**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 «Нарды»

Р.02069337. 22/2379-72

Листов 80

**Руководитель разработки**:

доцент каф. ИВК, к.т.н.

*Шишкин Вадим Викторинович*

« » 2023 г.

**Исполнитель**:

студент гр. ИСТбд-23

*Штакал Илья Борисович*

« » 2023 г.

**2023**

**Содержание**

Аннотация.......................................................................................1

Техническое задание.....................................................................3

Пояснительная записка ................................................................12

Руководство программиста ..........................................................20

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

**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 «Нарды»

**Инв. № подл.Инв. № подл.**

**Подп. и датаПодп. и дата**

**Взам. инв. №Взам. инв. №**

**Инв. № дубл.Инв. № дубл.**

**Подп. и датаПодп. и дата**

Р.02069337. 22/2379-72 ТЗ-02

Листов 8

**Исполнитель**:

студент гр. ИСТбд-23

*Штакал Илья Борисович*

« » 2023 г.

**2023**

**Введение**

Компьютерная логическая игра «Нарды». Игра для 2 игроков.

Приложение должно соответствовать правилам игры, приведенным ниже.

**Компоненты Игры**

Игровая доска, 15 светлых и 15 темных круглых шашек, игральные кубики.

# Терминология

У каждого игрока на доске имеется свой «Двор» и «Дом».

**Бар:** область на месте перегиба доски, разделяющая 2 части поля.

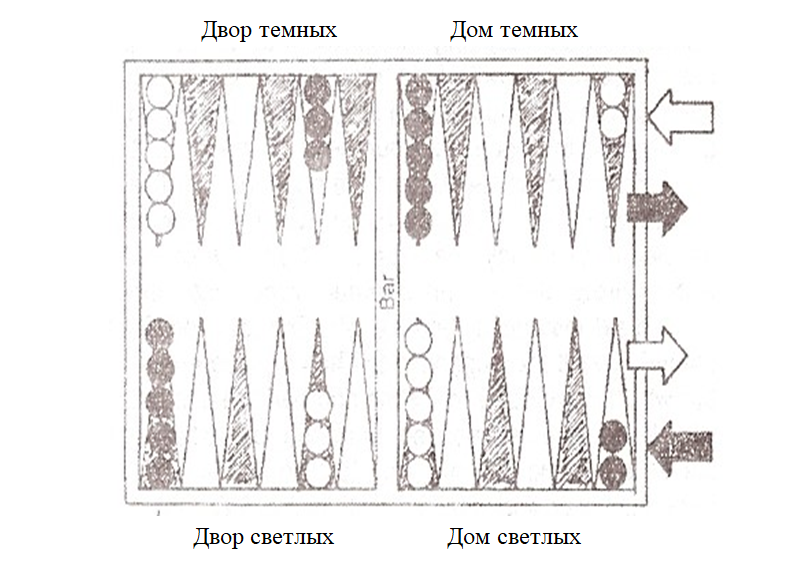
**Пункт:** остроконечный треугольник на поле, отмечающий места, где могут стоять шашки

# Цель Игры

Цель игры - перевести все свои шашки в свой дом и снять их с доски. Побеждает игрок, который первым снял все свои шашки.

# Начальная Расстановка

Шашки нужно расставить в начальную позицию, показанную ниже (рис. 1). Светлые здесь играют от правого верхнего угла, через всю левую сторону поля, в правый нижний угол, то есть против часовой стрелки. Темные играют в обратном направлении, то есть по часовой стрелке. Эти направления должны соблюдаться, ходы в обратном направлении запрещены.



# Рис. 1. Начальная расстановка

# Ход Игры

**Бросок кубиков:** в каждом ходе используются 2 кубика. Игрок бросает кубики и перемещает шашки.

**Перемещение шашек:** шашки можно переместить на число, указанное на кубиках. Возможно:

а) сделать один ход на расстояние, равное сумме значений двух кубиков. При этом надо учитывать, что такой ход все же состоит из двух перемещений, и промежуточный пункт, с которого начинается второе перемещение, не должен быть занят противником (условия для этого см. примечание ниже). Однако, игрок вправе сам выбирать, на какое значение одного из кубиков делается первое перемещение.

б) сделать 2 хода разными шашками, каждый для своего кубика.

**Пример:** бросок кубиков дал числа 2 и 5. У игрока есть выбор: либо сделать два хода одной шашкой (2 + 5 или 5 + 2), либо переместить две шашки, одну на 2 пункта, другую на 5.

**Примечание:** все пункты могут быть заняты, кроме тех, на которых стоит 2 и более шашек противника. Если возможен ход только на одно из чисел, используется большее число. Если ни один ход не возможен, ход заканчивается, и действует другой игрок.

**Захват шашек:** одиночную шашку противника в пункте можно «побить» и отправить на бар, если Ваша шашка останавливается в этом пункте (или это промежуточный пункт, см. «Перемещение шашек»). Эта шашка снимается с доски и кладется на бар. Владельцу захваченных шашек запрещается делать какие-либо ходы, пока он не вернет свои шашки с бара в игру, в дом противника. Шашка может вернуться в игру, только если может сделать ход в свободный пункт в доме противника. Одиночная шашка противника при этом может быть захвачена. Если игрок не может вывести шашки с бара, он пропускает ход.

**Дубль:** если на кубиках выпало два одинаковых числа, то каждое из них удваивается.

**Пример:** если выпали двойки на обоих кубиках, у игрока есть разные варианты комбинаций. Он может передвинуть одну шашку 4 раза на 2 пункта; или же походить три раза на 2 пункта одной шашкой, и на два пункта – другой; или, например, двумя шашками по два пункта, а одной – на четыре (дважды по два), и т.д.

**Выбрасывание шашек:** выбрасывать шашки можно, только если все 15 шашек игрока находятся в его доме. Выбрасывая необходимое значение на кубиках, игрок снимает свои шашки с доски. Шашка снимается, если выпавшее число равно или больше расстояния шашки до края доски. Можно использовать как сумму броска, так и отдельные значения кубиков, а также перемещать шашки внутри дома ближе к краю.

# Конец Игры

Один изигроков успел выбросить все свои шашки с поля раньше оппонента.

**1. Основания для разработки**

Учебный план направления 09.03.02 «Информационные системы и технологии» и распоряжение по факультету от 20.10.2023 №12-p.

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

**2.1. Функциональное назначение**

Требуется разработать однопользовательское десктопное приложение по игре в нарды с графическим интерфейсом в среде Windows.

**2.2 Требования к функциональным характеристикам**

2.2.1 Требования к структуре приложения

Приложение должно быть разработано в виде нескольких модулей, взаимодействующих между собой с использованием дополнительных информационных файлов.

2.2.2 Требования к составу функций приложения

В приложении должны быть реализованы в графическом режиме следующие основные функции:

- регистрация/авторизация пользователя;

- отрисовка игрового поля;

- взаимодействие с пользователем ;

- интерактивные прием, проверка правильности и отрисовка хода пользователя;

- проверка окончания игры;

- вычисление, проверка правильности и отрисовка хода компьютера;

- информирование пользователя об окончании игры и победителе.

2.2.3 Требования к организации информационного обеспечения, входных и выходных данных

В приложении должен быть реализован графический интерфейс взаимодействия с пользователем. Отдельно выделены папки под графические файлы, шрифт, заготовку объектов и карты, аудио эффектов, а также для самого кода. Логин и пароль пользователя должны вводиться с клавиатуры. Логины и пароли зарегистрированных пользователей должны храниться в отдельном файле или базе данных в зашифрованном виде.

**2.3 Требования к надёжности**

Поддержка непрерывной и стабильной работы компьютера.

**2.4 Требования к информационной и программной совместимости**

Рекомендуется к использованию на Windows 10 и выше.

При создании программы используются встроенные библиотеки “random”, “os”, ”sys”, “timeit”. И сторонние библиотеки “pygame”, “PyQt5”

Разработка ведётся в “PyCharm community edition 2023.2.1” на версии языка программирования Python 3.11.5.

**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

Диск CD-R должен храниться при комнатной температуре, в диапазоне от 20°C до 25°C. Рекомендуется хранить диск в условиях с относительной влажностью воздуха от 20% до 50%. Диск CD-R должен храниться в темном месте, защищенном от прямых солнечных лучей и других источников яркого света. Для предотвращения повреждения диска CD-R рекомендуется хранить его в специальных пластиковых коробках или футлярах, предназначенных для хранения CD-дисков.

2.6 3 Сроки хранения

Срок хранения – до июля 2024 года.

**3. Требования к программной документации**

1. «Техническое задание» на реализуемое приложение должно соответствовать ГОСТ 19.201-78 «Техническое задание. Требования к содержанию и оформлению»;
2. «Пояснительная записка» должна соответствовать ГОСТ 19.404-79 «Пояснительная записка. Требования к содержанию и оформлению»;
3. «Руководство программиста» должна соответствовать ГОСТ 19.504-79 «Руководство программиста. Требования к содержанию и оформлению»;
4. Оформление программного кода приложения должно быть в соответствии с ГОСТ 19.401-79 «Текст программы. Требования к содержанию и оформлению».

**4. Стадии и этапы разработки**

Выполнение работы проходит следующие этапы:

1) Написание технического задания;

2) Написание пояснительной записки;

3) Разработка и отладка прототипа приложения;

4) Разработка и отладка окончательного варианта приложения;

5) Написание руководства программиста;

6) Оформление пояснительной записки курсовой работы;

7) Подготовка презентации.

8) Защита курсовой работы.

**5. Порядок контроля и приёмки**

После утверждения и закрепления тем за студентами, руководитель проводит необходимые консультации на практических занятиях, контролирует ход выполнения работы в целом.

Созданный программный продукт предоставляется студентом руководителю с помощью репозитория GitHub в исходном виде. В случае наличия каких-либо замечаний, студент может доработать свой программный продукт, записав новую версию по той же ссылке. Программные документы также размещаются в репозитории GitHub. Окончательная (принятая) версия документации предоставляется в печатном виде.

График защиты курсовых работ составляется руководителем и доводится до сведения студентов. Студент обязан убедиться в корректности функционирования приложения до защиты. Проблемы, обнаруженные непосредственно при защите, являются основанием для её переноса (в соответствии с графиком).

В начале защиты студент сообщает название приложения, кратко формулирует его назначение и указывает основные особенности. В процессе защиты – демонстрирует используемую(мые) структуры данных и алгоритмы, акцентируя внимание на наиболее важных и интересных, демонстрирует работоспособность самого приложения.

Оценка работы осуществляется руководителем с учётом качества её выполнения, включая качество документации, полноты учёта предъявляемых требований, выступления с презентацией и ответов на вопросы в ходе защиты, а также соблюдения сроков выполнения и защиты курсовой работы, включая работу на практических занятиях.

Основаниями для получения неудовлетворительной оценки могут являться серьёзное несоответствие программного продукта предъявляемым требованиям, неработоспособность приложения, наличие существенных элементов заимствования из чужих работ как в программном коде или интерфейсе приложения, так и в документации, а также слабая ориентация студента в представляемой работе. При неудовлетворительной оценке руководитель определяет направления и объём доработки программного продукта.

**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 «Нарды»

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

Пояснительная записка

Р.02069337. 22/2379-72 ПЗ-02

Листов 7

**Руководитель разработки**:

доцент каф. ИВК, к.т.н.,

*Шишкин Вадим Викторинович*

« » 2023 г.

**Исполнитель**:

студент гр. ИСТбд-23

*Штакал Илья Борисович*

« » 2023 г.

**2023**

**Введение**

Приложение «Нарды».

В работе выбраны такие структуры данных, как массив и словарь. Структура данных — массив была выбрана потому, что эта структура имеет множество преимуществ:

1. Массивы обеспечивают произвольный доступ к элементам. Это ускоряет доступ к элементам по положению;
2. Массивы хранят несколько данных похожих типов с одним и тем же именем;
3. В массиве данные организованны таким образом, что ими легко и удобно манипулировать.

Структура данных — словарь был выбран, потому что он также обладает рядом преимуществ:

1. **В словарях доступ к элементам выполняется по ключу, а не по индексу**;
2. Словари могут содержать объединенные данные в виде записей;
3. **Словари имеют переменную длину;**
4. **Словари представляют неупорядоченные коллекции произвольных объектов.**

**1. Проектная часть**

**1.1 Постановка задачи на разработку приложения**

Определяется заданием на курсовую работу

**1.2 Математические методы**

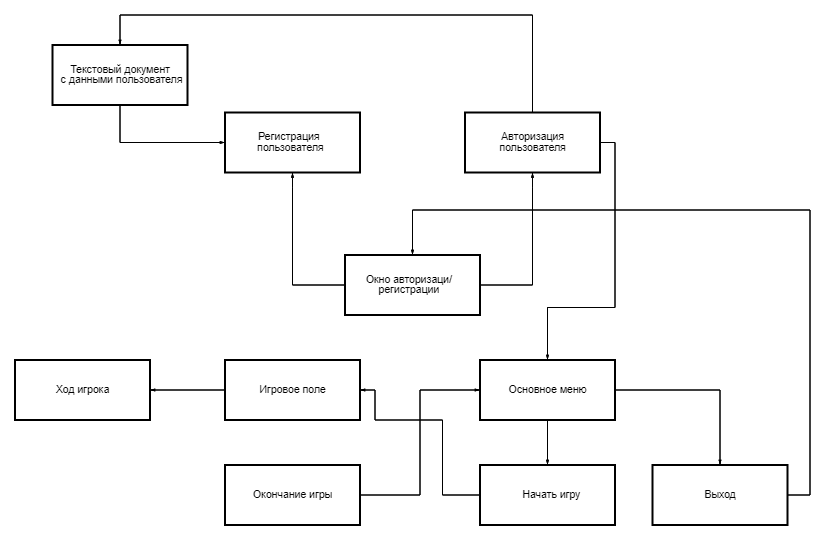
В качестве математической модели для представления поля был выбран двумерный массив, он позволяет легко записать положение всех шашек и оперировать ими, а также может быть легко изменен, что упрощает вывод хода пользователя и компьютера. Каждый элемент двумерного массива является экземпляром класса описывающего ячейку поля. Каждая шашка на игровом поле представляет собой экземпляр класса описывающего единичную шашку.

Для визуализация игрового поля и шашек используются изображения в формате .png. Визуализация элементов индикации взаимодействия реализуется с помощью методов графической библиотеки.

Расчёт хода компьютера осуществляется в соответствии со следующим алгоритмом:

* функция проверяет, возможен ли ход в соответствии со значениями выпавших кубиков
* если ход невозможен, очередь хода переходит к противнику
* если ход возможен, функция проверяет, есть ли съеденные шашки, если есть, то функция сначала возвращает их на поле, затем (или же если их нет), проходиться по всем шашкам и для каждой совершает комбинацию ходов, в соответствии со значениями выпавших кубиков
* далее каждой комбинации ходов присваивается вес (расчет веса комбинации вычисляет отдельная вспомогательная функция)
* все возможные комбинации ходов добавляются в отдельный массив
* с помощью ещё одной вспомогательной функции выбирается лучшая комбинация ходов из массива
* компьютер совершает ход в соответствии с лучшей комбинацией

**1.3 Архитектура и алгоритмы**

1.3.1. Архитектура

1.3.2 Алгоритм авторизации

Алгоритм осуществляет авторизацию пользователя. Пользователь вводит логин и пароль, а затем алгоритм проверяет, зарегистрирован ли такой пользователь.

1.3.3 Алгоритм регистрации

Данный алгоритм осуществляет регистрацию пользователя, зашифровывая данные с помощью алгоритма шифрования и записывая данные в текстовый документ .txt

1.3.4 Алгоритм шифрования

Данный криптографический алгоритм предназначен для защиты логина и пароля пользователя. Каждый символ введённой строки сначала преобразовывается в число из таблицы Unicode, а потом шифруется с помощью дополнительной функции. Зашифрованные данные записываются в текстовый документ.



