**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

федеральное государственное бюджетное образовательное учреждение

высшего образования

**«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Кафедра «Измерительно-вычислительные комплексы»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

на курсовую работу

по дисциплине «Алгоритмы и структуры данных»

Тема «Компьютерная логическая игра «Канадские шашки»

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

Р.02069337. 21/294-20 ПЗ-01

Листов 62

Руководитель разработки:

доцент каф. ИВК, к.т.н., доцент

*Шишкин Вадим Викторинович*

« » 2024 г.

Исполнитель:

студент гр. ИСТбд-32

*Карпович Кирилл Александрович*

« » 2024 г.

2024

Содержание

Аннотация………………………………………………………………………...3

Техническое задание……………………………………………………………..4

Пояснительная записка…………………………………………………………..9

Руководство программиста……………………………………………………..25

Текст программы………………………………………………………………...34

Аннотация

Данная курсовая работа посвящена реализации игры в канадские шашки с использованием языка программирования Python и библиотеки Tkinter для создания графического интерфейса. Канадские шашки представляют собой захватывающую стратегическую настольную игру, которая требует от игроков не только хороших навыков ведения борьбы, но и умение планировать свои ходы.

В работе будут рассмотрены основные правила игры в канадские шашки, включая правила передвижения шашек, правила захвата и правила окончания игры. Затем будет представлена архитектура программы, включая структуру данных для представления игрового поля, алгоритмы для определения возможных ходов и стратегии выбора оптимального хода.

Использование библиотеки Tkinter позволит создать графический интерфейс для игры, обеспечивая удобное взаимодействие пользователя с программой.

Целью данной работы является разработка полноценной игры в канадские шашки на языке Python с использованием библиотеки Tkinter, а также изучение алгоритмов и структур данных, лежащих в основе данной игры, что способствует развитию навыков программирования и анализа алгоритмов.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Измерительно-вычислительные комплексы»

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

на курсовую работу

по дисциплине «Алгоритмы и структуры данных»

Тема «Компьютерная логическая игра «Канадские шашки»

Р.02069337. 21/294-20 ТЗ-01

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

Листов 5

Исполнитель:

студент гр. ИСТбд-32

*Карпович Кирилл Александрович*

« » 2022 г.

2024

**Введение**

Игра “Канадские шашки”

Вариант игры в “Шашки” на доске 12х12 с 30 шашками. Шашки расставляются на чёрных полях первых пяти горизонтальных рядов с каждой стороны. Играющий белыми ходит первый, далее ходы делаются поочередно. Шашки делятся на простые и дамки. В начальном положении все шашки простые.

Правила:

– Простая шашка ходит по диагонали вперёд на одну клетку. При достижении любого поля последней горизонтали, простая шашка превращается в дамку.

– Дамка ходит по диагонали на любое свободное поле как вперёд, так и назад.

– Взятие обязательно, если оно возможно. Шашки снимаются с доски лишь после того, как берущая шашка остановилась.

– При взятии применяется правило турецкого удара — если при бое нескольких шашек противника шашка или дамка повторно выходит на уже побитую шашку, то ход останавливается (то есть, запрещается дважды брать одну и ту же шашку, при этом можно пересекать дважды одно и то же пустое поле).

– Если есть несколько вариантов взятия, нужно выполнить тот из них, который снимает максимальное количество шашек соперника (независимо от их качества — и простая, и дамка считается одной шашкой). Если всё ещё вариантов несколько, можно выбрать любой из них. Если имеется выбор боя дамкой или простой, то можно брать любой из них, соблюдая и в этом случае правило взятия наибольшего количества шашек.

– Если простая шашка в процессе взятия достигает дамочного поля и может бить дальше в роли простой шашки, то она этим ходом продолжает бой и остаётся простой. В противном случае она превращается в дамку и останавливается. Право боя по правилам дамки она приобретает лишь со следующего хода.

Функциональные возможности приложения: авторизация, регистрация, игра в канадские шашки.

**1. Основания для разработки**

Основанием для разработки является учебный план направления 09.03.02 «Информационные системы и технологии» и распоряжение по факультету.

**2. Требования к программе или программному изделию**

**2.1. Функциональное назначение**

Функциональное назначение – игра канадские шашки.

Перечень автоматизируемых процессов:

1. Регистрация и авторизация пользователей.
2. Отрисовка игрового поля.
3. Обработка событий мыши.
4. Обработка правильности ходов.
5. Подсчет позиции с помощью оценочной функции.
6. Реализация ходов ботом.
7. Проверка на конец игры.

**2.2 Требования к функциональным характеристикам**

2.2.1 Требования к структуре приложения

Приложение должно иметь:

1. Модуль для аутентификации пользователя.
2. Модуль для логики игры канадских шашек.

2.2.2 Требования к составу функций приложения

Приложение должно выполнять следующие функции:

1. Регистрация / авторизация пользователя.
2. Шифрование логина и пароля.
3. Отображение игрового процесса.
4. Проверка правильности хода.
5. Выбор хода компьютера на основе МИНИМАКС.
6. Выявление победителя.

2.2.3 Требования к организации информационного обеспечения, входных и выходных данных

Ввод данных осуществляется в форме регистрации/авторизации. Входные данные должны удовлетворять следующим условиям:

1. На вход подаётся строковый тип данных.
2. Данные могут состоять из любого набора символов.
3. Ограничение на размер данных: Минимум – 3 символа, максимум – 20.
4. Проверка на уже существующий логин.

Также входные данные представлены в виде программного кода, при выполнении следующих действий:

1. Нажатия клавиш мыши.
2. Движения мышью.

**2.3 Требования к надёжности**

Требования к надежности не предъявляются

**2.4 Требования к информационной и программной совместимости**

1. ОС: Windows 10 21H2.
2. Среда разработки: PyCharm Community Edition Version: 2022.3.
3. Версия языка: Python 3.12
4. Библиотеки: Tkinter

**2.5 Требования к маркировке и упаковке**

Определяются заданием на курсовую работу.

**2.6 Требования к транспортированию и хранению**

2.6.1 Условия транспортирования

Требования к условиям транспортирования не предъявляются.

2.6 2 Условия хранения

Обеспечение свободного доступа к проекту в репозитории до окончания срока учебы

2.6 3 Сроки хранения

Срок хранения – до окончания срока учебы

**3. Требования к программной документации**

Определяются заданием на курсовую работу.

**4. Стадии и этапы разработки**

Определяются заданием на курсовую работу.

**5. Порядок контроля и приёмки**

Определяются заданием на курсовую работу.

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

федеральное государственное бюджетное образовательное учреждение

высшего образования

**«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Кафедра «Измерительно-вычислительные комплексы»

**Курсовая работа**

**По дисциплине «Алгоритмы и структуры данных»**

**Тема «Компьютерная логическая игра «Канадские шашки»**

**Пояснительная записка**

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

Р.02069337. 21/294-20 ПЗ-01

Листов 17

**Руководитель разработки**:

доцент каф. ИВК, к.т.н., доцент

*Шишкин Вадим Викторинович*

« » 2024 г.

**Исполнитель**:

студент гр. ИСТбд-32

*Карпович Кирилл Александрович*

« » 2024 г.

**2024**

**Введение**

Приложение “Канадские шашки”

Игра “Канадские шашки” на доске 12х12 с 30 шашками. Шашки расставляются на чёрных полях первых пяти горизонтальных рядов с каждой стороны. Играющий белыми ходит первый, далее ходы делаются поочередно. Шашки делятся на простые и дамки. В начальном положении все шашки простые.

