МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет вычислительной техники

Кафедра «Вычислительная техника»

Направление подготовки: 09.04.01 «Информатика и вычислительная техника»

Магистерская программа: «Прикладной искусственный интеллект»

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

на тему

БОТ-ВЕДУЩИЙ С ЭЛЕМЕНТАМИ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА ДЛЯ РОЛЕВОЙ ИГРЫ

Студент 05.06.2026 Кочергин Владислав Павлович
(подпись, дата) (ФИО полностью)

Руководитель 05.06.2026 Карамышева Н.С.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр 05.06.2026 Бычков А.С.
(подпись, дата) (фамилия, инициалы)

Рецензент _____ Севастьянов А.В.
(должность, место работы) (подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 22.05.2026 г. № 14)

Заведующий кафедрой _____ Митрохин М.А.
(подпись) (фамилия, инициалы)

Работа защищена с отметкой отлично (протокол заседания ГЭК от 23 июня 2026 г. № 27)

Секретарь ГЭК _____ Гурин Е.И.
(подпись) (фамилия, инициалы)

Пенза 2026

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет	*Кафедра*
вычислительной техники	«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
_____М.А. Митрохин
«20» 09 2024 г.

ЗАДАНИЕ
НА МАГИСТЕРСКУЮ ДИССЕРТАЦИЮ

1. Студент Кочергин Владислав Павлович гр. 24ВВИм1 факультета ВТ

2.Тема работы «Бот-ведущий с элементами искусственного интеллекта для ролевой игры»

Тема утверждена приказом ПГУ № 920/0 от "20"сентября 2024г.

3. Руководитель магистерской диссертации Карамышева Надежда Сергеевна

4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать программу для чат-бота с функциями работы с искусственным интеллектом, который сможет помогать пользователю совершать наиболее правильные ходы в игре «Шашки». Для поставленной цели требуется обучить две модели для определения игровой доски, фигур и их цветов. Взаимодействие с ботом должно быть удобным и интуитивно понятным. Бот должен быть рассчитан на игроков разного уровня опыта.

5. Перечень подлежащих разработке вопросов

Требуется произвести анализ предметных областей: изучить существующие аналогичные сайты, приложения или боты для помощи в игре, выделить их достоинства и недостатки. Определиться с языком программирования, который подойдет для создания графического интерфейса. Выбрать архитектуру приложения и вспомогательные средства разработки. Изучить функционал библиотек для обеспечения безопасности и сохранности данных.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Выбор темы и разработка индивидуального плана проведения работ	5%	21.09.2024 – 15.10.2024	
2	Разработка интерфейса системы	5%	16.10.2024 – 07.11.2024	
3	Подготовка датасетов для обучения	36%	08.11.2024 – 05.09.2025	
4	Подготовка моделей	17%	05.10.2025 – 09.01.2026	
5	Разработка телеграм-бота	5%	10.01.2026 – 08.02.2026	
6	Тестирование	8%	09.02.2026 – 28.02.2026	
7	Отладка	9%	01.03.2026 – 15.04.2026	
8	Описание взаимодействия пользователя с системой	9%	16.04.2026 – 06.05.2026	
9	Оформление отчёта	6%	07.05.2026 – 21.05.2026	

Дата выдачи задания "20" сентября 2024 г.

Руководитель магистерской диссертации _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан факультета _____ А.Р. Якимова
19.06.2026 *(подпись, дата)*

РЕФЕРАТ

Магистерская работа содержит 94 листа, 16 рисунков, 23 источника, 2 приложения

ШАШКИ, ЧАТ-БОТ, ПОМОЩНИК, МАШИННОЕ ОБУЧЕНИЕ, НЕЙРОСЕТЬ, ИИ-МОДЕЛЬ, ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ, СБОР ДАННЫХ, ОБРАБОТКА ДАННЫХ, РАСПОЗНАВАНИЕ ПО ФОТО, PYTHON, ТЕЛЕГРАМ, LIDRAUGHTS.ORG, CHECKERS.COM, PYCHARM, IDE, ТЕСТИРОВАНИЕ, РЕАЛИЗАЦИЯ, РАСПОЗНАВАНИЕ, ТРАНСФОРМИРОВАНИЕ

Основная цель работы заключается в разработке интеллектуального чат-бота для мессенджера Telegram, способного находить и подсказывать оптимальные ходы в шашечной партии.

Объектом исследования в магистерской диссертации выступают процессы и методы автоматического распознавания игровой ситуации по фото, а также совместная работа компьютерного зрения с логическими движками

Предметом исследования являются свёрточные нейронные сети, методы детекции объектов, алгоритмы коррекции перспективы и схемы взаимодействия компонентов системы.

В рамках данной работы проведен обзор аналогов, сбор и обработка данных, обзор алгоритмов машинного обучения, исследование взаимодействия между пользователем и сервером, обучение моделей и разработка бота.

В результате магистерской работы было подготовлено 2 датасета для обучения двух моделей для точного распознавания шашечной доски и фигур и разработана программа бота, использующая эти модели. Проведено тестирование качества работы моделей и самого бота.

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ 5
ВВЕДЕНИЕ 6
1 Постановка задачи 8
2 Анализ аналогов системы для ведения игры в шашки с элементами искусственного интеллекта 10
2.1 «Lidraughts.org» 11
2.2 «Checkers.com» 12
2.3 «Шашки» 14
3 Выбор инструментов разработки 15
3.1 Выбор расположения ИИ-помощника 15
3.2 Выбор языка программирования 16
3.3 Выбор среды разработки 17
3.4 Выбор фреймворков для реализации проекта 18
4 Обзор алгоритмов машинного обучения 20
5 Требования к функциональным характеристикам 22
6 Требования к организации входных данных 24
7 Исследование взаимодействия между приложением и пользователем в рамках клиент-серверной архитектуры «long-polling» 26
7.1 Достоинства архитектуры «Long-polling» 27
7.2 Недостатки архитектуры «Long-polling» 27
8 Подготовка моделей для обработки изображений 29
9 Описание алгоритма 31
9.1 Обнаружение и выравнивание доски 31
9.2 Сегментация и анализ клеток 31
9.3 Создание цифровой копии и верификация 31
9.4 Игровой интеллект и рекомендации 32
10 Реализация 34
10.1 Файл обучения модели распознавания доски и крайних клеток 34
10.2 Файл для заготовки датасета для обучения модели распознавания клеток, фигур и их цветов 35
10.3 Файл обучения модели распознавания клеток, фигур и их цветов 37
10.4 Файл обучения модели распознавания клеток, фигур и их цветов 41

11 Разработка макета пользовательского интерфейса 46
12 Тестирование основного функционала 51
ЗАКЛЮЧЕНИЕ 59
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 61
ПРИЛОЖЕНИЕ А 64
ПРИЛОЖЕНИЕ Б 88

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящей работе применяют следующие сокращения и обозначения

ИИ – Искусственный интеллект

API (Application Programming Interface) – Программный интерфейс приложения

CNN (Convolutional Neural Network) – Свёрточная нейронная сеть

CSV (Comma-Separated Values) – Текстовый формат, предназначенный для представления табличных данных

HTTP (HyperText Transfer Protocol) – Протокол передачи гипертекста

IoU (Intersection over Union) – Метрика «пересечение поверх объединения» (используется для оценки точности совпадения границ распознанных объектов)

JSON (JavaScript Object Notation) – Простой текстовый формат обмена данными

mAP / mAP50 (Mean Average Precision) – Усредненная средняя точность (ключевая метрика оценки качества детекции объектов алгоритмами компьютерного зрения)

ML (Machine Learning) – Машинное обучение

MLP (Multilayer Perceptron) – Многослойный перцептрон (базовая архитектура нейронных сетей прямого распространения)

OpenCV (Open Source Computer Vision Library) – Библиотека алгоритмов компьютерного зрения, обработки изображений и численных алгоритмов с открытым исходным кодом

SVM (Support Vector Machine) – Метод опорных векторов

URL (Uniform Resource Locator) – Единый указатель ресурса

YOLO (You Only Look Once) – Алгоритм и архитектура свёрточных нейронных сетей, применяемая для детекции и распознавания объектов в реальном времени

ВВЕДЕНИЕ

Современные технологии искусственного интеллекта (ИИ) открывают новые возможности для создания интерактивных систем, способных адаптироваться к действиям пользователя. Одной из перспективных областей применения таких технологий являются настольные игры, такие как шашки. В эту игру регулярно играют миллионы людей и немало из них являются новичками, желающими быстро научиться побеждать. Данный проект посвящён разработке телеграмм-бота с ИИ, способного сопровождать игрока по ходу всей игры, если он того пожелает. Бот будет помогать совершать правильные ходы и подсказывать как правильно поступить в той или иной ситуации. Взаимодействие будет выполняться путём предоставления боту текущего расположения фигур на доске через отправку фотографии. После чего бот сообщит игроку наиболее выигрышный ход из доступных.

Популярность и интерес к шашкам не угасает в современном обществе, но разница в опыте между игроками ограничивает вовлечение новых игроков. На данный момент не существует идеальной альтернативы в интернете, которая была бы способна сопровождать игрока по ходу игры для победы.

Цель данной работы заключается в анализе аналогов системы для ведения игры в шашки с элементами искусственного интеллекта, и популярных библиотек на языке Python использующихся для обработки естественного языка, создания ИИ моделей и их обучения. Также требуется провести обзор алгоритмов машинного обучения, методов сбора данных для составления датасетов. Необходимо провести анализ достоинств и недостатков клиент-серверной архитектуры «Long-polling», а также разработать пользовательский интерфейс

Гипотезой исследования является отсутствие приложений или сервисов, которые бы обладали возможностью анализировать игровую доску с фигурами по фотографии и выявлять наилучший из доступных ходов.

В результате научно-исследовательской работы был проведено тестирование качества распознавания необходимых объектов на фотографиях,

трансформации этих фотографий и определения лучших ходов. Также был проведён подробный разбор кода модулей программы.

1 Постановка задачи

В рамках текущего исследования создается умный телеграм-бот, который сможет подсказывать игроку в русские шашки. Текущая позиция будет распознаваться программой по фотографии, находящиеся на доске фигуры — классифицироваться, а пользователю — выдаваться рекомендация для наиболее выигрышного хода. Чтобы достичь этой цели, нам необходимо решить следующие задачи:

Проанализировать предметную область. Сюда входит разбор правил игры в русские шашки, изучение методов компьютерного зрения для поиска досок и фигур, а также оценка готовых алгоритмов расчета позиций. Дополнительно мы исследуем уже созданные аналогичные шашечные приложения и боты, чтобы найти их слабые места и составить список требований к собственной системе.

Разработать общую архитектуру. Была спроектирована система, связывающая воедино три компонента: телеграм-бота для общения с пользователем, свёрточную нейросеть в роли модуля компьютерного зрения и специальный шашечный движок, который вычисляет оптимальные ходы.

Сделать и обучить модель, классифицирующую клетки шашечной доски. Ей предстоит распознавать пять базовых вариантов: пустую клетку, белую или чёрную простую шашку, белую или чёрную дамку. Для обучения подготовлены реальные и синтетические наборы данных, а также задействованы технологии аугментации (искусственного расширения) датасета.

Построить конвейер, обрабатывающий изображения. На фотографии сначала будет обнаруживаться сама доска, затем система исправит перспективу и разделит поле на отдельные клетки. После этого обученная модель распознает содержимое каждой клетки, а программа соберет эти данные в единое цифровое описание позиции.

Подключить шашечный движок. Нам удалось настроить модуль, который будет анализировать распознанную позицию и подбирать лучший

ход для того цвета, который выберет пользователь. Результат вычислений бот выдаст в виде понятного текстового сообщения, где укажет клетку отправления и клетку прибытия фигуры.

Создать телеграм-бота на базе асинхронной библиотеки aiogram. Бот будет принимать от пользователей команды и фотографии, обрабатывать картинки в фоновом режиме и присылать ответы.

Процессы автоматического распознавания игровой ситуации по фото в настольных играх, а также совместная работа компьютерного зрения с игровыми движками для помощи игрокам выступают объектом нашего исследования. При этом предметом исследования стали свёрточные нейросети, методы компьютерного зрения, обработка картинок с нарушенной перспективой, а также схемы взаимодействия телеграм-бота с движком и обученной моделью.

Чтобы прийти к поставленным результатам, мы используем методы машинного и глубокого обучения (свёрточные нейросети, трансферное обучение, аугментацию данных), технологии компьютерного зрения (поиск контуров, сегментацию, перспективную коррекцию), а также подходы к интеграции компонентов. Бот пишется на языке Python с применением трёх ключевых фреймворков и библиотек: OpenCV, aiogram и PyTorch

2 Анализ аналогов системы для ведения игры в шашки с элементами искусственного интеллекта

В настоящее время ботов, приложений или сайтов для помощи начинающим игрокам не так много и их функционал не позволяет помогать игроку по ходу его текущей партии.

Изучив данную область, удалось выделить основной функционал, который должна иметь идеальная система для качественного сопровождения и обучения новичков для игры в шашки: в первую очередь это возможность игры с искусственным интеллектом с настраиваемым уровнем сложности, а также режим игры с другим человеком как на одном устройстве, так и по сети. К обязательным функциям относятся подсказки ходов в текущей партии, запись и сохранение сыгранных партий, а также их последующий анализ с выделением ошибок. Полезным дополнением служат обучающие разделы, содержащие теорию, типовые комбинации и тактические приёмы. Однако все перечисленные возможности реализованы только в замкнутой виртуальной среде самого приложения или сайта, что заставляет игрока полностью перенести партию в цифровой формат и не позволяет получать помощь во время реальной игры на физической доске.

Главное новшество разрабатываемой системы, отсутствующее в известных аналогах, заключается в возможности анализа произвольной игровой позиции по обычной фотографии, сделанной пользователем. Бот не требует ручного ввода расположения фигур — достаточно прислать снимок текущего состояния доски. За счёт технологий компьютерного зрения и машинного обучения система самостоятельно распознаёт цвета шашек, отличает простые фигуры от дамок, определяет их точные координаты на поле и передаёт полученную позицию встроенному шашечному движку. В ответ пользователь получает рекомендацию наиболее выигрышного хода с указанием клетки, откуда и куда следует переместить фигуру. Таким образом, бот становится полноценным «умным помощником», способным

сопровождать игрока прямо во время реальной партии, не требуя переключения между приложениями и ручного переноса позиции.

Ниже представлен анализ существующих аналогов, которые способны помогать начинающему игроку научиться играть в шашки.

2.1 «Lidraughts.org»

Данный аналог представляет из себя бесплатный сайт–тренажёр для поэтапного обучения игре в шашки, а также есть возможность играть с компьютером на разных уровнях сложности и другими игроками.

Функционал сайта предлагает множество различных задачек разной сложности. Всего на сайте их насчитывается более 6000. Представляют из себя они различные игровые ситуации, с расставленными по доске фигурами, где нужно выстроить правильную последовательность перемещений своих шашек, для наиболее выигрышной ситуации. Помимо режима тренировки на сайте есть и возможность опробовать свои силы в игре либо с ИИ различной сложности, либо с реальным игроком. Можно как создавать самому игру, так и подключаться к чужим. Интерфейс приложения показан на рисунке 1.

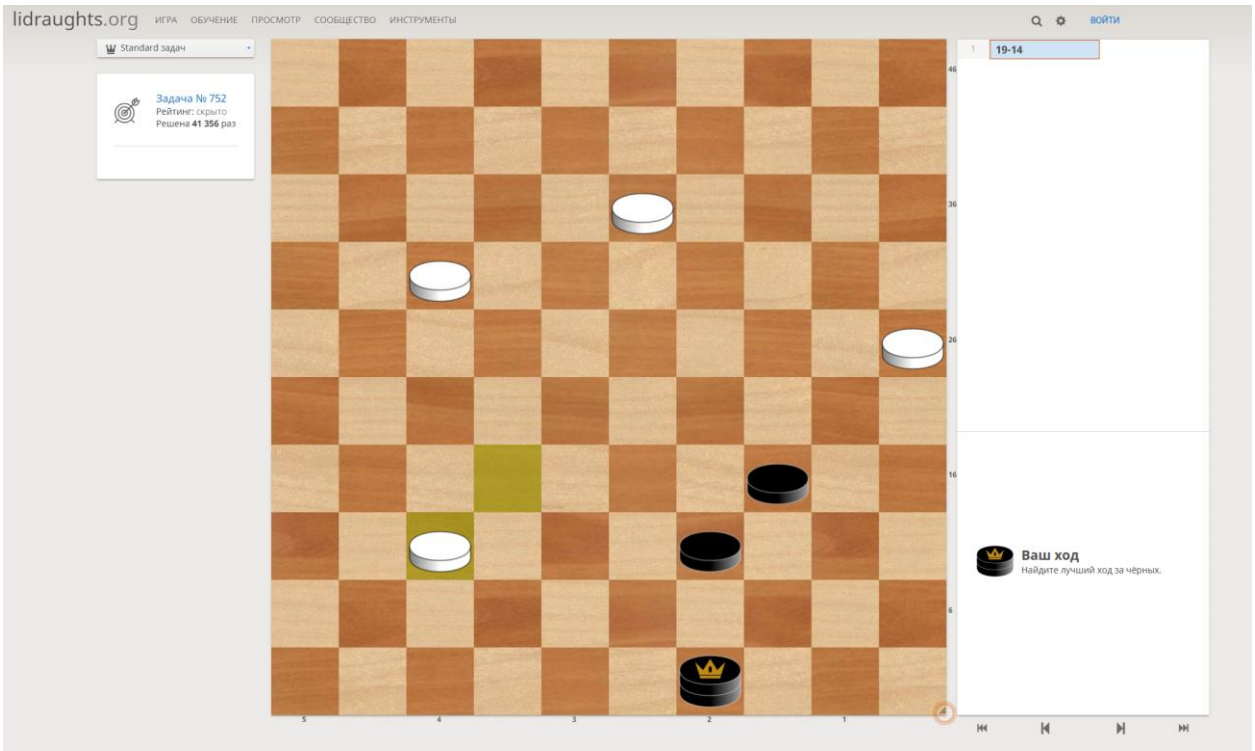


Рисунок 1 - Интерфейс на экране выполнения тренировочного задания

Преимуществом данного сайта является большая база различных тренировочных заданий, выполнив которые можно достичь высокого уровня профессионализма в игре в шашки. Также стоит отметить, что это сайт и он способен работать на любых устройствах с доступом в интернет.

