

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Пензенский государственный университет»

Политехнический институт
(наименование института полностью)

Кафедра «Системы автоматизированного проектирования»
(наименование)

09.03.01 «Информатика и вычислительная техника»
(код и наименование направления подготовки, специальности)

«Системы автоматизированного проектирования»
(направленность (профиль) / специализация)

КУРСОВАЯ РАБОТА **по курсу «Технология разработки программного** **обеспечения»**

на тему «Разработка программного обеспечения для лазертаг клубов»

Обучающийся

Н.А. Горбунов
(Инициалы Фамилия)

(личная подпись)

Руководитель

к.т.н, доцент каф. САПР В.В. Эпп

(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Пенза 2023

Оглавление

Введение.....	3
1 Анализ предметной области	5
1.1 Протокол обмена данными между устройствами и программным обеспечением.....	6
1.2 Протокол обмена данными между программным обеспечением и сервером	10
1.4 Создание и анализ технического задания и способов решения.....	15
2 Разработка.....	19
2.1 Создание проекта и сцены под Desktop.....	19
2.3 Разработка многопоточных методов и очереди	22
2.4 Отправка данных по COMPORT и BLE	28
2.4 разработка классов взаимодействия с сервером.....	33
2.5 Отладка и тестирование приложений	35
Заключение	44
Список литературы	45
Приложения	46
Приложение 1 Commader.cs.....	46
Приложение 2 Translator.cs	53
Приложение 3 WebManager.cs.....	61
Приложение 4 WebSender.cs.....	65

Введение

Развлечения всегда были неотъемлемой частью жизни человека. По этой причине следует разобрать одну из самых передовых и быстро развивающихся технологий в области развлечений – «лазертаг». Лазертаг – это игра, требующая физической активности, командной работы и стратегического мышления. Игроки соревнуются друг с другом с помощью специального оружия, стреляющего вместо пули инфракрасными лучами. Лазертаг оборудование приобретают владельцы специализированных арен, которые ограничены пропускной способностью света для наиболее точной работы сигнала передачи через инфракрасный луч, на основе которого и строится всё взаимодействие между игроками. Время не стоит на месте и игрокам хочется испытать больше ощущений от игры, поэтому производители лазертаг оборудования стремятся добавлять новый функционал и оптимизировать уже существующие возможности. Но к каждому клиенту нужен свой подход, поэтому, чтобы не создавать каждое устройство подстраиваясь к определенному игроку, было решено дать возможность владельцам клубов самостоятельно настраивать оборудование в любой момент. Одним из наиболее удобных вариантов настройки оборудования является передача данных с помощью программного обеспечения, так как персональный компьютер или смартфон, на данный момент есть почти у каждого. Программное обеспечение для лазертага позволяет настроить устройства в соответствии с пожеланиями игроков, например можно изменить определенные параметры, например здоровье, броню, урон и скорость перезарядки, а также правила и сценарии игры. Исходя из этого можно понять что программное обеспечение является неотъемлемой частью любой игры в лазертаг. Оно позволяет сделать игру более увлекательной, разнообразной и динамичной.

Целью курсовой работы является разработка программного обеспечения, для настройки всех типов устройств специального лазертаг оборудования, учёта

статистики, автоматизации игровых событий и взаимодействия с сервером для сохранения характеристик игроков. Для достижения цели следовало выполнить следующие задачи:

1. Проанализировать предметную область
2. Изучить протокол передачи данных на устройства
3. Изучить протокол передачи данных на сервер
4. Сформулировать техническое задание
5. Реализовать два приложения под разные платформы
6. Реализовать конфигуратор оборудования и передачу данных по COMPORT/BLE
7. Реализовать меню взаимодействия с сервером на стороне клиента.

Для реализации программного обеспечения был язык программирования C# и игровой движок Unity, так как он поддерживает разработку под разные платформы и имеет удобный интерфейс.

Курсовая работа содержит 44 листа, 42 рисунка, 18 таблиц, 5 источников литературы.

1 Анализ предметной области

В конце 70-х годов была представлена одна из первых версий взаимодействия устройств по ИК сигналу. Для обмена данными был придуман устоявшийся протокол передачи – «Miles tag», который дошел до современного вида лазертага. По этому протоколу построены все правила передачи информации между устройствами. Устройствами являются муляжи оружия, повязки на голову и дополнительное оборудование для разнообразия сценариев игры. Муляж оружия – «тагер», умеет излучать ИК сигнал для имитации выстрела, является slave устройством для повязки. Повязка на голову или жилет имеют несколько модулей для приёма ИК сигнала, которые оснащены световыми индикаторами, является Master устройством для тагера. Радиобаза – устройство для передачи информации между игроками и компьютером со специальным программным обеспечением. Программное обеспечение является способом взаимодействия администратора лазертаг клуба с оборудованием для настройки игровых параметров и ведения статистики игры. Существует множество способов подключения радиобазы для дальнейшей передачи информации, в разработанных приложениях было реализовано два: соединение с радиобазой по COMPORT и Bluetooth, для компьютеров и телефонов соответственно. Протокол команд для соединения и передачи данных идентичен в обоих случаях, отличаются лишь физические способы отправки.

На рисунке 1, представлена абстрактная схема взаимодействия всех устройств. Радиобаза пересылает данные на частоте 2.4 ГГц для Master устройств. Повязки и жилеты связываются с тагерами на частоте 13,56 МГц, после чего начинают общаться между собой на частоте 2.4 ГГц. Тагер излучает инфракрасный луч, передавая по нему данные о выстреле для других Master устройств.

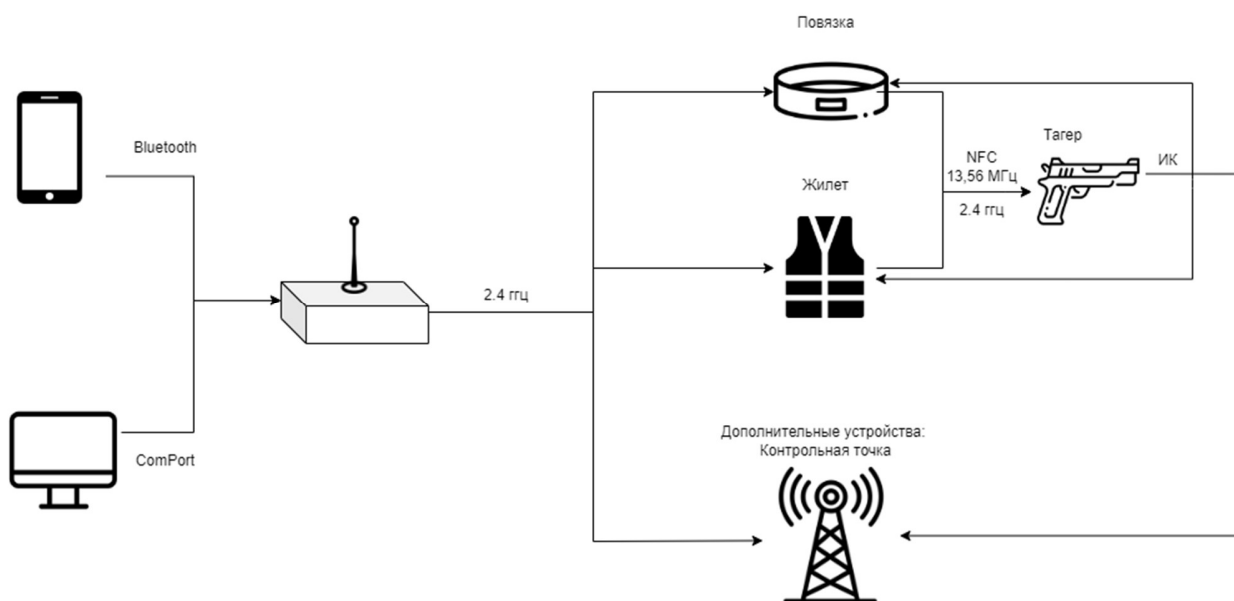


Рисунок 1 - Абстрактное представление работы оборудования

Было необходимо реализовать обмен данными только на стороне программного обеспечения.

1.1 Протокол обмена данными между устройствами и программным обеспечением

Для полноценной работы оборудования должно быть реализовано 3 протокола обмена данными:

- 1) Для обмена между радиобазой и программным обеспечением, по COMPORT и Bluetooth
- 2) Для обмена между радиобазой и устройствами на частоте 2.4 ГГц
- 3) Для обмена между устройствами и тагерами на частоте 2.4 ГГц и 13.56 МГц
- 4) Для передачи данных по ИК-порту от тагеров на устройства

В случае с оборудованием, представленным для разработки программного обеспечения, радиобазы являются ретрансляторами, поэтому 1 и 2 протоколы обмена были объединены в один.

Протокол обмена данными между ПО и устройствами был разделен на группы команд:

1) Команды для радиобазы:

Таблица 1 - Команды для радиобазы

Название команды	Первый байт	Второй байт
Поиск базы	0xFF	-
Группа команд для базы	0xFE	...
Данные о базе (UID, ревизия, версия)	0xFE	0x01
Выключить поиск новых игроков	0xFE	0x10
Включить поиск игроков	0xFE	0x11
Выключить обратную связь	0xFE	0x12
Включить обратную связь	0xFE	0x13

2) Команды на радиобазу для общей рассылки на устройства:

Таблица 2 - Команды для общей рассылки

Команда	Описание
0x02	Пауза (не воспринимаем выстрелы, не можем подвязать тагеры, и переводим свои тагеры в состояние не активен, с сохранением жизни и здоровья)
0x03	Стоп раунда (не воспринимаем выстрелы, не можем подвязать тагеры, переводим свои тагеры в состояние не активен, запись стоп игра, стоп сбора статистики)
0x04	Конец игры (не воспринимаем выстрелы, не можем подвязать тагеры, перевожу свои тагеры в состояние не активен, жизнь и здоровье в ноль, запись конец игры, стоп сбора статистики) Переход через 30с в настройки
0x05	Разблокировать игрока (принимаем выстрел и тагер активен)
0x06	Старт раунда (принимаем выстрелы, активируем тагеры, начинаем сбор статистики, передаем статистику, оживление и восстановление патронов)

3) Команды на базу для индивидуальной рассылки на устройства. Такие команды обновляются, добавляются и чистятся с выходом новых версий устройств. Также они делятся еще на несколько групп

- a. Команды на главные устройства (повязки, жилеты)
- b. Команды на дополнительные устройства (контрольные точки)
- c. Команды на привязываемые устройства (тагеры)

Пример таких команд соответственно каждой группе описаны в таблице 3.

Таблица 3 - Пример индивидуальных команд

Номер	UID главного устройства	UID привязываемого устройства	Описание	Размер значения	Диапазон значений
0x35	UID повязки	-	Здоровье	2 байта	1-999
0xB0	UID повязки	UID тагера	Урон	1 байт	0-15
0x41	UID контрольной точки	-	Сценарий	1 байт	0-3

4) Команды с базы – возможные ответы, которые база может прислать на разные команды описаны в таблице 4.

Таблица 4 - Ответы с радиобазы

Номер	Размер	Описание ответа
0x80	1 байт	ОК. Ответ на посылки, не требующие ответа
0x81	4 байта	UID не подключен к базе
0x82	21 байт	Конфиг повязки
0x83	####	UID повязки. (0xFF Конец) Ответ на команду 0x19
0x84	####	UID тагеров. (0xFF Конец) Одним пакетом. Ответ на команду 0x29
0x90	1 байт	Таймаут. Не получил ответ на индивидуальную посылку от повязки.
0x91	1 байт	Значение в команде не верное
0x92	1 байт	Команда не распознана

Важно учитывать, что, по мере роста функционала оборудования, будут добавляться новые команды, поэтому следовало реализовать архитектуру обмена

данными в программном обеспечении так, чтобы при изменении протокола взаимодействия с радиобазой не требовалось добавлять новые методы для каждой нововведённой команды. Отправка и приём команд должны быть гибкими и универсальными.