В приложении реализована регистрация и авторизация пользователя, его личный кабинет, графический интерфейс.

**1. Проектная часть**

**1.1 Постановка задачи на разработку приложения**

Определяется заданием на курсовую работу. Детализируется в разработанном техническом задании (приложение 1).

**1.2 Математические методы**

Математические методы не применялись.

**1.3 Архитектура и алгоритмы**

1.3.1. Архитектура

Архитектура приложения "Канадские шашки" включает несколько ключевых компонентов:

1. Интерфейс пользователя (GUI):

- Используется библиотека Tkinter для создания графического интерфейса.

- Главное окно содержит различные виджеты, такие как кнопки, метки и поле для отображения игры.

- Пользователь может взаимодействовать с игрой, щелкая мышью на доске.

2. Логика игры:

- Отвечает за правила игры, ходы игроков и логику выигрыша.

- Включает класс `Game`, который управляет ходом игры, включая отрисовку доски, обработку действий игрока и компьютера, анимацию ходов и т. д.

- Ходы игрока обрабатываются в методе `mouse\_down`, который определяет выбранную клетку и обрабатывает действия игрока (ход или выбор шашки).

- Логика компьютера реализована в методе `\_\_handle\_enemy\_turn`, который использует алгоритм минимакс для выбора лучшего хода.

3. Авторизация и регистрация:

- Реализованы функции `authenticate`, `register` и `user\_exist` для авторизации и регистрации пользователей.

- Включает экраны авторизации/регистрации и личного кабинета.

4. Шифрование данных:

- Используется шифрование данных при авторизации и регистрации пользователей.

- Реализованы функции шифрования и дешифрования для обеспечения безопасности пользовательских данных.

5. Основные структуры данных:

- Классы `Move` и `Point` используются для представления ходов и координат на доске.

- Константы и перечисления для хранения значений, таких как размер доски, размер клетки, цвета и стороны шашек.

6. Главная функция:

- Точка входа в приложение, запускающая главное окно и начинающая игру.

Взаимодействие между компонентами происходит следующим образом:

- Пользователь взаимодействует с интерфейсом пользователя, делая ходы или регистрируясь/авторизуясь.

- Интерфейс пользователя вызывает соответствующие функции для обработки действий пользователя, такие как ходы игрока или процесс авторизации/регистрации.

- Логика игры управляет ходом игры, отрисовкой доски и обработкой действий игрока и компьютера.

- Авторизация и регистрация обрабатывают запросы пользователя на аутентификацию или регистрацию, обеспечивая безопасность данных пользователя.

- Шифрование данных используется для защиты пользовательских данных при авторизации и регистрации.

1.3.2. Алгоритм МИНИМАКС

Алгоритм реализует поиск оптимального хода(Рис 1).

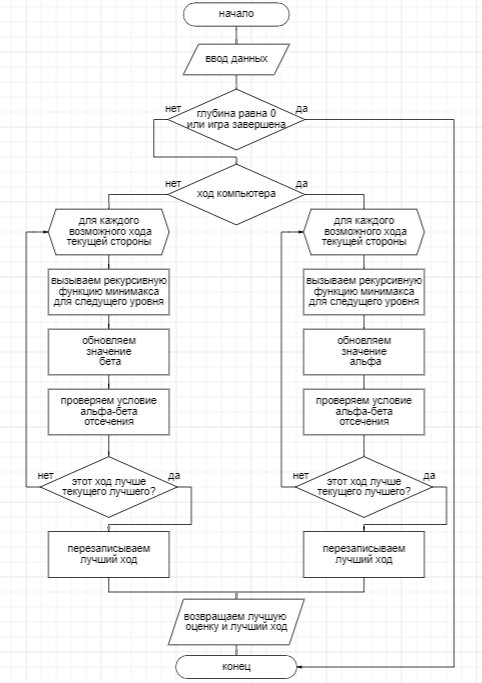


Рис. 1 Блок-схема алгоритма минимакс.

1.3.2 Алгоритм шифрования Виженера.

Алгоритм реализует шифрование пароля и логина при регистрации(Рис. 2).

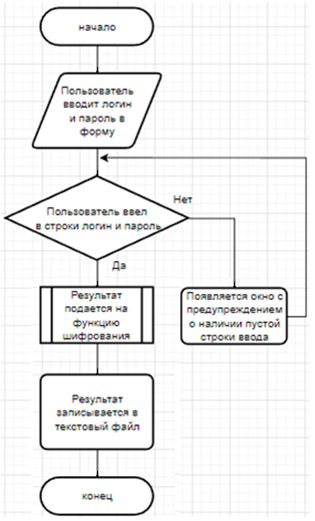


Рис. 2 Блок-схема Алгоритм шифрования Виженера.

1.3.3 Алгоритм дешифрования Виженера

Алгоритм реализует дешифрование пароля и логина при авторизации(Рис. 3).

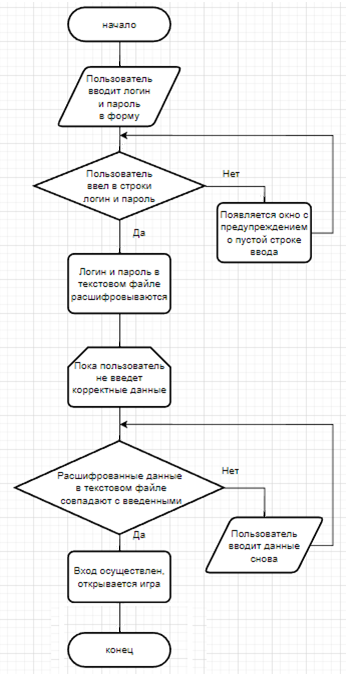


Рис. 3 Блок-схема Алгоритм дешифрования Виженера.

1.3.4 Алгоритм составление списка обязательных ходов

Алгоритм составляет список обязательных ходов(Рис.4)

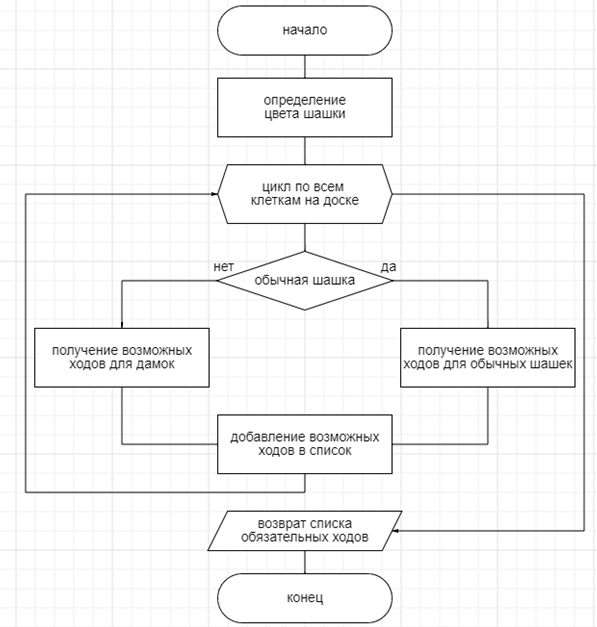


Рис. 4 Блок-схема алгоритма обязательных ходов.

1.3.5 Алгоритм составление списка необязательных ходов.

