МИНИСТЕРСТВО ВЫСШЕГО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«Ульяновский государственный технический университет»

«Факультет информационных систем и технологий»

«Измерительно-вычислительные комплексы»

Дисциплина   
**«Алгоритмы и структуры данных»**

09.03.02 Информационные системы и технологии

Курсовая работа

**Разработка компьютерной логической игры «Русские циклические шашки»**

|  |  |  |
| --- | --- | --- |
| Научный руководитель:  Кандидат технических наук, доцент кафедры ИВК  Шишкин Вадим Викторинович |  | Выполнил: студент 2 курса, очной формы обучения,  группы ИСТбд-21  Костригин Всеволод Дмитриевич |

Ульяновск

2024

Оглавление

[ТЕХНИЧЕСКОЕ ЗАДАНИЕ 3](#_Toc185762324)

[ПОЯСНИТЕЛЬНАЯ ЗАПИСКА 7](#_Toc185762325)

[РУКОВОДСТВО ПРОГРАММИСТА 13](#_Toc185762326)

[ТЕСТОВАЯ ДОКУМЕНТАЦИЯ 17](#_Toc185762327)

[ЛИСТИНГ КОДА 22](#_Toc185762328)

[ЗАКЛЮЧЕНИЕ 37](#_Toc185762329)

[ИСТОЧНИКИ ИСПОЛЬЗОВАННЫЕ 39](#_Toc185762330)

[ПРИЛОЖЕНИЕ 1: MINDMAP 40](#_Toc185762331)

УТВЕРЖДЕНО

Распоряжением по кафедре ИВК

№\_\_\_ от «\_\_» \_\_\_\_\_\_\_\_\_\_\_\_2024

# ТЕХНИЧЕСКОЕ ЗАДАНИЕ

на выполнение курсовой работы

по дисциплине «Алгоритмы и структуры данных»

Специальность 09.03.02 – «Информационные системы и технологии»

Тема: «**Разработка компьютерной логической игры "Русские циклические шашки"»**

1. **Введение**

Настоящее техническое задание распространяется на разработку программы для моделирования и управления игровым процессом "Русские циклические шашки" с использованием графического интерфейса. Требуется разработать программу, обеспечивающую основные функции игры: ввод ходов, проверку корректности по правилам, а также возможность игры против другого игрока. Программа должна автоматически определять победу или ничью и визуализировать текущее состояние игрового поля. Интерфейс будет интуитивно понятным, обеспечивая пользователям удобное взаимодействие и увлекательный игровой процесс.

1. **Основания для разработки:**

Разработка осуществляется на основании учебного плана направления «Информационные системы и технологии».

1. **Назначения разработки:**

Программа предназначена для обеспечения игрового процесса "Русские циклические шашки", предоставляя выполнения ходов с учетом правил циклических шашек, автоматической проверки победителя и объявления результатов партии.

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

Приложение должно быть разработано в виде 1 модуля с дополнительными информационными файлами.

Приложение должно соответствовать следующим правилам.

**Поле и игроки:**

Игра проводится на квадратном поле размером 8×8 клеток. Поле раскрашено в черно-белую шахматную раскраску. В игре принимают участие два игрока, каждый из которых управляет шашками своего цвета: белые и чёрные. В начале игры у каждого игрока есть по 12 шашек, которые располагаются на первых трех горизонталях своего игрового поля.

**Типы ходов в игре «Русские циклические шашки»:**

1. Перемещение на соседнюю клетку:

Игрок может переместить шашку на свободную клетку, находящуюся по диагонали. Белые шашки двигаются вверх, черные – вниз. Движение по прямым линиям запрещено.

1. Перепрыгивание через шашки:

Игрок может перепрыгивать через одну или несколько шашек (своих или противника), если за ними есть свободная клетка. Перепрыгивание возможно только по диагонали. Во время одного хода можно выполнить несколько прыжков подряд, если это позволяет позиция.

**Порядок ходов:**

Ходы совершаются поочередно. Первым ходит игрок с белыми шашками. Во время каждого хода игрок может перемещать только одну шашку

**Цель игры:**

Основная цель – уничтожить все шашки соперника или заблокировать их так, чтобы он не мог сделать ход. Победа достигается, если соперник не может выполнить ни одного действия на своем ходу.

**Завершение игры:**

Игра завершается, когда один из игроков полностью лишен возможности совершить ход. Побеждает тот, кто первым уничтожил или заблокировал все шашки противника. В случае, если на поле не осталось активных шашек у одного из игроков, другой автоматически признается победителем.

* + 1. Программа должна обеспечивать выполнение следующих функций:
       - * Моделирование игры в «Русские циклические шашки» с визуализацией игрового процесса на экране.
         * Ходы игроков по правилам «Русских циклических шашек».
         * Проверка победителя или ничьей.
         * Обновление доски после каждого хода.
         * Отображение сообщений о результате игры.
         * Возможность выхода из приложения.
    2. Организация входных данных и выходных данных:
       - * Входные данные: начальная позиция фигур, ходы пользователя.
         * Выходные данные: визуализация текущего состояния доски.
  1. Требования к надежности:  
     Программа "Русские циклические шашки" должна обеспечивать корректное восстановление после сбоев. В случае неожиданного завершения игры при повторном запуске пользователь должен иметь возможность начать новую партию без ошибок и с чистым игровым полем.
  2. Требования к составу и параметрам технических средств:  
     Программа "Русские циклические шашки" должна корректно работать на персональных компьютерах, поддерживающих Python версии 3.8 и выше, а также иметь установленный модуль Tkinter для реализации графического интерфейса
  3. Требования к информационной и программной совместимости:
     + - Программа "Русские циклические шашки" должна функционировать под управлением операционных систем, поддерживающих Python версии 3.8 и выше.
       - Для реализации графического интерфейса в программе должна использоваться библиотека Tkinter.
  4. Условия хранения: обеспечение свободного доступа к проекту в репозитории до окончания срока учебы.
  5. Срок хранения: до окончания срока учебы

1. **Требования к программной документации**

В состав программной документации должны входить:

* Техническое задание.
* Пояснительная записка.
* Руководство программиста.
* Тестовая документация

1. **Стадии и этапы разработки**

* Планирование

Определение целей проекта и разработка требований к функциональности игры "Русские циклические шашки".

* Проектирование

Разработка архитектуры программы, выбор библиотек и инструментов для реализации графического интерфейса и логики игры.

* Разработка

Написание кода игры с использованием библиотеки tkinter, реализация функционала для взаимодействия с пользователем и логики игры.

* Тестирование

Проверка программы на наличие ошибок, тестирование взаимодействия игрока и игры, обеспечение корректного функционирования всех игровых процессов.

* Документирование

Подготовка технического задания, руководство программиста и пояснительной записки для полного описания всех аспектов программы.

1. **Порядок контроля и приемки**
   1. Контроль

Проводится тестирование программы "Русские циклические шашки", включая проверку всех функциональных возможностей, стабильности работы, корректного отображения графического интерфейса и взаимодействия с игроком.

* 1. Приемка проекта

Завершающая приемка проходит в рамках лабораторной работы, в ходе которой повторно проверяется соответствие всех функций заявленным требованиям, оценивается качество реализации и интерфейса, а также общее соответствие целям и задачам проекта.

УТВЕРЖДЕНО

Распоряжением по кафедре ИВК

№\_\_\_ от «\_\_» \_\_\_\_\_\_\_\_\_\_\_\_2024

# ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

на выполнение курсовой работы

по дисциплине «Алгоритмы и структуры данных»

Специальность 09.03.02 – «Информационные системы и технологии»

Тема: «**Разработка компьютерной логической игры "Русские циклические шашки"»**

**Введение**

Разрабатываемое приложение представляет собой компьютерную логическую игру «Русские циклические шашки», созданную с использованием языка программирования Python и библиотеки tkinter для графического интерфейса.

