Федеральное государственное бюджетное образовательное учреждение высшего образования «Нижегородский государственный архитектурно-строительный университет» (ННГАСУ)

*Факультет инженерно-экологических систем и сооружений*

*Кафедра информационных систем и технологий*

КУРСОВАЯ РАБОТА

по дисциплине: «Инфокоммуникационные системы и сети»

на тему: «Разработка онлайн-игры»

Выполнил студент 3 курса гр. ИС-30 Булатова Н.В.

Проверил Морозов Н.С.

Нижний Новгород

2022 г.

Содержание

[Введение 3](#_Toc125407917)

[Задачи 3](#_Toc125407918)

[Теоретическая часть 4](#_Toc125407919)

[Маршрутизация 4](#_Toc125407920)

[Виды маршрутизации 4](#_Toc125407921)

[Протокол IP 5](#_Toc125407922)

[IP-адресс 5](#_Toc125407923)

[Маска подсети 6](#_Toc125407924)

[Модель TCP/IP 7](#_Toc125407925)

[Топология сети 8](#_Toc125407926)

[Построение топологии 9](#_Toc125407927)

[Код программы 10](#_Toc125407928)

[Программа сервера 10](#_Toc125407929)

[Программа клиента 11](#_Toc125407930)

[Пример работы программы 15](#_Toc125407931)

[Работа сервера 15](#_Toc125407932)

[Работа игры 15](#_Toc125407933)

[Заключение 17](#_Toc125407934)

[Список литературы 18](#_Toc125407935)

# Введение

Целью данной работы является написание простой онлайн-игры на языке программирования Python с использованием знаний и умений, полученных в ходе изучения дисциплины.

Необходимо написать игру «Камень, Ножницы, Бумага», в которой оба игрока выбирают значения с помощью кнопок и используют для общения чат. Счетчик побед/ничей/проигрышей должен подсчитываться программой после каждого раунда.

# Задачи

* Изучить систему маршрутизации;
* Изучить систему стека TCP/IP;
* Произвести настройку протокола транспортного уровня;
* Произвести настройку сокетов серверной и клиентской части игры.

# Теоретическая часть

## Маршрутизация

**Маршрутизация** - процесс определения пути передачи данных между узлами разных сетей, являющимся лучшим по одному из критериев. В большинстве современных сетей маршрутизация пакетов выполняется на основе заголовка IP (IP-адрес получателя).

**Маршрутизатором** или шлюзом называется узел сети с несколькими интерфейсами, каждый из которых имеет свой MAC-адрес и IP адрес.

### Виды маршрутизации

1. **Прямая маршрутизация.** При прямой маршрутизации отправитель в определенной IP-сети может напрямую передавать кадры любому получателю в той же сети. При этом не требуется функциональность IP-маршрутизации.

Например, узлу 10.1.1.1 необходимо передать пакет узлу 10.2.2.2. Первое, что он делает — определяет, находится ли IP-адрес получателя в одной с ним сети.

Для этого сравнивает свой номер сети 10 с номером сети получателя 10. Делает вывод, что узел-получатель находится в одном с ним сегменте сети. С помощью протокола ARP определяет MAC-адрес узла-получателя и посылает пакет по этому адресу.

ARP (англ. Address Resolution Protocol — протокол определения адреса) — протокол в компьютерных сетях, предназначенный для определения MAC-адреса, имея IP-адрес другого компьютера.

2. **Косвенная маршрутизация**. Косвенная маршрутизация происходит в том случае, если отправитель и получатель находятся в разных IP-сетях. Косвенная маршрутизация требует, чтобы отправитель передавал пакеты маршрутизатору для доставки их через распределённую сеть.

Например, узел 10.1.1.1 имеет пакет, который нужно отправить узлу 172.16.0.1.

* Узел назначения находится не на одной с передающим узлом сети. Узел 10.1.1.1 сконфигурирован так, что любые пакеты, требующие косвенной маршрутизации, передаются его шлюзу по умолчанию — маршрутизатору 1.
* Чтобы доставить пакет маршрутизатору 1, узлу 10.1.1.1 необходим MAC-адрес маршрутизатора 10.3.3.3. Если МАС-адрес узлу 10.1.1.1 неизвестен, он отправляет ARP-запрос, чтобы его получить. Затем пакет, предназначенный для 172.16.0.1 отправляется маршрутизатору 1.
* Маршрутизатор 1 осознает, что он подсоединен к сети 172.16. и полагает, что узел 172.16.0.1 должен быть частью этой сети. Маршрутизатор 1 реализует свою собственную процедуру прямой маршрутизации и посылает ARP-запрос, ища узел назначения.