1.2 Протокол обмена данными между программным обеспечением и сервером

Сервер является неотъемлемой частью для продвинутых лазертаг клубов. С помощью сервера, игроки могут отслеживать свои достижения из любой точки, анализировать свою игру и игру соперников. Клубы могут соревноваться между собой в интенсивности проводимых игр и привлекать новых игроков. Всё это возможно благодаря возможности взаимодействия программного обеспечения с сервером. Протокол обмена данными между сервером и ПО строится на:

1) HTTP метод POST, самый распространённый способ отправки и приёма данных. Он может отправлять любые типы данных: текст, файлы, изображения. Вся нужная информация отправляется в теле HTTP-запроса, что делает POST надежным и безопасным методом передачи данных.

2) JSON формат – текстовый формат обмена данными, состоящий из объектов, которые представляют собой пары ключ-значение, где ключ – строка, а значением может быть любой тип данных, даже другой объект

Любой POST запрос на сервер лазертага включает в себя объект JSON. На любой запрос возвращается ответ в JSON с обязательным полем status, которое может принимать значения ok и error.

Все запросы на сервер:

1) Получение данных о существующих игроках и командах клуба приведены в таблице 5.

Таблица 5 - Получение игроков и команд клуба

ключ	описание
act	10
login	логин клуба
hash	hash пароля
v	версия api

2) Добавление нового клуба приведено в таблице 6.

Таблица 6 - Добавление нового клуба

ключ	описание
act	11
login	уникальный логин клуба
hash	хеш пароля
name	название клуба
email	e-mail
contacts	телефон, другая контактная информация
city	название города
v	версия api

3) Добавление команды приведено в таблице 7

Таблица 7 - Добавление команды

Ключ	описание
act	12
idclub	идентификатор клуба
login	логин клуба
hash	хеш пароля клуба
name	название команды

Ключ	описание
dopinfo	комментарий
v	версия api

4) Добавление игрока приведено в таблице 8.

Таблица 8 - Добавление игрока

Ключ	Описание
act	13
idclub	идентификатор клуба
login	логин клуба
hash	хеш пароля клуба
name	имя игрока
dopinfo	Комментарий
v	версия api

5) Редактирование игрока приведено в таблице 9.

Таблица 9 - Редактирование игрока

Ключ	Описание
act	15
idclub	идентификатор клуба
login	логин клуба
hash	хеш пароля клуба
idplayer	идентификатор редактируемого игрока
name	имя игрока
fio	ФИО игрока
city	название города игрока
email	задать email (не влияет на email-логин пользователя)

Продолжение таблицы 9	
Ключ	Описание
phone	задать телефон (не влияет на контакты, указанные пользователем) – строка
bdate	дата рождения пользователя (строка в формате <i>04.04.1990</i>)
sex	пол (0 - мужской, 1 - женский)
dopinfo	Комментарий
currteam	текущая команда
status	Статус
v	версия api

6) Редактирование команды приведено в таблице 10.

Таблица 10 - Редактирование команды

Ключ	Описание
act	16
idclub	идентификатор клуба
login	логин клуба
hash	хеш пароля клуба
idteam	идентификатор редактируемой команды
name	название команды
dopinfo	Комментарий
v	версия api

7) Удаление игрока приведено в таблице 11.

Таблица 11 - Удаление игрока

Ключ	Описание
Act	17
idclub	идентификатор клуба
login	логин клуба
Hash	хеш пароля клуба
idplayer	идентификатор удаляемого игрока
V	версия api

8) Удаление команды приведено в таблице 12.

Таблица 12 - Удаление команды

Ключ	Описание
act	18
idclub	идентификатор клуба
login	логин клуба
hash	хеш пароля клуба
idteam	идентификатор удаляемой команды
v	версия api

9) Отправка игры на сервер:

Структура json первой версии описана в таблице

Таблица 13 - Структура Json отправки на сервер

Ключ	Описание
version	версия протокола отправки игры (int);
login	логин клуба (string)
hash	хэш пароля клуба (string)
hash_game	md5 для защиты от повторной загрузки одной и той же игры (string)
is_test	если 1, то запись в БД не происходит, но выводится тестовая информация
game_info	информация об игре
name	название игры (string)
poligon	название места проведения игры, если есть (string)
date_start_game	дата и время начала игры в unixtime по utc (int)
duration_of_the_game	продолжительность игры в секундах (int)
winning_team_ID	игровой идентификатор команды-победителя (int)
winning_player_ID	игровой идентификатор игрока-победителя (int)
teams	список команд (array)
players	список игроков (array)
events	игровые события (попадания, убийства и т.п.) (array)

1.4 Создание и анализ технического задания и способов решения

Исходя из анализа предметной области и вышеописанных протоколов было составлено техническое задание. Оно включает в себя следующие этапы:

1. Разработка программного обеспечения под операционные системы Windows и Android
2. Разработка конфигуратора устройств, который включает в себя:
 - a. Взаимодействие с радиобазой по COMPORT или BLE с помощью протокола передачи данных.
 - b. Удобный и понятный пользовательский интерфейс.
 - c. Передачу данных в реальном времени и без задержек.
3. Разработка меню управления игрой, которое включает в себя:

- a. Передачу основных команд на устройства, таких как «старт игры» и «стоп игры».
 - b. Считывание статистики игры в реальном времени.
 - c. Возможность отправить результат игры на сервер.
- 4. Разработка меню управления клубом, которая включает в себя:
 - a. Авторизацию и регистрацию клуба.
 - b. Запросы на получение всех игроков и команд, состоящих в клубе, а также их характеристики.
 - c. Запросы на изменение информации о клубе, игроках и командах в реальном времени.
- 5. Тестирование и отладка проекта.

Для реализации проекта был составлен список языков программирования и платформ разработки, которые подходили для написания программного обеспечения.

Плюсы и минусы использования Native C/C++ приведены в таблице 13.

Таблица 14 - Плюсы и минусы C/C++

Платформа	Язык программирования
Native	C/C++
Плюсы	Минусы
Производительность программного обеспечения	Сложность языка программирования
Полный контроль использовании памяти	Время на разработку
Кроссплатформенность	Необходимость оптимизации

Плюсы и минусы использования .NET Xamarin приведены в таблице 15.

Таблица 15 - Плюсы и минусы .Net и Xamarin

Платформа	Язык программирования
.NET Xamarin	C#
Плюсы	Минусы
Кроссплатформенность	Xamarin не подходит для приложений с высокопроизводительной графикой
Производительность	Сложность создания динамического интерфейса
Удобство при разработке	Большой размер приложений

Плюсы и минусы использования Unreal Engine приведены в таблице 16.

Таблица 16- Плюсы и минусы Unreal Engine

Платформа	Язык программирования
Unreal Engine	C/C++
Плюсы	Минусы
Производительность программного обеспечения	Ограниченные возможности создания UI элементов
Полный контроль использования памяти	Сложность портирования приложения на мобильные платформы

Плюсы и минусы использования Unity приведены в таблице 17

Таблица 17 - Плюсы и минусы Unity

Платформа	Язык программирования
Unity	C#
Плюсы	Минусы
Время разработки	Необходимость оптимизации
Кроссплатформенность	Сложность создания динамического интерфейса
Мощные функциональные возможности	Большой размер приложений

Исходя из того, что нужно реализовать программное обеспечение для нескольких платформ, было разработано 2 похожих по функционалу и дизайну приложения. Анализируя все способы реализации, было принято решение использовать игровой движок Unity, который позволяет разрабатывать игры и UI приложения на разные платформы. В Unity удобная среда разработки, где можно импортировать объекты на сцене и скрипты из других проектов, что сокращает время на создание нескольких приложений, так-как интерфейс и логика реализованных программ во многом схожа. Также Unity имеет Asset Store – магазин, где можно найти необходимые пакеты и ускорить разработку.

В результате выполнения проекта ожидалось получить 2 программных обеспечения на платформы Windows и Android, которые обеспечат администраторов лазертаг клубов возможностями настройки специализированного оборудования, управления игрой и ведением отчётности всех игр с помощью сохранения на сервере.

2 Разработка

В данной главе рассмотрена разработка всех классов и методов для построения основного функционала приложения.

2.1 Создание проекта и сцены под Desktop

Меню конфигуратора включает в себя множество параметров, которые были утверждены и расставлены в дизайне приложения. Под все основные меню выделены объекты, включающие в себя компонент «CanvasGroup». Данное решение было принято за счёт удобства взаимодействия с этим компонентом. Встроенный класс был расширен двумя методами, отвечающими за прозрачность, с помощью которых реализовано открытие и закрытие основных меню и представлен на рисунке 2.

```
public static class CanvasGroupExtension
{
    public static void AlphaAndRaycastAndInteractiveToggle(this CanvasGroup canvasGroup, bool isActive)
    {
        canvasGroup.blocksRaycasts = isActive;
        canvasGroup.interactable = isActive;
        canvasGroup.alpha = isActive ? 1 : 0;
    }

    public static void AlphaInteractive(this CanvasGroup canvasGroup, float alpha, bool isActive)
    {
        canvasGroup.blocksRaycasts = isActive;
        canvasGroup.interactable = isActive;
        canvasGroup.alpha = alpha;
    }
}
```

Рисунок 2 - Методы расширения CanvasGroup

На рисунке 3 представлены элементы конфигуратора были и разделены на «настройки повязки», «настройки тагера», а они, разбиты на «основные настройки» и «дополнительные настройки», согласно дизайну.



Рисунок 3 - Пример расположения элементов конфигулятора на сцене

Каждый параметр – объект с наследником класса «MultyTypeIntParameter», который содержит в себе встроенный InputField, нижний и верхний порог значения и событие изменения параметра. Каждый наследник переопределяет метод разбора внесенного пользователем значения. Например, в поле ввода «здоровье», пользователь может занести только целое число не меньше 1 и не больше 999. Скрипт на сцене приведен на рисунке 4.

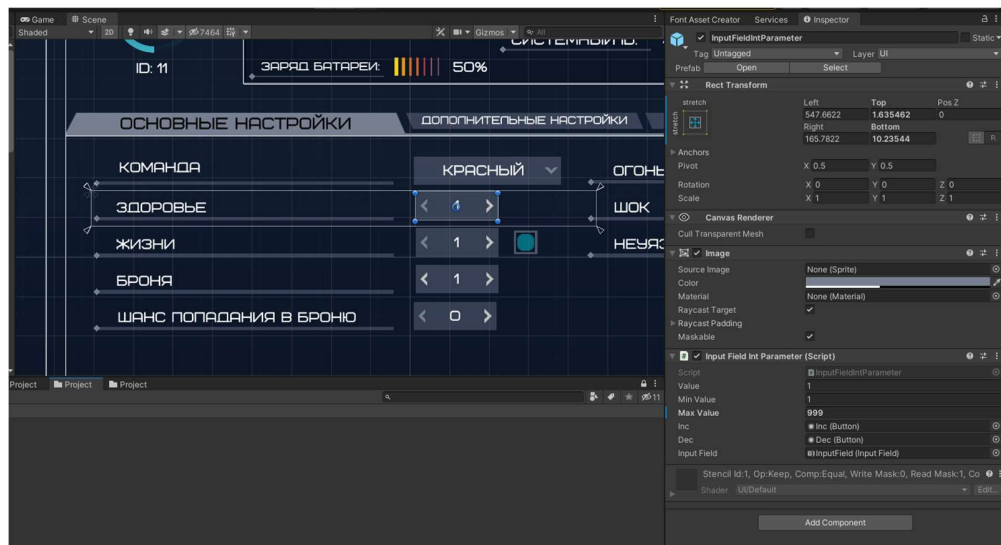


Рисунок 4 - Параметр конфигулятора

Скрипт объекта приведен на рисунке 5.

```
public class InputFieldIntParameter : MultyTypeIntParameter
{
    [SerializeField] private int m_value;
    [SerializeField] private int minValue, maxValue;
    [SerializeField] private Button inc, dec;
    [SerializeField] private InputField inputField;

    public int MaxValue
    {
        get => maxValue;
    }

    public int MinValue
    {
        get => minValue;
    }

    private bool isActive;

    public delegate void OnValueChanged(int value);
    public event OnValueChanged onValueChanged;

    public override int Value { get
    {
        return m_value;
    }
    set
    {
        SetClampedValue(value);
    }
    }
}
```

Рисунок 5 – Класс наследник MultyTypeIntParameter

Классы конфигулятора, как и UI, разделены на «конфигуратор повязки» и «конфигуратор тагера» - «HeadBandConfiguration», «TaggerConfiguration» соответственно. Каждый содержит в себе соответствующие сериализованные

объекты параметров, методы инициализации девайса и добавление слушателей на события изменения параметров представлен на рисунке 6.

