МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Измерительно-вычислительные комплексы»

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | |  | |  |  |
| |  |  | | --- | --- | | ***Подп. и дата*** |  | | ***Инв. № дубл.*** |  | | ***Взам. инв. №*** |  | | ***Подп. и дата*** |  | | ***Инв. № подл*** |  | |  | | **ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**  на курсовую работу  по дисциплине «Алгоритмы и структуры данных»  **Тема** «Компьютерная логическая игра «Канадские шашки»»  Р.02069337.23/709-05  Листов 50 | | | | | | |
|  | | |  | | Руководитель разработки:  Кандидат технических наук, доцент кафедры «ИВК»  Шишкин В. В.  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2024г | | | |
|  | | Исполнитель  студент гр. ИСТбд-22  Гончаров Т. Е  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2024 г. | | | |
|  | | 2024 г. | | | | | | |

Содержание

[АННОТАЦИЯ 3](#_Toc185971239)

[ТЕХНИЧЕСКОЕ ЗАДАНИЕ 4](#_Toc185971240)

[ПОЯСНИТЕЛЬНАЯ ЗАПИСКА 10](#_Toc185971246)

[РУКОВОДСТВО ПРОГРАММИСТА 33](#_Toc185971252)

[ЗАКЛЮЧЕНИЕ 49](#_Toc185971258)

[Листинг программы 51](#_Toc185971259)

# АННОТАЦИЯ

Данная курсовая работа посвящена разработке компьютерной игры "Канадские шашки" с использованием языка программирования Python и графической библиотеки tkinter. Канадские шашки представляют собой вариант шашек, играемый на доске 12x12 клеток, что делает игру более сложной и интересной по сравнению с классическими шашками. В работе реализованы все правила канадских шашек, включая обязательные взятия, превращение в дамки и особые правила ходов дамок.

Программа предоставляет полноценный игровой процесс для двух игроков с удобным графическим интерфейсом, анимацией ходов и визуальной подсветкой возможных перемещений. Особое внимание уделено системе подсчета очков, где за взятие простой шашки начисляется одно очко, а за взятие дамки - три очка. Реализована система авторизации пользователей с безопасным хранением учетных данных, что позволяет сохранять статистику игроков.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Измерительно-вычислительные комплексы»

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | |  | |  |  |
| |  |  | | --- | --- | | ***Подп. и дата*** |  | | ***Инв. № дубл.*** |  | | ***Взам. инв. №*** |  | | ***Подп. и дата*** |  | | ***Инв. № подл*** |  | |  | | ТЕХНИЧЕСКОЕ ЗАДАНИЕна курсовую работупо дисциплине «Алгоритмы и структуры данных»Тема «Компьютерная логическая игра «Канадские шашки»»Р.02069337.23/709-05Листов 5 | | | | | | |
|  | | |  | |  | | | |
|  | | Исполнитель  студент гр. ИСТбд-22  Гончаров Т. Е  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2024 г. | | | |
|  | | 2024 г. | | | | | | |

##### Введение

Название приложения «Канадские шашки», условное обозначение «*Canadian Checkers*».

Канадские шашки являются вариантом международных шашек, но имеют существенные отличия в размере игрового поля и количестве игровых фигур. Игра ведется на доске размером 12×12 клеток, между двумя соперниками (игрок против игрока). В начальной позиции у каждого игрока по 30 шашек, расположенных на чёрных клетках первых пяти горизонталей с каждой стороны. Основной целью игры является взятие всех шашек противника или запереть шашки противника.

Правила игры предусматривают, что простые шашки могут ходить только по диагонали вперед на одно поле. Если шашка противника находится на соседней клетке, и за ней имеется свободная клетка, игрок может взять эту шашку. При достижении последней горизонтали обычная шашка имеет два типа ходов. Первый - Обычная шашка превращается в дамку и ход переходит другому игроку. Второй – Обычная шашка имеет возможность продолжить взятие шашки противника, но при этом дамкой не становится. Взятие шашек противника является обязательным, причем при наличии нескольких вариантов взятия игрок обязан выбрать тот, при котором будет взято максимальное количество шашек противника.

Функциональные возможности:

* реализация авторизации и регистрации;
* простая и интуитивно понятная графика игрового поля;
* реализация пошагового игрового процесса;
* возможность выбора направления движения шашек;
* автоматическое удаление взятых шашек с игрового поля;
* автоматическое завершение игры.

##### Основания для разработки

Учебная программа по дисциплине «Алгоритмы и структуры данных» по специальности 09.03.02 «Информационные системы и технологии».

##### Требования к программе или программному изделию

* 1. **Функциональное назначение**

Программа предназначена для реализации настольной игры «Канадские шашки» между двумя игроками на одном компьютере. Основной целью является предоставление игрокам удобного и интуитивно понятного интерфейса для игры, обучения правилам, а также анализа игровых стратегий.

Перечень автоматизируемых процессов:

* автоматизация расстановки и перемещения шашек на игровом поле;
* проверка корректности ходов и соблюдения правил игры;
* автоматическое удаление взятых шашек с доски;
* ведение счета взятых шашек для каждого игрока;
* подсчет очков взятых шашек и определение победителя.

Основная группа пользователей – игроки, желающие сыграть друг с другом на одном устройстве.

* 1. **Требования к функциональным характеристикам**
     1. **Требования к структуре приложения**

Приложение должно быть разработано в виде одного модуля с дополнительными информационными файлами при необходимости.

* + 1. **Требования к составу функций приложения**

Основные функции приложения:

* регистрация/авторизация пользователя;
* отображение навигационного меню пользователя;
* отрисовка игрового поля;
* взаимодействие с пользователем;
* интерактивные прием, проверка правильности и отрисовка хода пользователя;
* проверка окончания игры;
* информирование пользователя об окончании игры и победителе.
  + 1. **Требования к организации информационного обеспечения, входных и выходных данных**

В приложении должен быть реализован графический интерфейс взаимодействия с пользователем. Логин и пароль пользователя должны вводиться с клавиатуры. Логины и пароли зарегистрированных пользователей должны храниться в отдельном файле или базе данных в зашифрованном виде. Пояснительные информационные сообщения для пользователя должны выводиться справа игрового поля по ходу игры.

* 1. **Требования к надежности**

Приложение должно обеспечивать устойчивую работу при любых корректных действиях пользователей.

* 1. **Требования к информационной и программной совместимости**

Операционная система: *Windows* 10/11.

Язык программирования: *Python* (рекомендуется версия 3.8+).

Среда разработки: *PyCharm*.

Библиотеки:

* tkinter *–* используется для создания графического интерфейса пользователя;
* hashlib – используется для хеширования данных,
* enum – используется для определения типов,
* Python Imaging Library(PIL) - используется для работы с изображениями,
* pathlib – используется для работы с путями к изображениям,
* time – используется для создания анимации,
* json – используется для хранения данных пользователей
  1. **Требования к маркировке и упаковке**

Требования к маркировке и упаковке не предъявляются.

* 1. **Требования к транспортированию и хранению**
     1. **Условия транспортирования**

Требования к условиям транспортирования не предъявляются.

* + 1. **Условия хранения**

Обеспечение свободного доступа к проекту в репозитории до окончания срока учебы.

* + 1. **Сроки хранения**

Срок хранения – до окончания срока учебы».

##### Требования к программной документации

«Техническое задание» на реализуемое приложение должно соответствовать ГОСТ 19.201-78 «Техническое задание. Требования к содержанию и оформлению»;

«Пояснительная записка» должна соответствовать ГОСТ 19.404-79 «Пояснительная записка. Требования к содержанию и оформлению»;

«Руководство программиста» должна соответствовать ГОСТ 19.504-79 «Руководство программиста. Требования к содержанию и оформлению»;

Оформление программного кода приложения должно быть в соответствии с ГОСТ 19.401-79 «Текст программы. Требования к содержанию и оформлению».

##### Стадии и этапы разработки

Этап 1: Анализ требований к проекту.

Этап 2: Разработка Технического задания.

Этап 3: Проектирование архитектуры программы.

Этап 4: Реализация программы.

Этап 5: Тестирование и исправление ошибок.

Этап 6: Документирование.

Этап 7: Сдача программного продукта и приемка.

##### Порядок контроля и приемки

Контроль и приемка программного обеспечения осуществляются в соответствии с планом тестирования, который включает проверку следующих функциональных возможностей:

• Корректная работа системы регистрации, авторизации, шифрования, а также процесса игры, включая отображение доски и обычных шашек, их перемещение;

• Проверка победителя по окончания игры

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Измерительно-вычислительные комплексы»

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | |  | |  |  |
| |  |  | | --- | --- | | ***Подп. и дата*** |  | | ***Инв. № дубл.*** |  | | ***Взам. инв. №*** |  | | ***Подп. и дата*** |  | | ***Инв. № подл*** |  | |  | | ПОЯСНИТЕЛЬНАЯ ЗАПИСКАна курсовою работупо дисциплине «Алгоритмы и структуры данных»Тема «Компьютерная логическая игра «Канадские шашки»»Р.02069337.23/709-05Листов 22 | | | | | | |
|  | | |  | |  | | | |
|  | | Исполнитель  студент гр. ИСТбд-22  Гончаров Т. Е  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2024 г. | | | |
|  | | 2024 | | | | | | |

##### Введение

Название приложения «Канадские шашки», условное обозначение «*Canadian Checkers*».

Разработанное приложение должна представлять собой полноценную реализацию игры в канадские шашки с графическим пользовательским интерфейсом. В основе приложения должна лежать система авторизации пользователей, которая обеспечивает безопасный доступ к игре. Данные пользователей должна хранится в JSON-файле с применением хеширования паролей для обеспечения безопасности.

Игровой процесс строится на классических правилах канадских шашек, где игра ведется на доске размером 12x12 клеток. Приложение полностью поддерживает все правила игры, включая обязательное взятие шашек противника, превращение простых шашек в дамки при достижении последней линии, а также специальные правила ходов для дамок (Первый – Обычная шашка превращается в дамку и ход переходит другому игроку. Второй – Обычная шашка имеет возможность продолжить взятие шашки противника, но при этом дамкой не становится). Игровая логика реализует автоматическое определение всех возможных ходов и проверку их корректности.

##### Проектная часть

* 1. **Постановка задачи на разработку приложения**

Разработка приложения выполнялась по следующим задачам:

* + - * Разработка системы авторизации пользователей, которая должна обеспечивать безопасную регистрацию новых игроков и вход существующих пользователей в систему. Система включает в себя хранение учетных данных в JSON-файле с обязательным хешированием паролей для обеспечения безопасности. Также требуется реализовать проверку корректности введенных данных и обработку ошибок при авторизации.
      * Создание базовой игровой логики канадских шашек. Все правила игры, включая движение обычных шашек и дамок, обязательного взятия шашек противника, превращение обычных шашек в дамки при достижении последней линии доски.
      * Разработка следующих алгоритмов: проверка доступности ходов, проверка выполнения ходов, проверка обязательного взятия, авторизации и проверка конца игры.
      * Разработка графического пользовательского интерфейса с использованием библиотеки Tkinter. Интерфейс должен включать адаптивное игровое поле размером 12x12 клеток, которое будет корректно масштабироваться при изменении размера окна.
      * Создание информационной панели для отображения текущего состояния игры, включая очередность хода и количество взятых шашек.
      * Реализация визуальных эффектов и анимации шашек при перемещении, подсветки возможных ходов и выделение выбранной шашки.
      * Разработка системы управления игровым процессом, включающую возможность начать новую игру, сдаться, выйти из приложения.
  1. **Математические методы**

В основе игры лежит ориентированный граф G = (V, E), где V - множество вершин, представляющих позиции на игровом поле, а E - множество ребер, представляющих возможные ходы. Игровое поле моделируется в виде двумерной прямоугольной сетки размером 12×12.

Множество вершин V определяется как:

V = {vi,j | i, j ∈ {0, 1, ..., 11}}, где i - номер строки, j - номер столбца.

Каждой вершине vi,j соответствует состояние si,j из множества состояний S:

S = {empty, white\_checker, black\_checker, white\_queen, black\_queen}

Множество ребер E состоит из упорядоченных пар вершин и определяется правилами ходов:

E = {(vi,j, vk,l) | существует допустимый ход из позиции (i,j) в позицию (k,l)}

Веса ребер w(e) для e ∈ E определяются следующим образом:

w(e) = 1 для простого хода

w(e) = 2 для хода со взятием одной шашки

w(e) > 2 для последовательного взятия нескольких шашек

Направления движения задаются векторами смещения:

D = {(-1,-1), (1,-1), (-1,1), (1,1)}, где каждая пара представляет смещение по координатам (Δi, Δj)

Для простых шашек множество достижимых вершин определяется как:

Reach(vi,j) = {vk,l | |k-i| = 1, |l-j| = 1, направление соответствует цвету шашки}

Для дамок множество достижимых вершин расширяется:

ReachQueen(vi,j) = {vk,l | k-i = ±n, l-j = ±n, n ∈ {1,...,11}, путь свободен}

Функция взятия шашек определяется как:

Capture(vi,j) = {vk,l | существует шашка противника на пути из vi,j в vk,l}

Условие превращения в дамку:

Transform(vi,j) = true, если:

* для белой шашки: j = 0
* для черной шашки: j = 11

Множество возможных ходов для позиции определяется как:

Moves(vi,j) = {

Capture(vi,j), если Capture(vi,j) ≠ ∅

Reach(vi,j), если Capture(vi,j) = ∅

}

Состояние игры в любой момент времени описывается подграфом G' = (V, E'), где E' ⊆ E - множество допустимых ходов в текущей позиции. После каждого хода происходит пересчет множества E' с учетом новых позиций шашек и правил игры.

Победное состояние достигается, когда для одного из игроков:

Win(player) = true, если:

Moves(vi,j) = ∅ для всех шашек противника, или

На поле не осталось шашек противника

**Обоснование выбора модели:**

* Графовая модель является оптимальным выбором для реализации канадских шашек благодаря нескольким ключевым преимуществам:
  + Естественным образом отражает структуру игрового поля и возможные перемещения шашек, где вершины представляют позиции, а ребра - допустимые ходы.
  + Эффективна при реализации сложных игровых механик, таких как взятие нескольких шашек за один ход, где последовательность взятий может быть представлена как путь в графе. Динамическая природа графа позволяет легко обновлять состояние игры после каждого хода, пересчитывая множество допустимых ребер.
  + Упрощает поиск возможных ходов и проверку правил игры. Алгоритмы обхода графа позволяют эффективно находить все доступные перемещения для каждой шашки.
  1. **Архитектура и алгоритмы**
     1. **Архитектура**

Приложение построено с использованием архитектуры на основе объектно-ориентированного программирования (ООП), где основным объектом является игровое поле. Архитектура приложения представлена на рисунке 1.

