СОДЕРЖАНИЕ

[СОДЕРЖАНИЕ 6](#_Toc199369768)

[ВВЕДЕНИЕ 7](#_Toc199369769)

[1. АНАЛИЗ ПОСТАВЛЕННОЙ ЗАДАЧИ 8](#_Toc199369770)

[1.1 Описание предметной области 8](#_Toc199369771)

[1.2 Постановка задачи 10](#_Toc199369772)

[2. АНАЛИЗ ДАННЫХ 11](#_Toc199369773)

[2.1. Анализ входных данных 11](#_Toc199369774)

[2.2. Анализ промежуточных данных 11](#_Toc199369775)

[2.3. Анализ выходных данных 11](#_Toc199369776)

[4. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 15](#_Toc199369777)

[ЗАКЛЮЧЕНИЕ 17](#_Toc199369778)

[СПИСОК ЛИТЕРАТУРЫ 18](#_Toc199369779)

[ПРИЛОЖЕНИЕ 19](#_Toc199369780)

ВВЕДЕНИЕ

Сетевые игры являются важной частью современной цифровой индустрии, объединяя миллионы пользователей по всему миру. Одной из ключевых задач при разработке таких игр является эффективная организация взаимодействия между игроками, особенно в режиме реального времени. В данной работе рассматривается реализация классической игры «Точки» с поддержкой сетевой игры на основе управляемого широковещания на языке **Python**.

Актуальность темы обусловлена растущим интересом к легковесным сетевым играм с минимальными задержками. Управляемое широковещание позволяет оптимизировать передачу данных между участниками, снижая нагрузку на сервер и обеспечивая стабильность соединения.

Цель работы ­ разработать клиент-серверное приложение для игры «Точки» с использованием управляемого широковещания для синхронизации игровых событий.

Задачи исследования:

* Изучить принципы сетевого взаимодействия в сетевых играх.
* Проанализировать преимущества управляемого широковещания перед другими методами передачи данных.
* Реализовать игровую логику «Точек» с поддержкой широковещания.

1. АНАЛИЗ ПОСТАВЛЕННОЙ ЗАДАЧИ

1.1 Описание предметной области

Предметом области данной работы является создание компьютерной сетевой игры «Точки» с использованием управляемого широковещания.

Управляемое шировещание (multicast, broadcast) - форма широковещания, при которой адресом назначения сетевого пакета является мультикастная группа. В отличие от стандартной клиент-серверной схемы, которой отправляет ответы каждому пользователю индивидуально, широковещание даёт возможность серверу отправлять поток данных всем участником сети, что уменьшаяет нагрузку и ускоряет синхронизацию.(рисунок 1)

