Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: «Компьютерные системы и сети»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему:

**Сетевая игра "Крестики-нолики"**

БГУИР КР 6 - 05 - 06 12 01 029 ПЗ

Студент Позняк В.С.

Руководитель Болтак С.В.

Минск 2025

Учреждение образования  
«Белорусский государственный университет информатики  
 и радиоэлектроники»  
 УТВЕРЖДАЮ  
 Заведующий кафедрой ПОИТ   
  *–––––––––––* (подпись)  
  *–––––––*2025 г.

ЗАДАНИЕ  
по курсовому проектированию  
Студенту    Позянку Валерию Сергеевичу–––––––––––––––––––

1. Тема работы   Сетевая игра «Крестики-нолики»––––––––––––––––––  
2. Срок сдачи студентом законченной работы––27.05.2025 г.–––     
3. Исходные данные к работе   *Программа реализует функционал пользователя. Язык программирования Python, среда разработки PyCharm 2025. Вид приложения – Desktop-приложение, парадигма программирования – ООП. Способ организации данных – классы, способ хранения данных – файлы. Оформление кода должно соответствовать «Robert Martin Clear Code». Текст пояснительной записки оформляется по стандарту СТП 01–2024*  
4. Содержание расчётно-пояснительной записки (перечень вопросов, которые подлежат разработке)  
*Титульный лист––––––––––––––––––––––––––––––––––––           –– ––*  
*Задание по курсовой работе–––––––––– ––––––––––––––––           ––––*  
*Содержание ––––––––––––––––\_\_\_\_\_\_\_\_\_\_\_\_\_––––––––––           ––––*   
*Введение ––––––––––––––––\_\_\_\_\_\_\_\_\_\_\_\_\_––––––––––            –––––*  
*1 Анализ предметной области––––––––––––––––––––––– ––––––––*  
*1.1 Обзор аналогов ––– ––––––            –––вммкмкмкмкккииииии*  
*1.2 Постановка задачи–––– ––––– –             –––*  
*2 Проектирование программного средства––––––––––––––––––––––––*  
*2.1 Структура программы ––––––––– ––––––––*  
*2.2 Проектирование интерфейса программного средства –––––––––*  
*2.3 Проектирование функционала программного средства    – ––*   
*3 Разработка программного средства –\_\_\_\_\_\_\_\_\_\_\_\_\_.*   
*3.1* *Выбор технологий и инструментов\_ \_   ––––\_\_\_\_\_\_\_\_\_\_\_ \_\_\_. \_\_*  
*3.2 Реализация игровой логики ( .п.*   
*3.3 Работа с сокетами и сетью ( .*   
*3.4 Обработка пользовательского интерфейса*   
*4 Тестирование программного средства ––––––           –––– ––––––– 5* *Руководство пользователя \_\_   ––––\_\_\_\_\_\_\_\_\_\_. \_\_\_\_\_\_\_\_\_\_\_*  
*5.1 Запуск и настройка\_\_\_\_\_\_\_\_\_\_\_\_ \_. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   ––––––*  
*5.1 Игровой процесс \_. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   ––––––*  
*Заключение  \_\_\_\_\_\_\_\_\_\_\_\_\_\_. \_\_\_\_\_\_\_\_\_\_\_\_\_\_*  
*Список используемых источников*   
*Приложение А: Листинг кода с комментариями––––––– –––––––––––*

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)

1. Схема алгоритма работы системы

6. Консультант по курсовой работе

Болтак С.В.

7. Дата выдачи задания 15.02.2025 г.–––––––––––––––––––––––   –

8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и процентом от общего объёма работы):

раздел 1,2 к 15.03.2025 – 15 % готовности работы;

разделы 3, 4 к 15.03.2025 – 30 % готовности работы;

разделы 5, 6 к 15.03.2025 – 60 % готовности работы;

раздел 7, 8, 9 к 15.03.2025 – 90 % готовности работы;

оформление пояснительной записки и графического материала к 27.05.2025 – 100 % готовности работы.

Защита курсового проекта с 28.05.2025 по 7.06.2025 г.–––––––––––––––––––––

РУКОВОДИТЕЛЬ–––––– С.В.Болтак

(подпись)

Задание принял к исполнению –––\_\_\_\_–– В.С.Позняк    15.02.2025 г.

**СОДЕРЖАНИЕ**

Введение.............................................................................................................................5

1 Анализ предметной области..........................................................................................6

1.1 Обзор аналогов.............................................................................................................6

1.2 Постановка задачи.......................................................................................................7

2 Проектирование программного средства.....................................................................9

2.1 Структура программы.................................................................................................9

2.2 Проектирование интерфейса программного средства.............................................9

2.3 Проектирование функционала программного средства........................................12

3 Разработка программного средства............................................................................15

3.1 Выбор технологий и инструментов.........................................................................15

3.2 Реализация игровой логики......................................................................................15

3.3 Работа с сокетами и сетью........................................................................................16

3.4 Обработка пользовательского интерфейса..............................................................17

4 Тестирование программного средства........................................................................18

5 Руководство пользователя...........................................................................................20

5.1 Запуск и настройка....................................................................................................20

5.2 Игровой процесс........................................................................................................20

Заключение.......................................................................................................................21

Список использованных источников.............................................................................22

Приложение А. Исходный код программы...................................................................23

# ВВЕДЕНИЕ

В последние годы компьютерные игры приобретают всё большую популярность не только в качестве развлечений, но и как инструменты обучения. «Крестики-нолики» — это классическая игра, которая часто используется для изучения базовых принципов программирования, алгоритмов и сетевого взаимодействия.  
 Цель данного курсового проекта — разработка сетевой версии игры «Крестики-нолики» на языке Python с использованием библиотеки Pygame для графического интерфейса и модуля socket для обеспечения сетевого взаимодействия между двумя игроками. Игра должна обеспечивать функционал создания и подключения к игровой сессии, обмен ходами, отображение результатов партии и возможность предложения рематча. Также должна быть реализована возможность локальной игры между двумя игроками без сетевого подключения.  
 Основные задачи проекта:  
1. Проанализировать существующие аналоги сетевых игр «Крестики-нолики".  
2. Спроектировать архитектуру приложения, включающую клиентскую и серверную части.  
3. Разработать графический интерфейс с помощью Pygame: отрисовка поля, кнопок, текста, отображение ходов и линии победы.  
4. Реализовать сетевую часть на основе TCP-сокетов: создание сервера, подключение клиента, обмен сообщениями о ходах, запросах рематча и выходе.  
5. Протестировать приложение: проверить корректность обмена данными, логику определения победы/ничьей, работу интерфейса и функционала рематча.   
 Практическая значимость проекта заключается в закреплении знаний по работе с Pygame, освоении основ сетевого программирования на Python и получении готового решения для организации сетевой игры «Крестики-нолики».

# АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

## Обзор аналогов

Сетевые реализации игры «Крестики-нолики» широко распространены в образовательных целях:  
1. Веб-версии на HTML5/JavaScript с использованием WebSocket (например, playtictactoe.org представленный на рисунке 1.1).

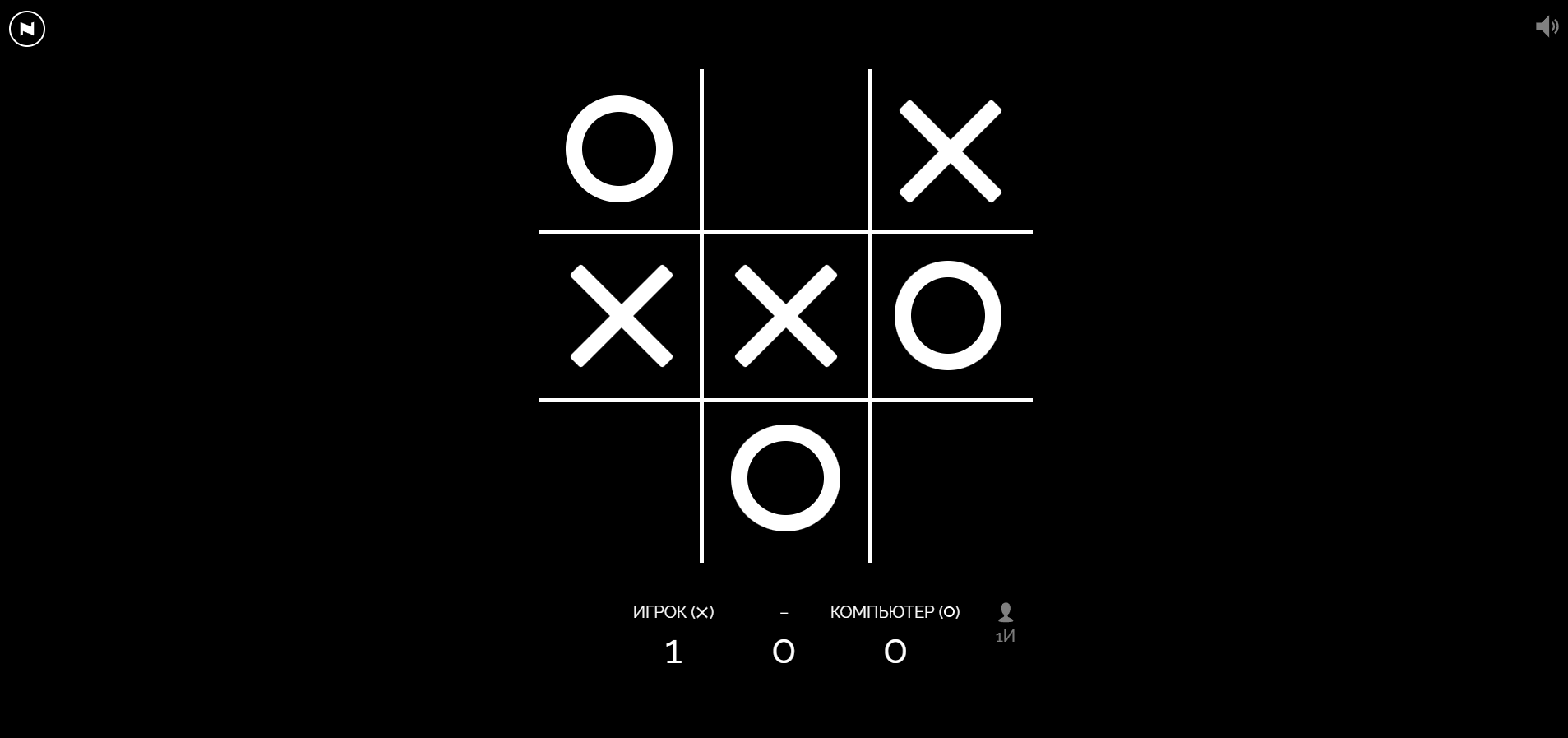


Рисунок 1.1 - playtictactoe.org

2. Локальные приложения на Java с сокетами, демонстрирующие основы клиент-серверного взаимодействия.  
3. Открытые примеры на Python и Pygame, в которых реализована только одиночная игра или базовая сетевая функциональность без рематча и обработки ошибок.  
 Разрабатываемая игра отличается:  
- Минимальными системными требованиями (требуется только Python и Pygame).  
- Поддержкой полной сетевой игры с сервером и клиентом, обработкой ошибок подключения и таймаутов.  
- Возможностью предлагать рематч и корректно обрабатывать отказ или выход соперника.  
- Интуитивным интерфейсом с графикой Pygame и четким разделением состояний приложения.

## 1.2 Постановка задачи

Цель проекта — создать сетевую версию игры «Крестики-нолики» с использованием Python, Pygame и сокетов.  
Основные требования:  
1. Главное меню с кнопками «Создать игру» и «Подключиться».  
 - При создании игры запускается сервер, ожидающий подключения клиента (таймаут 10 секунд).  
 - При подключении клиента осуществляется попытка соединения (таймаут 5 секунд).  
 - При ошибке (порт занят, отказ подключения) выводится сообщение об ошибке.  
2. После установления соединения отображается игровое поле 3×3 (600×600 пикселей).  
 - Сервер играет «X» и ходит первым, клиент — «O».  
3. Обработка кликов по клеткам: координаты рассчитываются как (x // CELL\_SIZE, y // CELL\_SIZE).  
 - При ходе отправляется сообщение «move:x,y».  
 - При получении хода на стороне соперника вызывается проверка победы.  
4. Логика победы:  
 - Проверяются горизонтали, вертикали, диагонали.  
 - При победе рисуется красная линия (толщина 10 пикселей) через выигрышную комбинацию.  
 - Выводятся сообщения «Вы выиграли!», «Вы проиграли!» или «Ничья!».  
5. После окончания партии появляются кнопки «Играть снова» и «Выход в меню».  
 - При «Играть снова» отправляется «rematch:request»; получатель видит надпись «Соперник предлагает реванш».  
 - При согласии отправляется «rematch:accept» и происходит сброс состояния, игра начинается заново.  
 - При отказе («rematch:reject») выводится сообщение «Соперник отказал в реванше».  
 - При нажатии «Выход в меню» отправляется «exit», оба игрока возвращаются в меню.  
6. Обработка ошибок:  
 - Если порт сервера занят, выводится «Не удалось создать сервер (уже запущен?)».  
 - Если потеряно соединение или соперник вышел, выводится сообщение «Соединение разорвано» или «Соперник вышел из игры».  
7. Интерфейс реализован в Pygame, размеры окна 600×700, используемый шрифт — стандартный системный.