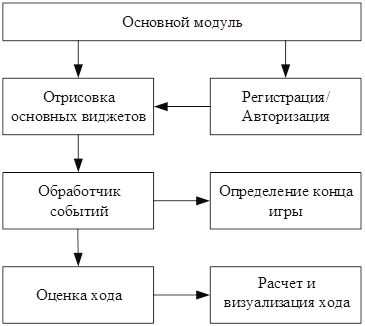


Рисунок 1. Архитектура приложения «Канадские шашки»

Основные функции приложения:

* handle\_move — обрабатывает ход игрока, проверяет его корректность, выполняет перемещение шашки и обновляет состояние игры. После выполнения хода проверяет возможность дополнительных взятий;
* get\_required\_moves\_list — анализирует текущую позицию и возвращает список всех обязательных ходов со взятием для текущего игрока. Проверяет возможности взятия как для обычных шашек, так и для дамок;
* get\_optional\_moves\_list — формирует список всех возможных ходов без взятия для текущего игрока в текущей позиции. Учитывает различные правила движения для простых шашек и дамок;
* animate\_move — выполняет анимацию перемещения шашки от начальной до конечной позиции. Обеспечивает плавное визуальное отображение хода;
* check\_for\_game\_over — проверяет условия завершения игры: отсутствие возможных ходов у игрока или полное уничтожение шашек одной из сторон. При выполнении условий объявляет победителя;
* draw\_possible\_moves — отображает на игровом поле все возможные ходы для выбранной шашки, включая как обычные перемещения, так и ходы со взятием. Помогает игроку визуально оценить доступные варианты;
* handle\_player\_turn — управляет очередностью ходов, переключает активного игрока после завершения хода, проверяет наличие дополнительных ходов при множественном взятии.
  + 1. **Алгоритм «Проверка доступности ходов»**

В первую очередь алгоритм проверяет, находится ли предполагаемый ход в пределах игрового поля. Если ход выходит за границы поля, он считается недопустимым.

Для обычной шашки существует строгое ограничение - она может двигаться только на одну клетку вперед по диагонали. Любое движение назад для обычной шашки запрещено.

Дамка имеет более широкие возможности для передвижения - она может ходить по диагонали как вперед, так и назад на любое количество свободных клеток.

Последним этапом проверки является контроль свободного пространства. Алгоритм проверяет, не занята ли целевая клетка другой шашкой. Ход возможен только на свободную клетку.

**Обоснование алгоритма:**

Первоначальная проверка нахождения хода в пределах доски является критически важной для предотвращения выхода за границы игрового поля. Это обеспечивает базовую безопасность игры.

Проверка свободной клетки необходима, поскольку правила игры запрещают ставить шашку на занятую позицию. Это фундаментальное правило шашечной игры.

Разделение логики для обычной шашки и дамки обусловлено различными правилами их передвижения. Обычная шашка имеет право ходить только вперёд на одну клетку по диагонали.

Проверка направления движения для обычной шашки отражает классические правила игры в шашки, где простая шашка не может двигаться назад.

Контроль диагонального движения является обязательным, так как это единственный допустимый способ перемещения шашек в игре.

На рисунке 2 представлен алгоритм проверки доступности ходов.

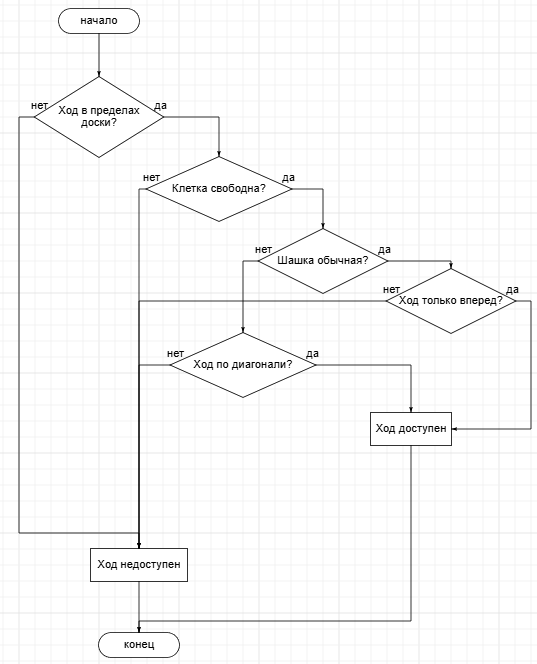


Рисунок 2. Алгоритм «Проверка доступности ходов»

* + 1. **Алгоритм «Выполнения хода»**

На начальном этапе алгоритм проверяет возможность выполнения выбранного хода согласно правилам игры.

При обнаружении возможности взятия шашки противника происходит её удаление с игрового поля.

После этого осуществляется перемещение активной шашки на новую позицию.

В случае достижения шашкой последней горизонтали происходит её автоматическое превращение в дамку.

Завершающим этапом алгоритма является передача права хода следующему игроку.

**Обоснование алгоритма:**

Начальная проверка доступности хода предотвращает выполнение некорректных действий и обеспечивает соблюдение правил игры.

Проверка наличия взятия и его обработка реализует одно из ключевых правил шашек - обязательное взятие шашки противника при наличии такой возможности.

Контроль достижения края доски необходим для реализации правила превращения обычной шашки в дамку, что существенно влияет на дальнейшую стратегию игры.

Передача хода другому игроку обеспечивает поочередность ходов, что является базовым принципом игры в шашки.

На рисунке 3 представлен алгоритм выполнения хода

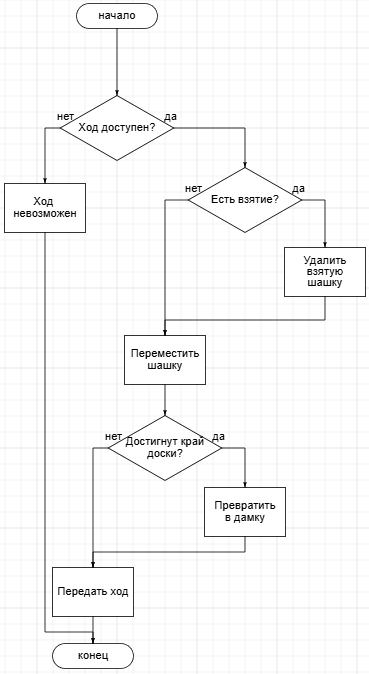


Рисунок 3. Алгоритм «Выполнения хода»

* + 1. **Алгоритм «Проверки обязательного взятия»**

Алгоритм начинает работу с последовательной проверки всех шашек, принадлежащих текущему игроку.

Для каждой шашки производится анализ возможности взятия шашки противника. При обнаружении хотя бы одной возможности взятия, данное действие становится обязательным для выполнения. В случае отсутствия возможности взятия игрок получает право выполнить обычный ход любой своей шашкой.

**Обоснование алгоритма:**

Полная проверка всех шашек игрока необходима для выявления всех возможных вариантов взятия, что обеспечивает соблюдение правила об обязательном взятии.

Определение наличия возможности взятия влияет на дальнейшие допустимые ходы игрока, ограничивая их только ходами со взятием.

Четкое разделение ситуаций с обязательным и необязательным взятием помогает игрокам понимать их текущие возможности и обязанности.

На рисунке 4 представлен алгоритм авторизации пользователя



Рисунок 4. Алгоритм «Проверки обязательного взятия»

* + 1. **Алгоритм «Авторизации»**

Работа алгоритма начинается с получения от пользователя его учетных данных - логина и пароля. Введенный пароль подвергается процедуре хеширования для обеспечения безопасного сравнения.

Система проверяет наличие указанного пользователя в базе данных. При обнаружении пользователя происходит сверка хешированного пароля с сохраненным значением. В случае успешной проверки всех данных осуществляется авторизация пользователя в системе.

При возникновении любых несоответствий пользователю выводится соответствующее сообщение об ошибке.

**Обоснование алгоритма:**

Хеширование пароля обеспечивает безопасность хранения пользовательских данных.

Двухэтапная проверка (существование пользователя и корректность пароля) позволяет предоставить более точную информацию об ошибке.

Разделение сообщений об ошибках помогает пользователю понять причину неудачной авторизации.

Успешная авторизация предоставляет доступ к функционалу игры.

На рисунке 5 представлен алгоритм авторизации пользователя

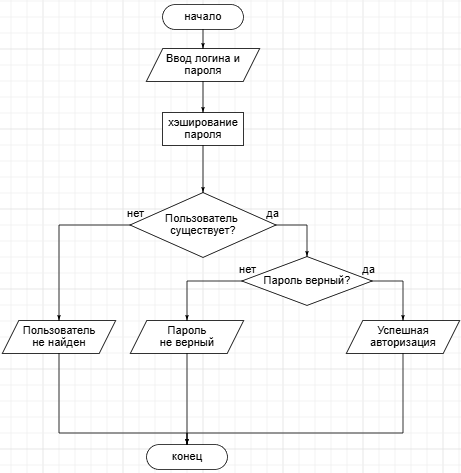


Рисунок 5. Алгоритм «Авторизация»

* + 1. **Алгоритм «Проверки конца игры»**

Алгоритм начинает работу с получения всех возможных ходов для первого игрока, который играет за белых на текущей позиции. При отсутствии возможных ходов у первого игрока фиксируется победа второго игрока, играющего за чёрных, и игра заканчивается.

В случае наличия ходов у первого игрока, алгоритм проверяет наличие возможных ходов у второго игрока. Если у второго игрока нет возможных ходов, фиксируется победа первого игрока и игра заканчивается. При наличии ходов у обоих игроков игра продолжается.

**Обоснование алгоритма:**

Проверка возможных ходов только для текущего игрока достаточна для определения конца игры, так как отсутствие ходов означает поражение.

Последовательная проверка сначала первого игрока, затем второго игрока оптимизирует процесс определения конца игры.

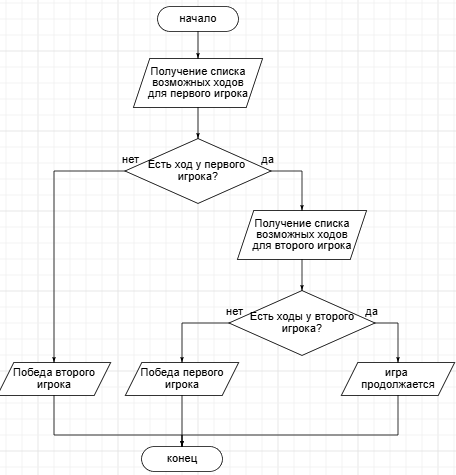


Рисунок 5. Алгоритм «Проверки конца игры»

**1.4. Тестирование**

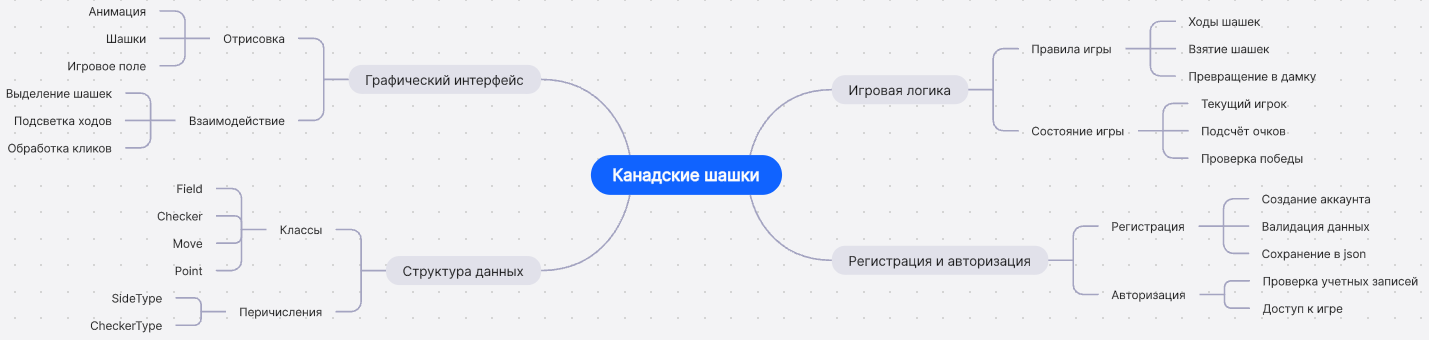


Рисунок 5. Mind Map

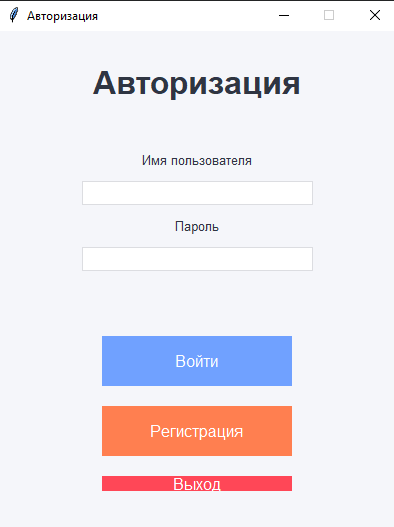


Рисунок 6. Окно «Авторизация»

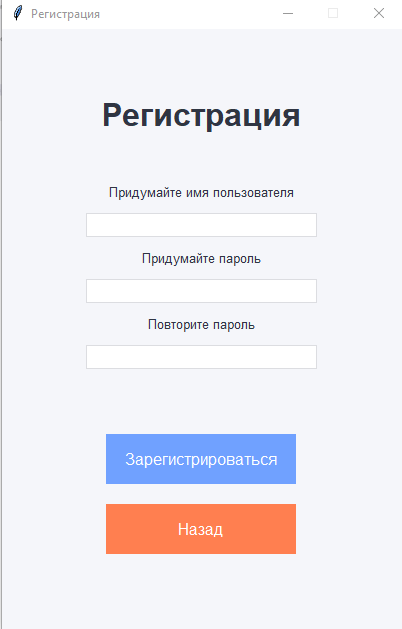


Рисунок 7. Окно «Регистрация»

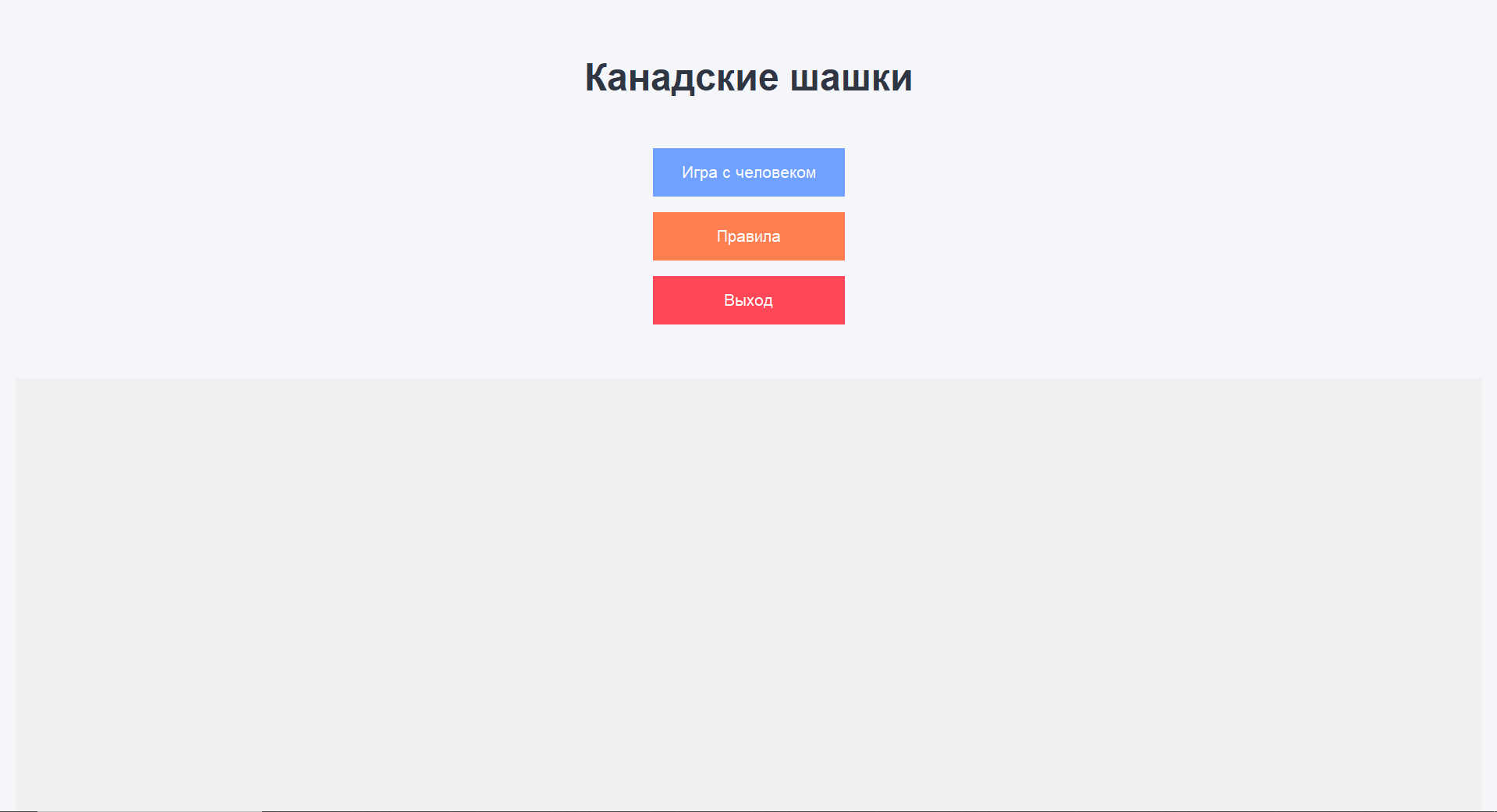


Рисунок 8. Окно «Навигационное меню»