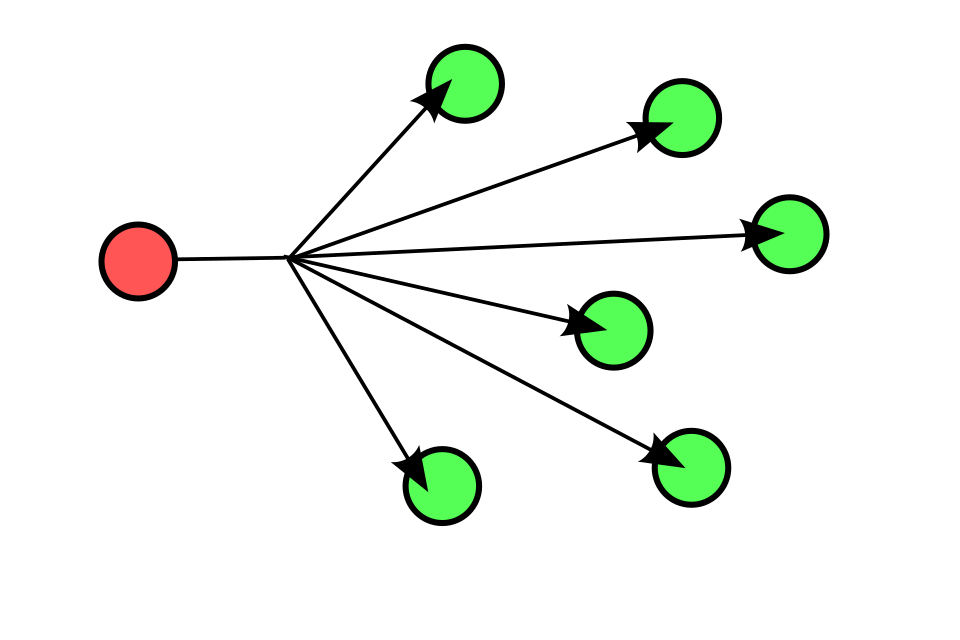
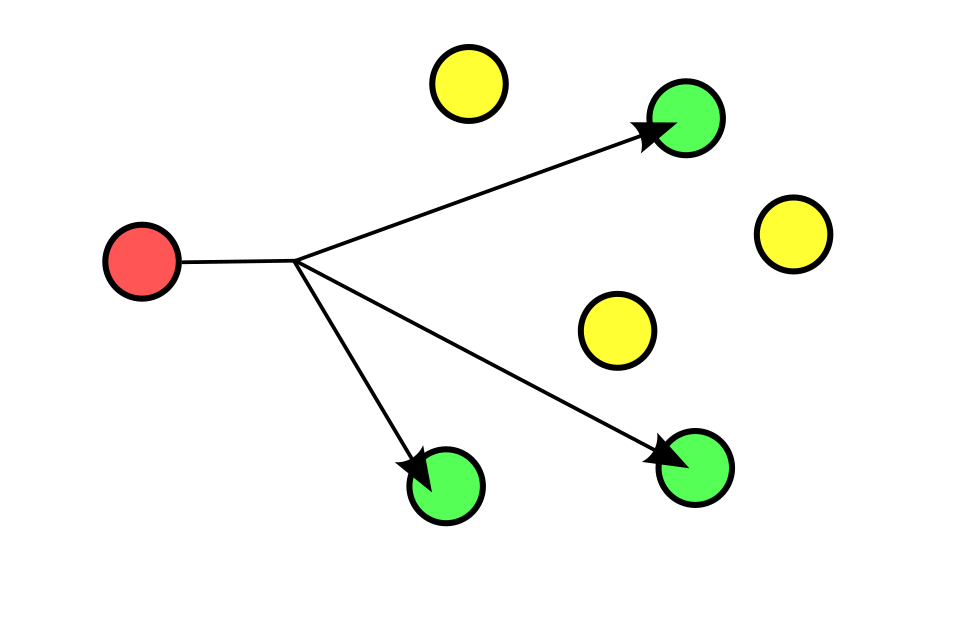


Рисунок 1 - broadcast

В нашем случает, а именно при создании сетевой игры, которая рассчитана на двух игроков, что должно учитывать хорошую синхронизацию игры, целостность передачи данных, безопасность, и возможность создания нескольких сессий одновременно. В этом нам может помочь multicast, в отличии от brodcast, групповая передача происходит не всем пользователям, а только группе лиц доступных для присоединения, что обеспечит безопасность и возможность создания нескольких сессий.(рисунок 2)

Рисунок 2 - multicast

Сама же игра «Точки» представляет собой квадратное поле из точек, где два игрока последовательно ставят точки с целью закрыть точки апонента. При создании замкнутой области все точки противника исчезают и превращаются в очки. Целью игры будет достижения максимального количества очков.(рисунок 3)

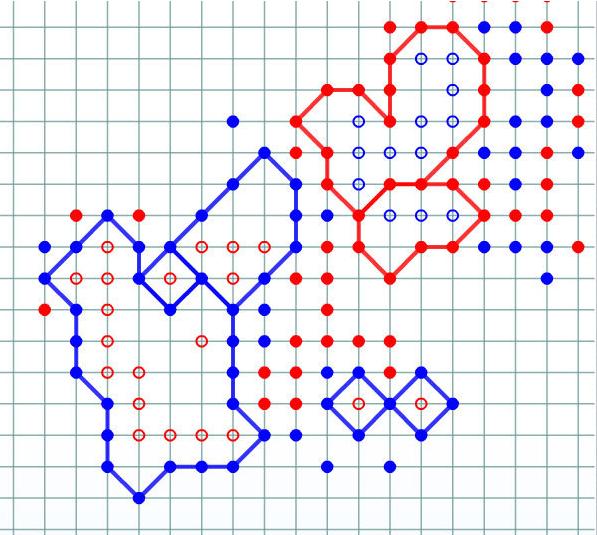


Рисунок 3 – игра “Точки”

1.2 Постановка задачи

Задачей данного проекта является реализация сетевой игры «Точки» с использованием управляемого широковещания.

Основными задачами разработки являются:

1. Выбрать язык программирования для создания первоначальной версии.
2. Разработать однопользовательскую игру “Точки” для реализации основных механик с учётом правил самой игры.
3. Обновить проект для возможности создания широковещательной связи и возможности игры по сети.
4. Провести тесты для выявления проблем и исправить.

