МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Измерительно-вычислительные комплексы»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

на курсовую работу

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

по дисциплине «Алгоритмы и структуры данных»

# Тема: Компьютерная логическая игра «Атари - Го»

# Р.02069337. 23/694-24

Листов 42

Руководитель разработки:  
Кандидат технических наук, доцент кафедры “ИВК”  
 Шишкин Вадим Викторинович «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2024 г.

Исполнитель:

студент гр. ИСТбд-22

Семин Артём Рудольфович

« » 2024 г.

2024

**Содержание**

[Аннотация 3](#_Toc185935004)

[ТЕХНИЧЕСКОЕ ЗАДАНИЕ 4](#_Toc185935005)

[Пояснительная записка 10](#_Toc185935008)

[Руководство программиста 17](#_Toc185935009)

[Тестовая документация 37](#_Toc185935010)

[Полный код программы: 44](#_Toc185935011)

# Аннотация

Данный документ представляет собой пояснительную записку на

курсовую работу на тему «Компьютерная логическая игра Атари – Го».

Документ содержит следующие разделы: техническое задание,

пояснительная записка, руководство программиста, тестовая документация и код программы; в нем излагается постановка задачи и описание реализуемой программы, ее назначение. Документ может быть использован в качестве инструкции для применения рассматриваемого программного средства.

**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 «Атари-Го»

# Р.02069337. 23/694-24

# Листов 5

**Исполнитель**:

студент гр. ИСТбд-22

Семин Артём Рудольфович

« » 2024 г.

**2024**

**Введение:**

Указывается наименование и условное обозначение разрабатываемого приложения, наименования реализуемой игры. Даётся краткий свод правил игры, общая характеристика функциональных возможностей, которое должно предоставлять приложение.

Курсовая работа представляет собой однопользовательское приложение, реализующее логическую игру "Атари - Го".

Краткие правила игры:

* Игра ведется на квадратной доске, игроки по очереди ставят фишки (черные и белые) на пересечения линий.
* Необходимо захватить больше территории, окружив фишки противника.
* Побеждает игрок, первым окружившим один или несколько фишек противника.

# 1. Основания для разработки

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

# 2. Требования к программе или программному изделию

**2.1. Функциональное назначение**

Основное назначение приложения - предоставление интерактивной среды, позволяющей играть в «Атари-Го».   
Приложение должно обеспечивать:

1)Визуальное представление: отображение фишки на графическом интерфейсе.

2)Расставление фишек: Игроки по очереди кликают мышью по пустым клеткам на игровом поле, чтобы разместить свою фишку (черный или белый)

Автоматизируемые процессы:

1. Отрисовка игры: Приложение визуализирует игровое поле и фишки на нем, обновляя изображение после каждого хода.

2. Обработка пользовательского ввода: Приложение отслеживает клики мыши на игровом поле. Если игрок кликает по допустимой клетке, программа размещает его фишку на поле.

3. Проверка допустимости хода: Приложение проверяет, не стоит ли уже фишка на выбранной клетке и не нарушает ли ход правила игры.

4. Определение победителя: Приложение анализирует положение фишек на поле, чтобы определить, окружен ли какой-либо из фишек и, следовательно, был ли один из игроков победителем.

**2.2. Требования к функциональным характеристикам**

1. Правила игры:
   * Игра проводится на доске размера 18x18;
   * Ходы выполняются по очереди;
   * Играют два игрока: один использует черные фишки, другой — белые.
   * Игроки поочередно ставят свои фишки на пересечения линий на доске.
   * Фишка или группа фишек противника, которые вы окружили своими камнями, снимается с доски.
   * Побеждает игрок, первым окружившим один или несколько фишек противника.
2. Цель игры:
   * Захватить фишку или группу фишек противника и контролировать больше территории на доске.
3. Условия победы:
   * Победа достигается тогда, когда происходит захват фишек или несколько фишек на доске.

2.2.1 Требования к структуре приложения

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

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 приём, проверка правильности и отрисовка хода пользователя;

- проверка окончания игры;

- вычисление, проверка правильности и отрисовка хода противника;

- информирование пользователя об окончании игры и победителя.

2.2.3. Требования к организации информационного обеспечения, входных и выходных данных

В приложении должен быть реализован графический интерфейс взаимодействия с пользователем. Изображения фишек могут храниться в отдельных графических файлах. Логин и пароль пользователя должны вводиться с клавиатуры.

**2.3 Требования к надёжности**

Приложение работоспособно при последовательном нажатии пользователем ходов на доске, при любом поведении пользователя в окне меню. Надежность обеспечивается путем неоднократного тестирования приложения.

Работоспособность:

1) Программа должна функционировать без проблем и ошибок на указанных операционных системах (Windows, Linux, macOS).

2) Графический интерфейс приложения должна быть динамичной и оперативно откликаться на действия пользователя.

3) Все возможности программы должны выполняться без сбоев и исправно.

Восстановление при сбоях:

1) При возникновении ошибки, программа должна выдать понятное сообщение, не прекращая свою работу.

2) Необходимо предусмотреть возможность перезапуска приложения без утраты сохраненных данных.

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

Операционная система: Windows, Linux, MacOS.

Язык программирования: Python 3.9.1

Среда разработки: PyCharm Community Edition 2023.1

Библиотеки: pygame, numpy, itertools, sys, networkx, collections.

**2.5. Требования к маркировке и упаковке**

Определяются заданием на курсовую работу.

**2.6. Требования к транспортированию и хранению**

2.6.1 Условия транспортирования

Требования к условиям транспортирования не предъявляются.

2.6.2 Условия хранения

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

2.6.3 Сроки хранения

Срок хранения — до окончания срока учебы.

**3. Требования к программной документации**

Определяются заданием на курсовую работу.

**4. Стадии и этапы разработки**

Определяются заданием на курсовую работу.

**5. Порядок контроля и приемки**

Определяются заданием на курсовую работу.

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

федеральное государственное бюджетное образовательное учреждение

высшего образования

**«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Кафедра «Измерительно-вычислительные комплексы»

**Курсовая работа**

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

**Тема** Компьютерная логическая игра «Атари-Го»

**Пояснительная записка**

# Р.02069337. 23/694-24

Листов 7

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

**Исполнитель**:

студент гр. ИСТбд-22

Семин Артём Рудольфович

« » 2024 г.

2024

**Введение**

Указывается наименование и условное обозначение разрабатываемого приложения, наименования реализованной игры. Приводится описание и обоснование выбранного подхода, краткое описание реализованного приложения.

**1. Проектная часть**

**1.1 Постановка задачи на разработку приложения**

Определяется заданием на курсовую работу.

**1.2 Математические методы**

1. Описание состояния игры:

* + Положение фишек на доске. Необходимо использование двумерного массива размером 18x18, где каждый элемент представляет ячейку доски и содержит информацию о наличии или отсутствии фишки.
  + Состояние игры: текущий игрок, наличие взятия на проходе, возможность хода выбранной фигуры, отслеживание таких ситуаций, как: мат, пат и блокада, отслеживание состояния при котором пешка достигает вражеской стороны и имеет возможность поменять себя на другую фигуру.

1. Правила игры:
   * Игроки поочередно ставят свои фишки на пересечения линий на доске.
   * Ограничения на действия: нельзя ходить на занятую клетку.
   * Обязательное условие окончания игры, захватить фишку или группу фишек противника.
2. Алгоритмы:
   * Проверка возможности хода с учетом правил и ограничений.
   * Игроки могут запоминать результаты своих действий и адаптировать свою стратегию в зависимости от поведения соперника.
   * Один игрок может сосредоточиться на захвате территорий, а другой — на блокировке противника.

Выбор такой модели обоснован тем, что Атари-Го является стратегической настольной игрой, в которой игроки принимают решения на основе анализа возможных ходов и прогнозирования дальнейших ходов соперника. Математическая модель позволяет представить все состояния игры, правила и алгоритмы в виде логических и математических выражений, что упрощает анализ и программную реализацию игры.

**Основные функции:**

Функция makegrid():

* + **Назначение**: Создает сетку для игрового поля.
  + **Описание**: Возвращает список пар начальных и конечных точек, которые определяют линии сетки на доске. Используется для визуализации линий на игровом поле.

Функция xy\_to\_colrow():

* + **Назначение**: Преобразует координаты (x, y) в номера столбца и строки.
  + **Описание**: Определяет, в каком столбце и строке находится точка на игровом поле, что позволяет обработать клики мыши.

Функция colrow\_to\_xy():

* + **Назначение**: Преобразует номера столбца и строки в координаты (x, y).
  + **Описание**: Используется для определения положения камней на доске в пикселях на основе их столбца и строки.

Функция has\_no\_liberties():

* + **Назначение**: Проверяет, есть ли у группы камней "свободы".
  + **Описание**: Определяет, окружены ли камни противника, и могут ли они быть сняты с доски.

Функция get\_stone\_groups():

* + **Назначение**: Получает группы камней определенного цвета на доске.
  + **Описание**: Использует графы для нахождения соединенных компонентов камней одного цвета, что позволяет определить, какие камни могут быть сняты.

Функция is\_valid\_move():

* + **Назначение**: Проверяет, является ли ход допустимым.
  + **Описание**: Убедитесь, что выбранная клетка не занята и находится в пределах границ доски.

Функция draw\_menu():

* + **Назначение**: Отображает главное меню игры.
  + **Описание**: Рисует кнопки для начала игры, регистрации и отображения правил на экране.

Функция handle\_menu\_click()

* + **Назначение**: Обрабатывает нажатия кнопок в главном меню.
  + **Описание**: Проверяет, была ли нажата одна из кнопок меню, и возвращает соответствующее действие.

Функция show\_rules():

* + **Назначение**: Отображает правила игры на экране.
  + **Описание**: Заполняет экран правилами игры и ожидает, пока игрок не вернется в меню.

