

МИНОБРНАУКИ РОССИИ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ  
ВЫСШЕГО ОБРАЗОВАНИЯ  
“ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ”

Факультет компьютерных наук

Кафедра программирования и информационных технологий

Стратегическая мобильная игра “Botegy”

Курсовой проект

09.03.04 Программная инженерия

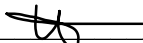
Информационные системы и сетевые технологии

Зав. кафедрой \_\_\_\_\_ С.Д. Махортов, д.ф.- м.н., доцент \_\_\_\_\_.20\_\_

Обучающийся \_\_\_\_\_ И.С. Шаталов, 3 курс, д/о

Обучающийся \_\_\_\_\_ Н.Ю. Савина, 3 курс, д/о

Руководитель \_\_\_\_\_ В.С. Тарасов, ст. преподаватель

Руководитель \_\_\_\_\_  И.В. Клейменов, ассистент

Воронеж 2022

## Содержание

|  |    |
|--|----|
| Содержание .....                                 | 2  |
| Введение .....                                   | 5  |
| 1 Постановка задачи.....                         | 7  |
| 1.1 Требования к разрабатываемой системе .....   | 7  |
| 1.1.1 Функциональные требования.....             | 7  |
| 1.1.2 Технические требования .....               | 7  |
| 1.2 Требования к интерфейсу .....                | 8  |
| 1.3 Задачи, решаемые в процессе разработки ..... | 8  |
| 2 Анализ предметной области .....                | 10 |
| 2.1 Анализ рынка стратегических игр .....        | 10 |
| 2.2 Обзор аналогов .....                         | 11 |
| 2.2.1 Gladiabots.....                            | 11 |
| 2.2.2 DumbBots.....                              | 13 |
| 3 Реализация .....                               | 16 |
| 3.1 Средства реализации .....                    | 16 |
| 3.2 Входные-выходные данные (IDEF0) .....        | 17 |
| 3.3 Диаграмма вариантов использования.....       | 17 |
| 3.3.1 Диаграмма Игрока .....                     | 17 |
| 3.3.2 Диаграмма Модератора .....                 | 18 |
| 3.3.3 Диаграмма Наблюдателя.....                 | 19 |
| 3.4 Диаграмма состояний.....                     | 20 |
| 3.5 Диаграмма потоков данных .....               | 21 |
| 3.6 Структурная схема приложения .....           | 23 |
| 3.7 Реализация серверной части приложения .....  | 24 |

|  |    |
|--|----|
| 3.7.1 Схема базы данных.....                           | 24 |
| 3.7.2 Диаграмма классов .....                          | 24 |
| 3.7.2.1 Диаграмма классов сущностей.....               | 25 |
| 3.7.2.2 Диаграмма классов репозиториев .....           | 25 |
| 3.7.2.3 Диаграмма классов сервисов.....                | 26 |
| 3.7.2.4 Диаграмма классов контроллеров.....            | 26 |
| 3.7.2.5 Диаграмма классов-оберток .....                | 26 |
| 3.7.2.6 Диаграмма служебных классов .....              | 27 |
| 3.7.3 Архитектура серверной части приложения .....     | 27 |
| 3.7.4 Слой доступа к данным .....                      | 27 |
| 3.7.5 Слой контроллеров .....                          | 28 |
| 3.7.6 Слой бизнес-логики .....                         | 29 |
| 3.7.6.1 Механика игры.....                             | 30 |
| 3.8 Реализация клиентской части приложения .....       | 33 |
| 3.8.1 Макеты интерфейса .....                          | 33 |
| 3.8.2 Диаграмма классов .....                          | 46 |
| 3.8.2.1 Диаграмма классов пакета ButtonScript.....     | 46 |
| 3.8.2.2 Диаграмма классов пакета Dialog .....          | 47 |
| 3.8.2.3 Диаграмма классов пакета Entity .....          | 47 |
| 3.8.2.4 Диаграмма классов пакета Model.....            | 48 |
| 3.8.2.5 Диаграмма классов пакета AppSceneManager ..... | 49 |
| 3.8.2.6 Диаграмма классов пакета CodeParser.....       | 50 |
| 3.8.3 Игровые сцены.....                               | 52 |
| 3.8.4 Редактор кода бота .....                         | 53 |
| 3.9 Сценарии воронок .....                             | 55 |

|  |    |
|--|----|
| 3.10 Календарный план .....            | 57 |
| Заключение .....                       | 59 |
| Список использованных источников ..... | 60 |

## Введение

Издавна человечество старалось упростить свою жизнь. Большая часть сил была направлена на облегчение задач в различных сферах жизни. Огромным шагом в этом направлении является развитие и удешевление ЭВМ, появление и постоянный прогресс смартфонов. Практически у каждого современного человека есть мобильный телефон, у некоторых даже больше одного. Мобильные телефоны нынешнего поколения перестали быть просто средством общения и переросли в нечто большее. Они стали переносимыми мобильными компьютерами, открывающими своим владельцам очень много дополнительных возможностей – доступ к большому каталогу информации, прослушивание музыки, предоставляет разные формы общения, построение оптимального пути, проведение свободного времени в развлечениях и многое другое.

Постоянный прогресс высоких технологий и увеличение их возможностей привели к росту спроса на программы и приложения для этих самых технологий. Среди прочих направлений в этой области отдельное, особое место занимают программы на мобильные устройства. Среди них есть как необходимые для повседневной жизни, упрощающие быт и разные сферы жизнедеятельности, так и приложения – игры (играть/өгрэт/өйрэт – с татарского «обучать». Это форма обучения), без которых процесс обучения был бы сложнее. Чем объясняется популярность игр. Это обусловлено тем, что кому-то хочется скоротать время, кто-то проникается азартом достижения цели и преодоления препятствия, кому-то приятно иметь возможность заниматься тем, что интересно для него в жизни, где и когда угодно (симулятор футбола), кто-то хочет развивать логику или реакцию и так далее. Все эти причины способствуют популярности мобильных игр и стимулированию их к разработке. Наличие бесплатных платформ для создания игр, таких как Unity позволяют разработчикам воплощать в жизнь самые разнообразные свои идеи.

Целью данной работы является разработка мобильной игры под Android на платформе Unity. Особенности игры будут нетипичное косвенное участие пользователя в игровом процессе, а именно написание стратегии, которая будет «играть» за него, концепция PvP, то есть возможность поиграть против стратегий других людей, возможность для анализа эффективности стратегии посредством просмотра визуализации матча.

## **1 Постановка задачи**

Целью данного курсового проекта является разработка мобильного приложения – игры «Botegy». К разрабатываемому мобильному приложению выдвинуты следующие требования:

- Возможность создания стратегии поведения юнитов на поле боя при помощи интегрированного редактора кода;
- Возможность создания нескольких стратегий – ботов;
- Возможность редактирования созданных ботов;
- Возможность проведения матчей между ботами разных игроков;
- Возможность поиска и выбора бота соперника;
- Возможность просмотра хода матча.

### **1.1 Требования к разрабатываемой системе**

#### **1.1.1 Функциональные требования**

К разрабатываемому приложению выдвигаются следующие требования:

- Возможность регистрации новых учетных записей;
- Возможность авторизации зарегистрированных пользователей;
- Осуществление поиска ботов по названию;
- Осуществление создания новых ботов зарегистрированных пользователей, редактирование и удаление уже созданных;
- Вывод информации об истории матчей ботов пользователей;
- Вывод информации о созданных ботах пользователей;
- Вычисление результата матча между двумя ботами;
- Вывод хода матча между ботами;
- Осуществление редактирования данных учетной записи пользователя;
- Осуществление блокировки учетных записей и назначение пользователей модераторами.

#### **1.1.2 Технические требования**

Приложение должно обеспечивать:

- Авторизацию его пользователей посредством электронной почты и пароля, которые хранятся и идентифицируются внешней (по отношению к приложению) системой авторизации;
- Удаление ботов пользователя при удалении его учетной записи;
- Вычисление результатов и хода всех матчей на сервере для уменьшения нагрузки на клиент.

## **1.2 Требования к интерфейсу**

Интерфейс должен быть выполнен в единой для всех экранов цветовой гамме, едином стиле. Все надписи должны быть легко читаемы, все элементы управления должны быть выполнены в едином стиле, размере, должны выделяться на фоне содержимого экранов. Интерфейс должен корректно отображаться при изменении размера экрана. Интерфейс должен поддерживать портретную ориентацию экрана.

Интерфейс должен содержать необходимую для пользователя информацию: информацию о самом пользователе, информацию о пользовательских ботах, информацию о других пользователях, их ботах, матчах. Информация должна находиться в тех местах приложения, где она будет актуальна, то есть, например, при просмотре результатов матча не нужно отображать список ботов текущего пользователя.

## **1.3 Задачи, решаемые в процессе разработки**

Были поставлены следующие задачи:

- Анализ предметной области;
- Анализ аналогов;
- Постановка задачи;
- Разработка макетов интерфейса;
- Определение используемой платформы;
- Построение use case диаграмм;
- Создание доски Trello и репозитория GitHub;
- Написание технического задания;



- Реализация слоя доступа к БД;
- Реализация интерфейса;
- Построение UML диаграмм;
- Подключение swagger;
- Разработка модели кода бота;
- Создание сценариев воронок;
- Реализация парсера кода бота;
- Реализация модуля симуляции матчей ботов;
- Размещение backend-части и БД на хостинге;
- Реализация модуля авторизации;
- Описание процесса разработки и результата.

## **2 Анализ предметной области**

### **2.1 Анализ рынка стратегических игр**

В современном мире создание видеоигр является одним из наиболее крупных сегментов индустрии развлечений. Масштабы игровой индустрии сопоставимы, например, с киноиндустрией. А по скорости роста за последние пять лет индустрия видеоигр существенно ее опережала.

По степени влияния на потребителей и вовлеченности их в интерактивное окружение, предлагаемое видеоиграми, этот сегмент уже давно выделяется среди других видов развлечений.

Геймдев или разработку игр невозможно рассматривать обособленно от индустрии компьютерных игр в целом. Непосредственно создание игр – это только часть комплексной «экосистемы», обеспечивающей полный жизненный цикл производства, распространения и потребления таких сложных продуктов, как компьютерные игры.

Согласно исследованию My.Games, самый быстрорастущий сегмент рынка за период — мобильные игры. Мобильный гейминг сегодня стал более доступным, чем когда-либо, благодаря развитию технических возможностей, улучшению качества подключения мобильных устройств и доступности. Именно поэтому сегодня наблюдается стремительный рост популярности мобильных игр во всем мире.

Жанр стратегических игр не является самым популярным среди других жанров видеоигр. Тем не менее, исходя из данных Facebook Gaming's report, в среднем пользователи проводят за игрой длительное время – более 30 минут за сессию. Кроме того, большинство игроков предпочитают онлайн-стратегии и отмечают важность взаимодействия с другими пользователями. Одной из главных причин, почему люди играют в стратегии, – это возможность участия в соревновании с другими игроками, возможность показать свое превосходство в стратегическом мышлении.

Среди поджанров стратегий набирают популярность 4X игры, например, Civilization. Стратегии же, в которых есть возможность программирования

поведения игровых персонажей, являются одними из наиболее молодых среди стратегических игр и большинство игр поддерживает программирование поведения отдельных боевых единиц, но не интеллекта управления ими.

## **2.2 Обзор аналогов**

Приступая к разработке новой игры в определенном жанре, необходимо проанализировать проекты, уже существующие внутри него. Рассмотрим их достоинства, недостатки. Насколько они удобны в использовании, содержат ли необходимую функциональность. И на основе этого анализа сделаем выводы какие моменты нужно учесть при разработке своего приложения, а именно каким образом будет построен удобно для пользователя интерфейс, и какую функциональность будет иметь игра.

### **2.2.1 Gladiabots**

Наиболее известной игрой, позволяющей создавать ИИ и испытывать его в бою, является GLADIABOTS - AI Combat Arena. Gladiabots – это стратегия о боевых ботах, где игрок создает ИИ для своего отряда и отправляет его в бой.



Рисунок 1 - Внешний вид игры Gladiabots

Основная функциональность игры заключается в следующем:

- Создание своего ИИ и наблюдение, как боты выполняют его на арене;
- Создание и настройка своего отряда ботов;
- Игра в режиме одиночной кампании;
- Игра в режиме многопользовательской онлайн игры, в том числе: рейтинговый, обычный и частный матч;
- Наличие 3 совершенно разных режимов игры: устранение, доминирование и сбор;
- Участие в турнирах или создание их;
- Сражение с другими игроками даже если они не онлайн, так как присутствует асинхронный мультиплеер;
- Использование режима песочницы для проверки своих тактик;
- Сражение с равными по силам игроками благодаря системе рангов.

Преимуществами Gladiabots являются:

- Насыщенный игровой процесс, реализуемый наличием различных режимов игры, большим количеством боевых юнитов;
- Система рангов, позволяющая подбирать равных по навыкам соперников;
- Возможность проведения турниров.

К недостаткам приложения можно отнести:

- Достаточно высокий порог вхождения;
- Интерфейс, перегруженный иконками, кнопками, сильное разрастание «площади» кода при усложнении стратегии;
- Сокращение возможностей при создании ИИ в целях упрощения использования приложения в мобильной версии.

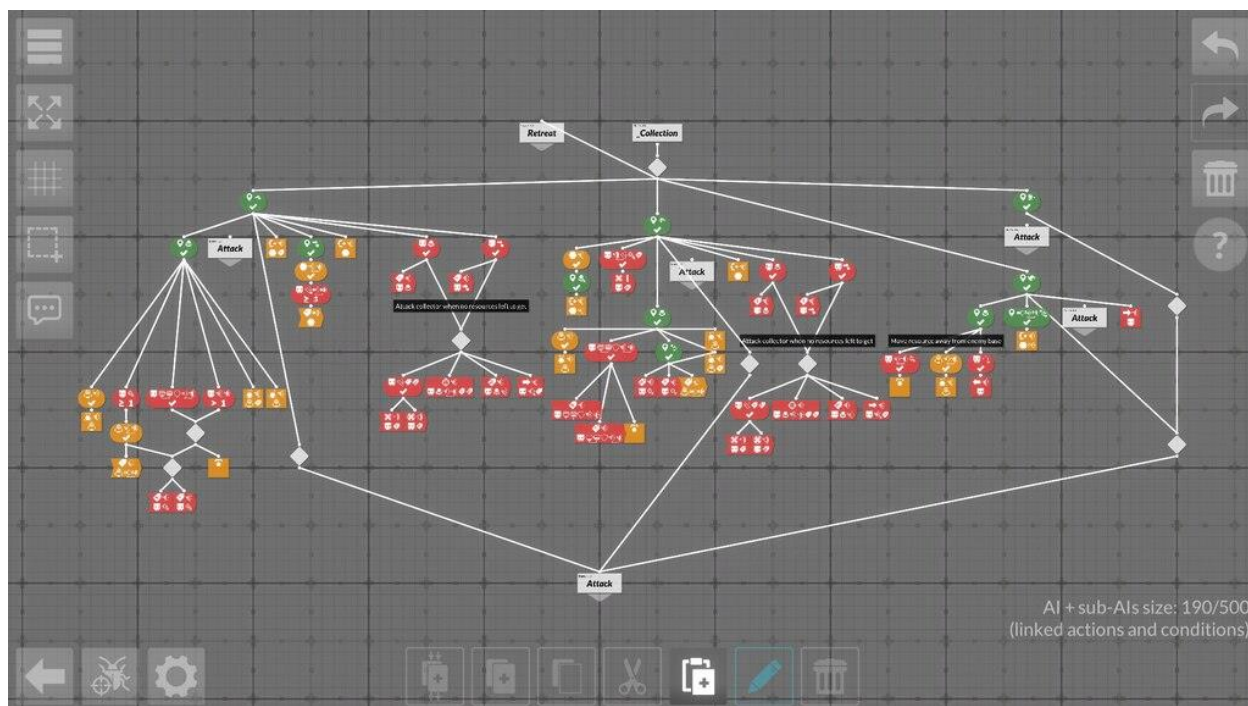


Рисунок 2 - Внешний вид интерфейса Gladiabots

### 2.2.2 DumbBots

DumbBots – игра, позволяющая игрокам программировать поведение ботов, которые потом используются для прохождения карт и уровней игры, состязаний с другими игроками.