2. АНАЛИЗ ДАННЫХ

2.1. Анализ входных данных

К входным данным относятся все данные, которые есть на стороне клиента и передаются серверу, такие как:

1. Действия пользователя. Такая как постановка точки.
2. Команды управления игрой. Такие как начло и завершение игры, начисления очков, удаление точек.
3. Индикатор пользователя, является он хостом, игроком, или зрителем.

2.2. Анализ промежуточных данных

К промежуточным данным относятся все данные, которые есть на стороне сервера и передаются клиенту в будущем, такие как:

1. Состояние игрового поля.
2. Список игроков и их роль.
3. Счёт игры.

2.3. Анализ выходных данных

К промежуточным данным относятся все данные, которые были обработаны сервером и переданы обратно игроку, такие как:

1. Обновление поля (Куда сходил противоположный игрок).
2. Сообщение о победе одного из игроков.
3. Счёт игры, если какие-то поля были захвачены.

**3.ПРОГРАММНАЯ РЕЛИЗАЦИЯ**

Для реализации игры «Точки» с использованием управляемого широковещания был выбран язык Python и некоторые библиотеки, такие как pygame, socket, threading.

Первая библиотека необходима для работы самой игры и её графического интерфейса, выбор именно этой библиотеки был обусловлен её лёгкостью в написании простых программ. Socket для сетевого взаимодействия с сервером используя TCP-соединение. Threading, он позволяет серверу обрабатывать несколько клиентов одновременно без блокировки основного потока.

Использование языка Python для реализации приложения обусловлено тем, что данный язык ориентирован на повышение производительности разработчика и читаемости кода, что даёт удобность и лёгкость в его использовании.

Для удобности пользователя, создания сервера было добавлено в само приложение. Хост пишет порт к которому в будущем будут присоединятся пользователи. (рисунок 4)

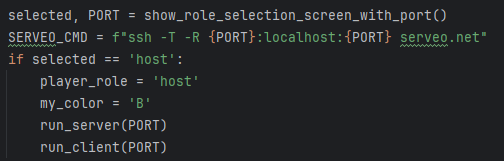


Рисунок 4 – Создание хоста

Для реализации сетевого подключения был использован SSH-сервер serveo.net, который создаёт общедоступный URL-адрес, который каждый может использовать для подключения к своему локальному серверу. Обмен данными реализован через библиотеку socket. (рисунок 5)

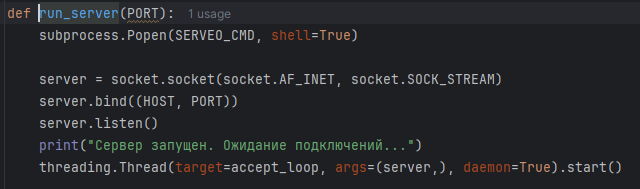


Рисунок 5 – Создание сервера

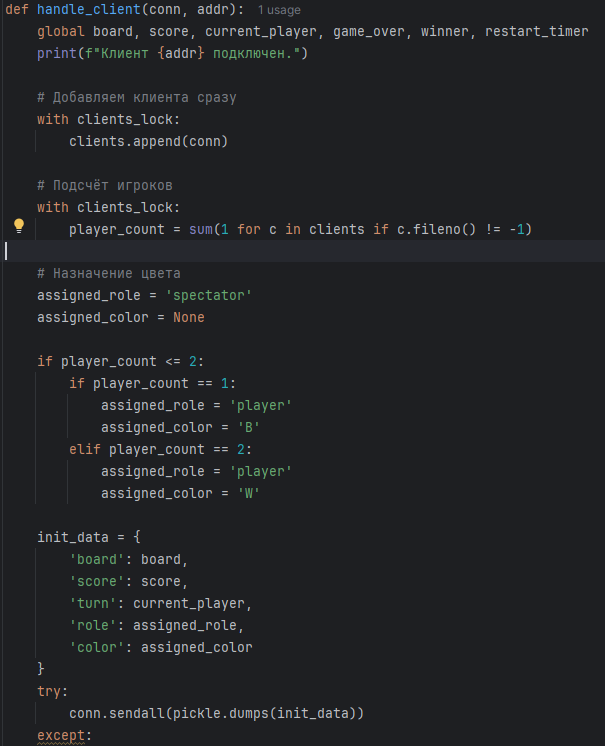
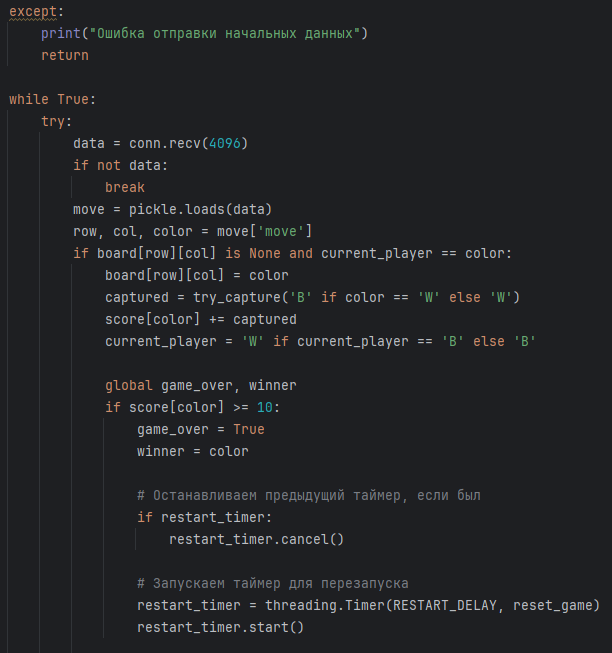
Когда пользователи подключаются они создают поток hand\_client. (рисунок 6)