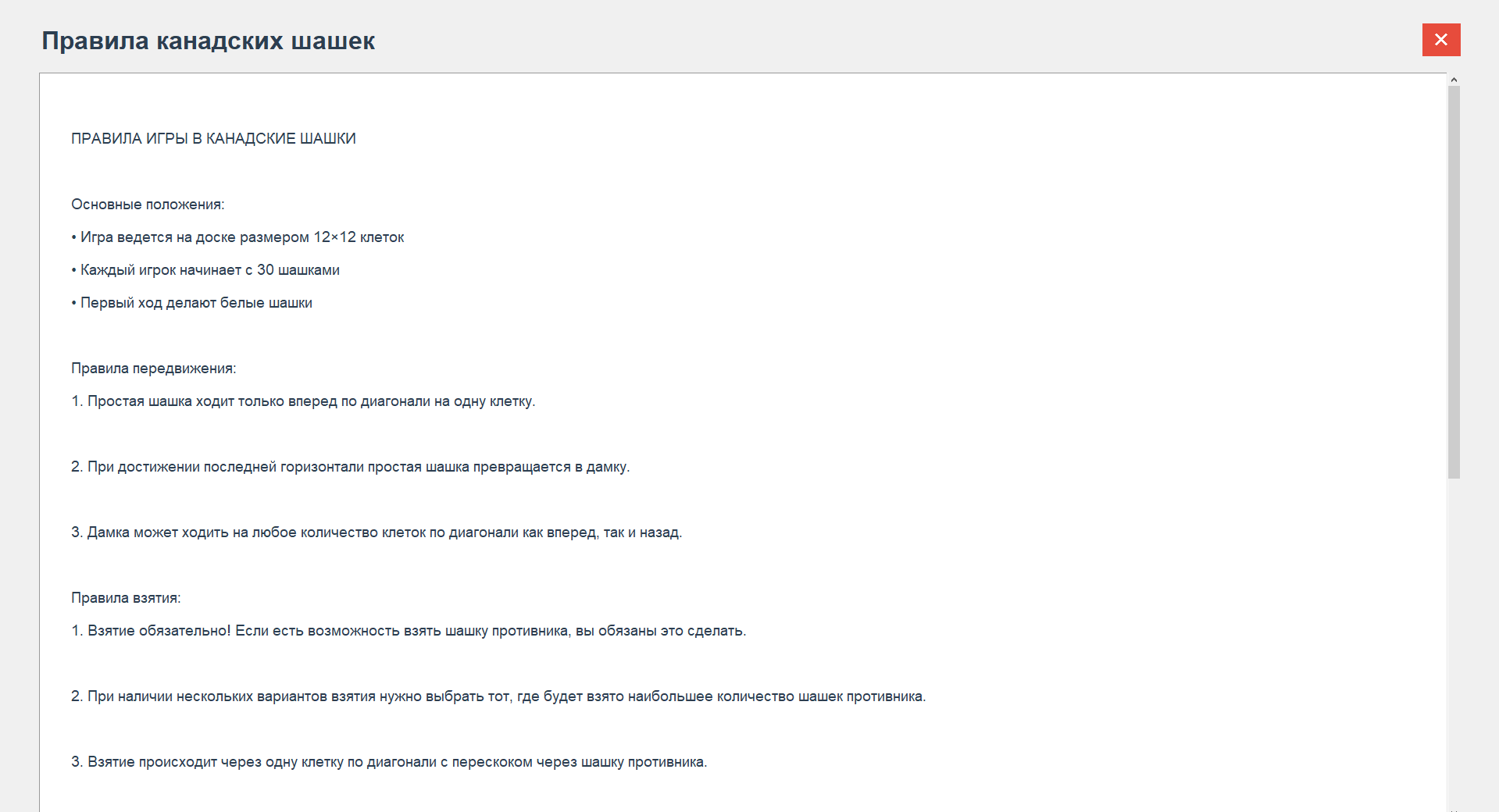


Рисунок 9. Окно «Правила канадских шашек»

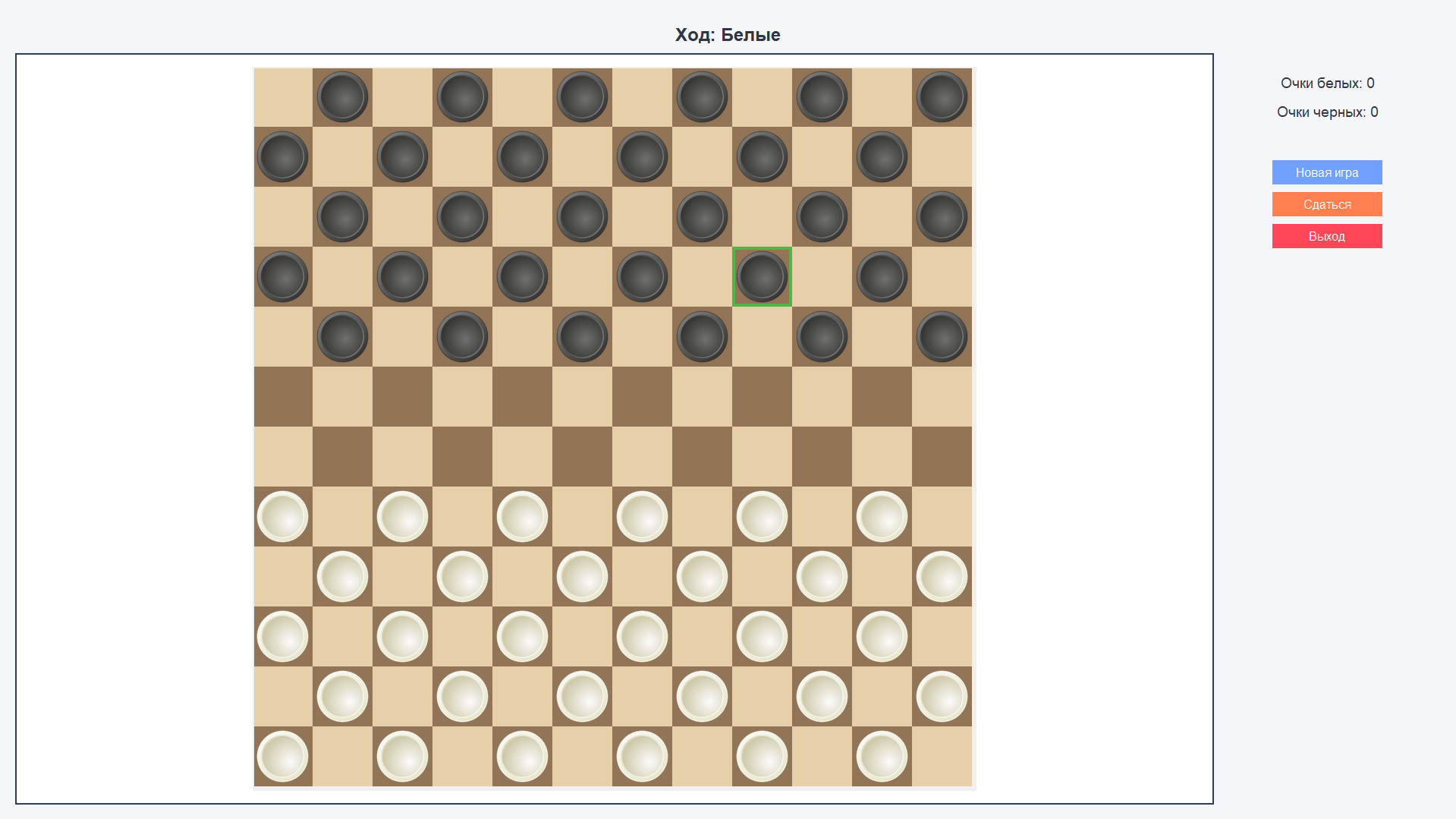


Рисунок 10. Окно «Игровое поле»

**1.4.1. Описание отчета о тестировании**

Отчет о тестировании приложения "Канадские шашки" представлены результаты тестирования документации и программного кода. Описаны проведенные тесты, их результаты и обнаруженные дефекты.

**1.4.2. Цель тестирования**

Целью тестирования является проверка соответствия ПО предъявляемым требованиям, а также выявление возможных багов. По результатам тестирования следует исправление выявленных багов.

**1.4.3. Методика тестирования**

Для тестирования приложения "Канадские шашки" применялись следующие методы:

* Статистическое тестирование: анализ и проверка кода без его запуска, выявление ошибок в технической документации.
* Ручное тестирование: запуск пользовательских сценариев программы с различными входными данными и проверка корректности полученных результатов.
* Автоматизированное тестирование: написание и запуск набора unit-тестов отдельных модулей программы.

**1.4.4. Проведенные тесты**

В ходе тестирования были проведены следующие тесты:

* Статистическое тестирование:

Количество обнаруженных ошибок в документации: 0

Количество обнаруженных и исправленных ошибок в программном коде: 0

* Ручное тестирование:

Написаны и проведены следующие тест-кейсы:

1. ТК1. Проверка хода дамкой.

Предварительные шаги:

* + Авторизоваться в системе.
  + Иметь дамку на поле.

Шаги:

* + 1. Выбрать дамку.
    2. Сделать ход по диагонали на несколько клеток.

Ожидаемый результат:

* + - Дамка переместится на выбранную позицию.
    - Ход перейдет к противнику.

Фактический результат:

* + Дамка переместилась на выбранную позицию.
  + Ход перешел к противнику.

1. ТК2. Проверка хода обычной шашкой

Предварительные шаги:

* + Авторизоваться в системе.
  + Начать новую игру.

Шаги:

* + 1. Запустить приложение.
    2. Авторизоваться в системе.
    3. Начать новую игру.
    4. Выбрать белую шашку.
    5. Сделать ход по диагонали вперед на одну клетку.
    6. Нажать на клетку для хода.

Ожидаемый результат:

* + - Шашка переместится на выбранную позицию.
    - Ход перейдет к черным.

Фактический результат:

* + Шашка переместилась на выбранную позицию.
  + Ход перешел к черным.

1. ТК3. Отработка загрузки игрового поля.

Шаги:

* + 1. Запустить приложение.
    2. В окне авторизации, в поле «логин» ввести игрок1, а в поле «пароль» ввести 123456.
    3. Нажать на кнопку «Войти».
    4. Нажать на кнопку «Игра с человеком»

Ожидаемый результат:

* + - Пользователь начнёт играть.

Фактический результат:

* Пользователь начал играть.

1. ТК4. Проверка взятия шашки противника.

Предварительные шаги:

* + - Авторизоваться в системе.
    - Расположить шашки для возможности взятия.

Шаги:

1. Выбрать белую шашку.
2. Выполнить ход через шашку противника.
3. Нажать на клетку для хода.

Ожидаемый результат:

* Шашка противника будет удалена.
* Шашка переместится через позицию взятой шашки.
* Игроку начислится 1 очко.

Фактический результат:

* Шашка противника удалена.
* Шашка переместилась через позицию взятой шашки.
* Игроку начислено 1 очко.

1. ТК5. Проверка превращения в дамку.

Предварительные шаги:

* + - Авторизоваться в системе.
    - Довести шашку до последней горизонтали.

Шаги:

1. Выбрать белую шашку на предпоследней горизонтали.
2. Сделать ход на последнюю горизонталь.
3. Нажать на клетку для хода.

Ожидаемый результат:

* Шашка превратится в дамку.
* Ход перейдет к противнику.

Фактический результат:

* Шашка превратилась в дамку.
* Ход перешел к противнику.

Таблица 1

|  |  |  |
| --- | --- | --- |
| Чек-лист для проверки ходов | | |
| Описание | Действие | Результат |
| Ход обычной шашкой | Ход по диагонали вперед на 1 клетку | Успешное перемещение |
| Ход обычной шашкой назад | Попытка хода назад | Ход не выполняется |
| Ход дамкой | Ход по диагонали на несколько клеток | Успешное перемещение |
| Обязательное взятие | Попытка обычного хода при возможности взятия | Ход не выполняется |
| Проверка превращения в дамку | Превращение в дамку | Успешное превращение |

* Автоматизированное тестирование

Для курсовой работы было составлено 8 unit-тестов.

Тесты покрывают следующие функции приложения:

1. Тестирование функции авторизации check\_user(username: str, password: str):
   * Проверка успешной авторизации с корректными данными
   * Проверка отказа в авторизации при неверном пароле
   * Проверка отказа в авторизации при несуществующем пользователе
2. Тестирование функции регистрации register\_user(username: str, password: str):

* Проверка успешной регистрации нового пользователя
* Проверка отказа в регистрации при существующем логине
* Проверка отказа в регистрации при некорректной длине логина
* Проверка отказа в регистрации при некорректной длине пароля

1. Тестирование функции проверки координат is\_within(x: int, y: int):

* Проверка корректных координат внутри поля (0,0)
* Проверка корректных координат внутри поля (11,11)
* Проверка некорректных отрицательных координат (-1,0)
* Проверка некорректных координат за пределами поля (12,0)

1. Тестирование функции определения типа шашки type\_at(x: int, y: int):

* Проверка пустой клетки
* Проверка наличия обычной белой шашки
* Проверка наличия обычной черной шашки
* Проверка наличия белой дамки
* Проверка наличия черной дамки

1. Тестирование функции изменения типа шашки change\_type(type: CheckerType):

* Проверка превращения обычной белой шашки в дамку
* Проверка превращения обычной черной шашки в дамку

1. Тестирование функций подсчета шашек white\_checkers\_count() и black\_checkers\_count():

* Проверка начального количества белых шашек (20)
* Проверка начального количества черных шашек (20)
* Проверка количества шашек после взятия
* Проверка количества шашек в конце игры

1. Тестирование функций подсчета очков white\_score() и black\_score():

* Проверка начального счета (по 20 очков)
* Проверка начисления очков за взятие обычной шашки (+1)
* Проверка начисления очков за взятие дамки (+3)
* Проверка общего счета после серии взятий

1. Тестирование функции анимации хода animate\_move(move: Move):

* Проверка корректности анимации обычного хода
* Проверка корректности анимации взятия шашки
* Проверка корректности анимации хода дамкой
* Проверка корректности анимации множественного взятия

Все unit-тесты прошли успешную проверку. Результаты тестирования показали корректную работу основных функций приложения, включая авторизацию, регистрацию, проверку ходов, подсчет очков и анимацию перемещения шашек.

**1.4.5. Выводы**

На основе проведенных тестов сделаны следующие выводы:

– Программа успешно прошла все тесты и работает корректно.