## Протокол IP

### IP-адрес

IP адреса нужны для уникальной идентификации компьютеров в крупной составной сети, которая может включать в себя весь мир, например, сети Интернет, и различные части сети интернет, построенные на разных технологиях канального уровня.

Сейчас есть 2-е версии протокола IP: версия IPv4 и IPv6. Основное отличие между версиями протоколов в длине IP адреса. В IPv4 длина адреса 4 байта, а в IPv6 длина адреса 16 байт.

Длина адреса IPv4 — 32 бита, 4 байта. И чтобы людям было удобно работать с такими IP адресами их делят на 4 части.

В каждой части по 8 бит, такая часть называется октет. Каждый октет записывают в десятичном формате, и форма записи IP адреса следующая: четыре октета разделенных точкой (213.180.193.3). С таким видом деления адресов людям гораздо удобнее работать, чем с записью в двоичной форме длиной в 32 бита.

Одна из задач сетевого уровня обеспечить масштабирование, построить такую сеть, которая может работать в масштабах всего мира. Для этого сетевой уровень работает не с отдельными компьютерами, а с подсетями, которые объединяют множество компьютеров.

В IP объединение происходит следующим образом: подсеть — это некое количество компьютеров, у которых одинаковая старшая часть IP-адреса. В примере ниже у данного диапазона адресов одинаковые первые 3 октета, и отличается только последний октет.

И маршрутизаторы - устройства, передающие информацию на сетевом уровне, работают уже не с отдельными IP адресами, а с подсетями.

### Маска подсети

Протокол IP предусматривает группировку адресов в сети с помощью масок сети. Маска сети применяется по отношению к IP-адресу, разделяя его на две части: идентификатор сети и идентификатор узла. Устройства, подключенные к одной сети, будут иметь одинаковый идентификатор сети и различные идентификаторы узлов. Для того, чтобы идентификатор сети на всех устройствах совпадал, необходимо использовать одинаковые значения маски сети при настройке устройств. Множество идентификаторов узлов позволяют сделать вывод о количестве устройств, которые могут быть подключены к этой сети, и их IP-адресах.

Маска сети имеет размер 32 бита и записывается в таком же формате, как IP-адрес, однако имеет одно отличие: маска состоит из последовательности единичных битов, за которыми следуют нулевые биты, т.е. набор масок предустановлен и состоит из 33 значений: от 0 до 32. Конечный диапазон возможных значений позволяет записывать маску сети в сокращённом формате, в котором через косую черту указывается количество единичных бит в маске.

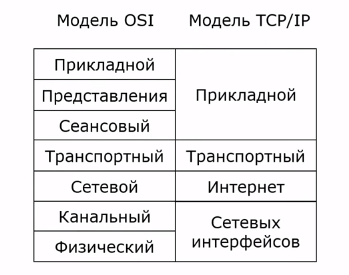
Единичные биты в маске сети определяют идентификатор сети: разряды IP-адреса, соответствующие единичным битам маски, должны быть зафиксированы и не могут быть изменены. Остальные разряды IP-адреса, соответствующие нулевым битам маски, могут принимать произвольные значения и определяют идентификатор узла.

При конфигурации устройств, подключенных к сети, IP-адреса не используются в отрыве от маски сети, т.к. правила маршрутизации подразумевают разный подход при передаче данных устройству из "своей" сети и остальным устройствам. Следует иметь в виду, что маска сети указывается в конфигурации устройств и не передаётся в служебном заголовке IP-пакета.

## Модель TCP/IP

Изначально данный стек создавался для объединения больших компьютеров в университетах по телефонным линиям связи соединения «точка-точка». Но когда появились новые технологии, широковещательные (Ethernet) и спутниковые, возникла необходимость адаптировать TCP/IP, что оказалось непростой задачей. Именно поэтому наряду с OSI появилась модель TCP/IP.

Через модель описывается, как необходимо строить сети на базе различных технологий, чтобы в них работал стек протоколов TCP/IP.