Из недостатков отсутствуют подсказки во время игры с ИИ. В случае, когда начинающий игрок выполнил множество тренировочных задач и решил перейти к реальной игре с ИИ, то он может столкнуться с тем, что даже на самой низкой сложности отсутствуют какие-либо подсказки или советы, которые помогут победить в партии. Из-за отсутствия помощи или советов по текущей игре новичок может вовсе бросить попытки научиться играть. А также отсутствует какая-либо возможность подсказывать какой ход сделать следующим при игре в реальной жизни.

2.2 «Checkers.com»

Следующий рассматриваемый аналог является уже мобильным приложением и наиболее продвинутым среди остальных подобных приложений. Оно является бесплатным и имеет самый расширенный функционал для обучения и игры в шашки.

Открыв приложение, мы сразу попадаем в меню выбора необходимой нам функции. Это может быть, как классическая игра с другими пользователями, так и возможность играть с ботом разной сложности. На этом возможности приложения не заканчиваются. Имеются также режимы для обучения и тренировки. Первым стоит обратить внимание на режим с прохождением уроков, который представляет из себя сводку правил игры в шашки с примерами. Этот режим полезен только игрокам, что ранее не играли в эту игру. Следующим режимом является тренажёр с задачами. Он представляет из себя уровневую систему, где чтобы открыть следующий, необходимо пройти предыдущий. Уровни же выглядят как различные игровые ситуации с расставленными по доске фигурами, где нужно решить какая последовательность перемещений шашек наиболее выигрышная. Сложность уровней также увеличивается с каждым выполненным заданием. И самый

интересный режим - это режим игры с тренером, где в роли тренера выступает бот, с которым можно сыграть обычную партию, но в любой момент можно его попросить дать подсказку какой ход сделать и объяснить почему. Этот режим является идеальным для закрепления полученных знаний после прохождения уровней с задачами. Интерфейс приложения показан на рисунке 2.

Рисунок 2 - Интерфейс главного меню Checkers.com

Преимуществом данного приложения является разнообразный функционал, позволяющий обучить игрока с нуля и сопроводить его до

высокого уровня профессионализма в игре в шашки. Стоит отметить, что данное приложение может запускаться без доступа к интернету и все функции, кроме игры по сети, будут доступны в полной мере.

Самым главным недостатком является то, что доступ ко всем тренировочным уровням и к игре с тренером предоставляется по подписке. Игра по сети и с ботами доступны и с бесплатным уровнем подписки. В приложении также отсутствует какая-либо возможность подсказывать какой ход сделать следующим при игре в реальной жизни.

2.3 «Шашки»

Следующий рассматриваемый аналог не является одним сайтом или приложением. Хочется объединить все неупомянутые аналоги в одну категорию по причине их схожести по функционалу и возможностям.

В большинстве своём сайты и приложения представляют из себя простую игру в шашки с возможностью играть, как с реальными игроками, так и с ботами разной сложности, не имея никакой обучающих или тренировочных функций. Некоторые приложения сильно монетизированы. Они позволяют на плату изменять внешний вид своих фигур, доски, аватарок. Также большинство имеют внутри систему рейтинга или рангов, которые зависят от количества и частоты побед.

Преимуществом можно выделить лишь разнообразие таких сайтов и приложений, которые отличаются своим внешним видом, рейтинговой системой и числом одновременно играющих игроков.

Главным недостатком является полное отсутствие обучающих материалов и тренировочных упражнений для новичков. Опытные игроки также не смогут совершенствовать свои навыки, без потери рейтинга, никак, кроме как игры с ботом. Ни в одном из приложений или сайтов не было обнаружено возможности подсказывать какой ход сделать следующим при игре в реальной жизни.

3 Выбор инструментов разработки

3.1 Выбор расположения ИИ-помощника

Размещение помощника на веб-сайте открывает широкие возможности для кастомизации интерфейса. Можно создать детализированное визуальное представление шашечной доски с анимацией ходов, добавить панель управления партией, историю ходов и дополнительные обучающие материалы. Сайт доступен с любого устройства, имеющего браузер, без установки дополнительного программного обеспечения. Также веб-решение проще монетизировать, например, с помощью рекламных баннеров или платной подписки на расширенный функционал. Однако главным недостатком такого подхода являются высокие требования к ресурсам разработки: необходимо создавать полноценный фронтенд и бэкенд, обеспечивать безопасность, разворачивать и поддерживать серверную инфраструктуру, что может оказаться избыточно сложным.

В случае размещения помощника в качестве телеграм-бота преимущества становятся иными. Telegram уже обладает многомиллионной аудиторией, и бот доступен пользователям без дополнительной регистрации — достаточно найти его по имени. Большинство пользователей пользуются Telegram со смартфонов, что идеально сочетается с задачей фотографирования физической шашечной доски прямо во время партии. Боты в Telegram изначально поддерживают удобные механики: обработку текстовых команд, приём и отправку изображений, работу с кнопками и встроенными клавиатурами. Вся история взаимодействия сохраняется в облаке Telegram, что избавляет от необходимости создавать собственную систему хранения прогресса пользователя. Создание и хостинг телеграм-бота не требует значительных вычислительных мощностей — для небольших проектов достаточно бесплатных или недорогих облачных серверов. Основным минусом является ограниченная гибкость интерфейса: нельзя полностью переопределить внешний вид бота под свой дизайн, а визуализация доски возможна только через отправку готовых изображений. Тем не менее для

проекта, где основное взаимодействие строится на приёме фотографий от пользователя и выдаче текстового ответа с координатами хода, этих ограничений вполне достаточно. Учитывая ограниченные ресурсы и наличие большого числа готовых библиотек для создания телеграм-ботов на Python, окончательным выбором становится платформа Telegram.

3.2 Выбор языка программирования

Для создания телеграм-бота с функцией распознавания шашечной позиции и интеграцией с игровым движком можно использовать несколько языков программирования, однако наилучшим выбором является Python. Этот язык обладает самой популярной и удобной библиотекой для работы с Telegram Bot API, что позволяет быстро реализовать приём фотографий, отправку сообщений и обработку команд. Кроме того, Python является безусловным лидером в области компьютерного зрения и машинного обучения: для него существуют такие мощные фреймворки, как TensorFlow, PyTorch, Keras, а также библиотеки для классической обработки изображений OpenCV и Pillow. В рамках проекта потребуется обучать нейросеть для классификации шашечных фигур по клеткам, и на Python это делается наиболее эффективно благодаря обширной экосистеме. Также Python легко интегрируется с внешними шашечными движками через вызовы командной строки или использование готовых библиотек наподобие python-draughts. При необходимости сохранять данные о пользователях (например, их цвет или историю запросов) можно подключить SQLite или PostgreSQL с минимальными усилиями. Наконец, Python отличается относительно низким порогом входа, что ускоряет разработку прототипа.

Среди альтернатив стоит упомянуть Java и C#, но они менее подходят для данного проекта. Java имеет библиотеки для создания телеграм-ботов, однако они хуже документированы и требуют больше настроек, чем python-аналоги. Для компьютерного зрения и глубокого обучения в Java существует фреймворк DeepLearning4j, но он значительно уступает TensorFlow и PyTorch по распространённости и удобству работы с предобученными моделями. C#

также предоставляет средства для работы с Telegram и машинным обучением (например, ML.NET), однако интеграция с нейросетями, особенно свёрточными, на C# более трудоёмка, а перенос моделей, обученных в популярных python-средах, сопряжён с дополнительными сложностями. Оба этих языка имеют более крутую кривую обучения, что при ограниченных сроках разработки является существенным минусом. Таким образом, для реализации чат-бота с компьютерным зрением и элементами ИИ язык Python выбирается как наиболее рациональный.

3.3 Выбор среды разработки

Для разработки на Python существует множество интегрированных сред, каждая из которых наилучшим образом подходит для определённых задач. В данном проекте можно выделить две основные активности: написание кода самого телеграм-бота и обучение модели компьютерного зрения для классификации клеток. Для каждой из этих активностей целесообразно выбрать специализированную среду.

PyCharm является одной из самых распространённых IDE для Python. Она предоставляет мощные инструменты для написания и отладки кода, удобную навигацию по проекту, интеграцию с системами контроля версий и виртуальными окружениями. PyCharm отлично подходит для реализации логики телеграм-бота: обработки сообщений, вызова функций компьютерного зрения, взаимодействия с базой данных и шашечным движком. С другой стороны, для обучения нейросетевой модели и анализа данных лучше подходит Jupyter Notebook. Эта интерактивная среда позволяет выполнять код блоками, визуализировать промежуточные результаты (например, отображать сегментированные клетки доски, строить графики функции потерь и точности), легко экспериментировать с гиперпараметрами и сохранять ход обучения. Jupyter Notebook идеально приспособлен для исследовательских задач, связанных с машинным обучением и обработкой изображений. Остальные среды, такие как Visual Studio Code и Spyder, также могут быть использованы, но не дают столь явных преимуществ для разделения процессов

разработки и обучения. Поэтому в рамках проекта принято решение использовать PyCharm для разработки основного кода бота и интеграционной логики, а Jupyter Notebook — для создания, обучения и валидации модели классификации шашечных клеток.

3.4 Выбор фреймворков для реализации проекта

Для успешной реализации проекта необходимо подобрать библиотеки, покрывающие три ключевые области: компьютерное зрение и обработка изображений, создание и обучение нейросетевой модели, а также взаимодействие с Telegram и шашечным движком.

Для работы с фотографиями шашечной доски потребуется библиотека OpenCV. Она предоставляет функции для обнаружения контуров доски, перспективной коррекции, сегментации на отдельные клетки и предварительной обработки изображений. Дополнительно для базовых операций с картинками (изменение размера, конвертация цветовых пространств) будет использоваться Pillow. Для построения и обучения модели классификации клеток основным фреймворком выбран PyTorch. Он позволяет быстро создать свёрточную нейросеть, организовать аугментацию данных и обучить модель распознавать пять классов: пусто, белая простая, чёрная простая, белая дамка, чёрная дамка. Для работы с числовыми данными и подготовки наборов для обучения пригодятся библиотеки NumPy и Pandas. NumPy будет использоваться для преобразования матриц клеток и работы с массивами, а Pandas — для анализа и балансировки датасета.

Для создания телеграм-бота выбрана библиотека aiogram. Это современная асинхронная библиотека, которая хорошо подходит для обработки фотографий и команд, поддерживает вебхуки и лонг-пуллинг, а также обладает удобной архитектурой с использованием роутеров и фильтров. Альтернатива в виде python-telegram-bot также возможна, но aiogram считается более производительной и удобной для асинхронных операций, что важно при ожидании ответа от шашечного движка. Для интеграции с шашечным движком потребуется библиотека python-draughts, которая

предоставляет правила русских шашек, представление позиции в виде FEN-строки и функцию поиска лучшего хода. Движок можно запускать как отдельный процесс, но для упрощения прототипа достаточно встроенной оценки позиции. Наконец, для взаимодействия между модулем распознавания и движком не потребуется отдельного фреймворка вроде FastApi, так как вся логика будет выполняться в рамках одного приложения: полученная от пользователя фотография обрабатывается OpenCV и PyTorch, формируется матрица позиции, которая передаётся в python-draughts для расчёта хода, и результат отправляется пользователю. Таким образом, выбранные библиотеки полностью покрывают потребности проекта и обеспечивают его успешную реализацию.

4 Обзор алгоритмов машинного обучения

Изменения освещения, ракурса и появление бликов вызывают серьезные сбои в работе традиционных алгоритмов компьютерного зрения, таких как сопоставление с эталонами и пороговая бинаризация. Размеченные данные и предварительное обучение для них не требуются, однако корректно обрабатывать они способны только идеальные снимки, из-за чего использовать их для реальных пользовательских фотографий невозможно. В свою очередь, метод опорных векторов заставляет вручную извлекать текстовые дескрипторы или гистограммы из изображений. К переобучению на малом объеме данных этот подход устойчив, но точность распознавания резко падает при сильных колебаниях света, а итоговое качество напрямую завязано на правильный выбор признаков. Из-за неспособности автоматически находить пространственные иерархические структуры нейросетевым подходам уступают и ансамблевые методы (например, случайный лес или градиентный бустинг). Они тоже работают на ручных признаках и показывают хорошие результаты лишь с табличной информацией.

Самым мощным инструментом анализа графических данных на сегодняшний день считаются свёрточные нейронные сети. Обучаясь прямо на сырых пикселях, они способны в автоматическом режиме выделять любые уровни признаков: от простейших углов и краев до сложных геометрических форм фигур. Достаточно использовать неглубокую сеть, чтобы быстро классифицировать отдельные клетки и гарантировать высокую скорость работы системы. Нейросети класса CNN не боятся перепадов освещения или смены ракурса, а дамки распознаются без предварительного ручного описания их параметров. Необходимость в наличии размеченного датасета считается их главным минусом. Мы можем обойти это ограничение, если создадим синтетический набор данных или применим трансферное обучение с использованием уже предобученных моделей.

Для проекта нашего телеграм-бота свёрточная нейронная сеть видится наиболее подходящим алгоритмом. Модель идеально сочетает в себе высокую

точность, скромные вычислительные затраты и устойчивость к сложным условиям съемки. Было решено взять за основу компактную архитектуру CNN с несколькими сверточными слоями, возможности которой расширены с помощью аугментации данных для улучшения обобщающей способности.

5 Требования к функциональным характеристикам

Нам необходимо разработать телеграм-бота для помощи в игре в русские шашки. Программа призвана стать удобным инструментом как для новичков, так и для опытных игроков, которые хотят получить квалифицированную подсказку во время реальной партии на обычной физической доске. Бот разворачивается в мессенджере Telegram и наделяется следующим функционалом:

– приём исходных данных. От пользователя принимается фотография шашечной доски (в любом ракурсе и при любом освещении), а также информация о выбранном цвете фигур. При частичном обнаружении игрового поля или недостаточном качестве снимка бот отправляет запрос на повторное фото, дополняя его советами по улучшению кадра,

– автоматическое распознавание. На фотографии вычисляются границы игровой доски, после чего выполняется перспективная коррекция для получения плоского вида сверху. Затем поле сегментируется на 64 отдельные клетки, и система определяет содержимое каждого тёмного участка: пустое место, простая шашка (белая/чёрная) или дамка (белая/чёрная). В соответствии с правилами шашек светлые клетки всегда распознаются как пустые,

– анализ позиции движком. На основе считанной ситуации формируется цифровая матрица размером 8×8, передаваемая затем встроенному шашечному движку для расчета наиболее выигрышного хода. При обнаружении нескольких равнозначных по силе вариантов бот предлагает пользователю сделать выбор самостоятельно или автоматически указывает любой из них,

– вывод результатов. Ответ выдается в понятном текстовом формате с фиксацией исходной клетки и клетки назначения по стандартной шахматной нотации (например, «c3-d4»). Дополнительно бот может отправлять изображение с наглядной визуализацией рекомендуемого хода,

– управление настройками. Пользовательские команды позволяют изменять уровень сложности (глубину просчёта ходов), оперативно менять цвет игрока или вызывать встроенную справку о правилах игры. База данных обеспечивает индивидуальное сохранение всех настроек для каждого человека,

– многопользовательский режим. Подсказки могут по очереди выдаваться сразу нескольким игрокам в рамках одного группового чата. Бот успешно различает участников группы и запоминает цвет фигур для каждого отдельного пользователя, что исключает путаницу в рекомендациях,

– ведение истории. База данных сохраняет историю запрошенных позиций и предложенных ходов. Это позволяет быстро вернуться к предыдущей ситуации или проанализировать прогресс обучения,

– обработка ошибок распознавания. Система корректно обрабатывает ситуации, когда компьютерное зрение выдает противоречивые результаты (например, на одной клетке найдены две фигуры). В таких случаях бот уведомляет пользователя о сбое распознавания и предлагает прислать новый снимок,

– обеспечение быстродействия. Время отклика от момента отправки картинки до выдачи готового ответа составляет не более пяти секунд при работе на стандартном облачном хостинге. Это предотвращает появление дискомфорта во время партии,

– масштабируемость системы. Базовый алгоритм рассчитан на работу с русскими шашками (обязательное взятие, ходы по тёмным клеткам, свободное перемещение дамки). Тем не менее, путем простой смены конфигурации движка мы можем расширить систему на другие популярные варианты: международные, английские, поддавки или бразильские шашки.

6 Требования к организации входных данных

Тщательно подготовленные наборы входных данных требуются для обучения и корректной работы телеграм-бота, распознающего шашечную позицию по фотографии. Все возможные вариации реальных игровых ситуаций, типов досок и условий съёмки должны покрываться этими материалами.

В первую очередь необходимо собрать или сгенерировать размеченный датасет изображений шашечных досок. Для этого подходят как синтетические изображения, созданные программно с различными расстановками фигур, так и реальные фотографии, сделанные на физических досках. Точная аннотация должна сопровождать каждое изображение: координаты углов доски и матрица 8×8, где для каждой клетки указано её состояние (пусто, белая простая, чёрная простая, белая дамка, чёрная дамка).

Мы подготавливаем выборку отдельных изображений клеток размером 80×80 пикселей, извлечённых из выровненных досок, чтобы обучить модель классификации. Пять классов должны быть сбалансированы в этой выборке. При этом для редких классов (дамки обоих цветов) применяется аугментация или дополнительный сбор данных, что помогает избежать перекоса.

Чтобы модель была устойчива к реальным условиям съёмки, входные данные включают фотографии, сделанные под разными углами обзора (от 30 до 45 градусов относительно плоскости доски), при различном освещении (дневной свет, искусственное освещение, смешанное, наличие теней и бликов), а также на четырех типах досок: деревянных, виниловых, картонных и пластиковых.

Изображения, где доска снята под сильным углом, на неоднородном фоне или частично закрыта руками, требуются для обучения этапа обнаружения поля на фотографии. Точные координаты четырёх углов доски должны содержаться в аннотациях к таким картинкам.

Тестовые позиции с известными оптимальными ходами необходимы для настройки и валидации шашечного движка, вычисляющего наилучший ход. Мы можем получить такие данные из открытых баз шашечных этюдов и партий мастеров или сгенерировать их с помощью самого движка на большой глубине

просчёта. Различные стадии игры (дебют, миттельшпиль и эндшпиль) должны охватываться этими позициями.

Был сформирован отдельный тестовый набор из реальных фотографий, не участвовавших в обучении, с ручной разметкой ожидаемого результата. Это необходимо для проверки корректности работы всего конвейера. Такой подход позволяет объективно оценить точность распознавания и правильность выдаваемых ходов.

