Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерного проектирования

Кафедра инженерной психологии и эргономики

Дисциплина: Компьютерные системы и сети (КСИС)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему:

**Разработка сетевой компьютерной игры "Го"**

БГУИР КП 6-05-0612-01 ПЗ

Студент Бладыко И.Е.

Руководитель Болтак С.В.

Минск 2025

СОДЕРЖАНИЕ

[Введение 5](#_Toc85934173)

[1 Анализ предметной области 6](#_Toc85934174)

[1.1 Обзор аналогов 6](#_Toc85934175)

[1.2 Постановка задачи 8](#_Toc85934176)

[2 Проектирование программного средства 9](#_Toc85934177)

[2.1 Структура программы 9](#_Toc85934178)

[2.2 Проектирование интерфейса программного средства 9](#_Toc85934179)

[2.3 Проектирование функционала программного средства 12](#_Toc85934180)

[3 Разработка программного средства 16](#_Toc85934181)

[3.1 Навигация по сайтам 16](#_Toc85934182)

[3.2 Реализация вкладок 17](#_Toc85934183)

[3.3 Работа с закладками веб-браузера 20](#_Toc85934184)

[3.4 Работа с историей посещения сайтов веб-браузера 21](#_Toc85934185)

[3.5 Работа с данными веб-браузера 23](#_Toc85934186)

[4 Тестирование программного средства 26](#_Toc85934187)

[5 Руководство пользователя 27](#_Toc85934188)

[5.1 Интерфейс программного средства 27](#_Toc85934189)

[5.2 Управление программным средством 29](#_Toc85934190)

[Заключение 30](#_Toc85934191)

[Список использованных источников 31](#_Toc85934192)

[Приложение А. Исходный код программы 32](#_Toc85934196)

# ВВЕДЕНИЕ

Стратегические игры на протяжении веков остаются неотъемлемой частью человеческой культуры, развивая логическое мышление, терпение и умение прогнозировать действия соперника. Одной из древнейших и наиболее сложных интеллектуальных игр является Го, возникшая в Китае более 2500 лет назад. Её глубина и элегантность правил сделали Го популярной во всём мире, а современные технологии позволили перенести эту игру в цифровое пространство, обеспечив возможность соревноваться с игроками из разных уголков планеты.

Развитие сетевых технологий и рост интереса к онлайн-взаимодействию обусловили необходимость создания цифровых платформ для классических игр. Сетевая версия Го не только сохраняет традиционные аспекты игры, но и добавляет новые возможности: автоматический подсчёт очков, анализ партий, многопользовательские режимы и глобальные рейтинги. Это делает её доступной как для новичков, изучающих основы, так и для опытных игроков, оттачивающих мастерство.

Целью данного курсового проекта является разработка сетевой компьютерной игры «Го», реализующей базовые правила игры, поддерживающей онлайн-взаимодействие между игроками и предоставляющей интуитивно понятный интерфейс.

Актуальность проекта обусловлена растущим спросом на цифровые платформы для классических игр, а также необходимостью изучения и применения современных технологий в области сетевого программирования и разработки пользовательских интерфейсов. Реализация игры Го позволяет углубить знания в области объектно-ориентированного программирования, многопоточности, работы с графическими библиотеками и протоколами передачи данных.

Разработанное программное средство демонстрирует возможность сочетания традиционных игровых механик с современными технологиями, что соответствует тенденциям цифровизации культурного наследия. Проект может быть использован как образовательный инструмент для изучения Го, так и основа для дальнейшего расширения функционала (например, внедрения искусственного интеллекта или турнирных систем).

# АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

## Обзор аналогов

На сегодняшний день существует несколько популярных платформ для онлайн-игры в Го, каждая из которых предлагает уникальный набор функций. В первую очередь стоит рассмотреть сервис Online Go Server (OGS). Внешний вид данного приложения представлен на рисунке 1.1.

