МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

РЕАЛИЗАЦИЯ КЛИЕНТ-СЕРВЕРНОЙ ПРОГРАММНОЙ СИСТЕМЫ «ИГРА В ШАХМАТЫ»

по дисциплинам «Введение в разработку интеллектуальных систем», «Программные средства разработки интеллектуальных систем»

Студент гр. 1308, _____ Мельник Д. А.

Студент гр. 1308, _____ Макаров М. В.

Студент гр. 1308, _____ Томилов Д. Д.

Студент гр. 1308, _____ Лепов А. В.

Санкт-Петербург
2022

СОДЕРЖАНИЕ

1. НАИМЕНОВАНИЕ СИСТЕМЫ

Реализация клиент-серверной программной системы «Игра в Шахматы».

2. ЦЕЛЬ РЕАЛИЗАЦИИ СИСТЕМЫ

Целью реализации данной программной системы является изучение и применение полученных знаний студентами в дисциплинах средств вычислительной техники и теории введения в разработку интеллектуальных систем.

3. ЗАДАЧИ, РЕШАЕМЫЕ СИСТЕМОЙ

Задачей реализации данной программной системы является расчет возможных ходов с достаточной глубиной для предоставления пользователю полноценного игрового опыта.

4. ОБЛАСТЬ ПРИМЕНЕНИЯ СИСТЕМЫ

Областью применения данной системы является обучение пользователей игре в шахматы.

5. ТРЕБОВАНИЯ К ФУНКЦИОНАЛЬНЫМ ХАРАКТЕРИСТИКАМ СИСТЕМЫ

- Полноценный шахматный интерфейс, поддерживающий все возможные ходы и позиции.
- Клиент-серверная архитектура
- Программный модуль для обработки ходов виртуального соперника.

6. ОБОСНОВАНИЕ ЦЕЛЕСООБРАЗНОСТИ РАЗРАБОТКИ СИСТЕМЫ

Данная программная система является целесообразной как учебный проект для применения практических навыков в области средств вычислительной техники и интеллектуальных систем.

7. ДИЗАЙН ИНТЕРФЕЙСА КЛИЕНТСКОЙ ЧАСТИ И ОПИСАНИЕ ПОЛЬЗОВАТЕЛЬСКОГО ОПЫТА

Описание пользовательского опыта:

- При запуске программы пользователь входит в систему в качестве гостя.
- Ему необходимо войти в систему, чтобы изменить роль на пользователя.

На рисунках 1-6 представлен дизайн интерфейса клиентской части.