Алгоритм составляет список необязательных ходов(Рис. 5)

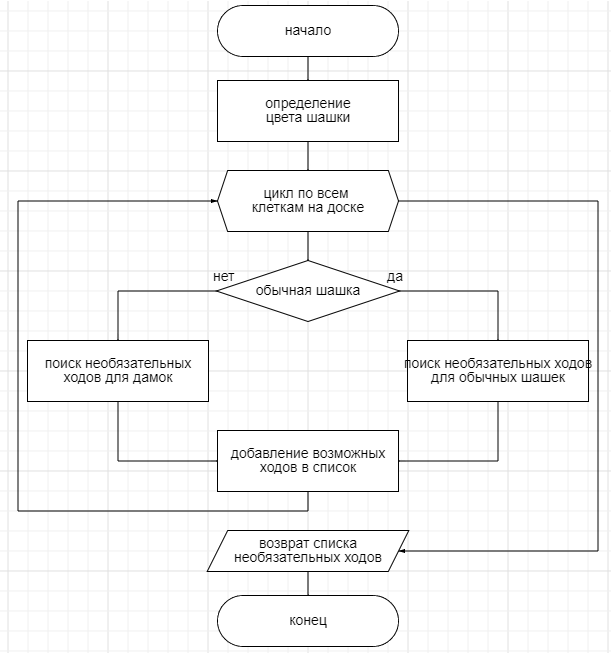


Рис. 5 Блок-схема алгоритма необязательных ходов.

1.3.6 Алгоритм оценочной функции

Данный алгоритм подсчитывает позицию с помощью веса фигур, дамки являются преимуществом для соперника, так как подставить свои пешки под дамки гораздо легче(Рис. 6).

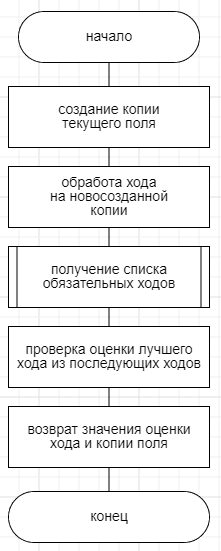


Рис. 6 Блок-схема алгоритма оценочной функции.

1.3.7 Алгоритм проверка на конец игры

Данный алгоритм проверяет, наличие ходов, если ходить некуда, то объявляется победа противоположного игрока(Рис. 7).

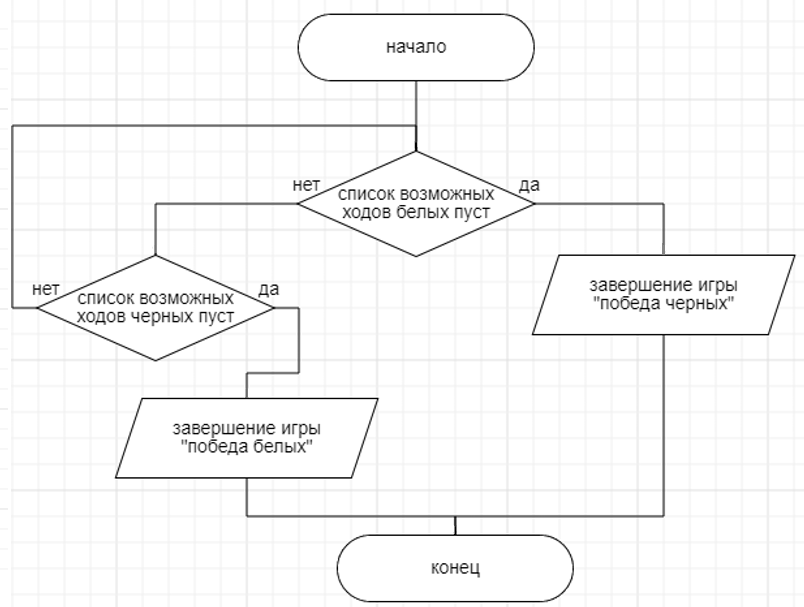


Рис. 7 Блок-схема алгоритма проверки на конец игры.

1.3.8 Алгоритм регистрации.

Данный алгоритм регистрирует новых пользователей(Рис. 8)

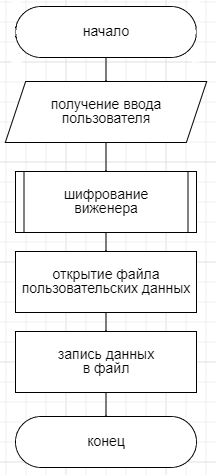


Рис. 8 Блок-схема алгоритма регистрации.

1.3.9 Алгоритм авторизации.

Данный алгоритм авторизирует существующих пользователей(Рис. 9)

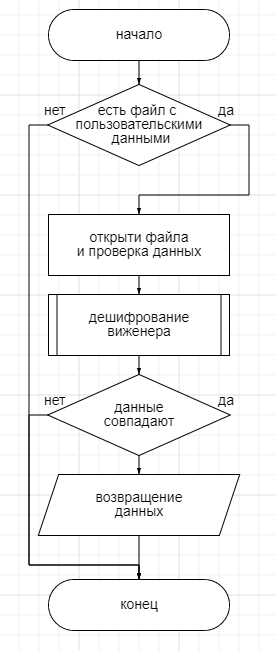


Рис. 9 Блок-схема алгоритма авторизации.

**1.4 Тестирование**

1.4.1 Описание отчета

В данном отчете представлены результаты тестирования программы

на основе разработанных чек-листов и статического 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

Тестирование проводилось с использованием следующих методов:

– Статическое 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.

1.4.4 Проведенные тесты

В ходе тестирования были проведены следующие тесты:

1.4.4.1 Статическое тестирование

Количество обнаруженных и исправленных ошибок в документации: 3

Количество обнаруженных и исправленных ошибок в программном коде: 3

1.4.4.2 Ручное тестирование.

Написаны и проведены следующие тест-кейсы и чек-листы:

ТК1. Отработка авторизации.

Предварительные шаги:

Зарегистрироваться с логином: TestTest и паролем 12345678.

Шаги:

1. Запустить приложение.

2. В окне регистрации, в поле «логин» ввести TestTest, а в поле «пароль» - 12345678.

3. Нажать кнопку «Войти».

Ожидаемый результат:

Пользователь получит уведомление «Авторизация прошла успешно»

Фактический результат:

Пользователь получит уведомление «Авторизация прошла успешно»

ТК2. Прохождение регистрации односимвольными логином и

паролем.

Шаги:

1. Запустить приложение.

2. В окне регистрации, в поле «логин» ввести 1, а в поле «пароль» - 1.

3. Нажать на кнопку «Регистрация».

Ожидаемый результат:

Пользователь получит сообщение об ошибке: «Имя пользователя и пароль должны иметь длину от 3 до 20 символов».

Фактический результат:

Пользователь зарегистрировался.

ТК3. Прохождение регистрации уже существующего пользователя.

Шаги:

1.Запустить приложение.

2.В окне регистрации, в поле «логин» ввести уже существующего пользователя TestTest, а в поле «пароль» - любое значение.

3. Нажать кнопку «Войти».

Ожидаемый результат:

Пользователь получит сообщение об ошибке: «Пользователь уже существует».

Фактический результат:

Пользователь зарегистрировался.

ТК4. Прохождение авторизации с неправильным логином.

Шаги:

1.Запустить приложение.

2.В окне регистрации, в поле «логин» ввести TestTest2, а в поле «пароль» - 123456789.

3. Нажать кнопку «Войти».

Ожидаемый результат:

Пользователь получит сообщение об ошибке: «Неверное имя пользователя или пароль».

Фактический результат:

Пользователь получил сообщение об ошибке: «Неверное имя пользователя или пароль».

ТК5. Прохождение авторизации с неправильным и паролем.