Целью приложения является предоставление пользователям возможности играть в шашки с уникальными правилами, включая использование резерва шашек. Программа демонстрирует навыки программирования, проектирования пользовательских интерфейсов и реализации алгоритмов игровой логики.

**Основания для разработки:**

Задание для курсового проекта по учебному плану направления «Информационные системы и технологии».

**Постановка задачи:**

Цель разработки заключается в создании компьютерной логической игры «Русские циклические шашки», которая включает в себя:

* Регистрация и авторизация в системе с помощью имени и пароля
* Интерактивная и динамичная инструкция к игре.

**Структура программы:  
Классы:**

1. **CheckersGame:** Этот класс отвечает за логику и интерфейс игры в шашки. Он управляет игровым процессом, включая создание доски, обработку ходов, проверку правил игры, таких как обязательное взятие, и определение победителя.
2. **MenuWindow:** Этот класс создает главное меню приложения. Он предоставляет пользователю возможность выбрать между просмотром правил игры и началом новой игры.
3. **LoginWindow:** Этот класс отвечает за авторизацию пользователя. Он предоставляет интерфейс для ввода имени пользователя и пароля, а также функции для регистрации нового пользователя. После успешного входа в систему или регистрации, он открывает главное меню

**Основные функции:**

1. **\_\_init\_\_:** Конструкторы классов, которые инициализируют объекты и создают начальные элементы интерфейса
2. **create\_board(self):** Создает и возвращает начальную конфигурацию доски для игры в шашки.
3. **draw\_board:** Отображает игровую доску и шашки на холсте, обновляя графический интерфейс.
4. **click:** Обрабатывает события кликов мыши на игровом поле, определяя действия в зависимости от текущего состояния игры.
5. **move\_piece:** Перемещает шашку с одной позиции на другую, проверяя корректность хода и обновляя состояние игры.
6. **is\_valid\_move:** Проверяет, является ли предложенный ход допустимым в соответствии с правилами игры.
7. **must\_capture:** Определяет, есть ли у текущего игрока обязательные взятия.
8. **can\_capture:** проверяет, может ли выбранная шашка выполнить взятие.
9. **switch\_turn:** Переключает ход между игроками.
10. **check\_game\_over:** проверяет условия окончания игры и отображает соответствующее сообщение, если игра завершена.
11. **show\_game\_over:** Отображает окно с сообщением об окончании игры.
12. **new\_game:** Сбрасывает состояние игры для начала новой партии.
13. **login:** Обрабатывает вход пользователя, проверяя введенные данные с сохраненными учетными записями.
14. **register:** Обрабатывает регистрацию нового пользователя, добавляя его данные в файл учетных записей.
15. **show\_rules:** Отображает окно с правилами игры.
16. **start\_game:** Запускает новую игру, открывая окно с игрой в шашки.

**Проектная часть:**

* 1. **Математические методы:**
* Оценочная функция вычисление для определения возможности хода
* Оценочная функция позиции
  1. **Архитектура и алгоритмы**
     1. **Архитектура:**
* **Игровое поле –** доска в черно-белую клетку 8х8
* **Шашки –** объекты, которые могут ходить только по правилам
* **Интерфейс –** кнопки для управления и информационные панели
* **Система авторизации –** хеширование паролей
  + 1. **Алгоритмы:**
* Алгоритм проверки допустимости хода
* Алгоритм взятия шашек
* Алгоритм переключения хода
* Алгоритм проверки окончания игры
* Алгоритм управления резервом шашек
* Алгоритм отображения доски
* Алгоритм авторизации и регистрации
  1. **Стадии и этапы разработки:**
     1. Изучение правил
     2. Проектирование архитектуры
     3. Реализация механик
     4. Тестирование механик
     5. Реализация регистрации и авторизации
     6. Тестирование и отладка
     7. Документирование результата
  2. **Тестирование** 
     1. **Цель тестирования:** Проверка корректности работы всех игровых механик и пользовательского интерфейса.
     2. **Метод тестирования:** Тестирование проводилось методом "черного ящика" с проверкой всех возможных сценариев использования.
     3. **Сценарии для тестирования:**
* Проверка интерфейса и правильности ходов шашек
* Проверка определения дамки
* Проверка взаимодействия с резервом
* Проверка определения победителя
* Проверка авторизации/регистрации
* Стресс-тестирование

Ошибки, которые были обнаружены при работе приложения, исправлены. Программа работает правильно и стабильно.

**Использованные методы и алгоритмы**

В данном проекте реализованы различные методы и алгоритмы, которые обеспечивают функциональность игры в шашки. Одним из ключевых алгоритмов является проверка допустимости хода, реализованная в методе **is\_valid\_move**. Этот алгоритм проверяет, соответствует ли предложенный ход правилам игры, включая движение по диагонали, наличие препятствий и возможность взятия.

Алгоритм взятия шашек, представленный в методах **can\_capture** и **must\_capture**, определяет, может ли шашка выполнить «поедание», и есть ли у игрока обязательные взятия. Это включает проверку соседних клеток и наличие противника, что позволяет игроку принимать стратегические решения.

Алгоритм переключения хода, реализованный в методе **switch\_turn**, обеспечивает чередование ходов между игроками, что является основой для поддержания игрового процесса.

Отображение доски, реализованное в методе **draw\_board**, отвечает за визуализацию игрового поля и шашек, включая выделение возможных ходов и взятий. Это позволяет игрокам легко ориентироваться в текущем состоянии игры.

Алгоритм проверки окончания игры, представленный в методе **check\_game\_over**, определяет, завершена ли игра, проверяя количество оставшихся шашек и наличие возможных ходов для текущего игрока. Это позволяет своевременно определить победителя и завершить игру.

Алгоритмы управления резервом шашек обрабатывают добавление и размещение шашек из резерва, включая проверку возможности размещения и обновление интерфейса, что добавляет дополнительный стратегический элемент в игру.

Эти алгоритмы обеспечивают основную функциональность игры, включая управление игровым процессом, проверку правил и взаимодействие с пользователем через графический интерфейс, что делает игру в шашки увлекательной и динамичной.

Управление состоянием игры в данном проекте включает в себя отслеживание текущего хода, управление резервом шашек и обработку особых ситуаций, таких как обязательное взятие и завершение игры. Система переключения хода между игроками обеспечивает плавный игровой процесс, а алгоритмы проверки допустимости и обязательности взятия гарантируют соблюдение правил игры.

Также предусмотрено управление резервом шашек, что добавляет игре стратегической глубины и позволяет игрокам использовать дополнительные ресурсы в критические моменты. Проверка окончания игры, включая определение победителя и отсутствие возможных ходов, обеспечивает своевременное завершение партии

**Заключение:**

Разработанная программа полностью соответствует поставленным задачам и обеспечивает возможность игры в русские циклические шашки. Программа имеет удобный пользовательский интерфейс, реализованный с помощью **tkinter**, и предоставляет пользователям возможность легко взаимодействовать с игрой

В заключение, данный проект представляет собой комплексную реализацию игры в шашки, в которой гармонично сочетаются различные алгоритмы и методы. Алгоритмы проверки допустимости и обязательности взятия обеспечивают соблюдение правил игры, а управление резервом шашек добавляет стратегической глубины. Система аутентификации и регистрации пользователей обеспечивает безопасность и удобство, позволяя сохранять учетные данные.

Графический интерфейс делает игру визуально привлекательной и интуитивно понятной, а алгоритмы управления игровым процессом обеспечивают динамичное и увлекательное взаимодействие. Все эти компоненты вместе создают захватывающий и реалистичный опыт игры в шашки, подходящий как для новичков, так и для опытных игроков.

# РУКОВОДСТВО ПРОГРАММИСТА

на выполнение курсовой работы

по дисциплине «Алгоритмы и структуры данных»

Специальность 09.03.02 – «Информационные системы и технологии»

Тема: «**Разработка компьютерной логической игры "Русские циклические шашки"»**

**1. Назначение и условия применения программы**

**1.1. Назначение программы:**

Программа представляет собой симулятор игры «Русские циклические шашки». Она визуализирует доску, шашки и позволяет играть против другого игрока.