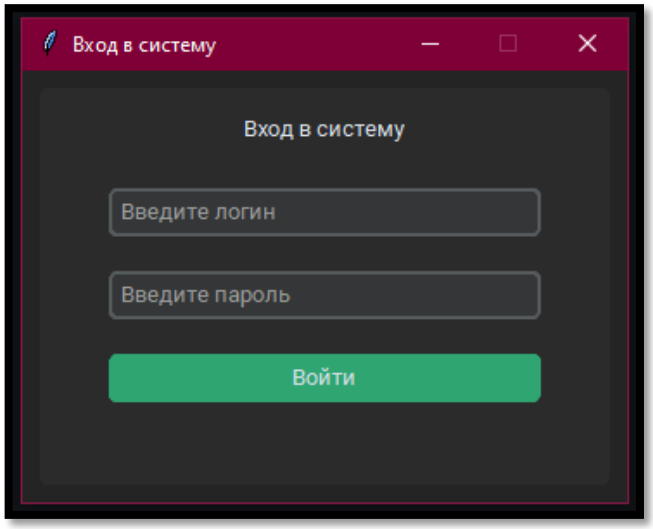

Рис. 1. Интерфейс входа в систему

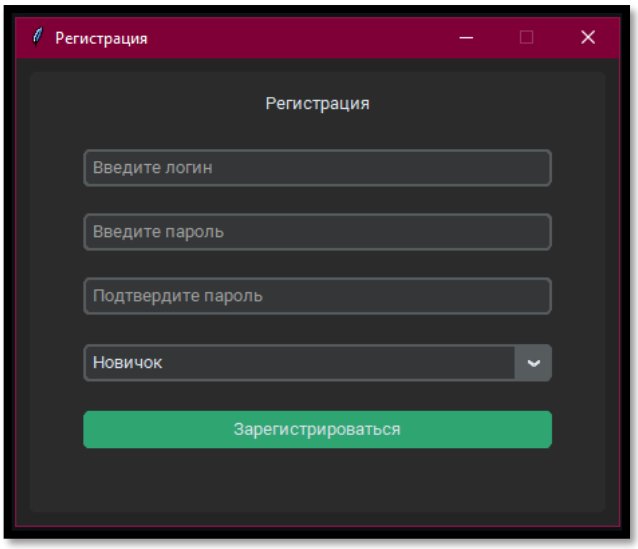

Рис. 2. Интерфейс входа в систему

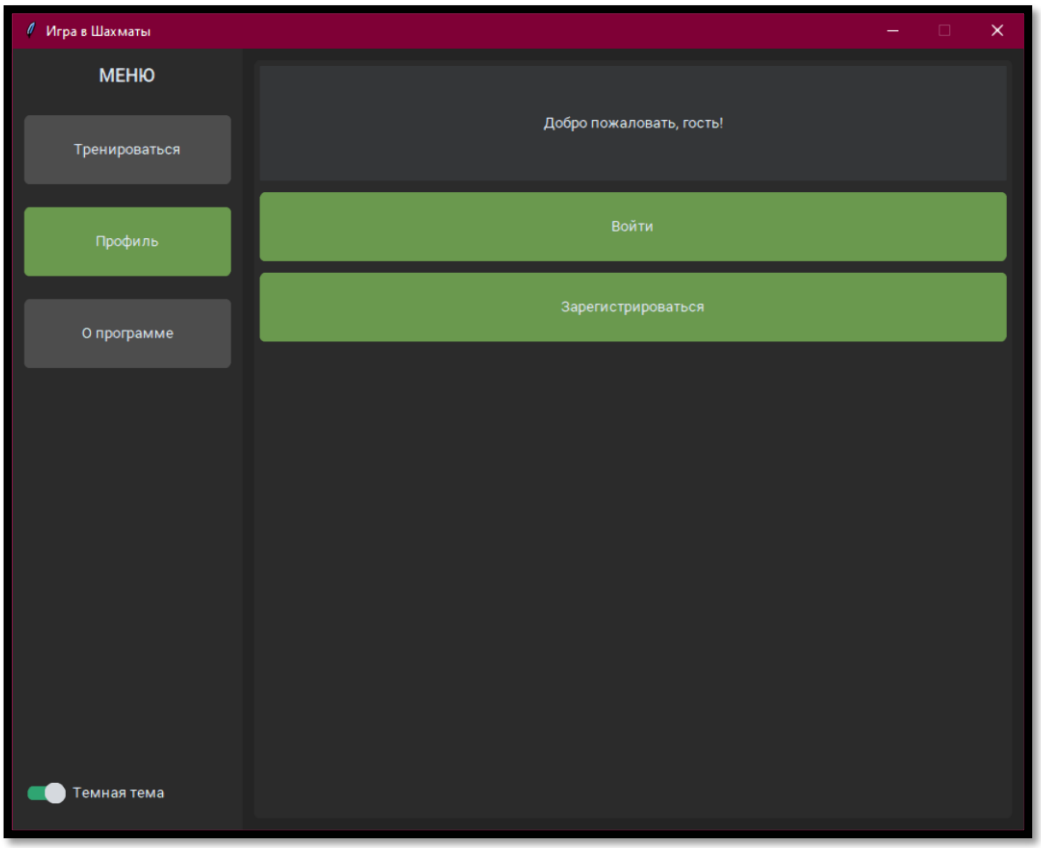

Рис. 3. Интерфейс входа в систему

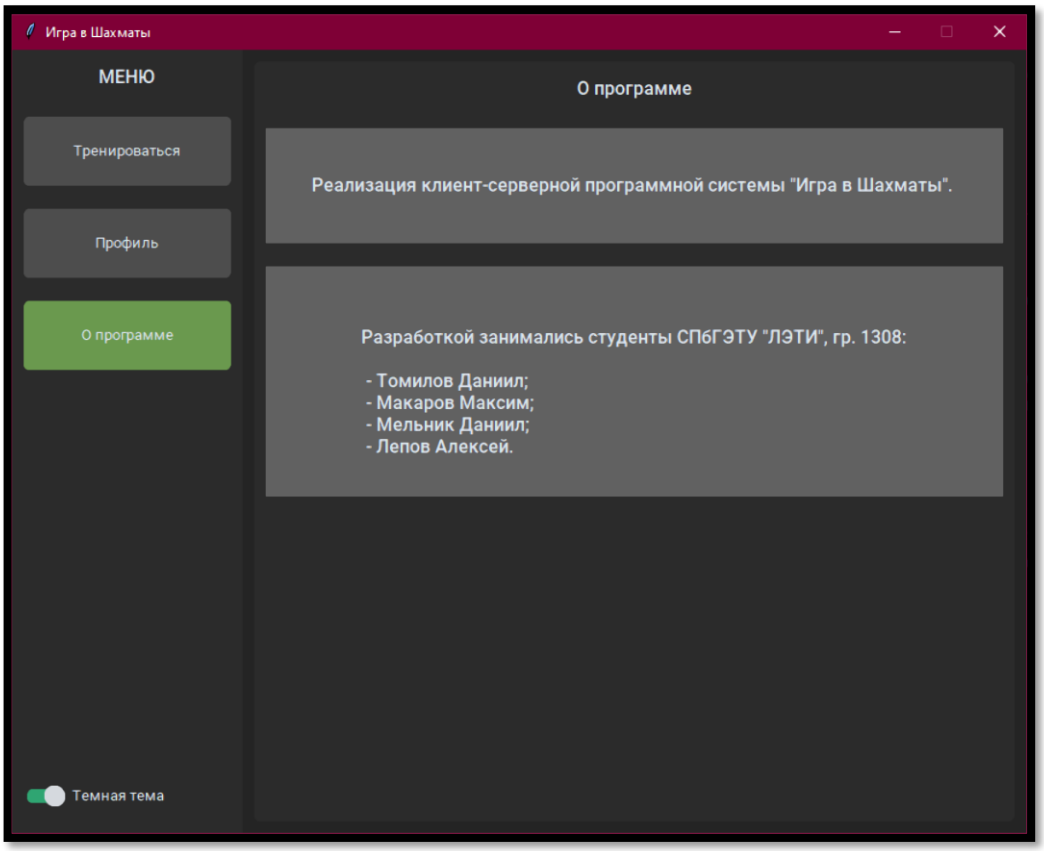

Рис. 4. Интерфейс входа в систему

Рис. 5. Интерфейс входа в систему

Рис. 6. Интерфейс входа в систему

8. ВЫБРАННЫЕ ТЕХНОЛОГИИ

Выбранные технологии:

- Python 3;
- TKinter;
- CustomTkinter;
- IDE: VS Code;
- IDE: PyCharm.

9. РЕАЛИЗАЦИЯ КЛИЕНТСКОЙ ЧАСТИ И СВЯЗЬ ЕЁ С СЕРВЕРНОЙ ЧАСТЬЮ

Технологии серверной части приложения:

При создании серверной части приложения была использованы средства библиотеки socket.

Для привязки сокета к адресу использован метод bind, принимающий присваемый ip и порт в нём.

Для получения сообщения от определённого (по адресу) клиента используется метод recvfrom(), принимающий на вход максимальный объём информации для получения, возвращающий данные и адрес, с которого они были получены

Для отправки сообщений использован метов sendto(), принимающий данные и адрес принимающей стороны.

Технологии клиентской части приложения:

При создании клиентской части приложения была использованы средства библиотек socket и threading.

Для отправки сообщений использован метов sendto(), принимающий данные и адрес принимающей стороны.

Для приёма сообщения был использован метод recv(), принимающий на вход максимальный объём информации, возможный к приёму.

Для корректности попеременной работы клиентов с сервером был реализован поток через объект Thread библиотеки Threading.