Шаги:

1.Запустить приложение.

2.В окне регистрации, в поле «логин» ввести TestTest2, а в поле «пароль» - 123456789.

3. Нажать кнопку «Войти».

Ожидаемый результат:

Пользователь получит сообщение об ошибке: «Неверное имя пользователя или пароль».

Фактический результат:

Пользователь получил сообщение об ошибке: «Неверное имя пользователя или пароль».

1.4.5 Выводы

На основе проведенных тестов сделаны следующие выводы:

– Программа успешно прошла все тесты и работает корректно.

– Обнаружены и исправлены следующие дефекты: лингвистическая ошибка в тесте, ошибка оформления технического задания, регистрация уже существующего пользователя.

– Рекомендации по дальнейшему развитию программы: добавление

ограничения времени на ход пользователя, звукового сопровождения,

таблицы лидеров.

**2. Источники, использованные при разработке**

1. Лутц, М. Изучаем Python [Текст]: учеб. - метод, пособие — 6-e изд. - 2019. (дата обращения: 14.03.2024).
2. Python // Doc URL: python.org/doc/ (дата обращения: 12.03.2024).
3. Wikipedia [Электронный ресурс]: WAKE – URL: https://ru.wikipedia.org/wiki/Шифр\_Виженера (дата обращения: 12.03.2024)
4. Chekers // wikipedia URL: en.wikipedia.org/wiki/Checkers (дата обращения: 13.03.2024).
5. Habr [Электронный ресурс]: Минимакс на примере игры в зайца и волков– URL: https://habr.com/ru/post/146088/ (дата обращения: 15.03.2024)
6. YouTube [Электронный ресурс]: Algorithms Explained – minimax and alpha-beta pruning – URL: https://www.youtube.com/watch?v=l-hh51ncgDI (дата обращения: 15.03.2024)

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

федеральное государственное бюджетное образовательное учреждение

высшего образования

**«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Кафедра «Измерительно-вычислительные комплексы»

**Курсовая работа**

**По дисциплине «Алгоритмы и структуры данных»**

**Тема «Компьютерная логическая игра «Киммерийские шашки»**

**Руководство программиста**

Р.02069337. 21/294-20 РП-01

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

Листов 9

Исполнитель:

студент гр. ИСТбд-32

*Карпович Кирилл Александрович*

« » 2024 г.

**2024**

**1. Назначение и условия применения программы**

**1.1 Назначение и функции, выполняемые приложением**

Назначение приложения – развлекательная игра в “канадские шашки” против “компьютера”

Правила:

* Доска 12х12, 5 рядов шашек.

– Простая шашка ходит по диагонали вперёд на одну клетку. При достижении любого поля последней горизонтали, простая шашка превращается в дамку.

– Дамка ходит по диагонали на любое свободное поле как вперёд, так и назад.

– Взятие обязательно, если оно возможно. Шашки снимаются с доски лишь после того, как берущая шашка остановилась.

– При взятии применяется правило турецкого удара — если при бое нескольких шашек противника шашка или дамка повторно выходит на уже побитую шашку, то ход останавливается (то есть, запрещается дважды брать одну и ту же шашку, при этом можно пересекать дважды одно и то же пустое поле).

– Если есть несколько вариантов взятия, нужно выполнить тот из них, который снимает максимальное количество шашек соперника (независимо от их качества — и простая, и дамка считается одной шашкой). Если всё ещё вариантов несколько, можно выбрать любой из них. Если имеется выбор боя дамкой или простой, то можно брать любой из них, соблюдая и в этом случае правило взятия наибольшего количества шашек.

– Если простая шашка в процессе взятия достигает дамочного поля и может бить дальше в роли простой шашки, то она этим ходом продолжает бой и остаётся простой. В противном случае она превращается в дамку и останавливается. Право боя по правилам дамки она приобретает лишь со следующего хода.

Функциональные возможности приложения: авторизация, регистрация, личный кабинет, игра в канадские шашки.

**1.2 Условия, необходимые для использования приложения**

1. ОС: Windows 10 21H2.
2. Среда разработки: PyCharm Community Edition Version: 2022.3.
3. Версия языка: Python 3.12.
4. Библиотеки: Tkinter.

В качестве БД использовать файл.

**2. Характеристики программы**

**2.1 Характеристики приложения**

Количество значимых строк кода – 660.

Количество алгоритмов – 8.

Количество методов – 38.

Порядок работы:

Запуск программы производится двойным щелчком мыши на файл «main.exe».

При запуске приложения появляется окно авторизации пользователя.(Рис. 1).

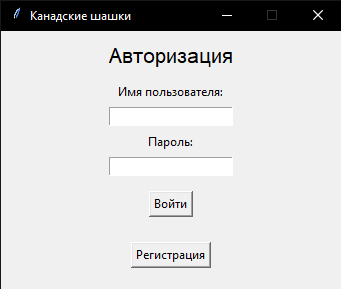


Рис. 1

При нажатии кнопки “Перейти к регистрации” появится окно регистрации пользователя(Рис. 2).

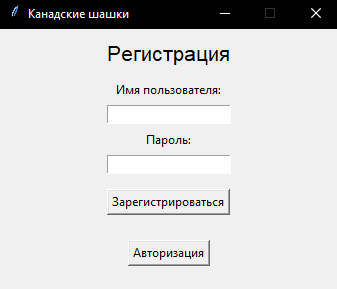


Рис. 2

После успешной регистрации и авторизации появляется окно личного кабинета(Рис. 3) с кнопками начала игры и выхода из профиля обратно на окно авторизации.

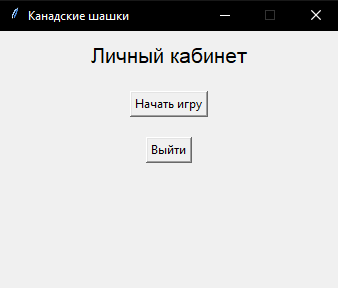


Рис. 3

При нажатии на кнопку «Выйти» в личном кабинете, пользователя возвращает

в меню выбора регистрации и авторизации. (Рис. 1)

В случае нажатия кнопки “Начать игру” запускается игра(Рис 4).

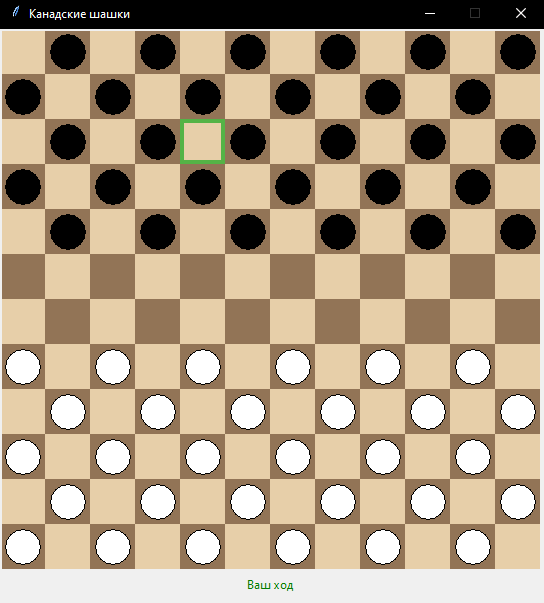


Рис. 4

Далее пользователю следует кнопкой мыши шашку которой он хочет пойти и сделать ход(Рис 5).