При недостатке реальных фотографий применяется синтетическая генерация с последующим дообучением на небольшом количестве реальных снимков (техника трансферного обучения). Также мы можем обратиться к открытым датасетам шахматных досок и адаптировать их под шашки, изменив расстановку фигур и правила.

Структуру каталогов с разделением на обучающую, валидационную и тестовую выборки рекомендуется использовать для хранения и организации всех входных данных. Также подключается файл аннотаций в формате json или csv, содержащий ссылки на изображения и соответствующие им матрицы позиций.

7 Исследование взаимодействия между приложением и пользователем в рамках клиент-серверной архитектуры «long-polling»

Обученная модель будет развернута на сервере. В роли клиента может выступать любой пользователь Telegram, который открывает мессенджер через web-версию или мобильное приложение. Взаимодействие выстраивается следующим образом: запросы сначала отправляются в буфер на серверы Telegram, откуда их забирает сервер бота посредством HTTP-запросов. Извлеченное сообщение сразу отправляется на обработку, после чего система формирует ответ и пересылает его обратно пользователю. Сервер бота находится в режиме постоянного опроса серверов Telegram, удерживая соединение в ожидании новых данных до 30 секунд. При отсутствии активности за этот период отправляется пустой ответ, после чего система незамедлительно формирует следующий запрос в буфер и возобновляет ожидание. Наглядно взаимодействие можно увидеть на рисунке 3.

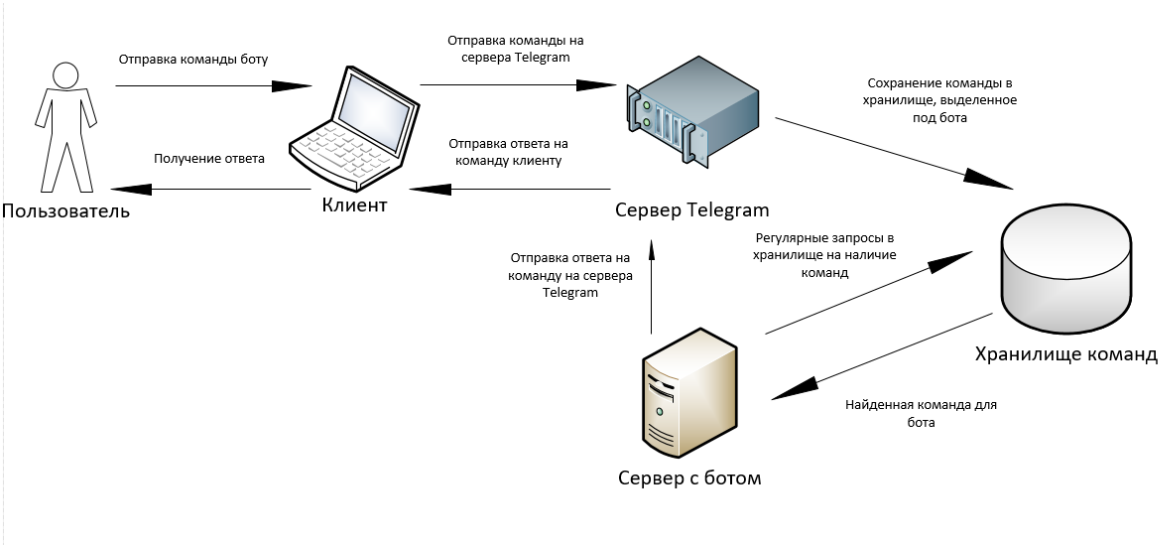


Рисунок 3 – Схема взаимодействия пользователя, клиента, и сервера

Выбранный подход открывает перед разработчиком несколько важных преимуществ.

7.1 Достоинства архитектуры «Long-polling»

– простота. Настроить стандартные механизмы обработки HTTP-запросов на стороне сервера гораздо легче, чем разворачивать полноценный WebSocket-сервер,

– эффективность. Этот метод выигрывает у обычного Polling. Данные приходят практически без задержек, поскольку сервер отправляет ответ в ту же секунду, когда они появляются, и ресурсы не тратятся на бесконечные пустые циклы,

– безопасность. За счет использования классического протокола HTTP запросы без проблем проходят через большинство файрволов и прокси-серверов,

– надежность. Каждая транзакция в формате «запрос-ответ» абсолютно независима. При внезапном разрыве соединения следующий цикл просто восстановит рабочий процесс.

7.2 Недостатки архитектуры «Long-polling»

– нагрузка на сеть. Трафик и ресурсы на парсинг расходуются сильнее, ведь под каждое сообщение создается полноценный HTTP-запрос со всеми сопутствующими заголовками,

– проблемы с задержками. Схема «запрос-ответ-запрос» неизбежно создает микропаузы, проигрывая постоянному двустороннему каналу WebSockets,

– сложность настройки таймаута. Слишком короткие промежутки ожидания заставляют систему постоянно перезапускать сессии, а излишне длинные повышают риск того, что соединение будет оборвано самим браузером или сетевым прокси,

– многопоточность. Каждое открытое ожидание занимает отдельный поток или процесс. Если к боту одновременно подключатся 10 000

пользователей, сервер должен будет удерживать 10 000 непрерывных соединений, что накладывает жесткие требования к оптимизации,

– зависимость от сети. И, наконец, работа приложения полностью прекращается в местах, где отсутствует стабильное подключение к Интернету.

Детальный разбор возможностей технологии «Long-polling» показывает, что для текущей стадии разработки проекта этот вариант подходит лучше всего. Его понятные принципы работы позволяют переключить внимание на более приоритетные задачи, а очевидные достоинства полностью перекрывают технологические недостатки.

8 Подготовка моделей для обработки изображений

На первом этапе перед нами стояла задача научить программу обнаруживать шашечную доску на любой фотографии и безошибочно определять четыре угла игрового поля. Современная нейросеть YOLO была взята за основу. Поскольку она изначально обучалась распознавать сложные объекты, нам оставалось лишь дообучить её на подготовленных снимках нашей доски.

Самым главным и ответственным делом стала правильная разметка кадров. Мы устанавливали точки строго по углам внутренних черно-белых клеток, а деревянная рамка с буквами и цифрами полностью игнорировалась. Исключительно за счет этого алгоритму удалось в дальнейшем идеально ровно вырезать само игровое поле.

Чтобы исключить привыкание программы к одинаковым ракурсам, во время тренировки мы намеренно усложняли процесс. Мы искусственно меняли масштаб, искажали перспективу, варьировали яркость и цветовую гамму кадров. При этом алгоритму строго запрещалось отзеркаливать картинки, поскольку такое действие заставило бы нейросеть путать левую и правую стороны доски, из-за чего точки начали бы съезжать к центру.

После того как программа научилась ровно находить и вырезать игровое поле, начался второй этап — распознавание самих фигур. С помощью первой нейросети фотографии досок были автоматически нарезаны на сотни маленьких квадратиков-клеток. Поскольку правила русских шашек предписывают игру только на темных полях, светлые клетки отбрасывались сразу при помощи математических алгоритмов. Оставшиеся изображения мы разделили на три понятные категории: пустая клетка, белая шашка и черная шашка.

Для распознавания этих мини-картинок вторая нейросеть (CNN) создавалась нами с нуля. Из-за небольшого количества обучающих фотографий существовал серьезный риск вызубривания материала программой, что привело бы к ошибкам на новых фото. Защитный механизм

был встроен нами для предотвращения этой проблемы: во время обучения нейросеть случайным образом «отключала» часть своих внутренних связей. Это заставило алгоритм не зубрить пиксели, а действительно понимать абстрактные формы и цвета шашек.

Строгая настройка цветопередачи стала финальным важным штрихом. Мы добились того, чтобы программа смотрела на цвета через одинаковые «фильтры» как на этапе обучения, так и при реальной работе бота. Это помогло полностью исключить эффект «дальтонизма», возникающий из-за того, что камеры разных смартфонов по-своему искажают цвета на снимках.

9 Описание алгоритма

Весь рабочий процесс деления бота включает четыре последовательных шага: обнаружение доски, распознавание фигур, формирование цифрового двойника и расчет оптимального хода.

9.1 Обнаружение и выравнивание доски

После того как пользователь отправляет в чат фотографию, программа приступает к поиску игрового поля на снимке. Для выполнения этой задачи применяется специальная модель детекции доски. Она обучена находить саму игровую поверхность и четыре крайние точки, которые представляют собой внутренние углы шашечной сетки.

Из-за того, что фотографии часто делаются сбоку или под наклоном, поле в кадре выглядит искаженным. Программа считывает координаты обнаруженных четырех углов и автоматически трансформирует изображение. На выходе формируется идеально плоский квадрат с ракурсом строго сверху.

9.2 Сегментация и анализ клеток

На следующем этапе подключается вторая модель, распознающая клетки, фигуры и их цвета. Так как в русских шашках игровой процесс сосредоточен исключительно на темных полях, светлые участки вычисляются и отбрасываются программой программно, что существенно экономит вычислительное время алгоритма. Оставшиеся изображения темных клеток поочередно передаются в нейросеть. Играя роль «глаз» бота, она внимательно анализирует каждую мини-картинку и определяет текущее состояние: пустая клетка, белая или черная шашка.

9.3 Создание цифровой копии и верификация

На основе ответов второй модели бот собирает в памяти компьютера виртуальную цифровую копию доски в виде матрицы 8x8. Однако системе необходима проверка, чтобы исключить любые ошибки распознавания. Прямо поверх выровненной фотографии программа наносит цветные маркеры (зеленые кружки обозначают белые шашки, красные — черные) и отправляет получившуюся картинку пользователю в чат. Бот переходит непосредственно

к анализу партии сразу после того, как игрок подтверждает правильность считанной позиции.

9.4 Игровой интеллект и рекомендации

Алгоритм просчета лучших ходов запускается сразу после подтверждения позиции. Он строго опирается на правила игры и вычисляет оптимальный шаг по следующим ключевым механизмам:

– проверка на обязательный бой. В первую очередь алгоритм проверяет наличие возможности срубить фигуру соперника. При обнаружении такого варианта стандартные ходы полностью блокируются, так как взятие в шашках является обязательным. Программа способна просчитывать длинные комбинации, когда за один ход уничтожается целая серия вражеских фигур,

– анализ будущих позиций. Для выбора по-настоящему сильного продолжения программа заглядывает далеко за рамки текущего полухода. Она виртуально перемещает шашку, просчитывает вероятные ответы оппонента, моделирует свои следующие действия и таким образом прогнозирует развитие событий на много шагов вперед,

– балльная оценка сценариев. Каждому смоделированному в будущем варианту система присваивает определенные очки. Бот начисляет положительные баллы за уничтожение чужих сил, захват центральных полей и проход шашек в дамки. И наоборот, вариант получает огромный штраф, когда алгоритм видит угрозу потери собственной фигуры на следующем шаге,

– выдача результата. В финале вычислений алгоритмом отбирается несколько ходов с наивысшим баллом. Бот отправляет их пользователю в текстовом виде, подкрепляя комментариями: например, маркирует ход как надежный или предупреждает о рисках.

Дополнительно система осуществляет непрерывный контроль за глобальным состоянием партии. Перед запуском аналитического модуля алгоритм проверяет наличие на доске шашек обоих цветов, и если фигуры одного из игроков полностью отсутствуют, бот автоматически фиксирует окончание игры. После получения рекомендаций пользователь совершает

свой ход на физической доске, затем делает свежую фотографию изменившейся позиции и отправляет её в чат, запуская весь описанный цикл обработки и анализа заново.

10 Реализация

10.1 Файл обучения модели распознавания доски и крайних клеток

В начале программы происходит инициализация легковесной версии нейросети, предназначенной для поиска ключевых точек. Выбор именно такой архитектуры обусловлен необходимостью соблюсти баланс между высокой скоростью работы алгоритма и точностью определения углов доски.

```
def main():
    model = YOLO('yolov8s-pose.pt')
```

Запуск процесса обучения начинается с базовых настроек: указывается путь к набору данных, устанавливается лимит эпох с механизмом ранней остановки и задается увеличенное разрешение входных изображений. Размер пакета данных снижен для экономии видеопамяти, а веса всех слоев открыты для полного обновления.

```
model.train(
    data='D:/DiplomCode/data.yaml',
    epochs=300,
    patience=80,
    imgsz=800,
    batch=8,
    device=0,
    workers=0,
    freeze=0,
```

Следующий блок параметров отвечает за пространственные искажения картинок во время тренировки. Алгоритм искусственно имитирует наклон камеры и легкие повороты доски, но при этом строго блокирует зеркальное отражение кадров, чтобы модель не путала левую и правую стороны игрового поля.

```
    perspective=0.001,
    shear=0.0,
    degrees=5.0,
    scale=0.2,
    translate=0.05,
    fliplr=0.0,
    flipud=0.0,
```

Завершающие настройки регулируют цветовые трансформации, изменяя яркость и насыщенность для защиты от перепадов освещения в реальных условиях. Изображения случайным образом склеиваются в мозаику, а

алгоритм стирания фрагментов отключается во избежание случайного удаления искомых угловых точек на фото.

```
        hsv_h=0.015,
        hsv_s=0.2,
        hsv_v=0.2,
        mosaic=0.4,
        erasing=0.0,
        name='checkers_pose'
```

В конце файла располагается стандартная конструкция языка, определяющая точку входа в программу. Она гарантирует, что процесс обучения будет корректно запущен только при непосредственном старте данного скрипта.

```
if __name__ == '__main__':
  main()
```

10.2 Файл для заготовки датасета для обучения модели распознавания клеток, фигур и их цветов

Функция извлечения игрового поля принимает на вход матрицу изображения и передает её в нейросеть. В случае успешного нахождения ровно четырех угловых точек, их координаты извлекаются из тензора и преобразуются в массив чисел с плавающей точкой для дальнейших геометрических вычислений.

```
    def detect_and_warp_board(image: np.ndarray):
        results = pose_model(image, conf=0.5, verbose=False)

        if results[0].keypoints is None or
len(results[0].keypoints) == 0:
            raise ValueError()

        kp = results[0].keypoints.xy.cpu().numpy()[0]
        pts = kp.astype(np.float32)

        if pts.shape[0] != 4:
            raise ValueError()
```

Полученный массив точек программно сортируется по суммам и разностям координат для строгого определения пространственного расположения каждого угла. После этого вычисляется матрица гомографии, которая проецирует найденный четырехугольник в идеальный ортогональный квадрат размером 800x800 пикселей.

```
      rect = np.zeros((4, 2), dtype=np.float32)
```

```
        s = pts.sum(axis=1)
        rect[0] = pts[np.argmin(s)]
        rect[2] = pts[np.argmax(s)]

        diff = np.diff(pts, axis=1)
        rect[1] = pts[np.argmin(diff)]
        rect[3] = pts[np.argmax(diff)]

        dst = np.array([
            [0, 0],
            [800, 0],
            [800, 800],
            [0, 800]
        ], dtype=np.float32)

        M = cv2.getPerspectiveTransform(rect, dst)
        warped = cv2.warpPerspective(image, M, (800, 800))

        return warped
```

Следующая функция отвечает за фрагментацию выровненной доски на 64 равные клетки размером 100x100 пикселей. Алгоритм проходит по матрице изображения двойным циклом, вырезает соответствующие участки и сохраняет их на диск, добавляя в название оригинального файла индексы текущей строки и столбца.

```
    def slice_and_save_cells(warped_img: np.ndarray,
original_name: str):
        os.makedirs(OUTPUT_DIR, exist_ok=True)
        h, w = 100, 100
        saved_count = 0

        base_name = os.path.splitext(original_name)[0]

        for row in range(8):
            for col in range(8):
                y = row * h
                x = col * w
                cell_img = warped_img[y:y + h, x:x + w]

                filename = os.path.join(OUTPUT_DIR,
f"{base_name}_r{row}_c{col}.jpg")
                cv2.imwrite(filename, cell_img)
                saved_count += 1

        return saved_count
```

Главная функция скрипта формирует список всех поддерживаемых графических файлов в целевой директории и запускает цикл их последовательной обработки. Для каждой фотографии вызываются методы

выравнивания и нарезки, а конструкция перехвата исключений защищает программу от аварийного завершения при обработке нетипичных ракурсов.

```
    def main():
        search_pattern = os.path.join(INPUT_DIR, "*.*")
        image_files = [f for f in glob.glob(search_pattern) if
f.lower().endswith(('.png', '.jpg', '.jpeg'))]

        if not image_files:
            return

        success_count = 0

        for img_path in image_files:
            filename = os.path.basename(img_path)
            img = cv2.imread(img_path)

            if img is None:
                continue

            try:
                warped_img = detect_and_warp_board(img)
                slice_and_save_cells(warped_img, filename)
                success_count += 1
            except ValueError:
                pass
            except Exception:
                pass

    if __name__ == '__main__':
        main()
```

10.3 Файл обучения модели распознавания клеток, фигур и их цветов

Архитектура нейронной сети построена на базе трех сверточных блоков, каждый из которых включает пакетную нормализацию для ускорения сходимости и пулинг для снижения размерности. Итоговая классификация выполняется полносвязными слоями, где применяется механизм исключения нейронов (Dropout) для предотвращения переобучения модели.

```
    class CheckerCellCNN(nn.Module):
        def __init__(self, num_classes=3):
            super().__init__()
            self.features = nn.Sequential(
                nn.Conv2d(3, 32, kernel_size=3, padding=1),
                nn.BatchNorm2d(32),
                nn.ReLU(),
                nn.MaxPool2d(2, 2),
```

```
            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(2, 2),

            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.MaxPool2d(2, 2)
        )

        self.classifier = nn.Sequential(
            nn.Dropout(0.5),
            nn.Linear(128 * 12 * 12, 128),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(128, num_classes)
        )

    def forward(self, x):
        x = self.features(x)
        x = x.view(x.size(0), -1)
        x = self.classifier(x)
        return x
```

Для загрузки обучающей выборки реализован класс датасета, считывающий изображения из структурированных директорий. При чтении файлов происходит автоматическое извлечение меток классов из названий папок и обязательная конвертация изображений в единое цветовое пространство RGB для исключения цветовых артефактов.

```
class CheckerCellDataset(Dataset):
    def __init__(self, root_dir, transform=None):
        self.classes = ['empty_dark', 'white_man',
'black_man']
        self.class_to_idx = {cls: idx for idx, cls in
enumerate(self.classes)}
        self.transform = transform
        self.images = []
        self.labels = []

        for class_name in self.classes:
            class_dir = os.path.join(root_dir, class_name)
            if not os.path.isdir(class_dir):
                continue
            for fname in os.listdir(class_dir):
                if fname.lower().endswith(('.png', '.jpg',
'.jpeg')):

self.images.append(os.path.join(class_dir, fname))

self.labels.append(self.class_to_idx[class_name])

    def __len__(self):
```

```
        return len(self.images)

    def __getitem__(self, idx):
        img_path = self.images[idx]
        image = Image.open(img_path).convert('RGB')
        label = self.labels[idx]
        if self.transform:
            image = self.transform(image)
        return image, label
```