Игра предоставляет:

- Встроенный редактор кода;
- Четыре режима игры;
- Возможность изменять внешность ботов;
- Более 20 карт для создания сценариев;
- Более 50 сценариев для прохождения;
- Возможность создания собственных сценариев;
- Возможность состязания с другими игроками;
- Возможность игры от первого лица против собственных ботов.



Рисунок 3 - Внешний вид игры DumbBots

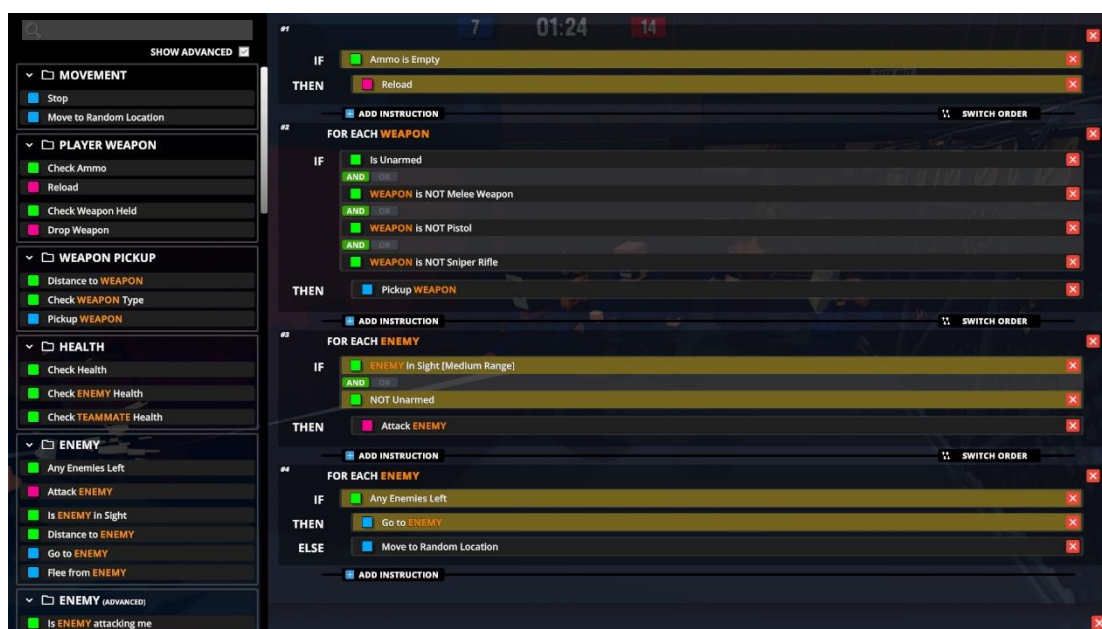


Рисунок 4 - Внешний вид редактора кода игры DumbBots

Преимуществом игры является наличие удобного редактора кода, приближенного к настоящему программированию, возможность расширения игры за счет собственных сценариев. Недостатком стало отсутствие мобильной версии игры.

Несмотря на разнообразие игровых активностей, режимов, юнитов в представленных аналогах, игры предоставляют возможность программирования поведения только отдельных боевых единиц. Создание же

бота, способного выставлять юниты на поле в зависимости от хода матча, не представляется возможным.

## 3 Реализация

### 3.1 Средства реализации

В качестве средств реализации мобильного приложения были выбраны:

Сервер:

- ОС Windows 10;
- Frameworks: Spring boot 2.6.6, Hibernate, Lombok;
- БД – PostgreSQL;
- Язык разработки - Java v. 17;
- Используемая IDE: IntelliJ IDEA 2021.3.2 (Ultimate Edition);
- Инструмент ведения документации API Swagger;
- Система контроля версий – Git;
- Хостинг исходного кода – Heroku.

Клиент:

- ОС Windows 10;
- Microsoft .NET Framework 4.5;
- Язык разработки - C#;
- Игровой движок Unity 2020.3.32f1;
- Используемая IDE: IntelliJ Rider;
- Система контроля версий – Git;
- Макеты интерфейсов – AxureRP.

Для серверной части был выбран стек Java + Spring Boot так как фреймворк Spring boot имеет большое количество преимуществ, среди них:

- Большое количество доступной документации;
- Встроенный сервер для развертывания приложения, что существенно облегчает процесс разработки;
- Огромный выбор плагинов, которые легко ставятся и сильно облегчают процесс разработки;
- Автоматическое внедрение зависимостей;
- Автоконфигурирование огромного количества компонентов.



Unity является не только игровым движком, но и полноценной средой разработки видеоигр. Unity делает создание игр максимально простым и комфортным. Кроме того, важной особенностью Unity является его кроссплатформенность, что позволяет охватить как можно большее количество игровых платформ. Unity обладает низким порогом вхождения, обладает широкой библиотекой ассетов и плагинов.

Использование языка программирования C# в клиенте обусловлено тем, что Unity поддерживает взаимодействие скриптов, написанных на C#, с внутриигровыми объектами.

### 3.2 Входные-выходные данные (IDEF0)

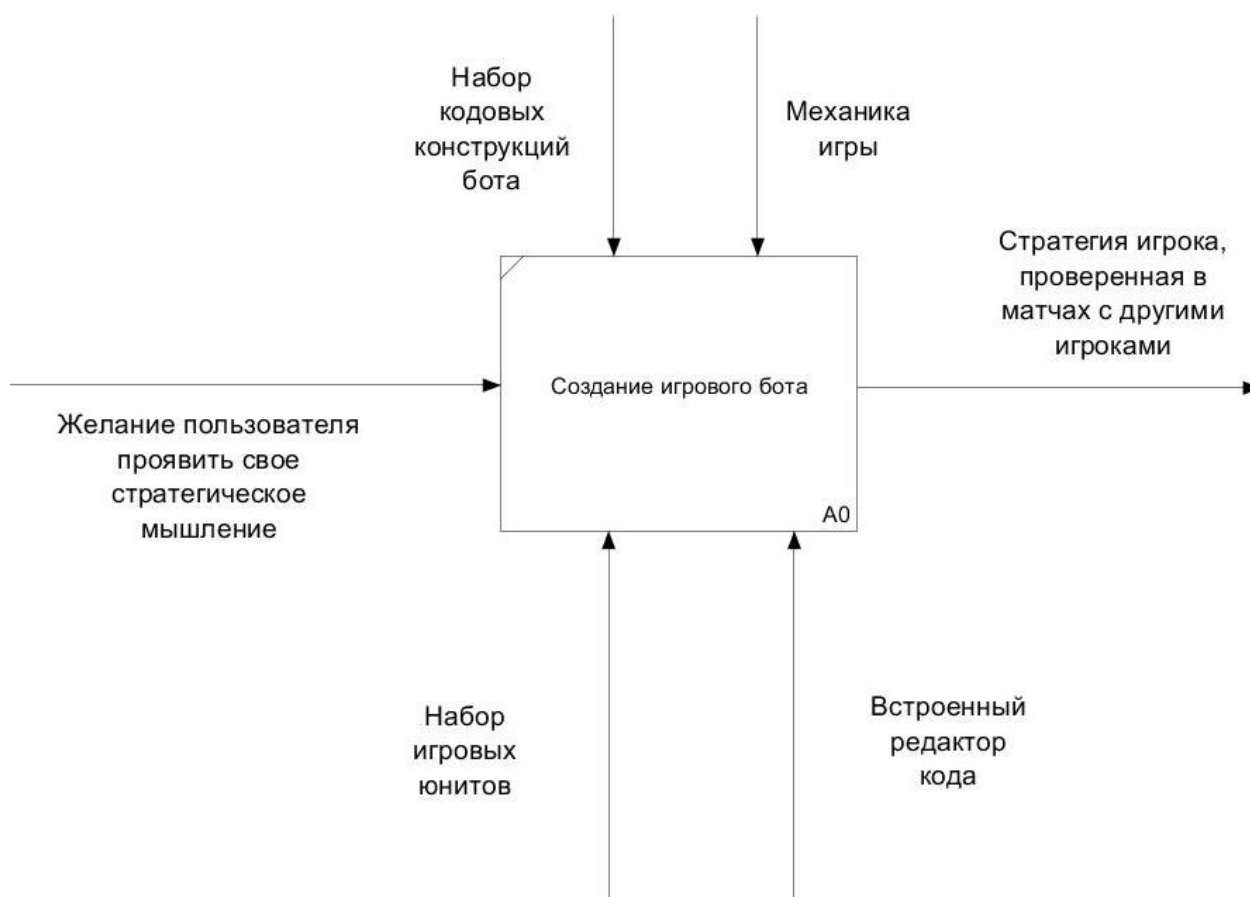


Рисунок 5 - Диаграмма IDEF0

### 3.3 Диаграмма вариантов использования

#### 3.3.1 Диаграмма Игрока

Игрок – авторизованный пользователь, не обладающий правами Модератора.

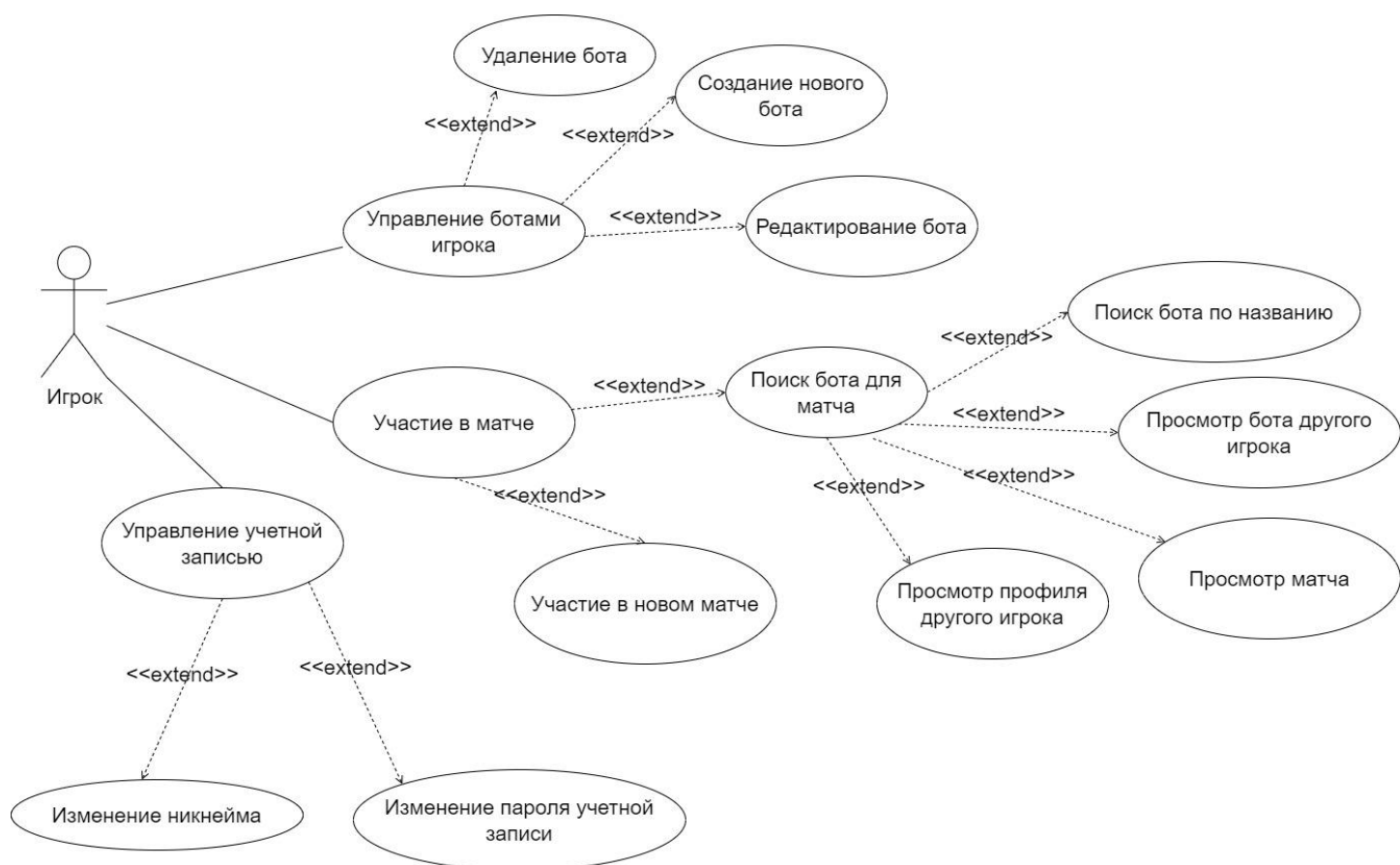


Рисунок 6 - Диаграмма вариантов использования для Игрока

### 3.3.2 Диаграмма Модератора

Модератор – авторизованный пользователь, обладающий правами Модератора.



Рисунок 7 - Диаграмма вариантов использования для Модератора

### 3.3.3 Диаграмма Наблюдателя

Наблюдателем считается пользователь, решивший пользоваться приложением без входа в учетную запись.