10.ТЕСТ-КЕЙСЫ

```
import unittest
import main

class TestStringMethods(unittest.TestCase):

    def test_castling_1(self):
        board = main.chessEngine.Board.from_FEN("r3k2r/8/8/8/8/8/8/R3K2R w KQkq -
0 1")
        board_arr = board.get_piece_arr()
        self.assertTrue(board.move_piece(board_arr[0][4],
main.chessEngine.Position(0,6)))

    def test_false_castling_2(self):
        board = main.chessEngine.Board.from_FEN("rrrrkrrr/8/8/8/8/8/8/RRRRKRRR w
KQkq - 0 1")
        board_arr = board.get_piece_arr()
        self.assertFalse(board.move_piece(board_arr[0][4],
main.chessEngine.Position(0,2)))

    def test_castling_3(self):
        board = main.chessEngine.Board.from_FEN("r3k2r/8/8/8/8/8/8/R3K2R b KQkq -
0 1")
        board_arr = board.get_piece_arr()
        self.assertTrue(board.move_piece(board_arr[7][4],
main.chessEngine.Position(7,6)))

    def test_false_castling_4(self):
        board = main.chessEngine.Board.from_FEN("rrrrkrrr/8/8/8/8/8/8/RRRRKRRR b
KQkq - 0 1")
        board_arr = board.get_piece_arr()
        self.assertFalse(board.move_piece(board_arr[7][4],
main.chessEngine.Position(7,2)))

    def test_piece_eat(self):
        board =
main.chessEngine.Board.from_FEN("rnbqkbnr/ppp1pppp/8/3p4/4P3/8/PPPP1PPP/RNBQKBNR
w KQkq - 0 1")
        board_arr = board.get_piece_arr()
        self.assertTrue(board.move_piece(board_arr[3][4],
main.chessEngine.Position(4,3)))