Подготовка данных включает блок динамической аугментации для обучающей выборки, применяющий случайные повороты, отражения и изменения цветового тона. Это искусственно расширяет набор данных, при этом для валидационной выборки используется только приведение к базовому разрешению и цветовая нормализация.

```
normalize = transforms.Normalize(mean=[0.5, 0.5, 0.5],
std=[0.5, 0.5, 0.5])

train_transform = transforms.Compose([
    transforms.Resize((100, 100)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomVerticalFlip(),
    transforms.RandomRotation(degrees=15),
    transforms.RandomAffine(degrees=0, translate=(0.1, 0.1),
scale=(0.9, 1.1)),
    transforms.ColorJitter(brightness=0.2, contrast=0.2,
saturation=0.1, hue=0.05),
    transforms.ToTensor(),
    normalize
])

val_transform = transforms.Compose([
    transforms.Resize((100, 100)),
    transforms.ToTensor(),
    normalize
])
```

Сформированные датасеты оборачиваются в специализированные загрузчики данных для оптимизации процесса ввода-вывода. Они автоматически группируют изображения в пакеты заданного размера и перемешивают обучающую выборку перед каждой итерацией для повышения качества генерализации модели.

```
train_dataset =
CheckerCellDataset('dataset_for_cell_detection/train',
transform=train_transform)
```

```
    val_dataset =
CheckerCellDataset('dataset_for_cell_detection/val',
transform=val_transform)

    train_loader = DataLoader(train_dataset, batch_size=32,
shuffle=True, num_workers=0)
    val_loader = DataLoader(val_dataset, batch_size=32,
shuffle=False, num_workers=0)
```

Цикл обучения использует алгоритм оптимизации Adam и функцию кросс-энтропийных потерь, вычисляя градиенты на каждом шаге. После каждой эпохи запускается процесс валидации, по результатам которого алгоритм сохраняет веса модели строго в том случае, если зафиксировано улучшение метрики общей точности.

```
    device = torch.device('cuda' if torch.cuda.is_available()
else 'cpu')
    model = CheckerCellCNN(num_classes=3).to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=0.001)

    num_epochs = 30
    best_val_acc = 0.0

    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0
        for images, labels in train_loader:
            images, labels = images.to(device),
labels.to(device)
            optimizer.zero_grad()
            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()

        model.eval()
        val_loss = 0.0
        correct = 0
        total = 0
        with torch.no_grad():
            for images, labels in val_loader:
                images, labels = images.to(device),
labels.to(device)
                outputs = model(images)

                v_loss = criterion(outputs, labels)
                val_loss += v_loss.item()

                _, predicted = torch.max(outputs, 1)
```

```
                total += labels.size(0)
                correct += (predicted == labels).sum().item()

        avg_train_loss = running_loss / len(train_loader)
        avg_val_loss = val_loss / len(val_loader)
        val_accuracy = 100 * correct / total if total > 0 else
0.0

        if val_accuracy >= best_val_acc:
            best_val_acc = val_accuracy
            torch.save(model.state_dict(),
'checker_model_new.pth')
```

10.4 Файл обучения модели распознавания клеток, фигур и их цветов

Эта функция формирует виртуальную матрицу игрового поля на основе выровненного изображения. Она проходит исключительно по темным клеткам, вырезает их и передает в классификатор, а также отрисовывает цветные маркеры поверх исходной картинки для визуальной проверки пользователем.

```
    def get_board_state(warped_img, top_left_coord):
        file_start = ord(top_left_coord[0]) - ord('a')
        rank_start = int(top_left_coord[1]) - 1

        if top_left_coord == 'a8':
            file_step, rank_step = 1, -1
        else:
            file_step, rank_step = -1, 1

        board = np.zeros((8, 8), dtype=int)
        overlay = warped_img.copy()

        for row in range(8):
            for col in range(8):
                real_file = file_start + col * file_step
                real_rank = rank_start + row * rank_step

                if (real_file + real_rank) % 2 != 0:
                    continue

                cell = extract_cell(warped_img, row, col)
                cls = classify_cell(cell)
                notation = f"{chr(ord('a') +
real_file)}{real_rank + 1}"

                if cls == 1:
                    board[row][col] = 1
```

```
                    color = (0, 255, 0)
                elif cls == 2:
                    board[row][col] = -1
                    color = (0, 0, 255)
                else:
                    board[row][col] = 0
                    color = (128, 128, 128)

                cx = col * 100 + 50
                cy = row * 100 + 50

                cv2.circle(overlay, (cx, cy), 30, color, 2)
                cv2.putText(overlay, notation, (cx - 15, cy -
40),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.5,
color, 2)

        return board, overlay
```

Данный метод реализует рекурсивный поиск обязательных взятий шашек противника. Алгоритм проверяет доступные диагонали и, в случае нахождения цели, виртуально совершает прыжок, после чего вызывает сам себя для поиска возможных серийных многоступенчатых взятий.

```
    def find_captures(board, player, r, c, current_path=None,
captured_pieces=None):
        if current_path is None: current_path = [(r, c)]
        if captured_pieces is None: captured_pieces = []

        found_further_capture = False
        results = []

        for dr, dc in [(-1, -1), (-1, 1), (1, -1), (1, 1)]:
            nr, nc = r + dr, c + dc
            nnr, nnc = r + 2 * dr, c + 2 * dc

            if 0 <= nnr < 8 and 0 <= nnc < 8:
                if board[nr][nc] == -player and (nr, nc) not in
captured_pieces:
                    if board[nnr][nnc] == 0 or (nnr, nnc) ==
current_path[0]:
                        found_further_capture = True
                        temp_board = board.copy()
                        temp_board[r][c] = 0
                        temp_board[nr][nc] = 0
                        temp_board[nnr][nnc] = player

                        res = find_captures(temp_board, player,
nnr, nnc,
                                            current_path +
[(nnr, nnc)],
```

```
                                                captured_pieces +
[(nr, nc)])
                            results.extend(res)

        if not found_further_capture and len(current_path) > 1:
            results.append({'path': current_path, 'is_capture':
True, 'captured_pieces': captured_pieces})

        return results
```

Ядро интеллектуального движка бота, просчитывающее развитие партии на несколько шагов вперед. Функция поочередно симулирует ходы за игрока и за противника, применяя альфа-бета отсечение для быстрого отбрасывания заведомо невыгодных ветвей развития событий.

```
    def minimax(board, depth, alpha, beta, maximizing_player_id,
current_player, top_left):
        if depth == 0:
            return evaluate_board(board, maximizing_player_id,
top_left)

        moves = get_moves(board, current_player, top_left)
        if not moves:
            return -10000 if current_player ==
maximizing_player_id else 10000

        if current_player == maximizing_player_id:
            max_eval = -float('inf')
            for move in moves:
                new_board = apply_move(board, move,
current_player)
                ev = minimax(new_board, depth - 1, alpha, beta,
maximizing_player_id, -current_player, top_left)
                max_eval = max(max_eval, ev)
                alpha = max(alpha, ev)
                if beta <= alpha:
                    break
            return max_eval
        else:
            min_eval = float('inf')
            for move in moves:
                new_board = apply_move(board, move,
current_player)
                ev = minimax(new_board, depth - 1, alpha, beta,
maximizing_player_id, -current_player, top_left)
                min_eval = min(min_eval, ev)
                beta = min(beta, ev)
                if beta <= alpha:
                    break
            return min_eval
```

Модуль итоговой оценки собирает все легальные ходы и ранжирует их на основе прогноза алгоритма Minimax. Метод начисляет бонус за множественные взятия и формирует текстовое предупреждение, если выбранный путь позволит противнику забрать фигуры на следующем ходу.

```
    def get_best_moves(board, player, top_left):
        moves = get_moves(board, player, top_left)
        if not moves: return []

        scored_moves = []
        for move in moves:
            new_board = apply_move(board, move, player)

            opp_moves = get_moves(new_board, -player, top_left)
            max_opp_captures = 0
            if opp_moves:
                max_opp_captures =
max(len(m.get('captured_pieces', [])) for m in opp_moves)

            future_score = minimax(new_board, 4, -float('inf'),
float('inf'), player, -player, top_left)

            num_captured = len(move.get('captured_pieces', []))
            immediate_bonus = num_captured * 100

            score = immediate_bonus + future_score +
random.uniform(0, 0.1)
            scored_moves.append({
                'move': move,
                'score': score,
                'num_captured': num_captured,
                'max_opp_captures': max_opp_captures
            })

        scored_moves.sort(key=lambda x: x['score'],
reverse=True)
        top_moves = scored_moves[:4]

        results = []
        for sm in top_moves:
            path_notations = [index_to_notation_real(r, c,
top_left) for r, c in sm['move']['path']]
            move_str = " ➡ ".join(path_notations)

            num_cap = sm['num_captured']
            if sm['move']['is_capture']:
                if num_cap >= 3:
                    reason = f"🔥 Блестящая комбинация!
Множественное взятие ({num_cap} шашки)."
                elif num_cap == 2:
```

```
                    reason = f"💥 Отличный двойной удар! За один
ход срубается {num_cap} шашки."
                else:
                    reason = f"⚔️ Обязательный бой. Забираем 1
шашку противника."
            else:
                score = sm['score']
                if score > 5:
                    reason = "🎯 Сильный позиционный ход. Берет
центр под контроль."
                elif score < -5:
                    reason = "⚠️ Единственный легальный ход, но
нужно быть осторожным."
                else:
                    reason = "🚶‍♂️ Надежный развивающий ход без
риска."

            if sm['max_opp_captures'] > 0:
                reason += f"\n   🚨 *Внимание: После этого хода
вы рискуете потерять до {sm['max_opp_captures']} фигур за раз.*"
            else:
                reason += f"\n   🛡️ *Отличные новости:* Ход
кажется безопасным."

            results.append({'text': move_str, 'reason': reason})

        return results
```

11 Разработка макета пользовательского интерфейса

Макет интерфейса начального экрана представлен на рисунке 4.

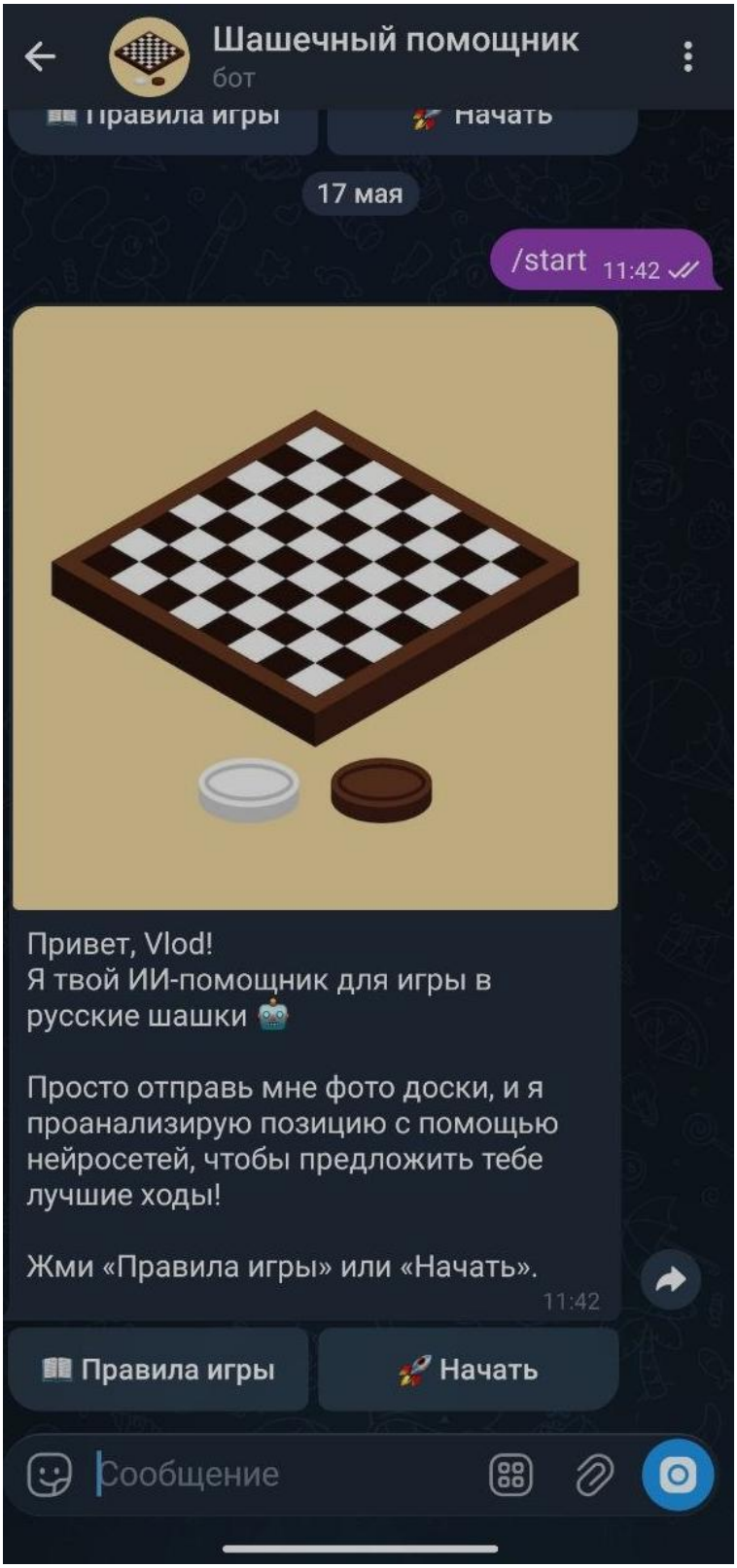


Рисунок 4 – Интерфейс начального экрана

При нажатии на кнопку «Правила игры» мы получим перечень правил игры и несколько советов для новичков. Интерфейс представлен на рисунке 5.

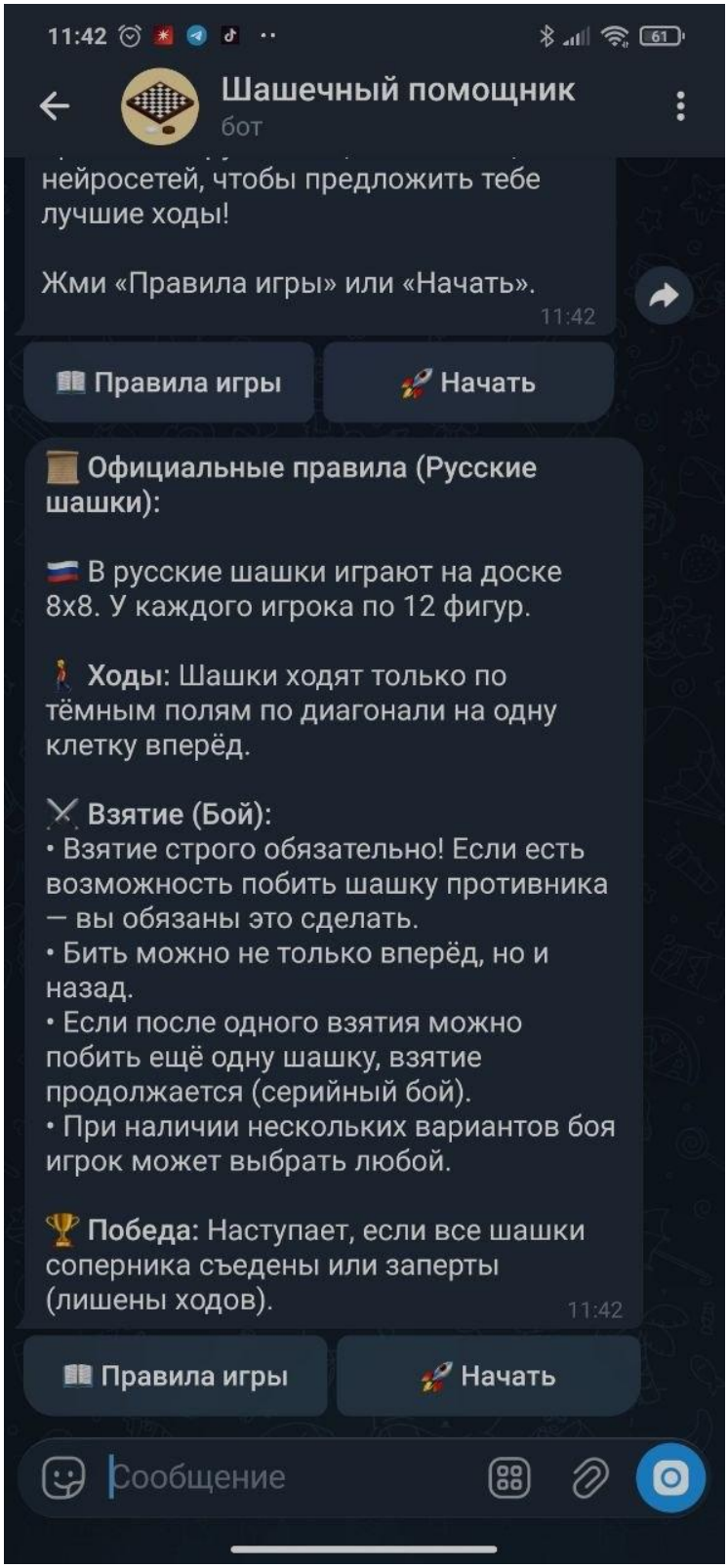


Рисунок 5 – Вкладка «Правила игры»

При нажатии на кнопку «Начать» мы получим сообщение о том, что бот готов получить фотографию с текущим расположением фигур на игровой

доске. После получения фотографии потребуется ещё указать координату верхней левой клетки, чтобы относительно неё простроить виртуальные координаты остальных клеток. Интерфейс представлен на рисунке 6.

Рисунок 6 – Вкладка «Начать»

После указания координаты бот приступит к считыванию изображения и распознаванию. Если доска на изображении не будет найдена или фигуры

будут распознаны неверно, то бот попросит отправить другое изображение. В случае успеха мы получим сообщение с результатом считывания доски. После подтверждения корректности распознавания клеток и фигур, необходимо указать цвет фигур, для которых требуется просчитать лучший ход. Интерфейс представлен на рисунке 7.