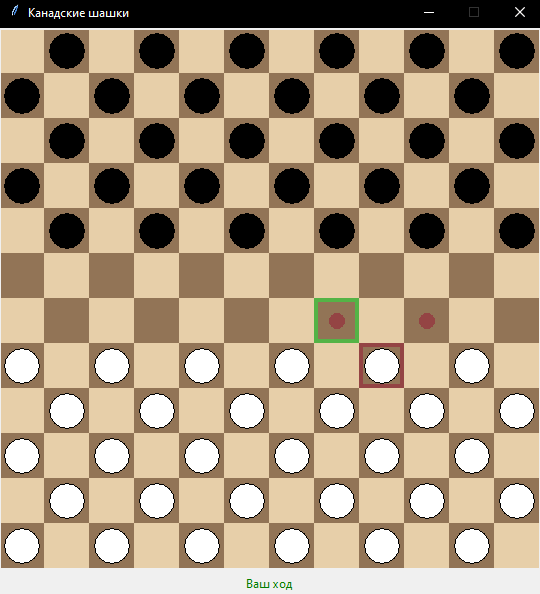


Рис. 5

После хода белыми, право хода переходит чёрным(компьютеру)(Рис 6).

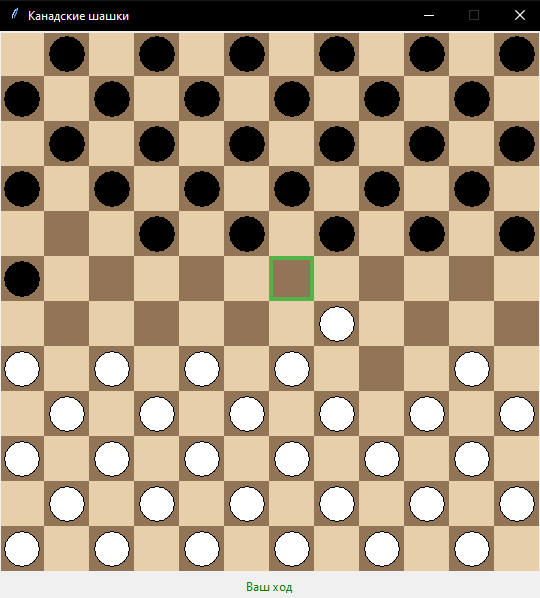


Рис. 6

После победы одной из сторон появляется уведомление(Рис. 7)

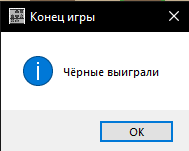


Рис. 7

При нажатии кнопки “ОК” игра начинается заново.

**2.2 Особенности реализации приложения**

В программе используются двумерный массив, для представления поля и массив для хранения всех возможных ходов.

Приложение используются библиотеки:

* Tkinter - это графическая библиотека, позволяющая создавать программы с оконным интерфейсом

**3. Обращение к программе**

Методы:

1. change\_type – метод изменения типа шашки.
2. opposite – метод берет противоположный цвет игрока.
3. copy – создает копию поля из образца.
4. generate – генерация поля с шашками.
5. type\_at – получение типа шашки на поле по координатам.
6. at – получение шашки на поле по координатам.
7. is\_within – Определяет лежит ли точка в пределах поля.
8. init\_images – Инициализация изображений.
9. animate\_move – Анимация перемещения шашки.
10. draw – Отрисовка сетки поля и шашек.
11. draw\_turn – Отрисовка чей ход.
12. draw\_field\_grid – Отрисовка сетки поля.
13. draw\_checkers – Отрисовка шашек.
14. mouse\_move – Событие перемещения мышки.
15. mouse\_down – Событие нажатия мышки.
16. handle\_move – Совершение хода.
17. handle\_player\_turn – Обработка хода игрока.
18. handle\_enemy\_turn – Обработка хода противника(компьютера).
19. check\_for\_game\_over - Проверка на конец игры.
20. minimax\_move – Минимакс.
21. get\_moves\_list – Получение списка ходов.
22. evaluate\_move – Оценка хода, насколько много вражеских шашек можно съесть, совершив указанный ход.
23. get\_required\_moves\_list – Получение списка обязательных ходов.
24. get\_optional\_moves\_list – Получение списка необязательных ходов.
25. register – Регистрация нового пользователя.
26. authenticate – Авторизация пользователя.
27. user\_exist – Проверка существования пользователя.
28. authenticate\_user – Функция авторизации.
29. register\_user – Функция регистрации.
30. vigenere – Алгоритм Виженера.
31. encrypt – Функиця Шифрования.
32. decrypt – Функция Дешифрования.
33. set\_exit\_callback – Выход из личного кабинета.
34. start\_game – функция начала игры.
35. show\_lk – функция отображения личного кабинета.
36. show\_auth
37. show\_game – функция отображения игры.
38. Main –Точка входа.

Используемые библиотеки:

* Tkinter

**4. Сообщения**

При победе программа отображает победителя, в виде сообщений: «Чёрные выиграли» или «Белые Выиграли»

При вводе неправильного логина или пароля в окне авторизации выведется сообщение «Неверное имя пользователя или пароль»

При вводе логина или пароля длиной менее 3 и более 20 символов в окне регистрации выведется сообщение «Имя пользователя должно иметь длину от 3 до 20 символов»

При вводе логина уже зарегистрирован, в окне регистрации выведется сообщение «Пользователь с таким именем уже существует»

При пустых полях логина и пароля после нажатия кнопки “авторизация” в окне авторизации выведется сообщение «Неверное имя пользователя или пароль»

При успешной регистрации в окне регистрации выведется сообщение «Регистрация прошла успешно»

При успешной авторизации в окне авторизации выведется сообщение «Авторизация прошла успешно»

Текст программы

**Checkers:**

from checkers.enums import CheckerType

#Класс шашки

class Checker:

def \_\_init\_\_(self, type: CheckerType = CheckerType.NONE):

self.\_\_type = type

@property

def type(self):

return self.\_\_type

#Изменение типа шашки

def change\_type(self, type: CheckerType):

self.\_\_type = type

**Constanst:**

from checkers.point import Point

from checkers.enums import CheckerType, SideType

# Сторона за которую играет игрок

PLAYER\_SIDE = SideType.WHITE

# Размер поля

X\_SIZE = Y\_SIZE = 12

# Размер ячейки

CELL\_SIZE = 45

# Скорость анимации

ANIMATION\_SPEED = 20

# Ширина рамки

BORDER\_WIDTH = 2 \* 2

# Цвета игровой доски

FIELD\_COLORS = ['#E7CFA9', '#927456']

# Цвет рамки при наведении на ячейку мышкой

HOVER\_BORDER\_COLOR = '#54b346'

# Цвет рамки при выделении ячейки

SELECT\_BORDER\_COLOR = '#944444'

# Цвет кружков возможных ходов

POSIBLE\_MOVE\_CIRCLE\_COLOR = '#944444'

# Возможные смещения ходов шашек

MOVE\_OFFSETS = [

Point(-1, -1),

Point( 1, -1),

Point(-1, 1),

Point( 1, 1)

]

# Массивы типов белых и чёрных шашек [Обычная пешка, дамка]

WHITE\_CHECKERS = [CheckerType.WHITE\_REGULAR, CheckerType.WHITE\_QUEEN]

BLACK\_CHECKERS = [CheckerType.BLACK\_REGULAR, CheckerType.BLACK\_QUEEN]

GAME\_MODE = 'standard'

**Enums:**

from enum import Enum, auto

#Перечисление цветов игроков

class SideType(Enum):

WHITE = auto()

BLACK = auto()

def opposite(self):

if self == SideType.WHITE:

return SideType.BLACK

elif self == SideType.BLACK:

return SideType.WHITE

else:

raise ValueError()