В таблице представлено сравнение моделей OSI и TCP/IP. Последняя включает в себя 4 уровня:

1. Самый нижний, уровень сетевых интерфейсов, обеспечивает взаимодействие с сетевыми технологиями (Ethernet, Wi-Fi и т. д.). Это объединение функций канального и физического уровней OSI.
2. Уровень интернет стоит выше, и по задачам перекликается с сетевым уровнем модели OSI. Он обеспечивает поиск оптимального маршрута, включая выявление неполадок в сети. Именно на этом уровне работает маршрутизатор.
3. Транспортный отвечает за связь между процессами на разных компьютерах, а также за доставку переданной информации без дублирования, потерь и ошибок, в необходимой последовательности.
4. Прикладной объединил в себе 3 уровня модели OSI: сеансовый, представления и прикладной. То есть он выполняет такие функции, как поддержка сеанса связи, преобразование протоколов и информации, а также взаимодействие пользователя и сети.

## Топология сети

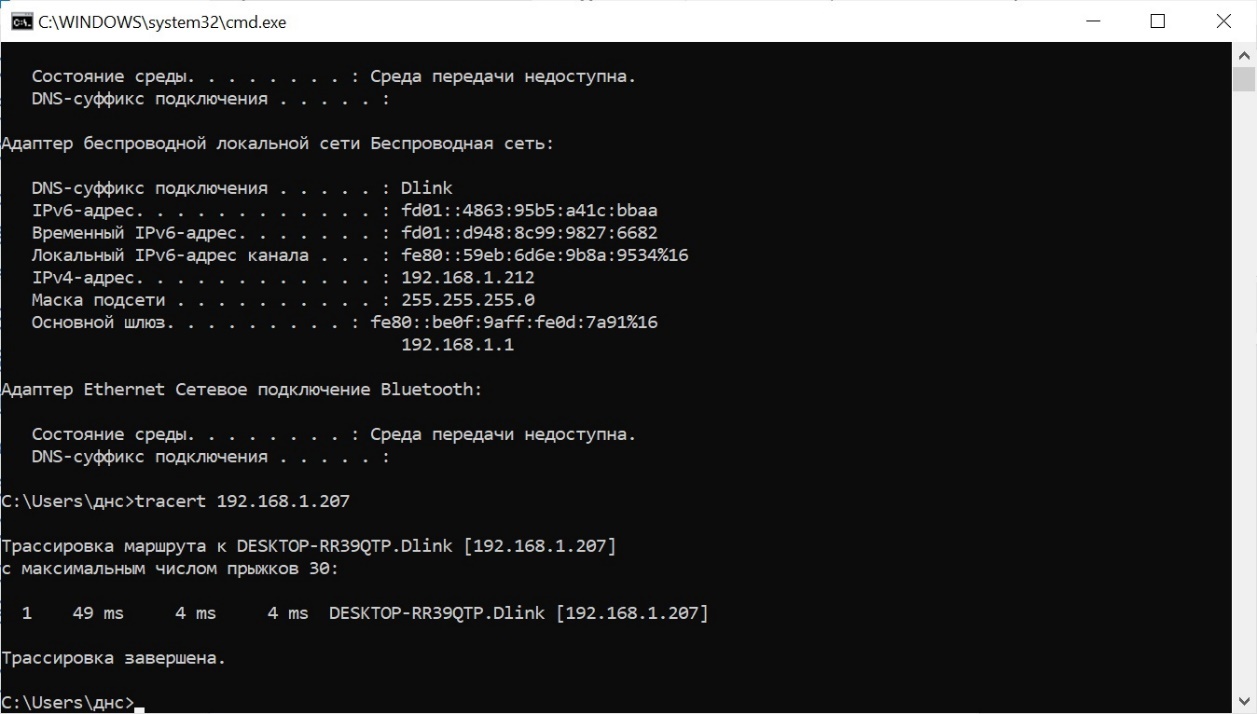
Программирование и построение компьютерных сетей выросли из математики и поэтому унаследовали математические расчеты и схематику построения устройств и связей. А самим термином топология сети охарактеризовали расположение и схему связей между устройствами. Устройствами выступают компьютеры, концентраторы, роутеры, серверы, принтеры и прочая вспомогательная электроника. Кроме расположения устройств, топология обуславливает компоновку кабелей, варианты размещения коммутирующего оборудования, систему обмена сигналами и прочие запросы потребителей компьютерных технологий.