Рисунок 8 - Диаграмма вариантов использования для Наблюдателя

### 3.4 Диаграмма состояний



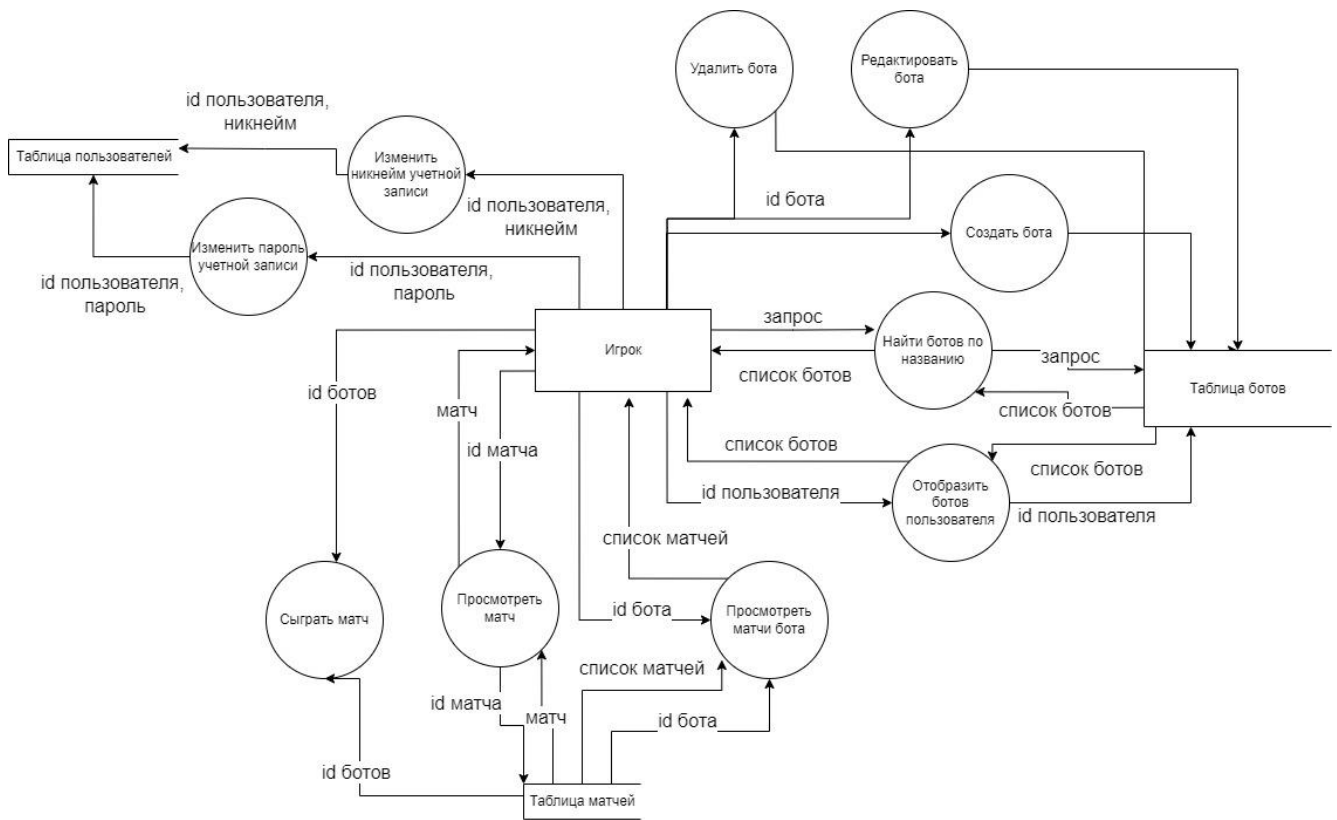


Рисунок 10 - Диаграмма потоков данных для Игрока

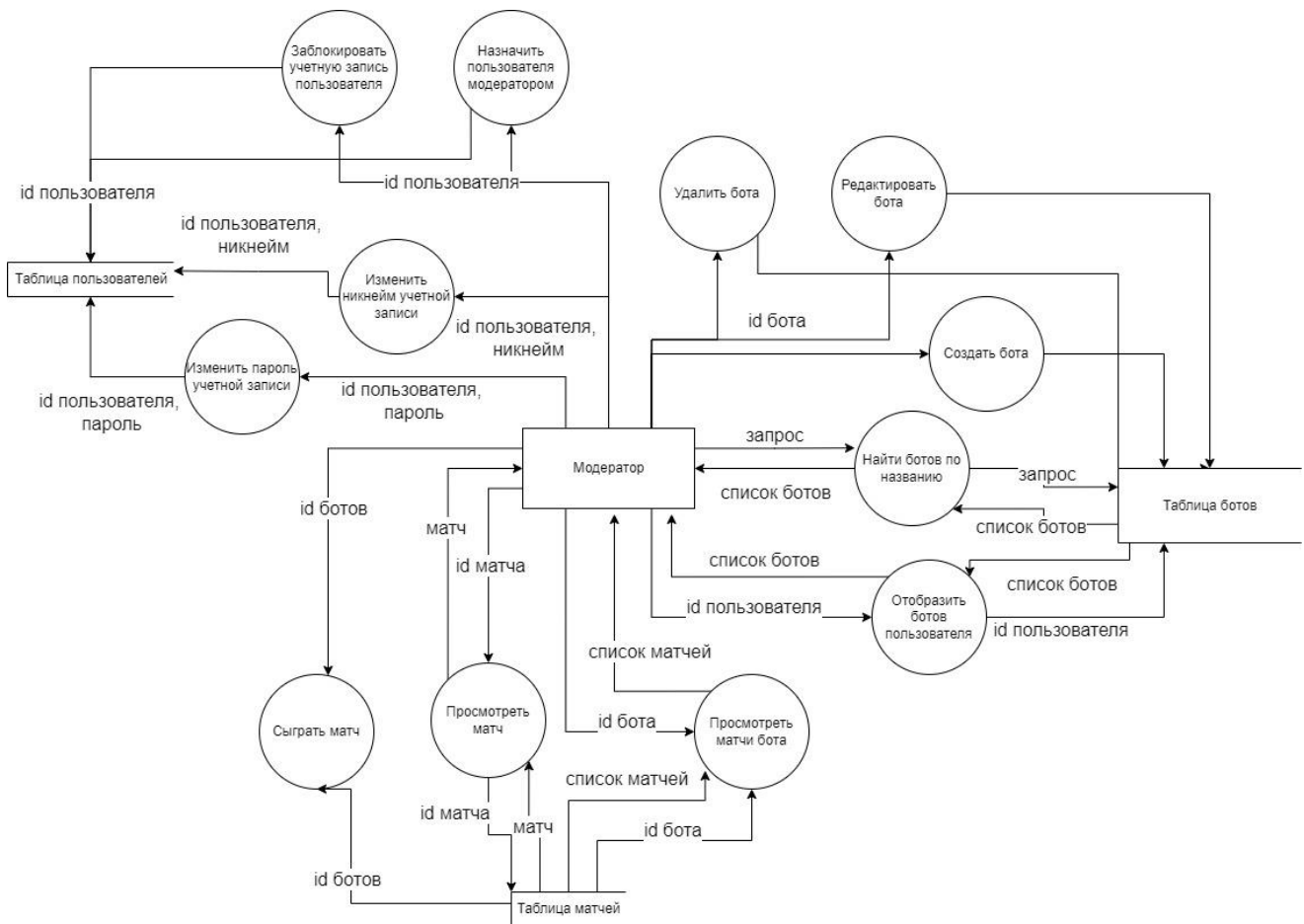


Рисунок 11 - Диаграмма потоков данных для Модератора

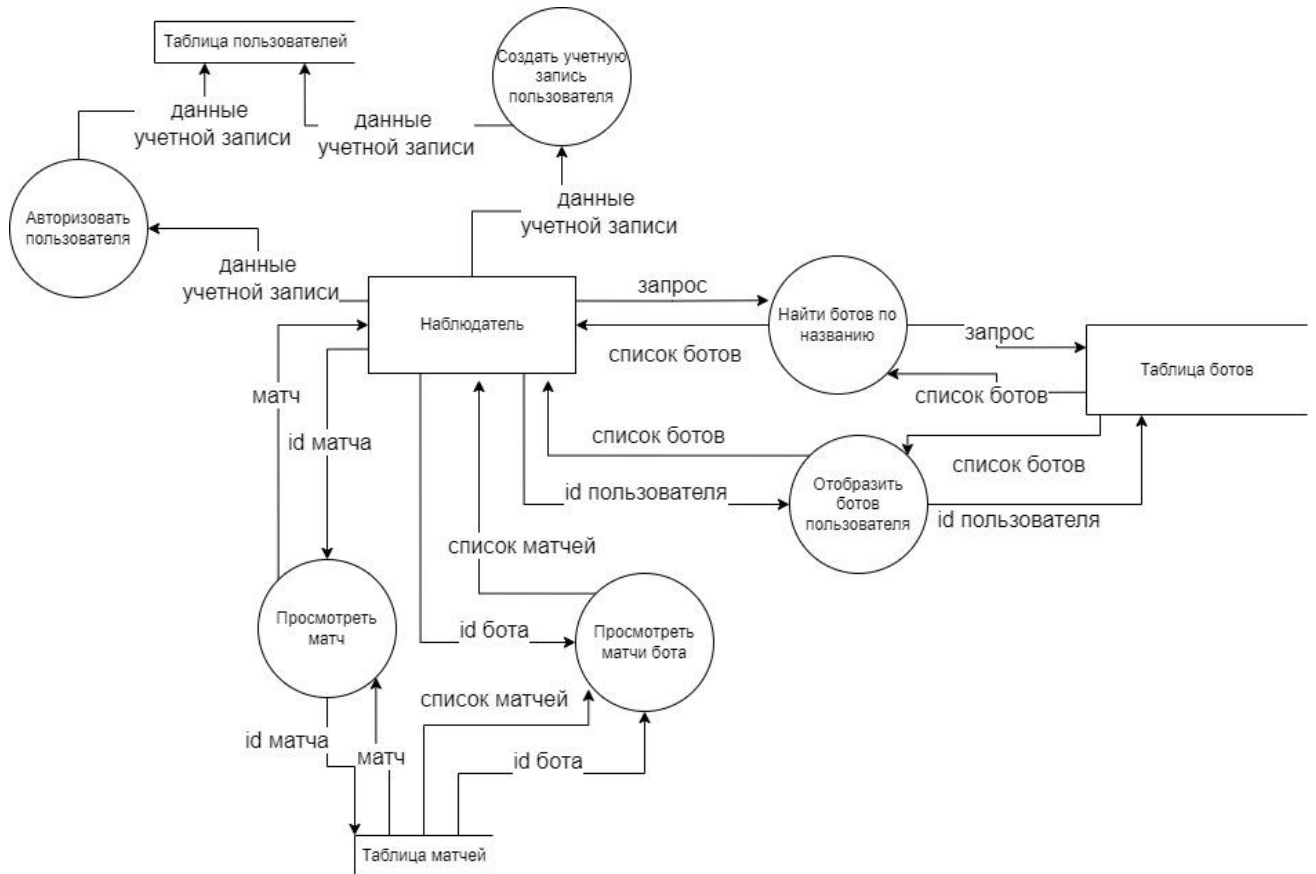


Рисунок 12 - Диаграмма потоков данных для Наблюдателя

### 3.6 Структурная схема приложения

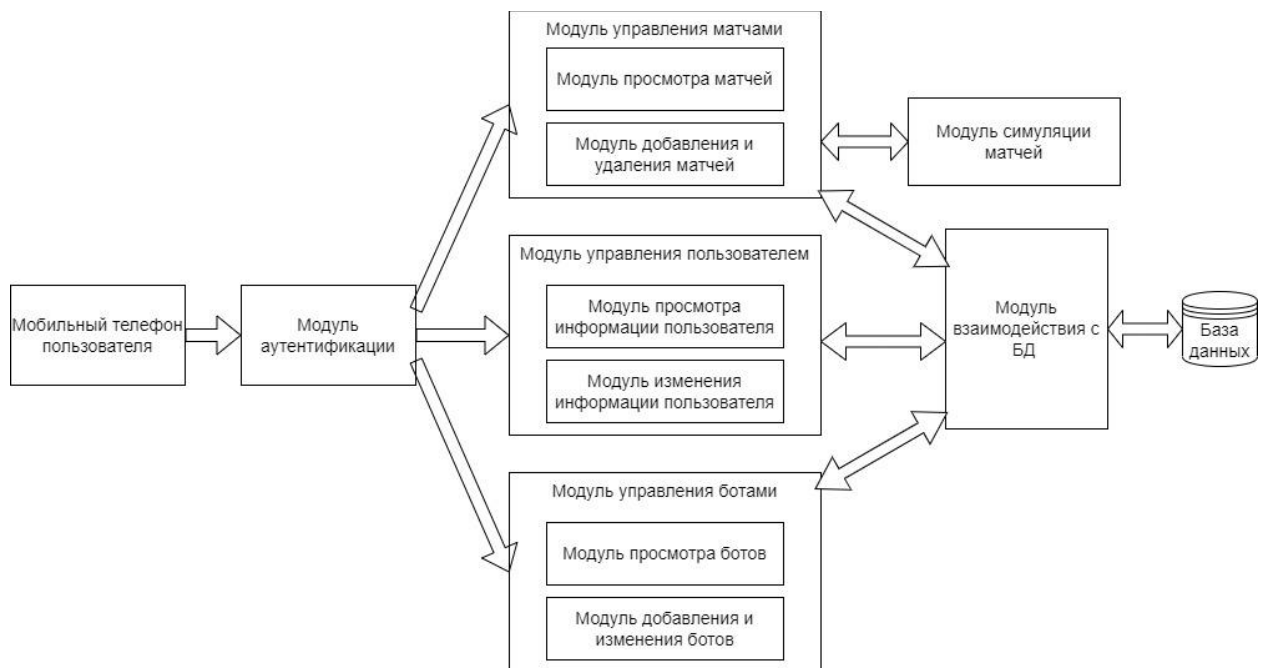


Рисунок 13 - Структурная схема приложения

## 3.7 Реализация серверной части приложения

### 3.7.1 Схема базы данных

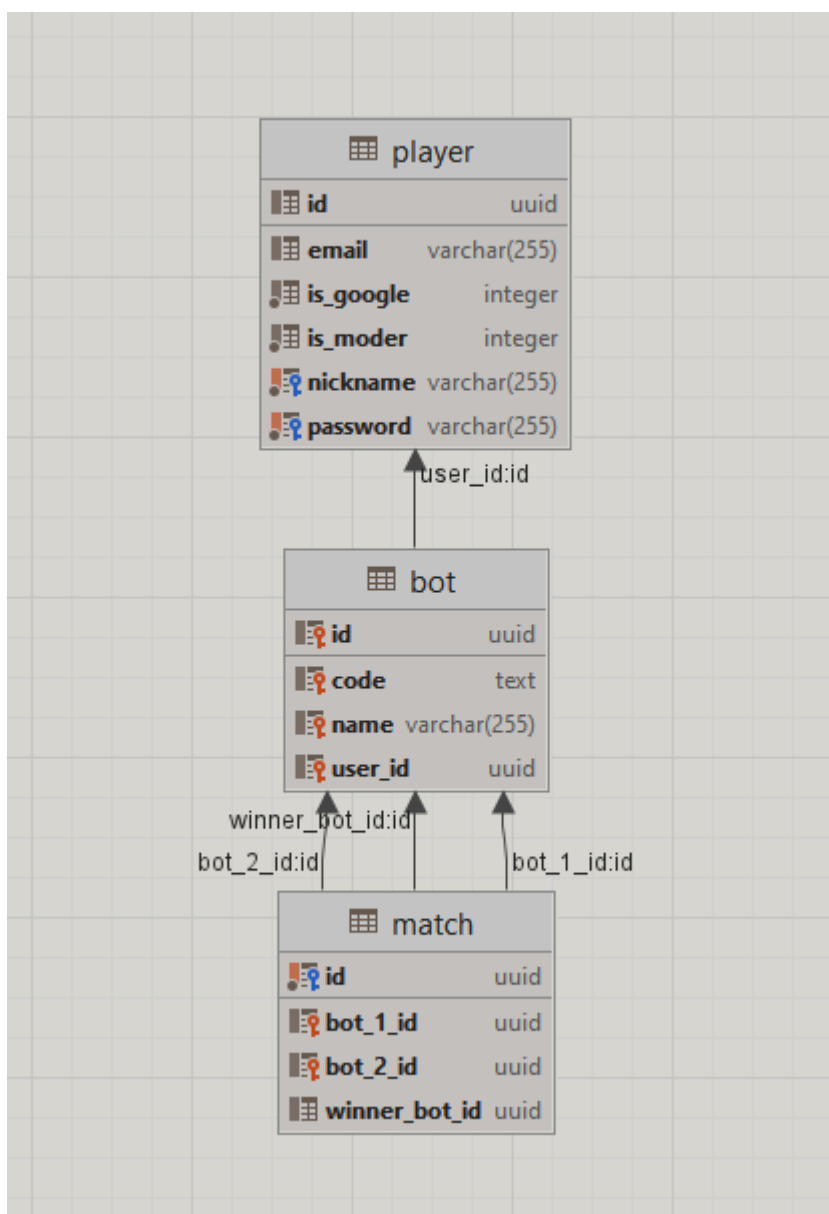


Рисунок 14 - Схема базы данных

player - таблица, содержащая информацию о зарегистрированных пользователях приложения

bot - таблица, содержащая информацию о стратегиях, которые создали зарегистрированные пользователи.

match - таблица, содержащая информацию о проведённых матчах между ботами.