**1.2. Функции программы:**

Программа выполняет следующие функции:

• Визуализация доски и шашек

• Симуляция партии

• Управление ходами шашек

• Обработка взаимодействия между шашками

• Авторизация и регистрация пользователей

**1.3. Условия применения:**

Для использования программы необходимы:

• Python версии 3.x

• Библиотека **tkinter**

• Файл данных пользователей (**accounts**)

1. **Характеристика программы**
   1. **Общие сведения:**

Программа реализована с использованием объектно-ориентированного подхода. Основными классами являются **CheckersGame** и **MenuWindow**. Первый класс обеспечивает соблюдение логики и отрисовку доски с шашками, второй – создание и работа меню

* 1. **Основные характеристики:**

Программа использует графический интерфейс на основе библиотеки **tkinter**. Игра происходит в реальном времени, где отображаются все элементы. Движение шашек и обработка ходов происходят в реальном времени.

1. **Обращение к программе**
   1. **Класс CheckersGame:**
      1. **Метод** **\_\_init\_\_(self, root)**

**Назначение:** Инициализация объекта игры в шашки, создание интерфейса и начальной конфигурации доски.

* + 1. **Метод create\_board(self)**

**Назначение:** Создание и возврат начальной конфигурации доски.

* + 1. **Метод draw\_board(self)**

**Назначение:** Отображение игрового поля и шашек на холсте.

* + 1. **Метод click(self, event)**

**Назначение:** Обработка кликов на доске, определение действий в зависимости от состояния игры.

* + 1. **Метод move\_piece(self, start, end)**

**Назначение:** Перемещение шашки с одной позиции на другую, проверка корректности хода.

* + 1. **Метод is\_valid\_move(self, start, end)**

**Назначение:** Проверка, является ли предложенный ход допустимым.

* + 1. **Метод must\_capture(self)**

**Назначение:** Определение, есть ли у игрока обязательные взятия.

* + 1. **Метод can\_capture(self, pos)**

**Назначение:** Проверка, может ли выбранная шашка выполнить взятие.

* + 1. **Метод switch\_turn(self)**

**Назначение:** Переключение хода между игроками.

* + 1. **Метод check\_game\_over(self)**

**Назначение:** Проверка условий окончания игры и определение победителя.

* + 1. **Метод show\_game\_over(self, message)**

**Назначение:** Отображение окна с сообщением об окончании игры.

* + 1. **Метод new\_game(self, game\_over\_window=None)**

**Назначение:** Сброс состояния игры для начала новой партии.

* 1. **Класс MenuWindow:**
     1. **Метод \_\_init\_\_(self)**

**Назначение:** Инициализация меню, создание интерфейса с кнопками для выбора действий.

* + 1. **Метод show\_rules(self)**

**Назначение:** Отображение окна с правилами игры.

* + 1. **Метод start\_game(self)**

**Назначение:** Запуск новой игры, открытие окна с игрой в шашки.

* 1. **Класс LoginWindow:**
     1. **Метод \_\_init\_\_(self)**

**Назначение:** Инициализация окна авторизации, создание интерфейса для ввода данных пользователя.

* + 1. **Метод register(self)**

**Назначение:** Обработка регистрации нового пользователя, добавление данных в файл учетных записей.

* + 1. **Метод login(self)**

**Назначение:** Обработка входа пользователя, проверка введенных данных с сохраненными учетными записями.

* + 1. **Метод start\_game(self)**

**Назначение:** Открытие меню напрямую после успешной авторизации.

1. **Входные и выходные данные**
   1. **Входные данные:**
      1. Имя пользователя и пароль
      2. Действия пользователя
      3. Файл учетных записей
   2. **Выходные данные:**
      1. Визуальное отображение доски и шашек
      2. Сообщение о результате игры (победа Черных, победа белых и т.д.)
2. **Сообщения:**

Программа выводит следующие сообщения:

* 1. Сообщение об ошибки или успешной регистрации.
  2. Сообщение при попытке выполнить недопустимые действия.
  3. Сообщение о результате игры.

1. **Используемы технические средства:**

Для работы программы используются Python библиотека **tkinter** для создания графического интерфейса и обработки событий

1. **Особенности реализации:**
   1. Программа использует объектно-ориентированный подход для моделирования поведения шашек и доски.
   2. Графический интерфейс включает в себя доску, панель управления и область для отображения сообщений.
   3. Реализована возможность игры против другого игрока.
   4. Поддерживается авторизация и регистрация пользователей.

# ТЕСТОВАЯ ДОКУМЕНТАЦИЯ

на выполнение курсовой работы

по дисциплине «Алгоритмы и структуры данных»

Специальность 09.03.02 – «Информационные системы и технологии»

Тема: «**Разработка компьютерной логической игры "Русские циклические шашки"»**

1. **Описание тестируемых функций:**В процессе тестирования игры необходимо проверить следующие функции:
   1. **click(event):**
      1. **Описание:** Обрабатывает клики на игровом поле, перемещает шашки и проверяет состояние игры.
      2. **Тестируемые моменты:**

* Корректное определение позиции клика.
* Корректное перемещение шашек.
* Проверка на обязательные взятия.
* Обновление состояния игры после хода.
* Корректное отображение возможных ходов.  
  1. **move\_piece(start, end):**
     1. **Описание:** Перемещает шашку с одной позиции на другую, проверяет корректность хода.
     2. **Тестируемые моменты:**
* Корректное перемещение шашки.
* Проверка на взятие шашек.
* Обновление доски после хода.
* Проверка на превращение в дамку.  
  1. **is\_valid\_move(start, end):**
     1. **Описание:** Проверяет, является ли предложенный ход допустимым.
     2. **Тестируемые моменты:**
* Корректная проверка допустимости хода.
* Проверка на наличие препятствий.
* Проверка на возможность взятия.
  1. **must\_capture():**
     1. **Описание:** Определяет, есть ли у игрока обязательные взятия.
     2. **Тестируемые моменты:**
* Корректное определение обязательных взятий.
* Проверка всех возможных взятий на доске.
  1. **can\_capture(pos):**
     1. **Описание:** Проверяет, может ли выбранная шашка выполнить взятие.
     2. **Тестируемые моменты:**
* Корректная проверка возможности взятия.
* Проверка всех направлений для взятия.
  1. **switch\_turn():**
     1. **Описание:** Переключает ход между игроками.
     2. **Тестируемые моменты:**
* Корректное переключение хода.
* Обновление интерфейса после переключения.  
  1. **check\_game\_over():**
     1. **Описание:** Проверяет условия окончания игры и определяет победителя.
     2. **Тестируемые моменты:**
* Корректное определение окончания игры.
* Проверка на отсутствие возможных ходов.
* Отображение сообщения о победителе.
  1. **login():**
     1. **Описание:** Обрабатывает вход пользователя, проверяет введенные данные.
     2. **Тестируемые моменты:**
* Корректная проверка логина и пароля.
* Обработка ошибок при неверных данных.
* Переход в главное меню после успешного входа.  
  1. **register():**
     1. **Описание:** Обрабатывает регистрацию нового пользователя.
     2. **Тестируемые моменты:**
* Корректная регистрация нового пользователя.
* Проверка на существование пользователя.
* Обработка ошибок при регистрации.  
  1. **draw\_board():**
     1. **Описание:** Отображает игровую доску и шашки на холсте.
     2. **Тестируемые моменты:**
* Корректное отображение доски и шашек.
* Подсветка возможных ходов.
* Обновление интерфейса после каждого хода.