1.3.5 Алгоритм дешифрования

Данный алгоритм осуществляет расшифровку логина и пароля пользователя. Каждый элемент сначала дешифруется с помощью дополнительной функции, а потом преобразовывается в символ из таблицы Unicode. После алгоритм сравнивает расшифрованные данные с введёнными данными пользователем.



1.3.6 Алгоритм проверки возможности хода

Данный алгоритм проверяет возможность сходить шашкой в соответствии с правилами игры. Проверяется выбрана ли верная шашка(своего цвета). Если всё выбрано верно выполняется алгоритм подсвечивания возможных ходов

1.3.7 Алгоритм подсвечивания возможных ходов

Алгоритм используя данные, полученные из алгоритма возможности хода, подсвечивает зеленым все возможные ходы.

1.3.8 Алгоритм хода игрока

Данный алгоритм совершает ход игрока. Для начала идёт проверка возможности хода, затем они подсвечиаются. После выбора хода игрока, шашка переририсовывается.

1.3.9 Алгоритм хода компьютера

Данный алгоритм осуществляет ход противника(компьютера). Данный алгоритм совершает ход шашкой, соответсвуя правилам игры.

1.3.10 Алгоритм определения победителя

Данный алгоритм определяет победителя в сыгранной игре.

**1.4 Тестирование**

Весь процесс тестирования проходил вручную, без привлечения специального ПО. На протяжении всего хода разработки, по мере добавления новых функций программы, использовалось системное тестирование новых функций, для устранения возникших в ходе написания ошибок. После положительных результатов тестирования функция считалась внедренной.

**2. Источники, использованные при разработке**

1. Rusnardy [Электронный ресурс]: Нарды Правила Игры – URL:

<https://rusnardy.ru/learn/back> (дата обращения: 5.11.2023)

1. YouTube [Электронный ресурс]: Уроки по Pygame – YouTube – URL:

<https://www.youtube.com/playlist?list=PLA0M1Bcd0w8xg_hyqpPpHdbZnPubSyIQ_> (дата обращения: 14.11.2023)

##### Pygame [Электронный ресурс]: Pygame documentation  – URL:

<https://pg1.readthedocs.io/en/latest/index.html> (дата обращения: 12.11.2023)

1. YouTube [Электронный ресурс]: Изучение библиотеки Pygame с нуля– YouTube – URL:

<https://www.youtube.com/playlist?list=PLAMm2eUmBSn1FuC_Smxom8mlEsRthy1PD> (дата обращения: 15.11.2023)

1. Wikipedia [Электронный ресурс]: Нарды – URL:

<https://ru.wikipedia.org/wiki/Нарды> (дата обращения 07.11.2023)

**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 «Нарды»

Руководство программиста

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

Р.02069337. 22/2379-72 РП-01

Листов 14

**Исполнитель**:

студент гр. ИСТбд-23

*Штакал Илья Борисович*

« » 2023 г.

**2023**

**1. Назначение и условия применения программы**

**1.1 Назначение и функции, выполняемые приложением**

Компьютерная логическая игра «Нарды». Игра для 2 игроков.

Приложение должно соответствовать правилам игры, приведенным ниже.

**Компоненты Игры**

Игровая доска, 15 светлых и 15 темных круглых шашек, игральные кубики.

# Терминология

У каждого игрока на доске имеется свой «Двор» и «Дом».

**Бар:** область на месте перегиба доски, разделяющая 2 части поля.

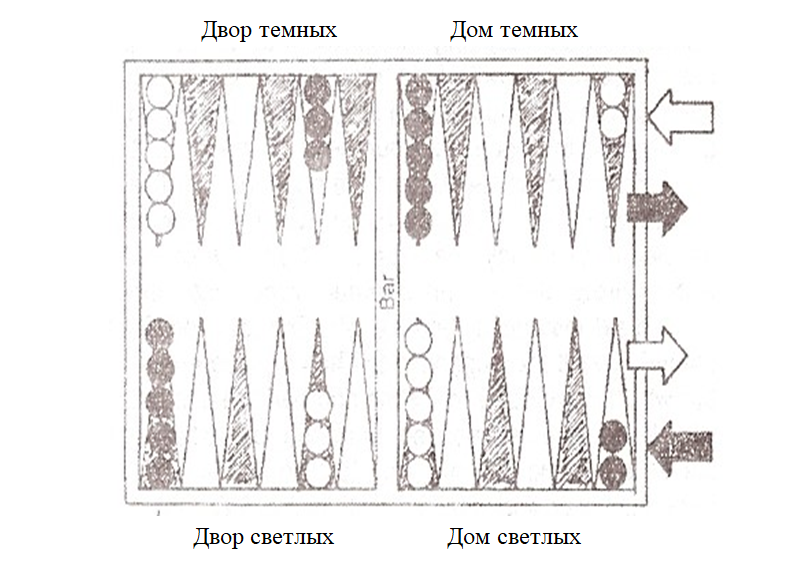
**Пункт:** остроконечный треугольник на поле, отмечающий места, где могут стоять шашки

# Цель Игры

Цель игры - перевести все свои шашки в свой дом и снять их с доски. Побеждает игрок, который первым снял все свои шашки.

# Начальная Расстановка

Шашки нужно расставить в начальную позицию, показанную ниже (рис. 1). Светлые здесь играют от правого верхнего угла, через всю левую сторону поля, в правый нижний угол, то есть против часовой стрелки. Темные играют в обратном направлении, то есть по часовой стрелке. Эти направления должны соблюдаться, ходы в обратном направлении запрещены.



# Рис. 1. Начальная расстановка

# Ход Игры

**Бросок кубиков:** в каждом ходе используются 2 кубика. Игрок бросает кубики и перемещает шашки.

**Перемещение шашек:** шашки можно переместить на число, указанное на кубиках.

Возможно:

а) сделать один ход на расстояние, равное сумме значений двух кубиков. При этом надо учитывать, что такой ход все же состоит из двух перемещений, и промежуточный пункт, с которого начинается второе перемещение, не должен быть занят противником (условия для этого см. примечание ниже). Однако, игрок вправе сам выбирать, на какое значение одного из кубиков делается первое перемещение.

б) сделать 2 хода разными шашками, каждый для своего кубика.

**Пример:** бросок кубиков дал числа 2 и 5. У игрока есть выбор: либо сделать два хода одной шашкой (2 + 5 или 5 + 2), либо переместить две шашки, одну на 2 пункта, другую на 5.

**Примечание:** все пункты могут быть заняты, кроме тех, на которых стоит 2 и более шашек противника. Если возможен ход только на одно из чисел, используется большее число. Если ни один ход не возможен, ход заканчивается, и действует другой игрок.

**Захват шашек:** одиночную шашку противника в пункте можно «побить» и отправить на бар, если Ваша шашка останавливается в этом пункте (или это промежуточный пункт, см. «Перемещение шашек»). Эта шашка снимается с доски и кладется на бар. Владельцу захваченных шашек запрещается делать какие-либо ходы, пока он не вернет свои шашки с бара в игру, в дом противника. Шашка может вернуться в игру, только если может сделать ход в свободный пункт в доме противника. Одиночная шашка противника при этом может быть захвачена. Если игрок не может вывести шашки с бара, он пропускает ход.

**Дубль:** если на кубиках выпало два одинаковых числа, то каждое из них удваивается.

**Пример:** если выпали двойки на обоих кубиках, у игрока есть разные варианты комбинаций. Он может передвинуть одну шашку 4 раза на 2 пункта; или же походить три раза на 2 пункта одной шашкой, и на два пункта – другой; или, например, двумя шашками по два пункта, а одной – на четыре (дважды по два), и т.д.

**Выбрасывание шашек:** выбрасывать шашки можно, только если все 15 шашек игрока находятся в его доме. Выбрасывая необходимое значение на кубиках, игрок снимает свои шашки с доски. Шашка снимается, если выпавшее число равно или больше расстояния шашки до края доски. Можно использовать как сумму броска, так и отдельные значения кубиков, а также перемещать шашки внутри дома ближе к краю.

# Конец Игры

Один изигроков успел выбросить все свои шашки с поля раньше оппонента.

**Основные возможности приложения:**

1. Осуществление регистрации/авторизации игрока;
2. Корректное графическое отображение ходов на поле;
3. Проверка правильности ходов в соответствии с правилами и запрет на неправильные ходы;
4. Проверка на окончание игры;
5. Определение победителя.

**1.2 Условия, необходимые для использования приложения**

1. Рекомендуется к использованию на Windows 10 и выше
2. Инструментальная среда разработки: PyCharm Community Edition 2023.2.1.
3. Версия языка Python: 3.11.5
4. Используемые библиотеки: “random”, “os”,“sys”,“timeit”, “pygame”, “PyQt5”

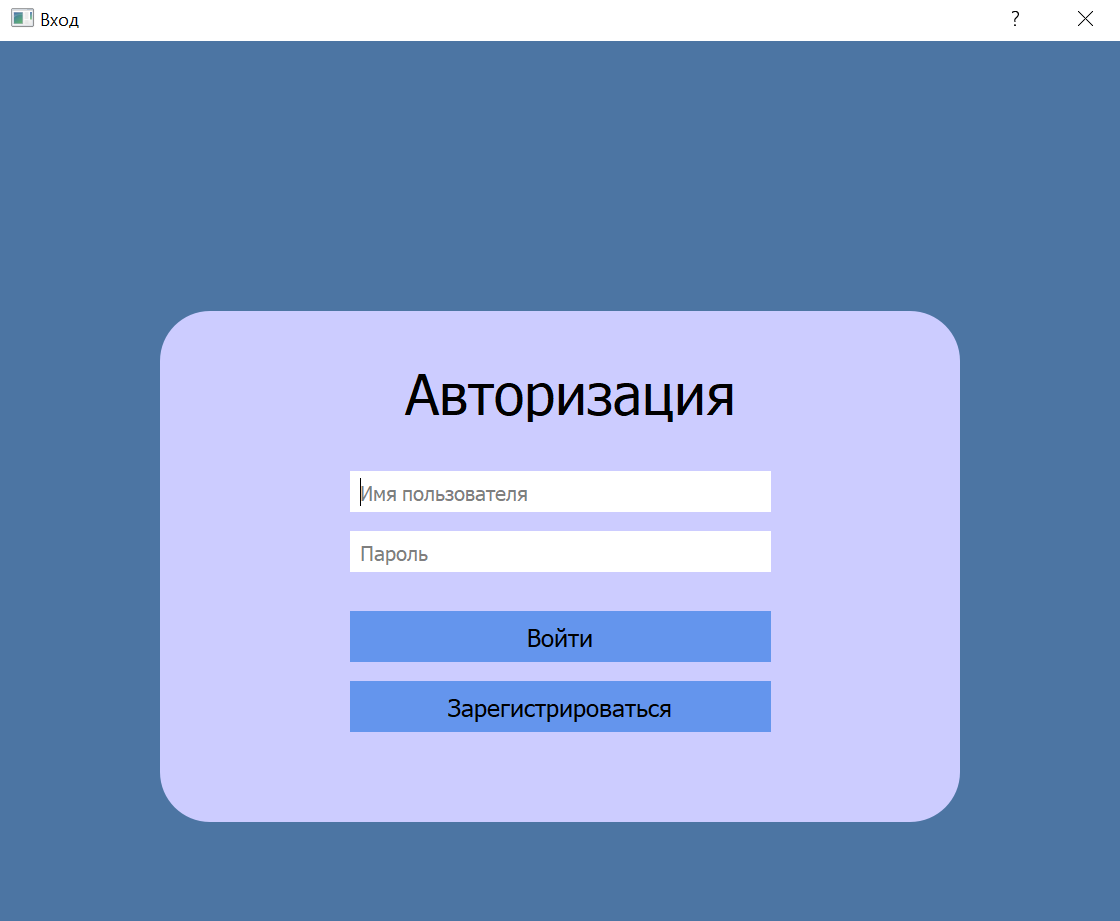
**2. Характеристики программы**

**2.1 Характеристики приложения**

Программный код состоит из 1800 строчек. Использованы 2 структуры данных (словарь и массивы) и реализованы 9 алгоритмов.

В приложении используются библиотеки:

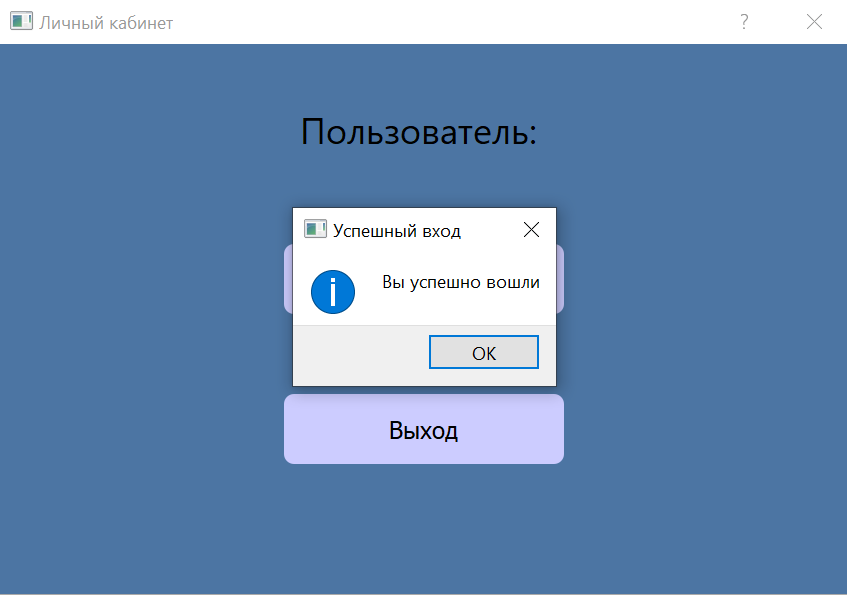
1. “pygame” – это кроссплатформенный набор модулей python, предназначенных для написания видеоигр.
2. **“PyQt5”**является одним из наиболее часто используемых модулей для создания**GUI приложений**в Python.
3. “os” – библиотека функций для работы с операционной системой.
4. “sys” – это встроенный модуль Python, который предоставляет доступ к системным функциям и переменным.
5. “timeit” – это простой интерфейс для быстрого измерения вре-мени выполнения небольших блоков кода.
6. “random” – этот модуль реализует генератор псевдослучайных чисел.
7. Запуск приложения. Окно авторизации

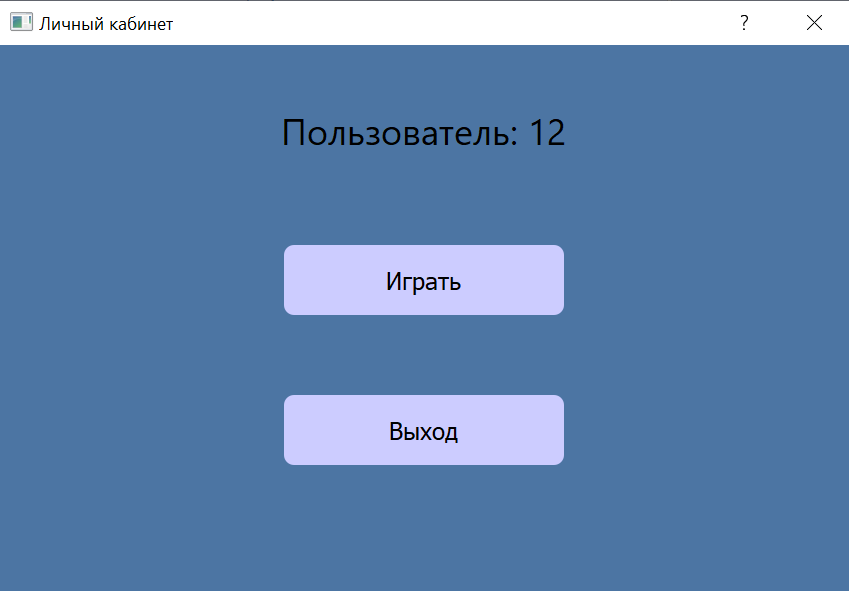
В данном окне игрок может зарегистрироваться, нажав на кнопку «Зарегистрироваться» или войти в личный кабинет, нажав на кнопку «Войти»

1. Личный кабинет игрока.