Соединение в сети вызвано необходимостью объединения ресурсов компьютеров, экономией на периферийных устройствах, и как следствие решением комплексных задач. Исходя из конкретных предполагаемых задач и выстраивается топология компьютерной сети. Существуют семь основных видов соединений.

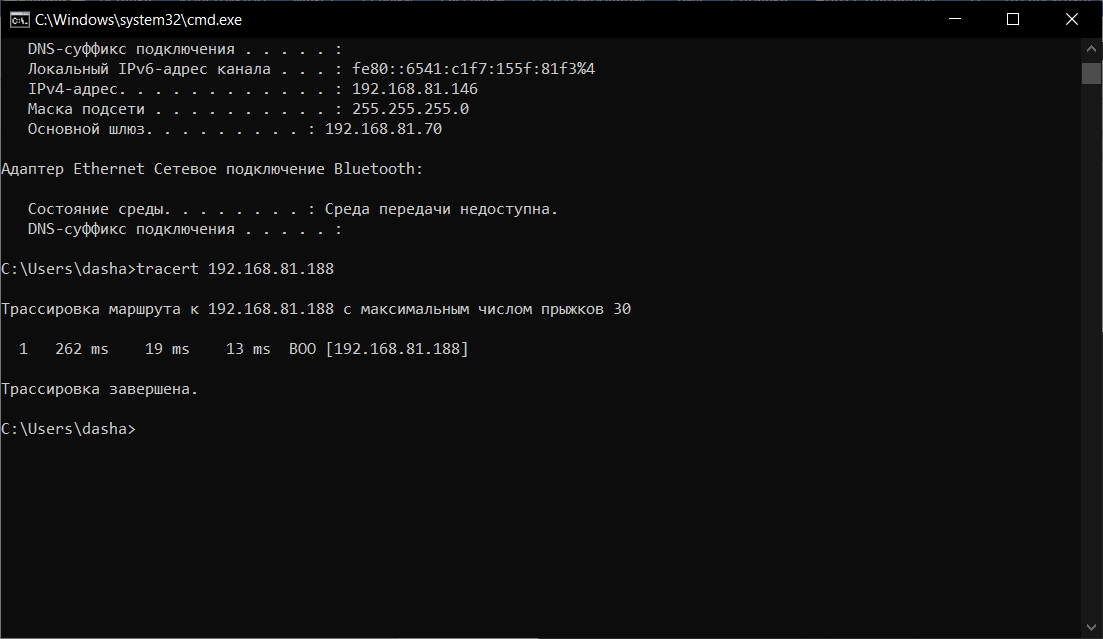
1. Топология шина
2. Топология кольцо
3. Топология звезда
4. Полносвязная или сеточная топология
5. Ячеистая топология
6. Топология дерево
7. Смешанная топология