1. **Mind map:**

Для тестирования игры можно организовать карту разума (mind map), чтобы структурировать процесс (Приложение 1)

1. **Чеклист:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Функция** | **Шаги выполнения** | **Ожидаемый результат** | **Результат прохождения** |
| 1 | **click(event)** | Нажатие на игровое поле | Обновление состояния игры после хода. | Пройдено/Не пройдено |
| 2 | **move\_piece(start, end)** | Перемещение шашки | Корректное перемещение шашки | Пройдено/Не пройдено |
| 3 | **is\_valid\_move(start, end)** | Проверка допустимости хода | Корректная проверка допустимости хода. | Пройдено/Не пройдено |
| 4 | **must\_capture()** | Определение обязательности взятия | Корректное определение обязательных взятий. | Пройдено/Не пройдено |
| 5 | **can\_capture(pos)** | Проверка шашки на выполнение взятие | Корректная проверка возможности взятия. | Пройдено/Не пройдено |
| 6 | **switch\_turn()** | Переключение ходов | Корректное переключение хода. | Пройдено/Не пройдено |
| 7 | **check\_game\_over()** | Проверка условий окончания игры | Корректное определение окончания игры. | Пройдено/Не пройдено |
| 8 | **login()** | Обработка входа в игру | Корректная проверка логина и пароля. | Пройдено/Не пройдено |
| 9 | **register()** | Обработка регистрации | Корректная регистрация нового пользователя. | Пройдено/Не пройдено |
| 10 | **draw\_board()** | Отображение доски и шашек | Корректное отображение доски и шашек. | Пройдено/Не пройдено |

1. **Набор тест-кейсов**
   1. **Тест-кейс 1:** Проверка нажатия на игровое поле
      1. **Цель:** Убедиться, что нажатие на игровое поле обрабатывается корректно.
      2. **Предусловия:** Игра запущена**.**
      3. **Шаги:**

* Нажать на игровое поле в клетку с шашкой текущего игрока.
* Нажать на допустимую клетку для перемещения.
* Проверить перемещение шашки и обновление состояния игры.
  + 1. **Ожидаемый результат:** Шашка перемещается, состояние игры обновляется. Отсутствие сбоев.
  1. **Тест-кейс 2: Проверка обязательного взятия**
     1. **Цель:** Убедиться, что обязательное взятие обрабатывается корректно.
     2. **Предусловия:** Игра запущена, на доске есть возможность обязательного взятия.
     3. **Шаги:**
* Нажать на шашку, которая может выполнить взятие.
* Нажать на клетку, куда можно выполнить взятие.
* Проверить, что взятие выполнено и шашка противника удалена.
  + 1. **Ожидаемый результат:** Взятие выполнено, шашка противника удалена. Отсутствие сбоев.
  1. **Тест-кейс 3: Проверка переключения хода**
     1. **Цель:** Убедиться, что ход переключается между игроками корректно.
     2. **Предусловия:** Игра запущена,
     3. **Шаги:**
* Выполнить ход текущим игроком.
* Проверить, что ход переключился на другого игрока.
  + 1. **Ожидаемый результат:** Ход переключается на другого игрока. Отсутствие сбоев.
  1. **Тест-кейс 4: Проверка окончания игры**
     1. **Цель:** Убедиться, что игра корректно завершает партию.
     2. **Предусловия:** Игра запущена, один из игроков не имеет возможных ходов.
     3. **Шаги:**
* Выполнить ход, который приводит к отсутствию возможных ходов у противника.
* Проверить, что игра завершена и отображено сообщение о победителе.
  + 1. **Ожидаемый результат:** Игра завершена, отображено сообщение о победителе. Отсутствие сбоев.
  1. **Тест-кейс 5: Проверка авторизации пользователя**
     1. **Цель:** Убедиться, что авторизация пользователя работает корректно.
     2. **Предусловия:** Приложение запущено, учетная запись существует.
     3. **Шаги:**
* Ввести корректные имя пользователя и пароль.
* Нажать кнопку "Вход".
* Проверить, что пользователь успешно авторизован и открыто главное меню.
  + 1. **Ожидаемый результат:** Ожидаемый результат: Пользователь авторизован, открыто главное меню. Отсутствие сбоев.
  1. **Тест-кейс 6: Проверка регистрации нового пользователя**
     1. **Цель:** Убедиться, что регистрация нового пользователя работает корректно.
     2. **Предусловия:** Приложение запущено.
     3. **Шаги:**
* Ввести новое имя пользователя и пароль.
* Нажать кнопку "Регистрация".
* Проверить, что учетная запись создана и отображено сообщение об успешной регистрации.
  + 1. **Ожидаемый результат:** Учетная запись создана, отображено сообщение об успешной регистрации. Отсутствие сбоев.

# 

# ЛИСТИНГ КОДА

на выполнение курсовой работы

по дисциплине «Алгоритмы и структуры данных»

Специальность 09.03.02 – «Информационные системы и технологии»

Тема: «**Разработка компьютерной логической игры "Русские циклические шашки"»**

import tkinter as tk  
  
class CheckersGame:  
 def \_\_init\_\_(self, root):  
 self.root = root  
 self.root.title("Русские циклические шашки")  
   
 *# Создаем фрейм для игрового поля и резерва* self.game\_frame = tk.Frame(root)  
 self.game\_frame.pack(side=tk.LEFT, padx=10)  
   
 *# Создаем холст для доски* self.canvas = tk.Canvas(self.game\_frame, width=400, height=400)  
 self.canvas.pack()  
   
 *# Создаем фрейм для резерва и кнопок* self.reserve\_frame = tk.Frame(root)  
 self.reserve\_frame.pack(side=tk.LEFT, padx=10)  
   
 *# Резерв для белых и черных шашек* self.white\_reserve = []  
 self.black\_reserve = []  
   
 *# Флаг для отслеживания режима размещения шашки из резерва* self.placing\_from\_reserve = False  
   
 *# Создаем кнопки резерва с изображением шашек  
 # Сначала черные* self.black\_reserve\_button = tk.Canvas(self.reserve\_frame, width=50, height=50)  
 self.black\_reserve\_button.pack(pady=5)  
 self.black\_reserve\_button.bind("<Button-1>", lambda e: self.start\_placing\_from\_reserve('b'))  
   
 *# Затем белые* self.white\_reserve\_button = tk.Canvas(self.reserve\_frame, width=50, height=50)  
 self.white\_reserve\_button.pack(pady=5)  
 self.white\_reserve\_button.bind("<Button-1>", lambda e: self.start\_placing\_from\_reserve('w'))  
   
 self.update\_reserve\_buttons()  
   
 self.board = self.create\_board()  
 self.selected\_piece = None  
 self.current\_turn = 'w'  
 self.undo\_button = tk.Button(self.game\_frame, text="Отменить", command=self.undo\_selection, state=tk.DISABLED)  
 self.undo\_button.pack()  
 self.draw\_board()  
 self.canvas.bind("<Button-1>", self.click)  
  
 def update\_reserve\_buttons(self):  
 *# Обновляем белую кнопку* self.white\_reserve\_button.delete("all")  
 self.white\_reserve\_button.create\_oval(5, 5, 45, 45, fill="white")  
 self.white\_reserve\_button.create\_text(25, 25, text=str(len(self.white\_reserve)))  
   