После того, как игрок авторизовался, появляется окно личного кабинета и успешной авторизации. В личном кабинете можно выбрать два действия:

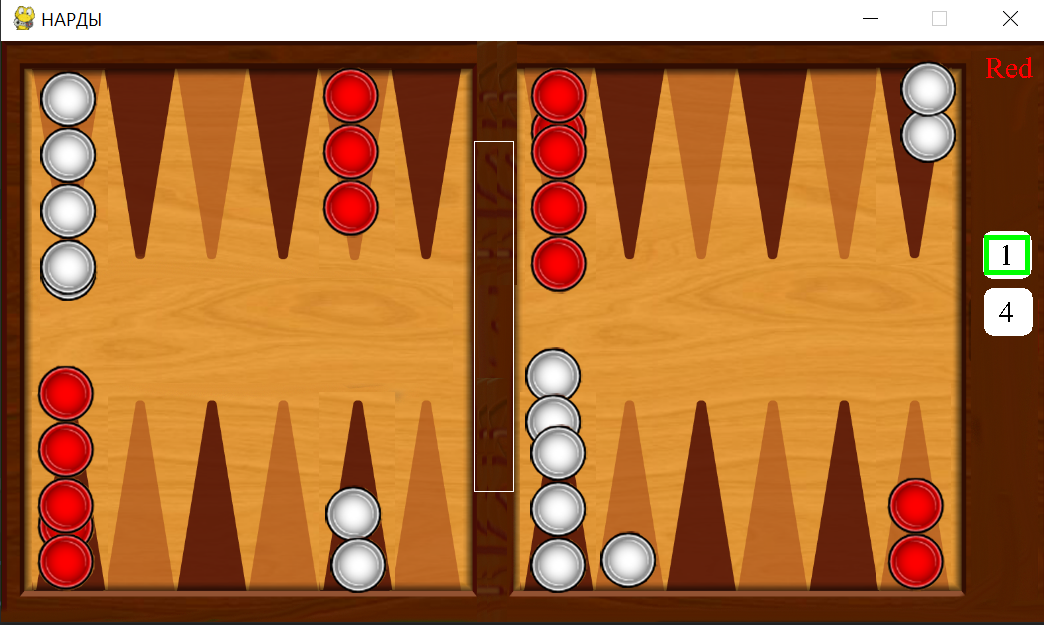
* Начать играть, нажав на кнопку «Играть»
* Выйти из личного кабинета, нажав на кнопку «Выйти»

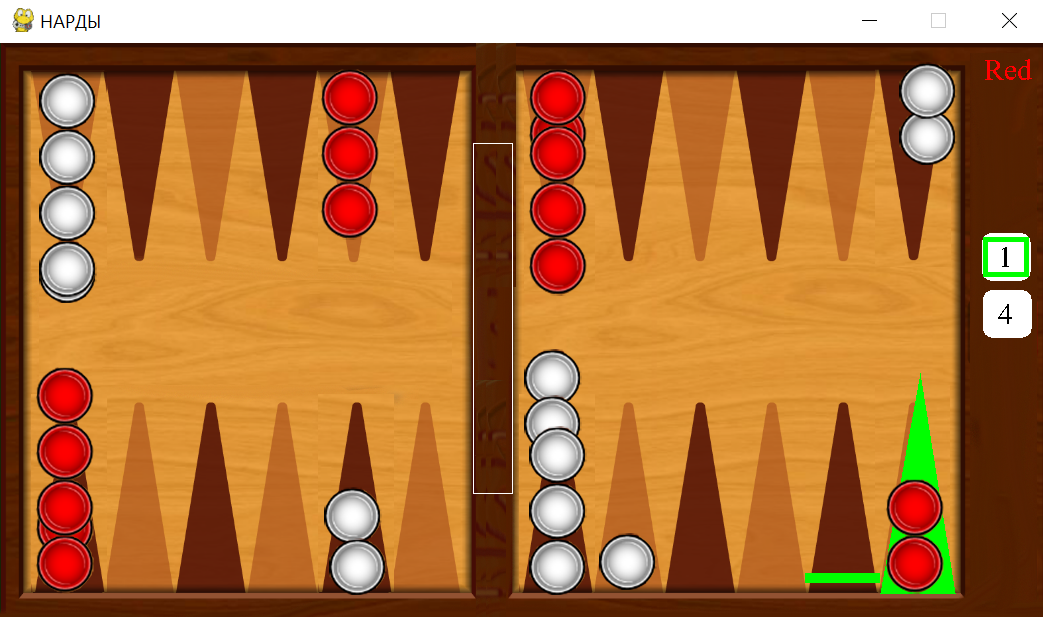
При нажатии на кнопку «Играть» открывается окно с игровым полем. А при нажатии на кнопку «Выйти» игрок возвращается в предыдущее окно авторизации.



1. Окно игрового поля.

Доска для игры в нарды состоит из 24 полей, или пунктов, имеющих вид узких вытянутых треугольников. Пункты сгруппированы по 6.

У каждого игрока имеется по 15 шашек одного цвета. Также имеется две игровые кости. На доске располагаются шашки игрока и шашки соперника. Ход можно совершить, нажав на шашку. При нажатии на шашку её возможные ходы подсвечиваются.



**2.2 Особенности реализации приложения**

В работе выбраны такие структуры данных, как массив и словарь. Массив, представляет собой структуру данных, состоящую из набора элементов (значений или переменных), каждый из которых идентифицируется, по крайней мере, одним индексом или ключом массива. Эта структура данных была выбрана потому, что эта структура имеет множество преимуществ:

1. Массивы обеспечивают произвольный доступ к элементам. Это ускоряет доступ к элементам по положению;
2. Массивы хранят несколько данных похожих типов с одним и тем же именем;
3. В массиве данные организованны таким образом, что ими легко и удобно манипулировать.

Словари представляют собой структуры данных, в которых уникальные ключи отображают значения. Ключ и значение разделяются двоеточием, пары ключ - значения отделяются запятыми, а словарь целиком ограничивается фигурными скобками {}. Эта структура данных была выбрана, потому что она обладает рядом значительных преимуществ:

1. **В словарях доступ к элементам выполняется по ключу, а не по индексу**;
2. Словари могут содержать объединенные данные в виде записей;
3. **Словари имеют переменную длину;**
4. **Словари представляют неупорядоченные коллекции произвольных объектов.**

Вместо массива и словаря можно было бы использовать другие структуры данных такие как, кортеж, множество. В работе были выбраны структуры данных массив и словарь, так как у них есть ряд преимуществ, который описан выше.

**3. Обращение к программе**

Алгоритмы:

1. Алгоритм регистрации

Данный алгоритм осуществляет регистрацию пользователя. Пользователь вводит данные в поля: логин и пароль. Затем каждая строка обрабатывается отдельно. Если такие данные уже есть в текстовом документе, то алгоритм не разрешает зарегистрироваться под этим логином. Данные зашифровываются с помощью алгоритма шифрования. Потом зашифрованные данные записываются в текстовый документ .txt

1. Алгоритм авторизации

Алгоритм осуществляет авторизацию пользователя. Пользователь вводит логин и пароль в соответствующие строки. Затем алгоритм сверяет вводимые данные с данными хранящиеся в текстовом документе. Если такие данные существуют, то осуществляется вход в личный кабинет, иначе алгоритм просит пользователя зарегистрироваться.

1. Алгоритм шифрования

Алгоритм предназначен для защиты логина и пароля пользователя. Вызываем функцию шифрования. Функция срабатывает, если поля логин и пароль не пустые. Сначала заводится массив, в который потом будет записваться зашифрованный логин и пароль. Затем заводится цикл, который идёт по строке, которую ввёл пользователь. Каждый символ строки сначала преобразовывается в число из таблицы Unicode, а потом шифруется с помощью дополнительной функции. После этого каждый символ строки записывается в массив в зашифрованном виде.

1. Алгоритм дешифрования

Алгоритм осуществляет расшифровку логина и пароля пользователя. Вызываем функцию дешифрования. Функция срабатывает, если текстовый документ не пустой. Сначала заводится массив, в который потом будет записваться дешифрованный логин и пароль. Затем заводится цикл, который идёт по строке, которую ввёл пользователь. Каждый элемент сначала дешифруется с помощью дополнительноой функции, а потом преобразовывается в символ из таблицы Unicode. После этого каждый символ записывается в массив в дешифрованном виде.

1. Алгоритм проверки возможности хода

Данный алгоритм проверяет возможность сходить шашкой в соответствии с правилами игры. Проверяется выбрана ли верная шашка(своего цвета). Если всё выбрано верно выполняется алгоритм подсвечивания возможных ходов

1. Алгоритм подсвечивания возможных ходов

Алгоритм, используя данные, полученные из алгоритма возможности хода, подсвечивает зеленым все возможные ходы.

1. Алгоритм хода игрока

Алгоритм начинает свою работу с вызова функции проверки возможности хода. Затем срабатывает следующая функция, она подсвечивает все эти возможные ходы. Далее пользователь выбирает куда сходить. После этого вызывается функция перемещения шашки. Шашка приобретает новые координаты.

1. Алгоритм хода компьютера

Данный алгоритм осуществляет ход противника(компьютера). Алгоритм начинает свою работу с функции проверки возможности хода. Затем вызывается функция которая проходиться по всем шашкам компьютера и для каждой проверяет лучшую комбинацию ходов. Эта функция возвращает координаты наилучшей строки и столбца, к которым компьютер может переместиться в соответствии со значениями кубиковю. После определения лучшей комбинации ходов шашка перемещается.

1. Алгоритм определения победителя

Данный алгоритм определяет победителя в сыгранной игре. Алгоритм проверяет, все ли шашки сняты с поля. Если один изигроков успел выбросить все свои шашки с поля раньше оппонента, то объявляется победитель.

Библиотеки, используемые во время разработки приложения:

“Pygame” — это «игровая библиотека», набор инструментов, помогающих программистам создавать игры. К ним относятся:

1. Графика и анимация
2. Звук (включая музыку)
3. Управление (мышь, клавиатура, геймпад и так далее)

“PyQt5” является одним из наиболее часто используемых модулей для создания GUI приложений в Python. GUI – это графический интерфейс пользователя. Данная библиотека позволяет корректно отобразить .ui фалы в программе, которые рисуются в графическом дизайнере QtDesigner.

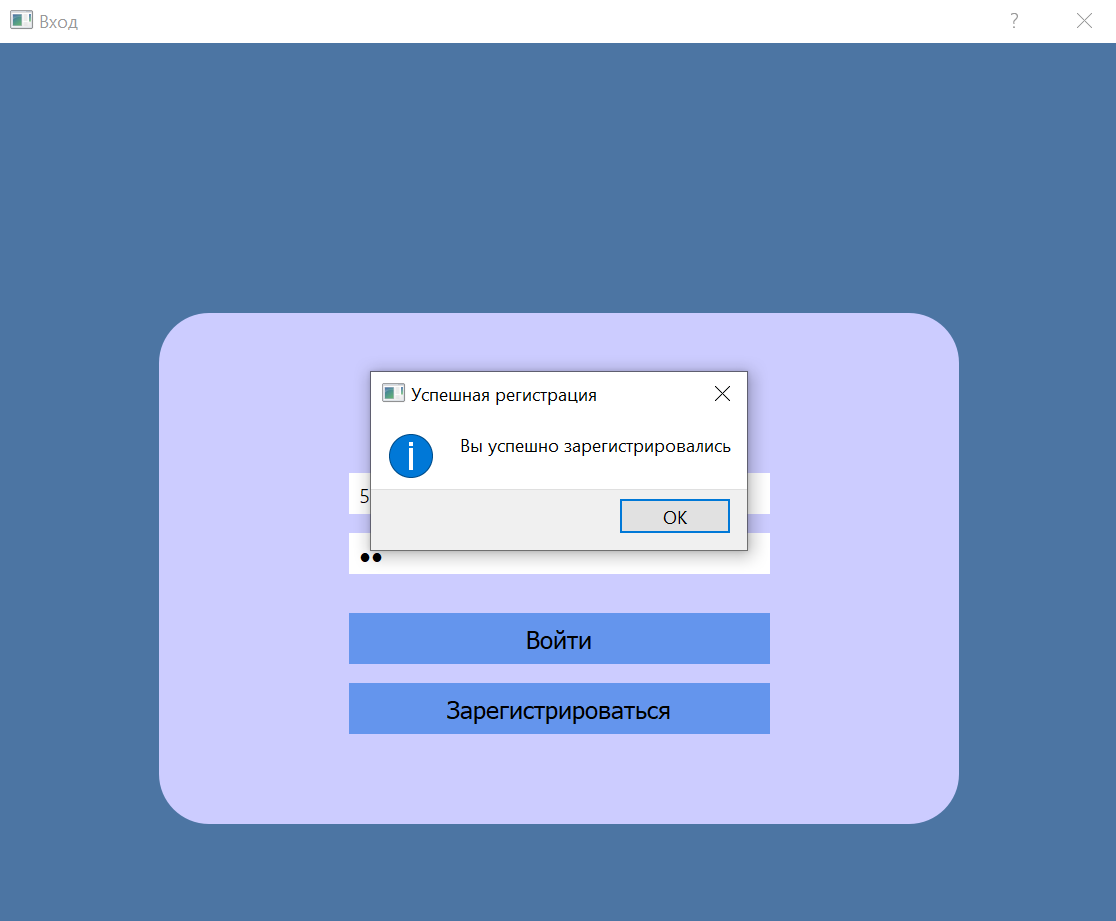
“os” – библиотека функций для работы с операционной системой. Методы, включенные в неё, позволяют определять тип операционной системы, получать доступ к переменным окружения, управлять директориями и файлами.

“sys” – это встроенный модуль Python, который предоставляет доступ к системным функциям и переменным.

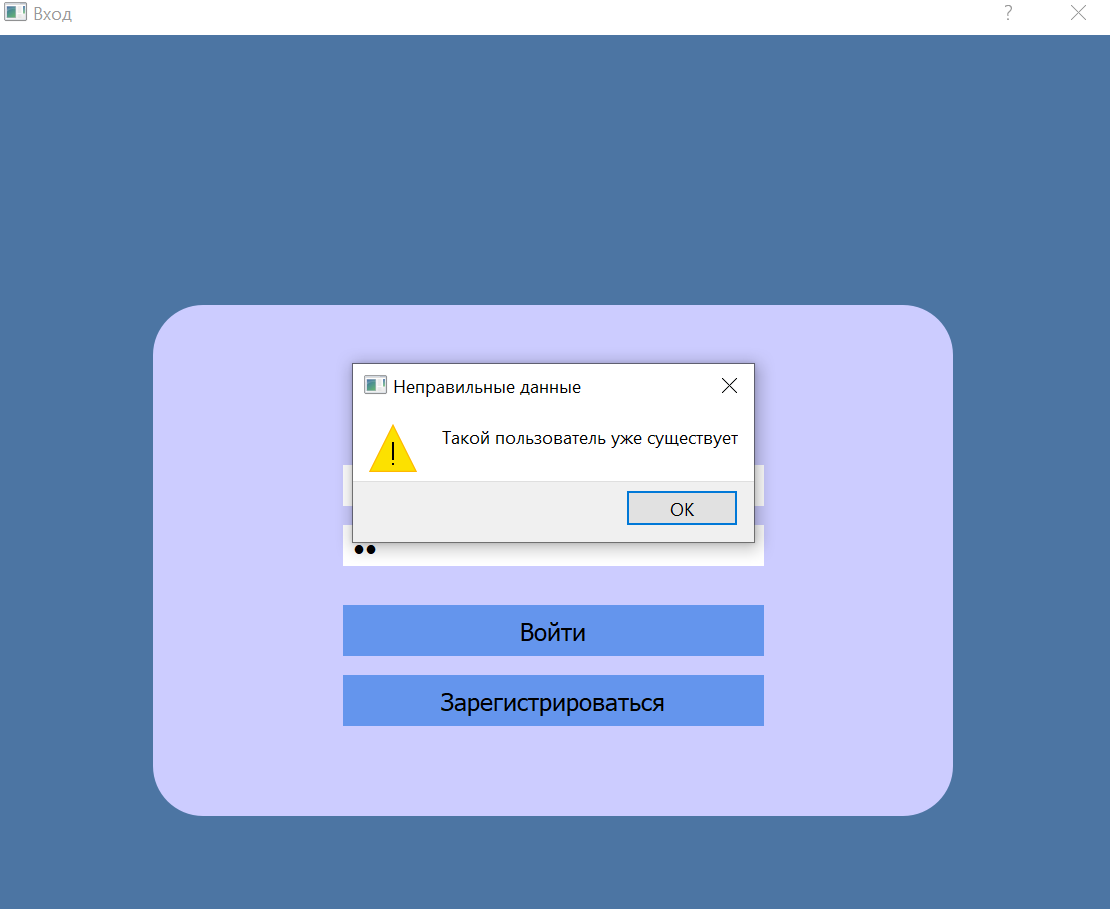
“timeit” – это простой интерфейс для быстрого измерения вре-мени выполнения небольших блоков кода.