– Не обнаружены дефекты.

– Рекомендации по дальнейшему развитию программы: добавление

ограничения времени на ход пользователя, звукового сопровождения,

таблицы лидеров, сохранение и загрузка статистики, создания второго режима против компьютера.

##### Источники, использованные при разработке

* 1. Вирт Н. Алгоритмы + структуры данных = программы: Пер. с англ. / Н. Вирт. – Москва : Мир, 1985. – 406 с.
  2. Архитектура программного обеспечения [Электронный ресурс]. – Режим доступа: URL: [https://dic.academic.ru/dic.nsf/ruwiki/14691](https://dic.academic.ru/dic.nsf/ruwiki/146913)3 (дата обращения : 01.12.2024)
  3. Назина, О.Е. Что такое тестирование. Курс молодого бойца / О.Е. Назина.  Санкт-Петербург : БХВ-Петербург, 2022.  592 с.: ил.
  4. ГОСТ 19.701-90 «Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения». – М.: Издательство стандартов, 1990.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Измерительно-вычислительные комплексы»

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | |  | |  |  |
| |  |  | | --- | --- | | ***Подп. и дата*** |  | | ***Инв. № дубл.*** |  | | ***Взам. инв. №*** |  | | ***Подп. и дата*** |  | | ***Инв. № подл*** |  | |  | | РУКОВОДСТВО ПРОГРАММИСТАна курсовую работупо дисциплине «Алгоритмы и структуры данных»Тема «Компьютерная логическая игра «Канадские шашки»»Р.02069337.23/709-05Листов 15 | | | | | | |
|  | | |  | |  | | | |
|  | | Исполнитель  студент гр. ИСТбд-22  Гончаров Т. Е  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2024 г. | | | |
|  | | 2024 | | | | | | |

**1. Назначение и условия применения программы**

**1.1. Назначение и функции, выполняемые приложением**

**Назначение:**

Приложение "Канадские шашки" представляет собой полноценную реализацию классической настольной игры с графическим пользовательским интерфейсом. Игра разработана для двух игроков, играющих на одном компьютере.

**Основные функциональные возможности:**

1. Авторизация и регистрация пользователей
2. Игра двух игроков на одном компьютере
3. Подсчет очков за взятые шашки
4. Отображение правил игры
5. Возможность сдаться и начать новую игру
6. Анимация перемещения шашек
7. Подсветка возможных ходов

**1.2. Условия, необходимые для использования приложения**

* **Операционная система:** Windows, macOS, Linux.
* **Язык программирования:** Python.
* **Инструментальная среда:** Любая среда разработки Python (например, PyCharm, Visual Studio Code).
* **Библиотеки:**
  + - tkinter - для создания графического интерфейса
    - PIL/Pillow - для работы с изображениями шашек
    - json - для хранения данных пользователей
    - hashlib - для шифрования паролей
    - pathlib - для работы с путями файлов
    - time - для управления анимацией
    - enum - для определения типов

**2. Характеристики программы**

**2.1. Характеристики приложения**

Программный код содержит около 900 значимых строк, выполняющих конкретные действия. В программе реализовано 8 основных классов и 2 перечисления (Enum).

**Основные структуры данных включают:**

* Двумерный массив для хранения состояния игрового поля
* Классы Point и Move для работы с координатами и ходами
* JSON-структуру для хранения данных пользователей
* Перечисления для типов шашек и сторон игроков

**В приложении используются следующие библиотеки:**

* tkinter - для создания графического интерфейса
* PIL/Pillow - для работы с изображениями шашек
* enum - для создания перечислений
* json - для работы с данными пользователей
* hashlib - для хеширования паролей
* pathlib - для работы с путями к файлам
* time - для управления анимацией

Работа приложения начинается с окна авторизации, где пользователь может войти в существующий аккаунт или зарегистрировать новый.

После успешной авторизации открывается главное меню с тремя кнопками: "Игра с человеком", "Правила" и "Выход".

При выборе игры открывается игровое поле размером 12x12 клеток с расставленными шашками. Справа от поля располагается панель с информацией о текущем счете и кнопками управления игрой.

**Для контроля корректности ввода/вывода применяются следующие средства:**

1. При авторизации:

* Проверка заполнения всех полей
* Валидация длины имени пользователя (минимум 3 символа)
* Проверка длины пароля (минимум 6 символов)
* Контроль совпадения паролей при регистрации

1. В игровом процессе:

* Проверка возможности хода
* Контроль выхода за границы поля
* Валидация очередности ходов
* Проверка обязательных взятий шашек

1. При работе с файлами:

* Проверка существования файла с данными пользователей
* Валидация формата JSON при чтении/записи
* Контроль доступа к файлам изображений

**2.2. Особенности реализации приложения**

**В программе используются следующие основные структуры данных:**

1. Игровое поле реализовано как двумерный массив объектов класса Checker:

Данная структура была выбрана вместо альтернативных вариантов (например, одномерного массива или словаря) по следующим причинам:

* Обеспечивает прямой доступ к любой клетке поля по координатам за O(1)
* Наглядно отражает двумерную природу шашечной доски
* Упрощает алгоритмы проверки возможных ходов и взятий
* Позволяет легко итерировать по строкам и столбцам доски

1. Для представления координат создан класс Point:

Вместо использования кортежей (x, y) был выбран класс, так как он:

* Обеспечивает инкапсуляцию данных
* Предоставляет удобные методы сравнения точек
* Делает код более читаемым благодаря именованным атрибутам
* Позволяет легко расширять функциональность при необходимости

1. Ходы представлены классом Move:

Данная реализация предпочтительнее простого хранения координат, поскольку:

* Объединяет начальную и конечную позиции в единую сущность
* Упрощает проверку валидности хода
* Облегчает анимацию перемещения шашек
* Позволяет добавлять дополнительные свойства хода

1. Для хранения данных пользователей используется JSON-формат:

Этот формат был выбран вместо реляционной базы данных или простого текстового файла, так как он:

* Обеспечивает удобное хранение пар логин-пароль
* Позволяет легко читать и модифицировать данные
* Не требует установки дополнительного ПО
* Поддерживает человекочитаемый формат
* Типы шашек и сторон реализованы через перечисления Enum:
* Использование Enum вместо простых констант обосновано тем, что оно:
* Предотвращает ошибки при сравнении значений
* Обеспечивает типобезопасность
* Делает код более понятным
* Упрощает отладку

1. Для анимации перемещения шашек используется система тегов Canvas:

Такой подход был выбран вместо прямого обновления координат, поскольку он:

* Обеспечивает плавную анимацию
* Позволяет легко управлять слоями отрисовки
* Упрощает обновление состояния доски
* Оптимизирует перерисовку элементов

Все программные решения являются оригинальными и разработаны специально для данного проекта с учетом специфики канадских шашек и требований к пользовательскому интерфейсу**.**

**3. Обращение к программе**

1. **Класс SideType**
   1. Метод opposite

Метод opposite является статическим методом класса SideType, который принимает один параметр side типа SideType. Основная задача метода - вернуть противоположную сторону для переданного значения. Если передана белая сторона (WHITE), метод вернет черную сторону (BLACK), и наоборот. В случае если передано некорректное значение, метод вернет None. Этот метод активно используется в игровой логике для определения следующего хода и при проверке вражеских шашек.

1. **Класс Point**
   1. Метод \_\_init\_\_

Конструктор класса Point принимает два необязательных параметра: x и y типа int, которые по умолчанию имеют значение -1. Эти параметры представляют координаты точки на игровом поле. При создании объекта координаты сохраняются в защищенных атрибутах x и \_y.

* 1. Свойство x

Свойство x является декорированным методом, который предоставляет доступ только для чтения к приватному атрибуту x. Это свойство позволяет получить значение координаты x точки, при этом защищая её от прямого изменения извне.

* 1. Свойство y

Свойство y является декорированным методом, который предоставляет доступ только для чтения к приватному атрибуту y. Это свойство позволяет получить значение координаты y точки, при этом защищая её от прямого изменения извне.

* 1. Метод \_\_eq\_\_:

Метод eq реализует операцию сравнения двух точек на равенство. Он принимает параметр other и проверяет, является ли он объектом класса Point. Если это так, метод сравнивает координаты x и y обеих точек и возвращает True, если они совпадают. В противном случае возвращает NotImplemented, позволяя Python использовать альтернативные методы сравнения.

1. **Класс Move**
2. Метод \_\_init\_\_

Конструктор класса Move принимает четыре необязательных параметра: from\_x, from\_y, to\_x и to\_y типа int, которые по умолчанию имеют значение -1. Эти параметры представляют начальные (откуда) и конечные (куда) координаты хода. При создании объекта все координаты сохраняются в защищенных атрибутах from\_x, \_from\_y, \_to\_x и \_to\_y.

1. Свойство from\_x

Свойство from\_x является декорированным методом, который предоставляет доступ только для чтения к защищенному атрибуту from\_x. Это свойство позволяет получить значение начальной координаты x хода.

1. Свойство from\_y

Свойство from\_y является декорированным методом, который предоставляет доступ только для чтения к защищенному атрибуту from\_y. Это свойство позволяет получить значение начальной координаты y хода.

1. Свойство to\_x

Свойство to\_x является декорированным методом, который предоставляет доступ только для чтения к защищенному атрибуту to\_x. Это свойство позволяет получить значение конечной координаты x хода.

1. Свойство to\_y

Свойство to\_y является декорированным методом, который предоставляет доступ только для чтения к защищенному атрибуту to\_y. Это свойство позволяет получить значение конечной координаты y хода.

1. Метод \_\_str\_\_

Метод str возвращает строковое представление хода в формате "from\_x-from\_y -> to\_x-to\_y". Этот метод используется для удобного отображения информации о ходе в понятном для человека виде.

1. Метод \_\_repr\_\_

Метод repr возвращает то же строковое представление, что и str. Он используется для отладки и более технического представления объекта хода.

1. Метод \_\_eq\_\_

Метод eq реализует операцию сравнения двух ходов на равенство. Он принимает параметр other и проверяет, является ли он объектом класса Move. Если это так, метод сравнивает все координаты обоих ходов и возвращает True, если они полностью совпадают. В противном случае возвращает NotImplemented, позволяя Python использовать альтернативные методы сравнения.

1. **Класс Game**
2. Метод \_\_init\_\_:

Метод инициализирует новую игру. Принимает холст для отрисовки и размеры поля. Создает игровое поле заданного размера, устанавливает белых как начинающего игрока, инициализирует координаты выбранной и наведенной клеток значениями по умолчанию. Устанавливает начальный счет игры в 0 для обеих сторон. Вызывает метод загрузки изображений.

1. Метод init\_images:

Загружает изображения всех типов шашек из директории assets. Для каждого типа шашек (белая обычная, черная обычная, белая дамка, черная дамка) создает объект ImageTk.PhotoImage, масштабируя изображение до размера клетки поля. Сохраняет все изображения в словаре self.images для быстрого доступа при отрисовке.

1. Метод animate\_move:

Создает плавную анимацию перемещения шашки. Устанавливает флаг анимации, сохраняет координаты анимируемой шашки. Создает временное изображение шашки для анимации. Вычисляет векторы движения по x и y. Пошагово перемещает шашку от начальной до конечной позиции с заданной скоростью, используя временные задержки для плавности.

1. Метод draw:

Полностью перерисовывает игровое поле. Очищает холст от всех элементов. Последовательно вызывает методы отрисовки сетки поля и шашек. Метод вызывается при любом изменении состояния игры.

1. Метод draw\_field\_grid:

Отрисовывает клетки игрового поля в шахматном порядке, используя два цвета из констант. Добавляет рамки для выбранной клетки (красного цвета) и наведенной клетки (зеленого цвета). Если есть выбранная клетка, вызывает метод отрисовки возможных ходов.

1. Метод draw\_possible\_moves:

Удаляет предыдущие метки возможных ходов. Получает список возможных ходов для текущего игрока и противника. Отрисовывает круги на клетках, куда может походить выбранная шашка. Использует разные цвета для обязательных и необязательных ходов.

1. Метод draw\_checkers:

Проходит по всем клеткам поля. Для каждой непустой клетки, не содержащей анимируемую шашку, создает изображение соответствующей шашки на холсте, используя сохраненные в словаре изображения.

1. Метод mouse\_move:

Вычисляет координаты клетки под курсором. Если координаты изменились, обновляет позицию наведенной клетки и перерисовывает поле для отображения подсветки.

1. Метод mouse\_down:

Сложный метод, обрабатывающий выбор шашки и выполнение хода. Проверяет валидность координат и отсутствие анимации. Определяет шашки текущего игрока. Проверяет наличие обязательных ходов. Обрабатывает выбор новой шашки или выполнение хода выбранной шашкой.

1. Метод handle\_move:

Основной метод обработки хода. Если включена отрисовка, запускает анимацию перемещения. Изменяет позицию шашки на поле, перемещая её с начальной позиции на конечную. Вычисляет направление движения для определения съеденных шашек. Проверяет наличие шашек противника на пути движения и удаляет их, начисляя очки (1 за простую шашку, 3 за дамку). Возвращает флаг, указывающий было ли взятие.

1. Метод handle\_player\_turn:

Выполняет ход и обрабатывает его последствия. Проверяет, была ли взята шашка противника. Проверяет достижение последней линии для превращения в дамку. Анализирует наличие дополнительных обязательных ходов для той же шашки. Если дополнительных ходов нет, переключает ход на другого игрока и проверяет условия окончания игры.

1. Метод get\_required\_moves\_list\_for\_checker:

Анализирует возможные обязательные ходы для конкретной шашки. Определяет типы дружественных и вражеских шашек. Для обычной шашки проверяет возможность взятия в четырех диагональных направлениях. Для дамки проверяет возможность взятия на любое расстояние по диагоналям. Возвращает список всех возможных обязательных ходов.

1. Метод check\_for\_game\_over:

Проверяет наличие возможных ходов у обеих сторон. Если у одной из сторон нет возможных ходов, объявляет победу противоположной стороны через диалоговое окно. При завершении игры инициализирует новую игру с теми же параметрами.

1. Метод get\_moves\_list:

Сначала проверяет наличие обязательных ходов. Если обязательных ходов нет, возвращает список необязательных ходов. Этот метод обеспечивает соблюдение правила об обязательном взятии шашек противника.

1. Метод get\_required\_moves\_list:

Проверяет все шашки указанной стороны на наличие возможности взятия. Для обычных шашек проверяет наличие шашки противника и пустой клетки за ней. Для дамок проверяет возможность взятия на любое расстояние. Возвращает список всех возможных взятий.

1. Метод get\_optional\_moves\_list:

Проверяет все шашки указанной стороны на возможность обычного хода. Для простых шашек проверяет ход по диагонали вперед на одну клетку. Для дамок проверяет ход по диагонали на любое расстояние до первой занятой клетки. Возвращает список всех возможных обычных ходов.

1. **Класс Field**
2. Метод \_\_init\_\_

Конструктор класса Field принимает два параметра: x\_size и y\_size типа int, определяющие размеры игрового поля. При создании объекта размеры сохраняются в приватных атрибутах x\_size и \_y\_size. После сохранения размеров вызывается метод generate() для создания начального расположения шашек.

1. Свойство x\_size

Свойство x\_size является декорированным методом, который предоставляет доступ только для чтения к приватному атрибуту x\_size. Возвращает ширину игрового поля.

1. Свойство y\_size

Свойство y\_size является декорированным методом, который предоставляет доступ только для чтения к приватному атрибуту y\_size. Возвращает высоту игрового поля.

1. Свойство size

Свойство size является декорированным методом, который возвращает максимальное значение между шириной и высотой поля. Используется для определения максимального размера поля.