 *# Обновляем черную кнопку* self.black\_reserve\_button.delete("all")  
 self.black\_reserve\_button.create\_oval(5, 5, 45, 45, fill="black")  
 self.black\_reserve\_button.create\_text(25, 25, text=str(len(self.black\_reserve)), fill="white")  
  
 def start\_placing\_from\_reserve(self, color):  
 if color == self.current\_turn and not self.placing\_from\_reserve:  
 *# Проверяем, есть ли возможность обязательного взятия* if self.must\_capture():  
 *# Если есть возможность взятия, показываем сообщение* message\_window = tk.Toplevel(self.root)  
 message\_window.title("Внимание")  
 message\_window.geometry("300x100")  
   
 label = tk.Label(message\_window,   
 text="Вы обязаны выполнить взятие шашки противника!",   
 wraplength=250,  
 pady=20)  
 label.pack()  
   
 button = tk.Button(message\_window,   
 text="OK",   
 command=message\_window.destroy)  
 button.pack()  
   
 return  
   
 *# Если нет обязательного взятия, проверяем наличие шашек в резерве* if (color == 'w' and self.white\_reserve) or (color == 'b' and self.black\_reserve):  
 self.placing\_from\_reserve = True  
 self.selected\_piece = None *# Сбрасываем выбранную шашку* self.undo\_button.config(state=tk.NORMAL) *# Активируем кнопку отмены* self.highlight\_available\_cells\_for\_reserve()  
 self.draw\_board()  
  
 def highlight\_available\_cells\_for\_reserve(self):  
 self.canvas.delete("highlight")  
 start\_row = 4 if self.current\_turn == 'w' else 0  
 end\_row = 8 if self.current\_turn == 'w' else 4  
   
 for row in range(start\_row, end\_row):  
 for col in range(8):  
 if (row + col) % 2 == 1 and self.board[row][col] == ' ':  
 x1 = col \* 50  
 y1 = row \* 50  
 self.canvas.create\_rectangle(x1, y1, x1+50, y1+50,   
 outline="green", width=2, tags="highlight")  
  
 def update\_reserve\_display(self):  
 self.white\_reserve\_label.config(text=f"Белые в резерве: {len(self.white\_reserve)}")  
 self.black\_reserve\_label.config(text=f"Черные в резерве: {len(self.black\_reserve)}")  
  
 def can\_capture\_own\_piece(self, pos):  
 if self.must\_capture(): *# Если есть обязательное взятие, нельзя брать свои шашки* return False  
 x, y = pos  
 piece = self.board[x][y]  
 directions = [(-2, -2), (-2, 2), (2, -2), (2, 2)]  
 for dx, dy in directions:  
 nx, ny = x + dx, y + dy  
 if 0 <= nx < 8 and 0 <= ny < 8 and self.board[nx][ny] == ' ':  
 mx, my = x + dx // 2, y + dy // 2  
 if self.board[mx][my].lower() == self.current\_turn: *# Проверяем, что это наша шашка* return True  
 return False  
  
 def is\_valid\_move(self, start, end):  
 sx, sy = start  
 ex, ey = end  
 if self.board[ex][ey] != ' ':  
 return False  
 dx = ex - sx  
 dy = ey - sy  
 piece = self.board[sx][sy]  
   
 if piece.isupper(): *# Дамка* if abs(dx) == abs(dy):  
 step\_x = 1 if dx > 0 else -1  
 step\_y = 1 if dy > 0 else -1  
 x, y = sx + step\_x, sy + step\_y  
 enemy\_found = False  
 own\_piece\_found = False  
   
 if self.must\_capture(): *# Если есть возможность взятия* while x != ex and y != ey:  
 if self.board[x][y] != ' ':  
 if enemy\_found: *# Если уже нашли шашку противника* return False  
 if self.board[x][y].lower() != self.current\_turn: *# Если это шашка противника* enemy\_found = True  
 else: *# Если это наша шашка* return False  
 x += step\_x  
 y += step\_y  
 return enemy\_found *# Ход возможен только если взяли шашку противника* else: *# Обычный ход или взятие своей шашки* while x != ex and y != ey:  
 if self.board[x][y] != ' ':  
 if enemy\_found or own\_piece\_found:  
 return False  
 if self.board[x][y].lower() == self.current\_turn:  
 own\_piece\_found = True  
 else:  
 return False  
 x += step\_x  
 y += step\_y  
 return True  
 else: *# Обычная шашка* if self.must\_capture():  
 if abs(dx) == 2 and abs(dy) == 2:  
 mx, my = (sx + ex) // 2, (sy + ey) // 2  
 if self.current\_turn == 'w' and self.board[mx][my].lower() == 'b':  
 return True  
 if self.current\_turn == 'b' and self.board[mx][my].lower() == 'w':  
 return True  
 return False  
 else:  
 if abs(dx) == 2 and abs(dy) == 2:  
 mx, my = (sx + ex) // 2, (sy + ey) // 2  
 if self.board[mx][my].lower() == self.current\_turn: *# Взятие своей шашки* return True  
 if self.current\_turn == 'w' and dx == -1 and abs(dy) == 1:  
 return True  
 if self.current\_turn == 'b' and dx == 1 and abs(dy) == 1:  
 return True  
 return False  
  
 def move\_piece(self, start, end):  
 sx, sy = start  
 ex, ey = end  
 if self.is\_valid\_move(start, end):  
 piece = self.board[sx][sy]  
 self.board[ex][ey] = self.board[sx][sy]  
 self.board[sx][sy] = ' '  
  
 *# Проверяем взятие* if abs(ex - sx) == 2 or (piece.isupper() and abs(ex - sx) > 1):  
 if piece.isupper(): *# Для дамки* dx = 1 if ex > sx else -1  
 dy = 1 if ey > sy else -1  
 x, y = sx + dx, sy + dy  
 while x != ex and y != ey:  
 if self.board[x][y] != ' ':  
 taken\_piece = self.board[x][y]  
 if taken\_piece.lower() == self.current\_turn: *# Если взяли свою шашку* if self.current\_turn == 'w':  
 self.white\_reserve.append(taken\_piece)  
 else:  
 self.black\_reserve.append(taken\_piece)  
 self.board[x][y] = ' '  
 self.selected\_piece = (ex, ey)  
 if self.can\_capture((ex, ey)):  
 self.draw\_board()  
 self.update\_reserve\_buttons()  
 return  
 break  
 x += dx  
 y += dy  
 else: *# Для обычной шашки* mx, my = (sx + ex) // 2, (sy + ey) // 2  
 taken\_piece = self.board[mx][my]  
 if taken\_piece.lower() == self.current\_turn: *# Если взяли свою шашку* if self.current\_turn == 'w':  
 self.white\_reserve.append(taken\_piece)  
 else:  
 self.black\_reserve.append(taken\_piece)  
 self.board[mx][my] = ' '  
 self.selected\_piece = (ex, ey)  
 if self.can\_capture((ex, ey)):  
 self.draw\_board()  
 self.update\_reserve\_buttons()  
 return  
  
 *# Превращение в дамку* if (self.current\_turn == 'w' and ex == 0) or (self.current\_turn == 'b' and ex == 7):  
 self.board[ex][ey] = self.board[ex][ey].upper()  
   
 self.selected\_piece = None  
 self.undo\_button.config(state=tk.DISABLED)  
 self.switch\_turn()  
 self.draw\_board()  
 self.update\_reserve\_buttons()  
   
 *# Добавляем проверку окончания игры* self.check\_game\_over()  
  
 def create\_board(self):  
 board = [[' ' for \_ in range(8)] for \_ in range(8)]  
 for row in range(3):  
 for col in range(8):  
 if (row + col) % 2 == 1:  
 board[row][col] = 'b' *# Черные шашки* for row in range(5, 8):  
 for col in range(8):  
 if (row + col) % 2 == 1:  
 board[row][col] = 'w' *# Белые шашки* return board  
  
 def draw\_board(self):  
 self.canvas.delete("all")  
 for row in range(8):  
 for col in range(8):  
 x1 = col \* 50  
 y1 = row \* 50  
 x2 = x1 + 50  
 y2 = y1 + 50  
 color = "white" if (row + col) % 2 == 0 else "gray"  
 self.canvas.create\_rectangle(x1, y1, x2, y2, fill=color)  
 piece = self.board[row][col]  
 if piece != ' ':  
 piece\_color = "black" if piece.lower() == 'b' else "white"  
 self.canvas.create\_oval(x1+5, y1+5, x2-5, y2-5, fill=piece\_color)  
 if piece.isupper():  
 self.canvas.create\_rectangle(x1+15, y1+15, x2-15, y2-15, fill="blue")  
 if self.selected\_piece:  
 self.highlight\_moves(self.selected\_piece)  
 elif self.placing\_from\_reserve: *# Добавляем проверку режима размещения из резерва* self.highlight\_available\_cells\_for\_reserve()  
  