“random” – этот модуль реализует генератор псевдослучайных чисел для различных распределений, включая целые и вещественные числа с плавающей запятой.

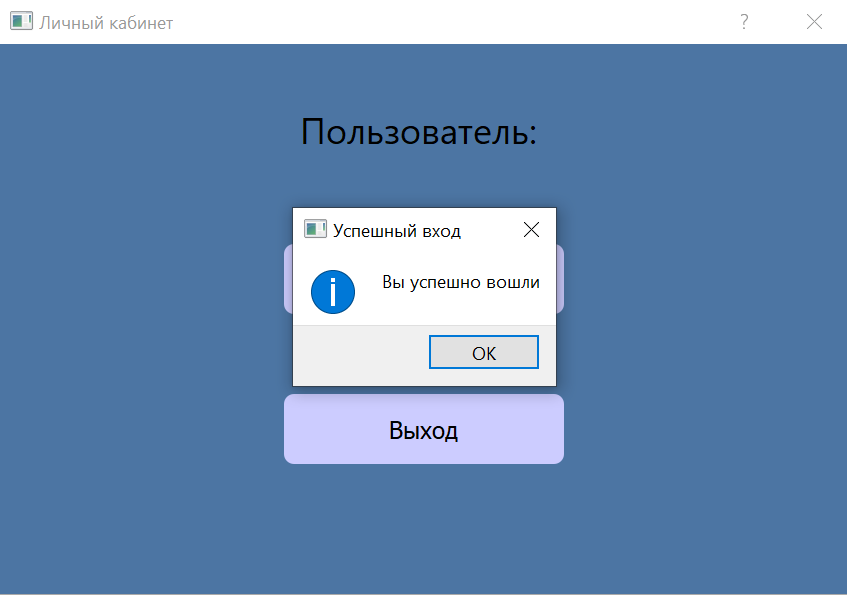
**4. Сообщения**

1. Успешная регистрация пользователя.

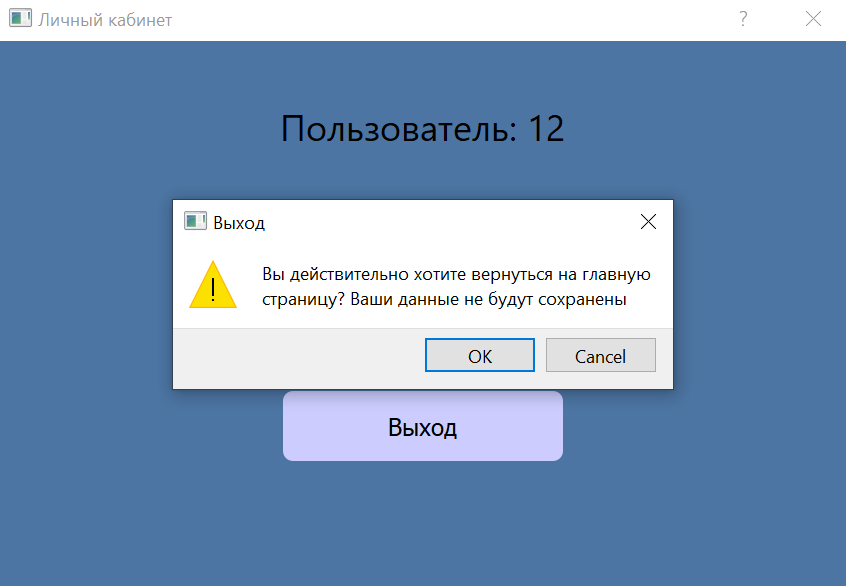
2. Пользователь с таким именем уже зарегистрирован



1. Успешная авторизация пользователя



1. Выход из личного кабинета



### **Текст программы**

1. Файл Ai\_player.py
2. **from** Move **import** Move
3. **from** Player **import** Player
4. **import** copy

7. **class** Ai(Player):
9. **def** \_\_init\_\_(self, type, positions, board, turn):
10. super().\_\_init\_\_(type, positions)
11. self.board **=** board
12. self.turn **=** turn
13. self.pieces\_columns **=** {None}
14. self.init\_pieces\_columns(self.type, self.pieces\_columns)
16. **def** init\_pieces\_columns(self, color, pieces\_columns):
17. **for** i **in** range(2):
18. **for** j **in** range(12):
19. stack **=** self.board.board[i][j]
20. **try**:
21. piece **=** stack.pick()
22. **if** piece.color !**=** color:
23. **continue**
24. pieces\_columns.add(piece)
25. **except** IndexError:
26. **continue**
27. **try**:
28. pieces\_columns.remove(None)
29. **except** KeyError:
30. print("Key Error")
32. #  лучший ход для AI
33. **def** play\_ai\_moves(self):
34. played\_moves **=** []
35. path **=** []
36. # если есть съеденные шашки
37. **if** len(self.board.eaten\_pieces.get("white")) > 0:
38. cubes **=** self.turn.cubes
39. current\_cube\_ind **=** 0
40. entrances **=** self.board.can\_piece\_entrances\_to\_column(cubes[0].get\_value(), cubes[1].get\_value(), "white")
41. **while** len(self.board.eaten\_pieces.get("white")) > 0 **and** len(entrances) > 0:
42. row **=** entrances[0].get("row")
43. col **=** entrances[0].get("column")
44. eat\_piece **=** entrances[0].get("eat\_piece")
46. **if** col !**=** **-**1:
47. self.turn.set\_destination(Move(row, col), eat\_piece, False)
48. self.board.return\_eaten\_piece\_to\_game("white", self.turn.destination)
49. self.turn.set\_cube\_played(current\_cube\_ind, True)
50. played\_moves.append([Move(row, col), eat\_piece, True])
52. current\_cube\_ind **=** current\_cube\_ind **+** 1
53. print(entrances.pop(0))
55. self.turn.remove\_destination()
57. # Если не все шашки дома и нет съеденных
58. **if** **not** (self.turn.has\_all\_cubes\_playes()) **and** len(self.board.eaten\_pieces.get("white")) **==** 0:
59. path **=** self.get\_all\_paths(played\_moves)
60. **if** path:
61. **for** cell **in** path:
62. move **=** cell[0]
63. eat\_piece **=** cell[1]
64. **if** move.col\_to **==** **-**1:
65. **continue**
66. **if** move.col\_to > 11:
67. self.board.out\_piece(move, "white")
68. **continue**
69. stack\_source **=** self.board.board[move.row\_from][move.col\_from]
70. stack\_dest **=** self.board.board[move.row\_to][move.col\_to]
71. **if** eat\_piece:
72. **try**:
73. self.board.eat\_piece(stack\_dest.pick())
74. **except** IndexError:
75. print("Не важно")
76. print(move)
78. # self.board.make\_move(move)
79. # self.board.change\_piece\_coordinates(move.row\_from, move.col\_from, move.row\_to, move.col\_to)
80. **return** path
82. # Это основная функция для  the AI\_player
83. # Эта функция проходиться по всем шашкам AI\_player и для каждой проверяет лучшую комбинацию ходов
84. # И лучшее место назначения в соответствии с наилучшей комбинацией
85. # функция возвращает координаты наилучшей строки и столбца, к которым AI может переместиться в соответствии со значениями кубиков
86. **def** get\_all\_paths(self, moves):
87. cubes **=** self.turn.cubes
88. double **=** cubes[0].get\_value() **==** cubes[1].get\_value()
89. paths **=** []
90. played\_moves **=** moves.copy()
91. append **=** 0
92. best\_score **=** **-**1000
93. best\_path **=** None
94. **for** piece **in** self.pieces\_columns.copy():
95. **for** i **in** range(len(cubes)):
96. **if** **not** (cubes[i].get\_played()): # если не разыгран ход
97. final\_move **=** Move(piece.row, piece.col)
98. destination **=** self.board.get\_destination(self.type, final\_move, cubes[i].get\_value())
99. final\_move.set\_row\_col\_to(destination.row\_from, destination.col\_from)
101. source\_legal, eat\_piece **=** self.board.is\_source\_legal(self.type, final\_move, self.all\_home)
103. **if** source\_legal: # если ходить можно
104. self.make\_virtual\_move(destination.row\_from, destination.col\_from, piece)
105. moves.clear()
107. **if** len(played\_moves) > 0:
108. moves **=** played\_moves.copy()
110. moves.append([final\_move, eat\_piece])
111. **if** **not**(cubes[abs((2**-**i)**-**1)].get\_played()):
112. self.get\_all\_paths\_for\_first\_move(len(cubes) **-** len(moves), cubes[abs((2**-**i)**-**1)].get\_value(), moves, paths, double)
113. **else**:
114. paths.append(moves.copy())
115. self.undo\_move(destination.row\_from, destination.col\_from, piece)
117. **return** self.get\_best\_path(paths)
119. **def** get\_best\_path(self, paths):
120. best\_score **=** **-**1000
121. best\_path **=** []
122. **for** path **in** paths:
123. path\_score **=** self.evaluate(path)
124. **if** path\_score > best\_score:
125. best\_score **=** path\_score
126. best\_path **=** path.copy()
127. **return** best\_path
129. # Эта функция возвращает все возможные ходы после первого хода
130. # Получает:
131. # cubes\_number: число кубиков, если дубль - 3, если нет - 1
132. # steps: количество ходов
133. # moves: на каждой итерации этот массив хранит все возможные ходы,
134. # и в конце мы добавляем этот массив в массив path
135. # path: все возможные ходы
136. # double: если кубики одинаковые, вернет True, если нет - False
137. **def** get\_all\_paths\_for\_first\_move(self, cubes\_number, steps, moves, all\_paths, double):
138. **if** cubes\_number **==** 0:
139. **return**
141. **else**:
142. **for** piece **in** self.pieces\_columns.copy():
143. final\_move **=** Move(piece.row, piece.col)
144. destination **=** self.board.get\_destination(self.type, final\_move, steps)
145. final\_move.set\_row\_col\_to(destination.row\_from, destination.col\_from)
147. source\_legal, eat\_piece **=** self.board.is\_source\_legal(self.type, final\_move, self.all\_home)
149. **if** source\_legal:
150. self.make\_virtual\_move(destination.row\_from, destination.col\_from, piece)
151. moves.append([copy.deepcopy(final\_move), eat\_piece])
152. self.get\_all\_paths\_for\_first\_move(cubes\_number **-** 1, steps, moves, all\_paths, double)
153. **if** len(moves) **==** 2 **and** **not** double **or** len(moves) **==** 4 **and** double:
154. all\_paths.append(copy.deepcopy(moves))
155. self.undo\_move(destination.row\_from, destination.col\_from, piece)
156. moves.pop(len(moves) **-** 1)
158. # Эта функция делает виртуальный ход
159. # Она получает строку и столбец назначения, а также шашку, которая сделает ход
160. # И установливает текущую строку и столбец шашки на строку и столбец, переданные функции
161. **def** make\_virtual\_move(self, dest\_row, dest\_col, piece):
162. stack **=** self.board.board[piece.row][piece.col]
163. **try**:
164. piece **=** stack.pop()  # убираем шашку с доски
165. temp **=** copy.deepcopy(piece)  # копия шашки
167. **if** stack.stack\_len() **==** 0:  # если стек пустой
168. self.pieces\_columns.discard(piece)  # удалить этуу шашку из массива pieces\_columns
170. temp.set\_row(dest\_row)
171. temp.set\_col(dest\_col)
173. self.board.board[dest\_row][dest\_col].push(temp)
174. self.pieces\_columns.add(temp)
175. **except** IndexError:
176. **if** dest\_col > 11:
177. self.board.out\_pieces.get("white").append(piece)
178. **else**:
179. print("ПУСТАЯ ЯЧЕЙКА")
181. # Эта функция отменяет ход
182. # Функция получает исходную строку, исходный столбец и шашку
183. # И возвращает шашку в исходную строку и исходный столбец
184. # + функция возвращает шашку обратно в исходную строку и исходный столбец на доске
185. **def** undo\_move(self, dest\_row, dest\_col, piece):
186. piece\_row **=** piece.row
187. piece\_col **=** piece.col
188. **try**:
189. stack **=** self.board.board[dest\_row][dest\_col]
190. **try**:
191. temp **=** copy.deepcopy(stack.pick())
192. **if** stack.stack\_len() **==** 0:
193. self.pieces\_columns.discard(temp)
194. **except** IndexError:
195. print("ПУСТАЯ ЯЧЕЙКА")
196. # отмена хода
197. move **=** Move(dest\_row, dest\_col)
198. move.set\_row\_col\_to(piece.row, piece.col)
199. self.board.make\_move(move)
200. self.pieces\_columns.add(piece)
201. **except** IndexError:
202. **if** dest\_col > 11:
203. self.board.board[piece.row][piece.col].push(piece)
204. self.pieces\_columns.add(piece)
205. **else**:
206. print("ПУСТАЯ ЯЧЕЙКА")
208. # Эта функция оценивает ходы ]
209. **def** evaluate(self, path):
210. score **=** 0
212. **for** tmp **in** path:
213. move **=** tmp[0]
214. s\_row, s\_col **=** move.row\_from, move.col\_from
215. d\_row, d\_col **=** move.row\_to, move.col\_to
217. stack\_source **=** self.board.board[s\_row][s\_col]
218. **try**:
219. stack\_dest **=** self.board.board[d\_row][d\_col]
220. **except** IndexError:
221. **if** d\_col > 11:
222. **if** stack\_source.stack\_len() > 2:
223. score **=** score **+** 3000
225. print("ВЫБРОС")
227. eat\_piece **=** tmp[1]
228. **if** eat\_piece:
229. score **=** score **+** 150
231. **try**:
232. **if** stack\_source.stack\_len() **==** 2:
233. score **=** score **-** 100
234. **else**:
235. score **=** score **+** 50
236. **except** IndexError:
237. score **=** score **+** 50
239. **if** d\_col > 11:
240. score **=** score **+** 200
241. **continue**
243. **if** stack\_dest.stack\_len() >**=** 1:
244. score **=** score **+** 50
245. **else**:
246. score **=** score **-** 100
248. **return** score

.