    def test_false_piece_eat(self):
        board =
main.chessEngine.Board.from_FEN("rnbqkbnr/ppp1pppp/8/3p4/4P3/8/PPPP1PPP/RNBQKBNR
w KQkq - 0 1")
```

```
        board_arr = board.get_piece_arr()
        self.assertFalse(board.move_piece(board_arr[4][3],
main.chessEngine.Position(3,4)))

    def test_queenValue(self):
        self.assertEqual(main.chessEngine.Piece.get_value("queen"), 9)

    def test_getStartWhitePownPos(self):
        self.assertEqual(main.chessEngine.Piece.get_start_position("pawn",
main.chessEngine.Color.WHITE), [main.chessEngine.Position(1, i) for i in
range(8)])

    def test_isupper(self):
        self.assertFalse((main.chessEngine.Piece.__str__(main.chessEngine.Piece(m
ain.chessEngine.Color.BLACK,"pawn",1,None,True))=="White"))

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

11.ПРИМЕР РАБОТЫ

С примерами работы можно ознакомиться, перейдя на онлайн-репозиторий проекта: https://github.com/AlexeyLepov/ClientServerChessApp

12.ВЫБРАННЫЕ ТЕХНОЛОГИИ

Для реализации данной задачи были выбраны следующие технологии:

- Flask и Socket.IO для реализации общения между сервером и клиентом
- MySQL для хранения данных об играх, пользователях и их очках.
- IDE VS Code для написания кода.

13.АРХИТЕКТУРА БД

Подробнее рассмотрим архитектуру создаваемой базы данных.

В ней присутствуют три таблицы:

1. Пользователи (users) - хранит информацию о пользователях