1. Метод generate

Метод generate создает начальное расположение шашек на поле. Создает двумерный массив объектов класса Checker и заполняет его шашками согласно правилам игры: черные шашки располагаются в верхней части поля (первые 5 рядов), а белые - в нижней (последние 5 рядов).

1. Метод type\_at

Метод type\_at принимает координаты x и y и возвращает тип шашки (CheckerType) в указанной позиции на поле. Используется для получения информации о типе шашки без доступа к самому объекту шашки.

1. Метод at

Метод at принимает координаты x и y и возвращает объект шашки (Checker) в указанной позиции на поле. Используется для получения прямого доступа к объекту шашки.

1. Метод is\_within

Метод is\_within принимает координаты x и y и проверяет, находятся ли они в пределах игрового поля. Возвращает True, если координаты валидны, и False в противном случае.

1. Свойство white\_checkers\_count

Подсчитывает и возвращает количество белых шашек (обычных и дамок) на поле.

1. Свойство black\_checkers\_count

Подсчитывает и возвращает количество черных шашек (обычных и дамок) на поле.

1. Свойство white\_score

Подсчитывает и возвращает счет белых шашек, где обычная шашка стоит 1 очко, а дамка - 3 очка.

1. Свойство black\_score

Подсчитывает и возвращает счет черных шашек, где обычная шашка стоит 1 очко, а дамка - 3 очка.

1. **Класс Checker:**
2. Метод init

Конструктор класса Checker принимает один необязательный параметр type типа CheckerType, который по умолчанию имеет значение CheckerType.NONE (пустая клетка). При создании объекта тип шашки сохраняется в приватном атрибуте type. Этот параметр определяет, какой тип шашки будет создан: пустая клетка, обычная белая, обычная черная, дамка белая или дамка черная.

1. Свойство type

Свойство type является декорированным методом, который предоставляет доступ только для чтения к приватному атрибуту type. Это свойство позволяет получить текущий тип шашки, не давая возможности изменить его напрямую.

1. Метод change\_type:

Изменяет тип шашки на указанный. Используется при превращении простой шашки в дамку или при удалении шашки с поля (установка типа NONE).

1. **Методы аутентификации**
2. Метод check\_user:

Проверяет существование пользователя и корректность пароля. Читает данные из файла users.json, хеширует введенный пароль и сравнивает с сохраненным хешем. Возвращает True при успешной аутентификации.

1. Метод register\_user:

Регистрирует нового пользователя. Проверяет уникальность имени пользователя, хеширует пароль и сохраняет данные в файл users.json. Возвращает True при успешной регистрации.

1. Класс GameGui
2. Метод init:

Инициализирует главное окно игры, устанавливает полноэкранный режим и создает основной холст для игрового поля.

1. Метод exit\_game:

Закрывает игру, уничтожая главное окно приложения.

1. Метод draw\_gui:

Создает и отображает главное окно игры с меню.

1. Метод draw\_gui:

Создает и отображает главное меню игры. Включает:

* Заголовок игры
* Кнопку "Игра с человеком"
* Кнопку "Правила"
* Кнопку "Выход"
* Все элементы оформлены в едином стиле с использованием современного дизайна и цветовой схемы.

1. Метод show\_rules:

Отображает новое окно с правилами игры. Создает прокручиваемый текстовый виджет с форматированным текстом правил и кнопкой закрытия.

1. Метод start\_game\_man:

Закрывает текущее окно и запускает новую игру, вызывая метод start\_game.

1. Метод start\_game:

Создает и настраивает игровое окно с полем для игры и боковой панелью. Включает:

* Отображение текущего хода
* Счет игроков
* Кнопки управления (новая игра, сдаться, выход)
* Адаптивное изменение размера игрового поля
* Обновление игровой информации в реальном времени

1. Метод surrender:

Обрабатывает сдачу игры. Показывает диалоговое окно подтверждения и, при положительном ответе, объявляет победителя и начинает новую игру.

1. **Метод auth\_gui:**

Создает окно авторизации размером 400x500 пикселей, центрированное на экране. Включает поля для ввода логина и пароля, а также кнопки "Войти", "Регистрация" и "Выход". Устанавливает единый стиль оформления с использованием светло-серого фона и современных шрифтов.

* 1. Функция clicked:

Проверяет введенные учетные данные, при успешной авторизации запускает игру, при ошибке выводит сообщение об ошибке.

* 1. Функция exit\_program:

Закрывает окно авторизации и завершает работу программы.

* 1. Функция open\_registration:

Закрывает окно авторизации и открывает окно регистрации.

1. **Метод reg\_gui:**

Создает окно регистрации размером 400x600 пикселей, центрированное на экране. Включает поля для ввода логина, пароля и подтверждения пароля, а также кнопки "Зарегистрироваться" и "Назад". Использует тот же стиль оформления, что и окно авторизации.

* 1. Функция register:

Проверяет корректность введенных данных (длина имени не менее 3 символов, длина пароля не менее 6 символов, совпадение паролей), регистрирует нового пользователя и при успехе возвращает к окну авторизации.

* 1. Функция back\_to\_auth:

Закрывает окно регистрации и возвращает пользователя к окну авторизации.

**4. Сообщения**

В программе реализована система следующих информационных сообщений, которые выводятся пользователю в различных ситуациях для контроля корректности действий:

1. Сообщения системы авторизации:

* При работе с формой авторизации пользователю могут быть показаны следующие сообщения об ошибках:
* 1. При попытке входа с пустыми полями выводится сообщение "Заполните все поля", которое информирует пользователя о необходимости заполнить все обязательные поля формы.
* В случае ввода неверных учетных данных появляется сообщение "Неверное имя пользователя или пароль", указывающее на ошибку в введенных данных.

1. Сообщения при регистрации:

* Во время регистрации нового пользователя система может вывести следующие предупреждения:
* При попытке создать учетную запись с коротким именем пользователя появляется сообщение "Имя пользователя должно содержать минимум 3 символа".
* Если введенный пароль слишком короткий, система информирует пользователя сообщением "Пароль должен содержать минимум 6 символов".
* В случае несовпадения паролей в полях ввода и подтверждения выводится сообщение "Пароли не совпадают".
* 4. При попытке зарегистрировать уже существующее имя пользователя появляется предупреждение "Пользователь с таким именем уже существует".

1. Игровые сообщения:

* В процессе игры пользователю показываются следующие информационные сообщения:
* При завершении игры выводится одно из сообщений:
* "Белые выиграли" - в случае победы белых
* "Черные выиграли" - в случае победы черных
* При попытке сдаться появляется диалоговое окно с вопросом "Вы действительно хотите сдаться?", требующее подтверждения действия.

1. Системные сообщения

* При возникновении системных ошибок пользователь может увидеть следующие сообщения:
* В случае отсутствия файла с изображениями шашек выводится сообщение "Не удалось загрузить изображения шашек".
* 2. При проблемах с файлом пользовательских данных появляется предупреждение "Ошибка доступа к файлу пользователей".

1. Информационные подсказки:

* В интерфейсе игры постоянно отображается следующая информация:
* Текущий ход: "Ход: Белые" или "Ход: Черные".
* Счет игры в формате:
* "Очки белых: X"
* "Очки черных: Y" , где X и Y - количество набранных очков каждой стороной.

# ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы была разработана программа "Канадские шашки" на языке Python с использованием библиотеки tkinter. В ходе разработки были успешно решены следующие задачи:

1. Создана полноценная игровая система с графическим интерфейсом, включающая:
   * + - Реализацию игрового поля 12x12 клеток
       - Систему ходов с учетом всех правил канадских шашек
       - Механику превращения в дамки
       - Систему обязательных взятий шашек противника
       - Подсчет очков для каждой стороны
2. Разработана система авторизации и хранения данных:
   * + - Регистрация новых пользователей
       - Авторизация существующих пользователей
       - Хеширование паролей для безопасности
       - Сохранение данных в JSON-формате
       - Защита от несанкционированного доступа
3. Реализована игровая логика:
   * + - Проверка возможных ходов
       - Определение обязательных взятий
       - Расчет траектории движения дамок
       - Подсчет набранных очков
       - Определение условий победы
4. Создан удобный пользовательский интерфейс:
   * + - Интуитивно понятное управление мышью
       - Анимация перемещения шашек
       - Подсветка выбранных клеток
       - Отображение возможных ходов
       - Информативные сообщения о ходе игры

Программная документация разработана в соответствии с требованиями к оформлению и содержанию. Код программы структурирован, оптимизирован и снабжен подробными комментариями.

В процессе разработки были получены практические навыки:

* + - * Объектно-ориентированного программирования на Python
      * Работы с графической библиотекой tkinter
      * Создания анимаций и визуальных эффектов
      * Проектирования пользовательских интерфейсов
      * Работы с файловой системой и JSON
      * Реализации систем аутентификации
      * Обработки пользовательского ввода
      * Создания игровой логики

Созданная программа полностью соответствует поставленным требованиям и может быть использована как для игры в канадские шашки между двумя игроками, так и для обучения правилам игры. Интуитивно понятный интерфейс и наглядная визуализация возможных ходов делают игру доступной даже для начинающих игроков.

# Листинг программы

import tkinter as tk  
from tkinter import messagebox  
from enum import Enum, auto  
from PIL import Image, ImageTk  
from pathlib import Path  
from time import sleep  
import json  
import hashlib  
  
*# Определение типов шашек и сторон*class SideType(Enum):  
 WHITE = auto()  
 BLACK = auto()  
  
 @staticmethod  
 def opposite(side):  
 if side == SideType.WHITE:  
 return SideType.BLACK  
 elif side == SideType.BLACK:  
 return SideType.WHITE  
 else:  
 return None  
  
  
class CheckerType(Enum):  
 NONE = auto()  
 WHITE\_REGULAR = auto()  
 BLACK\_REGULAR = auto()  
 WHITE\_QUEEN = auto()  
 BLACK\_QUEEN = auto()  
  
  
*# Определение точки*class Point:  
 def \_\_init\_\_(self, x: int = -1, y: int = -1):  
 self.\_\_x = x  
 self.\_\_y = y  
  
 @property  
 def x(self):  
 return self.\_\_x  
  
 @property  
 def y(self):  
 return self.\_\_y  
  
 def \_\_eq\_\_(self, other):  
 if isinstance(other, Point):  
 return self.x == other.x and self.y == other.y  
 return NotImplemented  
  
  
*# Определение движения*class Move:  
 def \_\_init\_\_(self, from\_x: int = -1, from\_y: int = -1, to\_x: int = -1, to\_y: int = -1):  
 self.\_from\_x = from\_x  
 self.\_from\_y = from\_y  
 self.\_to\_x = to\_x  
 self.\_to\_y = to\_y  
  
 @property  
 def from\_x(self):  
 return self.\_from\_x  
  
 @property  
 def from\_y(self):  
 return self.\_from\_y  
  
 @property  
 def to\_x(self):  
 return self.\_to\_x  
  
 @property  
 def to\_y(self):  
 return self.\_to\_y  
  
 def \_\_str\_\_(self):  
 return f'{self.\_from\_x}-{self.\_from\_y} -> {self.\_to\_x}-{self.\_to\_y}'  
  
 def \_\_repr\_\_(self):  
 return f'{self.\_from\_x}-{self.\_from\_y} -> {self.\_to\_x}-{self.\_to\_y}'  
  
 def \_\_eq\_\_(self, other):  
 if isinstance(other, Move):  
 return (  
 self.\_from\_x == other.\_from\_x and  
 self.\_from\_y == other.\_from\_y and  
 self.\_to\_x == other.\_to\_x and  
 self.\_to\_y == other.\_to\_y  
 )  
 return NotImplemented  
  
*# Определение шашки*class Checker:  
 def \_\_init\_\_(self, type: CheckerType = CheckerType.NONE):  
 self.\_\_type = type  
  
 @property  
 def type(self):  
 return self.\_\_type  
  
 def change\_type(self, type: CheckerType):  
 *'''Изменение типа шашки'''* self.\_\_type = type  
  
  
*# Определение игровых констант*PLAYER\_SIDE = SideType.WHITE  
X\_SIZE = Y\_SIZE = 12  
CELL\_SIZE = 75  
ANIMATION\_SPEED = 4  
MAX\_PREDICTION\_DEPTH = 3  
BORDER\_WIDTH = 2 \* 2  
FIELD\_COLORS = ['#E7CFA9', '#927456']  
HOVER\_BORDER\_COLOR = '#54b346'  
SELECT\_BORDER\_COLOR = '#944444'  
POSIBLE\_MOVE\_CIRCLE\_COLOR = '#944444'  
  
MOVE\_OFFSETS = [  
 Point(-1, -1),  
 Point(1, -1),  
 Point(-1, 1),  
 Point(1, 1)  
]  
  
WHITE\_CHECKERS = [CheckerType.WHITE\_REGULAR, CheckerType.WHITE\_QUEEN]  
BLACK\_CHECKERS = [CheckerType.BLACK\_REGULAR, CheckerType.BLACK\_QUEEN]  
  
  
*# Определение игрового поля*class Field:  
 def \_\_init\_\_(self, x\_size: int, y\_size: int):  
 self.\_\_x\_size = x\_size  
 self.\_\_y\_size = y\_size  
 self.generate()  
  
 @property  
 def x\_size(self) -> int:  
 return self.\_\_x\_size  
  
 @property  
 def y\_size(self) -> int:  
 return self.\_\_y\_size  
  
 @property  
 def size(self) -> int:  
 return max(self.x\_size, self.y\_size)  
  
 def generate(self):  
 *'''Генерация поля с шашками и задаёт количество всего'''* self.checkers = [[Checker() for x in range(self.x\_size)] for y in range(self.y\_size)]  
 for y in range(self.y\_size):  
 for x in range(self.x\_size):  
 if (y + x) % 2:  
 if (y < 5):  
 self.checkers[y][x].change\_type(CheckerType.BLACK\_REGULAR)  
 elif (y >= self.y\_size - 5):  
 self.checkers[y][x].change\_type(CheckerType.WHITE\_REGULAR)  
  
 def type\_at(self, x: int, y: int) -> CheckerType:  
 *'''Получение типа шашки на поле по координатам'''* return self.checkers[y][x].type  
  
 def at(self, x: int, y: int) -> Checker:  
 *'''Получение шашки на поле по координатам'''* return self.checkers[y][x]  
  
 def is\_within(self, x: int, y: int) -> bool:  
 *'''Определяет лежит ли точка в пределах поля'''* return (0 <= x < self.x\_size and 0 <= y < self.y\_size)  
  
 @property  
 def white\_checkers\_count(self) -> int:  
 *'''Количество белых шашек на поле'''* count = 0  
 for row in self.checkers:  
 for checker in row:  
 if checker.type in WHITE\_CHECKERS:  
 count += 1  
 return count  
  
 @property  
 def black\_checkers\_count(self) -> int:  
 *'''Количество чёрных шашек на поле'''* count = 0  
 for row in self.checkers:  
 for checker in row:  
 if checker.type in BLACK\_CHECKERS:  
 count += 1  
 return count  
  
 @property  
 def white\_score(self) -> int:  
 *'''Счёт белых'''* score = 0  
 for row in self.checkers:  
 for checker in row:  
 if checker.type == CheckerType.WHITE\_REGULAR:  
 score += 1  
 elif checker.type == CheckerType.WHITE\_QUEEN:  
 score += 3  
 return score  
  
 @property  
 def black\_score(self) -> int:  
 *'''Счёт чёрных'''* score = 0  
 for row in self.checkers:  
 for checker in row:  
 if checker.type == CheckerType.BLACK\_REGULAR:  
 score += 1  
 elif checker.type == CheckerType.BLACK\_QUEEN:  
 score += 3  
 return score  
  