# Построение топологии

**Wi-Fi**



**Mobil**



# Код программы

## Программа сервера

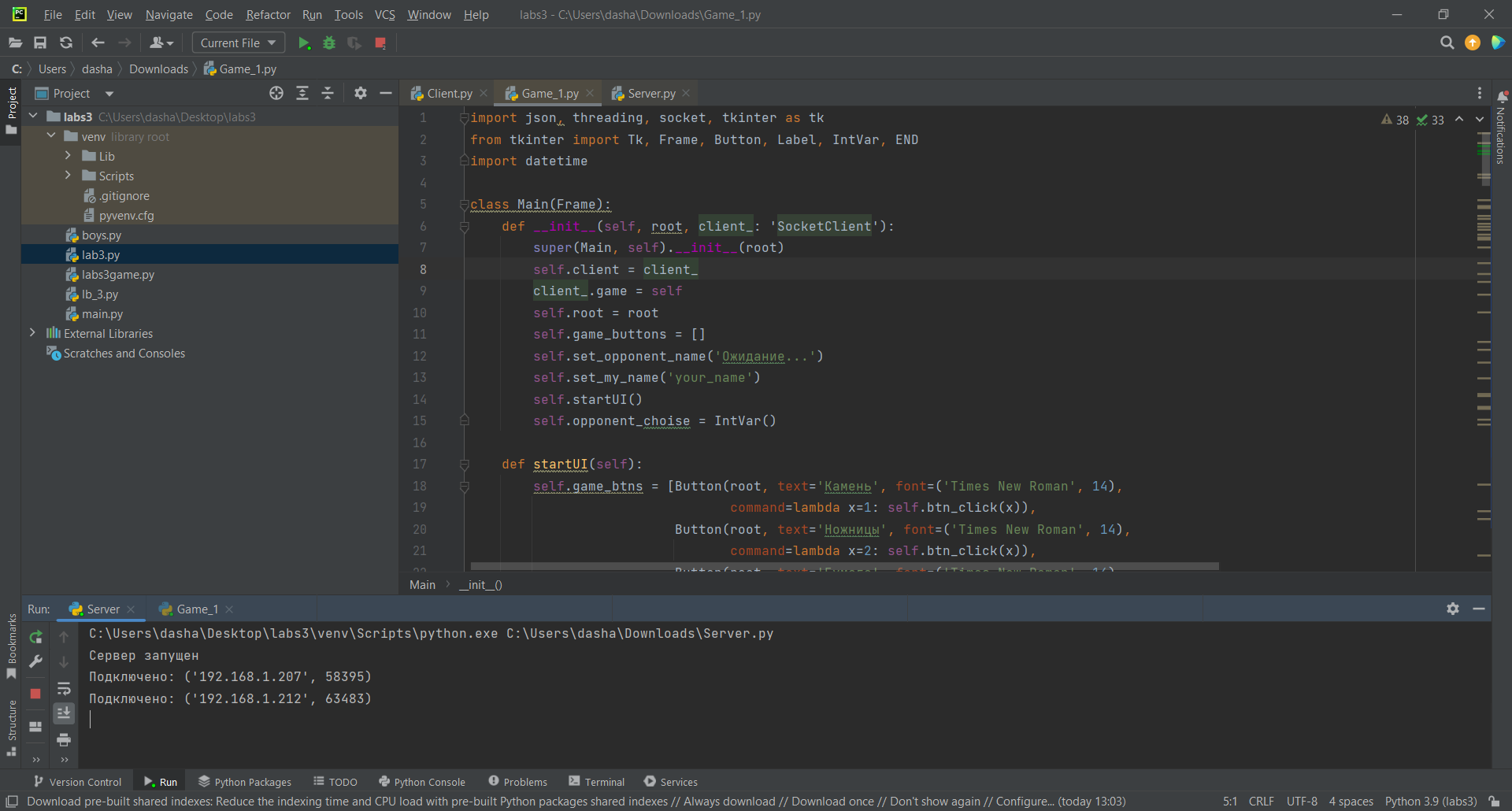
import json, socket, threading  
from typing import Optional, Union  
  
class Server:  
 def \_\_init\_\_(self, host, port):  
 self.sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 self.sock.bind((host, port))  
 self.clients: list[socket.socket] = []  
 self.actions: dict[socket.socket, dict[str, Union[int, str]]] = {}  
  
 def listen(self):  
 self.sock.listen(2)  
 while True:  
 client, address = self.sock.accept()  
 print('Подключено:', address)   
 threading.Thread(target=self.client\_handler, args=(client,)).start()  
 self.clients.append(client)  
  
 def distribute(self, data: str, author: Optional[socket.socket] = None):  
 for client in self.clients:  
 if client == author:  
 continue  
 client.send(data.encode())  
  
 def client\_handler(self, client: socket.socket):  
 while True:  
 try:  
 data = json.loads(client.recv(1024).decode())  
 if not data:  
 continue  
   
 command = data['command']  
 nickname = data['nickname']  
 message = data['message']  
 if command == 'chat':  
 self.distribute(json.dumps(data), client)  
 continue  
 if command == 'action':  
 self.actions[client] = {  
 'action': int(message),  
 'nickname': nickname,  
 }  
 if len(self.actions) == 2:  
 self.send\_result()  
 self.actions = {}  
 except Exception as e:  
 print(e)  
 self.clients.remove(client)  
 client.close()  
 return False  
   