    a. id - персональный идентификатор пользователя
    b. username - его логин в системе
    c. email - его почтовый адрес
    d. password - его пароль (в зашифрованном виде)
    e. score - ELO рейтинг пользователя

2. Игры (games) - хранит информацию о проведенных играх:
    a. idgames - id игры
    b. date - когда была проведена игра
    c. w_player - игрок за белых
    d. b_player - игрок за черных
    e. moves - список ходов игроков
3. Ходы (moves) - хранит информацию о ходах:
    a. idmoves - id хода
    b. file - информация о ходе
    c. w_timeleft - оставшееся время на часах белых
    d. b_timeleft - оставшееся время на часах черных

Связи между ними представлены на следующей ER-диаграмме:

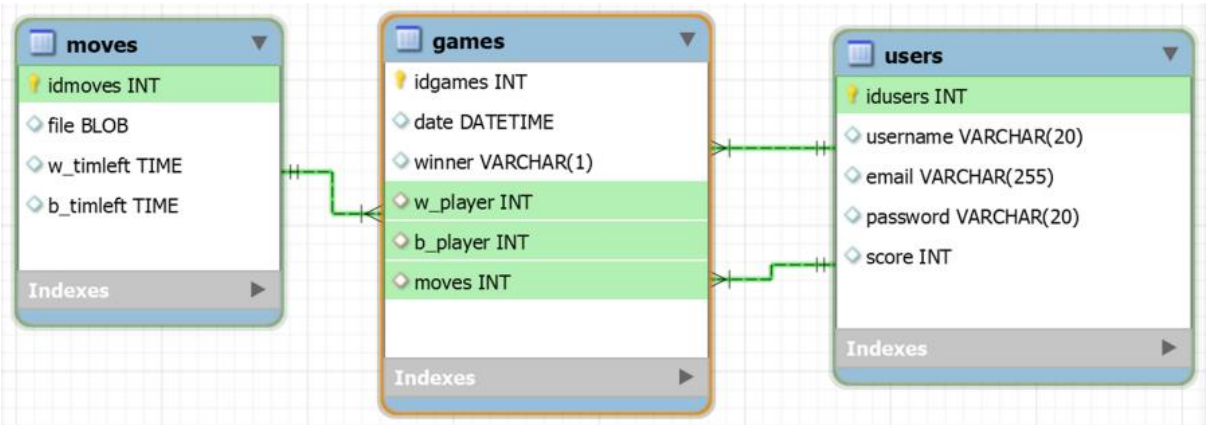

Рис.1 - ER-диаграмма связей в базе данных

14.USE-CASE ДИАГРАММА

Далее на рис.2 представлена use-case диаграмма нашей системы.

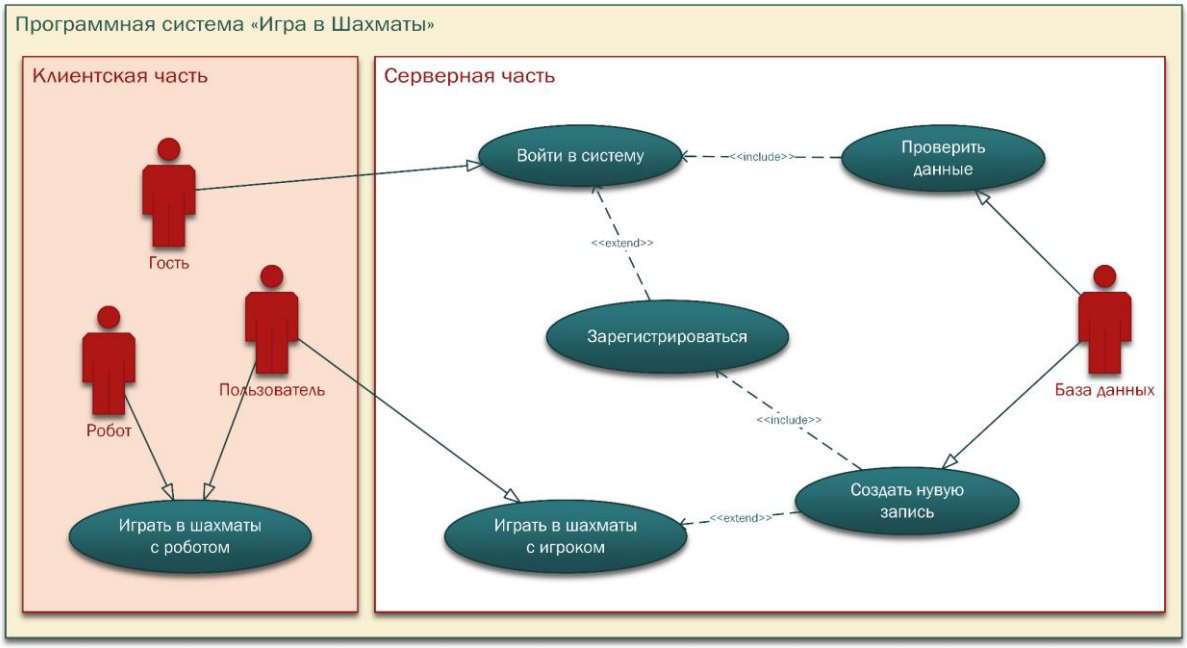

Рис. 2. use-case диаграмма

Как видно из диаграммы, наша программная система разделена на клиентскую и серверную части.

В клиентской части находятся следующие акторы:

- гость
- пользователь (требует входа в систему через сервер)
- робот (для оптимизации работы сервера он перенесен в клиентскую часть)

В то время как сервер содержит только одного актора: базу данных.

Также на диаграмме присутствуют прецеденты:

- прецедент “играть в шахматы с роботом” - для пользователя и робота