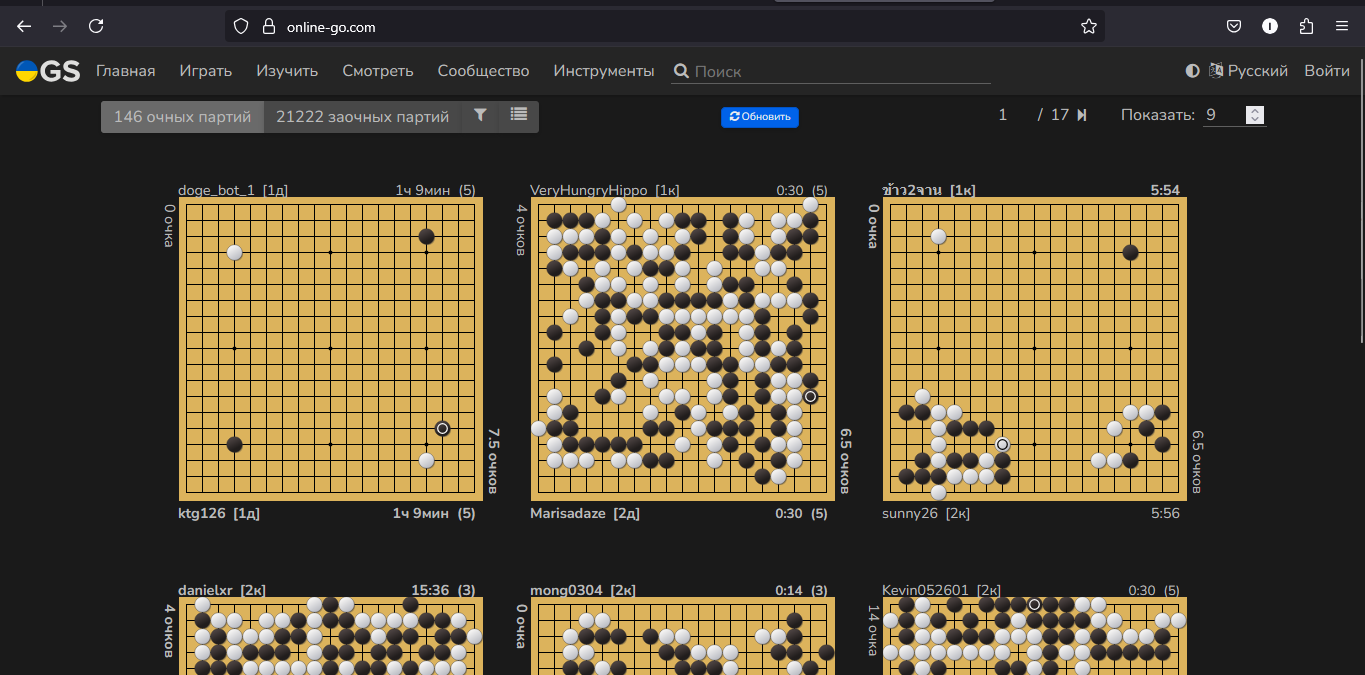


Рисунок 1.1 - Online Go Server

OGS представляет собой браузерную платформу, которая позволяет играть в Го с соперниками из разных стран. Сервис поддерживает рейтинговую систему, автоматический подсчёт очков и возможность анализа завершённых партий. Платформа интегрирована с социальными функциями: игроки могут общаться в чате, создавать клубы по интересам и участвовать в турнирах. Важной особенностью OGS является открытый исходный код, что позволяет сообществу разработчиков вносить улучшения. Однако интерфейс сервиса может показаться сложным для новичков из-за обилия настроек и инструментов.

Далее стоит рассмотреть мобильное приложение BadukPop. Внешний вид данного приложения представлен на рисунке 1.2.



Рисунок 1.2 - BadukPop

BadukPop ориентирован на обучение и игру начинающих. Приложение предлагает упрощённый интерфейс с минималистичным дизайном, систему достижений и ежедневные задания для мотивации пользователей. Важной особенностью является возможность игры против искусственного интеллекта с регулируемым уровнем сложности. Однако BadukPop уступает другим платформам в функциональности: отсутствует поддержка браузерной версии, а анализ партий ограничен базовыми возможностями.

Ещё одним известным решением является сервер KGS Go Server. Данная платформа существует более двух десятилетий и популярна среди профессиональных игроков. KGS предоставляет стабильное соединение, голосовой чат и возможность наблюдать за партиями других пользователей в реальном времени. Недостатком сервера можно считать устаревший интерфейс и отсутствие мобильной версии, что делает его менее удобным для современных пользователей.

Среди азиатских аналогов выделяется Tygem — платформа, популярная в Корее и Китае. Tygem предлагает высокую скорость игры, интеграцию с профессиональными лигами и обширную базу обучающих материалов. Особенностью сервиса является система «тайных комнат», где игроки могут соревноваться анонимно. Однако языковой барьер и сложности с регистрацией для пользователей из других регионов ограничивают аудиторию платформы.

Анализ существующих решений показывает, что большинство платформ ориентировано либо на профессиональных игроков, либо на новичков. Разрабатываемая сетевая игра «Го» призвана объединить преимущества аналогов, предложив интуитивный интерфейс для начинающих, базовые инструменты анализа и стабильную работу в многопользовательском режиме.

## Постановка задачи

В данной курсовом проекте планируется разработать сетевую компьютерную игру «Го» для операционных систем Windows. В процессе разработки должны быть реализованы базовые функции, соответствующие классическим правилам игры, а также возможности для онлайн-взаимодействия между игроками.

Базовый функционал включает:  
– создание игровой доски стандартного размера 9×9.  
– размещение камней игроками поочередно с автоматической проверкой валидности ходов;  
– механизм захвата камней, включая проверку условий *атари* и удаление групп без дыхательных точек;  
– подсчёт территории и очков по окончании партии с учётом коми (компенсации за первый ход);  
– реализацию правила для запрета повторяющихся позиций.

Для обеспечения сетевого взаимодействия будет реализована синхронизация состояния доски между клиентами в реальном времени.

Техническая реализация предполагает использование языка программирования Python с применением следующих инструментов и технологий:  
– графическая библиотека Pygame для создания интерфейса клиента;  
– фреймворк Django Channels для организации асинхронного сетевого взаимодействия через WebSocket;  
– СУБД PostgreSQL для хранения данных пользователей и истории партий;  
– алгоритмы проверки правил на основе матричных операций библиотеки NumPy.

Проект будет реализован в среде разработки PyCharm с использованием системы контроля версий Git. Тестирование функционала проводится на локальном сервере с последующим развёртыванием на облачной платформе Heroku для демонстрации многопользовательского режима.

# ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

## 2.1 Структура программы

В разработке приложения использовано 6 модулей:

* Сlient – модуль, для инициализации клиента в игре
* Gui – модуль, обеспечивающий работу с графическим интерфейсом
* GameLogic – модуль, ответственный за состояние программы в различных случаях, содержит в себе правила игры и обработку ввода клиентов
* GameState – модуль, содержащий в себе функции ходов клиентов
* Server – модуль, отвечающий за принятие клиентов, прослушивание сообщений, обработку и передачу ответов клиентам и взаимодействие их с игрой
* Init – модуль, служащий за самоинициализацию игры

## 2.2 Проектирование интерфейса программного средства

При разработке интерфейса сетевой игры «Го» за основу взят классический вид игровой доски с сохранением традиционных элементов. Главное окно приложения разделено на три основные зоны. В центральной части расположена игровая доска стандартного размера 9×9, выполненная в виде сетки с горизонтальными и вертикальными линиями. На пересечениях линий отмечены «звёздные точки» — специальные маркеры, используемые для ориентации на доске. Внешний вид главного окна представлен на рисунке 2.1.

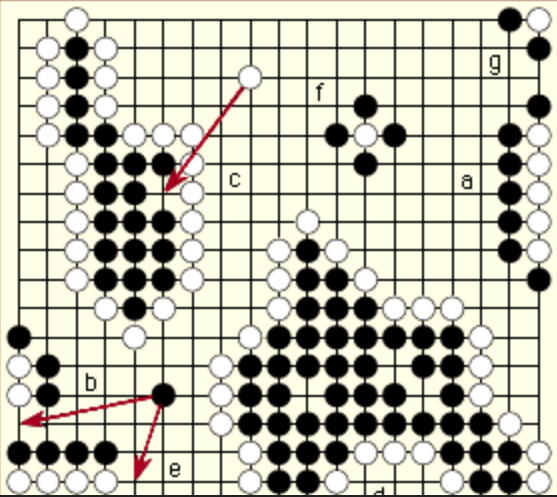


Рисунок 2.1 – Главное окно игры

В нижней части окна размещена информационная панель. Слева отображается текущий игрок (чёрные или белые камни), справа — счёт партии, включая территорию и захваченные камни. Шрифты и цветовая гамма подобраны для обеспечения максимальной читаемости. Подсветка текущего хода реализована через изменение фона метки игрока.

Ниже информационной панели расположена панель управления. Центральным элементом является кнопка «Пас», позволяющая пропустить ход. Рядом с ней размещена кнопка «Правила», открывающая окно с кратким описанием основных правил игры.

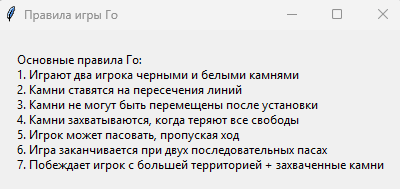


Рисунок 2.2 – Окно «Правила»

Игровая доска реализована с использованием графического холста, поддерживающего взаимодействие с мышью. При наведении курсора на пересечение линий отображается полупрозрачный камень текущего игрока, что помогает визуализировать предполагаемый ход. Для размещения камня достаточно кликнуть левой кнопкой мыши.

При завершении партии появляется модальное окно с итоговыми результатами. Оно содержит информацию о победителе, совмещённый счёт по территориям и захваченным камням, а также кнопку для начала новой игры.

## 2.3 Проектирование функционала программного средства

Основу игровой логики составляет механизм размещения камней на доске. Процесс начинается с проверки валидности хода: игрок не может поставить камень на занятую точку, нарушить правило ко или совершить самоубийственный ход. Если ход допустим, камень размещается на доске, после чего проверяются соседние точки на наличие групп камней противника, потерявших свободы.

Все захваченные камни удаляются с доски, и очки записываются в счёт текущего игрока. Блок-схема метода размещения камня представлена на рисунке 2.3.

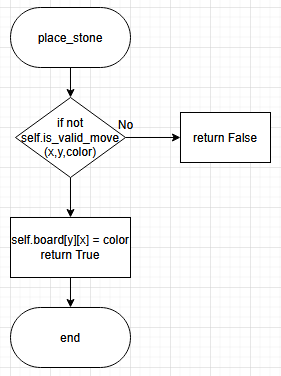


Рисунок 2.3 – Блок-схема функции place\_stone

Важным аспектом является проверка условия окончания игры. Игра завершается при двух последовательных пасах, или когда ни один из игроков не может сделать допустимый ход. В этом случае запускается подсчёт очков: учитывается окружённая территория и захваченные камни, с добавлением коми (компенсации) для белых. Блок-схема завершения игры представлена на рисунке 2.4.

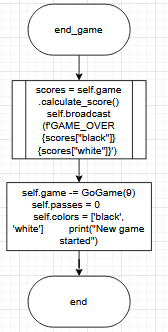


Рисунок 2.4 – Блок-схема функции end\_game

Сетевое взаимодействие реализовано через обмен сообщениями между клиентом и сервером. При получении хода от игрока сервер проверяет его корректность, обновляет состояние доски и рассылает изменения всем подключённым клиентам. В случае паса или невозможности дальнейших ходов сервер инициирует завершение партии и передачу финальных результатов. Блок-схема обработки сетевых сообщений представлена на рисунке 2.5.