1. Файл Ai\_player.py
2. **import** copy
3. **from** Stack **import** Stack
4. **from** Move **import** Move
6. PIECE\_WIDTH, PIECE\_HEIGHT **=** 75, 69
7. SPACE\_BETWEEN\_PIECES **=** 46
8. COLUMN\_OVERFLOW\_MULTIPLY **=** 20
9. ROW1\_BOTTOM **=** 481
10. EATEN\_AREA\_WIDTH **=** 50
11. PADDING\_BOARD\_BORDERS **=** 25

14. # Класс доска
15. **class** Board:
16. **def** \_\_init\_\_(self, red\_player, white\_player):
17. self.red\_player **=** red\_player
18. self.white\_player **=** white\_player
19. self.board **=** [[], []]
20. self.init\_board()
21. self.set\_columns(self.red\_player.pieces)
22. self.set\_columns(self.white\_player.pieces)
23. self.organize\_pieces **=** [0, 0, 5, 10, 15, 15, 15, 15, 15, 15, 15, 15] # Эта часть параметров предназначена для уменьшения пикселей в столбцах слева направо
24. self.eaten\_pieces **=** {"red": [], "white": []}
25. self.out\_pieces **=** {"red": [], "white": []}
27. # инициализируеv доску путем вставки в каждую ячейку  экземпляр Stack
28. **def** init\_board(self):
29. **for** i **in** range(2):
30. **for** j **in** range(12):
31. self.board[i].append(Stack())
33. # функция устанавливает вовсе столбцы доски стеки
34. **def** set\_columns(self, pieces\_places):
35. **for** i **in** range(len(pieces\_places)):
36. self.insert\_piece(pieces\_places[i])
38. # устанавливаем новую шашку на место где она должна быть
39. **def** insert\_piece(self, piece):
40. row **=** piece.row
41. col **=** piece.col
42. self.board[row][col].push(piece)
44. **def** change\_piece\_coordinates(self, row\_s, col\_s, row\_d, col\_d):
45. piece **=** self.board[row\_s][col\_s].pop()
46. piece.set\_row(row\_d)
47. piece.set\_col(col\_d)
49. self.board[row\_d][col\_d].push(piece)
51. #  определяем цвет игрока и возвращает его данные
52. **def** get\_player\_details(self, color):
53. **if** color **==** "red":
54. **return** self.red\_player
55. **return** self.white\_player
57. # Эта функция проверяет, является ли выбранная позиция верной
58. # 2 значения возвращает :
59. # 1) Если позиция верная , возвращает True
60. # 2) Если игрок сделает ход из этой позиции и съест фигуру противника вернет  true
61. **def** is\_source\_legal(self, color, move, all\_pieces\_in\_home):
62. row\_source **=** move.get\_row\_from()
63. col\_source **=** move.get\_col\_from()
65. row\_dest **=** move.get\_row\_to()
66. col\_dest **=** move.get\_col\_to()
67. **try**:
68. stack\_source **=** self.board[row\_source][col\_source]
69. **if** col\_dest > 11:  # Если столбец назначения отсутствует на доске
70. **if** **not** all\_pieces\_in\_home:
71. **return** False, False
72. **elif** stack\_source.pick().get\_color() **==** color:
73. **return** True, False
74. **else**:
75. **return** False, False
77. **if** stack\_source.pick().get\_color() !**=** color:  #Если цвет шашки не совпадает с цвеом  текущего хода
78. **return** False, False
79. **except** IndexError as e:
80. **return** False, False
82. **try**:
83. dest **=** self.board[row\_dest][col\_dest]
84. **if** dest.pick().get\_color() !**=** color:  # Если цвет шашки, который находится в ячейке, не равен текущему цвету
85. **if** dest.stack\_len() **==** 1:  # Если в ячейке 1 шашка, значит ее можно сьесть
86. **return** True, True
87. **return** False, False
88. **except** IndexError as e:
89. **return** True, False
91. **return** True, False
93. # проверяем правильно ли игрок выбрал ячейку и досткпна ли она
94. **def** is\_destination\_legal(self, clicked\_coordinates, source\_dest):
95. row\_click **=** clicked\_coordinates.get\_row\_from()
96. col\_click **=** clicked\_coordinates.get\_col\_from()
98. row\_dest **=** source\_dest.get\_row\_to()
99. col\_dest **=** source\_dest.get\_col\_to()
101. **if** row\_click **==** row\_dest **and** col\_click **==** col\_dest:
102. **return** True
104. **return** False
106. #  функция получает цвет шашки, начальную позицию и количество ходов, которые она должна выполнить
107. **def** get\_destination(self, color, start, steps):
108. row\_start **=** start.get\_row\_from()
109. col\_start **=** start.get\_col\_from()
110. direction **=** 1
112. **if** color **==** "red":
113. direction **=** **-**1
115. **if** row\_start **==** 0 **and** color **==** "white" **or** row\_start **==** 1 **and** color **==** "red":
116. steps **=** steps **\*** (**-**1)
118. diff **=** col\_start **+** steps
120. **if** diff < 0:
121. row\_start **+=** direction
122. diff **=** abs(diff) **-** 1
124. **return** Move(row\_start, diff)
126. # Эта функция выполняет перемещение от начальной позиции к назначенной
127. **def** make\_move(self, move):
128. row\_source **=** move.get\_row\_from()
129. col\_source **=** move.get\_col\_from()
131. row\_dest **=** move.get\_row\_to()
132. col\_dest **=** move.get\_col\_to()
134. stack\_dest\_len **=** self.board[row\_dest][col\_dest].stack\_len()
136. move\_piece **=** copy.deepcopy(self.board[row\_source][col\_source].pop())
137. move\_piece.set\_row(row\_dest)
138. move\_piece.set\_col(col\_dest)
140. self.change\_row\_col\_visual(move\_piece, stack\_dest\_len, row\_dest, col\_dest)
141. self.insert\_piece(move\_piece)
143. **def** change\_row\_col\_visual(self, move\_piece, stack\_dest\_len, row\_dest, col\_dest):
144. row\_mul **=** SPACE\_BETWEEN\_PIECES
146. **if** 4 **-** stack\_dest\_len < 0:
147. stack\_dest\_len **=** abs(4 **-** stack\_dest\_len)
148. row\_mul **=** COLUMN\_OVERFLOW\_MULTIPLY
150. **if** row\_dest **==** 0:
151. row\_dest **=** (row\_dest **+** stack\_dest\_len **\*** row\_mul) **+** 10
152. **else**:
153. row\_dest **=** (ROW1\_BOTTOM **-** stack\_dest\_len **\*** row\_mul)
155. **if** col\_dest < 6:
156. col\_dest **=** col\_dest **\*** PIECE\_WIDTH **+** PADDING\_BOARD\_BORDERS **-** self.organize\_pieces[col\_dest]
157. **else**:
158. col\_dest **=** col\_dest **\*** PIECE\_WIDTH **+** PADDING\_BOARD\_BORDERS **-** self.organize\_pieces[col\_dest] **+** EATEN\_AREA\_WIDTH
160. move\_piece.set\_row\_visual(col\_dest)
161. move\_piece.set\_col\_visual(row\_dest)
163. # Эта функция съедает тот шашку, которую ей передали
164. # Для этого функция вызывает функцию, которая хранится в классе Player
165. # она добавляет шашку в массив съеденных шашек
166. # и в то же время удаляет ее из массива доступных шашек
167. **def** eat\_piece(self, piece):
168. row **=** piece.get\_row()
169. col **=** piece.get\_col()
170. eaten\_pieces **=** self.eaten\_pieces.get("red")
171. player **=** self.red\_player
173. temp **=** self.board[row][col].pop()
174. **if** piece.get\_color() **==** "white":
175. eaten\_pieces **=** self.eaten\_pieces.get("white")
176. player **=** self.white\_player
178. temp.set\_col\_visual(100 **+** (len(eaten\_pieces) **\*** PIECE\_HEIGHT))
179. temp.set\_row\_visual(453)
180. temp.set\_row(**-**1)
181. temp.set\_col(**-**1)
182. eaten\_pieces.append(temp)
184. # Эта функция проверяет, может ли съеденная шашка попасть в один из  столбцов, которые сгенерировали кубики
185. **def** can\_piece\_entrances\_to\_column(self, cube1\_value, cube2\_value, eaten\_piece\_color):
186. available\_entrances **=** []
187. enemy\_home\_row **=** 0  # RED --> 0, WHITE --> 1
189. **if** eaten\_piece\_color **==** "red":
190. enemy\_home\_row **=** 1
192. available\_entrances.append(self.piece\_entrance\_options(eaten\_piece\_color,
193. {"row": enemy\_home\_row, "column": (11 **-** cube1\_value) **+** 1}))
194. available\_entrances.append(self.piece\_entrance\_options(eaten\_piece\_color,
195. {"row": enemy\_home\_row,"column": (11 **-** cube2\_value) **+** 1}))
197. **return** available\_entrances
199. # Эта функция возвращает параметры входа --> {
200. # column: столбец входа (-1 если вход недоступен)
201. # eat\_piece: если есть шашка, которая была съедена, когда эта шашка попала в этот столбец
202. # }
203. **def** piece\_entrance\_options(self, eaten\_piece\_color, coordinates):
204. entrance\_details **=** {"row": **-**1, "column": **-**1, "eat\_piece": False}
206. row **=** coordinates.get("row")
207. col **=** coordinates.get("column")
209. column **=** self.board[row][col]
210. column\_len **=** column.stack\_len()
212. **if** column\_len **==** 0:  # Если в этом столбце нет шашки
213. entrance\_details["column"] **=** col
215. **elif** column\_len > 1:  # Если в этом столбце больше одной шашки
216. **if** column.pick().color **==** eaten\_piece\_color:
217. entrance\_details["column"] **=** col
219. **elif** column\_len **==** 1:  # Если в этом столбце есть только одна шашка
220. entrance\_details["column"] **=** col
221. **if** column.pick().color !**=** eaten\_piece\_color:  # Если цвет шашки не совпадает с цветом съеденной шашки
222. entrance\_details["eat\_piece"] **=** True
224. entrance\_details["row"] **=** row
226. **return** entrance\_details

229. **def** can\_enter(self, entrance\_options):
230. **for** entrance **in** entrance\_options:
231. row **=** entrance.get("row")
232. col **=** entrance.get
234. # возвращает съеденную шашку обратно в игру
235. **def** return\_eaten\_piece\_to\_game(self, color, entrance):
236. eaten\_pieces **=** self.eaten\_pieces.get("red")
237. player **=** self.red\_player
238. organize **=** 46
239. row, col **=** entrance.get("destination").get\_row\_from(), entrance.get("destination").get\_col\_from()
240. can\_eat\_piece **=** entrance.get("eat\_piece")
241. temp\_move **=** Move(row, col)
242. temp\_move.set\_row\_col\_to(row, col)
244. **if** color **==** "white":
245. eaten\_pieces **=** self.eaten\_pieces.get("white")
246. player **=** self.white\_player
247. organize **=** **-**46
249. piece **=** eaten\_pieces.pop(0)
250. piece.set\_row(row)
251. piece.set\_col(col)
253. **if** can\_eat\_piece:
254. self.eat\_piece(self.board[row][col].pick())
256. self.insert\_piece(piece)
257. player.add\_piece(piece)
259. self.make\_move(temp\_move)
260. piece **=** self.board[row][col].pick()
261. piece.set\_col\_visual(piece.col\_visual **+** organize)
263. # Эта функция вызывает функцию "are\_all\_pieces\_in\_home" в классе player,
264. # в соответствии с цветом игрока, который получила функция.
265. # Эта функция проверяет, все ли шашки игрока находятся в его доме
266. # Эта функция возвращает результат "are\_all\_pieces\_in\_home", который в классе Player
267. **def** are\_all\_pieces\_in\_home\_for\_player(self, color):
268. player **=** self.white\_player
269. row\_home **=** 1
270. **if** color **==** "red":
271. player **=** self.red\_player
272. row\_home **=** 0
274. **if** player.all\_home:
275. **return** True
277. counter **=** self.number\_of\_pieces\_in\_home(row\_home, color)
279. **if** counter **+** len(self.out\_pieces.get(color)) **==** 15:
280. player.set\_all\_home(True)
281. **return** True
283. player.set\_all\_home(False)
284. **return** False
286. # Эта функция возвращает количество шашек в доме
287. **def** number\_of\_pieces\_in\_home(self, row\_home, color):
288. counter **=** 0
289. **for** i **in** range(6, 12):
290. stack **=** self.board[row\_home][i]
292. **try**:
293. **if** stack.pick().color **==** color:
294. counter **+=** stack.stack\_len()
295. **except** IndexError:
296. **continue**
298. **return** counter
300. # Эта функция выполняется для всех столбцов, в которых есть шашки с цветом, полученным функцией
301. # и для каждого столбца функция проверяет, моожноли переместить шашку
302. # Если шашка может переместиться на один из кубиков, она выходит из циклов и возвращает значение true
303. # Если все шашки заблокированы, функция вернет значение false
304. **def** check\_if\_blocked(self, cubes, color, all\_pieces\_in\_home):
305. blocked **=** True
307. **for** i **in** range(2):
308. **for** j **in** range(12):
309. stack **=** self.board[i][j]
310. **try**:
311. **if** stack.pick().color !**=** color:
312. **continue**
313. final\_move **=** Move(i, j)
314. **for** cube **in** cubes:
315. **if** **not** self.check\_if\_cube\_blocked(color, final\_move, cube, all\_pieces\_in\_home):
316. blocked **=** False
318. **except** IndexError:
319. **continue**
321. **if** **not** blocked:
322. **break**
324. **else**:
325. **continue**
327. **break**
329. **return** blocked
331. # Эта функция проверяет, может ли игрок сходить по значению кубика
332. **def** check\_if\_cube\_blocked(self, color, final\_move, cube, all\_pieces\_in\_home):
333. temp\_destination **=** self.get\_destination(color, final\_move, cube.get\_value())
334. final\_move.set\_row\_col\_to(temp\_destination.get\_row\_from(), temp\_destination.get\_col\_from())
335. is\_legal, eat\_piece **=** self.is\_source\_legal(color, final\_move, all\_pieces\_in\_home)
336. **if** **not** is\_legal:
337. **return** True
339. **return** False
341. # Эту функцию нужно вызывать, когда все шашки игрока находятся у него дома, и теперь ему нужно их выбросить
342. **def** out\_piece(self, piece\_coordinates, color):
343. row **=** piece\_coordinates.get\_row\_from()
344. col **=** piece\_coordinates.get\_col\_from()
345. piece **=** self.board[row][col].pop()
346. self.out\_pieces.get(color).append(piece)
348. # Эта функция проверяет есть ли победитель
349. **def** is\_there\_winner(self):
350. counter\_pieces\_white **=** self.count\_pieces("white")
351. counter\_pieces\_red **=** self.count\_pieces("red")
352. winner **=** ""
354. **if** counter\_pieces\_white **==** 0:
355. winner **=** "white"
357. **if** counter\_pieces\_red **==** 0:
358. winner **=** "red"
360. **return** winner !**=** "", winner
362. # Эта функция подсчитывает количество шашек для игрока с цветом "color"
363. **def** count\_pieces(self, color):
364. counter **=** 0
365. **for** i **in** range(2):
366. **for** j **in** range(12):
367. **try**:
368. stack **=** self.board[i][j]
369. **if** stack.pick().get\_color() **==** color:
370. counter **=** counter **+** stack.stack\_len()
371. **except** IndexError:
372. **continue**
374. **return** counter