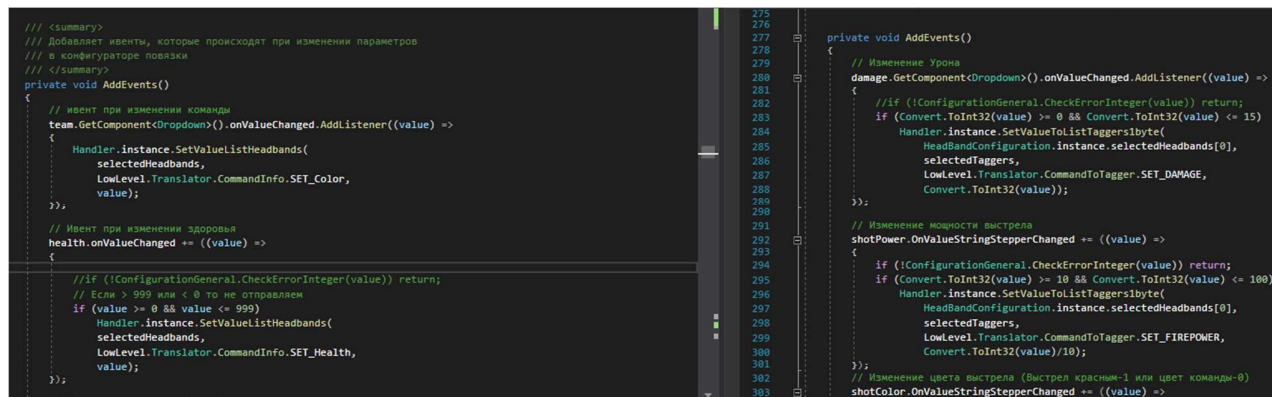


Рисунок 6 - Обработка событий изменения параметров

Каждое изменение параметра обращается к классу одиночке Handler, через который передаются все команды на COMPORT. Он содержит в себе объект реализации интерфейса ICommander, через который идёт основной обмен данными.

2.3 Разработка многопоточных методов и очереди

Первое, что требовалось реализовать – многопоточность. Для разработки многопоточных приложений на Unity следует использовать сторонние библиотеки или усложнённый функционал, потому что движок запрещает вызывать методы, в которых затрагивается UI, не из основного потока. Для выполнения поставленной задачи был взят готовый скрипт «UnityThread», который отслеживает возвращение из стороннего потока и синхронизируется с основным во встроенных методах – Update/FixedUpdate. При старте приложения требуется инициализировать UnityThread, после чего создаётся DontDestroyOnLoad объект, который будет производить синхронизацию ответов из других потоков. В Handler инициализируется объект типа «Intaracton», который также, как и Commader реализует все методы интерфейса ICommander. Всё это сделано для того, чтобы объект типа Interaction был мостом для вызова методов

Commander, которые должны выполняться уже не из главного потока. Пример работы метода отправки данных на тагер через новый поток:

- 1) TaggerConfigurator обращается к Handler, чтобы отправить введенные пользователем данные. Код представлен на рисунке 7.

```
// изменение яркости подсветки
brightLighting.onValueChanged += ((value) =>
{
    if (value >= 1 && value <= 10)
        Handler.instance.SetValueToListTaggers1byte(
            HeadBandConfiguration.instance.selectedHeadbands[0],
            selectedTaggers,
            LowLevel.Translator.CommandToTagger.SET_BRIGHT_LED,
            value);
});
```

Рисунок 7 - Обращение к Handler

- 2) Handler вызывает метод «SetToTagger» у объекта commander, который имеет тип Interaction. Код представлен на рисунке 8.

```
if (tags.Count == 0) return;
LoadingPanelManager.instance.Toggle(true);
commander.SetToTagger(tags[i].currentBand, tags[i], command, values, (msg) =>
{
    if (msg == Base.OK)
```

Рисунок 8 - Вызов метода в Commander

- 3) Класс Interaction реализовывает метод интерфейса ICommander так, чтобы создался новый поток и объект типа Commander исполнялся уже не из основного потока, а Action callback вернулся обратно в Interaction для дальнейшей синхронизации потоков в Update через UnityThread.

```

void ICommander.SetToTagger(HeadBand band, Tagger tag, Translator.CommandToTagger command, List<int> values, Action<Base> callback, Priority priority = Priority.Default)
{
    new Thread(() =>
    {
        commander.SetToTagger(band, tag, command, values, (msg) =>
        {
            UnityThread.executeInUpdate(() =>
            {
                callback(msg);
            });
        }, priority);
    }).Start();
}

```

Рисунок 9 - Создание нового потока

- 4) После исполнения операции через обратный вызов обрабатывается результат выполнения команды.

```

commander.SetToTagger(tags[i].currentBand, tags[i], command, values, (msg) =>
{
    if (msg == Base.OK)
    {
        // все применилось
        if(i < tags.Count)
        {
            tags[i].onUpdate?.Invoke();
            UnityEngine.Debug.Log($"{tags[i].uid.ToString("X")} - ok ");
        }
    }
    else if(msg == Base.UndefinedCommand)
    {
        Debug.Log($"{tags[i].uid.ToString("X")} Tagger undefined command");
    }
    else
    {
        if (i < tags.Count)
        {
            tags[i].connection -= 5;
            if (tags[i].connection != 0)
            {
                List<Tagger> errorTagers = new List<Tagger>();
                errorTagers.Add(tags[i]);
                SetValueRecursive(0, band, errorTagers, command, values);
            }
        }
    }
}

```

Рисунок 10 - Приём ответа из метода в Commander

Автоматизация фоновое взаимодействия программного обеспечения с радиобазой создала довольно весомое количество отправки и получения не важных данных, из-за чего могла возникнуть проблема переполнения буфера. Для избежания возможных ошибок, было принято разделить методы обмена на приоритеты. Например, запрос на обнаружение новых устройств, который вызывается раз в несколько секунд менее важен, чем запрос на изменения какой-

либо конфигурации, которую применил пользователь. Было принято решение о добавление 3 приоритетов, которые представлены в таблице 18.

Таблица 18- Приоритеты исполнения команд

Приоритет	Описание
Low	Низкий приоритет – выполняется только в том случае, если очередь и команд на выполнение пустая. Если на момент вызова метода с низким приоритетом очередь будет занята хотя-бы одним элементом на ожидание, то данный метод не будет выполняться и занимать очередь.
Default	Обычный приоритет – выполнится, как только очередь методов с обычным приоритетом дойдёт до него, а очередь с методами высокого приоритета будет пуста.
Hight	Высокий приоритет – метод займет очередь для высоких приоритетов и выполнится либо сразу, либо после вызванных ранее методов с таким-же приоритетом.

Для управления очередями были объявлены переменные, которые хранят в себе индексы методов на исполнение в очереди.

```

/// <summary>
/// Флаг, который запрещает пользоваться компортом,
/// когда он занят для исполнения другого метода
/// </summary>
static volatile bool ahShitHereWeGoAgain = false;

/// <summary>
/// Последний прибывший запрос с ПК
/// </summary>
static volatile int lastQuery = 0;

static volatile int lastQueryHightPriority = 0;

/// <summary>
/// Тот, который на данный момент выполняется
/// </summary>
static volatile int currentQuery = 0;

static volatile int currentQueryHightPriority = 0;

/// <summary>
/// Время ошибки очереди COUNT_ERROR_WAIT_QUEUE * 10
/// </summary>
private int COUNT_ERROR_WAIT_QUEUE = 6000;

```

Рисунок 11 - Переменные управления очередями

HereWeGoAgain – Флаг, который следит за тем, не занят ли COMPORT передачей данных.

lastQuery – Индекс последнего в очереди на выполнение метода

lastQueryHightPriority – Индекс последнего в высокоприоритетной очереди на выполнение метода

currentQuery – Индекс метода в очереди, который на данный момент выполняется.

currentQueryHightPriority – Индекс метода в высокоприоритетной очереди, который на данный момент выполняется.

COUNT_ERROR_WAIT_QUEUE – Переменная хранящая в себе время до ошибки. Если на протяжении этого времени, умноженного на 10, индексы очереди не изменят значения (очередь не сдвинется), то, возможно, какой-либо метод не смог закончить своё выполнение и не вернул значение. Переменная была создана для исключения необходимости перезапуска программного обеспечения в случае ошибки в коде.

Метод занимания очереди принимает в себя 1 аргумент – приоритет, от которого алгоритм изменяет определенные индексы. По умолчанию приоритет «Default».

```
private bool WaitQueue(Priority priority = Priority.Default)
```

Рисунок 12 - Аргумент метода ожидания очереди

Также WaitQueue возвращает ответ типа bool:

- 1) True – Можно начинать выполнение команды.
- 2) False – Следует отменить выполнение команды.

False вернется только в том случае, если будет низкий приоритет на выполнение.

Если приоритет «Default» - то WaitQueue вернёт значение на True, только тогда, когда очередь высокоприоритетных команд будет пуста и индекс очереди обычных команд дойдёт до нужной.

```
if(priority == Priority.Default)
{
    // Присваиваем методу номер в очереди
    int thisQuery = lastQuery;
    // Показываем, что очередь расширилась
    lastQuery++;
    // Вдруг я где-то забыл закончить очередь
    int count_error_check = 0;

    // ждёт своей очереди ...
    while ((thisQuery != currentQuery) || (lastQueryHighPriority != 0))
    {
        Thread.Sleep(10);
        count_error_check++;
        if (count_error_check >= COUNT_ERROR_WAIT_QUEUE)
        {
            UnityEngine.Debug.LogError("Кручусь в очереди уже минуту, где-то видимо забыл закончить очередь");
            break;
        }
        continue;
    }
    HereWeGoAgain = true;
    return true;
}
```

Рисунок 13 - Алгоритм ожидания очереди для команд с обычным приоритетом

Если приоритет «Low» - то WaitQueue вернёт значение True, только если ни одна из очередей не будет занята более приоритетными задачами на выполнение, иначе вернётся False и метод вызывавший WaitQueue должен будет прекратить выполнение.

```

else if (priority == Priority.Low)
{
    // Если в очередях никого нет, то занимаем очередь
    if (lastQuery == 0 && lastQueryHightPriority == 0)
    {
        lastQuery++;
        HereWeGoAgain = true;
        return true;
    }
    // Если в одной из очередей кто-то стоит - выходим
    else
    {
        return false;
    }
}

```

Рисунок 14 - Алгоритм ожидания очереди для команд с низких приоритетом

Если приоритет «Hight» - то WaitQueue вернёт значение на True, только тогда, когда индекс очереди высокоприоритетных команд дойдёт до нужной.

```

else if (priority == Priority.Hight)
{
    // Присваиваем методу номер в очереди
    int thisQuery = lastQueryHightPriority;
    // Показываем, что очередь расширилась
    lastQueryHightPriority++;
    // Вдруг я где-то забыл закончить очередь
    int count_error_check = 0;

    // ждёт своей очереди ...
    while ((thisQuery != currentQueryHightPriority) || HereWeGoAgain)
    {
        Thread.Sleep(10);
        count_error_check++;
        if (count_error_check >= COUNT_ERROR_WAIT_QUEUE)
        {
            UnityEngine.Debug.Log("Кручусь в очереди уже 30 секунд, где-то видимо забыл закончить очередь для приоритетных");
            break;
        }
        continue;
    }
    HereWeGoAgain = true;
    return true;
} else

```

Рисунок 15 - Алгоритм ожидания очереди для команд с высоким приоритетом

2.4 Отправка данных по COMPORT и BLE

Передача и приём данных были реализованы в классе Translator. В случае с десктопным приложением данный класс содержит в себе методы, которые взаимодействуют с классом из метаданных «SerialPort», через который и реализуется отправка и приём данных по физическому COMPORT'у. При написании Android приложения возникли сложности взаимодействия с BLE, так как Unity не встроено нужного функционала. Было решено написать собственную библиотеку JAR, так как Unity поддерживает возможность

подключения сторонних Android библиотек, а подходящих пакетов, которые работали бы в новом потоке, в Asset Store не нашлось.