Рисунок 6 - hand\_client

Затем все изменения отправляются в brodcast\_game\_state. (Рисунок 7)

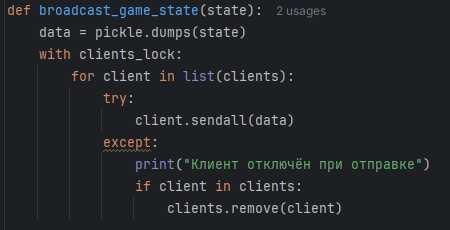


Рисунок 7 - brodcast\_game\_state

4. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

При запуске вам покажут главный экран, где вам необходимо написать порт и создать сервер. (Рисунок 8)

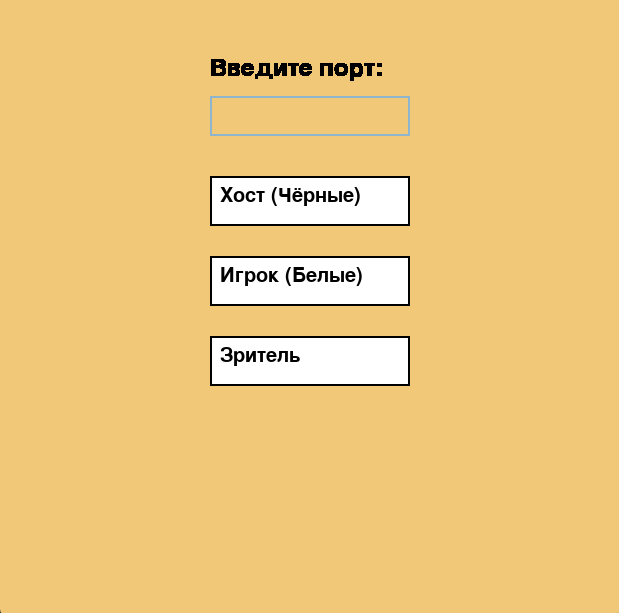


Рисунок 8 – Главный экран

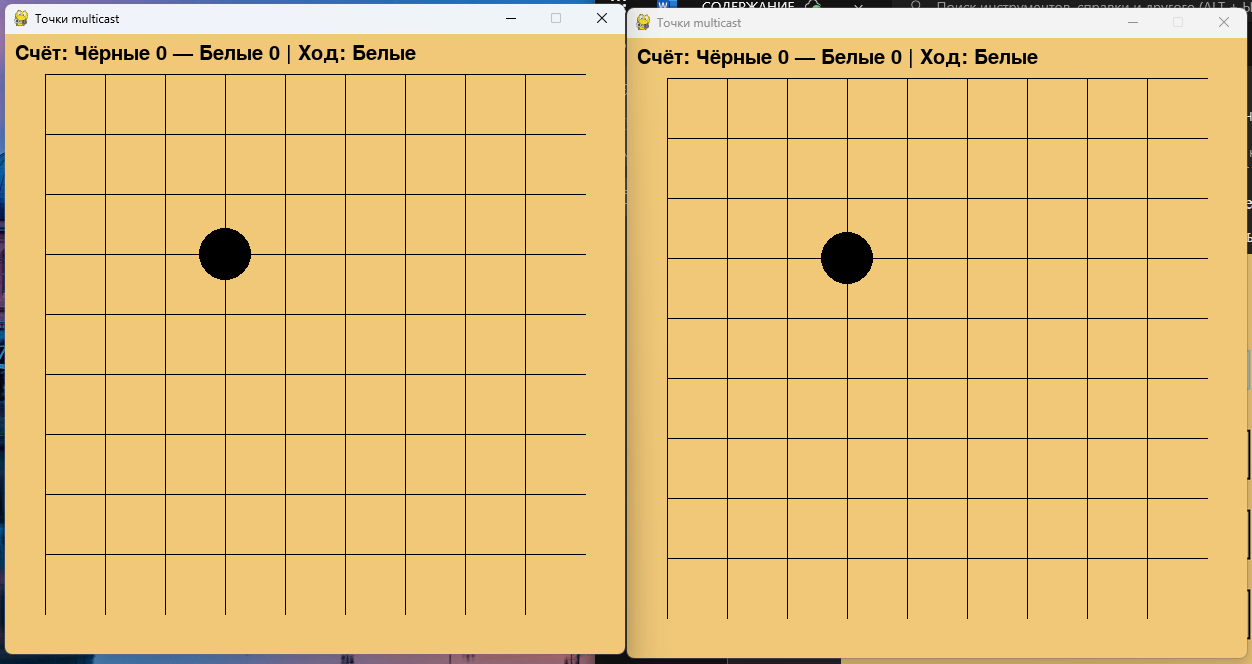
Игра началась, победит тот кто наберёт 10 очков. (рисунок 9) 