377. **def** set\_player(self, color, new\_player):
378. player **=** self.red\_player
380. **if** type **==** "white":
381. player **=** self.white\_player
383. player **=** new\_player
385. #
386. **def** \_\_str\_\_(self):
387. stacks **=** []
388. **for** i **in** range(2):
389. **for** j **in** range(12):
390. stacks.append(self.board[i][j].\_\_str\_\_())
391. **return** ", ".join(stacks)
392. Файл Cube.py
393. # Класс для одного кубика
394. **class** Cube():
396. **def** \_\_init\_\_(self, value):
397. self.value **=** value  # Значения кубика (1-6)
398. self.played **=** False  # True если кубик разыгран False если нет
400. # Функция возвращает текущее значение кубика
401. **def** get\_value(self):
402. **return** self.value
404. # Функция устанавливает новое значение кубика
405. **def** set\_value(self, value):
406. self.value **=** value
408. # Функция возвращает текущее значение параметра played
409. **def** get\_played(self):
410. **return** self.played
412. # Функция устанавливает новое значение параметра played
413. # Если игрок не сходил -  True
414. # Если сходил - False
415. **def** set\_played(self, played):
416. self.played **=** played
418. **def** \_\_str\_\_(self): # для отладки
419. **return** "Значение: {0}".format(self.value)
420. Файл Game.py
421. **import** pygame, os, sys
422. **from** Player **import** Player
423. **from** Board **import** Board
424. **from** Move **import** Move
425. **from** Turn **import** Turn
426. **from** Ai\_player **import** Ai
427. **from** timeit **import** default\_timer as timer
429. pygame.font.init()
431. FPS **=** 60
433. # Шрифты
434. FONT **=** pygame.font.SysFont('timesnewroman',  30)
436. # Размеры главного окна
437. WIDTH, HEIGHT **=** 1044, 581
439. # Цвета для интерфейса
440. WHITE **=** (255, 255, 255)
441. BLACK **=** (0, 0, 0)
442. RED **=** (255, 0, 0)
443. GREEN **=** (0, 255, 0)
444. YELLOW **=** (255, 242, 0)
446. # Позиции шашек для начала игры
447. WHITE\_PLACES **=** [(25, 20), (25, 76), (25, 132), (25, 188),
448. (25, 56), (315, 486), (315, 430), (315, 374), (515, 486), (515, 430), (515, 374), (515, 318),
449. (515, 450), (878, 20), (878, 76)]
451. RED\_PLACES **=** [(25, 486), (25, 430), (25, 374), (25, 318), (25, 450), (310, 20), (310, 76), (310, 132),
452. (518, 20), (518, 76), (518, 132), (518, 188), (518, 56), (875, 486), (875, 430)]

455. # Поле для сьеденных шашек (БАР)
456. EATEN\_AREA\_X, EATEN\_AREA\_Y **=** 473, 100
457. EATEN\_AREA\_WIDTH, EATEN\_AREA\_HEIGHT **=** 40, 351

460. # Загружаем изображения
461. BOARD **=** pygame.image.load(os.path.join("images", "board.png"))
462. RED\_PLAYER **=** pygame.image.load((os.path.join("images", "red.png")))
463. WHITE\_PLAYER **=** pygame.image.load(os.path.join("images", "white.png"))
465. NO\_MOVES\_IMG **=** pygame.image.load((os.path.join("images", "nomoves.png")))
466. NO\_MOVES\_IMG\_W, NO\_MOVES\_IMG\_H **=** 740, 124

469. # Изображения для победителя
470. WHITE\_WINNER\_IMG **=** pygame.image.load(os.path.join("images", "whitewinner.png"))
471. RED\_WINNER\_IMG **=** pygame.image.load(os.path.join("images", "redwinner.png"))
472. WINNER\_IMG\_WIDTH, WINNER\_IMG\_HEIGHT **=** 607, 119

475. # Координаты для отображения кубиков
476. CUBE1\_X, CUBE1\_Y **=** 983, 194
477. CUBE2\_X, CUBE2\_Y **=** 983, 250
478. CUBE\_W, CUBE\_H **=** 46, 40

481. # Значения для отрисовки полигонов
482. ORGANIZE **=** [0, 0, 0, 5, 5, 7, **-**5, **-**5, **-**2, **-**2, **-**2, 0]
484. # Основное окно
485. WIN **=** pygame.display.set\_mode((WIDTH, HEIGHT))

488. pygame.display.set\_caption("НАРДЫ")
490. # Функция отрисовки основных компонентов
491. **def** draw(board, turn, winner):
492. WIN.blit(BOARD, (0, 0)) # изображение доски
494. **if** turn.red\_turn: #  отображение кто сейчас ходит
495. WIN.blit(FONT.render("Red", False, RED), (WIDTH **-** 60, 10))
496. **else**:
497. WIN.blit(FONT.render("White", False, WHITE), (WIDTH **-** 75, 10))
499. cube1 **=** turn.cubes[0]
500. cube2 **=** turn.cubes[1]
501. current\_cube **=** turn.chosen\_cube
503. **if** turn.source:
504. source\_row **=** turn.source.get\_row\_from()
505. source\_col **=** turn.source.get\_col\_from()
506. stack **=** board.board[source\_row][source\_col]
508. # подсветка хода игрока
509. extra\_x **=** 25
510. y **=** 25
511. height **=** 220
512. **if** source\_col > 5:
513. extra\_x **=** extra\_x **+** 30
514. **if** source\_row **==** 1:
515. y **=** 550
516. height **=** y **-** height
518. final\_x **=** source\_col **\*** 75 **+** extra\_x **-** ORGANIZE[source\_col]
519. # pygame.draw.rect(WIN, YELLOW, (source\_col \* 75 + extra\_x, y, 75, 10))
520. pygame.draw.polygon(WIN, GREEN, [(final\_x, y), (final\_x **+** 40 **-** ORGANIZE[source\_col], height), (final\_x **+** 75, y)])
522. draw\_pieces(board.board, board.red\_player.get\_pieces())
523. draw\_pieces(board.board, board.white\_player.get\_pieces())
525. # отрисока значений  и подсветка кубиков
526. **if** cube1.get\_played():
527. pygame.draw.rect(WIN, RED, (CUBE1\_X, CUBE1\_Y, CUBE\_W, CUBE\_H))
528. **if** turn.cubes[1].get\_played():
529. pygame.draw.rect(WIN, RED, (CUBE2\_X, CUBE2\_Y, CUBE\_W, CUBE\_H))
531. **if** current\_cube **==** 0:
532. pygame.draw.rect(WIN, GREEN, (CUBE1\_X, CUBE1\_Y, CUBE\_W, CUBE\_H), 5)
533. **else**:
534. pygame.draw.rect(WIN, GREEN, (CUBE2\_X, CUBE2\_Y, CUBE\_W, CUBE\_H), 5)
536. WIN.blit(FONT.render(str(cube1.get\_value()), False, BLACK), (998, 197))
537. WIN.blit(FONT.render(str(cube2.get\_value()), False, BLACK), (998, 254))
539. # подсветка хода (снизу)
540. **if** turn.destination:
541. dest **=** turn.destination.get("destination")
542. extra\_x **=** 25
543. y **=** 20
544. **if** dest.get\_col\_from() > 5:
545. extra\_x **=** extra\_x **+** 30
546. **if** dest.get\_row\_from() **==** 1:
547. y **=** 530
549. pygame.draw.rect(WIN, GREEN, (dest.get\_col\_from() **\*** 75 **+** extra\_x, y, 75, 10))
551. **if** len(board.eaten\_pieces.get("red")) > 0:
552. draw\_eaten\_pieces(board.eaten\_pieces.get("red"), "red")
553. **elif** len(board.eaten\_pieces.get("white")) > 0:
554. draw\_eaten\_pieces(board.eaten\_pieces.get("white"), "white")
555. pygame.draw.rect(WIN, WHITE, (EATEN\_AREA\_X, EATEN\_AREA\_Y, EATEN\_AREA\_WIDTH, EATEN\_AREA\_HEIGHT), 1)
557. # победитель
558. **if** winner:
559. **if** winner[1] **==** "red":
560. WIN.blit(RED\_WINNER\_IMG, (WIDTH **/** 2 **-** WINNER\_IMG\_WIDTH **/** 2, HEIGHT **/** 2 **-** 50))
561. **else**:
562. WIN.blit(WHITE\_WINNER\_IMG, (WIDTH **/** 2 **-** WINNER\_IMG\_WIDTH **/** 2, HEIGHT **/** 2 **-** 50))
563. pygame.display.update()
565. # функция для отрисовки шашек
566. **def** draw\_pieces(board, pieces):
567. **for** i **in** range(2):
568. **for** j **in** range(len(pieces)):
569. **try**:
570. stack\_pieces **=** board[i][j].get\_all\_stack\_elements()
571. color **=** stack\_pieces[0].get\_color()
572. **for** piece **in** stack\_pieces:
573. row\_visual **=** piece.get\_row\_visual()
574. col\_visual **=** piece.get\_col\_visual()
575. **if** color **==** "red":
576. WIN.blit(RED\_PLAYER, (row\_visual, col\_visual))
577. **else**:
578. WIN.blit(WHITE\_PLAYER, (row\_visual, col\_visual))
580. **except** Exception as error:
581. **continue**
583. # отрисовка съеденных шашек
584. **def** draw\_eaten\_pieces(eaten\_pieces, color):
585. piece\_img **=** RED\_PLAYER
587. **if** color **==** "white":
588. piece\_img **=** WHITE\_PLAYER
590. **for** eaten **in** eaten\_pieces:
591. row\_visual **=** eaten.get\_row\_visual()
592. col\_visual **=** eaten.get\_col\_visual()
593. WIN.blit(piece\_img, (row\_visual, col\_visual))
595. # функция для отрисовки хода AI
596. **def** draw\_ai\_path(ai\_path, board, turn):
597. **for** move **in** ai\_path:
598. row\_dest **=** move[0].row\_to
599. col\_dest **=** move[0].col\_to
600. eat\_piece **=** move[1]
602. **if** col\_dest **==** **-**1:
603. **continue**
604. **try**:
605. stack\_dest **=** board.board[row\_dest][col\_dest]
606. turn.set\_source(move[0])
607. turn.set\_destination(Move(row\_dest, col\_dest), eat\_piece, False)
608. draw(board, turn, False)
609. delay(1.5)
610. board.make\_move(move[0])
611. turn.remove\_destination()
612. turn.remove\_source()
613. draw(board, turn, False)
614. **except** IndexError:
615. **continue**

618. # Эта функция возвращает номер кубика на который кликнули, если нет -1
620. **def** clicked\_on\_cube(click\_coordinates):
621. x **=** click\_coordinates[0]
622. y **=** click\_coordinates[1]
624. **if** CUBE1\_X < x < CUBE1\_X **+** CUBE\_W **and** CUBE1\_Y < y < CUBE1\_Y **+** CUBE\_H:
625. **return** 0
626. **elif** CUBE2\_X < x < CUBE2\_X **+** CUBE\_W **and** CUBE2\_Y < y < CUBE2\_Y **+** CUBE\_H:
627. **return** 1
629. **return** **-**1

632. # Функция для задержки хода
633. **def** delay(sec):
634. start **=** timer()
635. **while** timer() **-** start < sec:
636. **continue**
637. end **=** timer()
638. print(end **-** start)
640. # ОСНОВНОЙ МОДУЛЬ
641. **def** main():
642. red\_player **=** Player("red", RED\_PLACES)  # КРАСНЫЙ игрок
643. white\_player **=** Player("white", WHITE\_PLACES)  # БЕЛЫЙ игрок
644. board **=** Board(red\_player, white\_player)  # ДОСКА
646. run **=** True
647. waiting\_for\_enter **=** False  # эта переменная ждет пока игрок выберет ход, когда его шашку сьели
649. game\_mode **=** "2" #input("Choose a game mode:\n1 - Human VS Human\n2 - Human VS Computer")
651. turn **=** Turn(game\_mode **==** "2")  # обеькт ХОД
652. ai\_player **=** Ai("white", WHITE\_PLACES, board, turn)
653. **if** turn.ai\_game:
654. board.set\_player("white", ai\_player)
656. **while** run:
657. has\_winner, winner **=** board.is\_there\_winner()  #  есть ли победитель
658. all\_in\_home **=** board.are\_all\_pieces\_in\_home\_for\_player(turn.color) # все ли шашки дома
660. **if** turn.ai\_game:
661. **if** turn.ai\_turn:
662. print("Ход AI ...")
663. **if** all\_in\_home:
664. ai\_player.set\_all\_home(True)
665. print(turn.cubes[0], turn.cubes[1])
666. ai\_path **=** ai\_player.play\_ai\_moves()
667. draw\_ai\_path(ai\_path, board, turn)
668. turn.change\_turn()
669. print("AI ходит...")
670. **continue**
672. **if** has\_winner:  # Если есть победитель
674. draw(board, turn, [True, winner])
675. start\_ticks **=** pygame.time.get\_ticks()
676. delay(5)

679. run **=** False
680. **return**
681. pygame.display.quit()


685. **while** run:
686. **for** event **in** pygame.event.get():
687. **if** event.type **==** pygame.QUIT:
688. run **=** False
689. seconds **=** (pygame.time.get\_ticks() **-** start\_ticks) **/** 1000
690. **if** seconds > 5:
691. run **=** False
693. #   есть ли  доступные ходы
694. **elif** board.check\_if\_blocked(turn.cubes, turn.color, all\_in\_home):
695. start\_ticking **=** pygame.time.get\_ticks()
696. print("НЕТ ДОСТУПНЫХ ХОДОВ!")
697. WIN.blit(NO\_MOVES\_IMG, (WIDTH **/** 2 **-** NO\_MOVES\_IMG\_W **/** 2, HEIGHT **/** 2 **-** 50))
698. delay(2)
699. turn.change\_turn() # ходит другой игрок
700. **continue**
701. **else**:  # Если пока нет победителя
702. **if** len(board.eaten\_pieces.get(turn.color)) > 0:  # Если враг съел шашки текущего игрока
703. entrance\_details **=** board.can\_piece\_entrances\_to\_column(turn.cubes[0].get\_value(), turn.cubes[1].get\_value(),turn.color)
704. entrance1 **=** entrance\_details[0]
705. entrance2 **=** entrance\_details[1]
707. **if** entrance1.get("column") **==** **-**1:  # Если кубик 1 не доступен для хода
708. turn.set\_cube\_played(0, True)  # устанавливаем played в true
709. **if** turn.double\_ind >**=** 0:  # Если дубль и игрок не может ходить
710. **for** i **in** range(4):
711. turn.set\_cube\_played(i, True)
713. **if** entrance2.get("column") **==** **-**1:   # Если кубик 2 не доступен для хода
714. turn.set\_cube\_played(1, True)   # устанавливаем played в true
716. **if** turn.has\_all\_cubes\_playes():  # Если игрок сходил все свои ходы (все кубики разыграны)
717. turn.change\_turn()
718. **continue**
719. **else**:
720. entrance **=** entrance\_details[turn.chosen\_cube]
721. turn.set\_destination(Move(entrance.get("row"), entrance.get("column")), entrance.get("eat\_piece"), False)
722. **if** len(turn.player\_clicks) **==** 0:
723. waiting\_for\_enter **=** True
725. **for** event **in** pygame.event.get():
726. **if** event.type **==** pygame.QUIT:
727. run **=** False
728. # обработчик нажатия КНОПКИ МЫШИ
729. **if** event.type **==** pygame.MOUSEBUTTONUP:
730. pos **=** pygame.mouse.get\_pos()
731. cube **=** clicked\_on\_cube(pos)
732. **if** cube !**=** **-**1:  # Если игрок решил выбрать кубик
733. **if** turn.double\_ind **==** **-**1:  # Если текущий ход не дубль
734. **if** turn.cubes[cube].get\_played():  # Если кубик был разыгран
735. cube **=** 3 **-** (2 **+** cube)
736. turn.set\_chosen\_cube(cube) # смена кубика
738. # снимаем координаты для хода
739. **else**:
740. col **=** int(pos[0] **/** 75)
741. row **=** int(pos[1] **/** 262)
742. **if** col > 5:
743. col **=** col **-** 1
745. **if** len(turn.player\_clicks) < 1:  # Если игрок выбрал исходную шашку
746. move **=** Move(row, col)
748. **if** board.check\_if\_blocked(turn.cubes, turn.color, all\_in\_home):  # Если у игрока нет доступных ходов
749. WIN.blit(FONT.render("У Вас Нет Возможности Ходить, Ждите Следующего Хода!", False, GREEN),
750. (WIDTH **/** 2 **-** 50, HEIGHT **/** 2))
751. delay(1.5)
752. turn.change\_turn()
753. **continue**
755. **if** waiting\_for\_enter:  # Если у игрока есть сьеденные шашки ждём пока игрок выберет куда их поставить
756. temp **=** turn.destination.get("destination")
757. **if** temp.get\_row\_from() **==** move.get\_row\_from() **and** temp.get\_col\_from() **==** move.get\_col\_from():
759. board.return\_eaten\_piece\_to\_game(turn.color, turn.destination)
760. turn.set\_cube\_played(turn.chosen\_cube, True)
761. waiting\_for\_enter **=** False
762. turn.set\_source(None)
763. turn.remove\_destination()
765. **if** turn.double\_ind >**=** 0:
766. turn.increase\_double\_index()
768. **if** turn.has\_all\_cubes\_playes():  # Если игрок разыграл все кубики
769. turn.change\_turn()
770. turn.clear\_clicks()
772. **else**:
773. temp **=** board.get\_destination(turn.color, move, turn.get\_number\_of\_steps())
774. move.set\_row\_col\_to(temp.get\_row\_from(), temp.get\_col\_from())
775. turn.add\_click(move)
777. source\_legal, piece\_eat **=** board.is\_source\_legal(turn.color, move, all\_in\_home)
778. **if** source\_legal:  # Если ход возможен
779. out\_own\_piece **=** False
780. **if** move.get\_col\_to() > 11:
781. out\_own\_piece **=** True
782. move.set\_row\_col\_to(move.get\_row\_to(), 12)
783. turn.set\_source(move)
784. turn.set\_destination(Move(move.row\_to, move.col\_to), piece\_eat, out\_own\_piece)
785. **else**:
786. turn.clear\_clicks()
787. turn.set\_source(None)
788. turn.remove\_destination()
790. **else**:   # Если игрок выбрал желаемую ячейку назначения
791. **if** board.is\_destination\_legal(Move(row, col), move):  # Если выбранная ячейка корректна
792. current\_cube **=** turn.chosen\_cube
794. **if** turn.destination.get("out\_own\_piece"):
795. board.out\_piece(move, turn.color)
797. **else**:
798. **if** turn.destination.get("eat\_piece"):
799. board.eat\_piece(board.board[move.get\_row\_to()][move.get\_col\_to()].pick())
801. board.make\_move(move)
802. turn.clear\_clicks()
804. **if** turn.double\_ind >**=** 0:  # Если дубль
805. current\_cube **=** turn.double\_ind
806. turn.increase\_double\_index()
808. turn.set\_source(None)
809. turn.remove\_destination()
810. turn.set\_cube\_played(current\_cube, True)
812. **if** turn.has\_all\_cubes\_playes():  # Если игрок разыграл все кубики
813. turn.change\_turn()
814. turn.clear\_clicks()
816. **else**:   # Если выбранная ячейка некорректна
817. turn.clear\_clicks()
818. turn.set\_source(None)
819. turn.remove\_destination()
821. draw(board, turn, None) # отрисовка
822. pygame.quit()
823. quit()