 def highlight\_moves(self, pos):  
 x, y = pos  
 piece = self.board[x][y]  
   
 if piece.isupper(): *# Дамка* if self.must\_capture(): *# Если есть возможность боя* directions = [(-1, -1), (-1, 1), (1, -1), (1, 1)]  
 for dx, dy in directions:  
 nx, ny = x + dx, y + dy  
 enemy\_found = False  
   
 while 0 <= nx < 8 and 0 <= ny < 8:  
 if self.board[nx][ny] != ' ':  
 if enemy\_found: *# Если уже нашли шашку противника* break  
 if self.board[nx][ny].lower() != self.current\_turn: *# Если это шашка противника* enemy\_found = True  
 else: *# Если это наша шашка* break  
 elif enemy\_found: *# Если нашли пустую клетку после шашки противника* cx, cy = ny \* 50 + 25, nx \* 50 + 25  
 self.canvas.create\_oval(cx-5, cy-5, cx+5, cy+5, fill="red")  
 nx += dx  
 ny += dy  
 else: *# Обычный ход или взятие своей шашки* directions = [(-1, -1), (-1, 1), (1, -1), (1, 1)]  
 for dx, dy in directions:  
 nx, ny = x + dx, y + dy  
 own\_piece\_found = False  
   
 while 0 <= nx < 8 and 0 <= ny < 8:  
 if self.board[nx][ny] != ' ':  
 if own\_piece\_found: *# Если уже нашли свою шашку* break  
 if self.board[nx][ny].lower() == self.current\_turn: *# Если это наша шашка* own\_piece\_found = True  
 else: *# Если это шашка противника* break  
 else: *# Пустая клетка* cx, cy = ny \* 50 + 25, nx \* 50 + 25  
 if own\_piece\_found: *# Если до этого нашли свою шашку* self.canvas.create\_oval(cx-5, cy-5, cx+5, cy+5, fill="yellow")  
 else: *# Обычный ход* self.canvas.create\_oval(cx-5, cy-5, cx+5, cy+5, fill="red")  
 nx += dx  
 ny += dy  
 else: *# Обычная шашка* if self.must\_capture(): *# Если есть возможность боя* directions = [(-2, -2), (-2, 2), (2, -2), (2, 2)]  
 for dx, dy in directions:  
 nx, ny = x + dx, y + dy  
 if 0 <= nx < 8 and 0 <= ny < 8 and self.is\_valid\_move(pos, (nx, ny)):  
 cx, cy = ny \* 50 + 25, nx \* 50 + 25  
 self.canvas.create\_oval(cx-5, cy-5, cx+5, cy+5, fill="red")  
 else: *# Обычный ход или взятие своей шашки  
 # Проверяем возможность взятия своих шашек* directions = [(-2, -2), (-2, 2), (2, -2), (2, 2)]  
 for dx, dy in directions:  
 nx, ny = x + dx, y + dy  
 if 0 <= nx < 8 and 0 <= ny < 8:  
 mx, my = (x + nx) // 2, (y + ny) // 2  
 if (0 <= mx < 8 and 0 <= my < 8 and   
 self.board[nx][ny] == ' ' and   
 self.board[mx][my].lower() == self.current\_turn):  
 cx, cy = ny \* 50 + 25, nx \* 50 + 25  
 self.canvas.create\_oval(cx-5, cy-5, cx+5, cy+5, fill="yellow")  
   
 *# Обычные ходы* directions = [(-1, -1), (-1, 1)] if self.current\_turn == 'w' else [(1, -1), (1, 1)]  
 for dx, dy in directions:  
 nx, ny = x + dx, y + dy  
 if 0 <= nx < 8 and 0 <= ny < 8 and self.is\_valid\_move(pos, (nx, ny)):  
 cx, cy = ny \* 50 + 25, nx \* 50 + 25  
 self.canvas.create\_oval(cx-5, cy-5, cx+5, cy+5, fill="red")  
  
 def click(self, event):  
 col = event.x // 50  
 row = event.y // 50  
 if 0 <= row < 8 and 0 <= col < 8:  
 if self.placing\_from\_reserve:  
 self.place\_piece\_from\_reserve(row, col)  
 elif self.selected\_piece:  
 self.move\_piece(self.selected\_piece, (row, col))  
 elif self.board[row][col].lower() == self.current\_turn:  
 self.selected\_piece = (row, col)  
 self.undo\_button.config(state=tk.NORMAL)  
 self.draw\_board()  
  
 def place\_piece\_from\_reserve(self, row, col):  
 start\_row = 4 if self.current\_turn == 'w' else 0  
 end\_row = 8 if self.current\_turn == 'w' else 4  
   
 if (start\_row <= row < end\_row and   
 (row + col) % 2 == 1 and   
 self.board[row][col] == ' '):  
   
 *# Размещаем шашку из резерва* self.board[row][col] = self.current\_turn  
 if self.current\_turn == 'w':  
 self.white\_reserve.pop()  
 else:  
 self.black\_reserve.pop()  
   
 *# Завершаем размещение* self.placing\_from\_reserve = False  
 self.switch\_turn()  
 self.draw\_board()  
 self.update\_reserve\_buttons()  
   
 *# Добавляем проверку окончания игры* self.check\_game\_over()  
  
 def undo\_selection(self):  
 self.selected\_piece = None  
 self.placing\_from\_reserve = False  
 self.undo\_button.config(state=tk.DISABLED)  
 self.draw\_board()  
  
 def must\_capture(self):  
 for row in range(8):  
 for col in range(8):  
 if self.board[row][col].lower() == self.current\_turn:  
 if self.can\_capture((row, col)):  
 return True  
 return False  
  
 def can\_capture(self, pos):  
 x, y = pos  
 piece = self.board[x][y]  
   
 if piece.isupper(): *# Дамка* directions = [(-1, -1), (-1, 1), (1, -1), (1, 1)]  
 for dx, dy in directions:  
 nx, ny = x + dx, y + dy  
 enemy\_found = False  
   
 while 0 <= nx < 8 and 0 <= ny < 8:  
 if self.board[nx][ny] != ' ':  
 if enemy\_found: *# Если уже нашли шашку противника* break  
 if self.board[nx][ny].lower() != self.current\_turn: *# Если это шашка противника  
 # Проверяем, есть ли свободное поле после шашки противника* next\_x = nx + dx  
 next\_y = ny + dy  
 if 0 <= next\_x < 8 and 0 <= next\_y < 8 and self.board[next\_x][next\_y] == ' ':  
 return True  
 break  
 else: *# Если это наша шашка* break  
 nx += dx  
 ny += dy  
 else: *# Обычная шашка* directions = [(-2, -2), (-2, 2), (2, -2), (2, 2)]  
 for dx, dy in directions:  
 nx, ny = x + dx, y + dy  
 if 0 <= nx < 8 and 0 <= ny < 8 and self.board[nx][ny] == ' ':  
 mx, my = x + dx // 2, y + dy // 2  
 if 0 <= mx < 8 and 0 <= my < 8:  
 if self.current\_turn == 'w' and self.board[mx][my].lower() == 'b':  
 return True  
 if self.current\_turn == 'b' and self.board[mx][my].lower() == 'w':  
 return True  
 return False  
  
 def switch\_turn(self):  
 self.current\_turn = 'b' if self.current\_turn == 'w' else 'w'  
  
 def has\_valid\_moves(self):  
 *# Проверяем все шашки текущего игрока* for row in range(8):  
 for col in range(8):  
 if self.board[row][col].lower() == self.current\_turn:  
 *# Проверяем возможность взятия* if self.can\_capture((row, col)):  
 return True  
   