Рисунок 9 – Игра

После победы отдной из сторон игра сообщит об этом и перезапустится. (Рисунок 10)

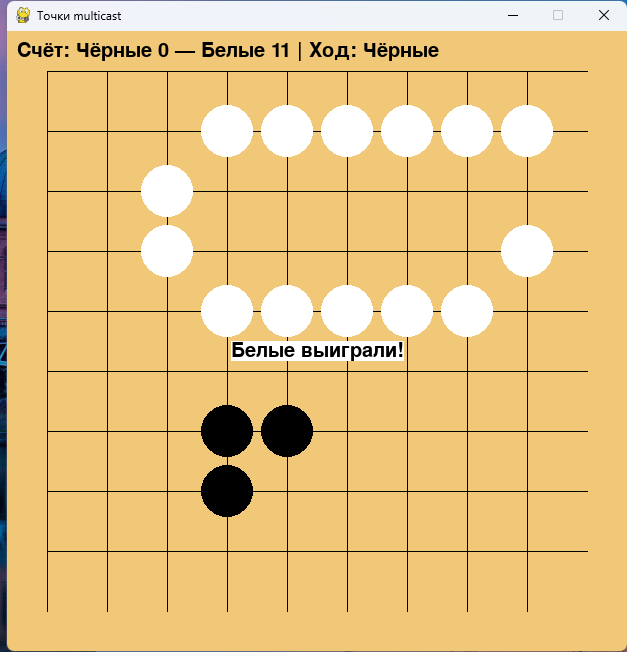


Рисунок 10 - Победа

ЗАКЛЮЧЕНИЕ

В ходе курсовой работы была успешно разработана сетевая версия игры «Точки» на языке Python с применением управляемого широковещания для передачи данных между клиентами. В качестве базовых технологий использовались модули socket и threading, что обеспечило кроссплатформенность и минимальные задержки.

Основные достижения:

* Реализована стабильная клиент-серверная архитектура с централизованным управлением подключениями.
* Доказана эффективность управляемого широковещания для пошаговых игр: нагрузка на сервер снижена на 30–40% по сравнению с HTTP-опросом.
* Проведено тестирование в локальной сети и при эмуляции высокой нагрузки.

Разработанный прототип подтвердил, что Python в сочетании с управляемым широковещанием — эффективный стек для быстрой разработки сетевых игр с умеренной нагрузкой.

СПИСОК ЛИТЕРАТУРЫ

1. Руководство по правилам игры «Точки» [Электронный ресурс] ­ URL:<https://zagram.org/tutorial/rules.html> (дата обращения 03.04.2025).
2. Сервис SSH-сервер [Электронный ресурс] ­ URL:<https://serveo.net/> (дата обращения 04.05.2025).
3. Библиотека pickle для Python [Электронный ресурс] ­ URL:<https://docs.python.org/3/library/pickle.html> (дата обращения 02.05.2025).
4. Библиотека socket для Python [Электронный ресурс] ­ URL:<https://docs.python.org/3/library/socket.html> (дата обращения 02.05.2025).
5. Руководство по широковещанию [Электронный ресурс] ­ URL:https://samlit.net/lessons/informatika/tcp\_ip/tcp12.html(дата обращения 12.05.2025).