8. Поддержка локального режима: два игрока поочередно делают ходы на одном устройстве. После окончания партии можно начать новую или выйти в главное меню.

**2 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

## 2.1 Структура программы

Программа собрана в одном файле main.py, в котором реализованы все функции:  
- draw\_menu(), draw\_waiting(), draw\_board(), draw\_status(), draw\_rematch\_ui().  
- handle\_network(), send\_move(), send\_rematch\_request(), send\_rematch\_response(), send\_exit().  
- receive\_moves() для фонового приема сообщений.  
- Функции check\_winner() и reset() реализуют логику игры.  
- Глобальные переменные хранят состояние игры и сетевые флаги.  
Добавлена глобальная переменная game\_mode, определяющая режим игры: 'local' — игра вдвоем на одном ПК, 'network' — по сети. Также переменная role получает значение None в локальном режиме.

## 2.2 Проектирование интерфейса программного средства

2.2.1 Главное меню(рисунок 2.1):  
- Заголовок «Крестики-нолики» по центру верхней части окна.  
- Кнопки «Создать игру», «Подключиться» и «Локальная игра» располагаются по центру.  
- При ошибке (порт занят) выводится сообщение «Не удалось создать сервер (уже запущен?)» под кнопками.

  
Рисунок 2.1 – Главное меню

2.2.2 Окно ожидания:  
- Отображается надпись «Ожидание подключения...» по центру окна(рисунок 2.2).  
- При возникновении ошибки (невозможно подключиться) выводится текст «Ошибка подключения: ...».



Рисунок 2.2 – меню ожидания

2.2.3 Игровое окно(рисунок 2.3):  
- Размер игрового поля 600×600 пикселей, разделено на 3×3 клетки.   
- Каждая клетка 200×200 пикселей.   
- При наличии победы рисуется красная линия толщиной 10 через центр выигрышной комбинации.  
- Снизу под полем выводится надпись «Ваш ход» или «Ход соперника»

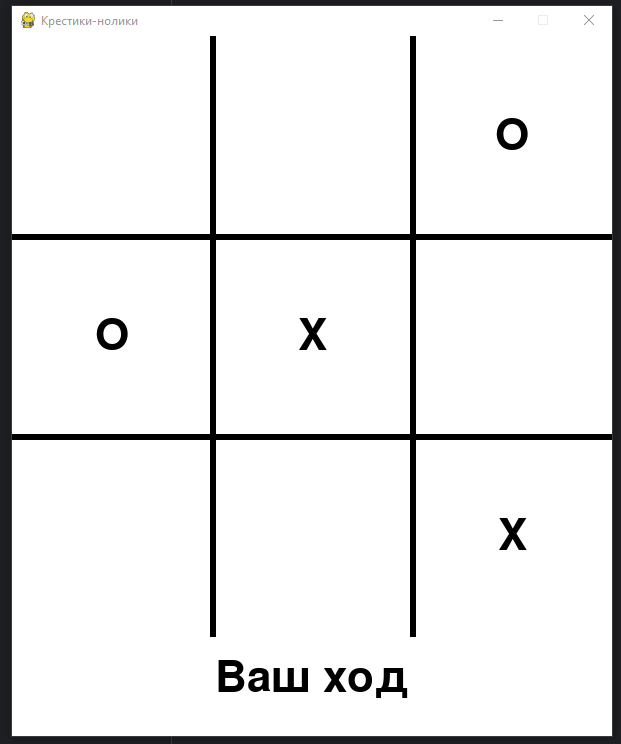


Рисунок 2.3 – игровое поле

2.2.4 Конец игры и рематч(Рисунок 2.4):  
- После завершения партии под полем отображается результат («Вы выиграли!», «Ничья!» или «Вы проиграли!»), и выводятся кнопки «Играть снова» и «Выход в меню»).  
- Если соперник предложил рематч, над кнопками появляется надпись «Соперник предлагает реванш»).