### 3.7.2 Диаграмма классов



### 3.7.2.1 Диаграмма классов сущностей

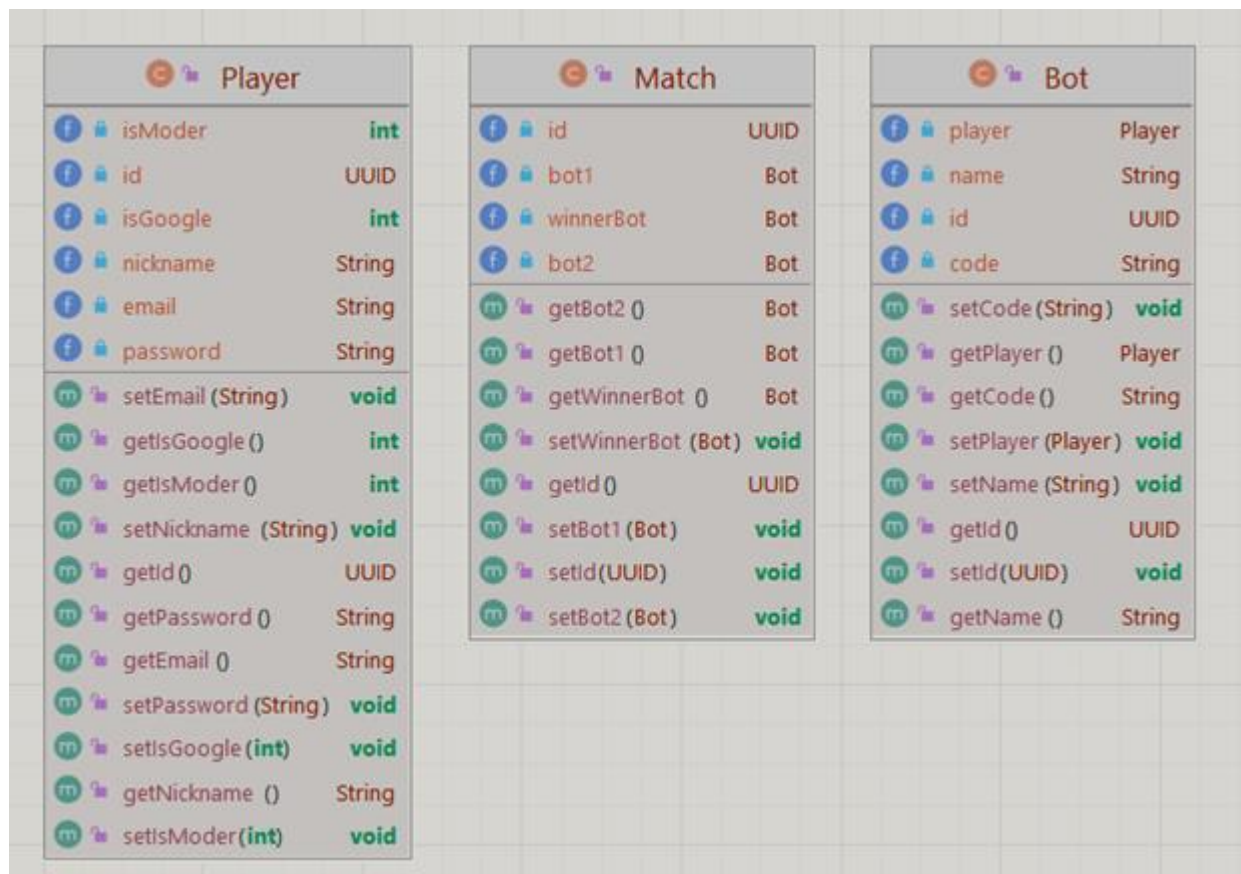


Рисунок 15 - Диаграмма классов сущностей

Поля каждого из этих классов эквивалентны атрибутам одноименных таблиц в БД.

Player - сущность пользователя, Match - сущность “игры”, Bot - сущность стратегии пользователя.

### 3.7.2.2 Диаграмма классов репозиториев



Рисунок 16 - Диаграмма классов репозиториев

Каждый из этих классов относится к слою доступа к данным, т.е. обеспечивает взаимодействие приложения с базой данных.

Через Match, Bot и Player репозитории обеспечивается взаимодействие с одноименными таблицами БД.

### 3.7.2.3 Диаграмма классов сервисов

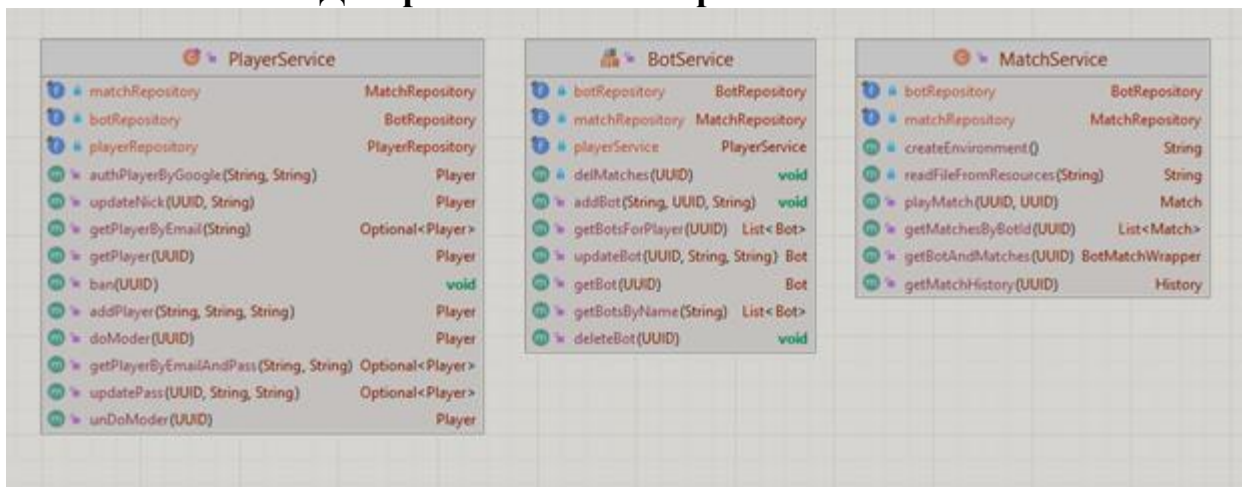


Рисунок 17 - Диаграмма классов сервисов

Каждый из этих классов является частью слоя бизнес-логики, т.е. выступают связующим звеном между слоем доступа к данным и контроллерами. Все вычисления и алгоритмы находятся в этих классах.

### 3.7.2.4 Диаграмма классов контроллеров

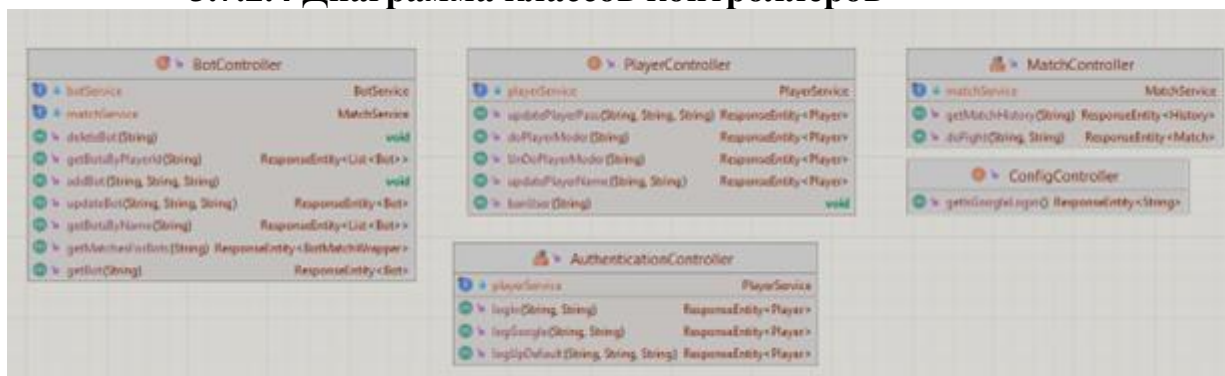


Рисунок 18 - Диаграмма классов контроллеров

Эти классы необходимы для общения с клиентом. Они получают запросы, вызывают методы слоя бизнес-логики и отправляют ответы назад на клиент.

### 3.7.2.5 Диаграмма классов-оберток

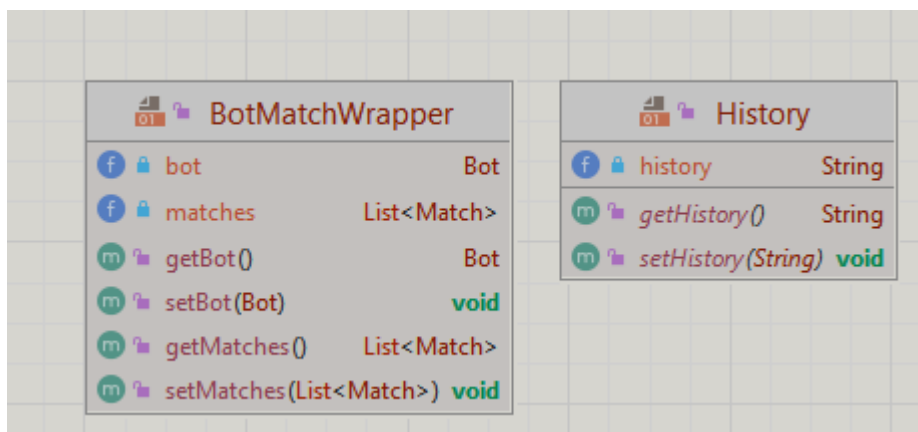


Рисунок 19 - Диаграмма классов-оберток

Эти классы нужны для правильной сериализации и отправки данных клиенту.

### 3.7.2.6 Диаграмма служебных классов

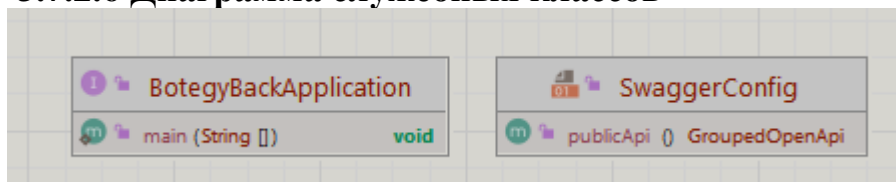


Рисунок 20 - Диаграмма служебных классов

Класс `BotegyBackApplication` является точкой входа для приложения, он запускает цепочку загрузки нужных зависимостей и классов, благодаря которым приложение функционирует.

Класс `SwaggerConfig` является классом конфигурация для инструмента создания документации API Swagger.

### 3.7.3 Архитектура серверной части приложения

Серверная часть приложения реализована соответственно трехслойной архитектуре веб-приложения с API Rest с использованием фреймворка Spring boot. Это фреймворк предоставляет возможности для удобной работы с базами данных, сам настраивает внедрение зависимостей.

### 3.7.4 Слой доступа к данным

Для каждой из сущностей был реализован интерфейс-репозиторий,

который является наследником JpaRepository. Такой подход позволил существенно уменьшить количество написанного кода, путем использования возможностей Spring Data, а именно больше количество уже реализованных методов для генерации запроса в базу данных и получения ответа и возможность объявления своих методов, при правильном написании названий которых Spring Data сам сгенерирует запрос в базу. На рисунке 21 представлена реализация интерфейса BotRepository.

```
public interface BotRepository extends JpaRepository<Bot, UUID> {  
  
    List<Bot> findByPlayer_Id(UUID id);  
  
    List<Bot> findByName(String name);  
  
}
```

Рисунок 21 - Реализация интерфейса BotRepository

### 3.7.5 Слой контроллеров

Контроллеры – такие классы, каждый метод из которых обрабатывает запрос с клиента на определенный маппинг и возвращает ответ в виде ResponceEntity в Rest API. Для каждой из сущностей были написаны контроллеры, методы которых отвечают за необходимые приложению действия с этими сущностями. Помимо этого, были реализованы еще два контроллера: аутентификации и конфигурационный. Первый отвечает за вход и регистрацию пользователей, второй – за предоставление клиенту каких-то конфигурационных данных. На данном этапе разработке приложения, этот контроллер имеет единственный метод, в котором считывается значение из соответствующей переменной среды, отвечающей за включение и выключение возможности входа используя Google, и исходя из результата отдает клиенту значение true или false. На рисунке 22 представлен простейший метод в классе BotController, который отдает объект Bot по его Id.

```

@GetMapping(value = "/getBot")
public ResponseEntity<Bot> getBot(@RequestParam String botId) {
    return ResponseEntity.ok(botService.getBot(UUID.fromString(botId)));
}

```

Рисунок 22 - Метод getBot

С помощью аннотации `GetMapping` указывается, что для получения данных клиенту нужно будет использовать `HttpMethod Get`. В качестве `value` как раз передается как раз маппинг, по которому будет доступен этот метод. С помощью `@RequestParam` указывается, что хотелось бы получить от клиента. В конце возвращается тело запроса со статусом `ok` (`Http status code 200`) и внутри лежит как раз объект `bot`, полученный вызовом метода `getBot()` класса `botService` с аргументом в виде полученного от клиента `botId`, который был предварительно переведен из `String` в `UUID`.