Рисунок 7 – Отправка изображения и его корректное считывание

После выбора цвета бот отправит нам сообщение с несколькими возможными ходами, где будут взвешены риски. Самым лучших ходом будет считать тот, что выше всех. Интерфейс представлен на рисунке 8.

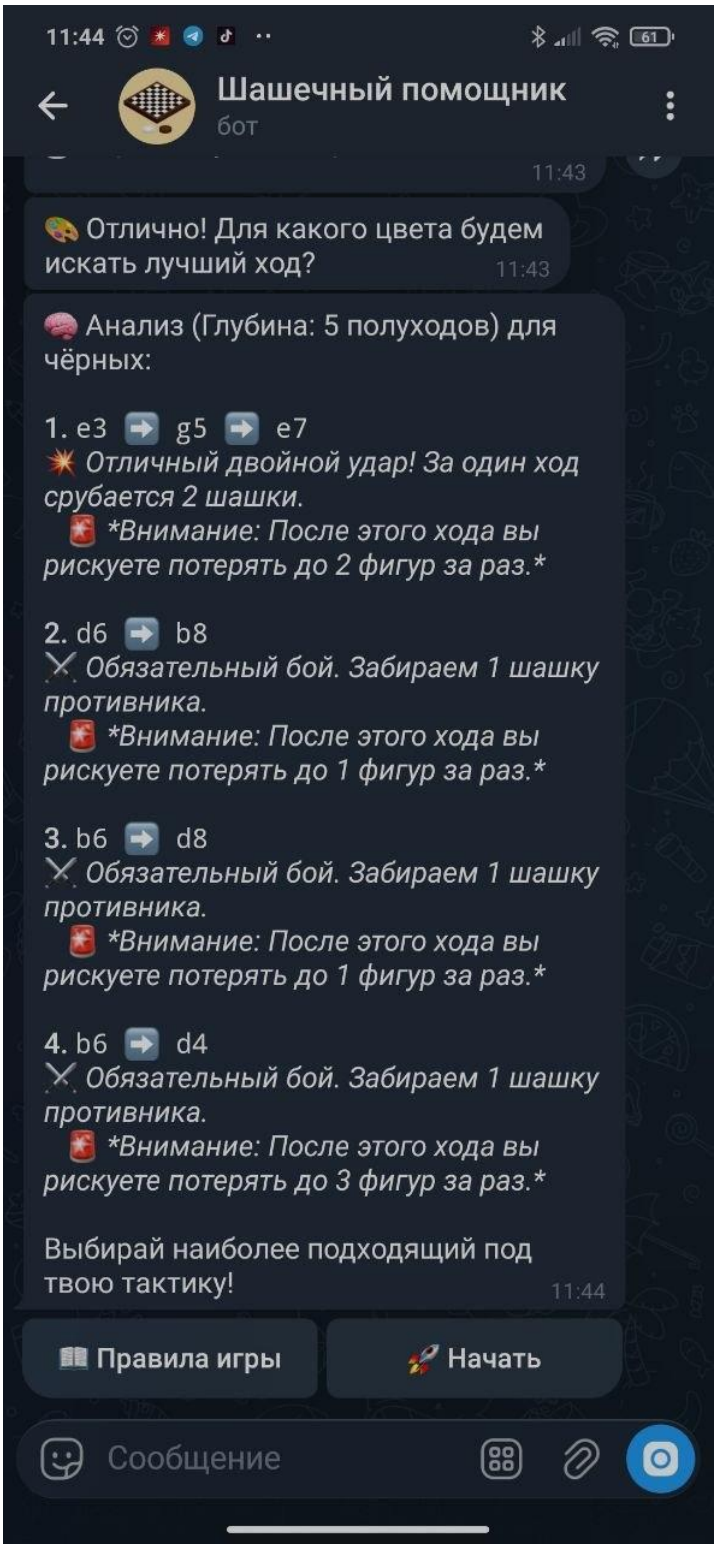


Рисунок 8 – Вкладка «Рекомендации», для пользователя без авторизации

12 Тестирование основного функционала

Для проведения тестирования потребуется подготовить несколько фотографий, которые не участвовали в процессе обучения или валидации. Основная цель тестирования насколько корректно отрабатывает алгоритм распознавания доски, клеток, фигур и их цветов, а также правильность итоговых ответов.

Отправляем первую фотографию для проверки, указываем координату крайней левой верхней клетки и получаем предварительно распознанное поле с фигурами и клетками. Всё перечисленное изображено на рисунке 9.

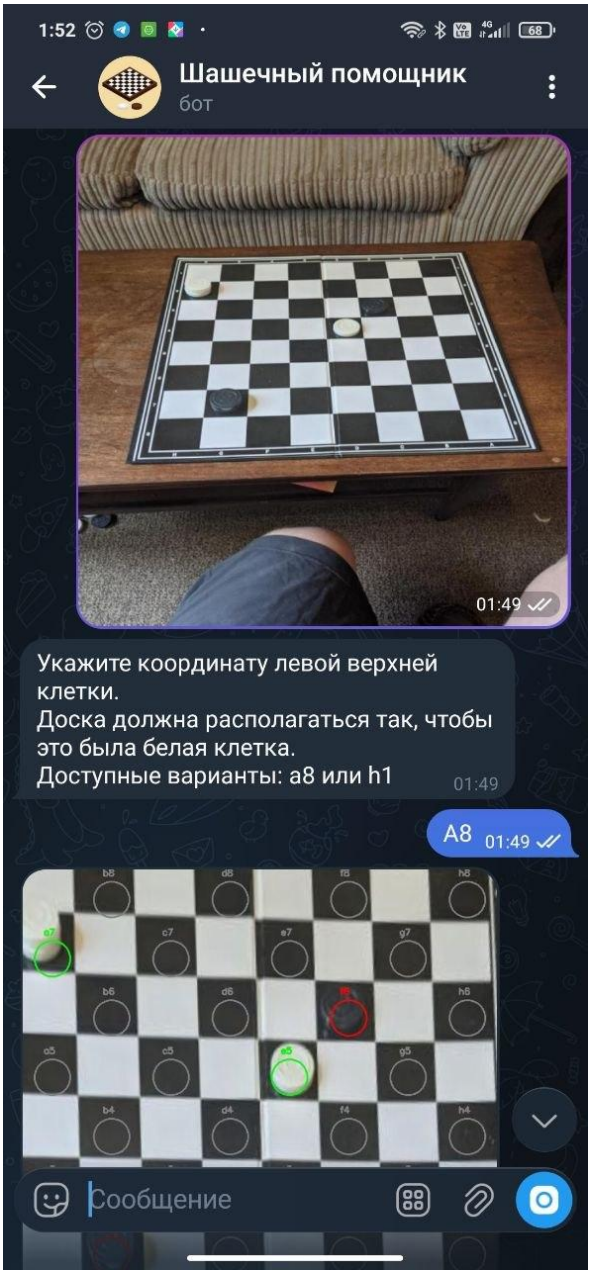


Рисунок 9 - Начало работы с 1 фотографией

Убедившись, что все клетки и фигуры распознаны корректно, подтверждаем успех. Далее выбираем цвет фигур, для которых требуется просчитать лучшие ходы. В конце получаем результат, который мы и ожидали, показанный на рисунке 10. Так как на поле белая и чёрная шашки расположены рядом друг с другом, то по правилам одна из них должна побить другую. Другие возможные ходы по этой причине невозможны. Координаты клеток виртуальной доски также совпадают с координатами реальной.

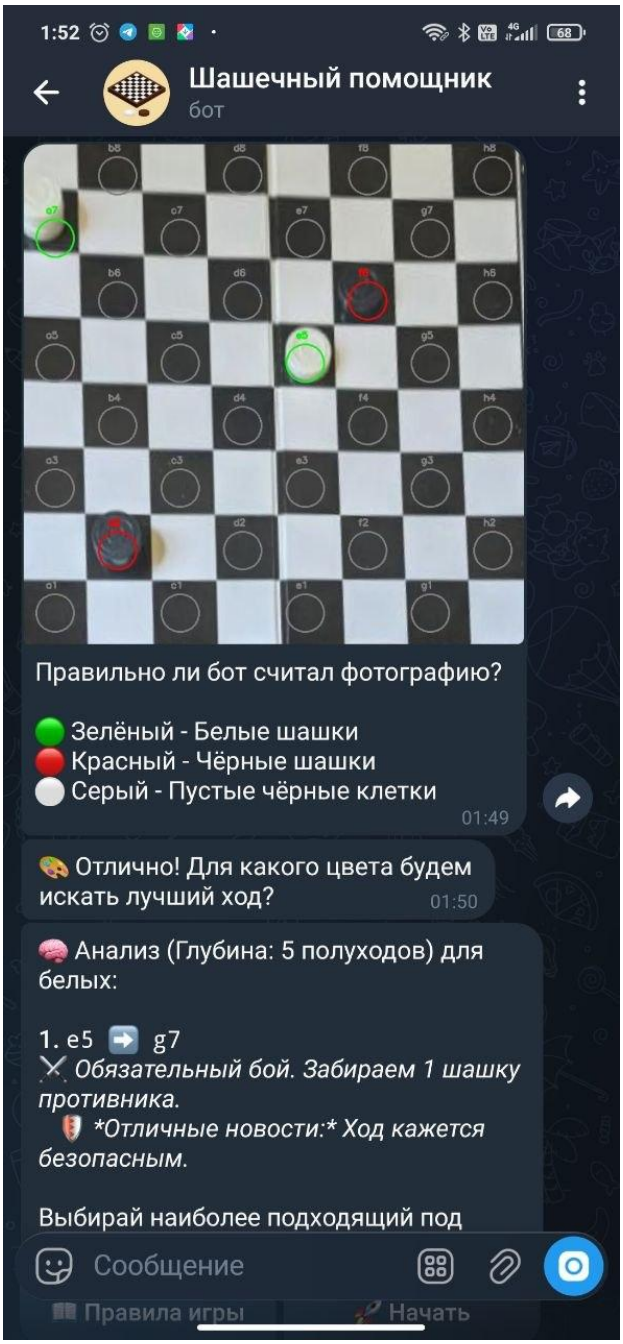


Рисунок 10 - Выбор цвета и получение перечня лучших ходов

Следующим шагом стоит проверить корректность расположения координат доски в случае, если крайней левой верхней клеткой будет H1 и если задать некорректную координату, Результат этого этапа можно видеть на рисунках 11 и 12.

Рисунок 11 – Начало работы с фотографией 2 и попытка указать некорректную координату

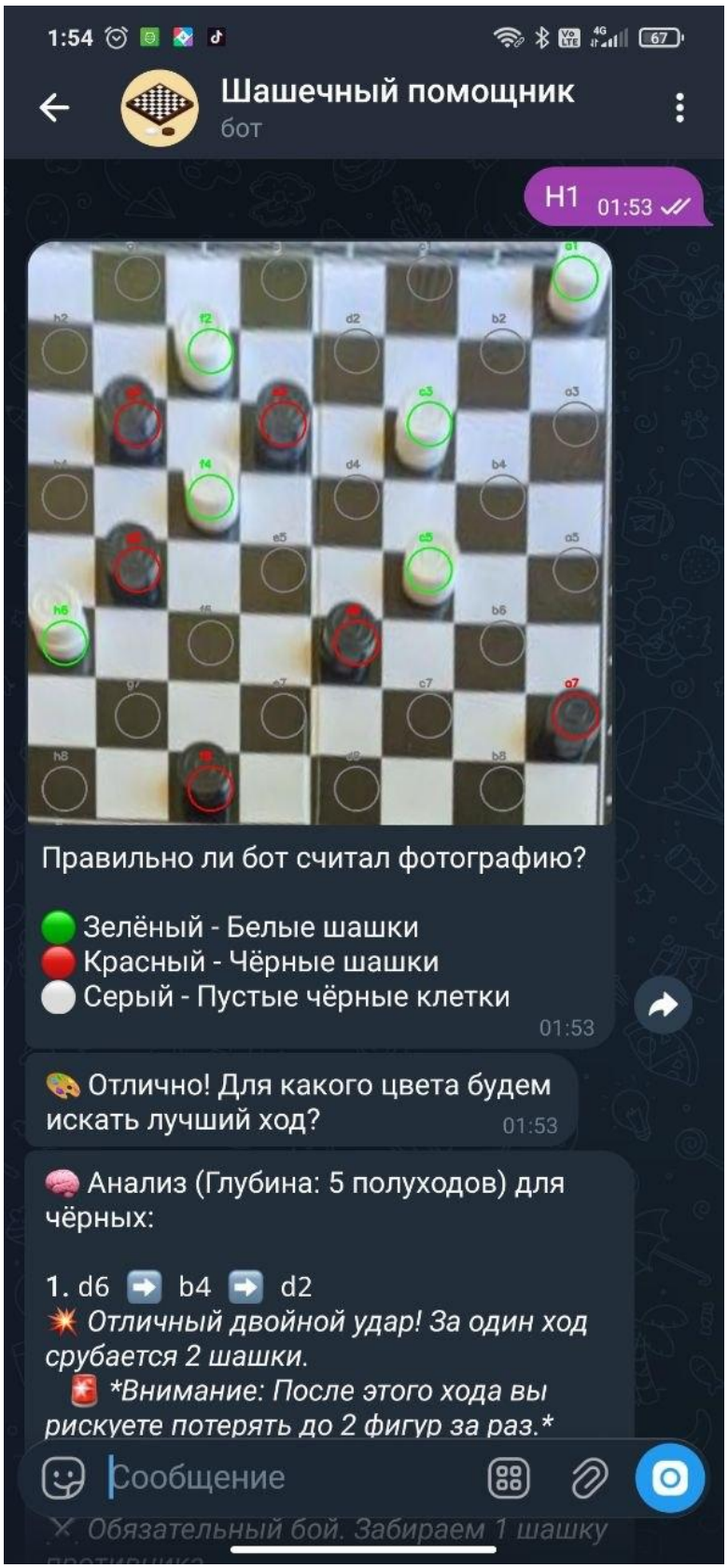


Рисунок 12 - Проверка отрисовки координат клеток относительно H1.

Как можно заметить координаты по-прежнему совпадают с ожидаемыми, а значит подтверждаем успех и выбираем нужный нам цвет фигур. В результате получаем перечень лучших ходов, который показан на рисунке 13. Здесь можно заметить, что ходы расположены в порядке от лучшего к худшему и описаны риски каждого из них.

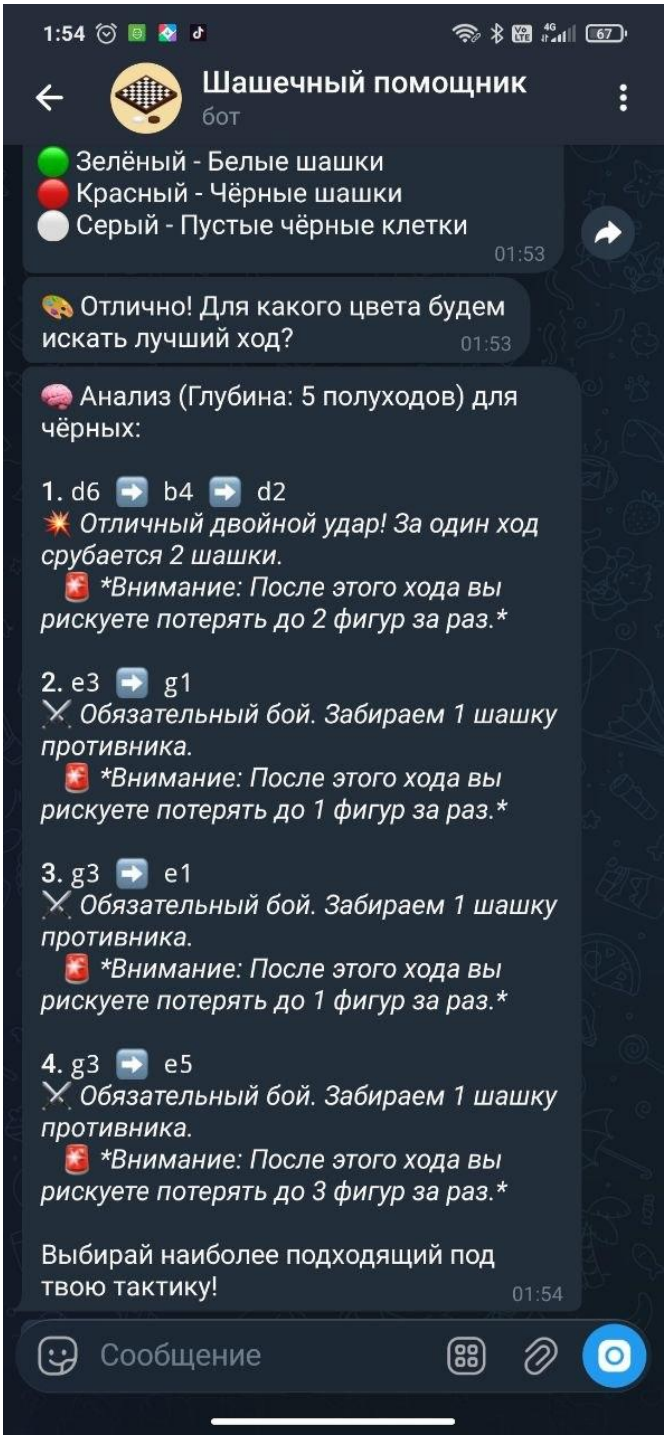


Рисунок 13 - Получение перечня лучших ходов

Теперь проверим сценарий, когда на доске белые и чёрные фигуры находятся далеко друг от друга и ситуация обязательного боя отсутствует. Загружаем фотографию, выбираем координату крайней клетки. Процесс можно видеть на рисунке 14.