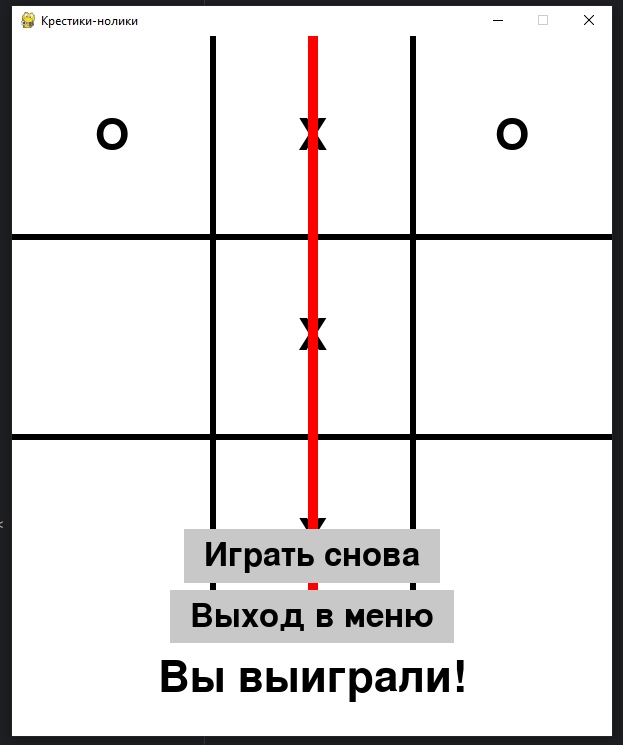


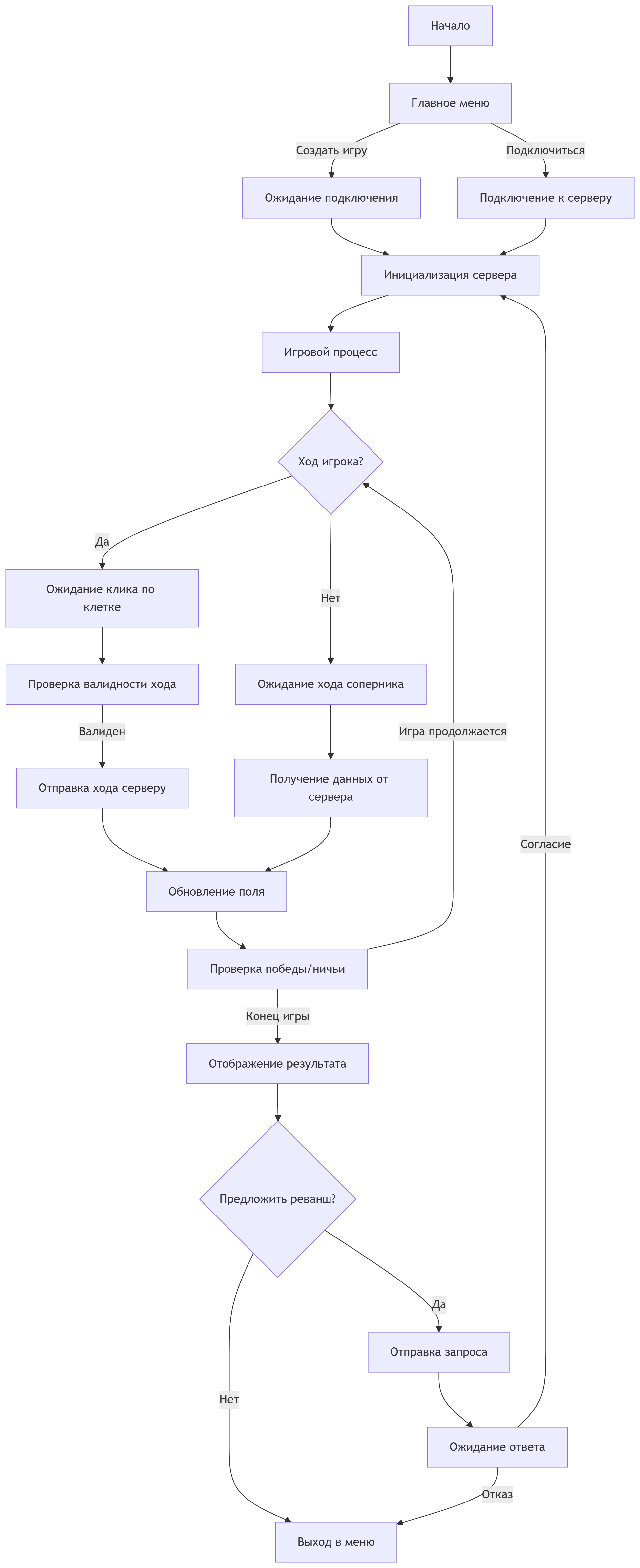
Рисунок 2.4 – окончание игры

## 2.3 Проектирование функционала программного средства

Состояния приложения (state):  
1. menu — главное меню.  
2. game — игровое окно.  
 - draw\_board(): отрисовка поля и фигур.  
 - draw\_status(): отображение текста хода.  
 - draw\_rematch\_ui(): кнопки и предложение рематча.  
3. waiting — ожидание подключения.  
4. end — состояние окончания партии  
  
Сетевые сообщения:  
- move:x,y — передача координат хода.  
- rematch:request — запрос рематча.  
- rematch:accept / rematch:reject — ответ на запрос рематча.  
- exit — выход в меню и разрыв соединения.

В режиме 'local' ходят поочередно два игрока. Переменная current\_player переключается после каждого хода. Сетевые функции (send\_move, receive\_moves, conn) не используются  
  
Обработка ошибок:  
- Если bind() выбрасывает OSError, выводится «Не удалось создать сервер (уже запущен?)».  
- Если connect() выбрасывает ConnectionRefusedError, через 5 секунд выходит ошибка подключения.  
- При обрыве соединения (recv() возвращает пустую строку) выводится «Соединение разорвано» и оба игрока возвращаются в меню.

Алгоритм работы сетевой части игры представлен на рисунке 2.5 ниже:



**3 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА**

## 3.1 Выбор технологий и инструментов

Для разработки использованы:  
- Python 3.x — основной язык, простота синтаксиса.  
- Pygame — библиотека для 2D-графики и обработки событий.  
- socket — модуль для работы с TCP-сокетами.  
- threading — для запуска фонового приема сообщений.   
  
IDE: PyCharm Community Edition.

## 3.2 Реализация игровой логики

Игровая логика реализована в файле main.py:  
1. Инициализация Pygame:  
 - pygame.init(), создание экрана (WIDTH=600, HEIGHT=700), установка заголовка.  
 - Загрузка шрифтов: font размером 64, large\_font размером 96, small\_font размером 48.  
2. Глобальные переменные:  
 - grid — матрица 3×3 для хранения состояний клеток (None, 'X' или 'O').  
 - current\_player — символ текущего игрока ('X' или 'O').  
 - game\_over, winner, my\_turn — флаги состояния игры.  
3. Функция check\_winner():  
 - Проверка всех возможных победных комбинаций по горизонталям, вертикалям и диагоналям.  
 - В случае победы запоминается символ победителя и координаты линии для отрисовки win\_line.  
 - Проверка на ничью: если все клетки заполнены без победителя, устанавливается winner='Ничья'.  
4. Функция reset():  
 - Сброс состояния: очистка grid, сброс флагов game\_over, winner, win\_line.  
 - Сброс флагов реванша waiting\_for\_rematch, rematch\_offered\_by\_opponent, rematch\_accepted.  
 - Установка my\_turn=True для сервера, False для клиента.  
5. В локальном режиме my\_turn всегда True, а current\_player меняется после каждого хода. Проверка победы и ничьей выполняется сразу после хода, без сетевого обмена.