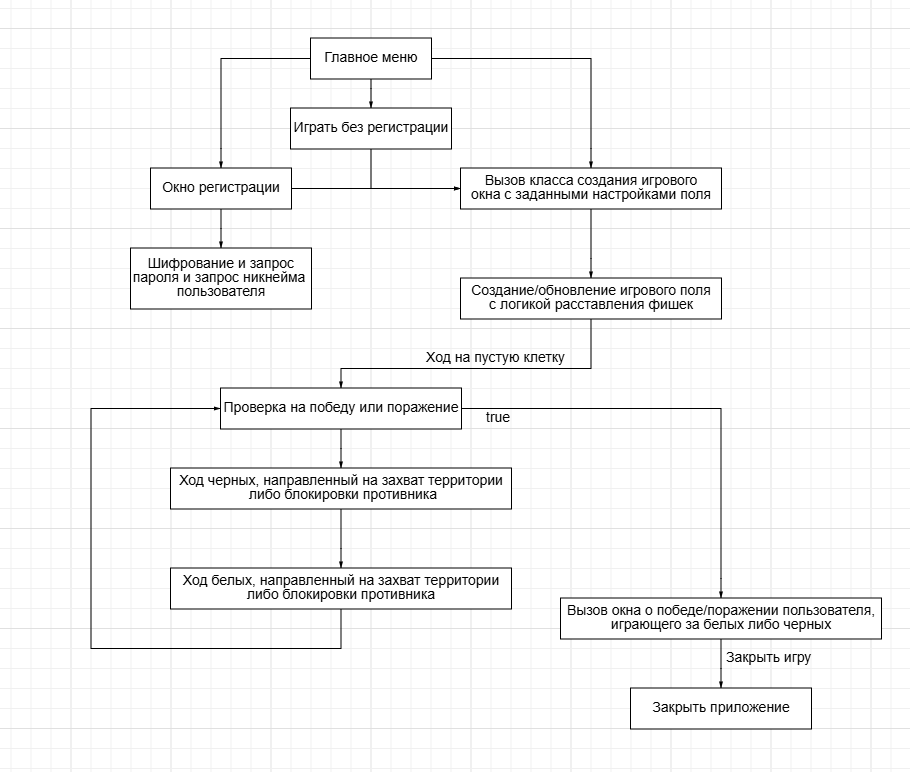
Функция Game.start():

* + **Назначение**: Запускает игровую сессию.
  + **Описание**: Инициализирует Pygame, отображает главное меню и обрабатывает ввод пользователя для начала игры, регистрации или просмотра правил.

**1.3 Архитектура и алгоритмы**

1.3.1. Архитектура

Основные структуры данных(классы и основные методы в классе):

****

**1.3.2 Алгоритм захват территории противника**

Стратегическое размещение для захвата территории (контроля):

• Контроль:

\* Игрок контролирует пространство, не давая противнику размещать свои фишки в ключевых местах.

\* Размещает фишки так, чтобы противник не мог занять стратегически важные позиции.

\* Создает свои "острова" и не дает противнику их окружить, чтобы тем самым гарантировать себе мобильность.

Анализ ситуации после ходов противника:

• После хода противника, игрок должен снова проанализировать ситуацию на доске:

\* Есть ли у противника уязвимые фишки?

\* Появились ли возможности для захвата территории (ограничения ходов противника)?

\* Какие шаги нужно предпринять, чтобы не дать противнику захватить вашу территорию?

Ключевые моменты алгоритма:

• Активное воздействие: Стратегия не пассивная защита, а активное ограничение действий противника.

• Многоходовка: Захват территории — это не всегда мгновенный результат. Это планирование и стратегия на несколько ходов.

• Гибкость: Необходимо быть готовым менять свою тактику в зависимости от действий противника.

**1.3.2 Алгоритм блокировки территории противника**

Как использовать алгоритм захвата для блокировки территории:

• Ограничение свободы: Ключевой принцип – лишение фишек противника "свобод" (дыханий) с помощью собственных фишек.

• Вы размещаете свои фишки так, чтобы уменьшать количество соседних пустых клеток у фишек противника.

• Если противник не может обеспечить своим фишкам свободные соседние клетки, его фишки будут захвачены.

• Создание "стен": Можно образовывать "стены" из своих фишек вокруг фишек противника. Это предотвратит их расширение и позволит вам контролировать территорию.

• Влияние на мобильность: Ограничивая свободы фишек противника, вы ограничиваете их мобильность, что не позволит им захватывать ваши фишки.

Ключевые моменты стратегии блокировки:

• Предвидение: Понимание, как работают функции (проверка наличия или отсутствия свободы у групп фишек, для последующего захвата) и (анализ групп фишек противника и определение их слабых мест) поможет вам прогнозировать ходы противника.

• Соединение и разделение: Соединяйте свои фишки в прочные группы и разделяйте группы противника, что повысит их уязвимость.

• Контроль пространства: Стремитесь создать такую конфигурацию, что противник не сможет занять выгодную позицию.

• Баланс: Необходимо одновременно пытаться блокировать противника и следить, чтобы ваши собственные фишки не остались без свобод, что приведёт к их захвату.

Количество рассмотренных алгоритмов должно полностью соответствовать техническому заданию. Для алгоритмов шифрования, дешифрования, проверки доступности хода, хода и оценочного добавить блок- схему. В схемах и комментариях не должны использоваться конструкции языка программирования, конкретные названия объектов программы и т.п. (пояснительная записка создаётся до реализации).

**2. Источники, использованные при разработке**

1. "Metanit.com" [Электронный ресурс]. - Режим доступа: https://metanit.com. - Дата доступа: 23.12.2024

2. "gomagic.org" [Электронный ресурс]. - Режим доступа: https://gomagic.org/ru/how-to-play-go-rules - Дата доступа: 23.12.2024

3. "atarigo.narod.ru" [Электронный ресурс]. - Режим доступа: https://atarigo.narod.ru - Дата доступа: 23.12.2024

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

федеральное государственное бюджетное образовательное учреждение

высшего образования

**«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Кафедра «Измерительно-вычислительные комплексы»

**Курсовая работа**

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

**Тема** Компьютерная логическая игра «Атари-Го»

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

# Руководство программиста

**Р.02069337. 23/694-24**

Листов 20

Исполнитель:

студент гр. ИСТбд-22

Семин Артём Рудольфович

« » 2024 г.

2024

**1. Назначение и условия применения программы**

**1.1 Назначение и функции, выполняемые приложением**

Формулируется назначение приложения. Даётся свод правил реализуемой игры, общая характеристика конкретных функциональных возможностей, которое предоставляет приложение.

Приложение предназначено для развлечения, развития интеллекта, совершенствования способности игры в шахматы.

Правила игры:

* + Игра проводится на доске размера 18x18;
  + Ходы выполняются по очереди;
  + Играют два игрока: один использует черные фишки, другой — белые.
  + Игроки поочередно ставят свои фишки на пересечения линий на доске.
  + Фишка или группа фишек противника, которые вы окружили своими фишками, снимается с доски.
  + Побеждает игрок, первым окружившим один или несколько фишек противника.

В приложении реализованы: главное меню, окно регистрации, выбор режима игры, модуль, который отрисовывает игру и проверяет на мат/пат/блокаду, логика искусственного интеллекта, меню для создания новой партии или выхода из приложения

**1.2 Условия, необходимые для использования приложения**

1. Операционная система: Приложение должно быть доступно для Windows, macOS и Linux.

2. Платформа: Приложение должно быть написано на Python и совместимо с версиями Python 3 и выше.

3. Инструментальная среда: Для использования приложения необходимо установить Python и настроить его на компьютере пользователя.

4. Библиотеки: Приложение должно использовать библиотеки, такие как pygame для создания графического интерфейса игры и управления вводом пользователя, numpy для обработки эффективной работы с массивами данных, представляющими игровое поле, itertools для эффективной итерации по комбинациям координат на игровом поле, sys для выхода из программы, networkx для представления игрового поля в виде графа и поиска связных компонент (групп фишек), collections для создания словаря, а также стандартные библиотеки языка Python версии 3. и выше.

**2. Характеристики программы**

**2.1 Характеристики приложения**

Количество значимых строк: 557

Количество структур данных: 2(структура для хранения состояния доски и соответствующими методами поиска допустимых ходов, структура для хранения состояния доски, которая может прочитаться алгоритмом отрисовки интерфейса)

Количество алгоритмов: 3(алгоритм для отрисовки игрового поля и возможных ходов, алгоритм обработки хода, алгоритм определение победителя.

Описание используемых библиотек.

1. Библиотека pygame: Это кросс-платформенная библиотека Python для создания игр. Она предоставляет инструменты для работы с графикой, звуком, вводом с клавиатуры и мыши, а также для управления событиями. Pygame упрощает разработку 2D игр, делая её более доступной для начинающих программистов. В данном коде Pygame используется для отображения игрового поля, фишек, и обработки пользовательского ввода.

2. Библиотека numpy: Это мощная библиотека Python для работы с многомерными матрицами. Она предоставляет высокопроизводительные функции для математических операций, обработки данных и линейной алгебры. В коде NumPy используется для эффективного представления игрового поля как матрицы и выполнения операций над ним. Использование NumPy ускоряет обработку данных по сравнению с обычными списками Python.

3. Библиотека itertools: Эта библиотека Python предоставляет набор итераторов для эффективной работы с итерациями. Она позволяет генерировать различные комбинации и перестановки элементов, что удобно для перебора возможных ходов или координат на игровом поле. В данном проекте itertools используется для эффективного перебора координат для отрисовки вспомогательных точек на доске.

4. Библиотека sys: Эта библиотека предоставляет доступ к переменным и функциям, связанным с интерпретатором Python. Она позволяет взаимодействовать с окружением выполнения скрипта, например, для завершения программы или получения информации о системе. В данном коде sys используется для корректного завершения работы игры при закрытии окна или других ошибках.

5. Библиотека networkx: Эта библиотека Python для создания, манипулирования и анализа структуры графов. Она предоставляет широкий набор алгоритмов для работы с графами, включая поиск путей, определение связных компонент и многое другое. В этом коде networkx используется для нахождения групп фишек на игровом поле.