826. **if** \_\_name\_\_ **==** "\_\_main\_\_":
827. main()
828. Файл main.py
829. **from** threading **import** Thread
830. **from** PyQt5 **import** QtCore, QtGui, QtWidgets
831. **from** PyQt5.QtCore **import** Qt
832. **from** PyQt5.QtWidgets **import** QMessageBox
833. **from** PyQt5.QtWidgets **import** **\***
835. **class** Ui\_Login(object):
836. **global** cipher, p\_cipher
837. **def** Correct(self):
838. **global** cipher, p\_cipher
839. cipher **=** ''
840. p\_cipher **=** ''
841. **def** encryption(text):
842. # Выбрать два простых различных числа
843. p, q **=** 89, 107
844. # Вычислить произведение
845. n **=** p **\*** q
846. # Вычислить функцию Эйлера
847. #fi = (p - 1) \* (q - 1)
848. # Выбрать открытую экспоненту
849. en **=** 3
851. # Вычислить шифротекст
852. **def** encrypt(val):
853. cypher **=** (val **\*\*** en) **%** n
854. **return** (cypher)
856. # Итоговая функция шифрования
857. **def** rsa\_encrypt(text):
858. **global** cipher,p\_cipher
859. **global** encrypte
860. encrypte **=** []
861. **for** i **in** range(len(text)):
862. encrypte.append(encrypt(ord(text[i])))
863. **return** encrypte
864. **global** cipher, p\_cipher
865. cipher **=** rsa\_encrypt(text)
866. cipher **=** ' '.join(map(str, cipher))
867. p\_cipher **=** rsa\_encrypt(text) # пароль
868. p\_cipher **=** ' '.join(map(str, p\_cipher))
870. u\_ind **=** 0
871. p\_ind **=** 0
872. encryption(self.lineEdit.text())
873. word **=** cipher
875. with open(r'src/username.txt', 'r') as fp:
877. lines **=** fp.readlines()
878. **for** line **in** lines:
879. line **=** ('\n'.join(filter(bool, line.split('\n'))))
880. u\_ind **+=** 1
881. **if** (word **==** line):
882. self.u\_ind **=** u\_ind
883. **break**
884. **if** ((self.lineEdit\_2.text() **==** "") or (self.lineEdit.text() == "")):
885. msg **=** QMessageBox()
886. msg.setIcon(QMessageBox.Warning)
887. msg.setText("Поле Username или Password не может быть пустым")
888. msg.setWindowTitle("Пустое поле")
889. msg.exec\_()
890. **break**
892. encryption(self.lineEdit\_2.text())
893. p\_word **=** p\_cipher
895. with open(r'src/password.txt', 'r') as fp:
897. p\_lines **=** fp.readlines()
898. **for** p\_line **in** p\_lines:
899. p\_line **=** ('\n'.join(filter(bool, p\_line.split('\n'))))
900. p\_ind **+=** 1
901. **if** (p\_word **==** p\_line):
902. self.p\_ind **=** p\_ind
904. **if** self.p\_ind **==** self.u\_ind:
905. **break**
907. **if** ((self.lineEdit\_2.text() **==** "") or (self.lineEdit.text() == "")):
908. **break**
910. **try**:
911. **if** (p\_word **==** p\_line) **and** (word **==** line):
912. print()
913. **except**:
914. line **=** ""
915. p\_line **=** ""
916. **if** (p\_word **==** p\_line) **and** (word **==** line):
917. **if** ((p\_ind **==** u\_ind) **and** ((p\_ind **or** u\_ind) !**=** 0)):
918. **global** Username
919. Username **=** cipher
920. LoginForm.close()
921. LK.show()
922. self.instance **=** Ui\_LK()
923. msg **=** QMessageBox()
924. msg.setIcon(QMessageBox.Information)
925. msg.setText(
926. "Вы успешно вошли")
927. msg.setWindowTitle("Успешный вход")
928. msg.exec\_()
929. li.label\_12.setText("Пользователь: " **+** ui.lineEdit.text())
930. li.label\_12.setAlignment(QtCore.Qt.AlignCenter)
931. self.lineEdit.setText("")
932. self.lineEdit\_2.setText("")
934. **else**:
935. msg **=** QMessageBox()
936. msg.setIcon(QMessageBox.Warning)
937. msg.setText("Такой Username или Password не существует")
938. msg.setWindowTitle("Неправильные данные")
939. msg.exec\_()
940. **else**:
941. msg **=** QMessageBox()
942. msg.setIcon(QMessageBox.Warning)
943. msg.setText("Такой Username или Password не существует")
944. msg.setWindowTitle("Неправильные данные")
945. msg.exec\_()
947. **def** encryptUser(self):
948. **global** cipher,p\_cipher
949. cipher **=** ''
950. p\_cipher **=** ''
951. **def** encryption(text):
952. **global** cipher, p\_cipher
953. # Выбрать два простых различных числа
954. p, q **=** 89, 107
955. # Вычислить произведение
956. n **=** p **\*** q
957. # Вычислить функцию Эйлера
958. #fi = (p - 1) \* (q - 1)
959. # Выбрать открытую экспоненту
960. en **=** 3
962. # Вычислить шифротекст
963. **def** encrypt(val):
964. cypher **=** (val **\*\*** en) **%** n
965. **return** (cypher)
967. # Итоговая функция шифрования
968. **def** rsa\_encrypt(text):
969. **global** encrypte
970. encrypte **=** []
971. **for** i **in** range(len(text)):
972. encrypte.append(encrypt(ord(text[i])))
973. **return** encrypte
974. cipher **=** rsa\_encrypt(text)
975. cipher **=** ' '.join(map(str, cipher))
976. p\_cipher **=** rsa\_encrypt(text)
977. p\_cipher **=** ' '.join(map(str, p\_cipher))
978. encryption(self.lineEdit.text())
979. word **=** cipher
981. **if** self.lineEdit.text() **==** "" or self.lineEdit\_2.text() == "" :
982. msg **=** QMessageBox()
983. msg.setIcon(QMessageBox.Warning)
984. msg.setText(
985. "Присутствуют пустые поля")
986. msg.setWindowTitle("Неправильные данные")
987. msg.exec\_()
988. **else**:
989. with open(r'src/username.txt', 'r') as fp:
990. # read all lines in a list
991. lines **=** fp.readlines()
992. **for** line **in** lines:
993. line **=** ('\n'.join(filter(bool, line.split('\n'))))
994. **if** word **==** line:
995. msg **=** QMessageBox()
996. msg.setIcon(QMessageBox.Warning)
997. msg.setText(
998. "Такой пользователь уже существует")
999. msg.setWindowTitle("Неправильные данные")
1000. msg.exec\_()
1001. **break**
1002. **else**:
1004. with open("src/username.txt", 'a', encoding**=**'utf-8') as file:
1005. file.write(f'{cipher}\n')
1006. msg **=** QMessageBox()
1007. msg.setIcon(QMessageBox.Information)
1008. msg.setText("Вы успешно зарегистрировались")
1009. msg.setWindowTitle("Успешная регистрация")
1010. msg.exec\_()

1013. encryption(self.lineEdit\_2.text())
1014. p\_word **=** ' '.join(map(str, p\_cipher))
1015. with open(r'src/password.txt', 'r') as fp\_file:
1016. # read all lines in a list
1017. p\_lines **=** fp\_file.readlines()
1018. **for** p\_line **in** p\_lines:
1019. p\_line **=** ('\n'.join(filter(bool, p\_line.split('\n'))))
1020. **if** word **==** line **or** word **==** "" or self.lineEdit.text() == "" or self.lineEdit\_2.text() == "":
1021. **break**;
1022. **else**:
1024. with open("src/password.txt", 'a', encoding**=**'utf-8') as p\_file:
1025. p\_file.write(f'{p\_cipher}\n')
1026. **return** cipher
1028. **def** setupUi(self, LoginForm):
1029. #  основное окно
1030. LoginForm.setWindowTitle("Вход")
1031. LoginForm.setObjectName("LoginForm")
1032. LoginForm.resize(1120, 880)
1033. LoginForm.setStyleSheet("background-color: rgb(76, 117, 163); border-radius: 50px")
1035. #  рамка
1036. self.frame **=** QtWidgets.QFrame(LoginForm)
1037. self.frame.setGeometry(QtCore.QRect(160, 270, 800, 511))
1038. self.frame.setLayoutDirection(QtCore.Qt.LeftToRight)
1039. self.frame.setAutoFillBackground(False)
1040. self.frame.setStyleSheet("background-color: rgb(204,204,255)")
1041. self.frame.setFrameShape(QtWidgets.QFrame.StyledPanel)
1042. self.frame.setFrameShadow(QtWidgets.QFrame.Raised)
1043. self.frame.setObjectName("frame")
1045. #  надпись АВТОРИЗАЦИЯ
1046. font **=** QtGui.QFont()  # шрифт
1047. font.setFamily("Nirmala UI Semilight")
1048. font.setPointSize(28)
1050. self.label\_2 **=** QtWidgets.QLabel(self.frame)
1051. self.label\_2.setText("  Авторизация")
1052. self.label\_2.setEnabled(True)
1053. self.label\_2.setGeometry(QtCore.QRect(190, 50, 421, 61))
1054. self.label\_2.setFont(font)
1055. self.label\_2.setCursor(QtGui.QCursor(QtCore.Qt.ArrowCursor))
1056. self.label\_2.setMouseTracking(False)
1057. self.label\_2.setFocusPolicy(QtCore.Qt.NoFocus)
1058. self.label\_2.setContextMenuPolicy(QtCore.Qt.DefaultContextMenu)
1059. self.label\_2.setAcceptDrops(False)
1060. self.label\_2.setLayoutDirection(QtCore.Qt.LeftToRight)
1061. self.label\_2.setAutoFillBackground(False)
1062. self.label\_2.setIndent(25)
1063. self.label\_2.setOpenExternalLinks(False)
1064. self.label\_2.setObjectName("label\_2")

1067. # шрифт для полей ввода
1068. font **=** QtGui.QFont()
1069. font.setFamily("Nirmala UI Semilight")
1070. font.setPointSize(10)

1073. #  поле ввода "Имя пользователя"
1074. self.lineEdit **=** QtWidgets.QLineEdit(self.frame)
1075. self.lineEdit.setPlaceholderText("Имя пользователя")
1076. self.lineEdit.setGeometry(QtCore.QRect(190, 160, 421, 41))
1077. self.lineEdit.setFont(font)
1078. self.lineEdit.setFocusPolicy(QtCore.Qt.StrongFocus)
1079. self.lineEdit.setStyleSheet("background-color: rgb(255,255,255); padding-left: 8px")
1080. self.lineEdit.setObjectName("lineEdit")
1082. #  поле ввода "Пароль"
1083. self.lineEdit\_2 **=** QtWidgets.QLineEdit(self.frame)
1084. self.lineEdit\_2.setPlaceholderText("Пароль")
1085. self.lineEdit\_2.setGeometry(QtCore.QRect(190, 220, 421, 41))
1086. self.lineEdit\_2.setFont(font)
1087. self.lineEdit\_2.setFocusPolicy(QtCore.Qt.StrongFocus)
1088. self.lineEdit\_2.setStyleSheet("background-color: rgb(255,255,255); padding-left: 8px")
1089. self.lineEdit\_2.setEchoMode(QtWidgets.QLineEdit.Password)
1090. self.lineEdit\_2.setObjectName("lineEdit\_2")

1093. # шрифт для кнопок
1094. font **=** QtGui.QFont()  # шрифт
1095. font.setFamily("Nirmala UI Semilight")
1096. font.setPointSize(12)

1099. #  кнопка "Зарегистрироваться"
1100. self.pushButton **=** QtWidgets.QPushButton(self.frame)
1101. self.pushButton.setText("Зарегистрироваться")
1102. self.pushButton.setGeometry(QtCore.QRect(190, 370, 421, 51))
1103. self.pushButton.setFont(font)
1104. self.pushButton.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
1105. self.pushButton.setStyleSheet("background-color: rgb(100, 149, 237)")
1106. self.pushButton.setObjectName("pushButton")
1107. self.pushButton.clicked.connect(self.encryptUser)  # оброботчик нажатия кнопки
1109. #  кнопка "Войти"
1110. self.pushButton\_2 **=** QtWidgets.QPushButton(self.frame)
1111. self.pushButton\_2.setText("Войти")
1112. self.pushButton\_2.setGeometry(QtCore.QRect(190, 300, 421, 51))
1113. self.pushButton\_2.setFont(font)
1114. self.pushButton\_2.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
1115. self.pushButton\_2.setStyleSheet("background-color: rgb(100, 149, 237)")
1116. self.pushButton\_2.setObjectName("pushButton\_2")
1117. self.pushButton\_2.clicked.connect(self.Correct)  # оброботчик нажатия кнопки