## 3.3 Работа с сокетами и сетью

Сетевая часть организована через TCP-сокеты:  
1. Функция handle\_network():  
 - Если role=='server':  
 • Создается сокет, устанавливается SO\_REUSEADDR;  
 • Производится s.bind((HOST, PORT)) и s.listen(1);  
 • Таймаут accept() установлен через s.settimeout(0.5) для предотвращения блокировки main-loop.  
 • В цикле пытается принять соединение, пока running и not connected.  
 • При успешном подключении устанавливаются connected=True и my\_turn=True.  
 • При OSError (порт занят) вызывается show\_error("Не удалось создать сервер (уже запущен?)").  
 - Если role=='client':  
 • Создается сокет и пытается подключиться к серверу в цикле с таймаутом 1 секунда.  
 • При успешном подключении connected=True, my\_turn=False.  
 • При ConnectionRefusedError или других сбоях выводится show\_error("Ошибка подключения: ...") и выход.  
2. Функции отправки сообщений:  
 - send\_move(x,y): отправляет байты вида "move:x,y".  
 - send\_rematch\_request(): отправляет "rematch:request".  
 - send\_rematch\_response(accepted): отправляет "rematch:accept" или "rematch:reject".  
 - send\_exit(): отправляет "exit" перед выходом.  
3. Фоновой поток receive\_moves():  
 - Пока running и connected:  
 • При получении "move:x,y" ставит в grid соответствующую фигуру, устанавливает my\_turn=True и проверяет победу.  
 • При получении "rematch:request" устанавливает флаги waiting\_for\_rematch=False, rematch\_offered\_by\_opponent=True.  
 • При получении "rematch:accept" вызывает reset() и сбрасывает флаги рематча.  
 • При получении "rematch:reject" сбрасывает флаги рематча и выводит сообщение об отказе.  
 • При получении "exit" выводится сообщение "Соперник вышел из игры" и вызывается return\_to\_menu().  
 - При ошибке сокета (отключение, исключение) выводится show\_error("Ошибка соединения: ...") и return\_to\_menu().

## 3.4 Обработка пользовательского интерфейса

Функции отрисовки и обработки интерфейса:  
1. draw\_menu():  
 - Отрисовка фона белым цветом, заголовка и кнопок.  
 - При наличии server\_error отображает его текст красным цветом под кнопками.  
2. draw\_waiting():  
 - Отображение надписи "Ожидание подключения...".  
 - Если server\_error не null, показывать ошибку подключения красным цветом ниже основной надписи.  
3. draw\_board():  
 - Отрисовка белого фона и сетки 3×3.  
 - Отображение крестиков и ноликов: метки по центру клетки.  
 - Если win\_line не None, рисовать красную линию толщиной 10 через выигрышную комбинацию.  
4. draw\_status():  
 - Вывод текста "Ваш ход" или "Ход соперника".  
 - В локальном режиме в нижней части экрана отображается надпись: «Ход крестиков» или «Ход ноликов» в зависимости от current\_player  
5. draw\_rematch\_ui():  
 - Отрисовка кнопок «Играть снова» и «Выход в меню».  
 - Если rematch\_offered\_by\_opponent=True, отображать текст «Соперник предлагает реванш».  
6. return\_to\_menu():  
 - Отправляет «exit», закрывает сокет, сбрасывает соединение, возвращает state в «menu», сбрасывает флаги.

# Тестирование программного средства

Тестирование проводилось вручную и включает следующие этапы:  
1. Проверка запуска сервера:  
 - Нажатие «Создать игру» при свободном порту: сервер запускается, отображается «Ожидание подключения...».  
 - Нажатие «Создать игру» при занятом порту: выводится «Не удалось создать сервер (уже запущен?)».  
2. Проверка подключения клиента:  
 - Нажатие «Подключиться» при запущенном сервере: клиент подключается, игра начинается.  
 - Нажатие «Подключиться» при отсутствии сервера: через 5 секунд выводится «Ошибка подключения: ...».  
3. Проверка игрового процесса:  
 - Расстановка крестиков и ноликов: при клике по свободной клетке на одном экране ставится крестик, а на втором экране появляется нолик.  
 - Чередование ходов: после хода игрока его кнопки блокируются, пока не придет ход соперника.  
 - Отображение надписи «Ваш ход» или «Ход соперника».  
4. Проверка победных условий:  
 - Проверка победы по горизонталям, вертикалям и диагоналям. При победе рисуется красная линия.  
 - Проверка ничьей: при заполнении всех клеток без победителя выводится «Ничья».  
5. Проверка рематча:  
 - При нажатии «Играть снова» в состоянии end отправляется «rematch:request» и отображается «Ожидание ответа...».  
 - При получении «rematch:request» второй игрок видит «Соперник предлагает реванш» и может согласиться.  
 - При согласии отправляется «rematch:accept» и игра сбрасывается.  
 - При отказе отправляется «rematch:reject» и отображается «Соперник отказал в рематче».  
6. Проверка выхода:  
 - При нажатии «Выход в меню» отправляется «exit»; оба игрока получают команду «exit» и возвращаются в меню.   
7. Тестирование локального режима:

- После нажатия «Локальная игра» запускается поле.

- При каждом клике игрока появляется соответствующий символ («X» или «O»).

- После победы или ничьей отображается правильный результат и предлагается начать новую игру или выйти в меню.

- При выборе «Играть снова» — поле очищается, игроки могут играть снова

# 5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

## 5.1 Запуск и настройка

1. Установите Python 3.7 или выше.  
2. Установите библиотеку Pygame:  
 pip install pygame  
3. Сохраните файл main.py с реализацией проекта в одной папке.  
4. Запустите игру двойным кликом или командой:  
 python main.py  
5. В главном меню:  
 - Нажмите «Создать игру» для запуска сервера. На экране появится «Ожидание подключения...».  
 - Нажмите «Подключиться» для подключения к серверу. Если сервер не запущен, через 5 секунд появится сообщение об ошибке.  
6. Для игры вдвоем на одном компьютере нажмите «Локальная игра». Сетевая настройка в этом случае не требуется.

## 5.2 Игровой процесс

1. После подключения оба игрока видят игровое поле 3×3:  
 - Сервер играет «X» и ходит первым. Клиент играет «O».  
 - Перед началом игры отображается «Ваш ход» (сервер) или «Ход соперника» (клиент).  
2. Чтобы сделать ход, кликните по нужной клетке. В поле соперника появится соответствующая фигура.  
3. Игра продолжается до тех пор, пока один из игроков не выстроит три фигуры подряд (горизонталь, вертикаль или диагональ). При победе рисуется красная линия через выигрышную комбинацию, и выводится «Вы выиграли!» или «Вы проиграли!».  
4. Если все клетки заполнены, но победитель не определен, отображается «Ничья!».  
5. После окончания игры появляются кнопки «Играть снова» и «Выход в меню».  
 - Нажав «Играть снова", отправляется запрос «rematch:request». Если соперник примет, игра начнется заново.  
 - Если соперник откажет, появится «Соперник отказал в реванше».  
 - Нажав «Выход в меню», оба игрока возвращаются в главное меню.