6. Библиотека collections: Библиотека collections расширяет возможности стандартных типов данных Python. Она содержит специализированные контейнеры, такие как namedtuple, deque, и defaultdict. В данном коде используется defaultdict для удобного подсчета взятых фишек, поскольку он автоматически создает новые записи с нулевым значением, если ключ отсутствует.

Основные компоненты библиотеки:

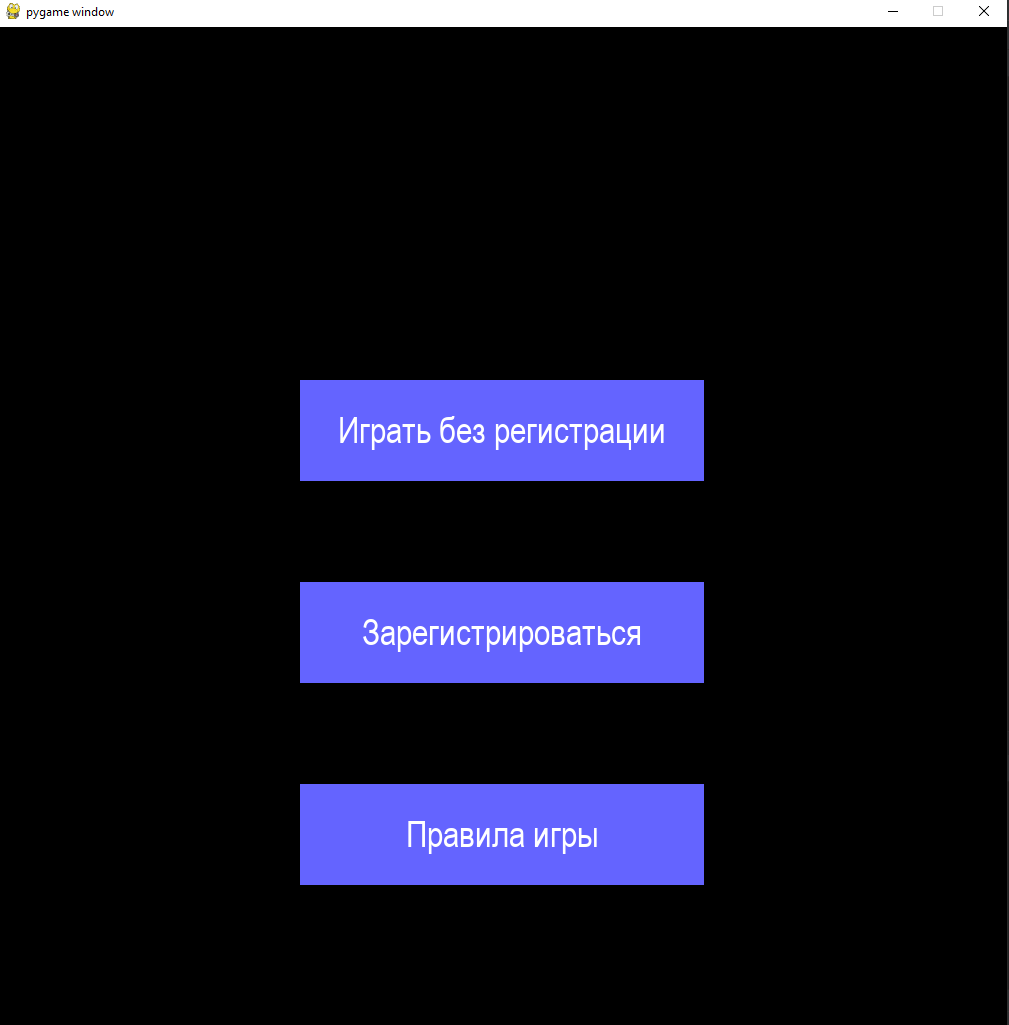
1. Доска (board): Двумерный массив NumPy, представляющий игровое поле. Он хранит информацию о расположении камней (черных и белых) на доске. Доска не имеет методов для выполнения ходов, а используется как структура данных в других алгоритмах.

2. Ход (place\_stone, handle\_captures): Функции place\_stone и handle\_captures вместе выполняют действие хода. place\_stone размещает фишку на доске, а handle\_captures проверяет и удаляет захваченные группы фишек. Нет отдельного объекта "Ход", а есть функции, выполняющие его обработку.

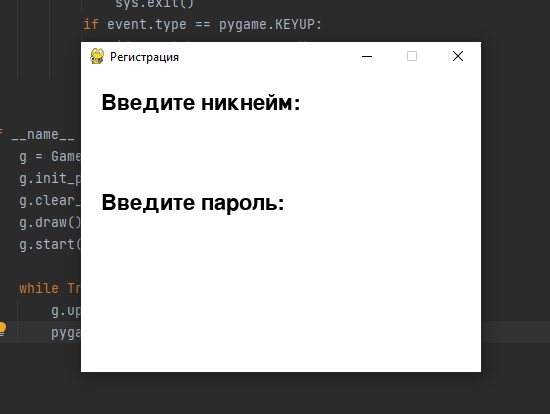
3. Позиция (board, prisoners): Состояние игры определяется двумя структурами данных: board (расположение фишек) и prisoners (количество взятых фишек). Эти данные вместе представляют текущую "позицию" на доске. Нет отдельного объекта "Позиция", эта информация распределена по нескольким переменным.

Описание работы приложения:

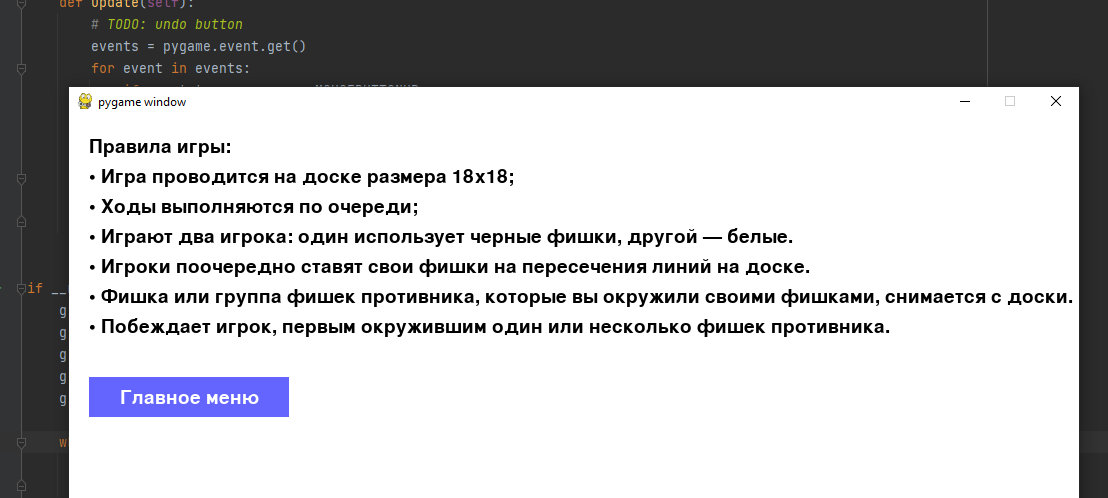
1.Изначально пользователь видит главное меню приложения в котором можно выбрать хочет ли он играть без регистрации или зарегистрироваться или посмотреть правила игры.

****

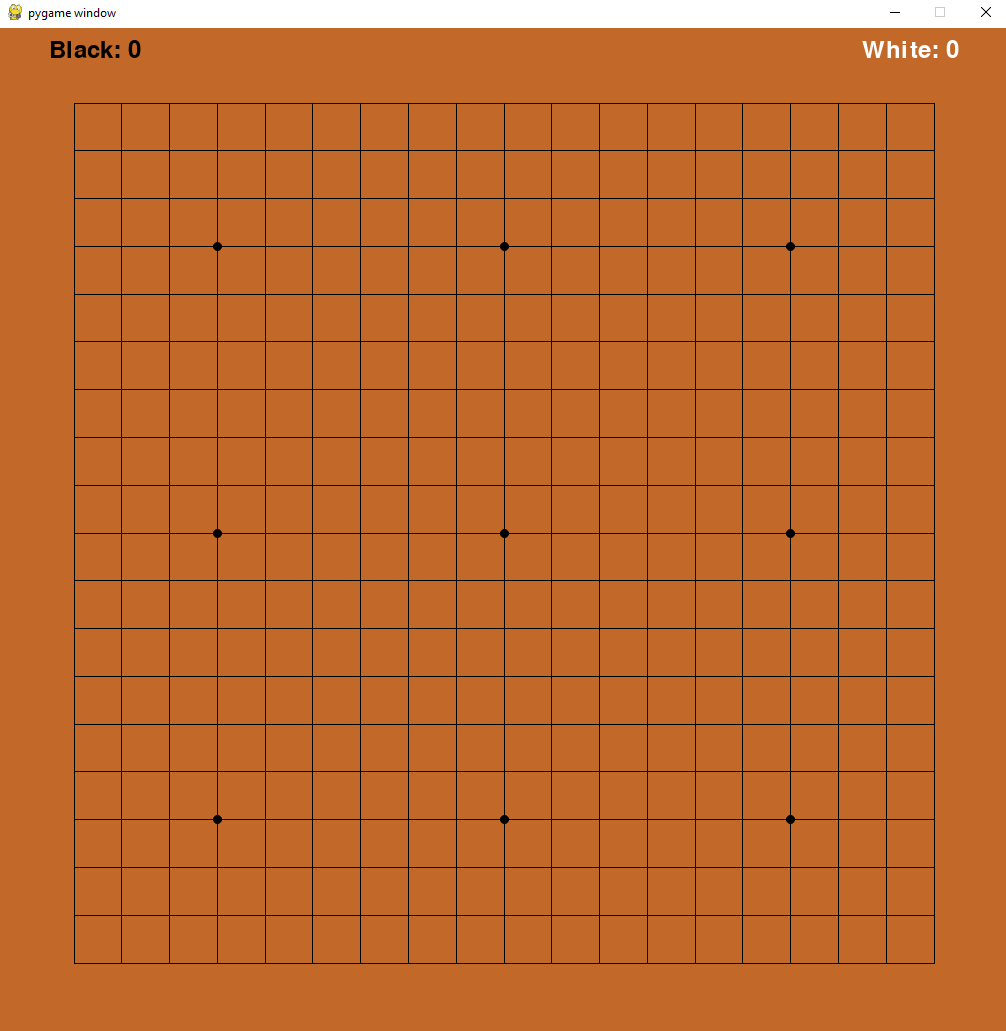
2.При нажатии на кнопку «Зарегистрироваться» открывается окно регистрации, при нажатии на кнопку «Играть без регистрации» открывается окно основной игры между двумя пользователями.