Для написания собственной библиотеки на Android понадобилось следующее:

- 1) Установить удобную IDE
- 2) Выбрать язык программирования, который поддерживает Android SDK
- 3) Собрать и подключить библиотеку к основному проекту в Unity

IDE была выбрана Android Studio, из-за надежности и схожести с Visual Studio. При выборе языка программирования выбор пал на Java, так как было легко перейти на него с C# и довольно обширная официальная документация. Далее был написан небольшой скрипт, который обращался к Android SDK и взаимодействовать с Bluetooth Low Energy. После отладки и успешной сборки, в основном проекте был создан класс BleManager, через который и было реализовано взаимодействие с написанным модулем. Библиотека хранилась в объекте типа AndroidJavaObject, а все методы вызывались с помощью CallStatic и Call.

```

public class BleManager
{
    private AndroidJavaClass unityClass;
    private AndroidJavaObject unityActivity;
    private AndroidJavaClass customClass;
    private AndroidJavaObject ble_instance;

    private const string PackageName = "com.lsd.ble.Manager";
    private const string UnityDefaultJavaClassName = "com.unity3d.player.UnityPlayer";

    public void Init()
    {
        AndroidJNI.AttachCurrentThread();

        unityClass = new AndroidJavaClass(UnityDefaultJavaClassName);
        unityActivity = unityClass.GetStatic<AndroidJavaObject>("currentActivity");
        customClass = new AndroidJavaClass(PackageName);
        customClass.CallStatic("InitInstance", unityActivity);
        ble_instance = customClass.GetStatic<AndroidJavaObject>("instance");
    }
}

```

Рисунок 16 - Класс для взаимодействия с BLE библиотекой

В Commander инициализирован объект типа Translator. Несколько основных методов, реализованных в классе Translator: «Opros_to» конвертирует все данные на отправку, переданные в аргументах, в тип byte, собирает их в массив и добавляет в буфер на отправку, через метод Write, заранее очистив буфер. У класса «SerialPort» данный метод встроен. Метод «Opros_to» переопределён под разные задачи:

```

#region OPROS_TO
/// <summary>
/// Команда на базу/повязку
/// </summary>
public int Opros_to(CommandInfo Command, UInt32 uid = 0, List<int> values = null)...
/// <summary>
/// Команда на тагер
/// </summary>
public int Opros_to(CommandToTagger Command, UInt32 uid, uint uidTag, List<int> values = null)...
/// <summary>
/// Команда пресета на тагер
/// </summary>
public int Opros_to_tagger_preset(CommandToTagger Command, UInt32 uid, uint uidTag, List<int> values = null)...
/// <summary>
/// Команда на обновление
/// </summary>
private int Opros_to(byte[] bytes, bool parityBit, Action<string> logCallback)...
/// <summary>
/// Команда на обновление тагера
/// </summary>
private int Opros_toTaggerCycle(byte[] bytes, int number)...
/// <summary>
/// Команда на контрольную точку
/// </summary>
public int Opros_to(CommandToControlPoint Command, UInt32 uid = 0, List<int> values = null)...

#endregion

```

Рисунок 17 - Методы для отправки данных

«Opros_From» принимает данные из буфера приёма.

```

/// <summary>
/// получение ответа
/// </summary>
/// <param name="CurrCommands">список возможных ответов</param>
/// <returns></returns>
public Response Opros_From(List<CommandFromInfo> CurrCommands, int timeout = 1000)...
/// <summary>
/// отдельный метод получения данных из статистики
/// </summary>
public Response Opros_From_Statistic(List<CommandFromInfo> CurrCommands, int timeout = 1000)...
/// <summary>
/// получение ответа на запрос UID всех девайсов
/// </summary>
/// <param name="CurrCommands">список возможных ответов</param>
/// <returns></returns>
public Response Opros_From_Get_Devices(List<CommandFromInfo> CurrCommands, int timeout = 1000)...

```

Рисунок 18 - Методы для приёма данных

Ответ на запрос приходит через разный интервал времени после отправки, поэтому был реализован цикл, в котором ведется опрос на наличие байт в буфере приёма.

```

//Если нет вообще байтов
try
{
    // 1
    while (ComPort.BytesToRead <= 0)
    {
        //Ждем новые байты
        Thread.Sleep(5);
        if (TimeOutRead <= DateTime.Now - startTimeRead)
        {
            //Не дождались ответа и вышли
            is_FailTimeout = true;
            break;
        }
    }
}
catch (Exception ex)
{
    resp.status = -1;
    // ошибка при ожидании данных
    return resp;
}

```

Рисунок 19 - Цикл проверки присутствия данных в буфере приёма COMPORT'a

```

try
{
    do
    {
        bytesToRead = bleManager.HaveBytesToRead();
        //UnityEngine.Debug.Log($"bytes to Read = {bytesToRead}");
        // Wait Response from plugin
        if(bytesToRead > 1)
        {
            ZdyLogger.LogError("Bytes to read more than 1");
        }
        Thread.Sleep(15);
        if (TimeOutRead <= DateTime.Now - startTimeRead)
        {
            //Timeout
            //ZdyLogger.Log("Timeout");
            is_FailTimeout = true;
            break;
        }
    }
    while (bytesToRead == 0);
    // пока 0 (не приходил Callback OnCharacteristicWrite)
}
catch (Exception ex)
{
    res.status = -1;
    ZdyLogger.LogError("exception " + ex.Message);
    return res;
}

```

Рисунок 20 - Цикл проверки присутствия данных в буфере приёма BLE библиотеки

Ответ на переданные данные возвращается через Callback, где синхронизируется с основным потоком программы, после чего результат

обрабатывается и исполняется нужная часть кода, например изменение конфигурации объекта и обновление информации в UI.

2.4 разработка классов взаимодействия с сервером

Для отправки POST запросов на сервер было реализовано 2 класса:

- 1) WebManager – Класс реализованный с паттерном Singleton, для вызова из любой части приложения. Он включает в себя все предоставленные в протоколе методы с соответствующими параметрами. После вызова одного из методов, класс обращается к объекту типа WebSender, который отвечает за от отправку POST запросов и возвращение результата.
- 2) WebSender – Класс реализующий методы отправки любого POST запроса в Coroutine с возвращением результата через Action callback, метод проверки интернет соединения и метод скачивания файлов по URL.

Пример работы метода отправки данных для добавления игрока на сервер: Для добавления игрока на сервер следует передать POST запрос со следующими параметрами: id клуба, логин клуба, захешированный пароль клуба, имя нового игрока, дополнительная информация о новом игроке. Данный запрос составит метод с названием «AddPlayer», после чего передаст в WebSender для дальнейшей отправки.

```
public const string ACT_ADD_PLAYER = "13";

public static void AddPlayer(string idclub,
    string login, string hash, string name, string dopinfo, Action<string> callback)
{
    string post = $"act={ACT_ADD_PLAYER}" +
        $"&idclub={idclub}" +
        $"&login={login}" +
        $"&hash={hash}" +
        $"&name={name}" +
        $"&dopinfo={dopinfo}" +
        $"&v={API_VERSION}";

    instance.webSender.PostQueryOnServer(post, LINK_POST, callback);
}
```

Рисунок 21 - Метод добавления игрока в WebManager

Для отправки данных на сервер используется метод `PostQueryOnServer`, который принимает следующие параметры:

- 1) Сформированный запрос с типом `string`
- 2) url адрес, по которому следует отправить запрос

```
private const string lsdURL = "https://laserstat.pro/";

public void PostQueryOnServer(string postData, string url, Action<string> callback)
{
    DevLog.requestCounter++;
    StartCoroutine(Download(postData, url, (data) =>
    {
        DevLog.answerCounter++;
        if (data != null)
        {
            string result = System.Text.Encoding.UTF8.GetString(data);
            ServerRequestReader.RequestLog(result);
            callback.Invoke(result);
        } else
        {
            callback.Invoke(null);
        }
    }));
}
```

Рисунок 22 - Метод отправки сформированного запроса на сервер

Далее создаётся `Coroutine` для приёма ответа с сервера, который придёт через несколько кадров. Внутри запущенной «корутины» создается объект типа «`UnityWebRequest`», заполняется параметрами и встаёт на исполнение.

```
IEnumerator Download(string postData, string url, Action<byte[]> callback)
{
    UnityWebRequest www = new UnityWebRequest(url);
    www.uploadHandler = new UploadHandlerRaw(Encoding.UTF8.GetBytes(postData));
    www.downloadHandler = new DownloadHandlerBuffer();
    www.uploadHandler.contentType = "application/x-www-form-urlencoded";
    www.method = UnityWebRequest.kHttpVerbPOST;

    yield return www.SendWebRequest();

    if (www.result != UnityWebRequest.Result.Success)
    {
        Debug.Log(www.error);
        callback.Invoke(null);
    }
    else
    {
        callback.Invoke(www.downloadHandler.data);
    }

    www.Dispose();
}
```

Рисунок 23 - Корутина для отправки и ожидания ответа

После исполнения, данные из ответа перекодируются в UTF-8 и передаются через Action callback в метод, откуда был вызван запрос. В данном примере результат будет конвертирован в JSON формат. Если ответ с сервера был успешный, то создается новый объект игрока клуба и добавится в список существующих.

```
WebManager.AddPlayer(ForumClub.ID_CLUB.ToString(), ForumClub.CLUB_LOGIN, ForumClub.CLUB_PASSWORD,
    username.text, dopInfo.text, (msg) =>
    {
        AnswerPlayer answer = JsonUtility.FromJson<AnswerPlayer>(msg);

        if(answer != null)
        {
            PlayerLSD newPlayer = new PlayerLSD(answer.idplayer, answer.temppas, username.text, dopInfo.text);
            playerLSDSpawner.AddPlayerLSD(newPlayer);

            ForumClub.PickPlayer(newPlayer);
        } else
        {
            // Игрок не добавлен
        }
    });
```

Рисунок 24 - Обработка ответа от сервера

2.5 Отладка и тестирование приложений

Отладка и тестирование написанного приложения являются неотъемлемой частью в разработке. Они позволяют выявить и устранить ошибки в коде и проверить работоспособность.

В данном пункте приведена проверка отправки валидных и не корректных данных.

Проверка отправки команды на изменение здоровья. Изначально здоровье равно 160, что представлено на рисунке 25.

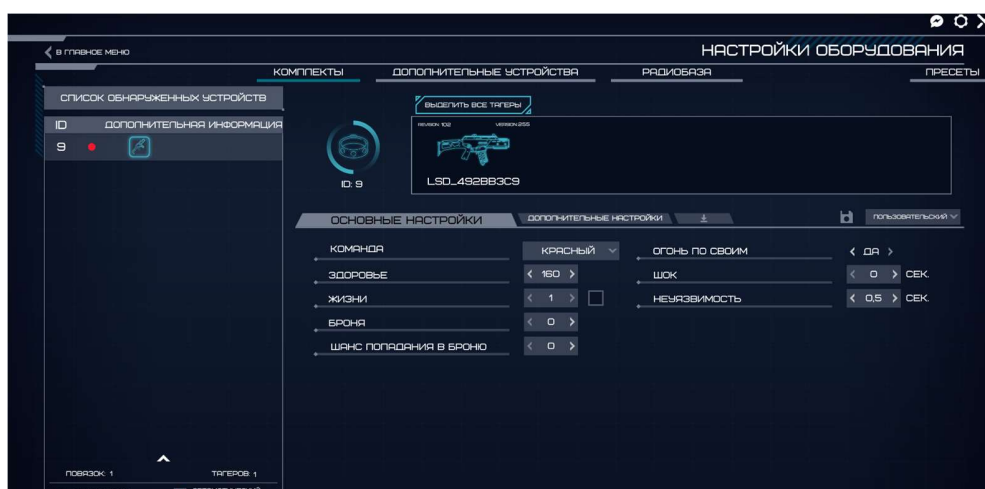


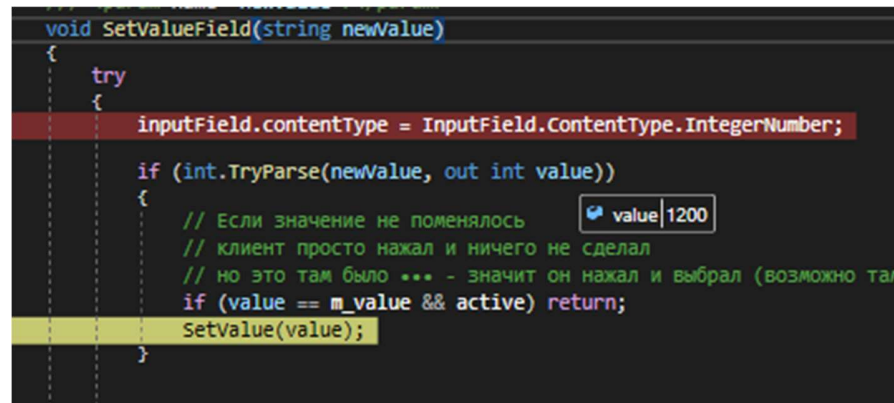
Рисунок 25 - Начальное значение параметра "здоровье"

После применения значения выше верхнего порога в 999, написанный класс «InputFieldIntParameter» перевело параметр в максимальное значение.