  
*# Определение игры*class Game:  
 def \_\_init\_\_(self, canvas: tk.Canvas, x\_field\_size: int, y\_field\_size: int):  
 self.canvas = canvas  
 self.field = Field(x\_field\_size, y\_field\_size)  
  
 self.current\_player = SideType.WHITE  
  
 self.hovered\_cell = Point()  
 self.selected\_cell = Point()  
 self.animated\_cell = Point()  
  
 self.is\_animating = False  
  
 self.white\_points = 0  
 self.black\_points = 0  
  
 self.init\_images()  
  
 def init\_images(self):  
 *'''Инициализация изображений'''* self.images = {  
 CheckerType.WHITE\_REGULAR: ImageTk.PhotoImage(  
 Image.open(Path('assets', 'white-regular.png')).resize((CELL\_SIZE, CELL\_SIZE), Image.LANCZOS)),  
 CheckerType.BLACK\_REGULAR: ImageTk.PhotoImage(  
 Image.open(Path('assets', 'black-regular.png')).resize((CELL\_SIZE, CELL\_SIZE), Image.LANCZOS)),  
 CheckerType.WHITE\_QUEEN: ImageTk.PhotoImage(  
 Image.open(Path('assets', 'white-queen.png')).resize((CELL\_SIZE, CELL\_SIZE), Image.LANCZOS)),  
 CheckerType.BLACK\_QUEEN: ImageTk.PhotoImage(  
 Image.open(Path('assets', 'black-queen.png')).resize((CELL\_SIZE, CELL\_SIZE), Image.LANCZOS)),  
 }  
  
 def animate\_move(self, move: Move):  
 *'''Анимация перемещения шашки'''* self.is\_animating = True *# Устанавливаем флаг анимаци* self.animated\_cell = Point(move.from\_x, move.from\_y)  
 self.draw()  
  
 *# Создание шашки для анимации* animated\_checker = self.canvas.create\_image(move.from\_x \* CELL\_SIZE, move.from\_y \* CELL\_SIZE,  
 image=self.images.get(  
 self.field.type\_at(move.from\_x, move.from\_y)), anchor='nw',  
 tag='animated\_checker')  
  
 *# Вектора движения* dx = 1 if move.from\_x < move.to\_x else -1  
 dy = 1 if move.from\_y < move.to\_y else -1  
  
 *# Анимация* for distance in range(abs(move.from\_x - move.to\_x)):  
 for \_ in range(100 // ANIMATION\_SPEED):  
 self.canvas.move(animated\_checker, ANIMATION\_SPEED / 100 \* CELL\_SIZE \* dx,  
 ANIMATION\_SPEED / 100 \* CELL\_SIZE \* dy)  
 self.canvas.update()  
 sleep(0.01)  
  
 self.animated\_cell = Point()  
 self.is\_animating = False  
  
 def draw(self):  
 *'''Отрисовка сетки поля и шашек'''* self.canvas.delete('all')  
 self.draw\_field\_grid()  
 self.draw\_checkers()  
  
 def draw\_field\_grid(self):  
 *'''Отрисовка сетки поля'''* for y in range(self.field.y\_size):  
 for x in range(self.field.x\_size):  
 self.canvas.create\_rectangle(x \* CELL\_SIZE, y \* CELL\_SIZE, x \* CELL\_SIZE + CELL\_SIZE,  
 y \* CELL\_SIZE + CELL\_SIZE, fill=FIELD\_COLORS[(y + x) % 2], width=0,  
 tag='boards')  
  
 *# Отрисовка рамок у необходимых клеток* if (x == self.selected\_cell.x and y == self.selected\_cell.y):  
 self.canvas.create\_rectangle(x \* CELL\_SIZE + BORDER\_WIDTH // 2, y \* CELL\_SIZE + BORDER\_WIDTH // 2,  
 x \* CELL\_SIZE + CELL\_SIZE - BORDER\_WIDTH // 2,  
 y \* CELL\_SIZE + CELL\_SIZE - BORDER\_WIDTH // 2,  
 outline=SELECT\_BORDER\_COLOR, width=BORDER\_WIDTH, tag='border')  
 elif (x == self.hovered\_cell.x and y == self.hovered\_cell.y):  
 self.canvas.create\_rectangle(x \* CELL\_SIZE + BORDER\_WIDTH // 2, y \* CELL\_SIZE + BORDER\_WIDTH // 2,  
 x \* CELL\_SIZE + CELL\_SIZE - BORDER\_WIDTH // 2,  
 y \* CELL\_SIZE + CELL\_SIZE - BORDER\_WIDTH // 2,  
 outline=HOVER\_BORDER\_COLOR, width=BORDER\_WIDTH, tag='border')  
  
 *# Отрисовка возможных точек перемещения, если есть выбранная ячейка* if (self.selected\_cell):  
 self.draw\_possible\_moves()  
  
 def draw\_possible\_moves(self):  
 self.canvas.delete('posible\_move\_circle')  
  
 *# Получаем возможные ходы для текущего игрока* player\_moves\_list = self.get\_moves\_list(self.current\_player)  
  
 *# Отрисовываем возможные ходы для текущего игрока* for move in player\_moves\_list:  
 if (self.selected\_cell.x == move.from\_x and self.selected\_cell.y == move.from\_y):  
 self.canvas.create\_oval(move.to\_x \* CELL\_SIZE + CELL\_SIZE / 3,  
 move.to\_y \* CELL\_SIZE + CELL\_SIZE / 3,  
 move.to\_x \* CELL\_SIZE + (CELL\_SIZE - CELL\_SIZE / 3),  
 move.to\_y \* CELL\_SIZE + (CELL\_SIZE - CELL\_SIZE / 3),  
 fill=POSIBLE\_MOVE\_CIRCLE\_COLOR, width=0,  
 tag='posible\_move\_circle')  
  
 *# Получаем возможные ходы для противника* opponent\_moves\_list = self.get\_moves\_list(SideType.opposite(self.current\_player))  
  
 *# Отрисовываем возможные ходы для противника* for move in opponent\_moves\_list:  
 if (self.selected\_cell.x == move.from\_x and self.selected\_cell.y == move.from\_y):  
 self.canvas.create\_oval(move.to\_x \* CELL\_SIZE + CELL\_SIZE / 3,  
 move.to\_y \* CELL\_SIZE + CELL\_SIZE / 3,  
 move.to\_x \* CELL\_SIZE + (CELL\_SIZE - CELL\_SIZE / 3),  
 move.to\_y \* CELL\_SIZE + (CELL\_SIZE - CELL\_SIZE / 3),  
 fill=POSIBLE\_MOVE\_CIRCLE\_COLOR, width=0,  
 tag='posible\_move\_circle')  
 def draw\_checkers(self):  
 *'''Отрисовка шашек'''* for y in range(self.field.y\_size):  
 for x in range(self.field.x\_size):  
 *# Не отрисовывать пустые ячейки и анимируемую шашку* if (self.field.type\_at(x, y) != CheckerType.NONE and not (  
 x == self.animated\_cell.x and y == self.animated\_cell.y)):  
 self.canvas.create\_image(x \* CELL\_SIZE, y \* CELL\_SIZE,  
 image=self.images.get(self.field.type\_at(x, y)), anchor='nw',  
 tag='checkers')  
  
 def mouse\_move(self, event: tk.Event):  
 *'''Событие перемещения мышки'''* x, y = (event.x) // CELL\_SIZE, (event.y) // CELL\_SIZE  
 if (x != self.hovered\_cell.x or y != self.hovered\_cell.y):  
 self.hovered\_cell = Point(x, y)  
  
 self.draw()  
  
 def mouse\_down(self, event: tk.Event):  
 *'''Событие нажатия мышки'''* x, y = (event.x) // CELL\_SIZE, (event.y) // CELL\_SIZE  
  
 *# Если точка не внутри поля или идет анимация* if not self.field.is\_within(x, y) or self.is\_animating:  
 return  
  
 *# Определяем, какие шашки у текущего игрока* if self.current\_player == SideType.WHITE:  
 player\_checkers = WHITE\_CHECKERS  
 else:  
 player\_checkers = BLACK\_CHECKERS  
  
 *# Получаем список всех обязательных ходов* all\_required\_moves = []  
 for check\_y in range(self.field.y\_size):  
 for check\_x in range(self.field.x\_size):  
 if self.field.type\_at(check\_x, check\_y) in player\_checkers:  
 moves = self.get\_required\_moves\_list\_for\_checker(self.current\_player, check\_x, check\_y)  
 all\_required\_moves.extend(moves)  
  
 *# Если есть обязательные ходы* if all\_required\_moves:  
 *# Если уже есть выбранная шашка с обязательным ходом* if self.selected\_cell.x != -1:  
 required\_moves = self.get\_required\_moves\_list\_for\_checker(  
 self.current\_player,   
 self.selected\_cell.x,   
 self.selected\_cell.y  
 )  
 if required\_moves:  
 *# Можно ходить только этой шашкой* if x == self.selected\_cell.x and y == self.selected\_cell.y:  
 return  
 move = Move(self.selected\_cell.x, self.selected\_cell.y, x, y)  
 if move in required\_moves:  
 self.handle\_player\_turn(move, x, y)  
 return  
   
 *# Если выбирается новая шашка* if self.field.type\_at(x, y) in player\_checkers:  
 moves = self.get\_required\_moves\_list\_for\_checker(self.current\_player, x, y)  
 if moves:  
 self.selected\_cell = Point(x, y)  
 self.draw()  
 return  
  
 *# Если нет обязательных ходов* if self.field.type\_at(x, y) in player\_checkers:  
 self.selected\_cell = Point(x, y)  
 self.draw()  
 elif self.selected\_cell.x != -1:  
 move = Move(self.selected\_cell.x, self.selected\_cell.y, x, y)  
 if move in self.get\_moves\_list(self.current\_player):  
 self.handle\_player\_turn(move, x, y)  
  
 def handle\_move(self, move: Move, draw: bool = True) -> bool:  
 *'''Совершение хода'''* if draw:  
 self.animate\_move(move)  
  
 *# Изменение позиции шашки* self.field.at(move.to\_x, move.to\_y).change\_type(self.field.type\_at(move.from\_x, move.from\_y))  
 self.field.at(move.from\_x, move.from\_y).change\_type(CheckerType.NONE)  
  
 *# Вектора движения* dx = -1 if move.from\_x < move.to\_x else 1  
 dy = -1 if move.from\_y < move.to\_y else 1  
  
 *# Удаление съеденных шашек* has\_killed\_checker = False  
 x, y = move.to\_x, move.to\_y  
 while x != move.from\_x or y != move.from\_y:  
 x += dx  
 y += dy  
 if self.field.type\_at(x, y) != CheckerType.NONE:  
 *# Подсчет очков в зависимости от типа съеденной шашки* checker\_type = self.field.type\_at(x, y)  
 if self.current\_player == SideType.WHITE:  
 if checker\_type == CheckerType.BLACK\_REGULAR:  
 self.white\_points += 1  
 elif checker\_type == CheckerType.BLACK\_QUEEN:  
 self.white\_points += 3  
 else:  
 if checker\_type == CheckerType.WHITE\_REGULAR:  
 self.black\_points += 1  
 elif checker\_type == CheckerType.WHITE\_QUEEN:  
 self.black\_points += 3  
 self.field.at(x, y).change\_type(CheckerType.NONE)  
 has\_killed\_checker = True  
  
 if draw:  
 self.draw()  
 return has\_killed\_checker  
  
 def handle\_player\_turn(self, move: Move, x, y):  
 *'''Обработка хода игрока'''  
  
 # Была ли убита шашка* has\_killed\_checker = self.handle\_move(move)  
  
 *# Проверяем достижение последней линии* reached\_end = (self.current\_player == SideType.WHITE and y == 0) or \  
 (self.current\_player == SideType.BLACK and y == self.field.y\_size - 1)  
  
 *# Проверяем, есть ли обязательные ходы для текущей шашки* required\_moves\_list = self.get\_required\_moves\_list\_for\_checker(self.current\_player, x, y)  
  
 *# Если есть обязательные ходы или достигнут край с возможностью взятия* if (has\_killed\_checker and required\_moves\_list) or (reached\_end and required\_moves\_list):  
 *# Игрок должен продолжать ходить той же шашкой* self.selected\_cell = Point(x, y) *# Оставить выбранной текущую ячейку* self.draw() *# Перерисовать поле* else:  
 *# Если нет обязательных ходов, проверяем на превращение в дамку* if self.current\_player == SideType.WHITE and y == 0 and self.field.type\_at(x, y) == CheckerType.WHITE\_REGULAR:  
 self.field.at(x, y).change\_type(CheckerType.WHITE\_QUEEN)  
 elif self.current\_player == SideType.BLACK and y == self.field.y\_size - 1 and self.field.type\_at(x, y) == CheckerType.BLACK\_REGULAR:  
 self.field.at(x, y).change\_type(CheckerType.BLACK\_QUEEN)  
   
 *# Переключаем игрока* self.current\_player = SideType.opposite(self.current\_player) *# Переключить игрока* self.selected\_cell = Point() *# Сбросить выбранную ячейку* self.draw() *# Перерисовать поле* self.check\_for\_game\_over()  
  
 def get\_required\_moves\_list\_for\_checker(self, side: SideType, x: int, y: int) -> list[Move]:  
 *'''Получение списка обязательных ходов для конкретной шашки'''* moves\_list = []  
  
 *# Определение типов шашек* if side == SideType.WHITE:  
 friendly\_checkers = WHITE\_CHECKERS  
 enemy\_checkers = BLACK\_CHECKERS  
 elif side == SideType.BLACK:  
 friendly\_checkers = BLACK\_CHECKERS  
 enemy\_checkers = WHITE\_CHECKERS  
 else:  
 return moves\_list  
  
 *# Для обычной шашки* if self.field.type\_at(x, y) == friendly\_checkers[0]:  
 for offset in MOVE\_OFFSETS:  
 if not self.field.is\_within(x + offset.x \* 2, y + offset.y \* 2):  
 continue  
  
 if self.field.type\_at(x + offset.x, y + offset.y) in enemy\_checkers and self.field.type\_at(  
 x + offset.x \* 2, y + offset.y \* 2) == CheckerType.NONE:  
 moves\_list.append(Move(x, y, x + offset.x \* 2, y + offset.y \* 2))  
  
 *# Для дамки* elif self.field.type\_at(x, y) == friendly\_checkers[1]:  
 for offset in MOVE\_OFFSETS:  
 if not self.field.is\_within(x + offset.x \* 2, y + offset.y \* 2):  
 continue  
  
 has\_enemy\_checker\_on\_way = False  
  
 for shift in range(1, self.field.size):  
 if not self.field.is\_within(x + offset.x \* shift, y + offset.y \* shift):  
 continue  
  
 *# Если на пути не было вражеской шашки* if not has\_enemy\_checker\_on\_way:  
 if self.field.type\_at(x + offset.x \* shift, y + offset.y \* shift) in enemy\_checkers:  
 has\_enemy\_checker\_on\_way = True  
 continue  
 *# Если на пути союзная шашка - то закончить цикл* elif self.field.type\_at(x + offset.x \* shift, y + offset.y \* shift) in friendly\_checkers:  
 break  
  