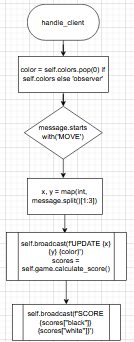


Рисунок 2.5 – Блок-схема функции handle\_client

# РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

**3.1 Реализация сервера**

Сервер запускается на указанном порту и ожидает подключений игроков. Для каждого нового клиента создается отдельный поток, обрабатывающий входящие сообщения. При подключении игроку присваивается цвет («чёрный» или «белый»), и он получает текущее состояние игры. Код инициализации сервера представлен ниже:

class GoServer:

def \_\_init\_\_(self, host='127.0.0.1', port=5555):

self.server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

self.server.bind((host, port))

self.server.listen()

self.clients = []

self.game = GoGame(9)

self.colors = ['black', 'white']

При получении хода от клиента (сообщение MOVE x y) сервер проверяет его валидность через метод place\_stone() класса GoGame. Если ход корректен, обновлённое состояние доски рассылается всем участникам. В случае двух последовательных пасов или отсутствия допустимых ходов игра завершается, и результаты передаются клиентам.

**3.2 Игровая логика**

Класс GoGame инкапсулирует правила игры. Метод is\_valid\_move() проверяет: не занята ли точка камнем, не нарушено ли правило ко (повтор позиции), не является ли ход самоубийственным (группа камней не теряет свободы).

Пример проверки свободы группы:

def has\_liberties(self, group):

for (x, y) in group:

for dx, dy in [(-1,0), (1,0), (0,-1), (0,1)]:

nx, ny = x+dx, y+dy

if 0 <= nx < self.size and 0 <= ny < self.size:

if self.board[ny][nx] is None:

return True

return False

После размещения камня вызывается метод capture\_stones(), который удаляет окружённые группы противника. Подсчёт очков учитывает захваченные камни и территорию, с добавлением коми для белых. Блок-схема обработки хода представлена на рисунке 3.1.

**3.3 Графический интерфейс клиента**

Интерфейс отображает доску 19×19, информацию о текущем игроке и счёт. При наведении курсора на пересечение линий появляется полупрозрачный камень, клик левой кнопкой отправляет ход на сервер. Отрисовка доски реализована на холсте Tkinter:

def draw\_board(self):

for i in range(self.board\_size):

x = self.cell\_size \* (i + 1)

self.canvas.create\_line(x, self.cell\_size, x, self.cell\_size\*self.board\_size, width=2)

self.canvas.create\_line(self.cell\_size, x, self.cell\_size\*self.board\_size, x, width=2)

**3.4 Сетевое взаимодействие**

Клиент подключается к серверу и в отдельном потоке обрабатывает входящие сообщения. При получении UPDATE x y color обновляется отображение доски. Сообщение SCORE изменяет счёт, а GAME\_OVER вызывает окно с итогами партии. Отправка хода реализована в методе:

def send\_move(self, x, y):

if self.color and self.running:

self.client.send(f'MOVE {x} {y}'.encode('utf-8'))

**3.5 Завершение игры**

При завершении партии сервер сбрасывает состояние и уведомляет клиентов. Клиент отображает окно с победителем и кнопкой для новой игры (рисунок 3.2). Метод show\_game\_over() формирует итоговое сообщение:

def show\_game\_over(self, scores):

winner = 'Black' if scores['black'] > scores['white'] else 'White'

result\_text = f"Победитель: {winner}\nОчки Black: {scores['black']}\nОчки White: {scores['white']}"

label = tk.Label(self.game\_over\_window, text=result\_text)

label.pack()

def is\_game\_over(self):

"""Проверяет условия окончания игры"""

# Если оба игрока не имеют допустимых ходов

return not self.has\_valid\_moves('black') and not self.has\_valid\_moves('white')

elif message.startswith('GAME\_OVER'):

\_, black\_score, white\_score = message.split() self.gui.show\_game\_over({

'black': float(black\_score), 'white': float(white\_score)

})

# ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

В процессе тестирования сетевого приложения для игры Го были выявлены некоторые ошибки и некорректное поведение программного обеспечения. Основной проблемой стала некорректная обработка ходов игроков и синхронизация состояния игры между сервером и клиентами. В частности, возникали ситуации, когда клиенты не получали своевременно обновления о ходе противника, или же игра некорректно завершалась при двух подряд пасах.

Проблема заключалась в отсутствии правильной проверки и обработки ходов на стороне сервера, а также недостаточной реализации протокола обмена сообщениями между сервером и клиентами. Для устранения неполадок был переработан метод handle\_client() сервера, который теперь корректно обрабатывает команды MOVE и PASS, обновляет состояние игры, ведет подсчет очков и рассылает обновления всем подключенным клиентам. Также добавлена проверка на окончание игры по двум подряд пасам или по отсутствию допустимых ходов.

Кроме того, на стороне клиента была улучшена логика приема и отображения сообщений от сервера. Теперь интерфейс пользователя своевременно обновляет доску, отображает текущего игрока, счет и информирует о завершении партии с указанием победителя. В графическом интерфейсе добавлена возможность показать правила игры и предпросмотр хода.

Так, например, метод handle\_client() сервера реализован следующим образом:

def handle\_client(self, client):

color = self.colors.pop(0) if self.colors else 'observer'

try:

client.send(f'COLOR {color}'.encode('utf-8'))

except:

self.clients.remove(client)

client.close()

return

while True:

try:

message = client.recv(1024).decode('utf-8')

if not message:

break

if message.startswith('MOVE'):

x, y = map(int, message.split()[1:3])

if self.game.place\_stone(x, y, color):

self.broadcast(f'UPDATE {x} {y} {color}')

scores = self.game.calculate\_score()

self.broadcast(f'SCORE {scores["black"]} {scores["white"]}')

if self.game.is\_game\_over():

self.end\_game()

except:

break

self.clients.remove(client)

client.close()

Данный метод теперь отвечает за корректную обработку ходов и передачу состояния игры всем участникам, что исключило рассинхронизацию и ошибки в логике. Аналогично, клиентская часть была доработана для правильного отображения текущего состояния и уведомлений.