- прецедент “войти в систему” - для гостя
- прецедент “зарегистрироваться” - для гостя
- прецедент “проверить данные” - база данных проверяет корректность данных для входа гостя в систему
- прецедент “создать новую запись” - база данных создает запись для зарегистрированного пользователя

15.Реализуемое API

За основу реализации клиент-серверной архитектуры взят концепт Flask-socketio.
Два клиента подключаются к серверу, после чего происходит следующий процесс:

1. Сервер отсылает обоим игрокам, кто ходит первым;
2. Сервер ждёт ответ от активного игрока;
3. Активный игрок делает ход и отправляет на сервер;
4. Сервер проверяет ход на легитимность, после чего есть два варианта:
    a. Ход возможен - новая доска отправляется всем игрокам и синхронизируются часы, активный игрок меняется и процесс продолжается с пункта 2;
    b. Ход невозможен - активному игроку приходит ответ о невозможности кода, процесс продолжается с пункта 2.

16.IDEF0 – ДИАГРАММА

Рис. 16.1. IDEF0 – ДИАГРАММА (TOP)

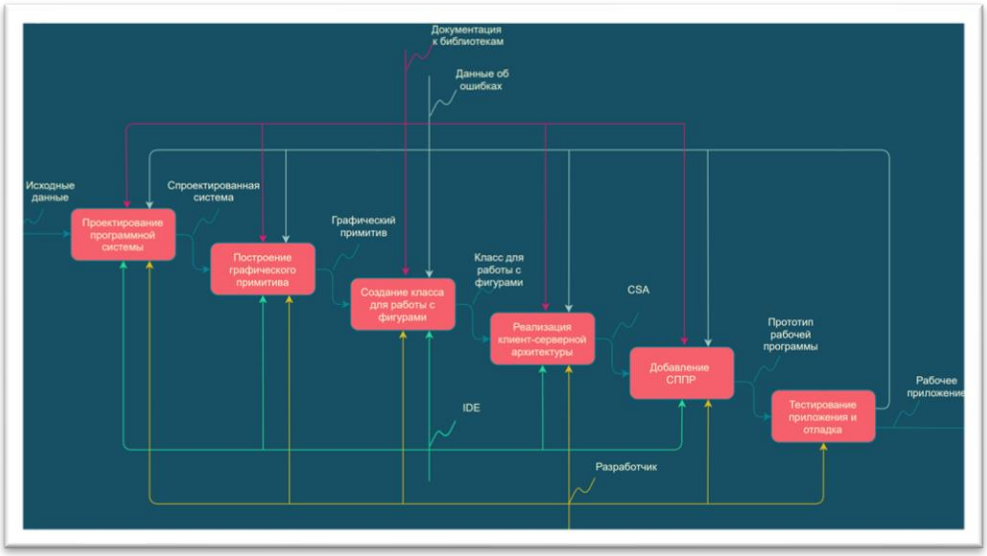

Рис. 16.1. IDEF0 – ДИАГРАММА (CONTEXT)

17.МОДЕЛЬ ПРИНЯТИЯ РЕШЕНИЙ

В качестве модели принятия решений выбран алгоритм минимакс.

Данный алгоритм использует поиск в глубину по древу возможных ходов – затем предлагает решение.

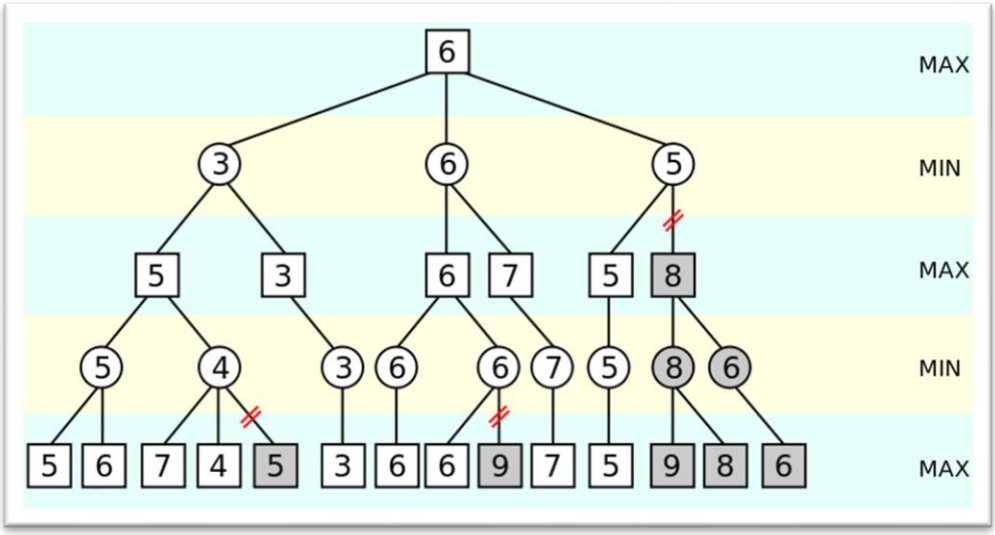

Рис. 17.1. Пример отсечения в модели принятия решений.