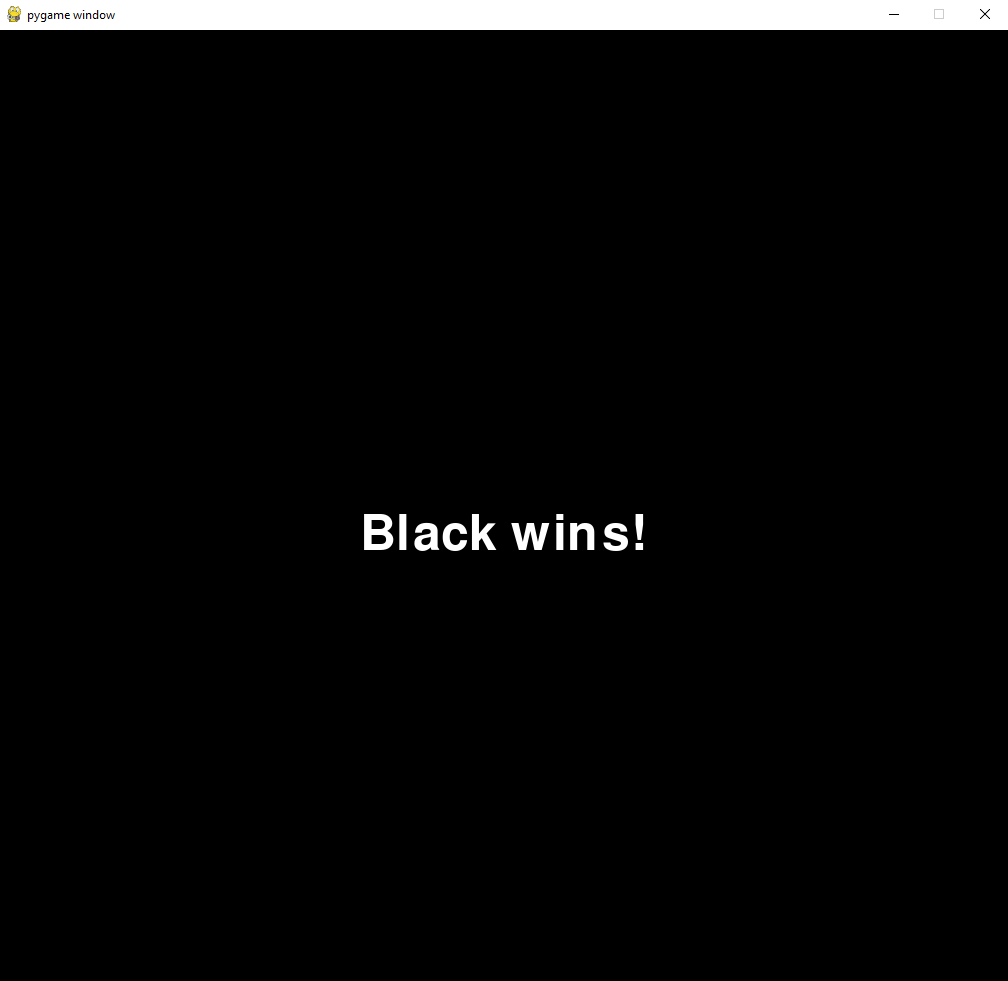
3.При нажатии на кнопку «Правила игры» открывается окно, в котором отображаются основные правила и инструкции по игре, позволяя игроку ознакомиться с игровым процессом, целями и условиями победы.



4.После окончания регистрации или нажатия на кнопку «Играть без регистрации», открывается игровое окно.



5.После окончания игры (2 случая) открывается соответствующее меню в котором можно закрыть приложение.





**2.2 Особенности реализации приложения**

**Структуры данных**

**1) Доска игры**

Для представления игровой доски используется двумерный массив NumPy:



Каждая ячейка массива может принимать значения:

* 0 — пустая клетка,
* 1 — черная фишка,
* 2 — белая фишка.

Выбор NumPy для хранения доски игры обусловлен его эффективной работой с многомерными массивами и возможностью быстрого выполнения операций над ними. Альтернативой мог бы стать список списков, однако NumPy предлагает более высокую производительность и удобные функции для работы с массивами.

**2) Хранение захваченных камней**

Для учета захваченных фишек используется словарь:

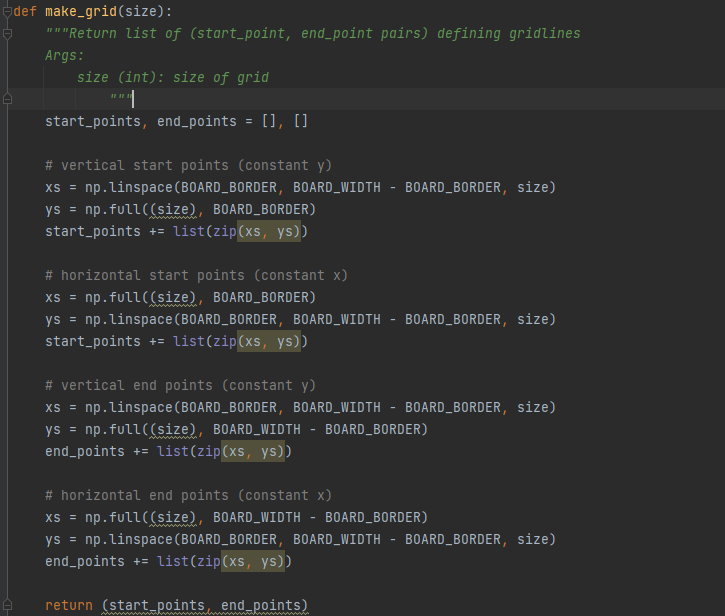


Словарь позволяет удобно отслеживать количество захваченных фишек для каждого игрока, где ключами являются цвета фишек (например, "black" и "white"), а значениями — количество захваченных фишек. Использование defaultdict позволяет избежать необходимости предварительной инициализации значений.

**Алгоритмы и функции**

**1) Генерация сетки**

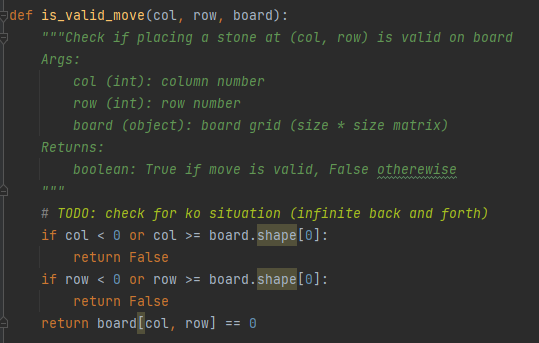
Функция make\_grid(size) генерирует координаты для линий сетки на доске:



Эта функция возвращает список начальных и конечных точек линий, что позволяет отрисовывать сетку на экране. Использование NumPy для вычисления координат позволяет эффективно обрабатывать данные и избегать циклов.

**2) Проверка легальности ходов**

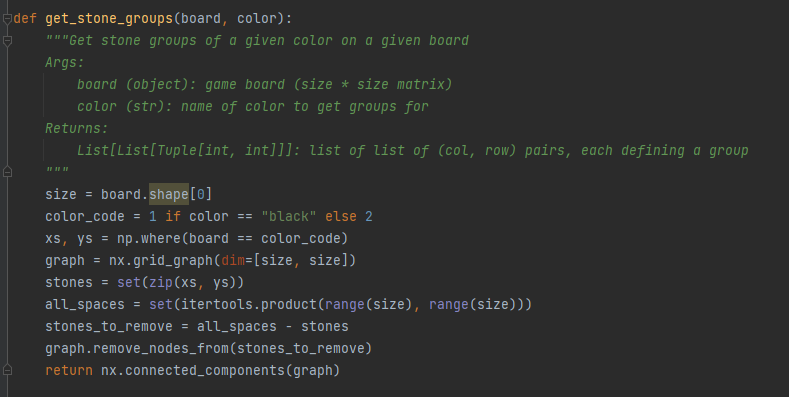
Функция is\_valid\_move(col, row, board) проверяет, допустим ли ход в заданной позиции:



Эта функция учитывает границы доски и проверяет, свободна ли клетка.

**3) Определение групп камней**

Функция get\_stone\_groups(board, color) определяет группы камней одного цвета на доске:



Использование библиотеки networkx позволяет строить граф из камней и находить связанные компоненты, что эффективно для определения групп камней и их свободных клеток.

**3. Обращение к программе**

Функции вне класса Game:

1. make\_grid(size):

Назначение: Создает список кортежей, определяющих начальные и конечные точки линий сетки для игрового поля. Используется для отрисовки сетки.

Назначение с аргументами и возвратом: Принимает size (целое число, размер сетки) и возвращает кортеж из двух списков кортежей координат: (start\_points, end\_points).

2. xy\_to\_colrow(x, y, size):

Назначение: Преобразует экранные координаты x, y (позицию клика мыши) в координаты столбца и строки на игровом поле.

Назначение с аргументами и возвратом: Принимает x (вещественное число, x-координата), y (вещественное число, y-координата), size (целое число, размер сетки) и возвращает кортеж из двух целых чисел: (col, row).

3. colrow\_to\_xy(col, row, size):

Назначение: Преобразует координаты столбца и строки на игровом поле в экранные координаты x, y.

Назначение с аргументами и возвратом: Принимает col (целое число, номер столбца), row (целое число, номер строки), size (целое число, размер сетки) и возвращает кортеж из двух целых чисел: (x, y).