Большинство выявленных недочетов возникло из-за недостаточного продумывания протокола взаимодействия и обмена сообщениями между клиентом и сервером. После внесенных исправлений и повторного тестирования программное средство продемонстрировало стабильную и корректную работу без сбоев и ошибок при игре нескольких пользователей в режиме реального времени.

# РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

**5.1 Интерфейс программного средства**

Главное окно приложения разделено на три основные зоны (рис. 5.1). В центре расположена игровая доска размером 9×9, выполненная в виде сетки с горизонтальными и вертикальными линиями. На пересечениях линий отмечены «звёздные точки» — специальные маркеры, помогающие ориентироваться на доске.

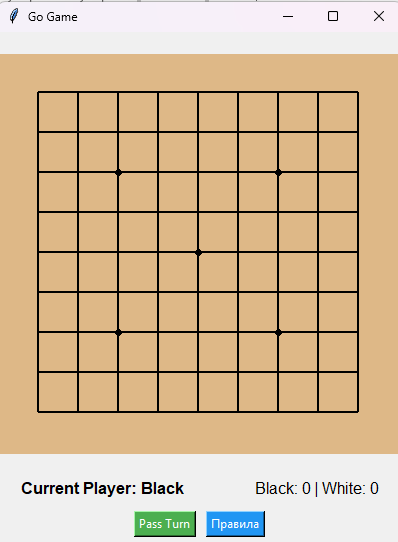


Рисунок 5.1 – Главное окно игры

В нижней части окна находится информационная панель. Слева отображается текущий игрок (чёрные или белые камни), справа — счёт партии, включая захваченные камни и территорию. Подсветка текущего хода реализована через изменение фона метки игрока. Также, ещё нижей расположены кнопки управления: пас — пропуск хода; правила — открывает окно с кратким описанием игры.

При наведении курсора на пересечение линий доски отображается полупрозрачный камень текущего игрока, что помогает визуализировать предполагаемый ход. Для размещения камня достаточно кликнуть левой кнопкой мыши.

**5.2 Дополнительные окна**

При нажатии кнопки «Правила» открывается окно с текстовым описанием основных принципов игры (рис. 5.2). Оно содержит информацию о ходе игры, условиях захвата камней и завершении партии.

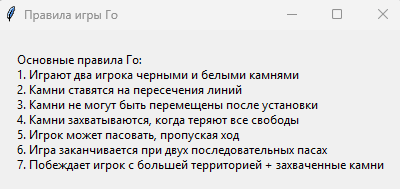


Рисунок 5.2 – Окно «Правила»

По окончании игры появляется модальное окно с результатами (рис. 5.3). В нём отображается победитель, детализированный счёт и кнопка «Новая игра» для перезапуска партии.

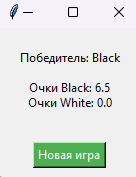


Рисунок 5.2 – Окно «Результаты»

## 5.3 Управление игрой

1 Размещение камня: Кликните левой кнопкой мыши на пересечение линий доски.

1. Пас: Нажмите кнопку Пас, чтобы пропустить ход.
2. Навигация по истории: После завершения партии используйте кнопки в окне результатов для просмотра ходов.
3. Новая игра: В окне результатов нажмите Новая игра, чтобы начать партию заново.

Важные особенности:

1. При попытке сделать недопустимый ход (например, самоубийственный) появится сообщение об ошибке.
2. Игра автоматически завершается при двух последовательных пасах или отсутствии допустимых ходов.
3. Для выхода из приложения закройте главное окно.

Интерфейс адаптирован для различных разрешений экрана. Все элементы управления имеют всплывающие подсказки при наведении курсора.

# ЗАКЛЮЧЕНИЕ

В настоящее время сетевые многопользовательские игры приобретают все большую популярность, а реализация надежного и удобного взаимодействия между игроками становится важной задачей. В рамках данного курсового проекта было разработано программное средство — клиент-серверное приложение для игры Го, которое обеспечивает одновременную игру нескольких пользователей по сети. В процессе разработки были успешно выполнены все поставленные задачи.

Реализован сервер, который обрабатывает подключения клиентов, принимает и проверяет ходы, ведет учет состояния игры и передает обновления всем участникам. Клиентская часть обеспечивает удобный графический интерфейс для отображения игрового поля, текущего состояния, счета и ходов игроков, а также предоставляет возможность просмотра правил игры и управления ходами. Между сервером и клиентами реализован протокол обмена текстовыми сообщениями, позволяющий синхронизировать состояние игры в реальном времени.

В ходе тестирования приложение показало стабильную работу, корректную обработку ходов, своевременное обновление интерфейса и правильное завершение партии. Были выявлены и устранены ошибки, связанные с обработкой пасов и завершением игры. За счет продуманной архитектуры и разделения логики игры, сетевого взаимодействия и пользовательского интерфейса достигнута высокая надежность и удобство использования.

Несмотря на достигнутые результаты, возможны дальнейшие улучшения, такие как расширение функционала клиента, добавление поддержки более сложных правил и вариантов игры, улучшение обработки сбоев сети, а также оптимизация производительности при большом количестве игроков. В целом разработанное программное средство демонстрирует возможности создания интерактивных многопользовательских приложений с применением современных технологий и может служить основой для дальнейших проектов в области сетевых игр.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Лутц М. Изучаем Python. — СПб.: Питер, 2019. — 1024 с.