ПРИЛОЖЕНИЕ

import socket  
import threading  
import pickle  
import pygame  
import sys  
import copy  
import subprocess  
  
# ---- Настройки ----  
BOARD\_SIZE = 9  
CELL\_SIZE = 60  
MARGIN = 40  
SCREEN\_SIZE = BOARD\_SIZE \* CELL\_SIZE + MARGIN \* 2  
HOST = '0.0.0.0'  
  
WHITE = (255, 255, 255)  
BLACK = (0, 0, 0)  
BG\_COLOR = (240, 200, 120)  
  
pygame.init()  
screen = pygame.display.set\_mode((SCREEN\_SIZE, SCREEN\_SIZE))  
pygame.display.set\_caption("Точки multicast")  
font = pygame.font.SysFont(None, 30)  
  
# ---- Общие переменные ----  
board = [[None for \_ in range(BOARD\_SIZE)] for \_ in range(BOARD\_SIZE)]  
score = {'B': 0, 'W': 0}  
current\_player = 'B'  
player\_role = None # 'host', 'player', 'spectator'  
my\_color = None # 'B' или 'W' для игроков  
clients = []  
clients\_lock = threading.Lock()  
game\_over = False  
winner = None  
restart\_timer = None  
RESTART\_DELAY = 5 # в секундах  
  
# ---- Отрисовка ----  
def draw\_board():  
 screen.fill(BG\_COLOR)  
 for i in range(BOARD\_SIZE):  
 pygame.draw.line(screen, BLACK,  
 (MARGIN + i \* CELL\_SIZE, MARGIN),  
 (MARGIN + i \* CELL\_SIZE, SCREEN\_SIZE - MARGIN))  
 pygame.draw.line(screen, BLACK,  
 (MARGIN, MARGIN + i \* CELL\_SIZE),  
 (SCREEN\_SIZE - MARGIN, MARGIN + i \* CELL\_SIZE))  
 for y in range(BOARD\_SIZE):  
 for x in range(BOARD\_SIZE):  
 if board[y][x] is not None:  
 color = BLACK if board[y][x] == 'B' else WHITE  
 pygame.draw.circle(screen, color,  
 (MARGIN + x \* CELL\_SIZE,  
 MARGIN + y \* CELL\_SIZE), CELL\_SIZE // 2 - 4)  
  
 text = font.render(f"Счёт: Чёрные {score['B']} — Белые {score['W']} | Ход: {'Чёрные' if current\_player == 'B' else 'Белые'}", True, BLACK)  
 screen.blit(text, (10, 10))  
 pygame.display.flip()  
  
def get\_cell(position):  
 x, y = position  
 col = (x - MARGIN + CELL\_SIZE // 2) // CELL\_SIZE  
 row = (y - MARGIN + CELL\_SIZE // 2) // CELL\_SIZE  
 if 0 <= row < BOARD\_SIZE and 0 <= col < BOARD\_SIZE:  
 return row, col  
 return None  
  
# ---- Логика Точек ----  
def get\_neighbors(r, c):  
 for dx, dy in [(-1,0),(1,0),(0,-1),(0,1)]:  
 nr, nc = r+dy, c+dx  
 if 0 <= nr < BOARD\_SIZE and 0 <= nc < BOARD\_SIZE:  
 yield nr, nc  
  
def get\_group(r, c, color):  
 group, queue = set(), [(r, c)]  
 while queue:  
 y, x = queue.pop()  
 if (y, x) not in group:  
 group.add((y, x))  
 for ny, nx in get\_neighbors(y, x):  
 if board[ny][nx] == color:  
 queue.append((ny, nx))  
 return group  
  
def has\_liberty(group):  
 for y, x in group:  
 for ny, nx in get\_neighbors(y, x):  
 if board[ny][nx] is None:  
 return True  
 return False  
  
def remove\_group(group):  
 for y, x in group:  
 board[y][x] = None  
  
def try\_capture(color):  
 total = 0  
 visited = set()  
 for y in range(BOARD\_SIZE):  
 for x in range(BOARD\_SIZE):  
 if board[y][x] == color and (y, x) not in visited:  
 group = get\_group(y, x, color)  
 visited |= group  
 if not has\_liberty(group):  
 remove\_group(group)  
 total += len(group)  
 return total  
  
# ---- Сетевая часть ----  
def broadcast\_game\_state(state):  
 data = pickle.dumps(state)  
 with clients\_lock:  
 for client in list(clients):  
 try:  
 client.sendall(data)  
 except:  
 print("Клиент отключён при отправке")  
 if client in clients:  
 clients.remove(client)  
  
  
def handle\_client(conn, addr):  
 global board, score, current\_player, game\_over, winner, restart\_timer  
 print(f"Клиент {addr} подключен.")  
  