Рисунок 26 - Внесенное значение параметра "здоровье" с превышением максимального порога

Для того, чтобы убедиться в работоспособности, следует воспользоваться активной отладкой. По изменении параметра в InputField производится обработка введенных данных:

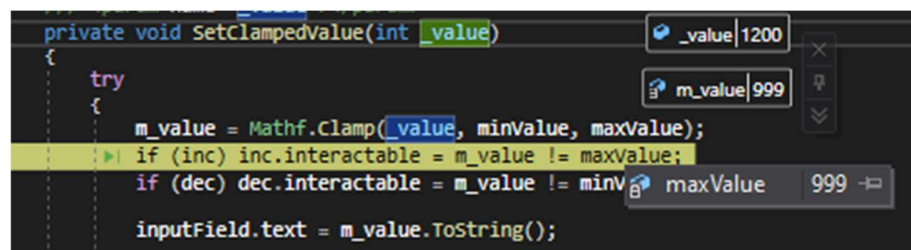
A screenshot of a code editor showing the implementation of the `SetFieldValue` method. The method takes a `string newValue` as input. It first sets `inputField.contentType = InputField.ContentType.IntegerNumber;`. Then, it attempts to parse the `newValue` into an integer using `int.TryParse`. If parsing is successful, it checks if the value is different from the current `m_value` and if the field is active. If both conditions are met, it calls `SetValue(value);`.

```
void SetFieldValue(string newValue)
{
    try
    {
        inputField.contentType = InputField.ContentType.IntegerNumber;

        if (int.TryParse(newValue, out int value))
        {
            // Если значение не поменялось
            // клиент просто нажал и ничего не сделал
            // но это там было ... - значит он нажал и выбрал (возможно та
            if (value == m_value && active) return;
            SetValue(value);
        }
    }
}
```

Рисунок 27 - Метод для обработки внесенных изменений в InputField

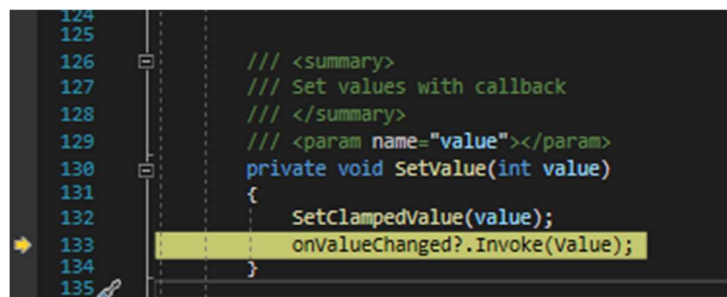
Далее с помощью метода `Mathf.Clamp` значение фильтруется и изменяется на максимально возможное:

A screenshot of a code editor showing the implementation of the `SetClampedValue` method. It takes an `int value` and clamps it between `minValue` and `maxValue` using `Mathf.Clamp`. It then checks if the clamped value is different from the current `m_value` and if the field is interactable. If both conditions are met, it updates `inputField.text` to the string representation of the clamped value. The right side of the image shows a Unity Inspector with variables `_value` (1200), `m_value` (999), and `maxValue` (999).

```
private void SetClampedValue(int value)
{
    try
    {
        m_value = Mathf.Clamp(_value, minValue, maxValue);
        if (inc.interactable == m_value != maxValue;
        if (dec.interactable == m_value != minv
        inputField.text = m_value.ToString();
    }
}
```

Рисунок 28 - Фильтрация значения внесенного в InputField

После вызывается объект типа делегата “OnValueChanged”:

A screenshot of a code editor showing the `SetFieldValue` method. It calls `SetClampedValue(value);` and then triggers the `onValueChanged` event using `onValueChanged?.Invoke(value);`. The left side of the image shows line numbers 124 through 135.

```
124
125
126 /// <summary>
127 /// Set values with callback
128 /// </summary>
129 /// <param name="value"></param>
130 private void SetFieldValue(int value)
131 {
132     SetClampedValue(value);
133     onValueChanged?.Invoke(value);
134 }
135
```

Рисунок 29 - Вызов события на изменения параметра

Нужное значения помещается в List со смещением, если оно двухбайтное и передаётся в метод, который работает в новом потоке:

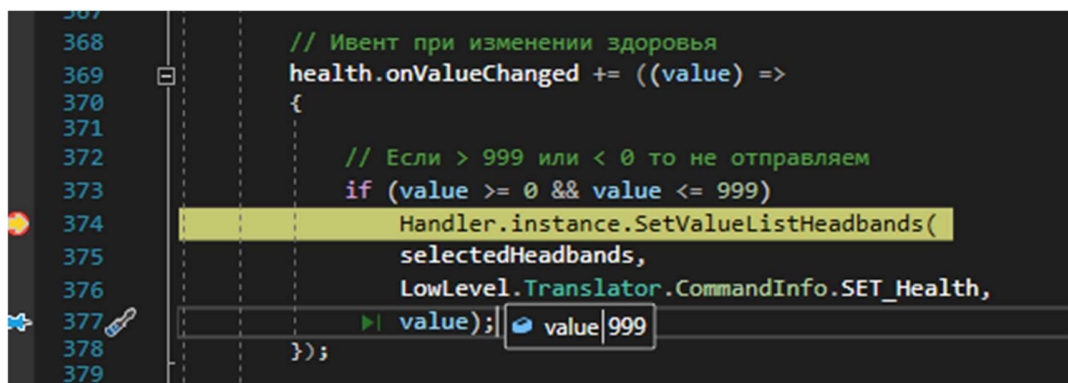


Рисунок 30 - Обработка события на изменения параметра

Значение смещается и записывается в List.

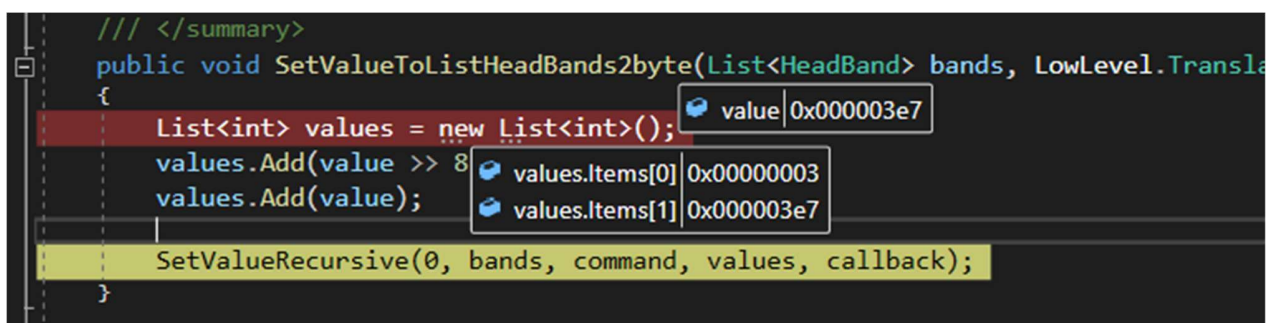


Рисунок 31 - Смещение и запись параметра в List

Значение записывается в буфер обмена.

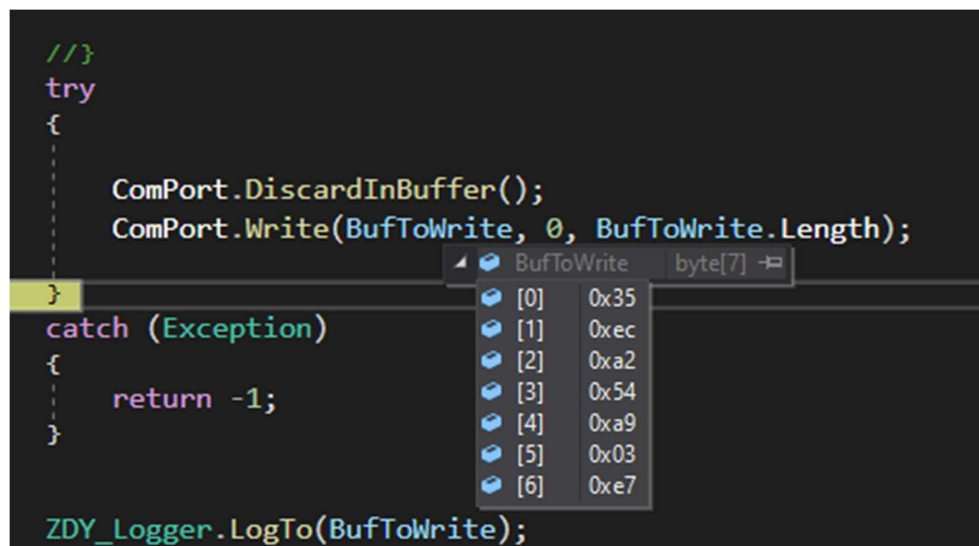


Рисунок 32 - Запись в буфер

Приходит ответ от радиобазы.

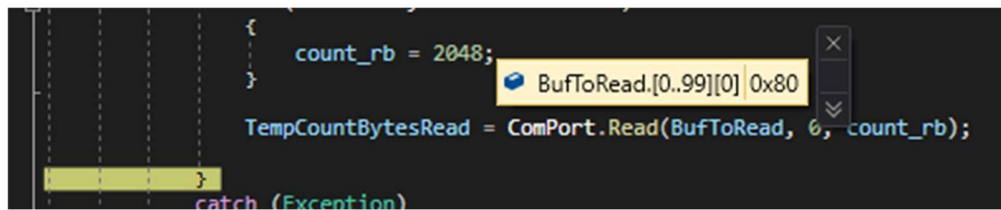


Рисунок 33 - Ответ от радиобазы

После отправки возвращается ответ. Согласно протоколу, ответ 0x80 является успешным, поэтому внесенные изменения применяются для объекта повязки:

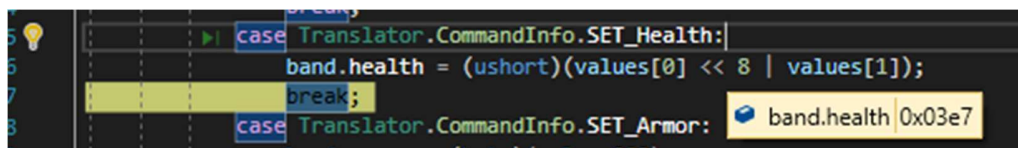


Рисунок 34 - Изменение параметров внутри объекта

Далее результат передается через Callback в основной поток и там уже обновляется UI игрока.