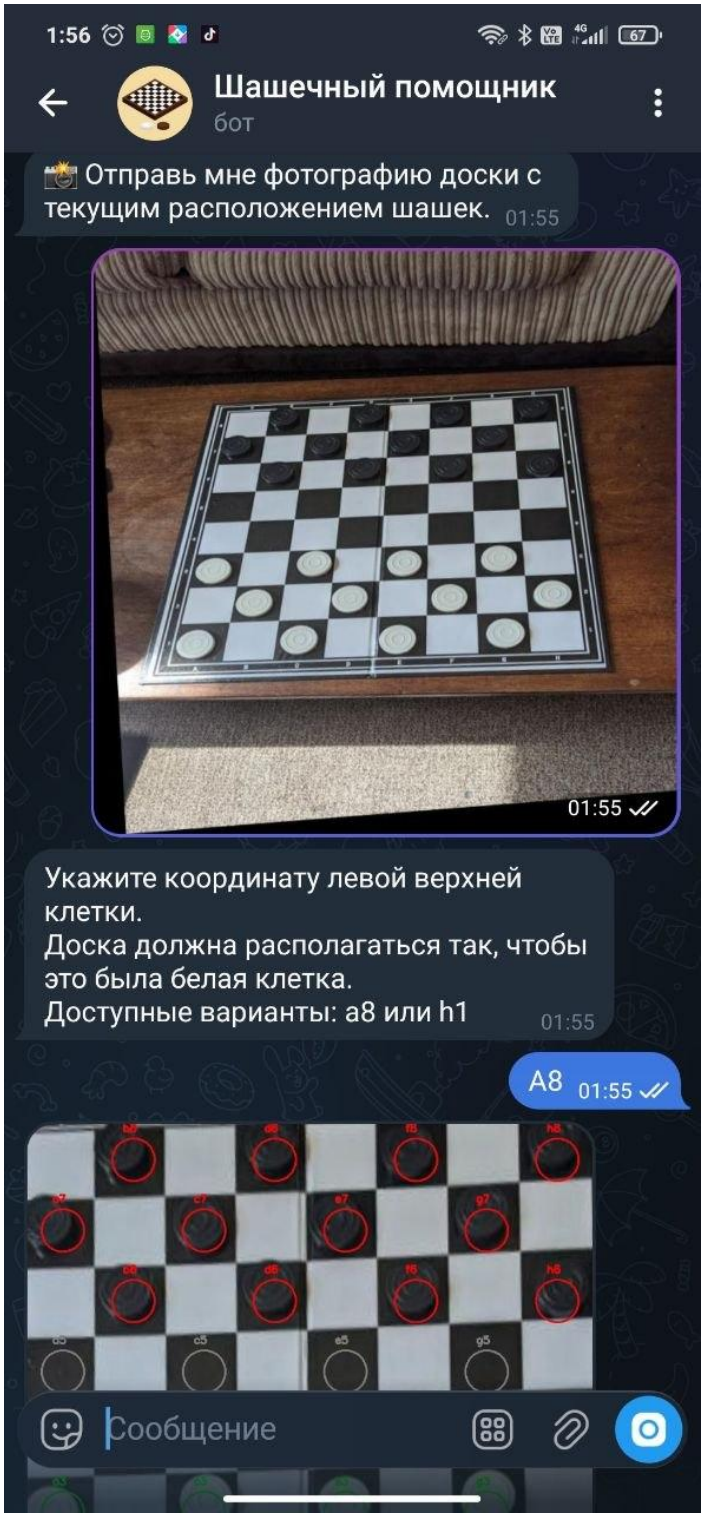


Рисунок 14 - Начало работы с фотографией 3

Проверяем правильность распознанных элементов и подтверждаем это. Выбираем белые фигуры для помощи. Процесс можно видеть на рисунке 15.

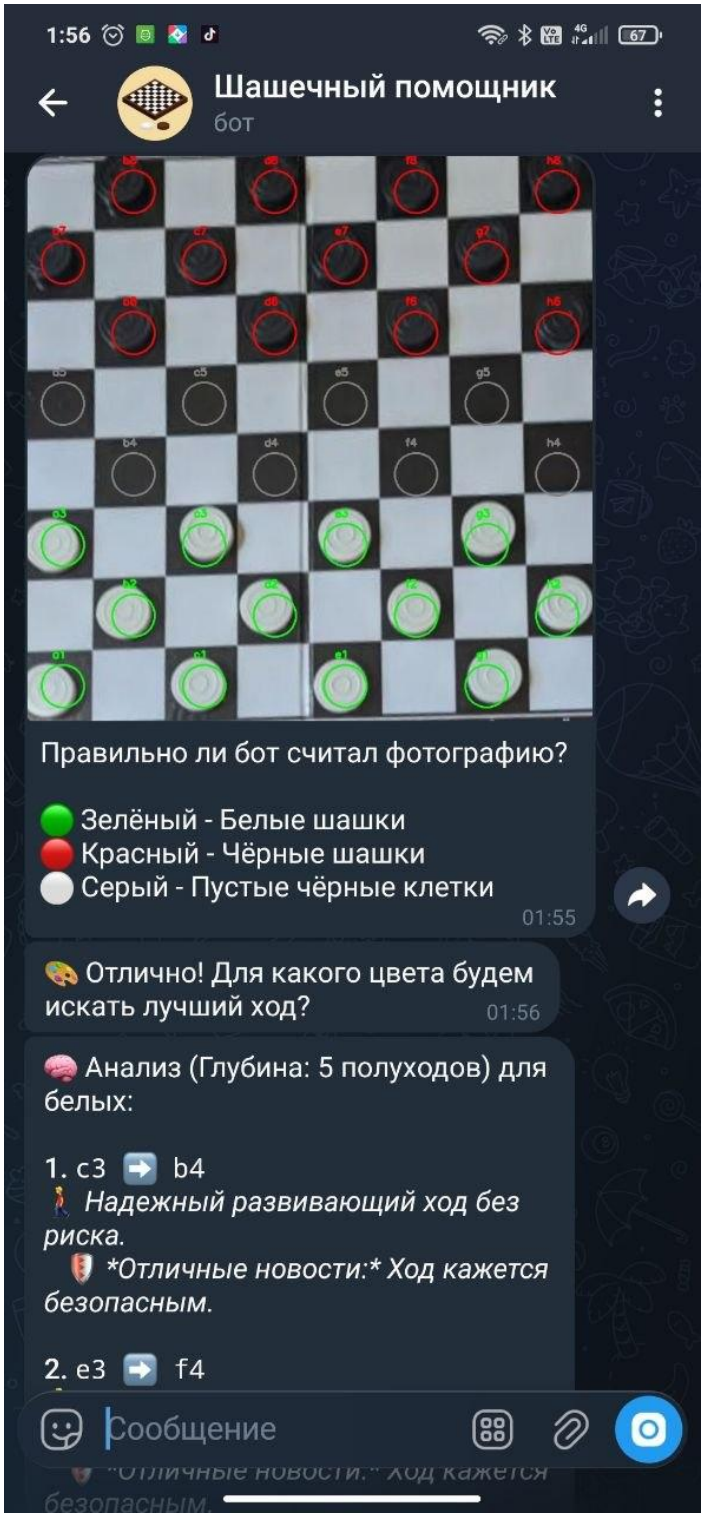


Рисунок 15 - Получение результата распознавания доски и фигур

И наконец получаем перечень лучших ходов для белой стороны. Результат можно видеть на рисунке 16.

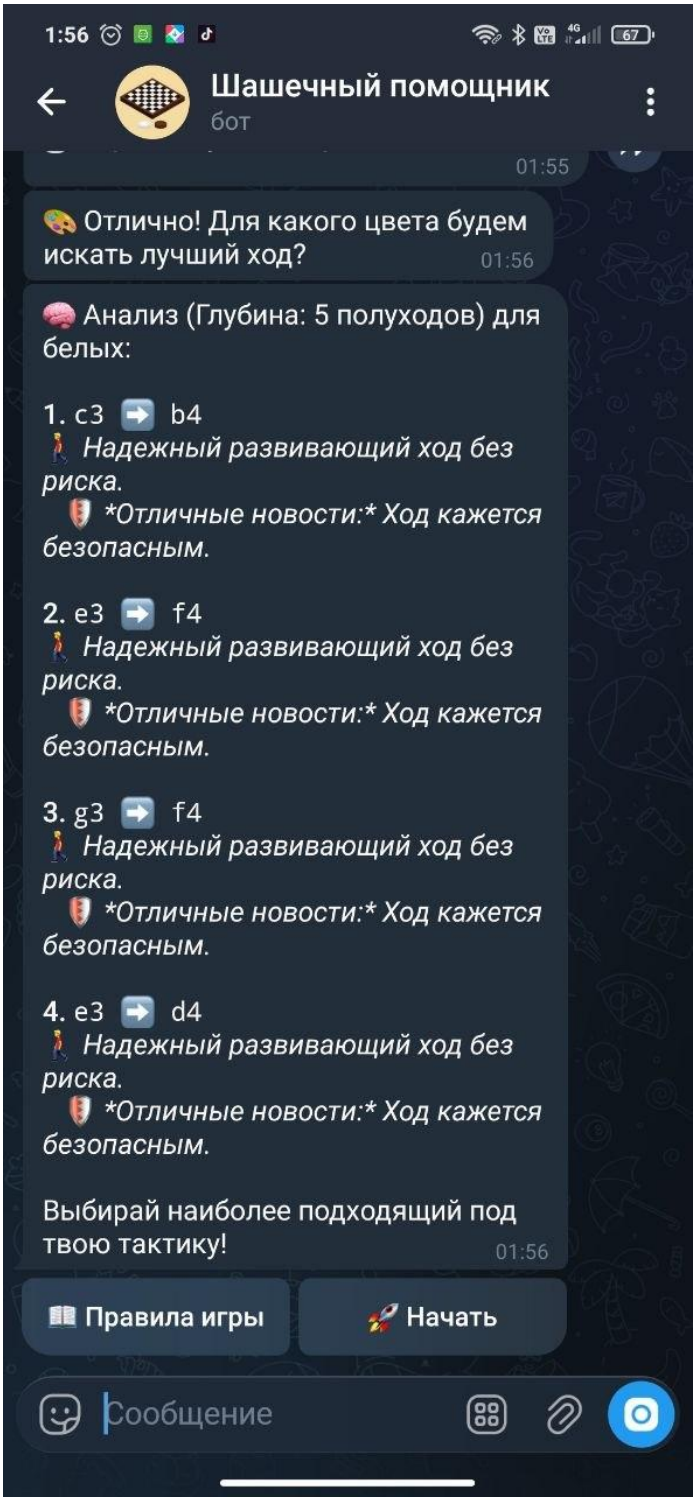


Рисунок 16 – Получение перечня лучших ходов

ЗАКЛЮЧЕНИЕ

Во время исследования и анализа предметной области были рассмотрены различные варианты сбора и обработки входных графических данных, включая ручную разметку снимков шашечных досок и программную синтетическую генерацию позиций через специализированные алгоритмы. Приведен перечень оптимальных методов аугментации данных, необходимых для обучения устойчивых моделей компьютерного зрения. Также подробно разобраны способы предварительной подготовки и нормализации изображений перед их передачей в нейросети, что позволило получить сбалансированные обучающие выборки клеток. В ходе выполнения проекта проведены эксперименты по распознаванию геометрии доски и классификации игровых ситуаций в условиях вариативного освещения.

Нами были подробно рассмотрены классические алгоритмы машинного обучения и подходы компьютерного зрения, такие как пороговая бинаризация, сопоставление с эталонами, метод опорных векторов (SVM) и ансамблевые модели на базе случайного леса или градиентного бустинга. Для решения задач магистерской диссертации проанализированы их ключевые ограничения при обработке реальных пользовательских снимков, что послужило обоснованием выбора глубоких свёрточных нейронных сетей (CNN, архитектура YOLO) в качестве основного инструмента детекции и распознавания.

В рамках данной выпускной квалификационной работы было исследовано влияние архитектурных решений на общую производительность системы и точность критериев финальной оценки позиций. Проектирование велось с применением языка программирования Python 3.13 и фреймворка PyTorch. В основу программного комплекса легли асинхронные модули библиотеки aiogram для развертывания многопоточного интерфейса, инструменты OpenCV для перспективной коррекции и библиотеки глубокого обучения PyTorch. Нами также была внедрена сетевая архитектура Long-

polling, обеспечивающая стабильный обмен данными между серверами Telegram и вычислительным ядром.

Кроме того, нами были выделены четкие технические и функциональные критерии автоматического анализа настольных игр по фотографиям. Была детально проверена применимость игрового шашечного движка для генерации текстовых подсказок в реальном времени.

Полученные результаты обладают высокой практической и научной ценностью, которая заключается в перспективе дальнейших исследований в области автоматизации распознавания сложных пространственных объектов на неоднородном фоне. Развитие предложенного нами подхода позволит улучшить качество комплексных интеллектуальных систем помощи пользователям, ведь возможности чистой оптимизации гиперпараметров ML-моделей конечны, а гибридная интеграция компьютерного зрения с логическими движками показывает высокую эффективность.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Савельев Д. А. Применение сверточных нейронных сетей в задачах сегментации и распознавания объектов на изображениях // Инновационный дискурс развития современной науки и технологий: Сборник статей V Международной научно-практической конференции. Петрозаводск, 2023. С. 112–118.

2. Костин А. В. Модификация архитектур сверточных нейросетей для локализации объектов на неоднородном фоне // Сборник работ студентов-магистрантов факультета ВМК МГУ имени М. В. Ломоносова. Москва, 2024. С. 142–155.

3. Морозов Р. С. Синтетическая генерация графических датасетов для обучения моделей компьютерного зрения: методы и критерии // Вызовы цифровой экономики: технологический суверенитет и экономическая безопасность: Сборник статей VII Всероссийской научно-практической конференции с международным участием. – 2024. – С. 210–215.

4. Документация по аугментации данных и разметке изображений в компьютерном зрении. – URL: https://roboflow.com/ formats/yolov8-pose-txt (дата обращения: 12.02.2026).

5. Федоров Н. С. Перспективная коррекция и сегментация игровых полей в прикладных задачах распознавания образов // Инженерный вестник Дона. 2023. № 8. С. 89–97.

6. Свёрточные нейронные сети и архитектура YOLOv8 для поиска ключевых точек: практическое руководство. – URL: https://habr.com/ru/articles/yolov8-pose-estimation/ (дата обращения: 03.03.2026).

7. Новикова Д. В. Сравнительный анализ эффективности методов кодирования признаков в задачах бинарной и многоклассовой классификации // Заметки по информатике и математике: Сборник научных статей. Ярославль, 2023. С. 45–53.

8. Дьяконов А. Г. Анализ изображений и методы выделения локальных признаков в компьютерном зрении // Прикладная математика и информатика:

Труды факультета Вычислительной математики и кибернетики МГУ имени М. В. Ломоносова. 2018. № 52. С. 77–94.

9. Меньшиков Я. С. Архитектурные подходы при организации взаимодействия Telegram-ботов со сторонними вычислительными модулями // Universum: технические науки. 2024. № 4-2 (115). С. 18–22.

10. Румянцев А. П., Васильев К. М. Исследование алгоритмов альфа-бета отсечения и минимакса в классических настольных играх // Теория дискретных систем и управление. 2021. № 3. С. 41–50.

11. Ultralytics YOLOv8 Docs: Real-time Object Detection and Pose Estimation. – URL: https://docs.ultralytics.com/modes/train/ (дата обращения: 19.04.2026).

12. Райан Митчелл. Современный скрапинг веб-сайтов с помощью Python. 2-е межд. издание. – М.: ДМК Пресс, 2021. – 268 с.

13. Боровская Е. В. Основы искусственного интеллекта / Е. В. Боровская, Н. А. Давыдова. – 4-е изд. – М.: Лаборатория знаний, 2020. – 130 с.

14. Джоши Патрик. Искусственный интеллект с примерами на Python / Патрик Джоши. – СПб.: ООО «Диалектика», 2019. – 448 с.

15. Шолле Ф. Глубокое обучение на Python. – СПб.: Питер, 2018. – 400 с.

16. Ведяхин А. Сильный искусственный интеллект: на подступах к сверхразуму / А. Ведяхин [и др.]. – М.: Интеллектуальная Литература, 2021. – 323 с.

17. Redmon J., Farhadi A. YOLOv3: An Incremental Improvement // Tech Report, arXiv:1804.02767. Computer Vision and Pattern Recognition. 2018.

18. Губин Е. И. Методика обработки и нормализации цветовых каналов графических данных для прогнозного анализа моделей CNN // Наука и бизнес: Пути развития. 2022. № 6 (121). С. 54–59.

19. Руководство по архитектуре асинхронного взаимодействия сетевых приложений на базе протокола HTTP Long-polling. – URL: https://na-journal.ru/tekhnologii/long-polling-architecture (дата обращения: 11.05.2026).

20. Asynchronous Input/Output: aiogram documentation. – URL: https://docs.aiogram.dev/en/latest/ (дата обращения: 14.05.2026).

21. OpenCV Library Official Reference. – URL: https://docs.opencv.org/4.x/ (дата обращения: 10.02.2026).

22. TensorFlow API Core Modules. – URL: https://www.tensorflow.org/api_docs (дата обращения: 26.03.2026).

23. PyTorch Documentation: Optimization and Neural Networks. – URL: https://pytorch.org/docs/stable/ (дата обращения: 05.05.2026).