# ЗАКЛЮЧЕНИЕ

В рамках курсового проекта была выполнена разработка сетевой версии игры «Крестики-нолики» с использованием Python, библиотеки Pygame и модуля socket. Реализованы:  
- Создание сервера и обработка ошибок портов.  
- Подключение клиента с таймаутом и обработкой ошибок.  
- Графический интерфейс: отрисовка поля, фигур, текста и кнопок.  
- Игровая логика: проверка победы, ничьи, отрисовка линии победы.  
- Возможность рематча с корректной обработкой запросов и ответов.  
- Корректный выход в главное меню при нажатии «Выход» или потере соединения.  
- Дополнительно была реализована возможность локальной игры, что делает приложение универсальным: им можно пользоваться как в сети, так и без подключения.  
 Тестирование показало, что приложение надежно работает при запуске двух экземпляров на одном компьютере или в локальной сети. Разработанная игра может служить наглядным примером сетевого программирования и работы с Pygame.

# Список использованных источников

1. Python Documentation. https://docs.python.org/3/

2. Pygame Documentation. https://www.pygame.org/docs/

3. Socket Programming HOWTO. https://docs.python.org/3/howto/sockets.html

4. PEP 8 -- Style Guide for Python Code. https://peps.python.org/pep-0008/

# Приложение А. Исходный код программы

import pygame  
import socket  
import threading  
import time  
import sys  
  
# Настройки окна и игры  
WIDTH, HEIGHT = 600, 700  
CELL\_SIZE = 200  
WHITE = (255, 255, 255)  
BLACK = (0, 0, 0)  
LINE\_COLOR = (0, 0, 0)  
WIN\_LINE\_COLOR = (255, 0, 0)  
LINE\_WIDTH = 6  
FONT\_SIZE = 64  
HOST = '127.0.0.1'  
PORT = 5000  
  
# Состояние игры  
grid = [[None] \* 3 for \_ in range(3)]  
current\_player = 'X'  
game\_over = False  
winner = None  
my\_turn = False  
role = None # 'server', 'client' или None для локальной игры  
conn = None  
connected = False  
running = True  
win\_line = None  
  
# Реванш  
waiting\_for\_rematch = False # отправитель ждет  
rematch\_offered\_by\_opponent = False # получатель видит предложение  
rematch\_accepted = False  
  
# Ошибки  
server\_error = None # сообщение об ошибке сервера  
error\_time = 0  
  
# Режим игры  
game\_mode = None # 'network' или 'local'  
  
pygame.init()  
screen = pygame.display.set\_mode((WIDTH, HEIGHT))  
pygame.display.set\_caption("Крестики-нолики")  
font = pygame.font.SysFont(None, FONT\_SIZE)  
large\_font = pygame.font.SysFont(None, 96)  
small\_font = pygame.font.SysFont(None, 48)  
clock = pygame.time.Clock()  
  
  
def show\_error(msg):  
 global server\_error, error\_time  
 server\_error = msg  
 error\_time = time.time()  
  
  
# Отрисовка меню  
def draw\_menu():  
 screen.fill(WHITE)  
 title = large\_font.render("Крестики-нолики", True, BLACK)  
 screen.blit(title, title.get\_rect(center=(WIDTH // 2, HEIGHT // 2 - 200)))  
 create\_text = font.render("Создать игру", True, BLACK)  
 connect\_text = font.render("Подключиться", True, BLACK)  
 local\_text = font.render("Локальная игра", True, BLACK)   
  
 create\_rect = create\_text.get\_rect(center=(WIDTH // 2, HEIGHT // 2 - 120))  
 connect\_rect = connect\_text.get\_rect(center=(WIDTH // 2, HEIGHT // 2))  
 local\_rect = local\_text.get\_rect(center=(WIDTH // 2, HEIGHT // 2 + 120))   
  
 pygame.draw.rect(screen, (200, 200, 200), create\_rect.inflate(40, 40))  
 pygame.draw.rect(screen, (200, 200, 200), connect\_rect.inflate(40, 40))  
 pygame.draw.rect(screen, (200, 200, 200), local\_rect.inflate(40, 40))   
  
 screen.blit(create\_text, create\_rect)  
 screen.blit(connect\_text, connect\_rect)  
 screen.blit(local\_text, local\_rect) # Отображение текста  
  
 if server\_error and time.time() - error\_time < 3:  
 err = small\_font.render(server\_error, True, (200, 0, 0))  
 screen.blit(err, err.get\_rect(center=(WIDTH // 2, HEIGHT // 2 + 200)))  
  
 return create\_rect, connect\_rect, local\_rect # Возвращаем все три кнопки  
  
  
# Ожидание подключения  
def draw\_waiting():  
 screen.fill(WHITE)  
 text = font.render("Ожидание подключения...", True, BLACK)  
 screen.blit(text, text.get\_rect(center=(WIDTH // 2, HEIGHT // 2)))  
 if server\_error and time.time() - error\_time < 3:  
 err = small\_font.render(server\_error, True, (200, 0, 0))  
 screen.blit(err, err.get\_rect(center=(WIDTH // 2, HEIGHT // 2 + 100)))  
 pygame.display.flip()  
  
  
# Отрисовка поля и фигур  
def draw\_board():  
 screen.fill(WHITE)  
 for i in range(1, 3):  
 pygame.draw.line(screen, LINE\_COLOR, (0, i \* CELL\_SIZE), (WIDTH, i \* CELL\_SIZE), LINE\_WIDTH)  
 pygame.draw.line(screen, LINE\_COLOR, (i \* CELL\_SIZE, 0), (i \* CELL\_SIZE, CELL\_SIZE \* 3), LINE\_WIDTH)  
 for y in range(3):  
 for x in range(3):  
 if grid[y][x]:  
 mark = font.render(grid[y][x], True, BLACK)  
 screen.blit(mark, mark.get\_rect(center=(x \* CELL\_SIZE + CELL\_SIZE // 2,  
 y \* CELL\_SIZE + CELL\_SIZE // 2)))  
 if win\_line:  
 pygame.draw.line(screen, WIN\_LINE\_COLOR, \*win\_line, LINE\_WIDTH + 4)  
  