### 3.7.6 Слой бизнес-логики

Вся бизнес-логика реализована в `service` слое. Этот слой является промежуточным звеном между контроллерами и работой с базой данных, поэтому чаще всего методы просто передают аргумент, полученный из контроллера, вызвавшего его, в соответствующий метод в репозитории, затем возвращают ответ в контроллер, или собирают объект из полученных аргументов и также передают его в репозиторий, а полученный ответ затем в возвращают контроллеру. Такие типичные методы представлены на рисунке 23.

```

public void addBot(String name, UUID userId, String code) {
    Bot bot = new Bot();
    bot.setName(name);
    bot.setPlayer(playerService.getPlayer(userId));
    bot.setCode(code);
    botRepository.save(bot);
}

public List<Bot> getBotsByName(String botName) { return botRepository.findByName(botName); }

```

### 3.7.6.1 Механика игры

Механика игры является неотъемлемой и самой сложной частью бизнес-логики. Для реализации потребовалось запускать javascript код внутри java, для этого и была подключена библиотека j2v8. Чтобы появилась возможность проводить матчи, сначала было подготовлено некоторое окружение. В папке с ресурсами были созданы 4 javascript файла: bot, field, match и unite. Bot хранит в себе изначально только количество здоровья и энергии, но в будущем в него будет интегрирован код, который писал пользователь на клиенте. Field описывает поведение на поле боя каждого из ботов, то есть их возможные действия. Также скрипт именно из этого файла при необходимости пишет историю матча. Match содержит в себе функции которые как раз начинают игру. Unite содержит описание юнитов с их характеристиками.

Далее в классе MatchService был реализован метод playMatch(), который продемонстрирован на рисунке 24.



```

public Match playMatch(UUID bot1Id, UUID bot2Id) {
    V8 runtime = V8.createV8Runtime();
    String defaultScript = createEnvironment();
    runtime.executeVoidScript(defaultScript);
    V8Array parameters = new V8Array(runtime);
    Bot bot1 = botRepository.findById(bot1Id).get();
    Bot bot2 = botRepository.findById(bot2Id).get();
    Match match = new Match();
    match.setBot1(bot1);
    match.setBot2(bot2);
    parameters.push(match.getBot1().getCode());
    parameters.push(match.getBot2().getCode());
    int func = -1;
    try {
        func = runtime.executeIntegerFunction( name: "get_match_results", parameters);
    } catch (V8ScriptExecutionException e) {
        e.printStackTrace();
    }

    if (func != -1) {
        switch (func) {
            case 0 -> match.setWinnerBot(bot1);
            case 1 -> match.setWinnerBot(bot2);
        }
        matchRepository.save(match);
        return match;
    } else return null;
}

```

Рисунок 24 - Метод playMatch

Создается объект runtime, далее в переменную defaultScript с помощью функции createEnviroment(), внутри которой происходит чтение всех вышеописанных javascript файлов в строки а затем конкатенация их. Затем в runtime при помощи executeVoidScript() передается окружение в виде defaultScript. После создается объект parameters, в который позже положатся коды двух ботов. Далее из базы данных по id достаются два бота, создается объект match, эти боты устанавливаются внутри него как противоборствующие. В parameters кладутся коды этих ботов и далее запускается функция get\_match\_results, с аргументами в виде parameters. Результат битвы присваивается переменной func и в зависимости от него

устанавливается бот-победитель. После этого объект match сохраняется в базу данных и возвращается клиенту.

Если же клиент запрашивает результат матча, то вызывается метод getMatchHistory, который представлен на рисунке

```
public History getMatchHistory(UUID matchId) {
    V8 runtime = V8.createV8Runtime();
    String defaultScript = createEnvironment();
    runtime.executeVoidScript(defaultScript);
    Match m = matchRepository.getById(matchId);
    V8Array parameters = new V8Array(runtime);
    parameters.push(m.getBot1().getCode());
    parameters.push(m.getBot2().getCode());

    InputStream s = new ByteArrayInputStream(
        (runtime.executeStringFunction( name: "get_match_log", parameters))
            .getBytes(StandardCharsets.UTF_8));
    String res = "";
    try {
        res = new String(s.readAllBytes());
    } catch (IOException e) {
        e.printStackTrace();
    }
    History history = new History();
    history.setHistory(res);
    return history;
}
```

Рисунок 25 - Метод getMatchHistory

Этот метод очень похож на метод playMatch(), который описан выше, но все же некоторые отличия есть. Во-первых, метод принимает на вход id матча, который уже произошёл. Во-вторых, в качестве результата выполнения функции get\_match\_log мы получаем объект ByteArrayInputStream, предварительно считывая байты в кодировке UTF\_8 из строки, которую вернул javascript. Это реализовано именно так для того, чтобы история матча правильно отображалась, так как она на русском языке. Ну и наконец, в качестве возвращаемого значения здесь не String, а класс обертка History с



одним полем history которое String. Это реализовано также чтобы русский язык дошел до клиента, не превратившись в артефакты при сериализации.

### **3.8 Реализация клиентской части приложения**

#### **3.8.1 Макеты интерфейса**

Сразу после запуска приложения пользователю будет показан экран, на котором ему будет предложено войти в свою учетную запись, зарегистрировать новую учетную запись или продолжить работу с приложением без входа в учетную запись.

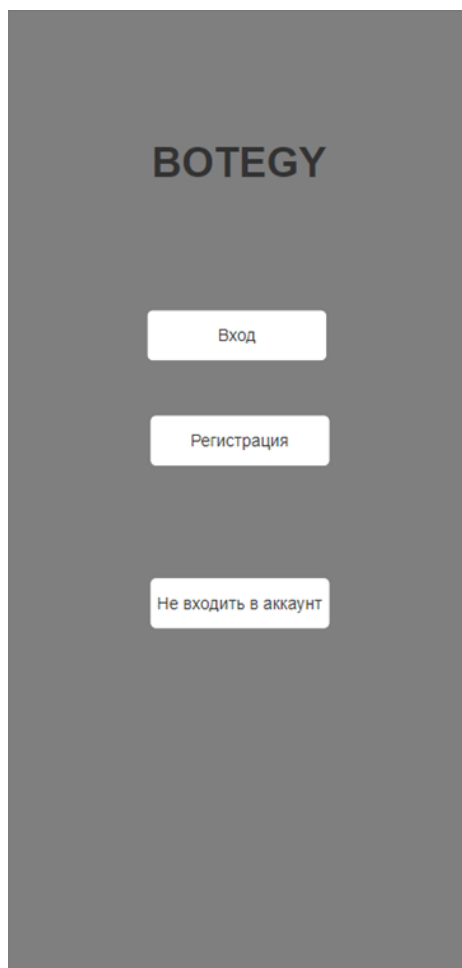


Рисунок 26 - Начальный экран приложения

Рассмотрим случай, когда пользователь в первый раз зашел в приложение и еще не зарегистрировал учетную запись.

В этом случае при нажатии на кнопку «Регистрация» пользователь увидит экран регистрации, где ему будет предложено ввести электронную почту, никнейм для новой учетной записи и пароль. Если будут введены

корректные данные и на введенную почту еще не была зарегистрирована учетная запись, то будет создана новая учетная запись, в которую сразу войдет пользователь, и он будет направлен на главный экран приложения.

Кроме введения данных пользователю предлагается создать новую учетную запись с помощью Google.

При нажатии на кнопку «Назад» пользователь вернется на начальный экран приложения.



Рисунок 27 - Экран регистрации

Теперь рассмотрим случай, когда пользователь уже зарегистрирован и хочет войти в свою учетную запись.

После нажатия кнопки «Вход» пользователь увидит экран, на котором ему будет предложено ввести свои учетные данные или войти в учетную запись с помощью Google.

После нажатия кнопки «Войти», если пользователь ввел нужные данные, он увидит главный экран приложения для пользователя, вошедшего в учетную запись.

При нажатии на кнопку «Назад» пользователь вернется на начальный экран приложения.

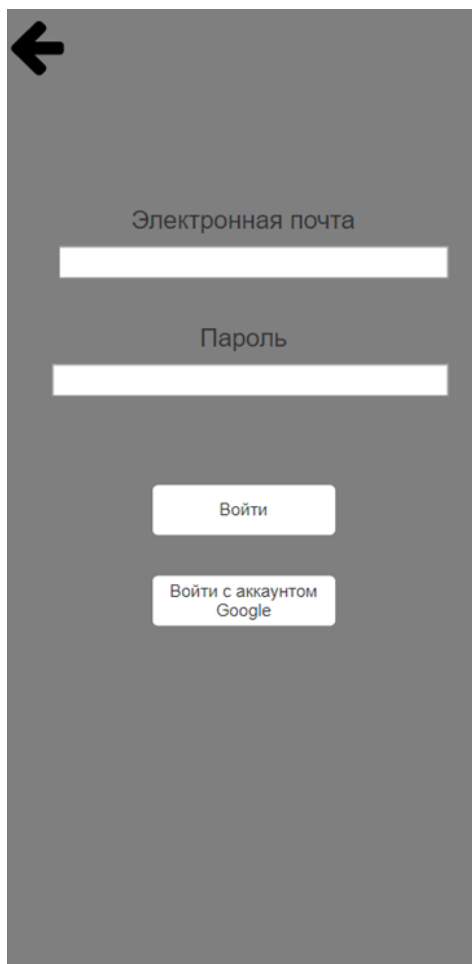


Рисунок 28 - Экран входа в учетную запись

На главном экране находятся кнопки для перехода к экрану с созданными им ботами, экрану поиска ботов других пользователей, экрану просмотра и редактирования информации учетной записи.

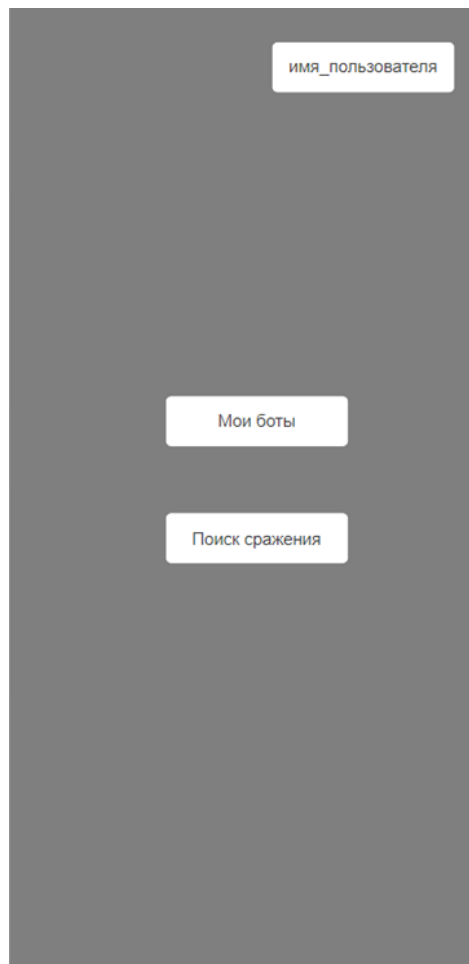


Рисунок 29 - Главный экран приложения

Если пользователь нажмет на кнопку с его никнеймом, то он увидит экран, на котором отобразятся никнейм пользователя, электронная почта, используемая для входа в учетную запись. Рядом с никнеймом находится кнопка для редактирования имени пользователя. Во всплывающем окне будет предложено ввести новое имя пользователя.

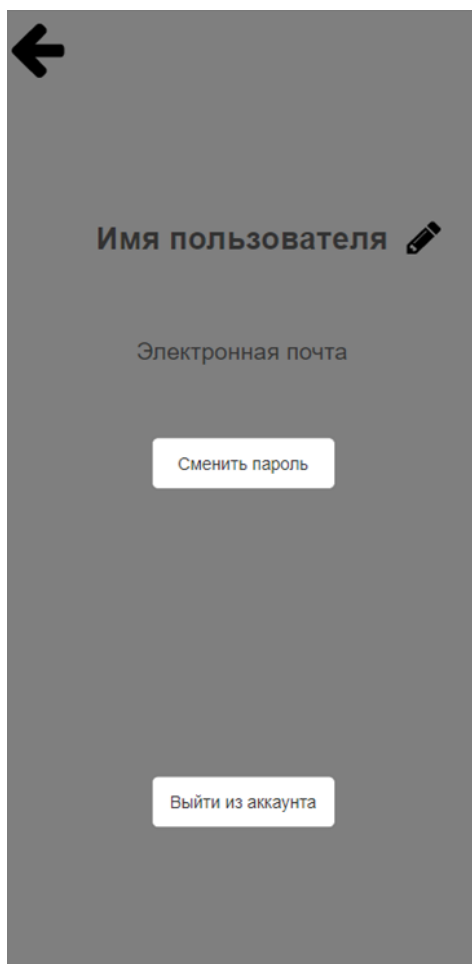


Рисунок 30 - Экран профиля пользователя

Кнопка «сменить пароль» ведет на экран, позволяющий изменить пароль учетной записи. На нем предлагается ввести старый и новый пароли. Если старый пароль введен верно, то пользователь вернется на экран с информацией о его учетной записи.

При нажатии на кнопку «Назад» пользователь вернется на главный экран приложения.

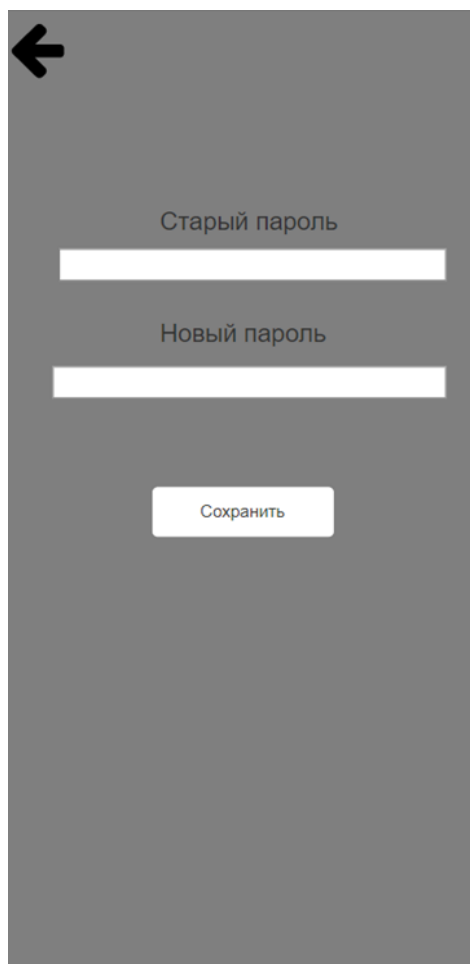


Рисунок 31 - Экран изменения пароля пользователя

Кнопка «Выйти из аккаунта» позволяет выйти из учетной записи и вернуться на начальный экран приложения.

При нажатии на кнопку «Назад» пользователь вернется на экран с информацией о текущем пользователе.

Вернемся к главному экрану приложения для пользователя, вошедшего в учетную запись. Нажатие на кнопку «Мои боты» позволит пользователю перейти к списку ботов, которых он создал.

Кнопки, расположенные слева, позволяют переключаться между ботами для просмотра их названия, истории матчей и кода, написанного пользователем. Кнопки внизу экрана позволяют изменять ботов.

При нажатии на кнопку «Назад» пользователь вернется на главный экран приложения.

При нажатии на кнопку удаления бота пользователь увидит всплывающее окно, на котором пользователю необходимо будет подтвердить, что он действительно хочет удалить бота.

При нажатии на кнопку редактирования бота пользователю будет предложено выбрать, что сделать с ботом: заменить выбранного или создать его копию, чтобы редактировать ее.

При нажатии на кнопку добавления бота будет создан новый бот.