ПРИЛОЖЕНИЕ А

(обязательное)

Бот-ведущий с элементами искусственного интеллекта для ролевой игры

Листинг программы

Листинг А.1 – Файл «BoardDetectionTraining.py»

```
from ultralytics import YOLO


def main():
    model = YOLO('yolov8s-pose.pt')

    model.train(
        data='D:/DiplomCode/data.yaml',

        epochs=300,
        patience=80,
        imgsz=800,
        batch=8,
        device=0,
        workers=0,
        freeze=0,

        perspective=0.001,
        shear=0.0,
        degrees=5.0,
        scale=0.2,
        translate=0.05,
        fliplr=0.0,
        flipud=0.0,

        hsv_h=0.015,
        hsv_s=0.2,
        hsv_v=0.2,
        mosaic=0.4,
        erasing=0.0,

        name='checkers_pose'
    )

if __name__ == '__main__':
    main()
```

Листинг А.2 – Файл «CellPicGenerator.py»

```
import os
import glob
import cv2
import numpy as np
from ultralytics import YOLO

INPUT_DIR = r"dataset_for_board_detection\train\images"
OUTPUT_DIR = "results_raw"
MODEL_PATH = "best.pt"

# Загрузка модели
print("Загрузка модели YOLO...")
pose_model = YOLO(MODEL_PATH)


def detect_and_warp_board(image: np.ndarray):
    """
    Использует YOLOv8-pose для поиска 4 углов доски и делает
перспективное преобразование.
    """
    results = pose_model(image, conf=0.5, verbose=False)

    if results[0].keypoints is None or len(results[0].keypoints)
== 0:
        raise ValueError("Углы доски не обнаружены")

    kp = results[0].keypoints.xy.cpu().numpy()[0]
    pts = kp.astype(np.float32)

    if pts.shape[0] != 4:
        raise ValueError(f"Обнаружено не 4 угла, а
{pts.shape[0]}")

    rect = np.zeros((4, 2), dtype=np.float32)
    s = pts.sum(axis=1)
    rect[0] = pts[np.argmin(s)]
    rect[2] = pts[np.argmax(s)]

    diff = np.diff(pts, axis=1)
    rect[1] = pts[np.argmin(diff)]
    rect[3] = pts[np.argmax(diff)]

    dst = np.array([
        [0, 0],
        [800, 0],
        [800, 800],
        [0, 800]
```

```
    ], dtype=np.float32)

    M = cv2.getPerspectiveTransform(rect, dst)
    warped = cv2.warpPerspective(image, M, (800, 800))

    return warped


def slice_and_save_cells(warped_img: np.ndarray, original_name:
str):
    """
    Режет выровненное изображение 800x800 на 64 клетки (100x100)
и сохраняет их.
    """
    os.makedirs(OUTPUT_DIR, exist_ok=True)
    h, w = 100, 100
    saved_count = 0

    # Убираем расширение (.jpg, .png) из имени файла
    base_name = os.path.splitext(original_name)[0]

    for row in range(8):
        for col in range(8):
            y = row * h
            x = col * w
            cell_img = warped_img[y:y + h, x:x + w]

            # Имя файла: оригинальноеимя_rСтрока_cСтолбец.jpg
            filename = os.path.join(OUTPUT_DIR,
f"{base_name}_r{row}_c{col}.jpg")
            cv2.imwrite(filename, cell_img)
            saved_count += 1

    return saved_count


def main():
    # Собираем список всех изображений в папке
    search_pattern = os.path.join(INPUT_DIR, "*.*")
    image_files = [f for f in glob.glob(search_pattern) if
f.lower().endswith(('.png', '.jpg', '.jpeg'))]

    if not image_files:
        print(f"В папке {INPUT_DIR} не найдено изображений.
Проверьте путь.")
        return

    print(f"Найдено изображений для обработки:
{len(image_files)}")
```

```
    print("-" * 40)

    success_count = 0

    # Обрабатываем каждое изображение
    for img_path in image_files:
        filename = os.path.basename(img_path)
        img = cv2.imread(img_path)

        if img is None:
            print(f"❌ Ошибка чтения файла (возможно, поврежден):
{filename}")
            continue

        try:
            # 1. Находим углы и выравниваем доску
            warped_img = detect_and_warp_board(img)

            # 2. Нарезаем на 64 клетки и сохраняем
            saved_cells = slice_and_save_cells(warped_img,
filename)

            print(f"✅ Успех: {filename} -> нарезано на
{saved_cells} клеток.")
            success_count += 1

        except ValueError as e:
            print(f"⚠️ Пропущено: {filename} ({e})")
        except Exception as e:
            print(f"❌ Критическая ошибка при обработке
{filename}: {e}")

    print("\n" + "=" * 40)
    print(f"Обработка завершена! Успешно распознано досок:
{success_count} из {len(image_files)}.")
    print(f"Всего создано {success_count * 64} файлов.")
    print(f"Результаты лежат в папке:
{os.path.abspath(OUTPUT_DIR)}")


if __name__ == '__main__':
    main()
```

Листинг А.3 – Файл «CellDetectionTraining.py»

```
import torch
from torch import nn, optim
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms
from PIL import Image
import os


# ---------- 1. Определение модели (продвинутая CNN) ----------
class CheckerCellCNN(nn.Module):
    def __init__(self, num_classes=3):
        super().__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(2, 2),

            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(2, 2),

            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.MaxPool2d(2, 2)
        )

        self.classifier = nn.Sequential(
            nn.Dropout(0.5),  # Мощная защита от переобучения
            nn.Linear(128 * 12 * 12, 128),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(128, num_classes)
        )

    def forward(self, x):
        x = self.features(x)
        x = x.view(x.size(0), -1)
        x = self.classifier(x)
        return x


# ---------- 2. Датасет ----------
class CheckerCellDataset(Dataset):
    def __init__(self, root_dir, transform=None):
```

```
        self.classes = ['empty_dark', 'white_man', 'black_man']
        self.class_to_idx = {cls: idx for idx, cls in
enumerate(self.classes)}
        self.transform = transform
        self.images = []
        self.labels = []

        for class_name in self.classes:
            class_dir = os.path.join(root_dir, class_name)
            if not os.path.isdir(class_dir):
                print(f"Предупреждение: папка {class_dir} не
найдена")
                continue
            for fname in os.listdir(class_dir):
                if fname.lower().endswith(('.png', '.jpg',
'.jpeg')):
                    self.images.append(os.path.join(class_dir,
fname))

self.labels.append(self.class_to_idx[class_name])
        print(f"Загружено {len(self.images)} изображений из
{root_dir}")

    def __len__(self):
        return len(self.images)

    def __getitem__(self, idx):
        img_path = self.images[idx]
        image = Image.open(img_path).convert('RGB')
        label = self.labels[idx]
        if self.transform:
            image = self.transform(image)
        return image, label


# ---------- 3. Трансформации ----------
normalize = transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5,
0.5, 0.5])

train_transform = transforms.Compose([
    transforms.Resize((100, 100)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomVerticalFlip(),
    transforms.RandomRotation(degrees=15),
    transforms.RandomAffine(degrees=0, translate=(0.1, 0.1),
scale=(0.9, 1.1)),
    transforms.ColorJitter(brightness=0.2, contrast=0.2,
saturation=0.1, hue=0.05),
    transforms.ToTensor(),
```

```
    normalize
])

val_transform = transforms.Compose([
    transforms.Resize((100, 100)),
    transforms.ToTensor(),
    normalize
])

# ---------- 4. Загрузка данных ----------
train_dataset =
CheckerCellDataset('dataset_for_cell_detection/train',
transform=train_transform)
val_dataset =
CheckerCellDataset('dataset_for_cell_detection/val',
transform=val_transform)

train_loader = DataLoader(train_dataset, batch_size=32,
shuffle=True, num_workers=0)
val_loader = DataLoader(val_dataset, batch_size=32,
shuffle=False, num_workers=0)

# ---------- 5. Обучение ----------
device = torch.device('cuda' if torch.cuda.is_available() else
'cpu')
model = CheckerCellCNN(num_classes=3).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

num_epochs = 30
best_val_acc = 0.0

print(f"Обучение на устройстве: {device}")

for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()

    # Валидация
    model.eval()
    val_loss = 0.0
```

```
    correct = 0
    total = 0
    with torch.no_grad():
        for images, labels in val_loader:
            images, labels = images.to(device),
labels.to(device)
            outputs = model(images)

            # Подсчет функции потерь на валидации
            v_loss = criterion(outputs, labels)
            val_loss += v_loss.item()

            # Подсчет точности
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    avg_train_loss = running_loss / len(train_loader)
    avg_val_loss = val_loss / len(val_loader)
    val_accuracy = 100 * correct / total if total > 0 else 0.0

    print(
        f'Epoch [{epoch + 1}/{num_epochs}] | Train Loss:
{avg_train_loss:.4f} | Val Loss: {avg_val_loss:.4f} | Val Acc:
{val_accuracy:.2f}%')

    if val_accuracy >= best_val_acc:
        best_val_acc = val_accuracy
        torch.save(model.state_dict(), 'checker_model_new.pth')
        print(f"   -> Найдена лучшая модель! Сохранено.
(Точность: {val_accuracy:.2f}%)")

print(f"Обучение завершено. Лучшая точность:
{best_val_acc:.2f}%")
```

Листинг А.4 – Файл «MainTelegramBot.py»

```
import asyncio
import io
import logging
import random
import cv2
import numpy as np
import torch
import torchvision.transforms as transforms
from torch import nn

from aiogram import Bot, Dispatcher, types, F
from aiogram.fsm.context import FSMContext
from aiogram.fsm.state import State, StatesGroup
from aiogram.fsm.storage.memory import MemoryStorage
from aiogram.filters import Command
from aiogram.types import InlineKeyboardMarkup,
InlineKeyboardButton, FSInputFile
from aiogram.utils.keyboard import InlineKeyboardBuilder
from ultralytics import YOLO

# ---------- Логирование и Инициализация ----------
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

TOKEN = "7961256432:AAFYVpz3qIL9uG9CWM_u2D6mZnXeaPLvuCw"
bot = Bot(token=TOKEN)
dp = Dispatcher(storage=MemoryStorage())

pose_model = YOLO('best.pt')


# ---------- Состояния FSM ----------
class Steps(StatesGroup):
    START = State()
    ASK_PHOTO = State()
    ASK_COORD = State()
    CONFIRM = State()
    ASK_COLOR = State()


# ---------- Модель CNN ----------
class CheckerCNN(nn.Module):
    def __init__(self, num_classes=3):
        super().__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, padding=1),
            nn.BatchNorm2d(32),
```

```
            nn.ReLU(),
            nn.MaxPool2d(2, 2),

            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(2, 2),

            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.MaxPool2d(2, 2)
        )

        self.classifier = nn.Sequential(
            nn.Dropout(0.5),
            nn.Linear(128 * 12 * 12, 128),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(128, num_classes)
        )

    def forward(self, x):
        x = self.features(x)
        x = x.view(x.size(0), -1)
        x = self.classifier(x)
        return x


device = torch.device("cuda" if torch.cuda.is_available() else
"cpu")
model = CheckerCNN(num_classes=3)
model.load_state_dict(torch.load("checker_model_new.pth",
map_location=device))
model.to(device)
model.eval()

transform = transforms.Compose([
    transforms.ToPILImage(),
    transforms.Resize((100, 100)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5,
0.5])
])


# ---------- Обработка изображений ----------
def detect_and_warp_board(image: np.ndarray):
    results = pose_model(image, conf=0.5, verbose=False)
```

```
    if results[0].keypoints is None or len(results[0].keypoints)
== 0:
        raise ValueError("Углы не распознаны")
    kp = results[0].keypoints.xy.cpu().numpy()[0]
    pts = kp.astype(np.float32)

    if pts.shape[0] != 4:
        raise ValueError("Обнаружено не 4 угла")

    rect = np.zeros((4, 2), dtype=np.float32)
    s = pts.sum(axis=1)
    rect[0] = pts[np.argmin(s)]
    rect[2] = pts[np.argmax(s)]
    diff = np.diff(pts, axis=1)
    rect[1] = pts[np.argmin(diff)]
    rect[3] = pts[np.argmax(diff)]

    dst = np.array([[0, 0], [800, 0], [800, 800], [0, 800]],
dtype=np.float32)
    M = cv2.getPerspectiveTransform(rect, dst)
    warped = cv2.warpPerspective(image, M, (800, 800))
    return warped


def extract_cell(image, row, col):
    h, w = 100, 100
    x = col * w
    y = row * h
    return image[y:y + h, x:x + w]


def classify_cell(cell_img):
    cell_rgb = cv2.cvtColor(cell_img, cv2.COLOR_BGR2RGB)
    tensor = transform(cell_rgb).unsqueeze(0).to(device)
    with torch.no_grad():
        output = model(tensor)
        _, pred = torch.max(output, 1)
    return pred.item()


def get_board_state(warped_img, top_left_coord):
    file_start = ord(top_left_coord[0]) - ord('a')
    rank_start = int(top_left_coord[1]) - 1

    if top_left_coord == 'a8':
        file_step, rank_step = 1, -1
    else:
        file_step, rank_step = -1, 1
```

```
    board = np.zeros((8, 8), dtype=int)
    overlay = warped_img.copy()

    for row in range(8):
        for col in range(8):
            real_file = file_start + col * file_step
            real_rank = rank_start + row * rank_step

            if (real_file + real_rank) % 2 != 0:
                continue

            cell = extract_cell(warped_img, row, col)
            cls = classify_cell(cell)
            notation = f"{chr(ord('a') + real_file)}{real_rank +
1}"

            if cls == 1:
                board[row][col] = 1
                color = (0, 255, 0)
            elif cls == 2:
                board[row][col] = -1
                color = (0, 0, 255)
            else:
                board[row][col] = 0
                color = (128, 128, 128)

            cx = col * 100 + 50
            cy = row * 100 + 50

            cv2.circle(overlay, (cx, cy), 30, color, 2)
            cv2.putText(overlay, notation, (cx - 15, cy - 40),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)

    return board, overlay


def index_to_notation_real(r, c, top_left):
    file_start = ord(top_left[0]) - ord('a')
    rank_start = int(top_left[1]) - 1

    if top_left == 'a8':
        file_step, rank_step = 1, -1
    else:
        file_step, rank_step = -1, 1

    real_file = file_start + c * file_step
    real_rank = rank_start + r * rank_step

    if (real_file + real_rank) % 2 != 0:
```

```
        return None

    return f"{chr(ord('a') + real_file)}{real_rank + 1}"


# ---------- Игровой Движок AI (Minimax + Alpha-Beta Pruning) --
--------

def get_forward_dir(player, top_left):
    if top_left == 'a8':
        return -1 if player == 1 else 1
    else:
        return 1 if player == 1 else -1


def apply_move(board, move, player):
    new_board = board.copy()
    start = move['path'][0]
    end = move['path'][-1]
    new_board[start[0]][start[1]] = 0
    new_board[end[0]][end[1]] = player
    for cr, cc in move.get('captured_pieces', []):
        new_board[cr][cc] = 0
    return new_board


def find_captures(board, player, r, c, current_path=None,
captured_pieces=None):
    if current_path is None: current_path = [(r, c)]
    if captured_pieces is None: captured_pieces = []

    found_further_capture = False
    results = []

    for dr, dc in [(-1, -1), (-1, 1), (1, -1), (1, 1)]:
        nr, nc = r + dr, c + dc
        nnr, nnc = r + 2 * dr, c + 2 * dc

        if 0 <= nnr < 8 and 0 <= nnc < 8:
            if board[nr][nc] == -player and (nr, nc) not in
captured_pieces:
                if board[nnr][nnc] == 0 or (nnr, nnc) ==
current_path[0]:
                    found_further_capture = True
                    temp_board = board.copy()
                    temp_board[r][c] = 0
                    temp_board[nr][nc] = 0
                    temp_board[nnr][nnc] = player
```

```
                    res = find_captures(temp_board, player, nnr,
nnc,
                                        current_path + [(nnr,
nnc)],
                                        captured_pieces + [(nr,
nc)])
                    results.extend(res)

    if not found_further_capture and len(current_path) > 1:
        results.append({'path': current_path, 'is_capture':
True, 'captured_pieces': captured_pieces})

    return results


def get_moves(board, player, top_left):
    captures = []
    for r in range(8):
        for c in range(8):
            if board[r][c] == player:
                captures.extend(find_captures(board, player, r,
c))

    if captures:
        unique_captures = []
        seen = set()
        for cap in captures:
            p_tuple = tuple(cap['path'])
            if p_tuple not in seen:
                seen.add(p_tuple)
                unique_captures.append(cap)
        return unique_captures

    normal_moves = []
    forward_dir = get_forward_dir(player, top_left)
    for r in range(8):
        for c in range(8):
            if board[r][c] == player:
                for dc in [-1, 1]:
                    nr, nc = r + forward_dir, c + dc
                    if 0 <= nr < 8 and 0 <= nc < 8 and
board[nr][nc] == 0:
                        normal_moves.append({'path': [(r, c),
(nr, nc)], 'is_capture': False, 'captured_pieces': []})
    return normal_moves


def evaluate_board(board, maximizing_player, top_left):
    player_pieces = np.sum(board == maximizing_player)
```

```
    opp_pieces = np.sum(board == -maximizing_player)

    if player_pieces == 0: return -1000
    if opp_pieces == 0: return 1000

    score = (player_pieces - opp_pieces) * 10

    for r in range(8):
        for c in range(8):
            if board[r][c] == maximizing_player:
                if 2 <= r <= 5 and 2 <= c <= 5: score += 0.5
                if c == 0 or c == 7: score += 0.5
    return score


def minimax(board, depth, alpha, beta, maximizing_player_id,
current_player, top_left):
    if depth == 0:
        return evaluate_board(board, maximizing_player_id,
top_left)

    moves = get_moves(board, current_player, top_left)
    if not moves:
        return -10000 if current_player == maximizing_player_id
else 10000

    if current_player == maximizing_player_id:
        max_eval = -float('inf')
        for move in moves:
            new_board = apply_move(board, move, current_player)
            ev = minimax(new_board, depth - 1, alpha, beta,
maximizing_player_id, -current_player, top_left)
            max_eval = max(max_eval, ev)
            alpha = max(alpha, ev)
            if beta <= alpha:
                break
        return max_eval
    else:
        min_eval = float('inf')
        for move in moves:
            new_board = apply_move(board, move, current_player)
            ev = minimax(new_board, depth - 1, alpha, beta,
maximizing_player_id, -current_player, top_left)
            min_eval = min(min_eval, ev)
            beta = min(beta, ev)
            if beta <= alpha:
                break
        return min_eval
```

```
def get_best_moves(board, player, top_left):
    moves = get_moves(board, player, top_left)
    if not moves: return []

    scored_moves = []
    for move in moves:
        new_board = apply_move(board, move, player)

        # --- ПРОВЕРКА ОТВЕТНОГО ВЗЯТИЯ ПРОТИВНИКА ---
        opp_moves = get_moves(new_board, -player, top_left)
        max_opp_captures = 0
        if opp_moves:
            # Находим ход противника, который забирает максимум
наших фигур
            max_opp_captures = max(len(m.get('captured_pieces',
[])) for m in opp_moves)
        # ---------------------------------------------

        # Глубина 4 = Наш ход + Ответ противника + Наш ответ +
Ответ противника
        future_score = minimax(new_board, 4, -float('inf'),
float('inf'), player, -player, top_left)

        num_captured = len(move.get('captured_pieces', []))
        immediate_bonus = num_captured * 100

        score = immediate_bonus + future_score +
random.uniform(0, 0.1)
        scored_moves.append({
            'move': move,
            'score': score,
            'num_captured': num_captured,
            'max_opp_captures': max_opp_captures
        })

    scored_moves.sort(key=lambda x: x['score'], reverse=True)
    # Расширено до 4 лучших ходов
    top_moves = scored_moves[:4]

    results = []
    for sm in top_moves:
        path_notations = [index_to_notation_real(r, c, top_left)
for r, c in sm['move']['path']]
        move_str = " ➡ ".join(path_notations)

        num_cap = sm['num_captured']
        if sm['move']['is_capture']:
            if num_cap >= 3:
```

```
                reason = f"🔥 Блестящая комбинация! Множественное взятие ({num_cap} шашки)."
            elif num_cap == 2:
                reason = f"💥 Отличный двойной удар! За один ход срубается {num_cap} шашки."
            else:
                reason = f"⚔️ Обязательный бой. Забираем 1 шашку противника."
        else:
            score = sm['score']
            if score > 5:
                reason = "🎯 Сильный позиционный ход. Берет центр под контроль."
            elif score < -5:
                reason = "⚠️ Единственный легальный ход, но нужно быть осторожным."
            else:
                reason = "🚶‍♂️ Надежный развивающий ход без риска."

        # --- ФОРМИРОВАНИЕ ПРЕДУПРЕЖДЕНИЯ ---
        if sm['max_opp_captures'] > 0:
            reason += f"\n🚨 *Внимание: После этого хода вы рискуете потерять до {sm['max_opp_captures']} фигур за раз.*"
        else:
            # Опционально: можно сообщать о безопасности хода
            reason += f"\n🛡️ *Отличные новости:* Ход кажется безопасным."

        results.append({'text': move_str, 'reason': reason})

    return results


# ---------- Клавиатуры ----------
def main_menu_kb():
    builder = InlineKeyboardBuilder()
    builder.button(text="📖 Правила игры", callback_data="rules")
    builder.button(text="🚀 Начать", callback_data="start_game")
    return builder.as_markup()


def game_menu_kb():
    builder = InlineKeyboardBuilder()
    builder.button(text="🏠 Главное меню", callback_data="main_menu")
```

```
    return builder.as_markup()


def yes_no_kb():
    builder = InlineKeyboardBuilder()
    builder.button(text="✅ Да", callback_data="yes")
    builder.button(text="❌ Нет", callback_data="no")
    return builder.as_markup()


def color_kb():
    builder = InlineKeyboardBuilder()
    builder.button(text="⚪ Белые", callback_data="color_white")
    builder.button(text="⚫ Чёрные", callback_data="color_black")
    return builder.as_markup()


# ---------- Обработчики aiogram ----------
@dp.message(Command("start"))
async def cmd_start(message: types.Message, state: FSMContext):
    await state.set_state(Steps.START)
    user = message.from_user
    try:
        photo = FSInputFile("StartBoard.jpg")
        caption = (f"Привет, {user.first_name}!\n"
                   "Я твой ИИ-помощник для игры в русские шашки
🤖\n\n"
                   "Просто отправь мне фото доски, и я
проанализирую позицию "
                   "с помощью нейросетей, чтобы предложить тебе
лучшие ходы!\n\n"
                   "Жми «Правила игры» или «Начать».")
        await message.answer_photo(photo, caption=caption,
reply_markup=main_menu_kb())
    except FileNotFoundError:
        await message.answer(
            f"Привет, {user.first_name}!\nЯ бот-помощник для
игры в шашки.\n"
            "Жми «Правила игры» или «Начать».",
            reply_markup=main_menu_kb()
        )


@dp.callback_query()
async def callback_handler(callback: types.CallbackQuery, state:
FSMContext):
    data = callback.data
    await callback.answer()
```

```
    if data == "rules":
        rules_text = (
            "📜 *Официальные правила (Русские шашки):*\n\n"
            "RU В русские шашки играют на доске 8x8. У каждого
игрока по 12 фигур.\n\n"
            "🚶‍♂️ *Ходы:* Шашки ходят только по тёмным полям по
диагонали на одну клетку вперёд.\n\n"
            "⚔️ *Взятие (Бой):*\n"
            "• Взятие строго обязательно! Если есть возможность
побить шашку противника — вы обязаны это сделать.\n"
            "• Бить можно не только вперёд, но и **назад**.\n"
            "• Если после одного взятия можно побить ещё одну
шашку, взятие продолжается (серийный бой).\n"
            "• При наличии нескольких вариантов боя игрок может
выбрать любой.\n\n"
            "🏆 *Победа:* Наступает, если все шашки соперника
съедены или заперты (лишены ходов)."
        )
        await callback.message.answer(rules_text,
parse_mode="Markdown", reply_markup=main_menu_kb())

    elif data == "start_game":
        await state.set_state(Steps.ASK_PHOTO)
        await
callback.message.edit_reply_markup(reply_markup=game_menu_kb())
        await callback.message.answer("📸 Отправь мне фотографию
доски с текущим расположением шашек.")

    elif data == "main_menu":
        await state.set_state(Steps.START)
        await
callback.message.edit_reply_markup(reply_markup=main_menu_kb())

    elif data == "yes":
        await confirm_yes(callback, state)

    elif data == "no":
        await confirm_no(callback, state)

    elif data in ["color_white", "color_black"]:
        await color_selected(callback, state)


@dp.message(F.photo)
async def photo_handler(message: types.Message, state:
FSMContext):
    current_state = await state.get_state()
```

```
    if current_state != Steps.ASK_PHOTO.state:
        await message.answer("Сейчас не ожидается фото. Нажми
/start для выхода в меню.")
        return

    msg = await message.answer("⚙️ Нейросети обрабатывают
фотографию, подожди пару секунд...")

    photo = message.photo[-1]
    file = await message.bot.get_file(photo.file_id)
    img_bytes = await message.bot.download_file(file.file_path)
    nparr = np.frombuffer(img_bytes.read(), np.uint8)
    img = cv2.imdecode(nparr, cv2.IMREAD_COLOR)

    try:
        warped = detect_and_warp_board(img)
    except Exception as e:
        await msg.edit_text(f"❌ Не удалось найти доску:
{e}\nПопробуйте сфотографировать ровнее.")
        return

    await state.update_data(warped_img=warped)
    await state.set_state(Steps.ASK_COORD)
    await msg.delete()
    await message.answer("Укажите координату левой верхней
клетки.\n"
                         "Доска должна располагаться так, чтобы
это была белая клетка.\n"
                         "Доступные варианты: **a8** или
**h1**", parse_mode="Markdown")


@dp.message(F.text)
async def text_handler(message: types.Message, state:
FSMContext):
    current_state = await state.get_state()
    if current_state == Steps.ASK_COORD:
        coord = message.text.strip().lower()

        if coord not in ['a8', 'h1']:
            await message.answer("❌ Ошибка! Введите строго
**a8** или **h1**.", parse_mode="Markdown")
            return

        data = await state.get_data()
        warped = data.get("warped_img")
        if warped is None:
            await message.answer("Сначала отправь фото.")
```

```
            await state.set_state(Steps.ASK_PHOTO)
            return

        msg = await message.answer("🔍 Распознаю шашки на
доске...")

        board, overlay = get_board_state(warped, coord)
        await state.update_data(board=board, top_left=coord)
        await state.set_state(Steps.CONFIRM)

        _, buf = cv2.imencode('.png', overlay)
        bio = io.BytesIO(buf.tobytes())
        bio.seek(0)

        caption_text = (
            "Правильно ли бот считал фотографию?\n\n"
            "🟢 Зелёный - Белые шашки\n"
            "🔴 Красный - Чёрные шашки\n"
            "⭕ Серый - Пустые чёрные клетки"
        )

        await msg.delete()
        await message.answer_photo(
            types.BufferedInputFile(bio.read(),
filename="board.png"),
            caption=caption_text,
            reply_markup=yes_no_kb()
        )
    else:
        await cmd_start(message, state)


async def confirm_yes(callback: types.CallbackQuery, state:
FSMContext):
    data = await state.get_data()
    board = data.get("board")

    white_count = np.sum(board == 1)
    black_count = np.sum(board == -1)

    if white_count == 0:
        await
callback.message.edit_reply_markup(reply_markup=None)
        await callback.message.answer("🏆 **Партия
окончена!**\nЧёрные победили, на доске не осталось белых
шашек.",
                                      parse_mode="Markdown",
reply_markup=main_menu_kb())
```

```
        await state.set_state(Steps.START)
        return
    elif black_count == 0:
        await
callback.message.edit_reply_markup(reply_markup=None)
        await callback.message.answer("🏆 **Партия
окончена!**\nБелые победили, на доске не осталось чёрных
шашек.",
                                      parse_mode="Markdown",
reply_markup=main_menu_kb())
        await state.set_state(Steps.START)
        return

    await callback.message.edit_reply_markup(reply_markup=None)
    await callback.message.answer("🎯 Отлично! Для какого цвета
будем искать лучший ход?", reply_markup=color_kb())
    await state.set_state(Steps.ASK_COLOR)


async def confirm_no(callback: types.CallbackQuery, state:
FSMContext):
    await callback.message.edit_reply_markup(reply_markup=None)
    await callback.message.answer("📸 Отправь новую фотографию
доски.", reply_markup=game_menu_kb())
    await state.set_state(Steps.ASK_PHOTO)


async def color_selected(callback: types.CallbackQuery, state:
FSMContext):
    await callback.message.edit_reply_markup(reply_markup=None)

    player_color = 1 if callback.data == "color_white" else -1
    color_name = "белых" if player_color == 1 else "чёрных"

    data = await state.get_data()
    board = data.get("board")
    top_left = data.get("top_left")

    msg_temp = await callback.message.answer("🧠 Анализирую
1000+ вариантов развития событий...")

    best_moves = get_best_moves(board, player_color, top_left)

    if not best_moves:
        await msg_temp.edit_text(f"У {color_name} нет доступных
ходов (шашки заперты). Игра окончена.",
                                 reply_markup=main_menu_kb())
    else:
```

```
        msg = f"🧠 **Анализ (Глубина: 5 полуходов) для
{color_name}:**\n\n"
        for i, move_info in enumerate(best_moves, 1):
            msg += f"*{i}.* `{move_info['text']}`\n"
            msg += f"_{move_info['reason']}_\n\n"

        msg += "Выбирай наиболее подходящий под твою тактику!"
        await msg_temp.edit_text(msg, parse_mode="Markdown",
reply_markup=main_menu_kb())

    await state.set_state(Steps.START)


# ---------- Точка входа ----------
async def main():
    await bot.delete_webhook(drop_pending_updates=True)
    await dp.start_polling(bot)


if __name__ == "__main__":
    asyncio.run(main())
```

