerializeField] private TMP_Dropdown dropdown;
    public int value = 0;
    private string path;

    public void GetDropdownValue(){
    int pickedEntryIndex = dropdown.value;
    Debug.Log(pickedEntryIndex);
    value = pickedEntryIndex;
    }

    public void CreateText(){
        switch (dropdown.name){
            case "LMBDropdown":
                path = Application.dataPath + "/Button Settings.txt";
                if (!File.Exists(path)) {
                    File.WriteAllText(path, "LMB: " + value + "\n");
                }else{
                    foreach (string line in File.ReadLines(path)){
                        bool containsInt = line.Any(char.IsDigit);
                        string output = string.Concat(line.Where(char.IsDigit));
                        if (line.Contains("LMB") && containsInt){
                            string replace = line.Replace(output,
value.ToString());
                            Debug.Log(string.Format("Replace text above with
{0:N}", replace));
                        }else{
                            Debug.Log(":(");
                        }
                    }
                }
            break;

            case "RMBDropdown":
                path = Application.dataPath + "/Button Settings.txt";
                if (!File.Exists(path)) {
                    File.AppendAllText(path, "RMB: " + value + "\n");
                }else{
                    string content = "RMB: " + value + "\n";
                    File.AppendAllText(path, content);
                }
            break;

            case "HDRumbleDropdown":
                path = Application.dataPath + "/Button Settings.txt";
                if (!File.Exists(path)) {
                    File.AppendAllText(path, "HDRumble: " + value + "\n");
                }else{
                    string content = "HDRumble: " + value + "\n";
                    File.AppendAllText(path, content);
                }
            break;
        }
    }

    public void ReadFile(){
        path = Application.dataPath + "/Button Settings.txt";
```

```
        if (File.Exists(path)){
            foreach (string line in File.ReadLines(path)){
                bool containsInt = line.Any(char.IsDigit);
                string output = string.Concat(line.Where(char.IsDigit));
                if (line.Contains("LMB") && containsInt){
                    Debug.Log(output);
                }else if (line.Contains("RMB") && containsInt){
                    Debug.Log(output);
                }else if (line.Contains("HDRumble") && containsInt){
                    Debug.Log(output);
                }else{
                    Debug.Log(":(");
                }
            }
        }
    }
}
```