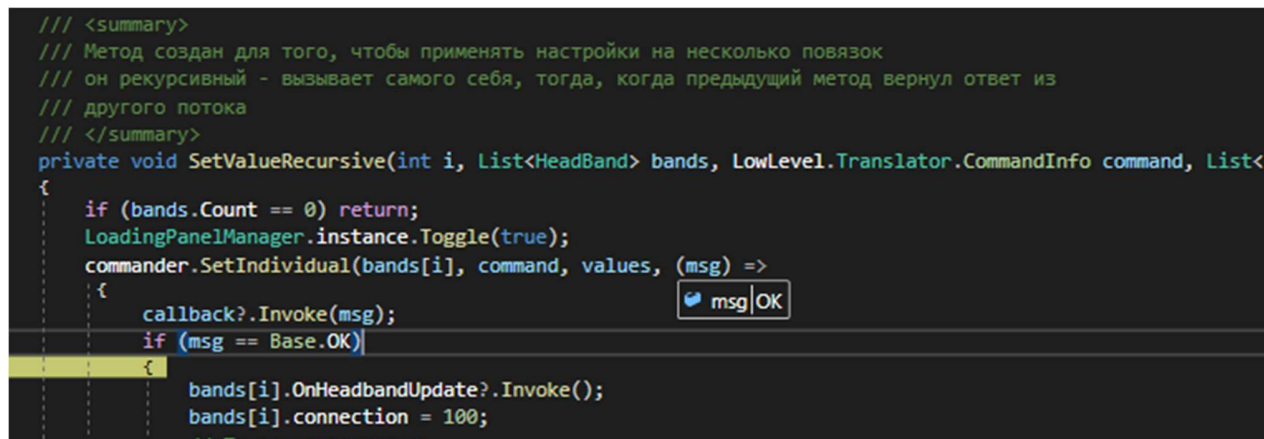


Рисунок 35 - Обработка результата

Тестирование работоспособности отправки команд прошло успешно. Далее следовало проверить взаимодействие программного обеспечения с сервером. Для проверки необходимо сделать следующие шаги:

- 1) Авторизоваться в тестовом клубе.
- 2) Создать нового игрока.

3) Изменить дополнительную информацию о нём.

На рисунке 36 представлено меню ввода данных для авторизации

Рисунок 36 - Ввод данных для авторизации клуба

Если введенные данные неверные.

Рисунок 37 - Результат ввода не верных данных

Если введенные данные верны, программа автоматически открывает меню клуба.



Рисунок 38 - Меню клуба, при вводе верных данных

Авторизация прошла проверку успешно. Далее работоспособность добавления нового игрока. На рисунке 39 представлено меню добавления нового игрока клуба.

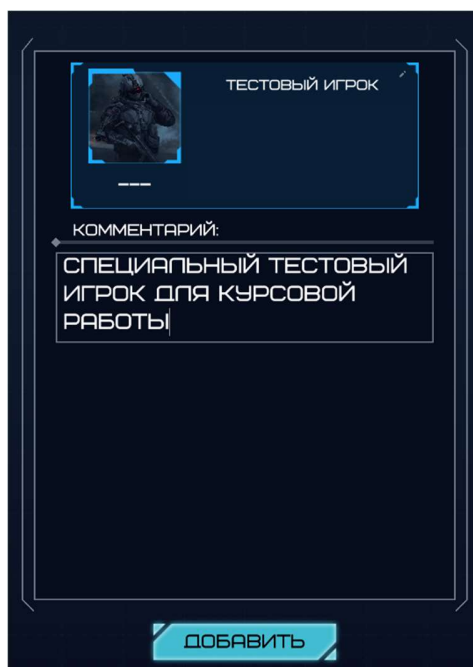


Рисунок 39 - Поля для создания нового игрока

Тестовый игрок был успешно создан и добавлен в список уже существующих, также автоматически открылось меню редактирования дополнительной информации об игроке.



Рисунок 40 - меню редактирования информации об игроке

После ввода дополнительной информации и применения изменений не создалось уведомления об ошибке, но для того, чтобы убедиться в работоспособности, можно обратиться к активной отладке и проверить ответ от сервера. На рисунке 41 представлены заполненные поля информации об игроке.

Рисунок 41 - изменения информации об игроке

Как видно на рисунке 42, сервер ответил положительно, что говорит о верной работе алгоритма.

```

public void ApplySettings()
{
    WebManager.EditPlayer(ForumClub.ID_CLUB.ToString(), ForumClub.CLUB_LOGIN, ForumClub.CLUB_PASSWORD, currentUsername.text, fioField.text, cityField.text, emailField.text, phoneField.text, bDateField.text, sexStepper.Value.ToString(), dopInfo.text, currentTeam.ToString(), Convert.ToInt32(isPassivNow).ToString(), (msg) =>
    {
        try
        {
            var answer = JsonUtility.FromJson<WebManager.WebAnswer>(msg);
            if (answer != null && answer.status == "ok")
            {
                #region FillPlayersOptions
            }
        }
    }
}

```

Рисунок 42 - Ответ от сервера на запрос об изменении информации игрока

Таким образом в данном пункте были проведены отладка и тестирования, которые показали верную работоспособность приложений.

Заключение

В ходе выполнения курсового проекта была проанализирована объектная область, изучены протоколы отправки данных на устройства и сервер. Сформулировано техническое задание, по которому были разработаны 2 приложения на разные платформы с идентичным функционалом. В основные части приложения входит конфигуратор с возможностью взаимодействия с физическим оборудованием и меню клуба с возможностью отправки, и приёма данных с сервера. Разработаны сцена и интерактивные UI объекты для разных типов устройств. Также была создана дополнительная библиотека под Android платформу, удобный и понятный пользовательский интерфейс и оптимизированная автоматизированная система подключения к радиобазе.

Программное обеспечение было написано на языке C# на игровом движке Unity, с вставками плагина, разработанного на языке программирования Java.

Программы протестированы на работоспособность и пригодность для неопытного пользователя, который может ввести не верные данные.

Список литературы

1. Методические указания к выполнению лабораторных работ по курсу «Технология разработки программного обеспечения». Составитель к.т.н., доцент кафедры САПР Эпп В. В.
2. Васильев А. С#. Объектно-ориентированное программирование / Алексей Васильев. – М.: Питер, 2012. – 320 с.
3. LaserStat web - <https://laserstat.pro/>
4. HABR bluetoth low energy - <https://habr.com/ru/articles/532298/>
5. Unity и С#. Геймдев от идеи до реализации. 2-е изд. Бонд Джереми Гибсон

Приложения

Приложение 1 Commader.cs

```
public class Commander : ICommander
{
    Translator MainTranslator = new Translator();
    int ComPortNumber = -1;

    public void SetIndividual(HeadBand band, Translator.CommandInfo command, List<int>
values, Action<Base> callback = null, Priority priority = Priority.Default)
    {
        var go = WaitQueue(priority);

        if (!go)
        {
            callback?.Invoke(Base.Busy);
            return;
        }

        Response res;

        Join = true;

        string valueStr = "";
        if (values != null)
        {
            for (int i = 0; i < values.Count; i++)
            {
                valueStr += values[i].ToString() + "-";
            }
        }

        //UnityEngine.Debug.Log("Отправляю запрос на команду " + command + " для id " +
band.id + " Со значением " + valueStr);
        res = MainTranslator.SendCommand(command, band.uid, values, 2000);

        if (res != null && res.data.Count == 12)
        {
            //UnityEngine.Debug.Log("Это пакет на статистику от " + res.data[0]);
        }
    }
}
```

```

        MainTranslator.ComPort.DiscardInBuffer();
        MainTranslator.ComPort.DiscardOutBuffer();
        res = MainTranslator.SendCommand(command, band.uid, values, 2000);
    }

    if (res.status < 0)
    {
        callback?.Invoke(Base.BaseConnectionFail);

    }
    else if (res.command == CommandFromInfo.Timeout)
    {
        //band.connection = 0;
        //UnityEngine.Debug.Log("timeout");
        callback?.Invoke(Base.Fail);
    }
    else if (res.command == CommandFromInfo.OK)
    {
        ChangeSettings(band, command, values);
        //UnityEngine.Debug.Log("OK");
        callback?.Invoke(Base.OK);
    }
    else if (res.command == CommandFromInfo.UndefinedCommand)
    {
        ZDY.Logger.ZDY_Logger.Log("Get Undefined command");
        callback?.Invoke(Base.UndefinedCommand);
    }
    else if(res.command == CommandFromInfo.UIDnotConnected)
    {
        callback?.Invoke(Base.UIDUndefined);
    }
    else
    {
        //if(values != null && values.Count > 0)
        //    UnityEngine.Debug.Log(values[0]);
        callback?.Invoke(Base.Dunno);
    }
    //UnityEngine.Debug.Log("Вышли из SET INDIVIDUAL");
    Join = false;
    //UnityEngine.Debug.Log("----- вышли из SetIndividual -----");
    EndSelfQueue(priority);
}

```

#region Query alghoritms

```

private static bool Join = false;

/// <summary>
/// Флаг, который запрещает пользоваться компортом,
/// когда он занят для исполнения другого метода
/// </summary>
static volatile bool ahShitHereWeGoAgain = false;

/// <summary>
/// Последний прибывший запрос с пк
/// </summary>
static volatile int lastQuery = 0;

static volatile int lastQueryHightPriority = 0;

/// <summary>
/// Тот, который на данный момент выполняется
/// </summary>
static volatile int currentQuery = 0;

static volatile int currentQueryHightPriority = 0;

private int COUNT_ERROR_WAIT_QUEUE = 6000;
/// <summary>
/// --Пока хз работает или нет, нужно проверить--
/// Отлично работает, метод вызывается в других методах,
/// чтобы занять очередь на отправку и принятие
///
/// Чтобы освободить очередь используй метод ниже
/// </summary>
private bool WaitQueue(Priority priority = Priority.Default)
{
    if(priority == Priority.Default)
    {
        // Присваиваем методу номер в очереди
        int thisQuery = lastQuery;
        //UnityEngine.Debug.Log($"зашёл в очередь с номером {thisQuery}, теперь
последний в очереди {lastQuery}" +
        //    $" с приоритетом default и {lastQueryHightPriority} с повышенным
приоритетом");
        // Показываем, что очередь расширилась
        lastQuery++;
    }
}

```



```

// Вдруг я где-то забыл закончить очередь
int count_error_check = 0;

// ждет своей очереди ...
while ((thisQuery != currentQuery) || (lastQueryHightPriority != 0))
{

    Thread.Sleep(10);
    count_error_check++;
    if (count_error_check >= COUNT_ERROR_WAIT_QUEUE)
    {
        UnityEngine.Debug.LogError("Кручусь в очереди уже минуту, где-то видимо
забыл закончить очередь");
        break;
    }
    continue;
}
//UnityEngine.Debug.Log($"Захожу в метод с номером {thisQuery} и приоритетом
default");
ahShitHereWeGoAgain = true;
return true;
}

////////////////////////////////////
////////////////////////////////////
////////////////////////////////////
else if (priority == Priority.Low)
{
    //UnityEngine.Debug.Log($"Захожу в очередь с номером {lastQuery} и
приоритетом LOW");
    // Если в очередях никого нет, то занимаем очередь
    if (lastQuery == 0 && lastQueryHightPriority == 0)
    {

        lastQuery++;
        ahShitHereWeGoAgain = true;
        //UnityEngine.Debug.Log($"Захожу в метод потому что очереди свободна");
        return true;
    }
    // Если в одной из очередей кто-то стоит, то ну его на....
    else
    {
        //UnityEngine.Debug.Log($"не захожу в метод, потому что очередь занята");

```

```

        // Ну я х.й знает, тут происходит какая-то дичь,
        // иногда как будто callback не срабатывает и именно "автоматический
поиск/проверка" повязок прекращается
        // PS: Всё нормально, я пень, метод запроса на конфиг таггера был без
вхождения в очередь
        return false;
    }

}

```

```

////////////////////////////////////
////////////////////////////////////
////////////////////////////////////
else if (priority == Priority.Hight)
{
    //UnityEngine.Debug.Log($"Захожу в очередь с номером {lastQueryHightPriority} и
приоритетом Hight");
    // Присваиваем методу номер в очереди
    int thisQuery = lastQueryHightPriority;
    // Показываем, что очередь расширилась
    lastQueryHightPriority++;
    // Вдруг я где-то забыл закончить очередь
    int count_error_check = 0;

    // ждет своей очереди ...
    while ((thisQuery != currentQueryHightPriority) || ahShitHereWeGoAgain)
    {
        //ConsoleLog("Жду очередь с приоритетом HIGHT");
        Thread.Sleep(10);
        count_error_check++;
        if (count_error_check >= COUNT_ERROR_WAIT_QUEUE)
        {
            UnityEngine.Debug.Log("Кручусь в очереди уже 30 секунд, где-то видимо
забыл закончить очередь для приоритетных");
            break;
        }
        continue;
    }
    //ConsoleLog("Захожу в метод с приоритетом HIGHT");
}