  
# Статус хода  
def draw\_status(msg):  
 text = font.render(msg, True, BLACK)  
 screen.blit(text, (WIDTH // 2 - text.get\_width() // 2, CELL\_SIZE \* 3 + 20))  
  
  
# Кнопки и предложение реванша  
def draw\_rematch\_ui():  
 rematch\_text = small\_font.render("Играть снова", True, BLACK)  
 exit\_text = small\_font.render("Выход в меню", True, BLACK)  
 rematch\_rect = rematch\_text.get\_rect(center=(WIDTH // 2, HEIGHT - 180))  
 exit\_rect = exit\_text.get\_rect(center=(WIDTH // 2, HEIGHT - 120))  
 pygame.draw.rect(screen, (200, 200, 200), rematch\_rect.inflate(40, 20))  
 pygame.draw.rect(screen, (200, 200, 200), exit\_rect.inflate(40, 20))  
 screen.blit(rematch\_text, rematch\_rect)  
 screen.blit(exit\_text, exit\_rect)  
 if rematch\_offered\_by\_opponent and game\_mode == 'network':  
 txt = small\_font.render("Соперник предлагает реванш", True, BLACK)  
 screen.blit(txt, txt.get\_rect(center=(WIDTH // 2, HEIGHT - 240)))  
 return rematch\_rect, exit\_rect  
  
  
# Проверка победы  
def check\_winner():  
 global winner, game\_over, win\_line  
 for i in range(3):  
 if grid[i][0] and grid[i][0] == grid[i][1] == grid[i][2]:  
 winner = grid[i][0]  
 win\_line = ((0, i \* CELL\_SIZE + CELL\_SIZE // 2), (WIDTH, i \* CELL\_SIZE + CELL\_SIZE // 2))  
 game\_over = True  
 return  
 if grid[0][i] and grid[0][i] == grid[1][i] == grid[2][i]:  
 winner = grid[0][i]  
 win\_line = ((i \* CELL\_SIZE + CELL\_SIZE // 2, 0), (i \* CELL\_SIZE + CELL\_SIZE // 2, CELL\_SIZE \* 3))  
 game\_over = True  
 return  
 if grid[0][0] and grid[0][0] == grid[1][1] == grid[2][2]:  
 winner = grid[0][0]  
 win\_line = ((0, 0), (CELL\_SIZE \* 3, CELL\_SIZE \* 3))  
 game\_over = True  
 return  
 if grid[0][2] and grid[0][2] == grid[1][1] == grid[2][0]:  
 winner = grid[0][2]  
 win\_line = ((CELL\_SIZE \* 3, 0), (0, CELL\_SIZE \* 3))  
 game\_over = True  
 return  
 if all(all(cell for cell in row) for row in grid):  
 winner = 'Ничья'  
 game\_over = True  
  
  
# Сброс игры  
def reset():  
 global grid, current\_player, game\_over, winner, my\_turn, win\_line  
 global waiting\_for\_rematch, rematch\_offered\_by\_opponent, rematch\_accepted  
 grid = [[None] \* 3 for \_ in range(3)]  
 current\_player = 'X'  
 game\_over = False  
 winner = None  
 win\_line = None  
 waiting\_for\_rematch = False  
 rematch\_offered\_by\_opponent = False  
 rematch\_accepted = False  
  
 # Для сетевой игры устанавливаем чей ход в зависимости от роли  
 if game\_mode == 'network':  
 my\_turn = (role == 'server')  
 else:  
 # Для локальной игры всегда начинает первый игрок  
 my\_turn = True  
  
  
# Сетевая часть  
def handle\_network():  
 global conn, connected, my\_turn, server\_error  
 if role == 'server':  
 s = socket.socket()  
 s.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)  
 try:  
 s.bind((HOST, PORT))  
 s.listen(1)  
 s.settimeout(0.5)  
 while running and not connected:  
 try:  
 conn, \_ = s.accept()  
 connected = True  
 my\_turn = True  
 except socket.timeout:  
 continue  
 except OSError as e:  
 show\_error(f"Ошибка сервера: {e}")  
 break  
 except OSError as e:  
 show\_error("Не удалось создать сервер (уже запущен?)")  
 else:  
 s = socket.socket()  
 while not connected and running:  
 try:  
 s.connect((HOST, PORT))  
 conn = s  
 connected = True  
 my\_turn = False  
 except ConnectionRefusedError:  
 time.sleep(1)  
 except Exception as e:  
 show\_error(f"Ошибка подключения: {e}")  
 break  
  
  
def send\_move(x, y):  
 if conn:  
 try:  
 conn.send(f"move:{x},{y}".encode())  
 except:  
 show\_error("Ошибка отправки хода")  
  
  
def send\_rematch\_request():  
 if conn:  
 try:  
 conn.send("rematch:request".encode())  
 except:  
 show\_error("Ошибка запроса реванша")  
  
  
def send\_rematch\_response(accepted):  
 if conn:  
 try:  
 conn.send(f"rematch:{'accept' if accepted else 'reject'}".encode())  
 except:  
 show\_error("Ошибка ответа на реванш")  
  
  
def send\_exit():  
 if conn:  
 try:  
 conn.send("exit".encode())  
 except:  
 pass  
  
  
def return\_to\_menu():  
 global state, connected, conn, waiting\_for\_rematch, rematch\_offered\_by\_opponent, game\_mode  
 if game\_mode == 'network':  
 send\_exit()  
 state = 'menu'  
 connected = False  
 if conn:  
 conn.close()  
 conn = None  
 waiting\_for\_rematch = False  
 rematch\_offered\_by\_opponent = False  
 game\_mode = None  
 reset()  
  