1121. # ЛИЧНЫЙ КАБИНЕТ
1123. **class** Ui\_LK(object):
1124. **def** start\_game(self):
1125. LK.hide()
1126. **import** main
1127. thread **=** Thread(target**=**main.main())
1129. LK.show()

1132. **def** ToLoginForm(self):
1133. msg **=** QMessageBox()
1134. msg.setIcon(QMessageBox.Warning)
1135. msg.setText("Вы действительно хотите вернуться на главную страницу? Ваши данные не будут сохранены")
1136. msg.setWindowTitle("Выход")
1137. msg.setStandardButtons(QMessageBox.Ok | QMessageBox.Cancel)
1138. msg.exec\_()
1139. result **=** msg.standardButton(msg.clickedButton())
1140. **if** result **==** 1024:
1141. LK.close()
1142. LoginForm.show()
1144. **def** setupUi(self, LK):
1145. # основное окно
1146. LK.setWindowTitle("Личный кабинет")
1147. LK.setObjectName("LK")
1148. LK.resize(850, 550)
1149. LK.setStyleSheet("background-color: rgb(76, 117, 163); border-radius: 10px")
1151. # шрифт для кнопок
1152. font **=** QtGui.QFont()
1153. font.setFamily("Nirmala UI Semilight")
1154. font.setPointSize(12)
1156. # кнопка "Играть"
1157. self.pushButton **=** QtWidgets.QPushButton(LK)
1158. self.pushButton.setText("Играть")
1159. self.pushButton.setGeometry(QtCore.QRect(285, 200, 280, 70))
1160. self.pushButton.setFont(font)
1161. self.pushButton.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
1162. self.pushButton.setStyleSheet("background-color: rgb(204,204,255);")
1163. self.pushButton.setObjectName("pushButton")
1164. self.pushButton.clicked.connect(self.start\_game) # оброботчик нажатия кнопки
1166. # кнопка "Выход"
1167. self.pushButton\_2 **=** QtWidgets.QPushButton(LK)
1168. self.pushButton\_2.setText("Выход")
1169. self.pushButton\_2.setGeometry(QtCore.QRect(285, 350, 280, 70))
1170. self.pushButton\_2.setFont(font)
1171. self.pushButton\_2.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
1172. self.pushButton\_2.setStyleSheet("background-color: rgb(204,204,255);")
1173. self.pushButton\_2.setObjectName("pushButton\_2")
1174. self.pushButton\_2.clicked.connect(self.ToLoginForm) # оброботчик нажатия кнопки
1176. # имя пользователя
1177. font **=** QtGui.QFont() # шрифт
1178. font.setFamily("Segoe UI")
1179. font.setPointSize(18)
1180. font.setBold(False)
1181. font.setItalic(False)
1182. font.setWeight(50)
1183. font.setKerning(False)
1185. self.label\_12 **=** QLabel(LK)
1186. self.label\_12.setAlignment(Qt.AlignCenter)
1187. self.label\_12.setText("Пользователь: ")
1188. self.label\_12.move(700, 70)
1190. self.label\_12.setGeometry(QtCore.QRect(0, 50, 850, 70))
1191. self.label\_12.setFont(font)
1192. self.label\_12.setStyleSheet("color: rgb(0,0,0);")
1193. self.label\_12.setObjectName("label\_12")
1194. self.label\_12.setAlignment(Qt.AlignCenter)
1196. **if** \_\_name\_\_ **==** "\_\_main\_\_":
1197. **import** sys
1198. app **=** QtWidgets.QApplication(sys.argv) #  создаем объект приложения
1199. # sys.argv - это список аргументов из командной строки.
1200. # Скрипты Python могут быть запущены из программной оболочки.
1201. # Это один из способов, как мы можем контролировать запуск наших скриптов.
1202. LoginForm **=** QtWidgets.QDialog()
1203. LK **=** QtWidgets.QDialog()
1204. ui **=** Ui\_Login()
1205. li **=** Ui\_LK()
1206. ui.setupUi(LoginForm)
1207. li.setupUi(LK)
1208. LoginForm.show()
1209. sys.exit(app.exec\_()) # главный цикл приложения
1210. Файл Move.py
1211. # Класс для каждого хода , при клике устанавливаются row\_from и col\_from ,
1212. # при повторном клике устанавливаются  the row\_to и col\_to params
1213. **class** Move:
1215. **def** \_\_init\_\_(self, row\_from, col\_from):
1216. self.row\_from **=** row\_from
1217. self.col\_from **=** col\_from
1218. self.row\_to **=** **-**1
1219. self.col\_to **=** **-**1
1220. self.have\_dest **=** False
1222. # Функция возвращает значение row\_from
1223. **def** get\_row\_from(self):
1224. **return** self.row\_from
1226. # Функция возвращает значение the col\_from
1227. **def** get\_col\_from(self):
1228. **return** self.col\_from
1230. # Функция устанавливает значение row\_from
1231. **def** set\_row\_from(self, row):
1232. self.row\_from **=** row
1234. # Функция устанавливает значение col\_from
1235. **def** set\_col\_from(self, col):
1236. self.col\_from **=** col
1238. # Функция возвращает значение row\_to
1239. **def** get\_row\_to(self):
1240. **return** self.row\_to
1242. # Функция возвращает значение col\_to
1243. **def** get\_col\_to(self):
1244. **return** self.col\_to
1246. # Функция устанавливает row\_to и  col\_to
1247. **def** set\_row\_col\_to(self, row\_to, col\_to):
1248. self.row\_to **=** row\_to
1249. self.col\_to **=** col\_to
1250. self.have\_dest **=** True # меняем флаг

1253. # для отладки
1254. **def** \_\_eq\_\_(self, move):
1255. **return** self.row\_from **==** move.row\_from **and** self.col\_from **==** move.col\_from
1257. **def** \_\_hash\_\_(self):
1258. **return** hash((self.row\_from, self.col\_from))
1260. **def** \_\_str\_\_(self):
1262. **return** "Откуда({0}, {1}) ==> Куда({2}, {3})".format(self.row\_from, self.col\_from, self.row\_to, self.col\_to)
1263. Файл Piece.py
1264. # Класс для одной шашки (красную или белую)
1265. **class** Piece:
1266. **def** \_\_init\_\_(self, color, image, row, col, row\_visual, col\_visual):
1267. self.color **=** color
1268. self.image **=** image
1269. self.row **=** row
1270. self.col **=** col
1271. self.row\_visual **=** row\_visual
1272. self.col\_visual **=** col\_visual
1274. # Функция возвращает цвет шашки
1275. **def** get\_color(self):
1276. **return** self.color
1278. #Функция возвращает картинку шашки
1279. **def** get\_img(self):
1280. **return** self.image
1282. # Функция возвращает текущую строку
1283. **def** get\_row(self):
1284. **return** self.row
1286. # Функция устанавливает строку из полученной
1287. **def** set\_row(self, row):
1288. self.row **=** row
1290. # Функция возвращает текущий столбец шашки
1291. **def** get\_col(self):
1292. **return** self.col
1294. # Функция устанавливает столбец
1295. **def** set\_col(self, col):
1296. self.col **=** col
1298. # Функция возвращает строку для ототбражения на доске
1299. **def** get\_row\_visual(self):
1300. **return** self.row\_visual
1302. #  Функция устанавливает строку для ототбражения на доске
1303. **def** set\_row\_visual(self, row\_visual):
1304. self.row\_visual **=** row\_visual
1306. # Функция возвращает столбец для ототбражения на доске
1307. **def** get\_col\_visual(self):
1308. **return** self.col\_visual
1310. #  Функция устанавливает столбец для ототбражения на доске
1311. **def** set\_col\_visual(self, col\_visual):
1312. self.col\_visual **=** col\_visual
1314. #  True если шашки имеют одно и ту же строку и столбец, False если нет
1315. **def** \_\_eq\_\_(self, piece):
1316. **return** self.row **==** piece.row **and** self.col **==** piece.col
1318. **def** \_\_hash\_\_(self):
1319. **return** hash((self.row, self.col))
1321. # все данные о шашке
1322. **def** \_\_str\_\_(self):
1323. **return** "Color = {0}\nImage = {1}\nRow = {2}\nColumn = {3}\n".format(self.color, self.image, self.row, self.col)
1324. Файл Player.py
1325. **from** Piece **import** Piece

1328. # Класс для игрока
1329. **class** Player:
1331. **def** \_\_init\_\_(self, type, positions):
1332. self.type **=** type
1333. self.pieces **=** []
1334. self.set\_pieces(positions)
1335. self.eaten **=** []
1336. self.row\_home **=** 0
1337. **if** type **==** "white":
1338. self.row\_home **=** 1
1339. self.all\_home **=** False
1341. # Функция возвращает все шашки игрока
1342. **def** get\_pieces(self):
1343. **return** self.pieces
1345. # Функция устанавливает все шашки игрока в параметр pieces(это класс Piece)
1346. **def** set\_pieces(self, positions):
1347. image **=** "white.png"
1348. **if** self.type **==** "red":
1349. image **=** "red.png"
1351. **for** i **in** range(15):
1352. x **=** int(positions[i][1]**/**263)
1353. y **=** int(positions[i][0]**/**75)
1354. self.pieces.append(Piece(self.type, image, x, y, positions[i][0], positions[i][1]))
1356. **def** add\_piece(self, piece):
1357. self.pieces.append(piece)
1359. #  Функция упорядочивает шашки для начала игры
1360. **def** start\_game(self):
1361. **if** self.type **==** "red":
1362. **pass**
1363. **elif** self.type **==** "white":
1364. **pass**
1366. # добавляет съеденную шашку в съеденный список
1367. **def** add\_piece\_eaten(self, piece):
1368. self.eaten.append(piece)
1370. # Функция для проверки все ли шашки дома(для вывода с доски)
1371. **def** set\_all\_home(self, all\_\_home):
1372. self.all\_home **=** all\_\_home
1374. # все данные вывод
1375. **def** \_\_str\_\_(self):
1376. pieces **=** []
1377. **for** piece **in** self.pieces:
1378. pieces.append(piece.\_\_str\_\_())
1379. pieces.append("-------------------\n")
1381. **return** " ".join(pieces)
1382. Файл Stack.py
1383. # Класс  для поля (стек)  список в котором все данные о шашках на текущей ячейке
1384. **class** Stack:
1386. **def** \_\_init\_\_(self):
1387. self.stack **=** []
1389. #  Функция возвращает верхний элемент
1390. **def** pick(self):
1391. **return** self.stack[0]
1393. #  Функция возвращает верхний элемент и удаляет его
1394. **def** pop(self):
1395. **return** self.stack.pop(0)
1397. # Функция вставляет шашку в конец(наверх) верхний элемент
1398. **def** push(self, value):
1399. self.stack.insert(0, value)
1401. # отладка в строку
1402. **def** \_\_str\_\_(self):
1403. **return** "\n".join(str(val) **for** val **in** self.stack)
1405. # все элементы стека
1406. **def** get\_all\_stack\_elements(self):
1407. temp\_s **=** Stack()
1408. values **=** []
1410. **for** i **in** range(len(self.stack)):
1411. values.append(self.pick())
1412. temp\_s.push(self.pop())
1414. **for** i **in** range(len(temp\_s.stack)):
1415. self.push((temp\_s.pop()))
1417. **return** values
1419. # Функция возвращает длину стека
1420. **def** stack\_len(self):
1421. **return** len(self.stack)
1422. Файл Turn.py
1423. **import** random, copy
1424. **from** Cube **import** Cube

1427. # Класс для единичного хода в игре
1428. **class** Turn:
1430. **def** \_\_init\_\_(self, ai\_game):
1431. self.red\_turn **=** True
1432. self.color **=** "red"
1433. self.player\_clicks **=** []
1434. self.cubes **=** []
1435. self.chosen\_cube **=** 0
1436. self.generate\_cubes()
1437. self.source **=** None
1438. self.destination **=** None
1439. self.double\_ind **=** **-**1
1440. self.ai\_game **=** ai\_game
1441. **if** ai\_game:
1442. self.red\_turn **=** False
1443. self.color **=** "white"
1444. self.ai\_turn **=** True
1446. # функция для смены  очереди хода
1447. **def** change\_turn(self):
1448. **if** self.ai\_game:
1449. **if** self.ai\_turn:
1450. self.ai\_turn **=** False
1451. self.color **=** "red"
1452. self.red\_turn **=** True
1453. **else**:
1454. self.ai\_turn **=** True
1455. self.color **=** "white"
1456. self.red\_turn **=** False
1457. **else**:
1458. **if** self.red\_turn:
1459. self.red\_turn **=** False
1460. self.color **=** "white"
1461. **else**:
1462. self.red\_turn **=** True
1463. self.color **=** "red"
1464. self.generate\_cubes()
1466. # Эта функция генерирует 2 случайных числа  1-6 и вставляет их в массив cubes
1467. **def** generate\_cubes(self):
1468. cube1 **=** Cube(random.randint(1, 6))
1469. cube2 **=** Cube(random.randint(1, 6))
1471. self.chosen\_cube **=** 0
1473. **if** cube1.get\_value() **==** cube2.get\_value(): # если дубль
1474. self.cubes **=** [cube1, copy.deepcopy(cube1), copy.deepcopy(cube1), copy.deepcopy(cube1)]
1475. self.double\_ind **=** 0
1476. **else**:
1477. self.cubes **=** [cube1, cube2]
1478. self.double\_ind **=** **-**1
1480. #  функция изменяет номер выбранного кубика
1481. **def** set\_chosen\_cube(self, cube\_number):
1482. self.chosen\_cube **=** cube\_number
1483. self.clear\_clicks()
1485. # функция возвращает номер хода
1486. **def** get\_number\_of\_steps(self):
1487. **return** self.cubes[self.chosen\_cube].get\_value()
1489. #  функция добавляет новый клик ко всем кликам игрока в этом ходу
1490. **def** add\_click(self, click):
1491. self.player\_clicks.append((click))
1493. #  функция очищает клики игрока
1494. **def** clear\_clicks(self):
1495. self.player\_clicks.clear()
1497. # функция устанавливает новый источник для этого хода
1498. **def** set\_source(self, source):
1499. self.source **=** source
1501. # функция устанавливает пункт назначения, в который игроку необходимо передать выбранную фигуру.
1502. **def** set\_destination(self, destination, eat\_piece, out\_own\_piece):
1503. self.destination **=** {"destination": destination, "eat\_piece": eat\_piece, "out\_own\_piece": out\_own\_piece}
1505. # функция удаляет destination, устанавливая его значение равным None
1506. **def** remove\_destination(self):
1507. self.destination **=** None
1509. **def** remove\_source(self):
1510. self.source **=** None
1512. # Эта функция получает номер кубика и его статус
1513. # и устанавливает их для кубика, который имеет значение параметра cube\_number
1514. **def** set\_cube\_played(self, cube\_number, played):
1515. self.cubes[cube\_number].set\_played(played)
1516. self.chosen\_cube **=** 3 **-** (2 **+** cube\_number)
1518. #  функция возвращает значение True, если все кубики были сыграны, и значение False, если нет
1519. **def** has\_all\_cubes\_playes(self):
1520. **for** i **in** range(len(self.cubes)):
1521. **if** **not** self.cubes[i].get\_played():
1522. **break**
1523. **else**:
1524. **return** True
1526. **return** False
1528. # функция увеличивает двойной индекс и тем самым переходит к следующему партии из 4 кубиков
1529. **def** increase\_double\_index(self):
1530. self.double\_ind **=** self.double\_ind **+** 1
1532. **def** \_\_str\_\_(self):
1533. **return** "Red turn: {0}, player clicks: {1}, cubes: {2}, chosen cube: {3}".format(self.red\_turn,
1534. self.player\_clicks,
1535. self.cubes,
1536. self.chosen\_cube)