 *# Если на пути была вражеская шашка* if has\_enemy\_checker\_on\_way:  
 if self.field.type\_at(x + offset.x \* shift, y + offset.y \* shift) == CheckerType.NONE:  
 moves\_list.append(Move(x, y, x + offset.x \* shift, y + offset.y \* shift))  
 else:  
 break  
  
 return moves\_list  
  
 def check\_for\_game\_over(self):  
 *'''Проверка на конец игры'''* game\_over = False  
  
 white\_moves\_list = self.get\_moves\_list(SideType.WHITE)  
 if not (white\_moves\_list):  
 *# Белые проиграли* answer = tk.messagebox.showinfo('Конец игры', 'Чёрные выиграли')  
 game\_over = True  
  
 black\_moves\_list = self.get\_moves\_list(SideType.BLACK)  
 if not (black\_moves\_list):  
 *# Чёрные проиграли* answer = tk.messagebox.showinfo('Конец игры', 'Белые выиграли')  
 game\_over = True  
  
 if (game\_over):  
 *# Новая игра* self.\_\_init\_\_(self.canvas, self.field.x\_size, self.field.y\_size)  
  
 def get\_moves\_list(self, side: SideType) -> list[Move]:  
 *'''Получение списка ходов'''* moves\_list = self.get\_required\_moves\_list(side)  
 if not (moves\_list):  
 moves\_list = self.get\_optional\_moves\_list(side)  
 return moves\_list  
   
 def get\_required\_moves\_list(self, side) :  
 *'''Получение списка обязательных ходов'''* moves\_list = []  
  
 *# Определение типов шашек* if (side == SideType.WHITE):  
 friendly\_checkers = WHITE\_CHECKERS  
 enemy\_checkers = BLACK\_CHECKERS  
 elif (side == SideType.BLACK):  
 friendly\_checkers = BLACK\_CHECKERS  
 enemy\_checkers = WHITE\_CHECKERS  
 else:  
 return moves\_list  
  
 for y in range(self.field.y\_size):  
 for x in range(self.field.x\_size):  
  
 *# Для обычной шашки* if (self.field.type\_at(x, y) == friendly\_checkers[0]):  
 for offset in MOVE\_OFFSETS:  
 if not (self.field.is\_within(x + offset.x \* 2, y + offset.y \* 2)): continue  
  
 if self.field.type\_at(x + offset.x, y + offset.y) in enemy\_checkers and self.field.type\_at(  
 x + offset.x \* 2, y + offset.y \* 2) == CheckerType.NONE:  
 moves\_list.append(Move(x, y, x + offset.x \* 2, y + offset.y \* 2))  
  
 *# Для дамки* elif (self.field.type\_at(x, y) == friendly\_checkers[1]):  
 for offset in MOVE\_OFFSETS:  
 if not (self.field.is\_within(x + offset.x \* 2, y + offset.y \* 2)): continue  
  
 has\_enemy\_checker\_on\_way = False  
  
 for shift in range(1, self.field.size):  
 if not (self.field.is\_within(x + offset.x \* shift, y + offset.y \* shift)): continue  
  
 *# Если на пути не было вражеской шашки* if (not has\_enemy\_checker\_on\_way):  
 if (self.field.type\_at(x + offset.x \* shift, y + offset.y \* shift) in enemy\_checkers):  
 has\_enemy\_checker\_on\_way = True  
 continue  
 *# Если на пути союзная шашка - то закончить цикл* elif (self.field.type\_at(x + offset.x \* shift,  
 y + offset.y \* shift) in friendly\_checkers):  
 break  
  
 *# Если на пути была вражеская шашка* if (has\_enemy\_checker\_on\_way):  
 if (self.field.type\_at(x + offset.x \* shift,  
 y + offset.y \* shift) == CheckerType.NONE):  
 moves\_list.append(Move(x, y, x + offset.x \* shift, y + offset.y \* shift))  
 else:  
 break  
  
 return moves\_list  
 def get\_optional\_moves\_list(self, side: SideType) -> list[Move]:  
 *'''Получение списка необязательных ходов'''* moves\_list = []  
  
 *# Определение типов шашек* if (side == SideType.WHITE):  
 friendly\_checkers = WHITE\_CHECKERS  
 elif (side == SideType.BLACK):  
 friendly\_checkers = BLACK\_CHECKERS  
 else:  
 return moves\_list  
  
 for y in range(self.field.y\_size):  
 for x in range(self.field.x\_size):  
 *# Для обычной шашки* if (self.field.type\_at(x, y) == friendly\_checkers[0]):  
 for offset in MOVE\_OFFSETS[:2] if side == SideType.WHITE else MOVE\_OFFSETS[2:]:  
 if not (self.field.is\_within(x + offset.x, y + offset.y)): continue  
  
 if (self.field.type\_at(x + offset.x, y + offset.y) == CheckerType.NONE):  
 moves\_list.append(Move(x, y, x + offset.x, y + offset.y))  
  
 *# Для дамки* elif (self.field.type\_at(x, y) == friendly\_checkers[1]):  
 for offset in MOVE\_OFFSETS:  
 if not (self.field.is\_within(x + offset.x, y + offset.y)): continue  
  
 for shift in range(1, self.field.size):  
 if not (self.field.is\_within(x + offset.x \* shift, y + offset.y \* shift)): continue  
  
 if (self.field.type\_at(x + offset.x \* shift, y + offset.y \* shift) == CheckerType.NONE):  
 moves\_list.append(Move(x, y, x + offset.x \* shift, y + offset.y \* shift))  
 else:  
 break  
 return moves\_list  
  
def check\_user(username: str, password: str) -> bool:  
 *"""Проверка существования пользователя"""* try:  
 with open('users.json', 'r') as file:  
 users = json.load(file)  
   
 *# Хешируем введенный пароль* hashed\_password = hashlib.sha256(password.encode()).hexdigest()  
   
 *# Проверяем существование пользователя и правильность пароля* if username in users and users[username] == hashed\_password:  
 return True  
 return False  
 except FileNotFoundError:  
 return False  
  
def register\_user(username: str, password: str) -> bool:  
 *"""Регистрация нового пользователя"""* try:  
 *# Пытаемся загрузить существующих пользователей* with open('users.json', 'r') as file:  
 users = json.load(file)  
 except FileNotFoundError:  
 *# Если файл не существует, создаем пустой словарь* users = {}  
   
 *# Проверяем, не существует ли уже такой пользователь* if username in users:  
 return False  
   
 *# Хешируем пароль перед сохранением* hashed\_password = hashlib.sha256(password.encode()).hexdigest()  
   
 *# Добавляем нового пользователя* users[username] = hashed\_password  
   
 *# Сохраняем обновленный список пользователей* with open('users.json', 'w') as file:  
 json.dump(users, file)  
   
 return True  
  
def auth\_gui():  
 window = tk.Tk()  
 window.title('Авторизация')  
   
 *# Устанавливаем размеры окна* window\_width = 400  
 window\_height = 500  
   
 *# Получаем размеры экрана* screen\_width = window.winfo\_screenwidth()  
 screen\_height = window.winfo\_screenheight()  
   
 *# Вычисляем координаты для центрирования окна* x = (screen\_width - window\_width) // 2  
 y = (screen\_height - window\_height) // 2  
   
 *# Устанавливаем размеры и положение окна* window.geometry(f'{window\_width}x{window\_height}+{x}+{y}')  
 window.resizable(False, False)  
 window.configure(bg='#f5f6fa') *# Светло-серый фон  
  
 # Создаем основной контейнер* main\_container = tk.Frame(window, bg='#f5f6fa')  
 main\_container.pack(expand=True)  
  
 *# Общие стили* font\_header = ('Arial', 24, 'bold')  
 font\_entry = ('Arial', 12)  
 font\_label = ('Arial', 10)  
  
 entry\_style = {  
 'bg': '#ffffff', *# Белый фон* 'fg': '#2c3e50', *# Темно-синий текст* 'font': font\_entry,  
 'relief': tk.FLAT,  
 'width': 25,  
 'highlightbackground': '#dcdde1', *# Цвет рамки в неактивном состоянии* 'highlightthickness': 1,  
 'insertbackground': '#2c3e50' *# Цвет курсора* }  
  
 button\_style = {  
 'font': font\_entry,  
 'relief': tk.FLAT,  
 'cursor': 'hand2',  
 'width': 20, *# Увеличенная ширина кнопок* 'height': 2  
 }  
  
 *# Заголовок* main\_label = tk.Label(main\_container,   
 text='Авторизация',   
 font=font\_header,  
 bg='#f5f6fa',  
 fg='#2f3542') *# Темно-серый текст* main\_label.pack(pady=(30, 20))  
  
 *# Контейнер для полей ввода* entry\_container = tk.Frame(main\_container, bg='#f5f6fa')  
 entry\_container.pack(pady=20)  
  
 *# Поля для ввода* username\_label = tk.Label(entry\_container,   
 text='Имя пользователя',   
 font=font\_label,  
 bg='#f5f6fa',  
 fg='#2f3542')  
 username\_label.pack(pady=5)  
   
 username\_entry = tk.Entry(entry\_container, \*\*entry\_style)  
 username\_entry.pack(pady=5)  
  
 password\_label = tk.Label(entry\_container,   
 text='Пароль',   
 font=font\_label,  
 bg='#f5f6fa',  
 fg='#2f3542')  
 password\_label.pack(pady=5)  
  
 password\_entry = tk.Entry(entry\_container, show="\*", \*\*entry\_style)  
 password\_entry.pack(pady=5)  
  
 *# Контейнер для кнопок* button\_container = tk.Frame(main\_container, bg='#f5f6fa')  
 button\_container.pack(pady=30)  
  
 *# Кнопки* send\_btn = tk.Button(button\_container,   
 text='Войти',   
 command=lambda: clicked(),  
 bg="#70a1ff", *# Голубой* fg="white",  
 activebackground="#1e90ff",  
 activeforeground="white",  
 \*\*button\_style)  
 send\_btn.pack(pady=10)  
  
 reg\_btn = tk.Button(button\_container,   
 text='Регистрация',   
 command=lambda: open\_registration(),  
 bg="#ff7f50", *# Коралловый* fg="white",  
 activebackground="#ff6348",  
 activeforeground="white",  
 \*\*button\_style)  
 reg\_btn.pack(pady=10)  
  
 exit\_btn = tk.Button(button\_container,   
 text='Выход',   
 command=lambda: exit\_program(),  
 bg="#ff4757", *# Красный* fg="white",  
 activebackground="#ff6b81",  
 activeforeground="white",  
 \*\*button\_style)  
 exit\_btn.pack(pady=10)  
  
 def clicked():  
 username = username\_entry.get()  
 password = password\_entry.get()  
  
 *# Проверяем, что поля не пустые* if not username or not password:  
 messagebox.showerror("Ошибка", "Заполните все поля")  
 return  
  
 if check\_user(username, password):  
 window.destroy()  
 GameGui().draw\_gui()  
 else:  
 messagebox.showerror("Ошибка", "Неверное имя пользователя или пароль")  
  
 def exit\_program():  
 window.destroy()  
   
 def open\_registration():  
 window.destroy()  
 reg\_gui()  
  
 window.mainloop()  
  
def reg\_gui():  
 window = tk.Tk()  
 window.title('Регистрация')  
   
 *# Устанавливаем размеры окна* window\_width = 400  
 window\_height = 600 *# Увеличена высота для нового поля  
   
 # Получаем размеры экрана* screen\_width = window.winfo\_screenwidth()  
 screen\_height = window.winfo\_screenheight()  
   
 *# Вычисляем координаты для центрирования окна* x = (screen\_width - window\_width) // 2  
 y = (screen\_height - window\_height) // 2  
   
 *# Устанавливаем размеры и положение окна* window.geometry(f'{window\_width}x{window\_height}+{x}+{y}')  
 window.resizable(False, False)  
 window.configure(bg='#f5f6fa') *# Светло-серый фон  
  
 # Создаем основной контейнер* main\_container = tk.Frame(window, bg='#f5f6fa')  
 main\_container.pack(expand=True)  
  
 *# Общие стили (такие же, как в auth\_gui)* font\_header = ('Arial', 24, 'bold')  
 font\_entry = ('Arial', 12)  
 font\_label = ('Arial', 10)  
  
 entry\_style = {  
 'bg': '#ffffff',  
 'fg': '#2c3e50',  
 'font': font\_entry,  
 'relief': tk.FLAT,  
 'width': 25,  
 'highlightbackground': '#dcdde1',  
 'highlightthickness': 1,  
 'insertbackground': '#2c3e50'  
 }  
  
 button\_style = {  
 'font': font\_entry,  
 'relief': tk.FLAT,  
 'cursor': 'hand2',  
 'width': 20, *# Такая же ширина, как в auth\_gui* 'height': 2  
 }  
  
 *# Заголовок* main\_label = tk.Label(main\_container,   
 text='Регистрация',   
 font=font\_header,  
 bg='#f5f6fa',  
 fg='#2f3542') *# Темно-серый текст* main\_label.pack(pady=(30, 20))  
  
 *# Контейнер для полей ввода* entry\_container = tk.Frame(main\_container, bg='#f5f6fa')  
 entry\_container.pack(pady=20)  
  
 *# Поля для ввода* username\_label = tk.Label(entry\_container,   
 text='Придумайте имя пользователя',   
 font=font\_label,  
 bg='#f5f6fa',  
 fg='#2f3542')  
 username\_label.pack(pady=5)  
   
 username\_entry = tk.Entry(entry\_container, \*\*entry\_style)  
 username\_entry.pack(pady=5)  
  
 password\_label = tk.Label(entry\_container,   
 text='Придумайте пароль',   
 font=font\_label,  
 bg='#f5f6fa',  
 fg='#2f3542')  
 password\_label.pack(pady=5)  
  
 password\_entry = tk.Entry(entry\_container, show="\*", \*\*entry\_style)  
 password\_entry.pack(pady=5)  
  
 *# Новое поле для повторения пароля* confirm\_password\_label = tk.Label(entry\_container,   
 text='Повторите пароль',   
 font=font\_label,  
 bg='#f5f6fa',  
 fg='#2f3542')  
 confirm\_password\_label.pack(pady=5)  
  
 confirm\_password\_entry = tk.Entry(entry\_container, show="\*", \*\*entry\_style)  
 confirm\_password\_entry.pack(pady=5)  
  
 *# Контейнер для кнопок* button\_container = tk.Frame(main\_container, bg='#f5f6fa')  
 button\_container.pack(pady=30)  
  
 *# Кнопки* reg\_btn = tk.Button(button\_container,   
 text='Зарегистрироваться',   
 command=lambda: register(),  
 bg="#70a1ff",  
 fg="white",  
 activebackground="#1e90ff",  
 activeforeground="white",  
 \*\*button\_style)  
 reg\_btn.pack(pady=10)  
  
 back\_btn = tk.Button(button\_container,   
 text='Назад',   
 command=lambda: back\_to\_auth(),  
 bg="#ff7f50",  
 fg="white",  
 activebackground="#ff6348",  
 activeforeground="white",  
 \*\*button\_style)  
 back\_btn.pack(pady=10)  
  
 def register():  
 username = username\_entry.get()  
 password = password\_entry.get()  
 confirm\_password = confirm\_password\_entry.get()  
  
 *# Проверяем, что все поля заполнены* if not username or not password or not confirm\_password:  
 messagebox.showerror("Ошибка", "Заполните все поля")  
 return  
  
 *# Проверяем длину имени пользователя и пароля* if len(username) < 3:  
 messagebox.showerror("Ошибка", "Имя пользователя должно содержать минимум 3 символа")  
 return  
  
 if len(password) < 6:  
 messagebox.showerror("Ошибка", "Пароль должен содержать минимум 6 символов")  
 return  
  