 *# Проверяем обычные ходы* piece = self.board[row][col]  
 if piece.isupper(): *# Для дамки* directions = [(-1, -1), (-1, 1), (1, -1), (1, 1)]  
 for dx, dy in directions:  
 nx, ny = row + dx, col + dy  
 while 0 <= nx < 8 and 0 <= ny < 8:  
 if self.board[nx][ny] != ' ':  
 break  
 if self.is\_valid\_move((row, col), (nx, ny)):  
 return True  
 nx += dx  
 ny += dy  
 else: *# Для обычной шашки* directions = [(-1, -1), (-1, 1)] if self.current\_turn == 'w' else [(1, -1), (1, 1)]  
 for dx, dy in directions:  
 nx, ny = row + dx, col + dy  
 if 0 <= nx < 8 and 0 <= ny < 8:  
 if self.is\_valid\_move((row, col), (nx, ny)):  
 return True  
   
 *# Проверяем возможность хода из резерва* if (self.current\_turn == 'w' and self.white\_reserve) or (self.current\_turn == 'b' and self.black\_reserve):  
 start\_row = 4 if self.current\_turn == 'w' else 0  
 end\_row = 8 if self.current\_turn == 'w' else 4  
   
 for row in range(start\_row, end\_row):  
 for col in range(8):  
 if (row + col) % 2 == 1 and self.board[row][col] == ' ':  
 return True  
   
 return False  
  
 def check\_game\_over(self):  
 *# Подсчет шашек на доске* white\_pieces = 0  
 black\_pieces = 0  
   
 for row in range(8):  
 for col in range(8):  
 piece = self.board[row][col]  
 if piece.lower() == 'w':  
 white\_pieces += 1  
 elif piece.lower() == 'b':  
 black\_pieces += 1  
   
 *# Добавляем шашки из резерва* white\_pieces += len(self.white\_reserve)  
 black\_pieces += len(self.black\_reserve)  
   
 *# Проверяем условия победы* if black\_pieces == 0:  
 self.show\_game\_over("Белые шашки выиграли!")  
 return True  
 elif white\_pieces == 0:  
 self.show\_game\_over("Черные шашки выиграли!")  
 return True  
   
 *# Проверяем наличие возможных ходов* if not self.has\_valid\_moves():  
 winner = "Белые" if self.current\_turn == 'b' else "Черные"  
 self.show\_game\_over(f"{winner} шашки выиграли!\nУ противника не осталось возможных ходов!")  
 return True  
   
 return False  
  
 def show\_game\_over(self, message):  
 game\_over\_window = tk.Toplevel(self.root)  
 game\_over\_window.title("Игра окончена")  
 game\_over\_window.geometry("400x200")  
   
 *# Центрируем окно* game\_over\_window.transient(self.root)  
 game\_over\_window.grab\_set()  
   
 *# Добавляем сообщение* label = tk.Label(game\_over\_window,   
 text=message,  
 font=("Arial", 14),  
 wraplength=350,  
 pady=30)  
 label.pack()  
   
 *# Создаем фрейм для кнопок* button\_frame = tk.Frame(game\_over\_window)  
 button\_frame.pack(pady=20)  
   
 *# Кнопка "Новая игра"* new\_game\_button = tk.Button(button\_frame,  
 text="Новая игра",  
 width=15,  
 command=lambda: self.new\_game(game\_over\_window))  
 new\_game\_button.pack(side=tk.LEFT, padx=10)  
   
 *# Кнопка "Закрыть"* close\_button = tk.Button(button\_frame,  
 text="Закрыть",  
 width=15,  
 command=self.quit\_game)  
 close\_button.pack(side=tk.LEFT, padx=10)  
  
 def quit\_game(self):  
 self.root.quit() *# Завершаем главный цикл событий* self.root.destroy() *# Уничтожаем главное окно* def new\_game(self, game\_over\_window=None):  
 *# Сбрасываем состояние игры* self.board = self.create\_board()  
 self.selected\_piece = None  
 self.current\_turn = 'w'  
 self.white\_reserve = []  
 self.black\_reserve = []  
 self.placing\_from\_reserve = False  
 self.undo\_button.config(state=tk.DISABLED)  
   
 *# Обновляем отображение* self.update\_reserve\_buttons()  
 self.draw\_board()  
   
 *# Закрываем окно, если оно существует* if game\_over\_window:  
 game\_over\_window.destroy()  
  
class MenuWindow:  
 def \_\_init\_\_(self):  
 self.root = tk.Tk()  
 self.root.title("Меню")  
 self.root.geometry("300x200")  
   
 *# Создаем фрейм для кнопок* button\_frame = tk.Frame(self.root)  
 button\_frame.pack(expand=True)  
   
 *# Кнопки "Правила" и "Играть"* tk.Button(button\_frame, text="Правила", command=self.show\_rules, width=20).pack(pady=10)  
 tk.Button(button\_frame, text="Играть", command=self.start\_game, width=20).pack(pady=10)  
   
 self.root.mainloop()  
   
 def show\_rules(self):  
 rules\_window = tk.Toplevel(self.root)  
 rules\_window.title("Правила игры")  
 rules\_window.geometry("800x600")  
   
 *# Создаем текстовое поле с полосой прокрутки* text\_frame = tk.Frame(rules\_window)  
 text\_frame.pack(expand=True, fill='both', padx=10, pady=10)  
   
 scrollbar = tk.Scrollbar(text\_frame)  
 scrollbar.pack(side=tk.RIGHT, fill=tk.Y)  
   
 text\_widget = tk.Text(text\_frame, wrap=tk.WORD, yscrollcommand=scrollbar.set)  
 text\_widget.pack(expand=True, fill='both')  
   
 scrollbar.config(command=text\_widget.yview)  
   
 *# Текст правил* rules\_text = """1) Во время партии у игрока есть 3 варианта хода: Тихий ход (то есть,  
движение своей шашки без взятия); Бой шашки соперника; Бой собственной шашки  
(Самовзятие) с дальнейшим введением её в игру.  
  
2) Бить шашки соперника, поставленные под бой (прямой удар),  
обязательно, а вот брать свои шашки игрок не обязан, но если он начал это делать, то  
должен будет побить все шашки, которые находятся под боем, независимо от того,  
чьи они. Если игрок должен бить чужую шашку и после этого у него появляется  
возможно взять свои шашки, он обязан это сделать.  
  
3) Если есть возможность бить в нескольких направлениях, то выбор предоставляется   
берущему. Но если надо выбирать бой либо своей шашки, либо чужой, то сначала надо   
бить шашку соперника. В случае, если при таком раскладе есть возможность бить по 2   
и более направлениям, то выбор взятия свободный. При варианте боя либо чужой шашки,  
либо своей, а потом чужой, приоритет отдаётся первому взятию.  
  
4) Если игроку приходится выбирать между боем 2 чужих шашек, либо чужой и  
своей, то выбирается 1 вариант, но это касается только непосредственного взятия. А если,  
например, у игрока есть возможность взятия, при котором можно побить чужую шашку,  
потом свою, и затем появляется выбор между боем шашки соперника или собственной, то  
выбирает сам бьющий.  
  
5) В случае, если игрок начинает бить свою шашку, а потом у него появляется  
возможность взять либо чужую, либо свою, также выбирает сам бьющий. Иначе говоря,  
обязанности бить шашку соперника после боя собственной, у игрока нет!  
  
6) Шашки из "резерва" можно сбрасывать на доску на любом ходу, то есть, если  
"Самовзятие" произошло на 5 ходу, то ввод резервной шашки возможен на 6, 7, 8 и т.д.  
ходу. Единственное ограничение для игрока: сбрасывать "резервиста" можно только в  
пределах 4 горизонталей своей зоны.  
  
7) По ходу партии может возникнуть ситуация, при которой игрок имеет  
возможность сбить одновременно свою шашку и шашку соперника (или несколько  
каждой из них), что правилами допускается. Свои сбитые шашки при этом уходят в  
"резерв" с дальнейшей возможностью их сброса, а чужие выходят из игры навсегда.  
  