4. has\_no\_liberties(board, group):

Назначение: Проверяет, имеет ли группа фишек на доске хотя бы одну свободную (пустую) соседнюю клетку (свободу). Используется для определения, захвачены ли фишки противника.

Назначение с аргументами и возвратом: Принимает board (массив numpy, игровая доска), group (список кортежей координат, группа фишек) и возвращает логическое значение: True, если у группы нет свобод, False в противном случае.

5. get\_stone\_groups(board, color):

Назначение: Находит и возвращает все группы фишек заданного цвета на игровой доске. Использует библиотеку networkx для определения связных групп.

Назначение с аргументами и возвратом: Принимает board (массив numpy, игровая доска), color (строка, цвет фишек) и возвращает генератор списков кортежей координат фишек.

6. is\_valid\_move(col, row, board):

Назначение: Проверяет, является ли ход в клетку с координатами и на доске допустимым. Используется для проверки правил игры (можно ли ставить фишку в данную клетку).

Назначение с аргументами и возвратом: Принимает col (целое число, номер столбца), row (целое число, номер строки), board (игровая доска) и возвращает логическое значение True, если ход допустимый, и False иначе.

7. draw\_menu(screen, font):

Назначение: Отрисовывает главное меню на экране. Создаёт и отрисовывает кнопки "Играть без регистрации", "Зарегистрироваться" и

"Правила игры".

Назначение с аргументами и возвратом: Принимает screen (объект экрана Pygame), font (объект шрифта Pygame) и возвращает кортеж из двух объектов Rect: (play\_button\_rect, register\_button\_rect), представляющие прямоугольные области кнопок.

8. handle\_menu\_click(pos, play\_button\_rect, register\_button\_rect):

Назначение: Определяет, какая кнопка меню была нажата пользователем.

Назначение с аргументами и возвратом: Принимает pos (кортеж, позиция клика мыши), play\_button\_rect (объект Rect, прямоугольная область кнопки "Играть без регистрации"), register\_button\_rect (объект Rect, прямоугольная область кнопки "Зарегистрироваться/Войти") и возвращает строку "play", "register", или None, если клик не попал ни на одну из кнопок.

9. registration(game):

Назначение: Отображает окно регистрации (текстовые поля ввода никнейма и пароля, а также кнопку "Зарегистрироваться").

Назначение с аргументами и возвратом: Принимает game (экземпляр класса Game) и возвращает логическое значение: True если регистрация прошла успешно, или False если окно регистрации было закрыто без регистрации.

10. show\_success\_message(game):

Назначение: Отображает экран с сообщением об успешной регистрации и кнопкой "ОК" для возврата к игровому процессу.

Назначение с аргументами и возвратом: Принимает game (экземпляр класса Game). Не возвращает значения (возвращается в игровой цикл после нажатия "ОК").

11. show\_registration\_error():

Назначение: Отображает экран с сообщением об ошибке, если при регистрации не были введены данные (никнейм и пароль), и кнопкой "ОК" для возврата к окну регистрации.

Назначение с аргументами и возвратом: Не принимает аргументов и не возвращает значений (возвращается в окно регистрации после нажатия "ОК").

12. show\_rules(screen):

Назначение: Отображает экран с правилами игры, кнопкой "Главное меню", для возврата в меню.

Назначение с аргументами и возвратом: Принимает screen (объект экрана Pygame) и не возвращает значения (возвращается в меню после нажатия "Главное меню").

Методы класса Game:

1. init(self, size):

Назначение: Конструктор класса Game. Инициализирует объект игры: создает игровую доску, задает ее размер, устанавливает, чья очередь хода, создает словарь для подсчета захваченных фишек, создает список кортежей для отрисовки сетки.

Назначение с аргументами и возвратом: Принимает self (ссылка на экземпляр класса Game), size (целое число, размер доски). Не возвращает значения.

2. start(self):

Назначение: Запускает главное меню игры, инициализирует окно, ожидает нажатия на кнопки меню.

Назначение с аргументами и возвратом: Принимает self (ссылка на экземпляр класса Game). Не возвращает значения.

3. play\_game(self, fullscreen=False):

Назначение: Запускает игровой процесс: инициализирует Pygame, очищает экран, отрисовывает доску, запускает игровой цикл.

Назначение с аргументами и возвратом: Принимает self (ссылка на экземпляр класса Game), fullscreen (логическое значение, определяет полноэкранный режим). Не возвращает значения.

4. init\_pygame(self, fullscreen=False):

Назначение: Инициализирует библиотеку Pygame, создает окно игры (с учетом полноэкранного режима), настраивает шрифт, загружает звук хода.

Назначение с аргументами и возвратом: Принимает self (ссылка на экземпляр класса Game), fullscreen (логическое значение, определяет полноэкранный режим). Не возвращает значения.

5. clear\_screen(self):

Назначение: Очищает экран, заливая его фоновым цветом, и отрисовывает сетку и направляющие точки на игровом поле.

Назначение с аргументами и возвратом: Принимает self (ссылка на экземпляр класса Game). Не возвращает значения.

6. pass\_move(self):

Назначение: Переключает ход между игроками.

Назначение с аргументами и возвратом: Принимает self (ссылка на экземпляр класса Game). Не возвращает значения.

7. handle\_click(self):

Назначение: Обрабатывает клик мыши на игровом поле: определяет координаты клика, проверяет допустимость хода, размещает фишку, воспроизводит звук, проверяет выигрыш и переключает ход.

Назначение с аргументами и возвратом: Принимает self (ссылка на экземпляр класса Game). Не возвращает значения.

8. is\_valid\_move(self, col, row):

Назначение: Проверяет, является ли ход в клетку с определенными координатами.

Назначение с аргументами и возвратом: Принимает self (ссылка на экземпляр класса Game), col (номер столбца), row (номер строки) и возвращает логическое значение: True если ход допустимый, False в противном случае.

9. place\_stone(self, col, row):

Назначение: Размещает фишку на доске в указанную позицию и вызывает метод обработки захвата фишек.

Назначение с аргументами и возвратом: Принимает self (ссылка на экземпляр класса Game), col (целое число, номер столбца), row (целое число, номер строки). Не возвращает значения.

10. handle\_captures(self):

Назначение: Обрабатывает захват фишек противника, вызывая метод capture\_stones и обновляя счетчик захваченных фишек.

Назначение с аргументами и возвратом: Принимает self (ссылка на экземпляр класса Game). Не возвращает значения.

11. capture\_stones(self, color):

Назначение: Захватывает фишки противника указанного цвета, если они не имеют свобод.

Назначение с аргументами и возвратом: Принимает self (ссылка на экземпляр класса Game), color (строка, цвет фишек, которые нужно захватить) и возвращает целое число, количество захваченных фишек.

12. check\_winner(self):

Назначение: Проверяет, достиг ли кто-либо из игроков выигрышного количества захваченных фишек.

Назначение с аргументами и возвратом: Принимает self (ссылка на экземпляр класса Game) и возвращает строку "Black" или "White" в случае выигрыша одного из игроков, или None, если никто не выиграл.

13. show\_winner\_message(self, winner):

Назначение: Отображает сообщение о победе на экране.

Назначение с аргументами и возвратом: Принимает self (ссылка на экземпляр класса Game), winner (строка, имя победившего игрока "Black" или "White"). Не возвращает значения.

14. draw(self):

Назначение: Отрисовывает текущее состояние игры: сетку, фишки на доске, счёт игроков.

Назначение с аргументами и возвратом: Принимает self (ссылка на экземпляр класса Game). Не возвращает значения.

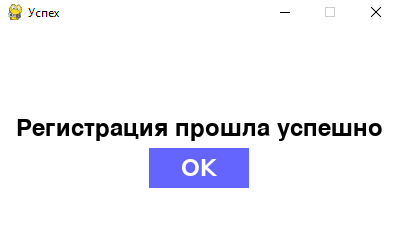
15. update(self):

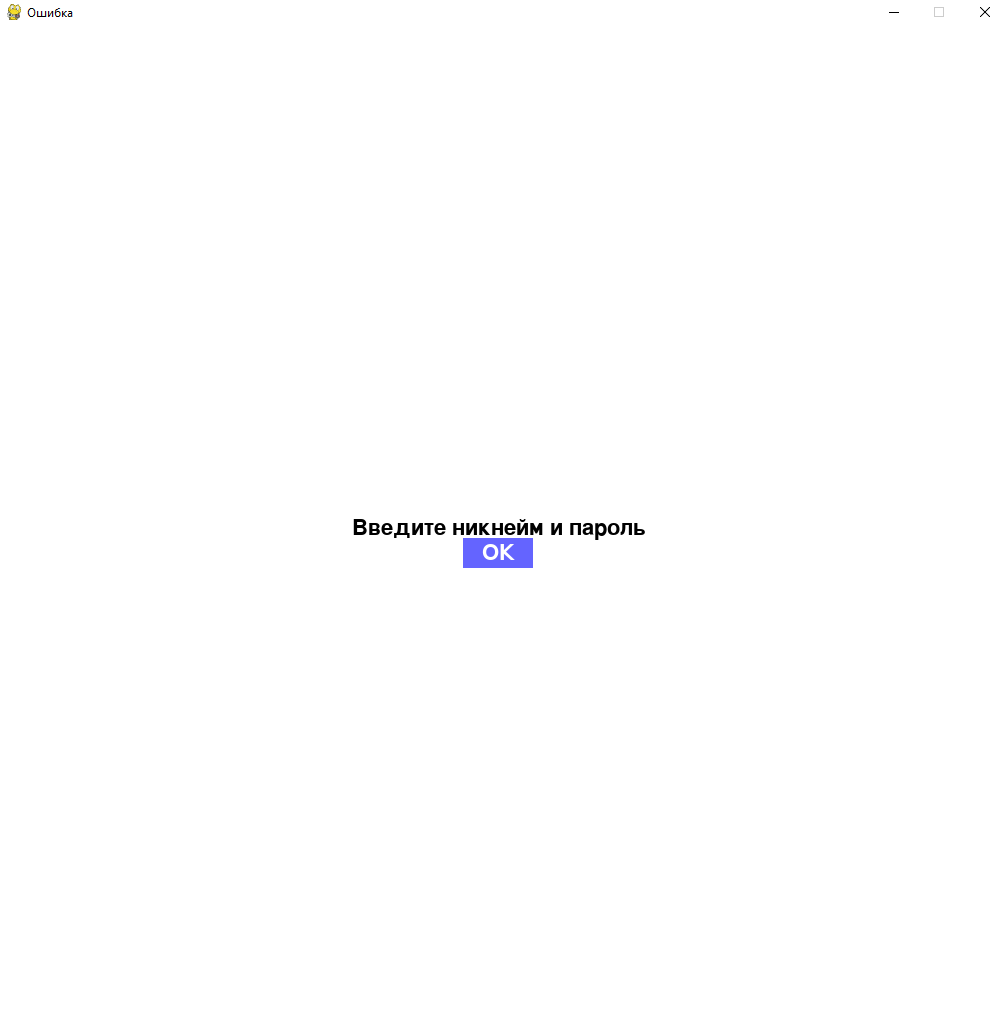
Назначение: Обрабатывает события Pygame (нажатия мыши, клавиш, закрытие окна), вызывает соответствующие методы, чтобы обновить игру.