[2] Марк Л. Флэнеган. JavaScript: Подробное руководство. — СПб.: Питер, 2018. — 1152 с.

[3] Свифт М. Программирование на Python для начинающих. — Москва: БХВ-Петербург, 2020. — 560 с.

[4] Бьёрн Страуструп. Язык программирования C++. 4-е изд. — СПб.: Питер, 2017. — 1360 с.

[5] Мацкевич А. Создание клиент-серверных приложений на Python. — Москва: ДМК Пресс, 2021. — 384 с.

[6] Ван Россум Г., Дрейк Ф. Python. Курс молодого бойца. — СПб.: Питер, 2016. — 576 с.

[7] О’Рейли Т. Сетевое программирование с Python. — Москва: Вильямс, 2019. — 432 с.

[8] Гутталл М. Графические интерфейсы на Python с Tkinter. — СПб.: Питер, 2018. — 312 с.

[9] Зельцер С. Архитектура клиент-серверных приложений. — Москва: Горячая линия – Телеком, 2017. — 400 с.

[10] Петров И. Основы разработки многопользовательских игр. — СПб.: БХВ-Петербург, 2020. — 480 с.

# ПРИЛОЖЕНИЕ А. Исходный код программы

Server.py:

import socket

import threading

from game\_logic import GoGame

class GoServer:

def \_\_init\_\_(self, host='127.0.0.1', port=5555):

self.server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

self.server.bind((host, port))

self.server.listen()

self.clients = []

self.scores = 0

self.game = GoGame(9)

self.colors = ['black', 'white']

print(f"Server listening on {host}:{port}")

def broadcast(self, message):

for client in self.clients[:]:

try:

client.send(message.encode('utf-8'))

except:

self.clients.remove(client)

client.close()

def handle\_client(self, client):

color = self.colors.pop(0) if self.colors else 'observer'

try:

client.send(f'COLOR {color}'.encode('utf-8'))

except:

self.clients.remove(client)

client.close()

return

while True:

try:

message = client.recv(1024).decode('utf-8')

if not message:

break

if message.startswith('MOVE'):

x, y = map(int, message.split()[1:3])

if self.game.place\_stone(x, y, color):

self.broadcast(f'UPDATE {x} {y} {color}')

scores = self.game.calculate\_score()

self.broadcast(f'SCORE {scores["black"]} {scores["white"]}')

# Проверка на окончание игры

if self.game.is\_game\_over():

self.end\_game()

elif message == 'PASS':

self.game.passes += 1

if self.game.passes >= 2 or self.game.is\_game\_over():

self.end\_game()

else:

self.game.current\_player = 'white' if self.game.current\_player == 'black' else 'black'

self.broadcast(f'TURN {self.game.current\_player}')

scores = self.game.calculate\_score()

self.broadcast(f'SCORE {scores["black"]} {scores["white"]}')

except:

break

self.clients.remove(client)

client.close()

def end\_game(self):

"""Обработка окончания игры"""

scores = self.game.calculate\_score()

self.broadcast(f'GAME\_OVER {scores["black"]} {scores["white"]}')

# Сброс состояния игры

self.game = GoGame(9)

self.passes = 0

self.colors = ['black', 'white']

print("New game started")

def run(self):

while True:

client, addr = self.server.accept()

print(f"Connected with {addr}")

self.clients.append(client)

thread = threading.Thread(target=self.handle\_client, args=(client,))

thread.start()

if \_\_name\_\_ == "\_\_main\_\_":

server = GoServer()

server.run()

class GameState:

def \_\_init\_\_(self, size):

self.size = size

self.board = [[None for \_ in range(size)] for \_ in range(size)]

self.current\_player = 'black'

def make\_move(self, x, y, color):

if color != self.current\_player:

return False

if self.board[y][x] is None:

self.board[y][x] = color

self.capture\_stones(x, y, color)

self.current\_player = 'white' if color == 'black' else 'black'

return True

return False

def place\_stone(self, x, y, color):

if not self.is\_valid\_move(x, y, color):

return False

self.board[y][x] = color

self.capture\_stones(x, y, color)

self.update\_ko()

self.current\_player = 'white' if color == 'black' else 'black'

return True

class GoGame:

def \_\_init\_\_(self, size=9):

self.size = size

self.board = [[None]\*size for \_ in range(size)]

self.current\_player = 'black'

self.captured = {'black': 0, 'white': 0}

self.previous\_states = []

self.passes = 0

self.ko = None

def is\_valid\_move(self, x, y, color):

if self.board[y][x] is not None or color != self.current\_player:

return False

# Проверка правила ко

temp\_board = [row.copy() for row in self.board]

temp\_board[y][x] = color

if self.ko and temp\_board == self.ko:

return False

# Проверка самоубийственного хода

if not self.check\_liberty(x, y, temp\_board):

return False

return True

def check\_liberty(self, x, y, board):

visited = set()

stack = [(x, y)]

color = board[y][x]

while stack:

cx, cy = stack.pop()

if (cx, cy) in visited:

continue

visited.add((cx, cy))

for dx, dy in [(-1,0), (1,0), (0,-1), (0,1)]:

nx, ny = cx+dx, cy+dy

if 0 <= nx < self.size and 0 <= ny < self.size:

if board[ny][nx] is None:

return True

elif board[ny][nx] == color and (nx, ny) not in visited:

stack.append((nx, ny))

return False

def place\_stone(self, x, y, color):

if not self.is\_valid\_move(x, y, color):