#Тип шашки

class CheckerType(Enum):

NONE = auto()

WHITE\_REGULAR = auto()

BLACK\_REGULAR = auto()

WHITE\_QUEEN = auto()

BLACK\_QUEEN = auto()

**Field:**

from checkers.enums import CheckerType

from checkers.checker import Checker

from checkers.constants import WHITE\_CHECKERS, BLACK\_CHECKERS, GAME\_MODE

from functools import reduce

#Класс поля

class Field:

def \_\_init\_\_(self, x\_size: int, y\_size: int):

self.\_\_x\_size = x\_size

self.\_\_y\_size = y\_size

if GAME\_MODE == 'standard':

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)

@classmethod

def copy(cls, field\_instance):

#Создаёт копию поля из образца

field\_copy = cls(field\_instance.x\_size, field\_instance.y\_size)

for y in range(field\_instance.y\_size):

for x in range(field\_instance.x\_size):

field\_copy.at(x, y).change\_type(field\_instance.type\_at(x, y))

return field\_copy

# Генерация поля с шашками

def \_\_generate(self):

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)

# Получение типа шашки на поле по координатам

def type\_at(self, x: int, y: int) -> CheckerType:

return self.\_\_checkers[y][x].type

# Получение шашки на поле по координатам

def at(self, x: int, y: int) -> Checker:

return self.\_\_checkers[y][x]

# Определяет лежит ли точка в пределах поля

def is\_within(self, x: int, y: int) -> bool:

return (0 <= x < self.x\_size and 0 <= y < self.y\_size)

# Количество белых шашек на поле

@property

def white\_checkers\_count(self) -> int:

return sum(reduce(lambda acc, checker: acc + (checker.type in WHITE\_CHECKERS), checkers, 0) for checkers in self.\_\_checkers)

#Количество чёрных шашек на поле

@property

def black\_checkers\_count(self) -> int:

return sum(reduce(lambda acc, checker: acc + (checker.type in BLACK\_CHECKERS), checkers, 0) for checkers in self.\_\_checkers)

# Счёт белых(для минимакса)

@property

def white\_score(self) -> int:

return sum(reduce(lambda acc, checker: acc + (checker.type == CheckerType.WHITE\_REGULAR) + (checker.type == CheckerType.WHITE\_QUEEN) \* 3, checkers, 0) for checkers in self.\_\_checkers)

# Счёт черных(для минимакса)

@property

def black\_score(self) -> int:

return sum(reduce(lambda acc, checker: acc + (checker.type == CheckerType.BLACK\_REGULAR) + (checker.type == CheckerType.BLACK\_QUEEN) \* 3, checkers, 0) for checkers in self.\_\_checkers)

**Game:**

from tkinter import Canvas, Event, messagebox

from pathlib import Path

from time import sleep

from checkers.field import Field

from checkers.move import Move

from checkers.constants import \*

from checkers.enums import CheckerType, SideType

#Класс игры

class Game:

def \_\_init\_\_(self, canvas: Canvas, x\_field\_size: int, y\_field\_size: int, exit\_callback):

self.\_\_canvas = canvas

self.\_\_field = Field(x\_field\_size, y\_field\_size)

self.\_\_player\_turn = True

self.\_\_exit\_callback = exit\_callback

self.\_\_hovered\_cell = Point()

self.\_\_selected\_cell = Point()

self.\_\_animated\_cell = Point()

self.\_\_draw()

# Если игрок играет за чёрных, то совершить ход противника

if PLAYER\_SIDE == SideType.BLACK:

self.\_\_handle\_enemy\_turn()

#Анимация перемещения шашки

def \_\_animate\_move(self, move: Move):

self.\_\_animated\_cell = Point(move.from\_x, move.from\_y)

self.\_\_draw()

# Создание шашки для анимации

animated\_checker = self.\_\_draw\_checker(move.from\_x, move.from\_y, is\_animated=True)

# 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):

for animated in animated\_checker:

self.\_\_canvas.move(animated, ANIMATION\_SPEED / 100 \* CELL\_SIZE \* dx, ANIMATION\_SPEED / 100 \* CELL\_SIZE \* dy)

self.\_\_canvas.update()

sleep(0.01)

self.\_\_animated\_cell = Point()

#Отрисовка сетки поля и шашек

def \_\_draw(self):

self.\_\_canvas.delete('all')

self.\_\_draw\_field\_grid()

self.\_\_draw\_checkers()

self.\_\_draw\_turn()

# Отрисовка чей ход

def \_\_draw\_turn(self):

if self.\_\_player\_turn:

self.\_\_canvas.create\_text(

self.\_\_field.x\_size \* CELL\_SIZE // 2,

self.\_\_field.y\_size \* CELL\_SIZE + 15,

text='Ваш ход',

anchor='center',

fill='green'

)

else:

self.\_\_canvas.create\_text(

self.\_\_field.x\_size \* CELL\_SIZE // 2,

self.\_\_field.y\_size \* CELL\_SIZE + 15,

text='Ход соперника',

anchor='center',

fill='red'

)

#Отрисовка сетки поля

def \_\_draw\_field\_grid(self):

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:

player\_moves\_list = self.\_\_get\_moves\_list(PLAYER\_SIDE)

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' )

# Отрисовка шашек

def \_\_draw\_checkers(self):

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.\_\_draw\_checker(x, y)

# Отрисовка отдельной шашки

def \_\_draw\_checker(self, x, y, is\_animated=False):

color = 'white' if self.\_\_field.type\_at(x, y) in WHITE\_CHECKERS else 'black'

checker = self.\_\_canvas.create\_oval(

x \* CELL\_SIZE + 5,

y \* CELL\_SIZE + 5,

(x + 1) \* CELL\_SIZE - 5,

(y + 1) \* CELL\_SIZE - 5,

fill=color,

\*\*({'tags': 'animated\_checker'} if is\_animated else {})

)

if self.\_\_field.type\_at(x, y) in (CheckerType.WHITE\_QUEEN, CheckerType.BLACK\_QUEEN):

queen\_mark = self.\_\_canvas.create\_oval(

x \* CELL\_SIZE + CELL\_SIZE // 2 - CELL\_SIZE // 10,

y \* CELL\_SIZE + CELL\_SIZE // 2 - CELL\_SIZE // 10,

x \* CELL\_SIZE + CELL\_SIZE // 2 + CELL\_SIZE // 10,

y \* CELL\_SIZE + CELL\_SIZE // 2 + CELL\_SIZE // 10,

fill='yellow',

\*\*({'tags': 'animated\_checker\_mark'} if is\_animated else {})

)

return [checker, queen\_mark]

return [checker]

# Событие перемещения мышки

def mouse\_move(self, event: Event):

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)

# Если ход игрока, то перерисовать

if self.\_\_player\_turn:

self.\_\_draw()

# Событие нажатия мышки

def mouse\_down(self, event: Event):

if not self.\_\_player\_turn: return

x, y = event.x // CELL\_SIZE, event.y // CELL\_SIZE

# Если точка не внутри поля

if not (self.\_\_field.is\_within(x, y)):

return

if PLAYER\_SIDE == SideType.WHITE:

player\_checkers = WHITE\_CHECKERS

elif PLAYER\_SIDE == SideType.BLACK:

player\_checkers = BLACK\_CHECKERS

else:

return

# Если нажатие по шашке игрока, то выбрать её

if self.\_\_field.type\_at(x, y) in player\_checkers:

self.\_\_selected\_cell = Point(x, y)

self.\_\_draw()

elif self.\_\_player\_turn:

move = Move(self.\_\_selected\_cell.x, self.\_\_selected\_cell.y, x, y)