Назначение с аргументами и возвратом: Принимает self (ссылка на экземпляр класса Game). Не возвращает значения.

**4. Сообщения**

Сообщения выдаваемые по результатам контроля корректности ввода/вывода.





**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 «Атари-Го»

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

# Тестовая документация

**Р.02069337. 23/694-24**

Листов 7

Исполнитель:

студент гр. ИСТбд-22

Семин Артём Рудольфович

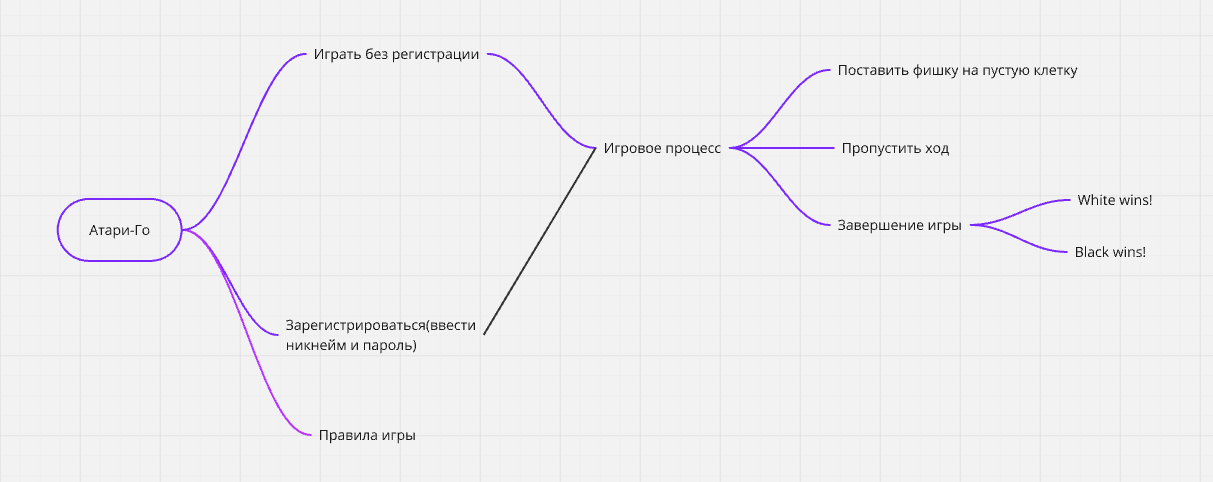
« » 2024 г.

2024

**Введение.**

Атари-Го — это одна из популярных вариаций шашечной игры, отличающаяся уникальными правилами и стратегиями. Важной частью разработки любой программы, симулирующей эту игру, является тестирование, которое гарантирует корректность реализации игровых правил, стабильность функционала и удобство использования приложения.

Данная тестовая документация предназначена для описания подхода к тестированию программы, обеспечивающей игру в Атари-Го. В ней детализированы все аспекты проверки: от базовой проверки правил игры до тестирования сложных стратегических ситуаций, пользовательского интерфейса и производительности.



Mind-map “Атари-Го”

**1.Области тестирования.**

В рамках тестирования разработанного приложения были выделены ключевые области такие как:

**2.1 Проверка соответствию правил.**

Убедиться, что все игровые механики реализованы правильно, включая расставление фишек на доске, пропуск хода, захват фишек и условия окончания игры.

**2.2 Тестирование интерфейса.**

* Тестирование отображения доски.

**2.3 Работа с пользовательским вводом.**

* Проверка на обработку корректного и некорректного ввода:
  + Попытка сделать ход в недопустимую клетку.
* Проверка реакции программы на экстремальные сценарии, например, многократное нажатие.

**3. Методы тестирования.**

**3.1 Модульное тестирование.**

На данном этапе тестирования проверяется корректность работы отдельных, изолированных модулей (или функций) кода. Цель — убедиться, что каждый модуль выполняет свою задачу в соответствии со спецификацией. В “Атари-Го” модульным тестирование может являться:

* Обработка кликов на игровой доске: Проверяется, что при клике на конкретную клетку доски игра корректно распознает координаты клетки и обрабатывает ход, корректно обрабатывая ситуации, когда клетка занята своей фишкой или противника.

**3.2 Общее тестирование.**

На этом уровне проверяется работа всей системы в целом, включая пользовательский интерфейс и выполнение сложных игровых ситуаций. Целью является - убедиться, что игра работает как законченное программное обеспечение и соответствует всем требованиям.

* Проверка полного игрового сценария: Воспроизведение нескольких игровых сценарий (победа, поражение).
* Тестирование графического интерфейса: Проверка удобства и отзывчивости интерфейса. Включает в себя проверку навигации по меню, отзывчивость кнопок, ясность сообщений об ошибках.

**3.3 Тестирование производительности.**

Проверка устойчивости приложения к высоким нагрузкам и длительному использованию. Цель — выявление узких мест и потенциальных ошибок, которые могут возникнуть при интенсивной эксплуатации.

* Оценка использования процессора и памяти программы во время игры, чтобы убедиться, что игра не потребляет слишком много системных ресурсов, особенно на слабых устройствах.
* Проверка работы игры при длительном игровом процессе, например, на протяжении нескольких часов.

**4. Проведенные тесты.**

В таблице представлены результаты тестирования.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Сценарий тестирования** | **Описание** | **Ожидаемый результат** | **Фактический результат** | **Статус** |
| Успешная регистрация | Ввод нового никнейма и пароля. | Пользователь зарегистрирован, происходит вход в игровое окно. | Работает как ожидается | Выполнено |
| Неуспешная регистрация | Попытка регистрации с пустыми полями никнейма и/или пароля. | Вывод сообщения об ошибке "Введите никнейм и пароль", регистрация не выполнена. | Работает как ожидается | Выполнено |
| Тест простого хода | Поставить фишку на корректную клетку. | Фишка ставится на пустую клетку, ход передается следующему игроку. | Работает как ожидается | Выполнено |
| Тест некорректного хода | Попытка сделать ход на занятую клетку или за пределы доски. | Программа отклоняет ход,  необходимо правильно сделать ход. | Работает как ожидается | Выполнено |
| Тест пропуска хода | Необходимо нажать на клавишу P, для того чтобы передать ход противнику. | Пользователь пропускает и передает ход противнику. | Работает как ожидается | Выполнено |
| Тест взятия фишки | Выполнение хода, в котором происходит процесс окружение фишки или группы фишек противника. | Фишка противника удаляется с доски. | Работает как ожидается | Выполнено |
| Обновление графического интерфейса | Проверить, что после каждого хода интерфейс отображает актуальное состояние доски и очередность хода. | Графический интерфейс обновляется после каждого действия игрока. | Работает как ожидается | Выполнено |
| Интуитивность интерфейса | Дать игру новому пользователю и проверить, понимает ли он, как использовать интерфейс без дополнительных объяснений. | Пользователь понимает правила игры и способ взаимодействия с доской. | Работает как ожидается | Выполнено |
| Тест звуков внутри программы | Пользователь нажимает на пустую клетку, чтобы сделать ход и происходит определенный звук. | Звук производится при нажатии на пустую клетку. | Работает как ожидается | Выполнено |

**5. Выводы.**

На основе проведенных тестов сделаны следующие выводы:

* Программа успешно прошла все тесты и работает корректно.