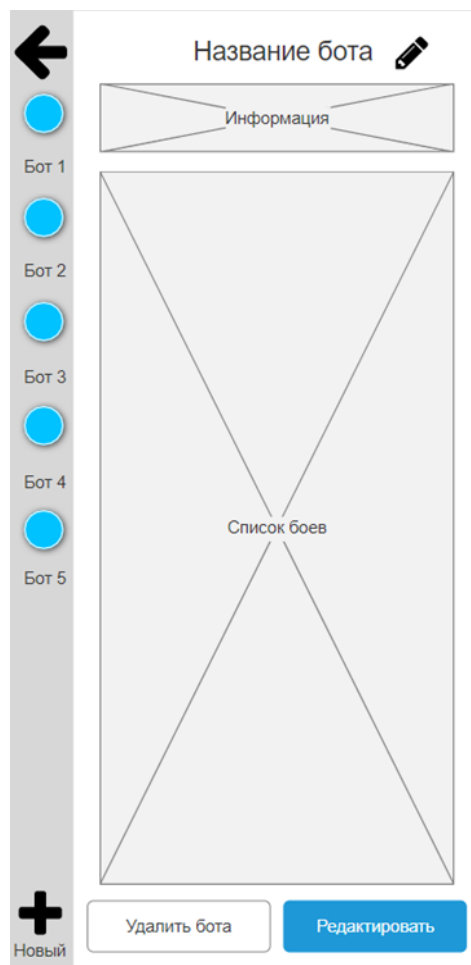


Рисунок 32 - Экран списка ботов пользователя

При редактировании бота и добавлении нового пользователь увидит экран редактора кода бота.

Кнопки с левой стороны экрана вызывают появление меню с теми блоками кода, которые пользователь может добавить в код своего бота, в зависимости от выбранной категории. При выборе блока кода из меню он

появится на экране и его можно будет редактировать: добавлять вложенные блоки, изменять значения, удалять.

При нажатии кнопки сохранения изменения будут сохранены для редактируемого бота, а пользователь вернется на экран со списком ботов.

Нажатие на кнопку «Назад» приведет к тому, что изменения не будут сохранены.

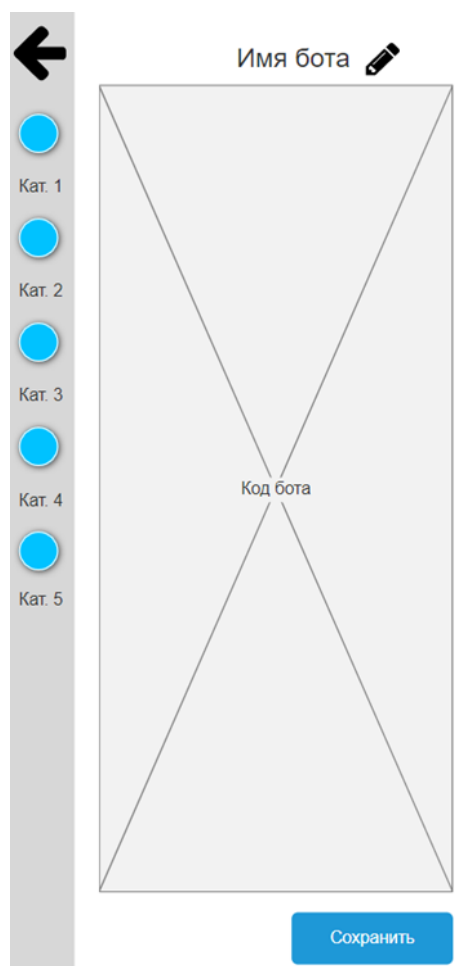


Рисунок 33 - Экран редактирования кода бота

Рассмотрим также случай, когда пользователь решил не входить в учетную запись. В этом случае он будет направлен на экран поиска ботов. На этот же экран направляются зарегистрированные пользователи при нажатии кнопки «Поиск сражения» на главном экране.

Пользователю будет предложено ввести поисковой запрос – название интересующего бота. При нажатии кнопки поиска пользователь увидит список



кнопок с названиями найденных ботов. При нажатии на кнопки пользователь сможет перейти на экраны с информацией о соответствующих ботах.

При нажатии на кнопку «Назад» зарегистрированный пользователь вернется на главный экран приложения, а незарегистрированный – на начальный.



Рисунок 34 - Экран поиска ботов

Экран с информацией о боте содержит название бота, кнопку с никнеймом пользователя, который его создал и кнопки с матчами, в которых бот участвовал.

При нажатии на кнопку с никнеймом создателя пользователь сможет перейти на экран с информацией о другом пользователе.

При нажатии на кнопку матча пользователь сможет посмотреть ход выбранного матча.

При нажатии на кнопку «Назад» пользователь вернется на экран поиска.

Для зарегистрированного пользователя экран бота содержит также кнопку «Сразиться», что позволит одному из ботов пользователя сыграть матч с просматриваемым ботом.

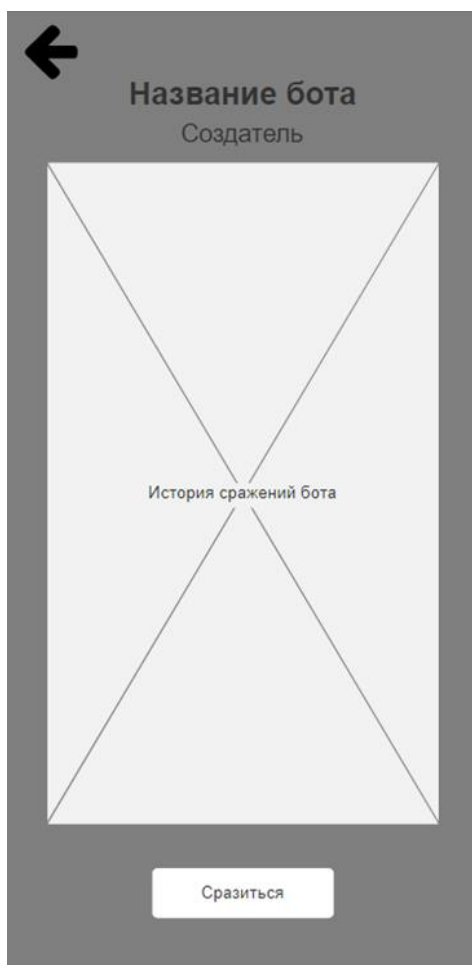


Рисунок 35 - Экран информации о боте другого пользователя

Экран с информацией о другом пользователе содержит никнейм этого пользователя и кнопки с названиями созданных им ботов. Нажатие на эти кнопки ведет на экраны с информацией о соответствующих ботах. Для зарегистрированного пользователя с правами модератора экран содержит кнопки для блокировки учетной записи пользователя и назначения его модератором.



Рисунок 36 - Экран информации о пользователе

Рассмотрим случай, когда зарегистрированный пользователь нажал «Сразиться» на экране бота. Тогда пользователь увидит экран выбора одного из своих ботов для матча.



Рисунок 37 - Экран выбора бота для матча

Кнопки, расположенные слева, позволяют переключаться между ботами для просмотра их названия и кода, написанного пользователем.

При нажатии на кнопку «Назад» пользователь вернется на экран просмотра информации о боте другого пользователя.

При нажатии на кнопку выбора бота будет проигран матч между ботом пользователя и выбранным им ботом другого пользователя.

Результат матча будет выведен во всплывающем окне. В окне также расположены кнопки, позволяющие закрыть окно и просмотреть ход матча.

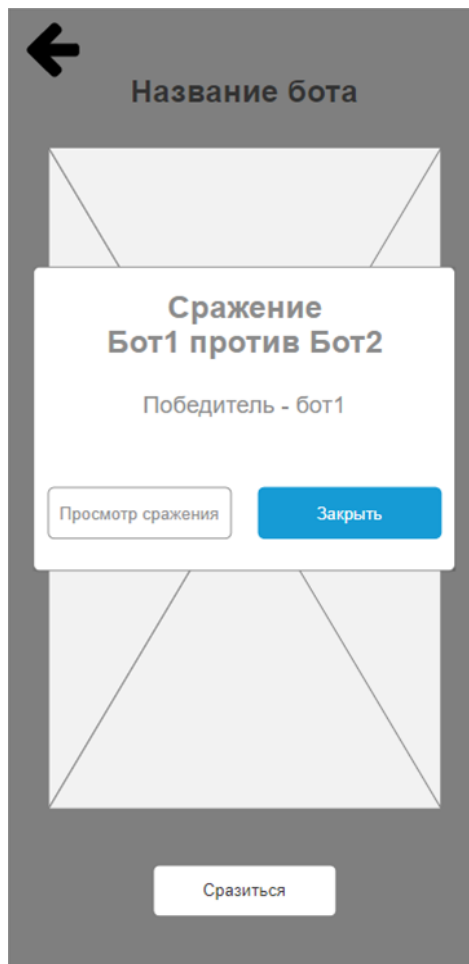


Рисунок 38 - Отображение результатов матча

При нажатии на кнопку «Просмотр сражения» пользователь будет направлен на экран просмотра матча.

Экран просмотра матча содержит названия ботов, участвовавших в матче, и поле, отображающее действия, предпринимаемые ботами на каждом шагу.

Нажатие на кнопки матчей на экранах информации о ботах ведут также на экран просмотра матча.



Рисунок 39 - Экран просмотра матча

### 3.8.2 Диаграмма классов

#### 3.8.2.1 Диаграмма классов пакета ButtonScript

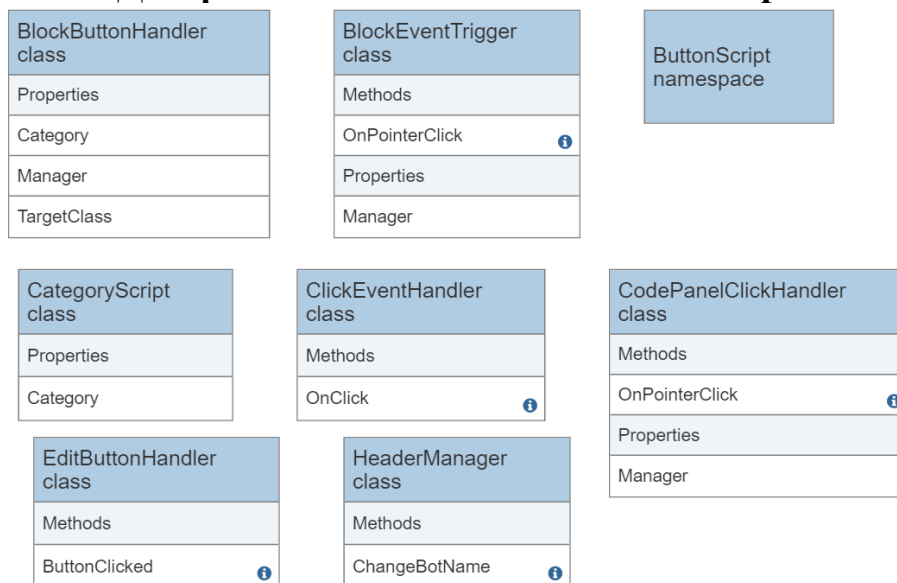


Рисунок 40 - Диаграмма классов пакета ButtonScript

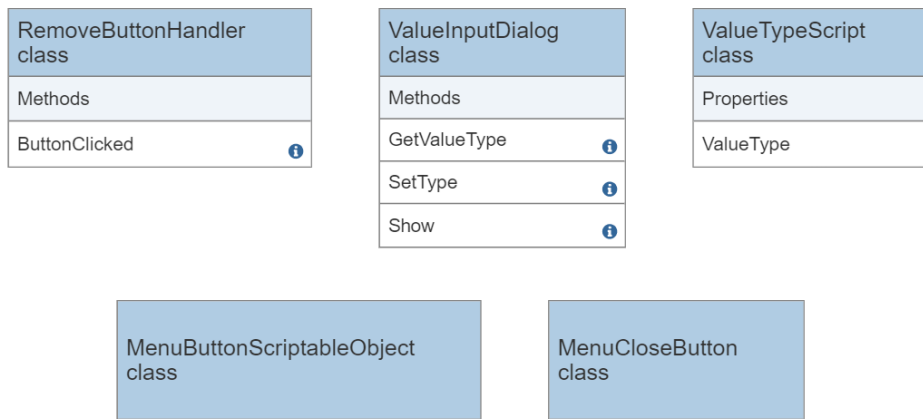


Рисунок 41 - Диаграмма классов пакета ButtonScript (продолжение)

В этот пакет входят классы обработчиков событий клиента. Каждый класс отвечает за свой элемент интерфейса.

### 3.8.2.2 Диаграмма классов пакета Dialog

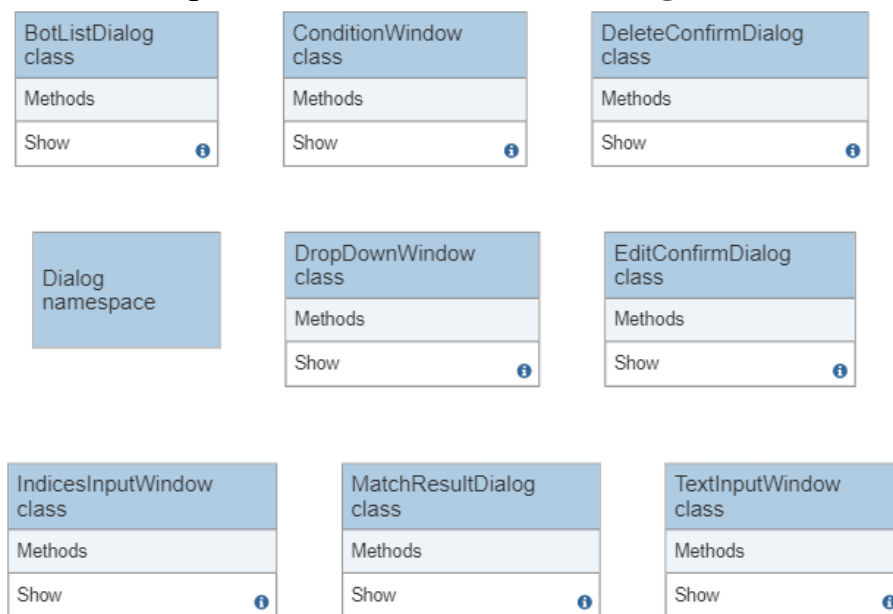


Рисунок 42 - Диаграмма классов пакета Dialog

В этот пакет входят классы всплывающих окон интерфейса. Каждое окно обладает своими элементами управления и отвечает за ввод какой-либо информации пользователем: подтверждение действия, ввод текста в текстовое поле, выбор из выпадающего списка.

### 3.8.2.3 Диаграмма классов пакета Entity

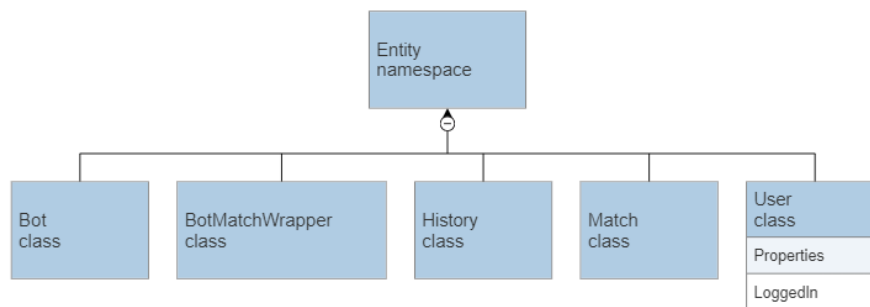


Рисунок 43 - Диаграмма классов пакета Entity

В этот пакет входят классы сущностей, данные которых клиент получает по запросу к серверу.