return False

self.board[y][x] = color

self.capture\_stones(x, y, color) # Захват камней

self.update\_ko()

self.current\_player = 'white' if color == 'black' else 'black'

return True

def capture\_stones(self, x, y, color):

opponent = 'white' if color == 'black' else 'black'

for dx, dy in [(-1, 0), (1, 0), (0, -1), (0, 1)]:

nx, ny = x + dx, y + dy

if 0 <= nx < self.size and 0 <= ny < self.size:

if self.board[ny][nx] == opponent:

group = self.get\_group(nx, ny)

if not self.has\_liberties(group):

self.remove\_group(group)

self.captured[color] += len(group)

def get\_group(self, x, y):

color = self.board[y][x]

visited = set()

stack = [(x, y)]

group = []

while stack:

cx, cy = stack.pop()

if (cx, cy) in visited:

continue

visited.add((cx, cy))

group.append((cx, cy))

for dx, dy in [(-1,0), (1,0), (0,-1), (0,1)]:

nx, ny = cx+dx, cy+dy

if 0 <= nx < self.size and 0 <= ny < self.size:

if self.board[ny][nx] == color and (nx, ny) not in visited:

stack.append((nx, ny))

return group

def has\_liberties(self, group):

for (x, y) in group:

for dx, dy in [(-1,0), (1,0), (0,-1), (0,1)]:

nx, ny = x+dx, y+dy

if 0 <= nx < self.size and 0 <= ny < self.size:

if self.board[ny][nx] is None:

return True

return False

def remove\_group(self, group):

for x, y in group:

self.board[y][x] = None

def update\_ko(self):

self.ko = [row.copy() for row in self.board]

def calculate\_score(self):

return {

'black': self.captured['black'] + 6.5,

'white': self.captured['white']

}

def has\_valid\_moves(self, color):

"""Проверяет наличие допустимых ходов для указанного цвета"""

if color != self.current\_player:

return False

for y in range(self.size):

for x in range(self.size):

if self.is\_valid\_move(x, y, color):

return True

return False

def is\_game\_over(self):

"""Проверяет условия окончания игры"""

# Если оба игрока не имеют допустимых ходов

return not self.has\_valid\_moves('black') and not self.has\_valid\_moves('white')

import tkinter as tk

from tkinter import messagebox

class GoGUI:

def \_\_init\_\_(self, client, board\_size=9):

self.client = client

self.board\_size = board\_size

self.cell\_size = 40

self.stone\_radius = self.cell\_size // 2 - 2

self.window = tk.Tk()

self.window.title("Go Game")

# Основной холст

self.canvas = tk.Canvas(self.window,

width=self.cell\_size\*(board\_size+1),

height=self.cell\_size\*(board\_size+1),

bg='#DEB887')

self.canvas.pack(pady=20)

# Панель информации

self.info\_frame = tk.Frame(self.window)

self.info\_frame.pack(fill=tk.X, padx=20)

self.current\_player\_label = tk.Label(

self.info\_frame,

text="Current Player: Black",

font=('Helvetica', 12, 'bold'))

self.current\_player\_label.pack(side=tk.LEFT)

self.score\_label = tk.Label(

self.info\_frame,

text="Black: 0 | White: 0",

font=('Helvetica', 12)

)

self.score\_label.pack(side=tk.RIGHT)

# Кнопки управления

self.control\_frame = tk.Frame(self.window)

self.control\_frame.pack(pady=10)

self.pass\_button = tk.Button(

self.control\_frame,

text="Pass Turn",

command=self.pass\_turn,

bg='#4CAF50',

fg='white'

)

self.pass\_button.pack(side=tk.LEFT, padx=5)

# Инициализация доски

self.draw\_board()

self.preview\_stone = None

self.canvas.bind("<Motion>", self.show\_preview)

self.canvas.bind("<Leave>", self.hide\_preview)

self.canvas.bind("<Button-1>", self.handle\_click)

self.rules\_button = tk.Button(

self.control\_frame,

text="Правила",

command=self.show\_rules,

bg='#2196F3',

fg='white'

)

self.rules\_button.pack(side=tk.LEFT, padx=5)

def show\_rules(self):

rules\_window = tk.Toplevel(self.window)

rules\_window.title("Правила игры Го")

rules\_text = """Основные правила Го:

1. Играют два игрока черными и белыми камнями

2. Камни ставятся на пересечения линий

3. Камни не могут быть перемещены после установки

4. Камни захватываются, когда теряют все свободы

5. Игрок может пасовать, пропуская ход

6. Игра заканчивается при двух последовательных пасах

7. Побеждает игрок с большей территорией + захваченные камни"""

rules\_label = tk.Label(rules\_window, text=rules\_text, justify=tk.LEFT, padx=20, pady=20)

rules\_label.pack()

def draw\_board(self):

# Отрисовка линий

for i in range(self.board\_size):

x = self.cell\_size \* (i + 1)

self.canvas.create\_line(

x, self.cell\_size,

x, self.cell\_size\*self.board\_size,

width=2, fill='black'

)

self.canvas.create\_line(

self.cell\_size, x,

self.cell\_size\*self.board\_size, x,

width=2, fill='black'

)

# Звездные точки

star\_points = [(2,2), (2,6), (6,2), (6,6), (4,4)]

for x, y in star\_points:

center\_x = (x + 1) \* self.cell\_size

center\_y = (y + 1) \* self.cell\_size

self.canvas.create\_oval(

center\_x - 3, center\_y - 3,

center\_x + 3, center\_y + 3,

fill='black'