# Полный код программы:

import pygame  
import numpy as np  
import itertools  
import sys  
import networkx as nx  
import collections  
from pygame import gfxdraw  
  
  
  
# Game constants  
BOARD\_BROWN = (194, 105, 42)  
BOARD\_WIDTH = 1010  
BOARD\_BORDER = 75  
STONE\_RADIUS = 22  
WHITE = (255, 255, 255)  
BLACK = (0, 0, 0)  
TURN\_POS = (BOARD\_BORDER, 20)  
SCORE\_POS = (BOARD\_BORDER, BOARD\_WIDTH - BOARD\_BORDER + 30)  
DOT\_RADIUS = 4  
WINNING\_SCORE = 1  
  
  
def make\_grid(size):  
 *"""Return list of (start\_point, end\_point pairs) defining gridlines  
 Args:  
 size (int): size of grid  
 Returns:  
 Tuple[List[Tuple[float, float]]]: start and end points for gridlines  
 """* start\_points, end\_points = [], []  
  
 # vertical start points (constant y)  
 xs = np.linspace(BOARD\_BORDER, BOARD\_WIDTH - BOARD\_BORDER, size)  
 ys = np.full((size), BOARD\_BORDER)  
 start\_points += list(zip(xs, ys))  
  
 # horizontal start points (constant x)  
 xs = np.full((size), BOARD\_BORDER)  
 ys = np.linspace(BOARD\_BORDER, BOARD\_WIDTH - BOARD\_BORDER, size)  
 start\_points += list(zip(xs, ys))  
  
 # vertical end points (constant y)  
 xs = np.linspace(BOARD\_BORDER, BOARD\_WIDTH - BOARD\_BORDER, size)  
 ys = np.full((size), BOARD\_WIDTH - BOARD\_BORDER)  
 end\_points += list(zip(xs, ys))  
  
 # horizontal end points (constant x)  
 xs = np.full((size), BOARD\_WIDTH - BOARD\_BORDER)  
 ys = np.linspace(BOARD\_BORDER, BOARD\_WIDTH - BOARD\_BORDER, size)  
 end\_points += list(zip(xs, ys))  
  
 return (start\_points, end\_points)  
  
  
def xy\_to\_colrow(x, y, size):  
 *"""Convert x,y coordinates to column and row number  
 Args:  
 x (float): x position  
 y (float): y position  
 size (int): size of grid  
 Returns:  
 Tuple[int, int]: column and row numbers of intersection  
 """* inc = (BOARD\_WIDTH - 2 \* BOARD\_BORDER) / (size - 1)  
 x\_dist = x - BOARD\_BORDER  
 y\_dist = y - BOARD\_BORDER  
 col = int(round(x\_dist / inc))  
 row = int(round(y\_dist / inc))  
 return col, row  
  
  
def colrow\_to\_xy(col, row, size):  
 *"""Convert column and row numbers to x,y coordinates  
 Args:  
 col (int): column number (horizontal position)  
 row (int): row number (vertical position)  
 size (int): size of grid  
 Returns:  
 Tuple[float, float]: x,y coordinates of intersection  
 """* inc = (BOARD\_WIDTH - 2 \* BOARD\_BORDER) / (size - 1)  
 x = int(BOARD\_BORDER + col \* inc)  
 y = int(BOARD\_BORDER + row \* inc)  
 return x, y  
  
  
def has\_no\_liberties(board, group):  
 *"""Check if a stone group has any liberties on a given board.  
 Args:  
 board (object): game board (size \* size matrix)  
 group (List[Tuple[int, int]]): list of (col,row) pairs defining a stone group  
 Returns:  
 [boolean]: True if group has any liberties, False otherwise  
 """* for x, y in group:  
 if x > 0 and board[x - 1, y] == 0:  
 return False  
 if y > 0 and board[x, y - 1] == 0:  
 return False  
 if x < board.shape[0] - 1 and board[x + 1, y] == 0:  
 return False  
 if y < board.shape[0] - 1 and board[x, y + 1] == 0:  
 return False  
 return True  
  
  
def get\_stone\_groups(board, color):  
 *"""Get stone groups of a given color on a given board  
 Args:  
 board (object): game board (size \* size matrix)  
 color (str): name of color to get groups for  
 Returns:  
 List[List[Tuple[int, int]]]: list of list of (col, row) pairs, each defining a group  
 """* size = board.shape[0]  
 color\_code = 1 if color == "black" else 2  
 xs, ys = np.where(board == color\_code)  
 graph = nx.grid\_graph(dim=[size, size])  
 stones = set(zip(xs, ys))  
 all\_spaces = set(itertools.product(range(size), range(size)))  
 stones\_to\_remove = all\_spaces - stones  
 graph.remove\_nodes\_from(stones\_to\_remove)  
 return nx.connected\_components(graph)  
  
  
def is\_valid\_move(col, row, board):  
 *"""Check if placing a stone at (col, row) is valid on board  
 Args:  
 col (int): column number  
 row (int): row number  
 board (object): board grid (size \* size matrix)  
 Returns:  
 boolean: True if move is valid, False otherewise  
 """* # *TODO: check for ko situation (infinite back and forth)* if col < 0 or col >= board.shape[0]:  
 return False  
 if row < 0 or row >= board.shape[0]:  
 return False  
 return board[col, row] == 0  
  
def draw\_menu(screen, font):  
 *"""Draws the main menu on the screen."""* screen.fill((0, 0, 0)) # Black background  
  
 screen\_width = screen.get\_width()  
 screen\_height = screen.get\_height()  
  
 button\_width\_ratio = 0.4  
 button\_height\_ratio = 0.1  
 button\_padding\_ratio = 0.1  
  
 button\_width = int(screen\_width \* button\_width\_ratio)  
 button\_height = int(screen\_height \* button\_height\_ratio)  
 button\_padding = int(screen\_height \* button\_padding\_ratio)  
 button\_color = (100, 100, 255)  
 text\_color = (255, 255, 255)  
  
 # Play button  
 play\_button\_y = (screen\_height - 3 \* button\_height - 2 \* button\_padding) // 2 + button\_height  
 play\_button\_rect = pygame.Rect((screen\_width - button\_width) // 2, play\_button\_y, button\_width, button\_height)  
 pygame.draw.rect(screen, button\_color, play\_button\_rect)  
 play\_text = font.render("Играть без регистрации", True, text\_color)  
 play\_text\_rect = play\_text.get\_rect(center=play\_button\_rect.center)  
 screen.blit(play\_text, play\_text\_rect)  
  
 # Register button  
 register\_button\_y = play\_button\_y + button\_height + button\_padding  
 register\_button\_rect = pygame.Rect((screen\_width - button\_width) // 2, register\_button\_y, button\_width, button\_height)  
 pygame.draw.rect(screen, button\_color, register\_button\_rect)  
 register\_text = font.render("Зарегистрироваться", True, text\_color)  
 register\_text\_rect = register\_text.get\_rect(center=register\_button\_rect.center)  
 screen.blit(register\_text, register\_text\_rect)  
  
 # Rules button  
 rules\_button\_y = register\_button\_y + button\_height + button\_padding  
 rules\_button\_rect = pygame.Rect((screen\_width - button\_width) // 2, rules\_button\_y, button\_width, button\_height)  
 pygame.draw.rect(screen, button\_color, rules\_button\_rect)  
 rules\_text = font.render("Правила игры", True, text\_color)  
 rules\_text\_rect = rules\_text.get\_rect(center=rules\_button\_rect.center)  
 screen.blit(rules\_text, rules\_text\_rect)  
  
 pygame.display.flip()  
 return play\_button\_rect, register\_button\_rect, rules\_button\_rect  
  
def handle\_menu\_click(pos, play\_button\_rect, register\_button\_rect, rules\_button\_rect):  
 if play\_button\_rect.collidepoint(pos):  
 return "play"  
 elif register\_button\_rect.collidepoint(pos):  
 return "register"  
 elif rules\_button\_rect.collidepoint(pos):  
 return "rules"  
 else:  
 return None  
  
def show\_rules(screen):  
 *"""Displays the game rules on the screen."""* screen.fill((255, 255, 255)) # White background  
 font = pygame.font.Font(None, 28)  
  
 rules = [  
 "Правила игры:",  
 "• Игра проводится на доске размера 18x18;",  
 "• Ходы выполняются по очереди;",  
 "• Играют два игрока: один использует черные фишки, другой — белые.",  
 "• Игроки поочередно ставят свои фишки на пересечения линий на доске.",  
 "• Фишка или группа фишек противника, которые вы окружили своими фишками, снимается с доски.",  
 "• Побеждает игрок, первым окружившим один или несколько фишек противника."  
 ]  
  
 for i, line in enumerate(rules):  
 text = font.render(line, True, (0, 0, 0)) # Black text  
 screen.blit(text, (20, 20 + i \* 30))  
  
 # Кнопка "Главное меню"  
 back\_button\_rect = pygame.Rect(20, 20 + len(rules) \* 30 + 30, 200, 40)  
 pygame.draw.rect(screen, (100, 100, 255), back\_button\_rect) # Button color  
 back\_text = font.render("Главное меню", True, (255, 255, 255)) # White text  
 back\_text\_rect = back\_text.get\_rect(center=back\_button\_rect.center)  
 screen.blit(back\_text, back\_text\_rect)  
  
 pygame.display.flip()  
  
 # Wait for user to return to menu  
 waiting = True  
 while waiting:  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.quit()  
 sys.exit()  
 if event.type == pygame.MOUSEBUTTONUP:  
 if back\_button\_rect.collidepoint(event.pos): # Check if back button is clicked  
 waiting = False  
 if event.type == pygame.KEYDOWN:  
 if event.key == pygame.K\_RETURN: # Press Enter to return  
 waiting = False  
  
def registration(game): # Pass Game instance  
 reg\_screen = pygame.display.set\_mode((400, 300))  
 pygame.display.set\_caption("Регистрация")  
 font = pygame.font.Font(None, 32)  
 username = ''  
 password = ''  
 input\_active = [True, False] # [username\_input\_active, password\_input\_active]  
  
 running = True  
 while running:  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 return False # Вернуться в главное меню, если окно закрыто  
 if event.type == pygame.KEYDOWN:  
 if input\_active[0]: # Ввод никнейма  
 if event.key == pygame.K\_RETURN:  
 input\_active[0] = False  
 input\_active[1] = True  
 elif event.key == pygame.K\_BACKSPACE:  
 username = username[:-1]  
 else:  
 username += event.unicode  
  
 elif input\_active[1]: # Ввод пароля  
 if event.key == pygame.K\_RETURN:  
 if username and password: # Проверка на пустые поля  
 show\_success\_message(game)  
 running = False # Выход из цикла после успешной регистрации  
 elif event.key == pygame.K\_BACKSPACE:  
 password = password[:-1]  
 else:  
 password += event.unicode  
  
 reg\_screen.fill((255, 255, 255)) # White background  
 username\_text = font.render('Введите никнейм:', True, (0, 0, 0)) # Black text  
 password\_text = font.render('Введите пароль:', True, (0, 0, 0))  
  
 reg\_screen.blit(username\_text, (20, 20))  
 reg\_screen.blit(font.render(username, True, (0, 0, 0)), (20, 60))  
  
 reg\_screen.blit(password\_text, (20, 120))  
 reg\_screen.blit(font.render('\*' \* len(password), True, (0, 0, 0)), (20, 160))  
 pygame.display.flip()  
  
 pygame.display.quit() # Quit ONLY the registration display  
 return True  
  
def show\_success\_message(game):  
 *"""Показывает сообщение об успешной регистрации и кнопку ОК."""* success\_screen = pygame.display.set\_mode((400, 200))  
 pygame.display.set\_caption("Успех")  
 font = pygame.font.Font(None, 36)  
 text = font.render("Регистрация прошла успешно", True, (0, 0, 0))  
 ok\_button\_rect = pygame.Rect(150, 120, 100, 40) # Кнопка ОК  
  
 while True:  
 success\_screen.fill(WHITE)  
 text\_rect = text.get\_rect(center=(200, 100))  
 success\_screen.blit(text, text\_rect)  
  