### 3.8.2.4 Диаграмма классов пакета Model

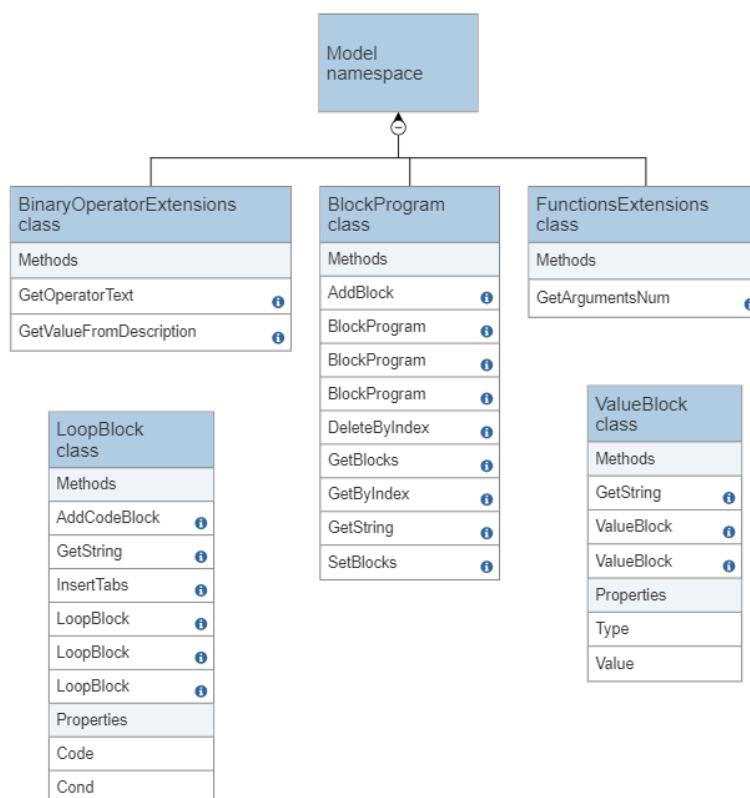


Рисунок 44 - Диаграмма классов пакета Model



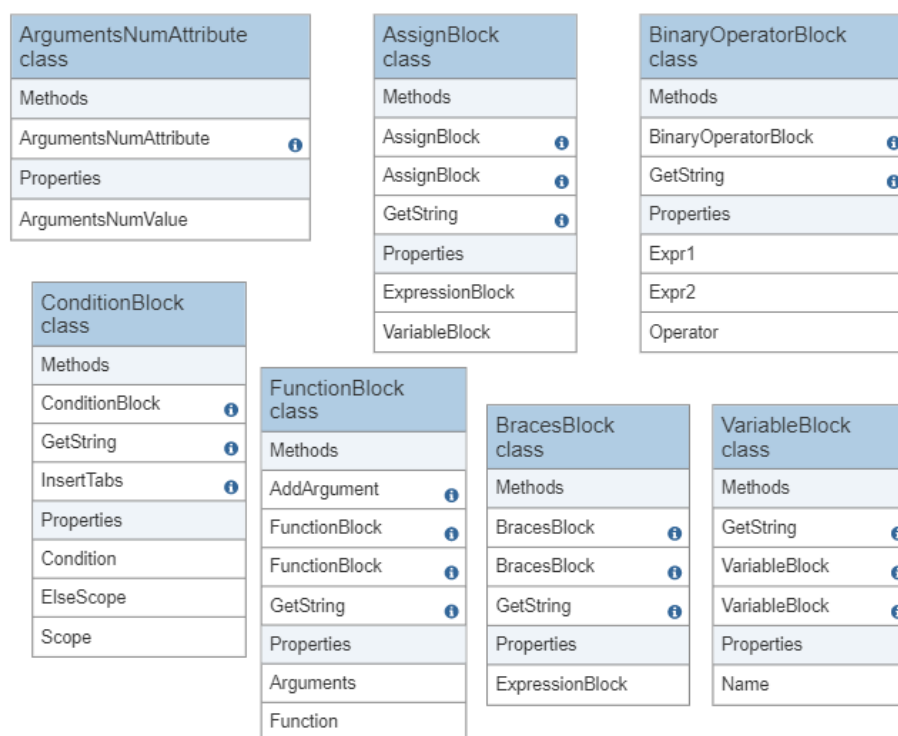


Рисунок 45 - Диаграмма классов пакета Model (продолжение)

В этот пакет входят классы, моделирующие кодовые конструкции, необходимые для редактора кода бота. По коду бота строится BlockProgram, состоящая из блоков кода IStatement и IExpression. Полученная программа выводится на экран редактора кода.

При сохранении ботов выстроенные на экране блоки преобразуются в BlockProgram для сохранения в базе данных. Модель повторяет модель, используемую интерпретатором при симуляции матчей между ботами.

### 3.8.2.5 Диаграмма классов пакета AppSceneManager



Рисунок 46 - Диаграмма классов пакета AppSceneManager

В этот пакет входят классы необходимые для построения сцен пользовательского интерфейса и переключения между ними.

### 3.8.2.6 Диаграмма классов пакета CodeParser

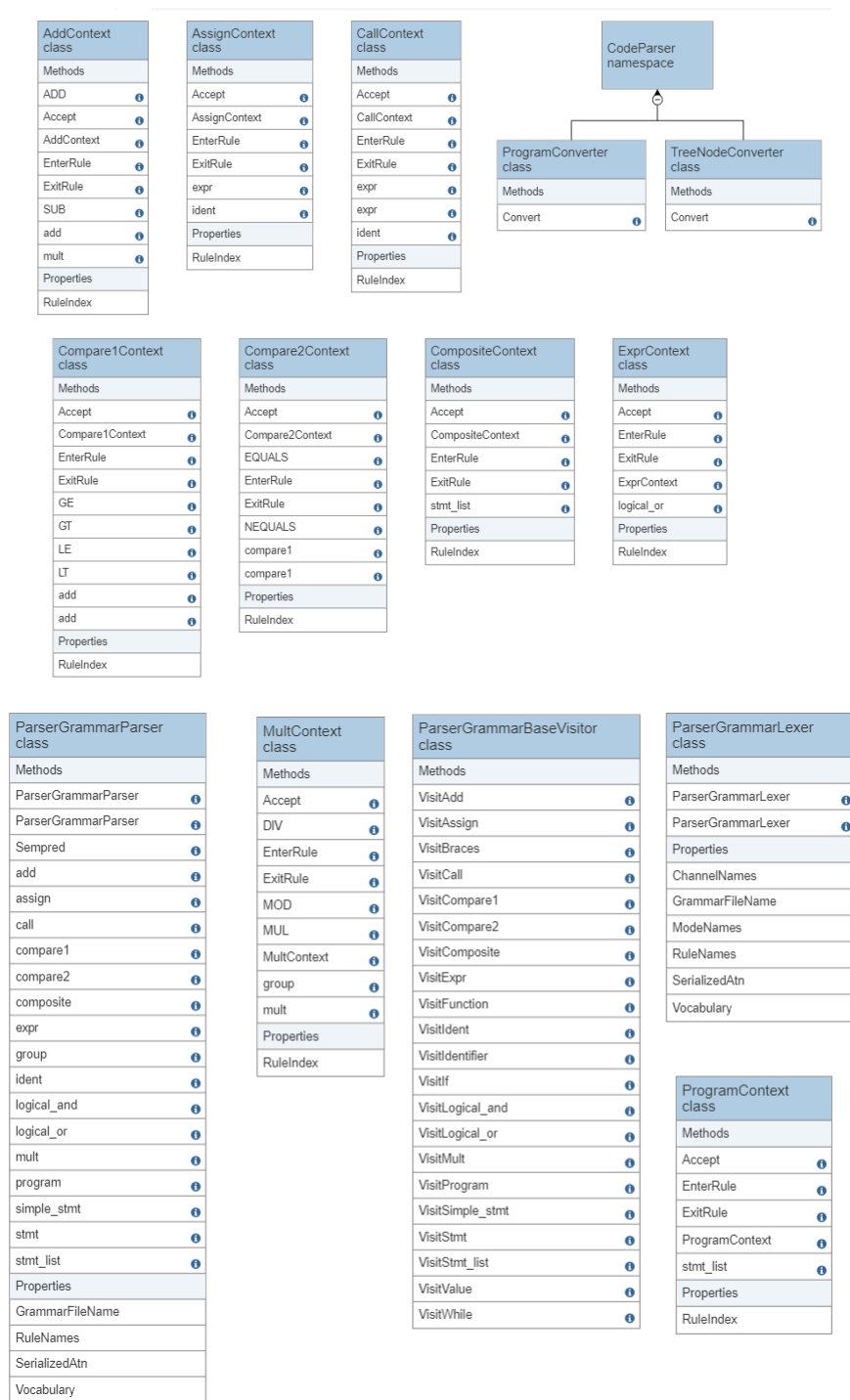


Рисунок 47 - Диаграмма классов пакета CodeParser

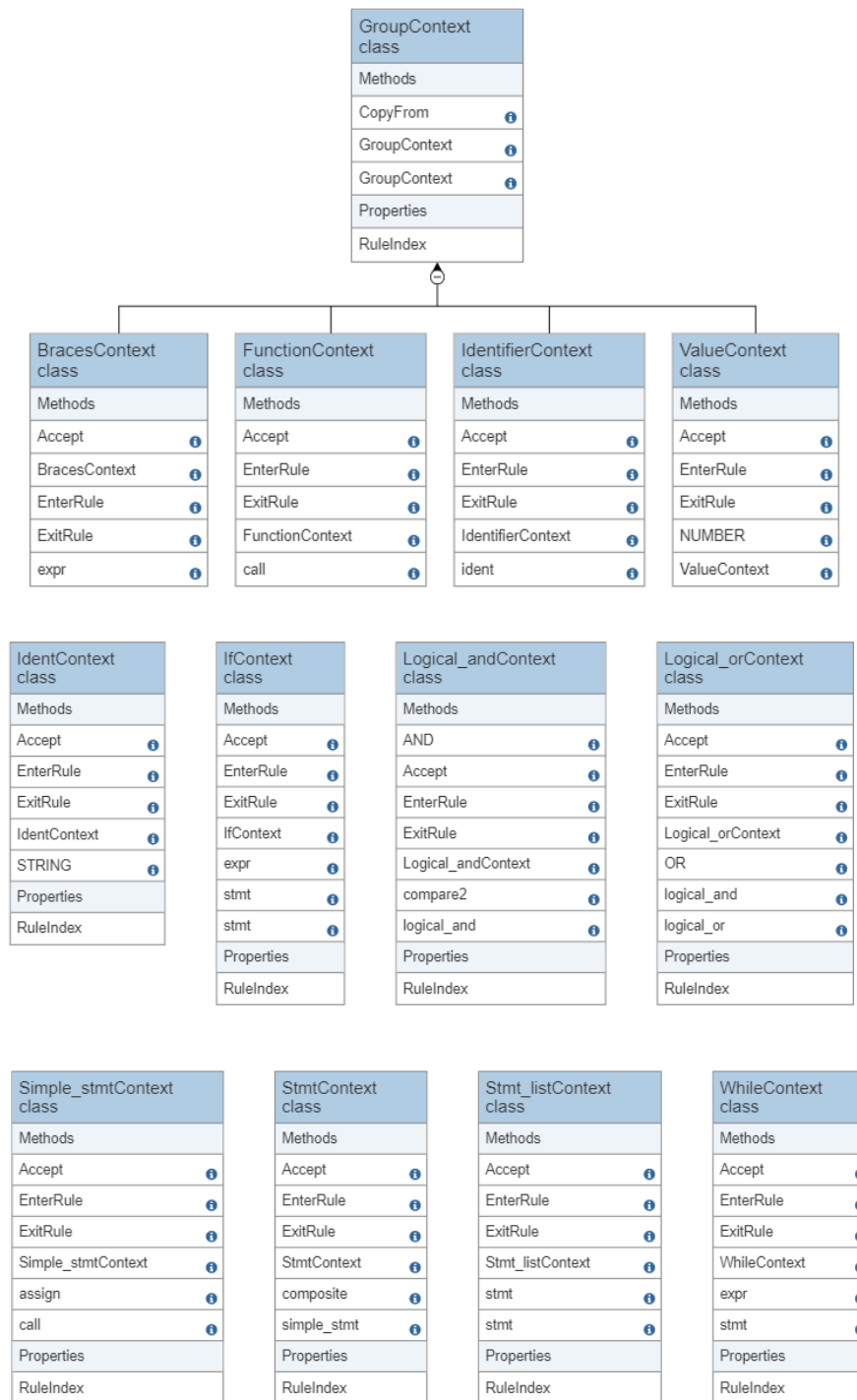


Рисунок 48 - Диаграмма классов пакета CodeParser (продолжение)

В этот пакет входят классы, сгенерированные библиотекой ANTLR4 для разбора кода бота.

### 3.8.3 Игровые сцены

Все экраны приложения разделены по категориям и собраны в сцены (Scene) Unity. Используются следующие сцены:

- StartScene – содержит в себе начальный экран приложения, экраны регистрации и входа в учетную запись;
- UserScene – содержит в себе главный экран приложения для пользователя, вошедшего в учетную запись, экран информации об учетной записи пользователя, экран изменения никнейма пользователя и экран справки;
- SearchScene – содержит в себе экраны поиска ботов, экран просмотра информации о найденном боте другого пользователя, экран просмотра информации о другом пользователе, диалоговое окно отображения результатов матча;
- BotListScene – содержит в себе экраны списка ботов пользователя, вошедшего в учетную запись, экран с историей матчей выбранного бота;
- BotEditorScene – содержит в себе экран редактора кода бота;
- MatchPlayBackScene – содержит в себе экран просмотра хода матча.

Каждая сцена обладает соответствующим скриптом `SceneManager`, который отвечает за переключение между экранами приложения, обращение к серверу, отображение информации, пришедшей от сервера, обработку ввода пользователя, переключение между сценами.

К элементам интерфейса каждой сцены подключен скрипт `Animator`, отвечающий за отображение анимированных переходов между экранами и сценами. К `StartScene` подключен скрипт `AppMetrica` для использования платформы для аналитики приложений `AppMetrica`.

Для формирования запросов к серверу и обработки ответа сервера используется класс `RequestProcessor`, использующий пакет `UnityEngine.Networking` для отправки GET, POST, PUT и DELETE запросов. Для обработки ответа сервера используется класс `Serializer`, отвечающий за преобразование JSON-объектов в объекты пакета `Entity`.

### **3.8.4 Редактор кода бота**

Сцена редактора кода бота используется для создания кода ботов. Создание кодовых блоков осуществляется с помощью скрипта BotEditorSceneManager. Именно он обрабатывает пользовательский ввод и отображает необходимые элементы интерфейса.