18.ОПИСАНИЕ ПОДХОДА К ПРОВЕДЕНИЮ ЭКСПЕРИМЕНТАЛЬНЫХ ИССЛЕДОВАНИЙ

Для проведения экспериментальных исследований написаны модули, включающие в себя проверку работоспособности бота в трех разных позициях:

- Дебют
- Миттельшпиль
- Эндшпиль

19.УСЛОВИЯ ПРОВЕДЕНИЯ ЭКСПЕРИМЕНТОВ

Характеристики ПК, на котором проводились эксперименты:

ЦП: Intel core i3-10110U

ОЗУ: 8 Гб

20.УСЛОВИЯ ПРОВЕДЕНИЯ ЭКСПЕРИМЕНТОВ

Шаги проведения эксперимента:

- Проведено 30 запусков бота на глубинах от 1 до 4 на трех исходных позициях.
- Было запущено 5 игр против другого бота
- Было проведено 5 игр с игроками разного уровня

21.ЭКСПЕРТНАЯ ОЦЕНКА

«Бот делает неожиданные ходы, которые могут застать врасплох.»

- Алексей (~350 ELO)

«В целом мне понравилось. Прикольно! Бот делает интересные ходы.»

- Аня (~750 ELO)

«Бот оказывал сопротивление. Старался не зевать свои ценные фигуры.»

- Никита (1450 ELO)

22.ИНТЕРПРЕТАЦИЯ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ

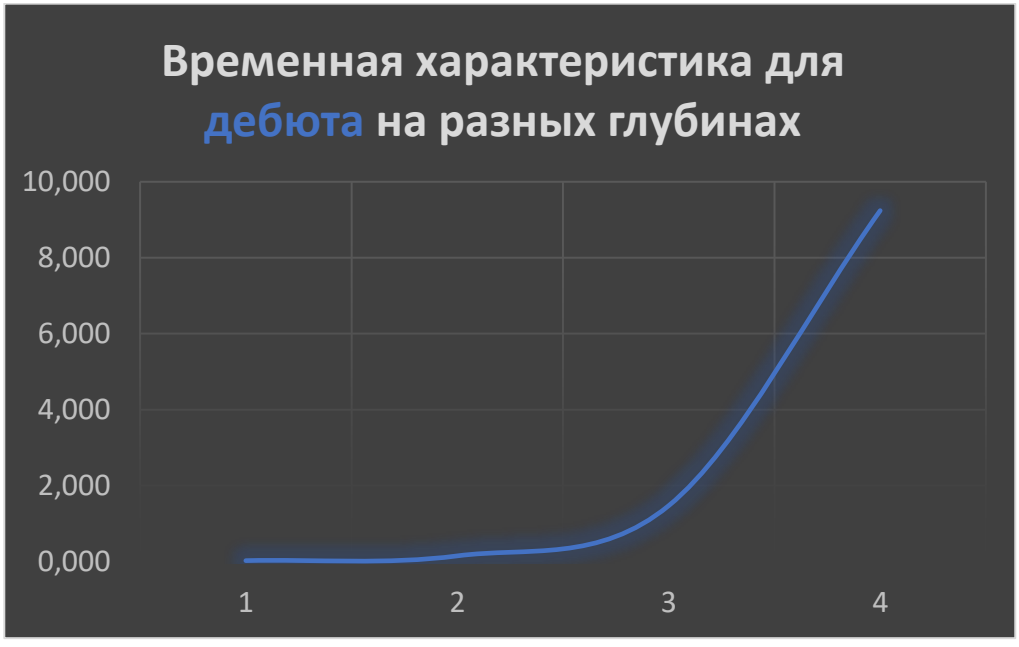

Рис. 22.1. Дебют

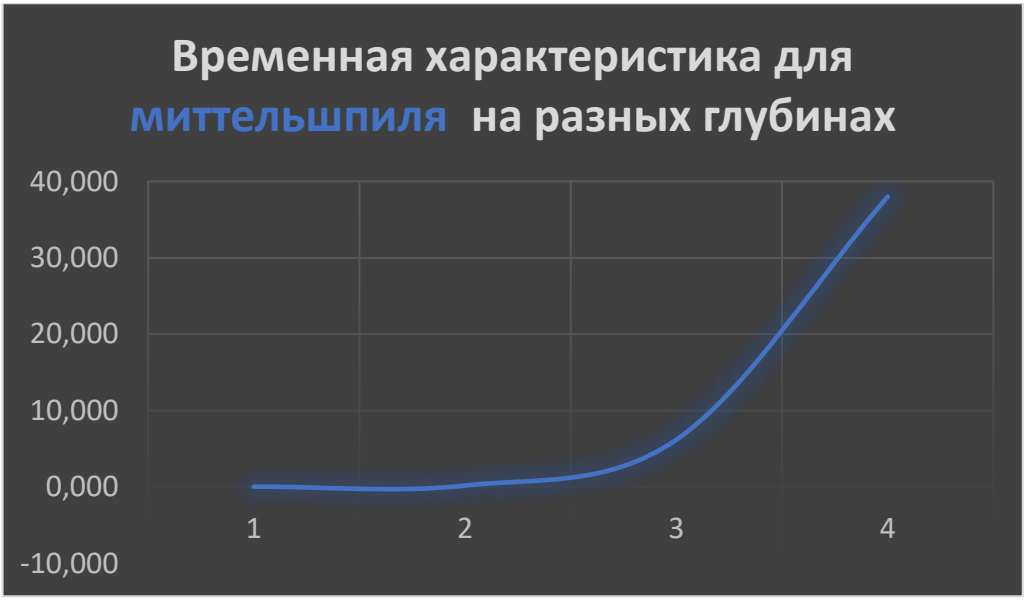

Рис. 22.2. Миттелшпиль

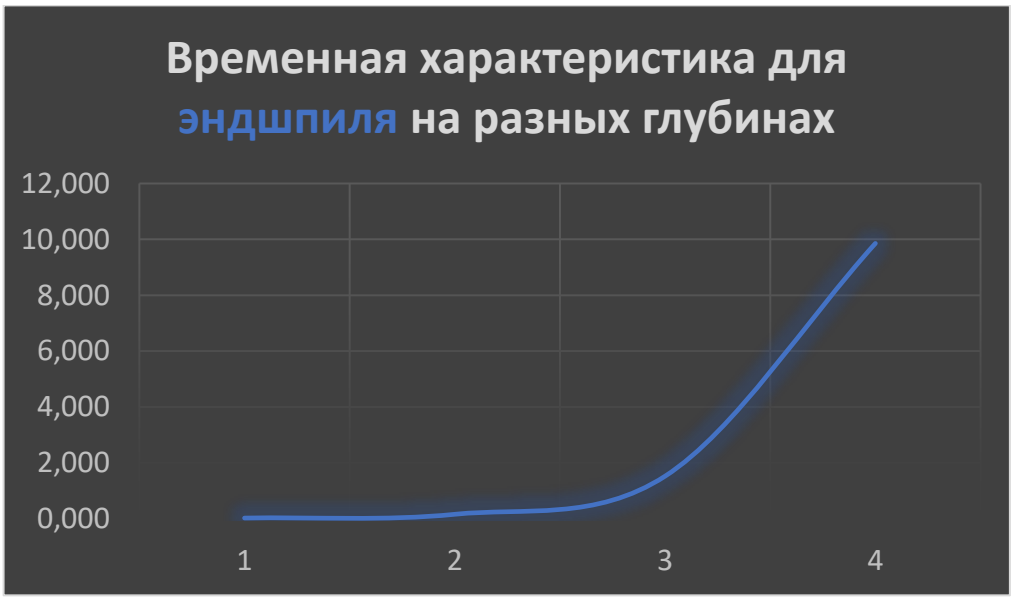

Рис. 22.3. Эндшпиль

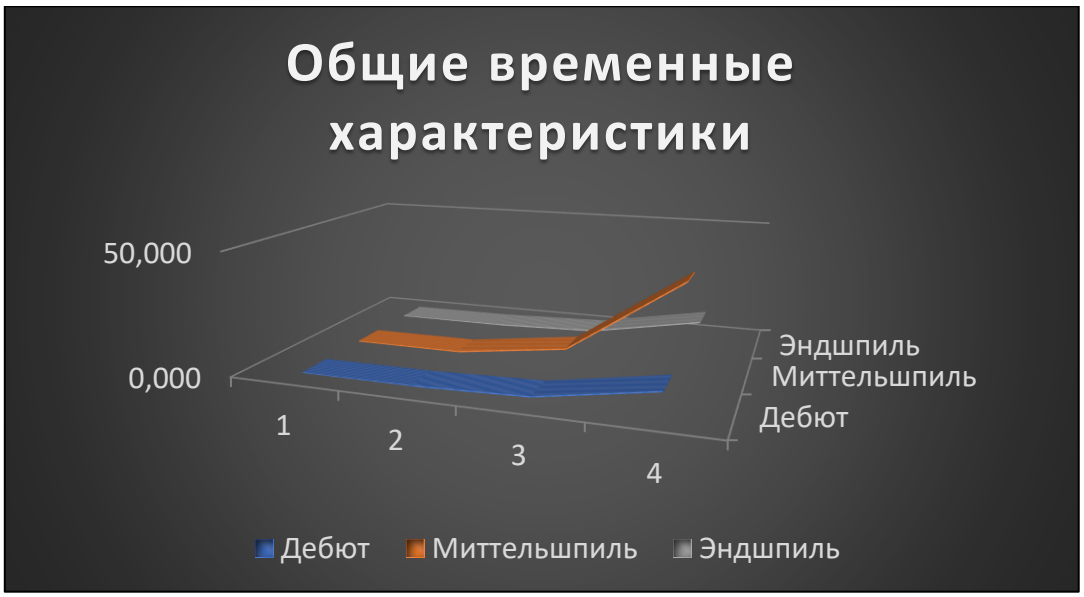

Рис. 22.4. Общие временные характеристики

23.ВЫВОДЫ ПО ПРОВЕДЕННОЙ РАБОТЕ

В результате работы нами была разработана система для игры в шахматы. Она способна на определенном уровне предоставить пользователю игровой опыт.