)

def show\_preview(self, event):

if self.client.color != self.client.current\_turn:

return

x, y = self.get\_grid\_coords(event)

if x is not None:

# Удаляем предыдущий предварительный камень, если он существует

if self.preview\_stone:

self.canvas.delete(self.preview\_stone)

# Отрисовка нового предварительного камня

color = '#888888' if self.client.color == 'black' else '#DDDDDD'

self.preview\_stone = self.draw\_stone(x, y, color)

def hide\_preview(self, event):

if self.preview\_stone:

self.canvas.delete(self.preview\_stone)

self.preview\_stone = None

def handle\_click(self, event):

try:

x, y = self.get\_grid\_coords(event)

if x is not None:

self.client.send\_move(x, y)

except Exception as e:

self.show\_error(str(e))

def get\_grid\_coords(self, event):

grid\_x = round((event.x - self.cell\_size)/self.cell\_size)

grid\_y = round((event.y - self.cell\_size)/self.cell\_size)

if 0 <= grid\_x < self.board\_size and 0 <= grid\_y < self.board\_size:

return (grid\_x, grid\_y)

return (None, None)

def draw\_stone(self, x, y, color):

x\_pixel = (x + 1) \* self.cell\_size

y\_pixel = (y + 1) \* self.cell\_size

return self.canvas.create\_oval(

x\_pixel - self.stone\_radius,

y\_pixel - self.stone\_radius,

x\_pixel + self.stone\_radius,

y\_pixel + self.stone\_radius,

fill=color,

outline='black' if color != 'white' else '#666666'

)

def update\_board(self, x, y, color):

stone\_color = 'black' if color == 'black' else 'white'

self.draw\_stone(x, y, stone\_color)

def update\_info(self, current\_player, scores):

"""Обновляет информацию о текущем игроке и счете"""

# Обновляем счет

score\_text = f"Black: {scores.get('black', 0)} | White: {scores.get('white', 0)}"

self.score\_label.config(text=score\_text)

# Обновляем текущего игрока

if current\_player:

player\_text = f"Current Player: {current\_player.capitalize()}"

self.current\_player\_label.config(text=player\_text)

def pass\_turn(self):

self.client.send\_pass()

def run(self):

self.window.mainloop()

def show\_error(self, message):

messagebox.showerror("Error", message)

def show\_game\_over(self, scores):

# Убираем предыдущее окно если есть

if hasattr(self, 'game\_over\_window'):

self.game\_over\_window.destroy()

self.game\_over\_window = tk.Toplevel(self.window)

self.game\_over\_window.title("Игра окончена!")

winner = 'Black' if scores['black'] > scores['white'] else 'White'

result\_text = (

f"Победитель: {winner}\n\n"

f"Очки Black: {scores['black']}\n"

f"Очки White: {scores['white']}"

)

label = tk.Label(self.game\_over\_window, text=result\_text, padx=20, pady=20)

label.pack()

new\_game\_button = tk.Button(

self.game\_over\_window,

text="Новая игра",

command=self.restart\_game,

bg='#4CAF50',

fg='white'

)

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

def restart\_game(self):

"""Обработчик кнопки новой игры"""

self.game\_over\_window.destroy()

import socket

import threading

from gui import GoGUI

class GoClient:

def \_\_init\_\_(self, server\_ip='127.0.0.1', port=5555):

self.client = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

self.client.connect((server\_ip, port))

self.gui = GoGUI(self)

self.color = None

self.current\_turn = 'black' # Initialize current\_turn

self.receive\_thread = threading.Thread(target=self.receive)

self.receive\_thread.start()

self.running = True

self.gui.run()

def receive(self):

while True:

try:

message = self.client.recv(1024).decode('utf-8')

if message.startswith('TURN'):

self.current\_turn = message.split()[1]

# Исправлено: передаем текущий счет (по умолчанию 0:0)

self.gui.update\_info(self.current\_turn, {'black': 0, 'white': 0})

elif message.startswith('COLOR'):

self.color = message.split()[1]

self.gui.update\_info(self.color, {'black': 0, 'white': 0})

elif message.startswith('UPDATE'):

x, y, color = message.split()[1:]

self.gui.update\_board(int(x), int(y), color)

self.current\_turn = 'white' if color == 'black' else 'black'

# Добавляем передачу счета при обновлении доски

self.gui.update\_info(self.current\_turn, {'black': 0, 'white': 0})

elif message.startswith('PASS'):

color = message.split()[1]

self.current\_turn = 'white' if color == 'black' else 'black'

self.gui.update\_info(self.current\_turn, {}) # Обновить текущего игрока

elif message.startswith('SCORE'):

black, white = message.split()[1:]

self.gui.update\_info(self.current\_turn, {'black': black, 'white': white})

elif message.startswith('GAME\_OVER'):

\_, black\_score, white\_score = message.split()

self.gui.show\_game\_over({

'black': float(black\_score),

'white': float(white\_score)

})

except ConnectionAbortedError:

self.gui.show\_error("Connection lost")

break

except Exception as e:

print(f"Error: {e}")

break

def send\_move(self, x, y):

if self.color and self.running:

try:

self.client.send(f'MOVE {x} {y}'.encode('utf-8'))

except Exception as e:

print("Send failed:", e)

self.running = False

self.client.close()

def send\_pass(self):

if self.color:

self.client.send('PASS'.encode('utf-8'))

if \_\_name\_\_ == "\_\_main\_\_":

client = GoClient()