```

```

        ahShitHereWeGoAgain = true;
        return true;

    } else
    {
        UnityEngine.Debug.Log("Что, как я сюда вообще попал?");
        return false;
    }

}

/// <summary>
/// То же самое что и сверху
/// Работает филигранно,
/// метод освобождает место в очереди и она вся двигается дальше
/// короче нужно вставлять этот метод в конце, чтобы начали работать другие методы
/// </summary>
private void EndSelfQueue(Priority priority = Priority.Default)
{
    if(priority == Priority.Default || priority == Priority.Low)
    {
        //UnityEngine.Debug.Log($"закончил метод с приоритетностью {priority}");
        currentQuery++;
        if (currentQuery == lastQuery)
        {
            lastQuery = 0;
            currentQuery = 0;
        }

        ahShitHereWeGoAgain = false;
    }
    else if(priority == Priority.Hight)
    {
        //UnityEngine.Debug.Log($"закончил метод с приоритетностью Hight");
        currentQueryHightPriority++;
        if (currentQueryHightPriority == lastQueryHightPriority)
        {
            lastQueryHightPriority = 0;
            currentQueryHightPriority = 0;
        }

        ahShitHereWeGoAgain = false;
    } else

```

```

    {
        UnityEngine.Debug.Log("Не знаю ,как я сюда попал!");
    }

}

// Время ожидания каждого не принятого пакета после последней статистики
int ERROR_PACKETS_TIME_AFTER_STATISTIC = 30;
private void EndSelfStatisticQueue(int error_packs)
{

    if (currentQuery+1 == lastQuery)
    {
        currentQuery++;
    }
    else
    {
        //UnityEngine.Debug.Log("А это у нас не последняя команда, которая должна
выполниться");

        for (int i = 0; i < 100; i++)
        {

            if (MainTranslator.ComPort.BytesToRead > 0)
            {
                MainTranslator.ComPort.DiscardInBuffer();

                MainTranslator.ComPort.DiscardOutBuffer();
            }
            Thread.Sleep(ERROR_PACKETS_TIME_AFTER_STATISTIC);

        }
        currentQuery++;
    }

    ahShitHereWeGoAgain = false;
}

#endregion
}

public enum Priority
{

```

```

Low,
Default,
Hight,
}

```

Приложение 2 Translator.cs

```

public class Translator
{
    public Response SendCommand(CommandInfo Command, UInt32 uid = 0, List<int> values =
null, int timeout = 1000)
    {
        // óääëÿåò äàííûå èç áóôôååðà
        //ComPort.DiscardInBuffer();

        int res = Opros_to(Command, uid, values);

        //Stopwatch timer = Stopwatch.StartNew();

        Response resp = new Response();
        if (res < 0)
        {
            resp.status = -4;
            return resp;
        }

        List<CommandFromInfo> CurrCommands = new List<CommandFromInfo>();
        switch (Command)
        {
            case CommandInfo.StartRoundPlayer:
            case CommandInfo.SET_Armor:
            case CommandInfo.SET_ArmorHitChance:
            case CommandInfo.SET_AutoRegenerationMin:
            case CommandInfo.SET_AutoRegenerationSec:
            case CommandInfo.SET_AutoRegenerationValue:
            case CommandInfo.SET_AutoRespawnMin:
            case CommandInfo.SET_AutoRespawnSec:
            case CommandInfo.SET_BackgroundGlow:
            case CommandInfo.SET_BleedingMin:
            case CommandInfo.SET_BleedingSec:
            case CommandInfo.SET_BleedingValue:
            case CommandInfo.SET_BrightnessLEDsHeadband:
            case CommandInfo.SET_Color:

```

```

case CommandInfo.SET_DeathSignalOffTime:
case CommandInfo.SET_DefeatFromTheir:
case CommandInfo.SET_DelayBeforeTaggerActivation:
case CommandInfo.SET_DelayTimeAfterPause:
case CommandInfo.SET_DisplayInversion:
case CommandInfo.SET_Health:
case CommandInfo.SET_Invulnerability:
case CommandInfo.SET_Life:
case CommandInfo.SET_Shock:
case CommandInfo.SET_UnlinkAllDevicesAfterDeath:
case CommandInfo.SET_Vibro:
case CommandInfo.Respawn:
case CommandInfo.PausePlayer:
case CommandInfo.PauseOff:
case CommandInfo.EndGamePlayer:
case CommandInfo.Set_MaxTaggersCount:
case CommandInfo.SET_Vibro_Zonality:

case CommandInfo.GET_HeadbandStatus:

    CurrCommands.Add(CommandFromInfo.OK);
    CurrCommands.Add(CommandFromInfo.UIDnotConnected);
    CurrCommands.Add(CommandFromInfo.UndefinedCommand);
    CurrCommands.Add(CommandFromInfo.Timeout);
    break;

case CommandInfo.GET_Statistics:
    CurrCommands.Add(CommandFromInfo.Timeout);
    break;

case CommandInfo.GET_HeadbandConfig:
    CurrCommands.Add(CommandFromInfo.Timeout);
    CurrCommands.Add(CommandFromInfo.ConfigHeadband);
    break;
case CommandInfo.GET_ListOfConnectedTaggers:
    CurrCommands.Add(CommandFromInfo.ListOfConnectedTaggers);
    break;
case CommandInfo.GET_TAGGER_CONFIG:
    CurrCommands.Add(CommandFromInfo.ConfigTagger);

    break;
case CommandInfo.GET_Radiobase_:
    CurrCommands.Add(CommandFromInfo.OK);
    CurrCommands.Add(CommandFromInfo.UndefinedCommand);

```

```

        break;
    case CommandInfo.SET_HeadbandConfig:
        CurrCommands.Add(CommandFromInfo.OK);
        CurrCommands.Add(CommandFromInfo.UIDnotConnected);
        CurrCommands.Add(CommandFromInfo.UndefinedCommand);
        CurrCommands.Add(CommandFromInfo.Timeout);
        CurrCommands.Add(CommandFromInfo.TaggerConnectionFailed);
        break;
    default:
        break;
}

if(CurrCommands.Count == 0)
{
    CurrCommands.Add(CommandFromInfo.OK);
    CurrCommands.Add(CommandFromInfo.UIDnotConnected);
    CurrCommands.Add(CommandFromInfo.UndefinedCommand);
    CurrCommands.Add(CommandFromInfo.Timeout);
}

if (CurrCommands.Count > 0)
{
    resp = OproS_From(CurrCommands, timeout: timeout);
    //timer.Stop();
    //TimeSpan timespan = timer.Elapsed;

    //string timeDebugResult = String.Format("{0:00}:{1:00}:{2:00}", timespan.Minutes,
timespan.Seconds, timespan.Milliseconds / 10);
    //UnityEngine.Debug.Log(timeDebugResult);
}

return resp;
}

public int OproS_to(CommandInfo Command, UInt32 uid = 0, List<int> values = null)
{
    byte bCommand = (byte)Command;
    //Áóôåð äëÿ îðîðàâêè êîîàíû
    byte[] BufToWrite;

    //îàìÿîü äëÿ îîäîîîâêè èîîðîàîèè äëÿ îðîðàâêè
    MemoryStream DataMem = null;

    //Êîîîîîî äëÿ çàèèñè áàéò á
    //îàìÿîü äëÿ îîäîîîâêè èîîðîàîèè äëÿ îðîðàâêè
    BinaryWriter DataWriter = null;

```

```

//Άϊòîâèì ìàìÿòü äëÿ çàìèñè èñîáíäü äëÿ îòïðààêè à òðáíñëÿòïð
DataMem = new MemoryStream();

DataWriter = new BinaryWriter(DataMem);

//Ñòàððòîâàÿ èñâèàòöëÿ (îäé áàéò)

byte l_value;

DataWriter.Write(bComand);
// Åñèè óèàçàí uid ñâÿçèè, òí çàìèññüâààì äñî â áóóâð
if (uid > 0)
{
    l_value = (byte)(uid >> 24);
    DataWriter.Write(l_value);
    l_value = (byte)(uid >> 16);
    DataWriter.Write(l_value);
    l_value = (byte)(uid >> 8);
    DataWriter.Write(l_value);
    l_value = (byte)uid;
    DataWriter.Write(l_value);
}

// Åñèè çààáíü çá÷áíèÿ, òí çàìèññüâààì èð â áóóâð
if (values != null)
{
    for (int loop1 = 0; loop1 < values.Count; loop1++)
    {
        DataWriter.Write((byte)values[loop1]);
    }
}

DataWriter.Flush();

BufToWrite = DataMem.ToArray();

//try
//{
//    string opos_ = "";
//    for (int i = 0; i < BufToWrite.Length; i++)
//    {
//        opos_ += BufToWrite[i].ToString("X") + " ";
//    }
//}

```



```

// }
// UnityEngine.Debug.Log("îïðàâëÿþ - " +opros_);
//}
//catch
//{

//}
try
{

    ComPort.DiscardInBuffer();
    ComPort.Write(BufToWrite, 0, BufToWrite.Length);

}
catch (Exception)
{
    return -1;
}

ZDY_Logger.LogTo(BufToWrite);
//LogTo.Add(BitConverter.ToString(BufToWrite));
return 1;
}
public Response OproS_From(List<CommandFromInfo> CurrCommands, int timeout = 1000)
{
    Response resp = new Response();

    byte[] BufToRead = new byte[2048];

    //byte[] BufToWrite = p_BufToWrite as byte[];
    //îàìÿòü äëÿ ïïããîðîâêê èíîðîëàöèè äëÿ îððàâëè
    MemoryStream DataMem = null;

    //Âðàìÿ ìà÷àèà ÷òàíèÿ
    //Ëññèüçóâîðîñÿ äëÿ òàèëàîòà
    DateTime startTimeRead;

    //Êññîðî çàèñè ààèò â
    //îàìÿòü äëÿ ïïããîðîâêê èíîðîëàöèè äëÿ îððàâëè
    BinaryReader DataReader = null;

    //Êññè÷àíîâîðîðîâèò ààèò
    int TempCountBytesRead = 0;

```

```

//Áàéò äëÿ ïðáíáðàçîâàíèé
//byte TempByte;

//Ãîðîâè ìàìÿòü äëÿ çàìèñë èíàíäü äëÿ îððààèè à èíððíèèäð
DataMem = new MemoryStream();
//Ãîðîâèñÿ ïðèÿòü çàãíèâîè

startTimeRead = DateTime.Now;

TimeSpan TimeOutRead;

//Òàéìàóò äëÿ âóõíà, áñèè ìè÷àñí íà ïðèõíàè à îðààò
//TimeOutRead = TimeSpan.FromMilliseconds(50.0);

TimeOutRead = TimeSpan.FromMilliseconds((double)timeout);

//Ôèàã òàéìàóò äëÿ îíí÷àíèÿ ààñòàèè ñ òðàíñÿòîðî
bool is_FailTimeout = false;

//Áñèè íà âñâà ààéòîâ
try
{// 1
    while (ComPort.BytesToRead <= 0)
    {
        //Æäâî ñââà ààéòü
        Thread.Sleep(5);
        if (TimeOutRead <= DateTime.Now - startTimeRead)
        {
            //Íà äèààèèñü îðààò è âóøèè
            is_FailTimeout = true;
            break;
        }
    }
    //var testTime = DateTime.Now - startTimeRead;
    //Debug.Log($"Test Timeout = {timeout} wait for = {testTime}");
}// 1
catch (Exception ex)
{
    resp.status = -1;
    ZDY_Logger.LogFrom("=== îðèáàè ïðè äèààíèè ààíüð ===");
    //LogFrom.Add("=== îðèáàè ïðè äèààíèè ààíüð ===");
    //UnityEngine.Debug.Log(ex);
    return resp;
}