# Если нажатие по ячейке, на которую можно походить

if move in self.\_\_get\_moves\_list(PLAYER\_SIDE):

self.\_\_handle\_player\_turn(move)

# Если не ход игрока, то ход противника

if not self.\_\_player\_turn:

self.\_\_handle\_enemy\_turn()

# Совершение хода

def \_\_handle\_move(self, move: Move, draw: bool = True) -> bool:

if draw:

self.\_\_animate\_move(move)

# Изменение позиции шашки

if self.\_\_field.at(move.from\_x, move.from\_y).type != CheckerType.NONE:

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:

self.\_\_field.at(x, y).change\_type(CheckerType.NONE)

has\_killed\_checker = True

# Изменение типа шашки, если она дошла до края, но только если на этом завершился её ход

if not has\_killed\_checker or not self.\_\_get\_required\_moves\_list(

SideType.BLACK if self.\_\_field.type\_at(move.to\_x, move.to\_y) in BLACK\_CHECKERS else SideType.WHITE,

move

):

if move.to\_y == 0 and self.\_\_field.type\_at(move.to\_x, move.to\_y) == CheckerType.WHITE\_REGULAR:

self.\_\_field.at(move.to\_x, move.to\_y).change\_type(CheckerType.WHITE\_QUEEN)

elif move.to\_y == self.\_\_field.y\_size - 1 and self.\_\_field.type\_at(move.to\_x,

move.to\_y) == CheckerType.BLACK\_REGULAR:

self.\_\_field.at(move.to\_x, move.to\_y).change\_type(CheckerType.BLACK\_QUEEN)

if draw:

self.\_\_draw()

return has\_killed\_checker

# Обработка хода игрока

def \_\_handle\_player\_turn(self, move: Move):

self.\_\_player\_turn = False

# Была ли убита шашка

has\_killed\_checker = self.\_\_handle\_move(move)

required\_moves\_list = list(filter(lambda required\_move: move.to\_x == required\_move.from\_x and move.to\_y == required\_move.from\_y, self.\_\_get\_required\_moves\_list(PLAYER\_SIDE)))

# Если есть ещё ход этой же шашкой

if has\_killed\_checker and required\_moves\_list:

self.\_\_player\_turn = True

self.\_\_selected\_cell = Point()

# Обработка хода противка(компьютера)

def \_\_handle\_enemy\_turn(self):

self.\_\_player\_turn = False

for move in self.\_\_minimax\_move(4, -float('inf'), float('inf'), PLAYER\_SIDE.opposite())[1]:

if not self.\_\_handle\_move(move) or not self.\_\_get\_required\_moves\_list(PLAYER\_SIDE.opposite(), move):

break

self.\_\_player\_turn = True

self.\_\_check\_for\_game\_over()

# Проверка на конец игры

def \_\_check\_for\_game\_over(self):

game\_over = False

white\_moves\_list = self.\_\_get\_moves\_list(SideType.WHITE)

if not white\_moves\_list:

# Белые проиграли

answer = 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 = messagebox.showinfo('Конец игры', 'Белые выиграли')

game\_over = True

if game\_over:

self.\_\_init\_\_(self.\_\_canvas, self.\_\_field.x\_size, self.\_\_field.y\_size, self.\_\_exit\_callback)

self.\_\_exit\_callback()

def \_\_minimax\_move(self, depth: int, alpha: float, beta: float, side: SideType) -> (float, list[Move]):

# Если достигнута максимальная глубина поиска или одна из сторон не имеет более доступных ходов (или обе), происходит выход из рекурсии

if depth == 0 or self.\_\_field.black\_score == 0 or self.\_\_field.white\_score == 0:

try:

# Возвращает оценку текущего состояния поля, учитывая отношение шашек одного игрока к другому

return self.\_\_field.white\_score / float(self.\_\_field.black\_score) \

if PLAYER\_SIDE == SideType.BLACK \

else self.\_\_field.black\_score / float(self.\_\_field.white\_score), []

except ZeroDivisionError:

# Если одна из сторон потеряла все шашки, возвращается положительная или отрицательная бесконечность

return float('inf'), []

# Если сторона, за которую играет компьютер, противоположна стороне игрока:

if side == PLAYER\_SIDE.opposite():

# Инициализация переменных для максимального значения и хода

max\_value = -float("inf")

max\_move = []

# Создание копии текущего состояния поля

field\_copy = Field.copy(self.\_\_field)

# Перебор всех доступных ходов для текущей стороны

for move in self.\_\_get\_moves\_list(side):

# Выполнение хода и проверка наличия обязательных ходов после него

killed = self.\_\_handle\_move(move, False) and bool(self.\_\_get\_required\_moves\_list(side, move))

# Рекурсивный вызов метода для оценки хода и получения последовательности ходов для следующих итераций

eval, seq\_moves = self.\_\_minimax\_move(depth - 1 if not killed else depth, alpha, beta,

side.opposite() if not killed else side)

# Обновление максимального значения и хода, если оценка лучше текущего максимума

if eval > max\_value:

max\_value = eval

max\_move = [move] + seq\_moves

# Обновление значения альфа и проверка условия для отсечения ветвления

alpha = max(alpha, eval)

if beta <= alpha:

break

# Восстановление состояния поля перед текущим ходом

self.\_\_field = Field.copy(field\_copy)

# Возврат максимального значения и соответствующего хода

return max\_value, max\_move

else:

# Аналогично, если сторона, за которую играет компьютер, совпадает со стороной игрока, но с противоположными условиями

# Код для минимизации оценки

# Инициализация переменных для максимального значения и хода

min\_value = float("inf")

min\_move = []

# Создание копии текущего состояния поля

field\_copy = Field.copy(self.\_\_field)

# Перебор всех доступных ходов для текущей стороны

for move in self.\_\_get\_moves\_list(side):

# Выполнение хода и проверка наличия обязательных ходов после него

killed = self.\_\_handle\_move(move, False) and bool(self.\_\_get\_required\_moves\_list(side, move))

# Рекурсивный вызов метода для оценки хода и получения последовательности ходов для следующих итераций

eval, seq\_moves = self.\_\_minimax\_move(depth - 1 if not killed else depth, alpha, beta,

side.opposite() if not killed else side)

# Обновление максимального значения и хода, если оценка лучше текущего максимума

if eval < min\_value:

min\_value = eval

min\_move = [move] + seq\_moves

# Обновление значения альфа и проверка условия для отсечения ветвления

beta = min(beta, eval)

if beta <= alpha:

break

# Восстановление состояния поля перед текущим ходом

self.\_\_field = Field.copy(field\_copy)

# Возврат минимального значения и соответствующего хода

return min\_value, min\_move

#Получение списка ходов

def \_\_get\_moves\_list(self, side: SideType, from\_move: Move = None) -> list[Move]:

moves\_list = self.\_\_get\_required\_moves\_list(side, from\_move=from\_move)

if self.\_\_player\_turn:

moves\_and\_evaluate = list(zip(moves\_list, (self.\_\_evaluate\_move(move) for move in moves\_list)))

moves\_and\_evaluate = sorted(moves\_and\_evaluate, key=lambda m: m[1], reverse=True)

moves\_list = list(map(lambda m: m[0], filter(lambda x: x[1] == moves\_and\_evaluate[0][1], moves\_and\_evaluate)))

if not moves\_list:

moves\_list = self.\_\_get\_optional\_moves\_list(side)

return moves\_list

#Оценка хода, насколько много вражеских шашек можно съесть, совершив указанный ход

def \_\_evaluate\_move(self, move: Move) -> int:

field\_copy = Field.copy(self.\_\_field)

killed = self.\_\_handle\_move(move, False)

value = int(killed) + max(list(self.\_\_evaluate\_move(m) for m in self.\_\_get\_required\_moves\_list(PLAYER\_SIDE, move)) + [0])

self.\_\_field = field\_copy

return value

#Получение списка обязательных ходов

def \_\_get\_required\_moves\_list(self, side: SideType, from\_move: Move = None) -> list[Move]:

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 list(filter(

lambda m: from\_move.to\_x == m.from\_x and from\_move.to\_y == m.from\_y,

moves\_list

)) if from\_move else moves\_list

#Получение списка необязательных ходов

def \_\_get\_optional\_moves\_list(self, side: SideType) -> list[Move]:

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

**Move:**

#Класс хода

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

**Point:**

#Класс координаты

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

**Auth:**

from pathlib import Path

from lk.cypher import encrypt, decrypt

# Путь к файлу с данными пользователей

USER\_DATA\_PATH = Path('users.data')

#Регистрация нового пользователя

def register(username, password):

user\_data = f"{encrypt(username)}:{encrypt(password)}\n"

with USER\_DATA\_PATH.open('a', encoding='utf8') as file:

file.write(user\_data)

#Авторизация пользователя

def authenticate(username, password):

#если файл отсутствует

if not USER\_DATA\_PATH.exists():

return False

#если файл открылся

with USER\_DATA\_PATH.open('r', encoding='utf8') as file:

for line in file:

encrypted\_username, encrypted\_password = line.strip().split(':')

if username == decrypt(encrypted\_username) and password == decrypt(encrypted\_password):

return True

return False

def user\_exist(username):

if not USER\_DATA\_PATH.exists():

return False

# если файл открылся

with USER\_DATA\_PATH.open('r', encoding='utf8') as file:

for line in file:

encrypted\_username, \_ = line.strip().split(':')

if username == decrypt(encrypted\_username):

return True

return False

**Cypher:**

secret\_key = "keykey"

#Алгоритм Виженера

def vigenere(text: str, key: str, encrypt=True):

result = ""

for i in range(len(text)):

letter\_n = ord(text[i])

key\_n = ord(key[i % len(key)])

if encrypt:

value = (letter\_n + key\_n) % 1114112

else:

value = (letter\_n - key\_n) % 1114112

result += chr(value)

return result

#Функиця Шифрования

def encrypt(text: str):

return vigenere(text=text, key=secret\_key, encrypt=True)

#Функция Дешифрования

def decrypt(text: str):

return vigenere(text=text, key=secret\_key, encrypt=False)

**Main:**

from tkinter import Tk, Canvas, PhotoImage, messagebox, Button, Label, Entry, END

from checkers.game import Game

from checkers.constants import X\_SIZE, Y\_SIZE, CELL\_SIZE

from lk.auth import authenticate, register, user\_exist

# функция начала игры

def start\_game():

# Создание окна

main\_window = Tk()

main\_window.title('Канадские шашки')

main\_window.resizable(False, False)

show\_auth(main\_window)

main\_window.mainloop()

# функция отображения личного кабинета

def show\_lk(tk):

tk.wm\_minsize(340, 260)

title\_label = Label(tk, text="Личный кабинет", font=("Arial", 16))

title\_label.pack(pady=10)

start\_game\_button = Button(tk, text='Начать игру', command=lambda: [start\_game\_button.pack\_forget(),

title\_label.pack\_forget(),

exit\_button.pack\_forget(),

show\_game(tk)])

start\_game\_button.pack(pady=10)

exit\_button = Button(tk, text='Выйти', command=lambda: [start\_game\_button.pack\_forget(),

title\_label.pack\_forget(),

exit\_button.pack\_forget(),

show\_auth(tk)])

exit\_button.pack(pady=10)

# функция отображения авторизации/регистрации

def show\_auth(tk):

tk.wm\_minsize(340, 260)

def switch\_to\_register():

username\_entry.delete(0, END)

password\_entry.delete(0, END)

login\_button.pack\_forget()

register\_button.pack(pady=10)

switch\_register\_button.pack\_forget()

switch\_login\_button.pack(pady=15)

title\_label.config(text="Регистрация")

def switch\_to\_login():

username\_entry.delete(0, END)

password\_entry.delete(0, END)

register\_button.pack\_forget()

login\_button.pack(pady=10)

switch\_login\_button.pack\_forget()

switch\_register\_button.pack(pady=15)

title\_label.config(text="Авторизация")

def perform\_login():

username = username\_entry.get()

password = password\_entry.get()

if authenticate(username, password):

title\_label.pack\_forget()

username\_label.pack\_forget()

username\_entry.pack\_forget()

password\_label.pack\_forget()

password\_entry.pack\_forget()

login\_button.pack\_forget()

switch\_register\_button.pack\_forget()

show\_lk(tk)

else:

messagebox.showerror("Ошибка", "Неверное имя пользователя или пароль")

def perform\_register():

username = username\_entry.get()

password = password\_entry.get()

if not username or not password:

return messagebox.showerror("Ошибка", "Необходимо заполнить все поля!")

if len(username) < 3 or len(username) > 20:

return messagebox.showerror("Ошибка", "Имя пользователя должно иметь длину от 3 до 20 символов")

if len(password) < 3 or len(password) > 20:

return messagebox.showerror("Ошибка", "Пароль должен иметь длину от 3 до 20 символов")

if user\_exist(username):

return messagebox.showerror("Ошибка", "Пользователь с таким именем уже существует")

try:

register(username, password)

messagebox.showinfo("Успешно", "Регистрация прошла успешно!")

except Exception as e:

messagebox.showerror("Ошибка")

title\_label = Label(tk, text="Авторизация", font=("Arial", 16))

title\_label.pack(pady=10)

username\_label = Label(tk, text="Имя пользователя:")

username\_label.pack()

username\_entry = Entry(tk)

username\_entry.pack(pady=5)

password\_label = Label(tk, text="Пароль:")

password\_label.pack()

password\_entry = Entry(tk, show="\*")

password\_entry.pack(pady=5)

register\_button = Button(tk, text="Зарегистрироваться", command=perform\_register)

login\_button = Button(tk, text="Войти", command=perform\_login)

login\_button.pack(pady=10)

switch\_register\_button = Button(tk, text="Регистрация", command=switch\_to\_register)

switch\_register\_button.pack(pady=15)

switch\_login\_button = Button(tk, text="Авторизация", command=switch\_to\_login)

# функция отображения игры

def show\_game(tk):

tk.wm\_minsize(width=CELL\_SIZE \* X\_SIZE, height=CELL\_SIZE \* Y\_SIZE + 30)

# Создание холста

main\_canvas = Canvas(tk, width=CELL\_SIZE \* X\_SIZE, height=CELL\_SIZE \* Y\_SIZE + 30)

main\_canvas.pack()

game = Game(main\_canvas, X\_SIZE, Y\_SIZE, lambda: [main\_canvas.pack\_forget(), show\_lk(tk)])

main\_canvas.pack()

main\_canvas.bind("<Motion>", game.mouse\_move)

main\_canvas.bind("<Button-1>", game.mouse\_down)

def main():

start\_game()

# Точка входа

if \_\_name\_\_ == '\_\_main\_\_':

main()