ПРИЛОЖЕНИЕ Б

(обязательное)

Бот-ведущий с элементами искусственного интеллекта для ролевой игры

Презентация

Сетевое приложение для создания и проведения тестирования, сбора данных и результатов

Автор работы: студент группы 24ВВП1
Кочергин В.П.

Научный руководитель: к.т.н. доцент кафедры ВТ
Карамышева Н.С.

Актуальность исследования

Проблема: Высокий порог вхождения для новичков в настольных играх и дефицит доступных систем интерактивного наставничества.

Недостатки существующих аналогов (Lidraughts.org, Checkers.com):

- Обучение происходит исключительно в заранее прописанной виртуальной среде, в виде уровней.
- Часто навязывают жесткую монетизацию и скрывают обучающие функции за платными подписками.

Научно-практическая новизна: Разработка «умного помощника», способного мгновенно распознавать произвольное состояние физической шашечной доски по обычной фотографии пользователя, преобразовывать её в цифровой формат и вычислять лучшие ходы.

Цель, объект, предмет исследования

Цель работы: Разработка интеллектуального чат-бота для Telegram, способного по фотографии находить и подсказывать оптимальные ходы в шашечной партии в режиме реального времени.

Объект исследования: Процессы и методы автоматического распознавания игровой ситуации по фото, а также совместная работа компьютерного зрения с логическими движками.

Предмет исследования: Свёрточные нейронные сети, методы детекции объектов, алгоритмы коррекции перспективы и схемы взаимодействия компонентов системы.

В процессе разработки должны быть решены следующие задачи

- Изучение правил русских шашек, методов компьютерного зрения для поиска досок/фигур и разбор недостатков существующих аналогов.
- Разработка структуры системы, объединяющей Telegram-интерфейс, CV-модуль на базе нейросетей и логический шашечный движок.
- Создание сверточной нейросети для классификации клеток доски по 5 классам (пусто, белая/черная простая, белая/черная дамка).
- Реализация алгоритмов обнаружения игрового поля, коррекции нарушенной перспективы и поклеточной сегментации кадра.
- Настройка аналитического модуля для расчета оптимального полухода и генерации понятной текстовой нотации для пользователя.
- Сборка всех компонентов в готовый продукт в мессенджере Telegram с использованием асинхронной библиотеки **aiogram**.

Алгоритм работы системы

Этап 1: Обнаружение и выравнивание доски
• Получение фотографии от пользователя и поиск игрового поля с помощью модели **YOLOv8-pose**.
• Вычисление 4 угловых точек и автоматическая перспективная коррекция (трансформация в плоский вид сверху).
Этап 2: Сегментация и анализ клеток
• Программное отбрасывание светлых полей для экономии ресурсов.
• Классификация темных клеток собственной нейросетью (**CNN**) на 3 класса: пусто, белая шашка, черная шашка.
Этап 3: Создание цифровой копии и верификация
• Формирование виртуальной матрицы 8x8 и наложение цветных маркеров на фото для проверки пользователем.
Этап 4: Игровой интеллект и рекомендации
• Проверка на обязательный бой (взятие).
• Запуск алгоритма **Minimax** для симуляции вариантов на несколько ходов вперед.
• Балльная оценка сценариев и выдача текстовых рекомендаций с описанием рисков.

Стек технологий

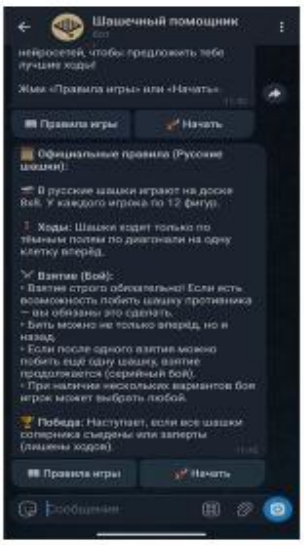
Язык программирования

Библиотеки

Платформа

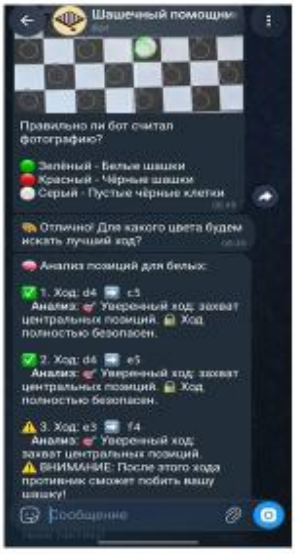
Среда разработки

Демонстрация функционала помощника

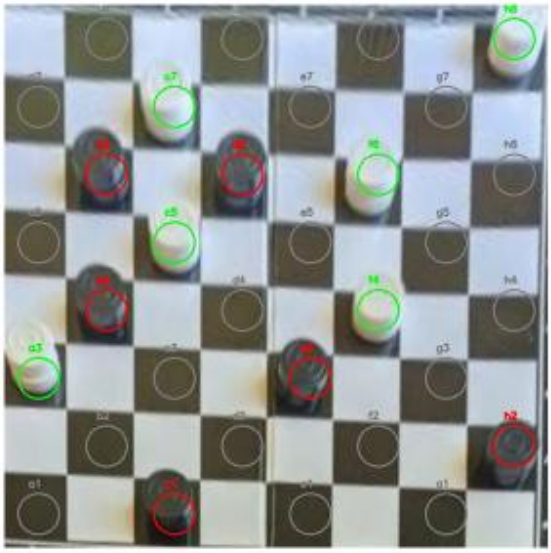


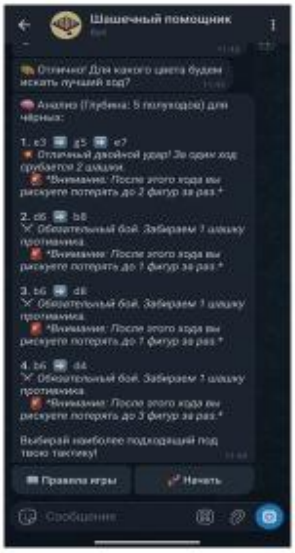
Демонстрация функционала помощника

Демонстрация функционала помощника

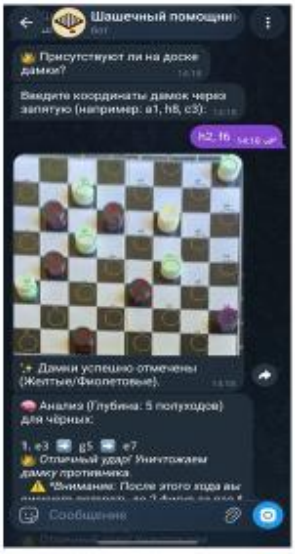


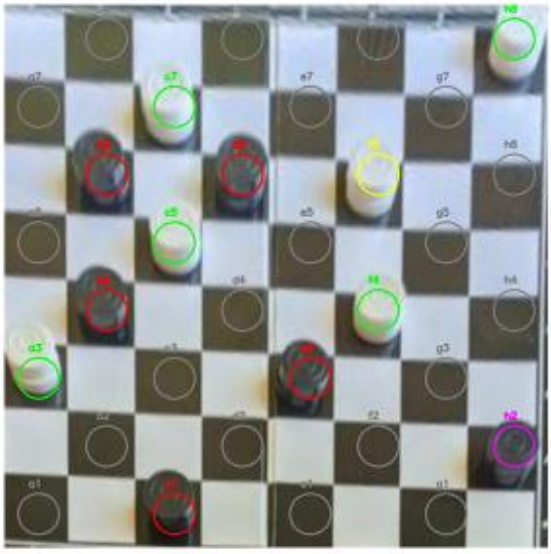
Демонстрация функционала помощника

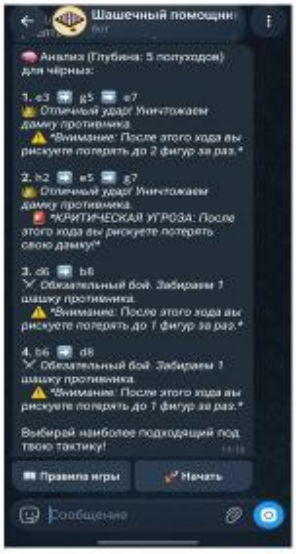




Демонстрация функционала помощника







Заключение

Цель диссертационного исследования успешно достигнута: спроектирован, реализован и протестирован функционирующий Telegram-бот для помощи в игре в русские шашки.

Основные результаты:

- Собраны два уникальных набора данных (датасета) для обучения моделей.
- Успешно обучены и интегрированы ИИ-модели: **YOLOv8-pose** (точное определение границ доски) и **CheckerCellCNN** (классификация состояния клеток).
- Реализован классический алгоритм **Minimax**, обеспечивающий надежный расчет тактики.
- Разработана отказоустойчивая асинхронная архитектура, обрабатывающая входящие фотографии в среднем менее чем за 3 секунды.

Перспективы дальнейшего развития:

- Интеграция поддержки других настольных игр (международные шашки, шахматы).
- Возможность распознавать дамки автоматически по фотографии.