 *# Проверяем совпадение паролей* if password != confirm\_password:  
 messagebox.showerror("Ошибка", "Пароли не совпадают")  
 return  
  
 if register\_user(username, password):  
 messagebox.showinfo("Успех", "Регистрация успешна!")  
 window.destroy()  
 auth\_gui()  
 else:  
 messagebox.showerror("Ошибка", "Пользователь с таким именем уже существует")  
  
 def back\_to\_auth():  
 window.destroy()  
 auth\_gui()  
  
 window.mainloop()  
  
class GameGui:  
 def \_\_init\_\_(self):  
 self.main\_window = tk.Tk()  
 self.main\_window.title('Канадские шашки')  
 self.main\_window.attributes("-fullscreen", True)  
   
 *# Создаем фрейм-контейнер* container = tk.Frame(self.main\_window)  
 container.pack(side=tk.BOTTOM, fill=tk.BOTH, expand=True, padx=20, pady=20)  
   
 *# Создаем Canvas с адаптивным размером* self.canvas = tk.Canvas(container)  
 self.canvas.pack(side=tk.TOP, anchor=tk.N)  
   
 *# Функция для изменения размера canvas при изменении окна* def resize\_canvas(event=None):  
 width = container.winfo\_width() - 40 *# Учитываем отступы* height = container.winfo\_height() - 40  
 size = min(width, height) *# Квадратный canvas* self.canvas.config(width=size, height=size)  
   
 *# Привязываем функцию изменения размера к изменению размера контейнера* container.bind('<Configure>', resize\_canvas)  
   
 *# Передаем canvas вместо main\_window* self.game = Game(self.canvas, X\_SIZE, Y\_SIZE)  
  
 def exit\_game(self):  
 self.main\_window.destroy()  
  
 def show\_rules(self):  
 rules\_window = tk.Toplevel(self.main\_window)  
 rules\_window.title("Правила канадских шашек")  
 rules\_window.attributes('-fullscreen', True)  
   
 *# Создание основного контейнера с отступами* main\_frame = tk.Frame(rules\_window, bg='#f0f0f0')  
 main\_frame.pack(fill='both', expand=True, padx=50, pady=30)  
   
 *# Создание верхней панели с заголовком и кнопкой закрытия* top\_frame = tk.Frame(main\_frame, bg='#f0f0f0')  
 top\_frame.pack(fill='x', pady=(0, 20))  
   
 *# Заголовок (слева)* title = tk.Label(top\_frame,   
 text="Правила канадских шашек",   
 font=("Arial", 24, "bold"),  
 bg='#f0f0f0',  
 fg='#2c3e50')  
 title.pack(side='left')  
  
 *# Кнопка закрытия (справа)* close\_button = tk.Button(  
 top\_frame,  
 text="✕",  
 command=lambda: on\_closing(),  
 font=("Arial", 16, "bold"),  
 bg="#e74c3c",  
 fg="white",  
 width=3,  
 height=1,  
 relief=tk.FLAT,  
 activebackground="#c0392b",  
 activeforeground="white"  
 )  
 close\_button.pack(side='right')  
  
 *# Создание фрейма для текста с прокруткой* text\_container = tk.Frame(main\_frame, bg='#f0f0f0')  
 text\_container.pack(fill='both', expand=True)  
  
 *# Добавление прокрутки* scrollbar = tk.Scrollbar(text\_container)  
 scrollbar.pack(side='right', fill='y')  
  
 *# Создание текстового виджета с правилами* rules\_text = tk.Text(text\_container,  
 wrap=tk.WORD,  
 font=("Arial", 14),  
 padx=20,  
 pady=20,  
 bg='white',  
 fg='#2c3e50',  
 spacing1=10, *# Отступ перед абзацем* spacing2=2, *# Межстрочный интервал* spacing3=10) *# Отступ после абзаца* rules\_text.pack(fill='both', expand=True)  
  
 *# Привязка прокрутки к тексту* scrollbar.config(command=rules\_text.yview)  
 rules\_text.config(yscrollcommand=scrollbar.set)  
  
 *# Текст правил с форматированием* rules\_content = """  
 ПРАВИЛА ИГРЫ В КАНАДСКИЕ ШАШКИ  
  
 Основные положения:  
 • Игра ведется на доске размером 12×12 клеток  
 • Каждый игрок начинает с 30 шашками  
 • Первый ход делают белые шашки  
  
 Правила передвижения:  
 1. Простая шашка ходит только вперед по диагонали на одну клетку.  
  
 2. При достижении последней горизонтали простая шашка превращается в дамку.  
  
 3. Дамка может ходить на любое количество клеток по диагонали как вперед, так и назад.  
  
 Правила взятия:  
 1. Взятие обязательно! Если есть возможность взять шашку противника, вы обязаны это сделать.  
  
 2. При наличии нескольких вариантов взятия нужно выбрать тот, где будет взято наибольшее количество шашек противника.  
  
 3. Взятие происходит через одну клетку по диагонали с перескоком через шашку противника.  
  
 4. Взятые шашки снимаются с доски только после завершения полного хода.  
  
 5. Простая шашка может бить как вперед, так и назад.  
  
 Особые правила:  
 • Если простая шашка во время взятия достигает последней горизонтали, но еще может продолжить взятие, она остается простой шашкой до завершения взятия.  
  
 • "Турецкий удар" запрещен - нельзя дважды перепрыгивать через одну и ту же шашку противника.  
  
 Окончание игры:  
 Победа присуждается игроку, который:  
 • Уничтожил все шашки противника  
 • Или лишил их возможности хода ("запер")  
  
 Подсчет очков:  
 • За взятие простой шашки: 1 очко  
 • За взятие дамки: 3 очка  
 """  
  
 *# Вставка текста и установка режима "только для чтения"* rules\_text.insert('1.0', rules\_content)  
 rules\_text.config(state='disabled')  
  
 *# Создание нижней панели с кнопкой* bottom\_frame = tk.Frame(main\_frame, bg='#f0f0f0')  
 bottom\_frame.pack(fill='x', pady=(20, 0))  
  
 *# Кнопка закрытия* close\_button = tk.Button(  
 bottom\_frame,  
 text="Закрыть правила",  
 command=rules\_window.destroy,  
 font=("Arial", 12),  
 bg="#e74c3c",  
 fg="white",  
 padx=20,  
 pady=10,  
 relief=tk.FLAT,  
 activebackground="#c0392b",  
 activeforeground="white"  
 )  
 close\_button.pack(pady=10)  
  
 *# Обработчик события закрытия окна* def on\_closing():  
 self.main\_window.deiconify() *# Показываем главное окно* rules\_window.destroy()  
  
 *# Скрываем главное окно* self.main\_window.withdraw()  
   
 *# Привязываем обработчик к закрытию окна* rules\_window.protocol("WM\_DELETE\_WINDOW", on\_closing)  
 rules\_window.bind('<Escape>', lambda e: on\_closing())  
 def start\_game\_man(self):  
 self.main\_window.destroy()  
 self.start\_game()  
 def surrender(self):  
 *"""Обработка сдачи игры"""  
 # Показываем диалоговое окно с подтверждением* answer = messagebox.askyesno(  
 "Подтверждение",   
 "Вы действительно хотите сдаться?",  
 icon='question'  
 )  
   
 *# Если игрок подтвердил сдачу* if answer:  
 winner = "Чёрные" if self.game.current\_player == SideType.WHITE else "Белые"  
 messagebox.showinfo(  
 "Конец игры",   
 f"{winner} выиграли!"  
 )  
 *# Начинаем новую игру* self.start\_game\_man()  
 def start\_game(self):  
 *# Создание окна игры* self.main\_window = tk.Tk()  
 self.main\_window.title('Канадские шашки')  
 self.main\_window.attributes("-fullscreen", True)  
 self.main\_window.configure(bg='#f5f6fa') *# Светло-серый фон  
   
 # Создаем фрейм-контейнер* container = tk.Frame(self.main\_window, bg='#f5f6fa')  
 container.pack(side=tk.BOTTOM, fill=tk.BOTH, expand=True, padx=20, pady=20)  
   
 *# Создаем верхнюю панель* top\_frame = tk.Frame(container, bg='#f5f6fa', height=50)  
 top\_frame.pack(side=tk.TOP, fill=tk.X)  
 top\_frame.pack\_propagate(False)  
  
 *# Информация о текущем ходе* current\_turn\_var = tk.StringVar(value="Ход: Белые")  
 current\_turn = tk.Label(top\_frame,   
 textvariable=current\_turn\_var,   
 font=("Arial", 18, "bold"),  
 bg='#f5f6fa',  
 fg='#2f3542')  
 current\_turn.pack(pady=10)  
  
 *# Создаем нижний контейнер для доски и правой панели* bottom\_container = tk.Frame(container, bg='#f5f6fa')  
 bottom\_container.pack(fill=tk.BOTH, expand=True)  
  
 *# Создаем фрейм для доски с белым фоном и рамкой* game\_frame = tk.Frame(bottom\_container,   
 bg='white',  
 highlightbackground='#2c3e50',  
 highlightthickness=2)  
 game\_frame.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)  
  
 *# Создаем правую панель* right\_panel = tk.Frame(bottom\_container, bg='#f5f6fa', width=300)  
 right\_panel.pack(side=tk.RIGHT, fill=tk.Y)  
 right\_panel.pack\_propagate(False)  
  
 *# Создаем Canvas с адаптивным размером* main\_canvas = tk.Canvas(game\_frame)  
 main\_canvas.pack(expand=True)  
  
 *# Функция для изменения размера canvas* def resize\_canvas(event=None):  
 *# Получаем доступное пространство* frame\_width = game\_frame.winfo\_width()  
 frame\_height = game\_frame.winfo\_height()  
   
 *# Вычисляем размер стороны квадратного canvas* size = min(frame\_width, frame\_height) - 40 *# Отступ по 20 пикселей с каждой стороны  
   
 # Обновляем размеры canvas* main\_canvas.config(width=size, height=size)  
   
 *# Обновляем размер ячейки* global CELL\_SIZE  
 CELL\_SIZE = size // 12 *# Делим на количество клеток (12x12)  
   
 # Перерисовываем игровое поле* if hasattr(self, 'game'):  
 self.game.init\_images() *# Обновляем размеры изображений* self.game.draw() *# Перерисовываем поле  
  
 # Привязываем функцию к изменению размера фрейма* game\_frame.bind('<Configure>', resize\_canvas)  
  
 *# Очки в правой панели* score\_frame = tk.Frame(right\_panel, bg='#f5f6fa')  
 score\_frame.pack(pady=20)  
  
 score\_style = {  
 'font': ("Arial", 14),  
 'bg': '#f5f6fa',  
 'fg': '#2f3542'  
 }  
  
 white\_score\_var = tk.StringVar(value="Очки белых: 0")  
 black\_score\_var = tk.StringVar(value="Очки черных: 0")  
   
 white\_score = tk.Label(score\_frame, textvariable=white\_score\_var, \*\*score\_style)  
 white\_score.pack(pady=5)  
   
 black\_score = tk.Label(score\_frame, textvariable=black\_score\_var, \*\*score\_style)  
 black\_score.pack(pady=5)  
  
 *# Кнопки в правой панели* buttons\_frame = tk.Frame(right\_panel, bg='#f5f6fa')  
 buttons\_frame.pack(pady=20)  
  
 button\_style = {  
 'font': ("Arial", 12),  
 'width': 15,  
 'relief': tk.FLAT,  
 'cursor': 'hand2'  
 }  
  
 new\_game\_btn = tk.Button(buttons\_frame,   
 text="Новая игра",   
 command=lambda: self.start\_game\_man(),  
 bg="#70a1ff",  
 fg="white",  
 activebackground="#1e90ff",  
 \*\*button\_style)  
 new\_game\_btn.pack(pady=5)  
  
 surrender\_btn = tk.Button(buttons\_frame,   
 text="Сдаться",   
 command=lambda: self.surrender(),  
 bg="#ff7f50",  
 fg="white",  
 activebackground="#ff6348",  
 \*\*button\_style)  
 surrender\_btn.pack(pady=5)  
  
 exit\_btn = tk.Button(buttons\_frame,   
 text="Выход",   
 command=lambda: self.exit\_game(),  
 bg="#ff4757",  
 fg="white",  
 activebackground="#ff6b81",  
 \*\*button\_style)  
 exit\_btn.pack(pady=5)  
  
 *# Создаем новую игру с новым canvas* self.game = Game(main\_canvas, X\_SIZE, Y\_SIZE)  
  
 *# Обновление информации об игре* def update\_game\_info():  
 current\_turn\_var.set(f"Ход: {'Белые' if self.game.current\_player == SideType.WHITE else 'Черные'}")  
 white\_score\_var.set(f"Очки белых: {self.game.white\_points}")  
 black\_score\_var.set(f"Очки черных: {self.game.black\_points}")  
 self.main\_window.after(100, update\_game\_info)  
  
 *# Привязка событий* main\_canvas.bind("<Motion>", self.game.mouse\_move)  
 main\_canvas.bind("<Button-1>", self.game.mouse\_down)  
  
 *# Запуск обновления информации* update\_game\_info()  
  
 self.main\_window.mainloop()  
  
 *# Обновление информации об игре* def update\_game\_info():  
 current\_turn\_var.set(f"Ход: {'Белые' if self.game.current\_player == SideType.WHITE else 'Черные'}")  
 white\_score\_var.set(f"Очки белых: {self.game.white\_points}")  
 black\_score\_var.set(f"Очки черных: {self.game.black\_points}")  
 self.main\_window.after(100, update\_game\_info)  
  
 *# Привязка событий* main\_canvas.bind("<Motion>", self.game.mouse\_move)  
 main\_canvas.bind("<Button-1>", self.game.mouse\_down)  
  
 *# Запуск обновления информации* update\_game\_info()  
  
 self.main\_window.mainloop()  
 def draw\_gui(self):  
 *"""Отрисовка главного меню"""* self.main\_window.title("Канадские шашки")  
 self.main\_window.attributes("-fullscreen", True)  
 self.main\_window.configure(bg='#f5f6fa') *# Светло-серый фон  
  
 # Создаем основной контейнер с отступами* main\_container = tk.Frame(self.main\_window, bg='#f5f6fa')  
 main\_container.pack(expand=True)  
  
 *# Заголовок игры* title\_label = tk.Label(main\_container,   
 text="Канадские шашки",   
 font=("Arial", 36, "bold"),  
 fg='#2f3542',  
 bg='#f5f6fa')  
 title\_label.pack(pady=(50, 30))  
  
 *# Создаем рамку для кнопок* button\_frame = tk.Frame(main\_container, bg='#f5f6fa')  
 button\_frame.pack(pady=20)  
  
 *# Стиль для кнопок* button\_style = {  
 'font': ("Arial", 16),  
 'width': 20,  
 'height': 2,  
 'relief': tk.FLAT,  
 'border': 0,  
 'cursor': 'hand2' *# Курсор в виде руки при наведении* }  
  
 *# Кнопка "Игра с человеком"* play\_button = tk.Button(button\_frame,  
 text="Игра с человеком",  
 command=self.start\_game\_man,  
 bg='#70a1ff', *# Голубой* fg='white',  
 activebackground='#1e90ff',  
 \*\*button\_style)  
 play\_button.pack(pady=10)  
  
 *# Кнопка "Правила"* rules\_button = tk.Button(button\_frame,  
 text="Правила",  
 command=self.show\_rules,  
 bg='#ff7f50', *# Коралловый* fg='white',  
 activebackground='#ff6348',  
 \*\*button\_style)  
 rules\_button.pack(pady=10)  
  
 *# Кнопка "Выход"* exit\_button = tk.Button(button\_frame,  
 text="Выход",  
 command=self.exit\_game,  
 bg='#ff4757', *# Красный* fg='white',  
 activebackground='#ff6b81',  
 \*\*button\_style)  
 exit\_button.pack(pady=10)  
  
*# Запуск интерфейса авторизации*auth\_gui()