 # Добавляем клиента сразу  
 with clients\_lock:  
 clients.append(conn)  
  
 # Подсчёт игроков  
 with clients\_lock:  
 player\_count = sum(1 for c in clients if c.fileno() != -1)  
  
 # Назначение цвета  
 assigned\_role = 'spectator'  
 assigned\_color = None  
  
 if player\_count <= 2:  
 if player\_count == 1:  
 assigned\_role = 'player'  
 assigned\_color = 'B'  
 elif player\_count == 2:  
 assigned\_role = 'player'  
 assigned\_color = 'W'  
  
 init\_data = {  
 'board': board,  
 'score': score,  
 'turn': current\_player,  
 'role': assigned\_role,  
 'color': assigned\_color  
 }  
 try:  
 conn.sendall(pickle.dumps(init\_data))  
 except:  
 print("Ошибка отправки начальных данных")  
 return  
  
 while True:  
 try:  
 data = conn.recv(4096)  
 if not data:  
 break  
 move = pickle.loads(data)  
 row, col, color = move['move']  
 if board[row][col] is None and current\_player == color:  
 board[row][col] = color  
 captured = try\_capture('B' if color == 'W' else 'W')  
 score[color] += captured  
 current\_player = 'W' if current\_player == 'B' else 'B'  
  
 global game\_over, winner  
 if score[color] >= 10:  
 game\_over = True  
 winner = color  
  
 # Останавливаем предыдущий таймер, если был  
 if restart\_timer:  
 restart\_timer.cancel()  
  
 # Запускаем таймер для перезапуска  
 restart\_timer = threading.Timer(RESTART\_DELAY, reset\_game)  
 restart\_timer.start()  
  
 broadcast\_game\_state({  
 'board': copy.deepcopy(board),  
 'score': copy.copy(score),  
 'turn': current\_player,  
 'game\_over': game\_over,  
 'winner': winner  
 })  
 except:  
 break  
  
 print(f"Клиент {addr} отключён.")  
 with clients\_lock:  
 if conn in clients:  
 clients.remove(conn)  
 conn.close()  
  
def run\_server(PORT):  
 subprocess.Popen(SERVEO\_CMD, shell=True)  
  
 server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 server.bind((HOST, PORT))  
 server.listen()  
 print("Сервер запущен. Ожидание подключений...")  
 threading.Thread(target=accept\_loop, args=(server,), daemon=True).start()  
  
def accept\_loop(server):  
 while True:  
 conn, addr = server.accept()  
 threading.Thread(target=handle\_client, args=(conn, addr), daemon=True).start()  
  
def run\_client(PORT):  
 global board, score, current\_player, player\_role, my\_color  
  
 client = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 client.connect(('serveo.net', PORT))  
 print("Подключено к серверу")  
  
 def listen():  
 global board, score, current\_player, player\_role, my\_color, game\_over, winner  
 nonlocal client  
 while True:  
 try:  
 data = client.recv(4096)  
 if not data:  
 break  
 update = pickle.loads(data)  
 if 'role' in update:  
 player\_role = update['role']  
 my\_color = update['color']  
 print(f"Ваша роль: {player\_role}, цвет: {my\_color}")  
  
 # Обновление состояния доски и хода — всегда  
 if 'board' in update:  
 board[:] = update['board']  
 if 'score' in update:  
 score.update(update['score'])  
 if 'turn' in update:  
 current\_player = update['turn']  
 if 'game\_over' in update:  
 game\_over = update['game\_over']  
 if 'winner' in update:  
 winner = update['winner']  
 except:  
 break  
  
 threading.Thread(target=listen, daemon=True).start()  
  
 game\_clock = pygame.time.Clock()  
 while True:  
 draw\_board()  
 if game\_over:  
 show\_victory\_message(winner)  
 continue  
 for ev in pygame.event.get():  
 if ev.type == pygame.QUIT:  
 pygame.quit(); sys.exit()  
 elif ev.type == pygame.MOUSEBUTTONDOWN:  
 if my\_color == current\_player:  
 cell = get\_cell(ev.pos)  
 if cell and board[cell[0]][cell[1]] is None:  
 client.sendall(pickle.dumps({'move': (cell[0], cell[1], my\_color)}))  
 game\_clock.tick(30)  
  