8) Самосбитая Дамка при уходе в резерв становится простой и только в этом  
качестве может в ходе партии возвратиться в игру. Конечно, чисто практически в такой  
"операции", как правило, нет особого смысла, но всё же ситуации в партии могут  
возникнуть разные. Иногда "замена " Дамки на простую с последующей постановкой её на  
доску может быть единственным способом выигрыша или спасения партии."""  
   
 text\_widget.insert(tk.END, rules\_text)  
 text\_widget.config(state=tk.DISABLED) *# Делаем текст только для чтения  
   
 # Кнопка закрытия окна правил* tk.Button(rules\_window, text="Закрыть", command=rules\_window.destroy).pack(pady=10)  
   
 def start\_game(self):  
 self.root.destroy()  
 game\_root = tk.Tk()  
 game = CheckersGame(game\_root)  
 game\_root.mainloop()  
  
class LoginWindow:  
 def \_\_init\_\_(self):  
 self.root = tk.Tk()  
 self.root.title("Авторизация")  
 self.root.geometry("300x200")  
   
 *# Создаем и размещаем элементы интерфейса* tk.Label(self.root, text="Имя:").pack(pady=5)  
 self.username\_entry = tk.Entry(self.root)  
 self.username\_entry.pack(pady=5)  
   
 tk.Label(self.root, text="Пароль:").pack(pady=5)  
 self.password\_entry = tk.Entry(self.root, show="\*")  
 self.password\_entry.pack(pady=5)  
   
 *# Создаем фрейм для кнопок* button\_frame = tk.Frame(self.root)  
 button\_frame.pack(pady=20)  
   
 *# Кнопки входа и регистрации* tk.Button(button\_frame, text="Вход", command=self.login).pack(side=tk.LEFT, padx=10)  
 tk.Button(button\_frame, text="Регистрация", command=self.register).pack(side=tk.LEFT, padx=10)  
   
 *# Метка для сообщений об ошибках* self.message\_label = tk.Label(self.root, text="", fg="red")  
 self.message\_label.pack(pady=10)  
   
 self.root.mainloop()  
   
 def register(self):  
 username = self.username\_entry.get().strip()  
 password = self.password\_entry.get().strip()  
   
 if not username or not password:  
 self.message\_label.config(text="Заполните все поля!")  
 return  
   
 *# Проверяем существование пользователя* try:  
 with open("D:\\python\\osnova\\accounts.txt", "r", encoding='cp1251') as file:  
 for line in file:  
 stored\_username = line.strip().split(" : ")[0]  
 if stored\_username == username:  
 *# Создаем всплывающее окно с предупреждением* warning\_window = tk.Toplevel(self.root)  
 warning\_window.title("Внимание")  
 warning\_window.geometry("300x100")  
   
 label = tk.Label(warning\_window,   
 text="Имя пользователя уже существует, введите другое",  
 wraplength=250,  
 pady=20)  
 label.pack()  
   
 button = tk.Button(warning\_window,   
 text="OK",   
 command=warning\_window.destroy)  
 button.pack()  
 return  
 except FileNotFoundError:  
 pass  
   
 *# Если пользователь не существует, регистрируем его* try:  
 with open("D:\\python\\osnova\\accounts.txt", "a", encoding='cp1251') as file:  
 file.write(f"{username} : {password}\n")  
   
 *# Создаем окно с сообщением об успешной регистрации* success\_window = tk.Toplevel(self.root)  
 success\_window.title("Успешная регистрация")  
 success\_window.geometry("300x100")  
   
 label = tk.Label(success\_window,   
 text="Данные сохранены, нажмите кнопку 'Вход'",  
 wraplength=250,  
 pady=20)  
 label.pack()  
   
 button = tk.Button(success\_window,   
 text="OK",   
 command=success\_window.destroy)  
 button.pack()  
   
 *# Очищаем сообщение об ошибках* self.message\_label.config(text="")  
   
 except Exception as e:  
 self.message\_label.config(text=f"Ошибка при регистрации: {str(e)}")  
   
 def login(self):  
 username = self.username\_entry.get().strip()  
 password = self.password\_entry.get().strip()  
   
 if not username or not password:  
 self.message\_label.config(text="Заполните все поля!")  
 return  
   
 try:  
 *# Пробуем сначала cp1251* try:  
 with open("D:\\python\\osnova\\accounts.txt", "r", encoding='cp1251') as file:  
 for line in file:  
 stored\_username, stored\_password = line.strip().split(" : ")  
 if stored\_username == username and stored\_password == password:  
 self.root.quit() *# Останавливаем главный цикл* self.root.destroy() *# Закрываем окно входа* MenuWindow() *# Открываем меню* return  
 except UnicodeDecodeError:  
 *# Если не получилось, пробуем utf-8* with open("D:\\python\\osnova\\accounts.txt", "r", encoding='utf-8') as file:  
 for line in file:  
 stored\_username, stored\_password = line.strip().split(" : ")  
 if stored\_username == username and stored\_password == password:  
 self.root.quit() *# Останавливаем главный цикл* self.root.destroy() *# Закрываем окно входа* MenuWindow() *# Открываем меню* return  
   
 *# Если дошли до этой точки, значит логин/пароль неверные* error\_window = tk.Toplevel(self.root)  
 error\_window.title("Ошибка")  
 error\_window.geometry("300x100")  
   
 label = tk.Label(error\_window,   
 text="Имя или пароль введены не правильно",  
 wraplength=250,  
 pady=20)  
 label.pack()  
   
 button = tk.Button(error\_window,   
 text="OK",   
 command=error\_window.destroy)  
 button.pack()  
   
 except FileNotFoundError:  
 error\_window = tk.Toplevel(self.root)  
 error\_window.title("Ошибка")  
 error\_window.geometry("300x100")  
   
 label = tk.Label(error\_window,   
 text="Имя или пароль введены не правильно",  
 wraplength=250,  
 pady=20)  
 label.pack()  
   
 button = tk.Button(error\_window,   
 text="OK",   
 command=error\_window.destroy)  
 button.pack()  
   
 def start\_game(self):  
 MenuWindow() *# Открываем меню напрямую*if \_\_name\_\_ == "\_\_main\_\_":  
 LoginWindow()

# 

# ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы была разработана программа "Русские циклические 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:
   1. Реализацию шашечной доски и фигур
   2. Систему ходов с учетом всех правил русских циклических шашек
   3. Механику игры с возможностью взятия своих и чужих шашек
   4. Управление резервом шашек для стратегического планирования
2. Разработана система авторизации и хранения данных:
   1. Регистрация и авторизация пользователей
   2. Хранение учетных записей в файле
   3. Защита пользовательских данных
3. Реализован удобный пользовательский интерфейс:
   1. Интуитивно понятное управление
   2. Визуальная обратная связь
   3. Подсветка возможных ходов и взятий
   4. Система сообщений и уведомлений для пользователей

Программная документация разработана в соответствии с ГОСТ 19.201-78 "Техническое задание. Требования к содержанию и оформлению". Код программы структурирован и прокомментирован согласно стандартам.

В процессе разработки были получены практические навыки:

* Объектно-ориентированного программирования
* Работы с графическими библиотеками
* Проектирования пользовательских интерфейсов
* Работы с файловой системой

Созданная программа полностью соответствует поставленным требованиям и может быть использована как для обучения русским циклическим шашкам, так и для развлекательных целей.

# ИСТОЧНИКИ ИСПОЛЬЗОВАННЫЕ

1. Пирогов, А.А. Python для начинающих. Изучаем программирование с нуля. – М.: Издательство "БХВ-Петербург", 2020.
2. ГОСТ 19.701-90 «Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения». – М.: Издательство стандартов, 1990.
3. В.В. Шишкин, Д.С. Афонин РАЗРАБОТКА ЛОГИЧЕСКИХ КОМПЬЮТЕРНЫХ ИГР С ГРАФИЧЕСКИМ ИНТЕРФЕЙСОМ В СРЕДЕ ПИТОН
4. Pygame Documentation. [Электронный ресурс]. - Режим доступа: https://www.pygame.org/docs/

# ПРИЛОЖЕНИЕ 1: MINDMAP