 # Рисуем кнопку ОК  
 pygame.draw.rect(success\_screen, (100, 100, 255), ok\_button\_rect)  
 ok\_text = font.render("ОК", True, (255, 255, 255))  
 ok\_text\_rect = ok\_text.get\_rect(center=ok\_button\_rect.center)  
 success\_screen.blit(ok\_text, ok\_text\_rect)  
  
 pygame.display.flip()  
  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.quit()  
 sys.exit()  
 if event.type == pygame.MOUSEBUTTONUP:  
 if ok\_button\_rect.collidepoint(event.pos):  
 return # Возврат к игровому процессу  
  
def show\_registration\_error():  
 *"""Показывает экран с ошибкой, если не введены данные."""* error\_screen = pygame.display.set\_mode((1010, 1010))  
 pygame.display.set\_caption("Ошибка")  
 font = pygame.font.Font(None, 32)  
 text = font.render("Введите никнейм и пароль", True, (0, 0, 0))  
 ok\_button\_rect = pygame.Rect(465, 510, 70, 30) # Кнопка ОК  
  
 while True:  
 error\_screen.fill(WHITE)  
 text\_rect = text.get\_rect(center=(500, 500))  
 error\_screen.blit(text, text\_rect)  
  
 # Рисуем кнопку ОК  
 pygame.draw.rect(error\_screen, (100, 100, 255), ok\_button\_rect)  
 ok\_text = font.render("ОК", True, (255, 255, 255))  
 ok\_text\_rect = ok\_text.get\_rect(center=ok\_button\_rect.center)  
 error\_screen.blit(ok\_text, ok\_text\_rect)  
  
 pygame.display.flip()  
  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.quit()  
 sys.exit()  
 if event.type == pygame.MOUSEBUTTONUP:  
 if ok\_button\_rect.collidepoint(event.pos):  
 return # Возврат к регистрации  
  
class Game:  
 def \_\_init\_\_(self, size):  
 self.board = np.zeros((size, size))  
 self.size = size  
 self.black\_turn = True  
 self.prisoners = collections.defaultdict(int)  
 self.start\_points, self.end\_points = make\_grid(self.size)  
  
 def start(self):  
 self.init\_pygame()  
 clock = pygame.time.Clock()  
 font = pygame.font.SysFont("arial", 36)  
  
 while True:  
 play\_button\_rect, register\_button\_rect, rules\_button\_rect = draw\_menu(self.screen, font)  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.quit()  
 sys.exit()  
 elif event.type == pygame.MOUSEBUTTONUP:  
 action = handle\_menu\_click(event.pos, play\_button\_rect, register\_button\_rect, rules\_button\_rect)  
 if action == "play":  
 self.play\_game()  
 return  
 elif action == "register":  
 if not registration(self): # Pass game instance  
 show\_registration\_error() # Показать ошибку, если регистрация была закрыта  
 continue  
 self.play\_game()  
 return  
 elif action == "rules":  
 show\_rules(self.screen) # Show game rules  
  
 clock.tick(30)  
  
 def play\_game(self, fullscreen=False):  
 if fullscreen:  
 self.init\_pygame(fullscreen=True)  
 else:  
 self.init\_pygame()  
  
 self.clear\_screen()  
 self.draw()  
 while True:  
 self.update()  
 pygame.time.wait(100)  
  
  
 def init\_pygame(self):  
 pygame.init()  
 screen = pygame.display.set\_mode((BOARD\_WIDTH, BOARD\_WIDTH))  
 self.screen = screen  
 self.font = pygame.font.SysFont("arial", 30)  
 self.place\_sound = pygame.mixer.Sound("move\_fishki.wav")  
 pygame.mixer.init() # Инициализация модуля микширования  
  
 def clear\_screen(self):  
  
 # fill board and add gridlines  
 self.screen.fill(BOARD\_BROWN)  
 for start\_point, end\_point in zip(self.start\_points, self.end\_points):  
 pygame.draw.line(self.screen, BLACK, start\_point, end\_point)  
  
 # add guide dots  
 guide\_dots = [3, self.size // 2, self.size - 4]  
 for col, row in itertools.product(guide\_dots, guide\_dots):  
 x, y = colrow\_to\_xy(col, row, self.size)  
 gfxdraw.aacircle(self.screen, x, y, DOT\_RADIUS, BLACK)  
 gfxdraw.filled\_circle(self.screen, x, y, DOT\_RADIUS, BLACK)  
  
 pygame.display.flip()  
  
 def pass\_move(self):  
 self.black\_turn = not self.black\_turn  
 self.draw()  
  
 def handle\_click(self):  
 x, y = pygame.mouse.get\_pos()  
  
 col, row = xy\_to\_colrow(x, y, self.size)  
  
 if not self.is\_valid\_move(col, row):  
 return  
  
 self.place\_stone(col, row)  
 self.place\_sound.play()  
 winner = self.check\_winner()  
 if winner:  
 self.show\_winner\_message(winner)  
  
 self.black\_turn = not self.black\_turn  
 self.draw()  
  
  
 def is\_valid\_move(self, col, row):  
 *"""Check if placing a stone at (col, row) is valid."""* if col < 0 or col >= self.board.shape[0] or row < 0 or row >= self.board.shape[0]:  
 return False  
 return self.board[col, row] == 0  
  
 def place\_stone(self, col, row):  
 *"""Places a stone on the board and handles captures."""* player = 1 if self.black\_turn else 2  
 self.board[col, row] = player  
 self.handle\_captures()  
  
  
 def handle\_captures(self):  
 *"""Handles captures for both players after each move."""* self\_color = "black" if self.black\_turn else "white"  
 other\_color = "white" if self.black\_turn else "black"  
  
 captured\_count = self.capture\_stones(other\_color)  
 self.prisoners[self\_color] += captured\_count  
  
  
 def capture\_stones(self, color):  
 *"""Captures stones of the specified color."""* color\_code = 1 if color == "black" else 2  
 captured\_count = 0  
 for group in list(get\_stone\_groups(self.board, color)):  
 if has\_no\_liberties(self.board, group):  
 for i, j in group:  
 self.board[i, j] = 0  
 captured\_count += 1  
 return captured\_count  
  
  
 def check\_winner(self):  
 *"""Checks if either player has reached the winning score."""* if self.prisoners['black'] >= WINNING\_SCORE:  
 return "Black"  
 elif self.prisoners['white'] >= WINNING\_SCORE:  
 return "White"  
 else:  
 return None  
  
 def show\_winner\_message(self, winner):  
 *"""Показывает сообщение о выигрыше."""* font = pygame.font.Font(None, 74)  
 text = font.render(f"{winner} wins!", True, (255, 255, 255))  
 text\_rect = text.get\_rect(center=(self.screen.get\_width() // 2, self.screen.get\_height() // 2))  
  
 # Заливка фона  
 self.screen.fill((0, 0, 0)) # Черный фон  
 self.screen.blit(text, text\_rect)  
 pygame.display.flip()  
  
 # Ждем, пока пользователь не закроет окно  
 waiting = True  
 while waiting:  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.quit()  
 sys.exit()  
 if event.type == pygame.KEYDOWN:  
 if event.key == pygame.K\_RETURN: # Нажмите Enter для выхода  
 waiting = False  
  
  
 def draw(self):  
 self.clear\_screen() # Clear the screen for redrawing  
  
 # Draw stones  
 for col, row in zip(\*np.where(self.board == 1)):  
 x, y = colrow\_to\_xy(col, row, self.size)  
 gfxdraw.aacircle(self.screen, x, y, STONE\_RADIUS, BLACK)  
 gfxdraw.filled\_circle(self.screen, x, y, STONE\_RADIUS, BLACK)  
 for col, row in zip(\*np.where(self.board == 2)):  
 x, y = colrow\_to\_xy(col, row, self.size)  
 gfxdraw.aacircle(self.screen, x, y, STONE\_RADIUS, WHITE)  
 gfxdraw.filled\_circle(self.screen, x, y, STONE\_RADIUS, WHITE)  
  
  
 # Display scores at the top  
 font = pygame.font.Font(None, 36) # Choose a suitable font and size  
 black\_score\_text = font.render(f"Black: {self.prisoners['black']}", True, BLACK)  
 white\_score\_text = font.render(f"White: {self.prisoners['white']}", True, WHITE)  
  
 # Position the score text near the top of the screen. Adjust these values as needed  
 screen\_width = self.screen.get\_width()  
 black\_score\_x = 50 # Adjust x-coordinate as needed  
 white\_score\_x = screen\_width - white\_score\_text.get\_width() - 50 # Right-align White score  
  
  
 self.screen.blit(black\_score\_text, (black\_score\_x, 10)) # Adjust y-coordinate as needed  
 self.screen.blit(white\_score\_text, (white\_score\_x, 10)) # Adjust y-coordinate as needed  
  
 pygame.display.flip()  
  
 def update(self):  
 # *TODO: undo button* events = pygame.event.get()  
 for event in events:  
 if event.type == pygame.MOUSEBUTTONUP:  
 self.handle\_click()  
 if event.type == pygame.QUIT:  
 sys.exit()  
 if event.type == pygame.KEYUP:  
 if event.key == pygame.K\_p:  
 self.pass\_move()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 g = Game(size=19)  
 g.init\_pygame()  
 g.clear\_screen()  
 g.draw()  
 g.start()  
  
 while True:  
 g.update()  
 pygame.time.wait(100)