  
# Поток приема  
def receive\_moves():  
 global my\_turn, waiting\_for\_rematch, rematch\_offered\_by\_opponent, rematch\_accepted, running, state, connected, conn  
 while running:  
 if connected and conn:  
 try:  
 data = conn.recv(1024).decode()  
 if not data:  
 show\_error("Соединение разорвано")  
 return\_to\_menu()  
 continue  
  
 if data.startswith("move:"):  
 x, y = map(int, data[5:].split(','))  
 if not grid[y][x]:  
 grid[y][x] = 'O' if current\_player == 'X' else 'X'  
 my\_turn = True  
 check\_winner()  
 elif data == "rematch:request":  
 waiting\_for\_rematch = False  
 rematch\_offered\_by\_opponent = True  
 elif data == "rematch:accept":  
 rematch\_accepted = True  
 reset()  
 elif data == "rematch:reject":  
 waiting\_for\_rematch = False  
 rematch\_offered\_by\_opponent = False  
 show\_error("Соперник отказал в реванше")  
 elif data == "exit":  
 show\_error("Соперник вышел из игры")  
 return\_to\_menu()  
 except Exception as e:  
 return\_to\_menu()  
 time.sleep(0.1)  
  
  
# Основная функция  
def main():  
 global role, my\_turn, current\_player, running, waiting\_for\_rematch, rematch\_offered\_by\_opponent, rematch\_accepted, state, connected, conn, server\_error, game\_mode  
  
 state = 'menu'  
 create\_button, connect\_button, local\_button = draw\_menu()  
 pygame.display.flip()  
  
 # Запускаем поток для приема сообщений  
 recv\_thread = threading.Thread(target=receive\_moves, daemon=True)  
 recv\_thread.start()  
  
 while running:  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 running = False  
 if conn:  
 send\_exit()  
 conn.close()  
 break  
  
 if event.type == pygame.MOUSEBUTTONDOWN and event.button == 1:  
 if state == 'menu':  
 server\_error = None # Сбрасываем ошибку при новом выборе  
 if create\_button.collidepoint(event.pos):  
 role = 'server'  
 game\_mode = 'network'  
 draw\_waiting()  
 pygame.display.flip()  
 # Запускаем сервер в отдельном потоке  
 server\_thread = threading.Thread(target=handle\_network, daemon=True)  
 server\_thread.start()  
 # Ждем подключения или ошибки  
 start\_time = time.time()  
 while not connected and not server\_error and time.time() - start\_time < 10 and running:  
 time.sleep(0.1)  
 draw\_waiting()  
 pygame.display.flip()  
  
 if connected:  
 state = 'game'  
 reset()  
 time.sleep(0.5) # Даем время для синхронизации  
 elif server\_error:  
 state = 'menu'  
  
 elif connect\_button.collidepoint(event.pos):  
 role = 'client'  
 game\_mode = 'network'  
 draw\_waiting()  
 pygame.display.flip()  
 # Пытаемся подключиться  
 connect\_thread = threading.Thread(target=handle\_network, daemon=True)  
 connect\_thread.start()  
 # Ждем подключения или ошибки  
 start\_time = time.time()  
 while not connected and not server\_error and time.time() - start\_time < 5 and running:  
 time.sleep(0.1)  
 draw\_waiting()  
 pygame.display.flip()  
  
 if connected:  
 state = 'game'  
 reset()  
 time.sleep(0.5) # Даем время для синхронизации  
 elif server\_error:  
 state = 'menu'  
  
 elif local\_button.collidepoint(event.pos):  
 game\_mode = 'local'  
 state = 'game'  
 reset()  
  
 elif state == 'game' and not game\_over:  
 # Обработка хода только если:  
 # - В сетевом режиме и наш ход  
 # - В локальном режиме всегда  
 if game\_mode == 'local' or (game\_mode == 'network' and my\_turn):  
 x, y = event.pos[0] // CELL\_SIZE, event.pos[1] // CELL\_SIZE  
 if 0 <= x < 3 and 0 <= y < 3 and not grid[y][x]:  
 grid[y][x] = current\_player  
  
 # В локальном режиме сразу меняем игрока  
 if game\_mode == 'local':  
 # Меняем игрока после хода  
 current\_player = 'O' if current\_player == 'X' else 'X'  
  
 # В сетевом режиме отправляем ход  
 if game\_mode == 'network':  
 send\_move(x, y)  
  
 check\_winner()  
  
 # В сетевом режиме передаем ход  
 if game\_mode == 'network':  
 my\_turn = False  
  
 elif state == 'game' and game\_over:  
 rematch\_rect, exit\_rect = draw\_rematch\_ui()  
 if rematch\_rect.collidepoint(event.pos):  
 if game\_mode == 'network':  
 if not waiting\_for\_rematch and not rematch\_offered\_by\_opponent:  
 send\_rematch\_request()  
 waiting\_for\_rematch = True  
 elif rematch\_offered\_by\_opponent:  
 send\_rematch\_response(True)  
 rematch\_accepted = True  
 reset()  
 else: # Локальный режим  
 reset()  
 elif exit\_rect.collidepoint(event.pos):  
 if game\_mode == 'network' and rematch\_offered\_by\_opponent:  
 send\_rematch\_response(False)  
 return\_to\_menu()  
  
 # Отрисовка текущего состояния  
 screen.fill(WHITE)  
 if state == 'menu':  
 create\_button, connect\_button, local\_button = draw\_menu()  
 elif state == 'game':  
 draw\_board()  
 if not game\_over:  
 if game\_mode == 'local':  
 # Для локальной игры показываем, чей сейчас ход  
 status\_text = "Ход крестиков" if current\_player == 'X' else "Ход ноликов"  
 draw\_status(status\_text)  
 else: # Сетевой режим  
 draw\_status("Ваш ход" if my\_turn else "Ход соперника")  
 else:  
 # Формируем текст результата в зависимости от режима  
 if game\_mode == 'local':  
 if winner == 'X':  
 result\_text = "Победили крестики!"  
 elif winner == 'O':  
 result\_text = "Победили нолики!"  
 else:  
 result\_text = "Ничья!"  
 else:  
 if winner == current\_player:  
 result\_text = "Вы выиграли!"  
 elif winner == 'Ничья':  
 result\_text = "Ничья!"  
 else:  
 result\_text = "Вы проиграли!"  
  
 draw\_status(result\_text)  
 rematch\_rect, exit\_rect = draw\_rematch\_ui()  
  
 if game\_mode == 'network':  
 if waiting\_for\_rematch and not rematch\_offered\_by\_opponent:  
 txt = small\_font.render("Ожидание ответа...", True, BLACK)  
 screen.blit(txt, txt.get\_rect(center=(WIDTH // 2, HEIGHT - 240)))  
  
 pygame.display.flip()  
 clock.tick(30)  
  
 pygame.quit()  
 sys.exit()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()