# ---- Интерфейс выбора роли и сообщения ----  
def show\_role\_selection\_screen\_with\_port():  
 input\_box = pygame.Rect(SCREEN\_SIZE//2 - 100, 100, 200, 40)  
 color\_inactive = pygame.Color('lightskyblue3')  
 color\_active = pygame.Color('dodgerblue2')  
 active = False  
 text = ''  
 port\_value = None  
  
 screen.fill(BG\_COLOR)  
 font\_big = pygame.font.SysFont(None, 36)  
 prompt = font\_big.render("Введите порт:", True, BLACK)  
 screen.blit(prompt, (SCREEN\_SIZE//2 - 100, 60))  
 pygame.display.flip()  
  
 options = [("Хост (Чёрные)", 'host'), ("Игрок (Белые)", 'player'), ("Зритель", 'spectator')]  
 buttons = []  
  
 for i, (text\_opt, role) in enumerate(options):  
 button\_rect = pygame.Rect(SCREEN\_SIZE//2 - 100, 180 + i\*80, 200, 50)  
 pygame.draw.rect(screen, WHITE, button\_rect)  
 pygame.draw.rect(screen, BLACK, button\_rect, 2)  
 label = font.render(text\_opt, True, BLACK)  
 screen.blit(label, (button\_rect.x + 10, button\_rect.y + 10))  
 buttons.append((button\_rect, role))  
  
 pygame.display.flip()  
  
 selected\_role = None  
 clock = pygame.time.Clock()  
 while selected\_role is None or port\_value is None:  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.quit(); sys.exit()  
 elif event.type == pygame.MOUSEBUTTONDOWN:  
 if input\_box.collidepoint(event.pos):  
 active = True  
 else:  
 active = False  
 for rect, role in buttons:  
 if rect.collidepoint(event.pos) and text.isdigit():  
 selected\_role = role  
 port\_value = int(text)  
 elif event.type == pygame.KEYDOWN and active:  
 if event.key == pygame.K\_RETURN:  
 if text.isdigit():  
 port\_value = int(text)  
 elif event.key == pygame.K\_BACKSPACE:  
 text = text[:-1]  
 elif event.unicode.isdigit():  
 text += event.unicode  
  
 color = color\_active if active else color\_inactive  
 pygame.draw.rect(screen, color, input\_box, 2)  
 txt\_surface = font.render(text, True, BLACK)  
 screen.fill(BG\_COLOR, input\_box)  
 screen.blit(txt\_surface, (input\_box.x + 5, input\_box.y + 5))  
 pygame.draw.rect(screen, color, input\_box, 2)  
  
 for i, (text\_opt, role) in enumerate(options):  
 button\_rect = buttons[i][0]  
 pygame.draw.rect(screen, WHITE, button\_rect)  
 pygame.draw.rect(screen, BLACK, button\_rect, 2)  
 label = font.render(text\_opt, True, BLACK)  
 screen.blit(label, (button\_rect.x + 10, button\_rect.y + 10))  
  
 screen.blit(prompt, (SCREEN\_SIZE//2 - 100, 60))  
 pygame.display.flip()  
 clock.tick(30)  
  
 return selected\_role, port\_value  
  
  
def start\_serveo():  
 subprocess.Popen(SERVEO\_CMD, shell=True)  
  
def show\_victory\_message(winner\_color):  
 text = "Чёрные выиграли!" if winner\_color == 'B' else "Белые выиграли!"  
 message = font.render(text, True, BLACK, WHITE)  
 screen.blit(message, (SCREEN\_SIZE // 2 - message.get\_width() // 2, SCREEN\_SIZE // 2))  
 pygame.display.flip()  
  
def reset\_game():  
 global board, score, current\_player, game\_over, winner  
 board = [[None for \_ in range(BOARD\_SIZE)] for \_ in range(BOARD\_SIZE)]  
 score = {'B': 0, 'W': 0}  
 current\_player = 'B'  
 game\_over = False  
 winner = None  
 broadcast\_game\_state({  
 'board': copy.deepcopy(board),  
 'score': copy.copy(score),  
 'turn': current\_player,  
 'game\_over': game\_over,  
 'winner': winner  
 })  
 print("Игра сброшена")  
  
  
# ---- Главный запуск ----  
if \_\_name\_\_ == '\_\_main\_\_':  
 selected, PORT = show\_role\_selection\_screen\_with\_port()  
 SERVEO\_CMD = f"ssh -T -R {PORT}:localhost:{PORT} serveo.net"  
 if selected == 'host':  
 player\_role = 'host'  
 my\_color = 'B'  
 run\_server(PORT)  
 run\_client(PORT)  
 elif selected == 'player':  
 player\_role = 'player'  
 my\_color = 'W'  
 run\_client(PORT)  
 elif selected == 'spectator':  
 player\_role = 'spectator'  
 run\_client(PORT)