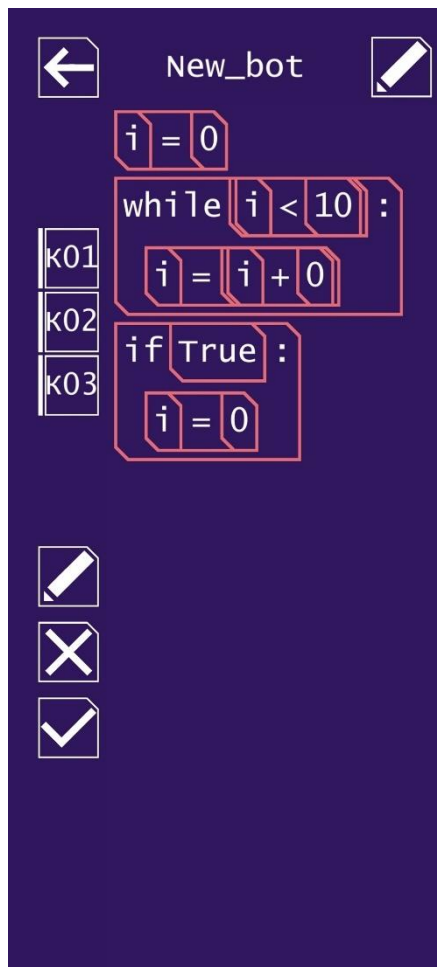


Рисунок 49 - Пример вывода кода бота в редакторе кода

Код бота хранится в базе данных в текстовом формате, поэтому для представления его в редакторе кода необходимо преобразование кода в объектную модель. Для этого используется библиотека лексического анализа ANTLR4. С помощью этой библиотеки код бота преобразуется в кодовое дерево, которое в дальнейшем преобразуется в объектную модель, необходимую для отображения на экране редактора.

При сохранении ботов в базе данных необходимо обратное преобразование объектов интерфейса в объектную модель программы, а дальше — в текст. Для получения объектной модели используется

UIToBlockConverter. Полученная модель в дальнейшем преобразуется в текстовое представление для сохранения в базе данных.

Используемая объектная модель заключается в следующем: каждая кодовая единица реализует один из интерфейсов: IExpression или IStatement. IStatement реализуют выражения присвоения значения переменным, конструкции условия и цикла. IExpression реализуют значения (числа, значения правда и ложь, значения перечисления (enum), отвечающего за юниты), бинарные операции над значениями, скобки, переменные. Вызовы функций реализуют IExpression и IStatement.

### 3.9 Сценарии воронок

Регистрация после предпросмотра

1

Вход без авторизации

Событие: WithoutLoginButtonClicked

+

2

Возвращение на старто

Событие: ReturnToStartSceneButtonClicked

+

3

Регистрация

Событие: SuccessfulSignUp

+

+

Добавить шаг

Детальные настройки ^

События между шагами ⓘ

Разрешены

Запрещены

Окно воронки ⓘ

Без ограничений

Ограничить

Регистрация событий ⓘ

Из любой сессии

Из одной сессии

Подсчет метрик ⓘ

По пользователям

По устройствам

Рисунок 50 - Воронка «Регистрация после предпросмотра»

Сценарий воронки регистрации после предпросмотра определяет, какой процент пользователей решит создать учетную запись после использования приложения в качестве наблюдателя.

Редактирование бота после матча

1

Участие в матче

Событие: MatchFinished

+

2

Редактирование бота

Событие: BotEditorOpened

+

+

Добавить шаг

Детальные настройки ^

События между шагами ⓘ

Разрешены

Запрещены

Окно воронки ⓘ

Без ограничений

Ограничить

Регистрация событий ⓘ

Из любой сессии

Из одной сессии

Подсчет метрик ⓘ

По пользователям

По устройствам

Рисунок 51 - Воронка «Редактирование бота после матча»

Сценарий воронки редактирования бота после матча призвана определить, какое количество пользователей решит отредактировать своего бота после участия этого бота в матче.

Просмотр хода матча после его завершения

1

Участие в матче

Событие: MatchFinished

+

2

Просмотр хода матча

Событие: MatchPlayBackButtonClicked

+

+

Добавить шаг

Детальные настройки ^

События между шагами ⓘ

Разрешены

Запрещены

Окно воронки ⓘ

Без ограничений

Ограничить

Регистрация событий ⓘ

Из любой сессии

Из одной сессии

Подсчет метрик ⓘ

По пользователям

По устройствам

Рисунок 52 - Воронка «Просмотр хода матча после его завершения»

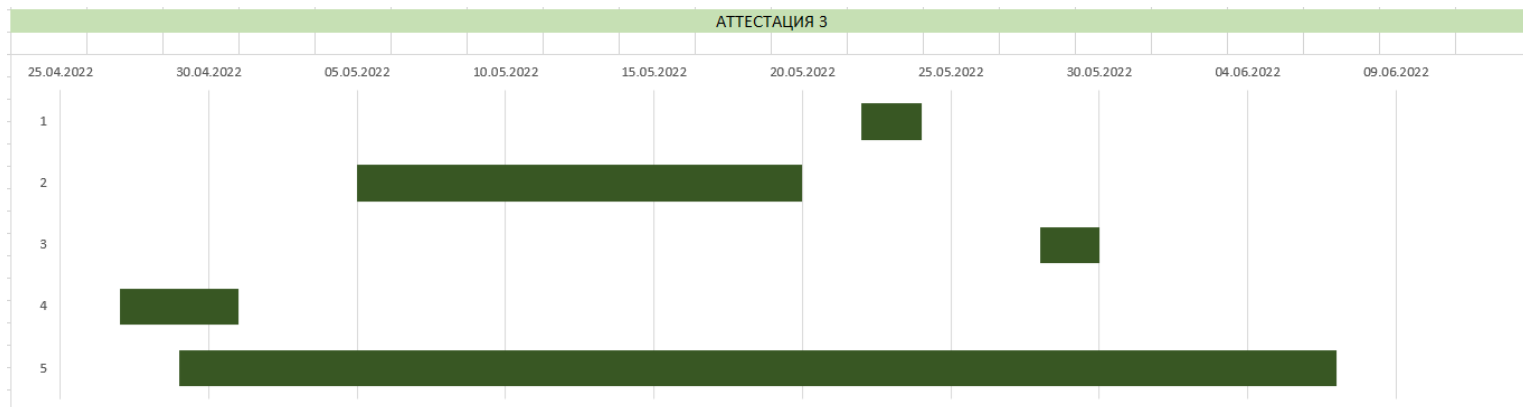
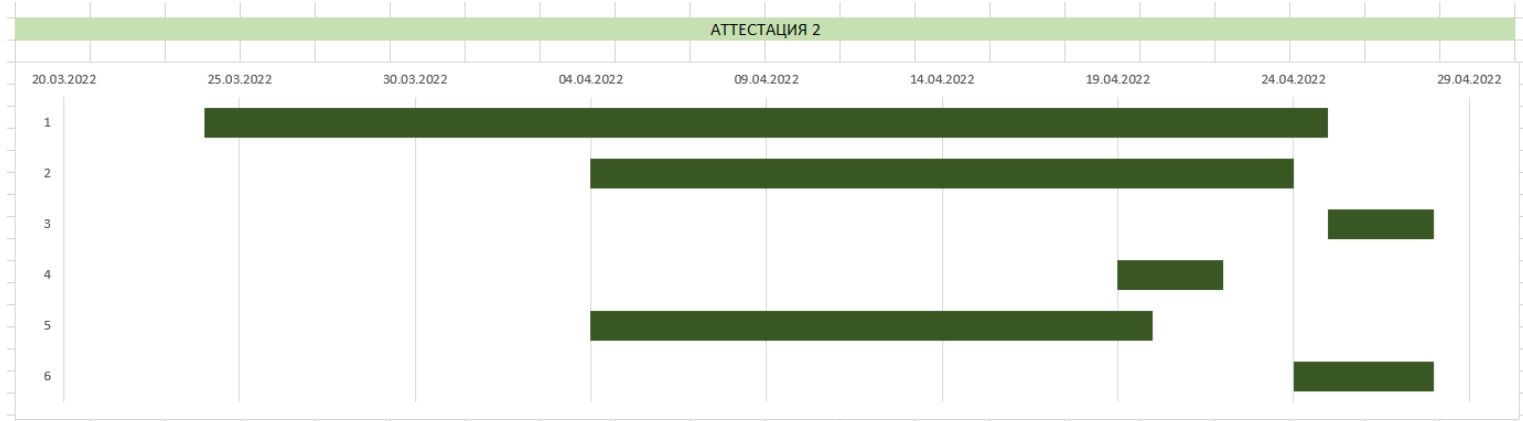
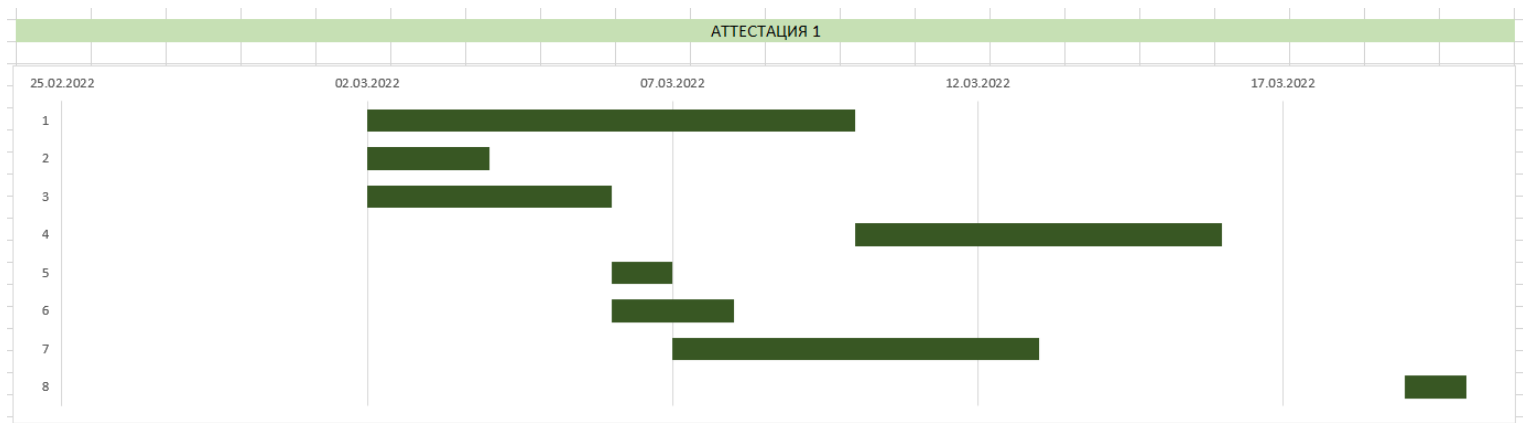


Сценарий воронки просмотра хода матча определяет, какое количество пользователей решит проанализировать матч, посмотрев его ход после получения результата.

### 3.10 Календарный план

При разработке приложения использовался следующий календарный план:

| Задача                                     | Дата начала | Дата окончания | Длительность |
|--|-------------|----------------|--------------|
| АТТЕСТАЦИЯ 1                               |             |                |              |
|  | 25.02.2022  | 20.03.2022     | 23           |
| Анализ предметной области                  | 02.03.2022  | 10.03.2022     | 8            |
| Анализ аналогов                            | 02.03.2022  | 04.03.2022     | 2            |
| Постановка задачи                          | 02.03.2022  | 06.03.2022     | 4            |
| Разработка макетов интерфейса              | 10.03.2022  | 16.03.2022     | 6            |
| Определение используемой платформы         | 06.03.2022  | 07.03.2022     | 1            |
| Построение use case диаграммы              | 06.03.2022  | 08.03.2022     | 2            |
| Создание доски Trello и репозитория GitHub | 07.03.2022  | 13.03.2022     | 6            |
| Корректировка технического задания         | 19.03.2022  | 20.03.2022     | 1            |
| АТТЕСТАЦИЯ 2                               |             |                |              |
|  | 24.03.2022  | 29.04.2022     | 36           |
| Реализация слоя доступа к БД               | 24.03.2022  | 25.04.2022     | 32           |
| Реализация интерфейса                      | 04.04.2022  | 24.04.2022     | 20           |
| Построение UML диаграмм                    | 25.04.2022  | 28.04.2022     | 3            |
| Подключение swagger                        | 19.04.2022  | 22.04.2022     | 3            |
| Разработка модели кода бота                | 04.04.2022  | 20.04.2022     | 16           |
| Создание сценариев воронок                 | 24.04.2022  | 28.04.2022     | 4            |
| АТТЕСТАЦИЯ 3                               |             |                |              |
|  | 27.04.2022  | 08.06.2022     | 42           |
| Реализация парсера кода бота               | 22.05.2022  | 24.05.2022     | 2            |
| Реализация модуля симуляции матчей ботов   | 05.05.2022  | 20.05.2022     | 15           |
| Реализация модуля Google авторизации       | 28.05.2022  | 30.05.2022     | 2            |
| Размещение backend-части и БД на хостинге  | 27.04.2022  | 01.05.2022     | 4            |
| Написание курсового проекта                | 29.04.2022  | 07.06.2022     | 39           |



## Заключение

В ходе выполнения данного курсового проекта был выполнен анализ предметной области и аналогов разрабатываемого приложения.

Для разработки приложения были разработаны макеты интерфейса, выбрана платформа приложения, построены UML диаграммы.

Для контроля версий был создан репозиторий GitHub.

При разработке приложения было реализовано следующее:

- Пользовательский интерфейс
- Слой доступа к базе данных
- Модель кода бота
- Механика игры
- Парсер кода бота
- Модуль авторизации

Были созданы сценарии воронок, подключен swagger. Back-end часть приложения и база данных были размещены на хостинге.

Разработанное приложение удовлетворяет поставленным требованиям.

Все поставленные задачи были выполнены.

В качестве дальнейшего развития проекта рассматриваются следующие усовершенствования:

- Реализация анимации для отображения хода матчей;
- Расширение набора юнитов и возможностей редактора кода;
- Реализация возможности проведения турниров;
- Создание IOS версии приложения;
- Реализация новых режимов игры.

### Список использованных источников

1. Документация Unity <https://docs.unity3d.com/Manual/index.html>
2. Документация C# <https://docs.microsoft.com/en-us/dotnet/csharp/>
3. Руководство ANTLR4 <https://pragprog.com/search/?q=the-definitive-antlr-4-reference>
4. Документация J2V8 <https://v8.dev/docs>
5. Подключение AppMetrica в приложение на Unity <https://appmetrica.yandex.ru/docs/mobile-sdk-dg/plugins/unity/unity-plugin.html>
6. Интеграция Google Sign-In в Android приложение <https://developers.google.com/identity/sign-in/android/sign-in>
7. Статья [https://new-retail.ru/business/rynok\\_mobilnykh\\_igr\\_v\\_mire\\_i\\_v\\_rossii8879/](https://new-retail.ru/business/rynok_mobilnykh_igr_v_mire_i_v_rossii8879/)
8. Статья <https://www.blog.udonis.co/mobile-marketing/mobile-games/marketing-mobile-strategy-game>
9. Документация Swagger <https://springdoc.org/#getting-started>
10. Документация Spring Data <https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#repositories.query-methods>
11. Документация Spring Boot <https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#documentation.first-step>