```

```

//Ãñèè è è ìäíñî áàéòà íå äîæääèèñü
if (is_FailTimeout && ComPort.BytesToRead <= 0)
{
    resp.status = -2;
    ZDY_Logger.LogFrom("=== íå ïðèðëî äàííüð (timeout) " + timeout + " ===");
    //LogFrom.Add("=== íå ïðèðëî äàííüð (timeout) " + timeout + " ===");
    return resp;
}
/*
// ïðîäðîäè, ïðèðîäÿò è è àùå áàéòó
try
{
    int thisCount = ComPort.BytesToRead;
    Thread.Sleep(WAIT_TO_BASE_WRITE);
    for (int i = 0; i < 200; i++)
    {
        // Ãñèè ÷èñëî áàéò íå èçìåíèñü (âñ., ÷î íîæîí áóî - çàèñëàèñü áàçî)
        if (thisCount == ComPort.BytesToRead) break;
        thisCount = ComPort.BytesToRead;
        Thread.Sleep(WAIT_TO_BASE_WRITE);
    }
}
catch {}
*/

try
{
    int count_rb = ComPort.BytesToRead;
    if (ComPort.BytesToRead > 2048)
    {
        count_rb = 2048;
    }

    TempCountBytesRead = ComPort.Read(BufToRead, 0, count_rb);
}
catch (Exception)
{
    //ClearBufferWaiting(TempCountBytesRead, BufToRead);
    /*byte[] bc = new byte[ComPort.BytesToRead];
    ComPort.Read(bc, 0, ComPort.BytesToRead);*/
    resp.status = -3;
    ZDY_Logger.LogFrom("=== îðåáèè ïðè ÷òáíèè ñ comport ===");
    //LogFrom.Add("=== îðåáèè ïðè ÷òáíèè ñ comport ===");
}

```

```

        return resp;
    }

    //try
    //{
    //    string opos_ = "";
    //    for (int i = 0; i < BufToRead.Length; i++)
    //    {
    //        opos_ += BufToRead[i].ToString("X") + " ";
    //    }
    //    UnityEngine.Debug.Log("İöèìèàp - " + opos_);
    //}
    //catch
    //{

    //}

    DataMem = new MemoryStream(BufToRead);
    DataReader = new BinaryReader(DataMem);

    List<byte> massbytes = new List<byte>();
    for (int loop1 = 0; loop1 < TempCountBytesRead; loop1++)
    {
        massbytes.Add(BufToRead[loop1]);
    }

    resp.status = 1;

    byte combyte = DataReader.ReadByte();
    resp.data.Add(combyte);

    for (int loop1 = 0; loop1 < CurrCommands.Count; loop1++)
    {
        if (combyte == (byte)CurrCommands[loop1])
        {
            resp.command = CurrCommands[loop1];
        }
    }
    for (int loop1 = 1; loop1 < TempCountBytesRead; loop1++)
    {
        resp.data.Add(DataReader.ReadByte());
    }
    try

```

```

    {
        if(massbytes != null)
        {
            ZDY_Logger.LogFrom(massbytes.ToArray());
        }

        //LogFrom.Add(BitConverter.ToString(massbytes.ToArray()));
    } catch
    {
    }

    if (resp.data.Count == 1 && resp.data[0] == (byte)CommandFromInfo.PLZwait)
    {
        resp = Opros_From(CurrCommands, timeout);
    }

    return resp;
}

```

Приложение 3 WebManager.cs

```

public class WebManager : MonoBehaviour
{
    public static WebManager instance;

    [SerializeField] private WebSender webSender;

    private void Awake()
    {
        if (instance != null) Destroy(instance);
        instance = this;
    }

    public static void RegisterClub(string login, string hash, string name, string email, string
contacts, string city, Action<string> callback)
    {
        string post = $"act={ACT_REGISTER_CLUB}" +
            $"&login={login}" +
            $"&hash={hash}" +

```

```

        $"&name={name}" +
        $"&email={email}" +
        $"&contacts={contacts}" +
        $"&city={city}" +
        $"&v={1}";

        instance.webSender.PostQueryOnServer(post, LINK_POST, callback);

    }

    public const string ACT_EDIT_PLAYER = "15";

    public static void EditPlayer(string idclub,
        string login, string hash, string idPlayer, string name, string fio, string city, string email,
        string phone, string bdate, string sex, string dopinfo, string currteam, string status,
        Action<string> callback)
    {
        string post = $"act={ACT_EDIT_PLAYER}" +
            $"&idclub={idclub}" +
            $"&login={login}" +
            $"&hash={hash}" +
            $"&idplayer={idPlayer}" +
            $"&name={name}" +
            $"&fio={fio}" +
            $"&city={city}" +
            $"&email={email}" +
            $"&phone={phone}" +
            $"&bdate={bdate}" +
            $"&sex={sex}" +
            $"&dopinfo={dopinfo}" +
            $"&currteam={currteam}" +
            $"&status={status}" +

            $"&v={1}";

        instance.webSender.PostQueryOnServer(post, LINK_POST, callback);
    }

    public const string ACT_EDIT_TEAM = "16";

```

```

public static void EditTeam(string idclub, string login, string hash,
    string idteam, string name, string dopinfo, Action<string> callback)
{
    string post = $"act={ACT_EDIT_TEAM}" +
        $"&idclub={idclub}" +
        $"&login={login}" +
        $"&hash={hash}" +
        $"&idteam={idteam}" +
        $"&name={name}" +
        $"&dopinfo={dopinfo}" +

        $"&v={1}";

    instance.webSender.PostQueryOnServer(post, LINK_POST, callback);
}

```

```

public const string ACT_ADD_PLAYER = "13";

public static void AddPlayer(string idclub,
    string login, string hash, string name, string dopinfo, Action<string> callback)
{
    string post = $"act={ACT_ADD_PLAYER}" +
        $"&idclub={idclub}" +
        $"&login={login}" +
        $"&hash={hash}" +
        $"&name={name}" +
        $"&dopinfo={dopinfo}" +
        $"&v={API_VERSION}";

    instance.webSender.PostQueryOnServer(post, LINK_POST, callback);
}

```

```

public const string ACT_ADD_TEAM = "12";

public static void AddTeam(string idclub,
    string login, string hash, string name, string dopinfo, Action<string> callback)
{
    string post = $"act={ACT_ADD_TEAM}" +

```

```

        $"&idclub={idclub}" +
        $"&login={login}" +
        $"&hash={hash}" +
        $"&name={name}" +
        $"&dopinfo={dopinfo}" +
        $"&v={1}";

instance.webSender.PostQueryOnServer(post, LINK_POST, callback);
}

```

```

public const string ACT_DELETE_PLAYER = "17";

public static void DeletePlayer(string idclub,
    string login, string hash, string idplayer, Action<string> callback)
{
    string post = $"act={ACT_DELETE_PLAYER}" +
        $"&idclub={idclub}" +
        $"&login={login}" +
        $"&hash={hash}" +
        $"&idplayer={idplayer}" +
        $"&v={API_VERSION}";

    instance.webSender.PostQueryOnServer(post, LINK_POST, callback);
}

```

```

public const string ACT_DELETE_TEAM = "18";

public static void DeleteTeam(string idclub,
    string login, string hash, string idteam, Action<string> callback)
{
    string post = $"act={ACT_DELETE_TEAM}" +
        $"&idclub={idclub}" +
        $"&login={login}" +
        $"&hash={hash}" +
        $"&idteam={idteam}" +
        $"&v={API_VERSION}";
}

```



```

        instance.webSender.PostQueryOnServer(post, LINK_POST, callback);
    }

    //"act=18&idclub=289&login=login&hash=4297F44B13955235245B2497399D7A93&idteam=444&
    v=1"
        //
    act=18&idclub=289&login=login&hash=4297F44B13955235245B2497399D7A93&idteam=443&v=1
        //
    act=17&idclub=289&login=login&hash=4297F44B13955235245B2497399D7A93&idplayer=5518&v
    =1

        //
    "act=17&idclub=289&login=login&hash=4297F44B13955235245B2497399D7A93&idplayer=6424&
    v=1"
        //
    "act=18&idclub=289&login=login&hash=4297F44B13955235245B2497399D7A93&idteam=411&v=
    1"

    #endregion

```

Приложение 4 WebSender.cs

```

public class WebSender : MonoBehaviour
{

    [SerializeField] private string[] urlsToCheckConnection;

    private const string lsdURL = "https://laserstat.pro/";

    public void PostQueryOnServer(string postData, string url, Action<string> callback)
    {
        DevLog.requestCounter++;
        StartCoroutine(Download(postData, url, (data) =>
        {
            DevLog.answerCounter++;
            if (data != null)
            {
                string result = System.Text.Encoding.UTF8.GetString(data);
                ServerRequestReader.RequestLog(result);
                callback.Invoke(result);
            } else

```

```

        {
            callback.Invoke(null);
        }

    });

}

public void GetFile(string url, Action<byte[]> callback)
{
    DevLog.requestCounter++;
    StartCoroutine(DownloadFile(url, (data) =>
    {
        DevLog.answerCounter++;
        if (data != null)
        {
            //string result = System.Text.Encoding.UTF8.GetString(data);
            callback.Invoke(data);
        }
        else
        {
            callback.Invoke(null);
        }
    }

    ));
}

```

```

IEnumerator Download(string postData, string url, Action<byte[]> callback)
{

    UnityWebRequest www = new UnityWebRequest(url);
    www.uploadHandler = new UploadHandlerRaw(Encoding.UTF8.GetBytes(postData));
    www.downloadHandler = new DownloadHandlerBuffer();
    www.uploadHandler.contentType = "application/x-www-form-urlencoded";
    www.method = UnityWebRequest.kHttpVerbPOST;

    yield return www.SendWebRequest();

    if (www.result != UnityWebRequest.Result.Success)
    {
        Debug.Log(www.error);
        callback.Invoke(null);
    }
}

```

```

    }
    else
    {
        callback.Invoke(www.downloadHandler.data);
    }

    www.Dispose();
}

```

```
IEnumerator DownloadFile(string url, Action<byte[]> callback)
```

```

{

    UnityWebRequest www = new UnityWebRequest(url);
    www.uploadHandler = new UploadHandlerRaw(Encoding.UTF8.GetBytes(url));
    www.downloadHandler = new DownloadHandlerBuffer();
    www.uploadHandler.contentType = "application/x-www-form-urlencoded";
    www.method = UnityWebRequest.kHttpVerbGET;

    yield return www.SendWebRequest();

    if (www.result != UnityWebRequest.Result.Success)
    {
        Debug.Log(www.error);
        callback.Invoke(null);
    }
    else
    {
        callback.Invoke(www.downloadHandler.data);
    }
    www.Dispose();
}

```

```

public void CheckInternetConnection(Action<ConnectionResult> callback, string link =
IsdURL)
{
    StartCoroutine(CheckInternetConnectionCoroutine(callback, link));
}

```

```

IEnumerator CheckInternetConnectionCoroutine(Action<ConnectionResult> callback,
string link = IsdURL)
{

```

```

// Åñëë ñãðããð LSD îòããðëë, òí ãñòü è èíòãðíãð è ñãðããð íã êðàøíóëñŸ
{
    UnityWebRequest www = UnityWebRequest.Get(link);

    // Try fix SSL CA certificate error
    var cert = new ForceAcceptAll();
    www.certificateHandler = cert;

    yield return www.SendWebRequest();

    if (www.result == UnityWebRequest.Result.Success)
    {
        callback.Invoke(ConnectionResult.success);
        www.Dispose();
        yield break;
    }
    else
    {
        Debug.Log($"UnityWebRequest result = {www.result}");
    }

    cert?.Dispose();

    www.Dispose();
}

// Ñþãà ìü ññãããã, åñëë lsd íã îòããðëë
foreach(var url in urlsToCheckConnection)
{
    UnityWebRequest www = UnityWebRequest.Get(url);

    yield return www.SendWebRequest();

    if (www.result == UnityWebRequest.Result.Success)
    {
        callback.Invoke(ConnectionResult.LsdServerCrushed);
        //callback.Invoke(ConnectionResult.success);
        www.Dispose();
        yield break;
    }
}

```

```

        www.Dispose();
    }

    // Ñpää määäää, äñëè èè îäèí èç ñàéòîâ íâ îòääòèë
    callback.Invoke(ConnectionResult.noInternet);

}

public enum ConnectionResult
{
    noInternet,
    IsdServerCrushed,
    success,
}

public class ForceAcceptAll : CertificateHandler
{
    protected override bool ValidateCertificate(byte[] certificateData)
    {
        return true;
    }
}
}

```