 def send\_result(self):  
 user = list(self.actions.keys())[0]  
 opponent = list(self.actions.keys())[1]  
 user\_choice = self.actions[user]['action']  
 opponent\_choice = self.actions[opponent]['action']  
 user\_result = 'lose'  
 opponent\_result = 'win'  
 if user\_choice == opponent\_choice:  
 user\_result = 'draw'  
 opponent\_result = 'draw'  
 elif user\_choice == 2 and opponent\_choice == 3:  
 user\_result = 'win'  
 opponent\_result = 'lose'  
 elif opponent\_choice == 2 and user\_choice == 3:  
 user\_result = 'lose'  
 opponent\_result = 'win'  
 elif (user\_choice + 1) % 3 == opponent\_choice:  
 user\_result = 'win'  
 opponent\_result = 'lose'  
 user.send(  
 json.dumps(  
 {  
 'command': 'result',  
 'message': user\_result,  
 'nickname': self.actions[opponent]['nickname'],  
 }  
 ).encode()  
 )  
 opponent.send(  
 json.dumps(  
 {  
 'command': 'result',  
 'message': opponent\_result,  
 'nickname': self.actions[user]['nickname'],  
 }  
 ).encode()  
 )  
if \_\_name\_\_ == '\_\_main\_\_':  
 print('Сервер запущен')  
 Server('192.168.1.207', 8081).listen()

## Программа клиента

import json, threading, socket, tkinter as tk  
from tkinter import Tk, Frame, Button, Label, IntVar, END  
import datetime  
  
class Main(Frame):  
 def \_\_init\_\_(self, root, client\_: 'SocketClient'):  
 super(Main, self).\_\_init\_\_(root)  
 self.client = client\_  
 client\_.game = self  
 self.root = root  
 self.game\_buttons = []  
 self.set\_opponent\_name('Ожидание...')  
 self.set\_my\_name('your\_name')  
 self.startUI()  
 self.opponent\_choise = IntVar()  
  
 def startUI(self):  
 self.game\_btns = [Button(root, text='Камень', font=('Times New Roman', 14),  
 command=lambda x=1: self.btn\_click(x)),  
 Button(root, text='Ножницы', font=('Times New Roman', 14),  
 command=lambda x=2: self.btn\_click(x)),  
 Button(root, text='Бумага', font=('Times New Roman', 14),  
 command=lambda x=3: self.btn\_click(x))]  
 self.game\_btns[0].place(x=10, y=120, width=120, height=50)  
 self.game\_btns[1].place(x=155, y=120, width=120, height=50)  
 self.game\_btns[2].place(x=300, y=120, width=120, height=50)  
 self.win = self.draw = self.lose = 0  
  
 self.line = Label(root, text='\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_', bg='#FFF',  
 font=('Times New Roman', 14))  
 self.line.place(x=7, y=165)  
  
 self.lbl = Label(root, text='Начало игры!', bg='#FFF',  
 font=('Times New Roman', 14, 'bold'))  
 self.lbl.place(x=150, y=65)  
 self.lbl2 = Label(root, justify='left', font=('Times New Roman', 14),  
 text=f'Побед: {self.win}\nПроигрышей: {self.lose}\nНичей: {self.draw}',  
 bg='#FFF')  
 self.lbl2.place(x=290, y=5)  
 self.lbl3 = Label(root, justify='right', font=('Times New Roman', 14),  
 text=f'Оппонент: {self.opponent\_name}',  
 bg='#FFF')  
 self.lbl3.place(x=10, y=5)  
  
 self.txt = tk.Text(root, font=(f'Times New Roman', 14), width=45, height=8, bg='#f0f8ff')  
 self.txt.configure(state='disabled')  
 self.txt.place(x=10, y=245)  
 scrollbar = tk.Scrollbar(self.txt)  
 scrollbar.place(relheight=1, relx=0.96)  
  
 self.entry = tk.Entry(root, font=('Times New Roman', 14), width=35, bg='#f0f8ff')  
 self.entry.place(x=10, y=212)  
  
 send = Button(root, text='Отправить', font=('Times New Roman', 14),   
 command=self.send\_button, width=8, height=1)  
 send.place(x=330, y=200)  
  
 def send\_button(self, \*args):  
 input\_text = self.entry.get()  
 if not input\_text:  
 return  
 self.txt.configure(state='normal')  
 self.txt.insert(END, f'{datetime.datetime.now().hour}:{datetime.datetime.now().minute}:{datetime.datetime.now().second} {nick} -> {input\_text}\n')  
 self.txt.see('end')  
 self.txt.configure(state='disabled')  
 self.entry.delete(0, END)  
 self.client.send('chat', input\_text)  
  
 def btn\_click(self, choise):  
 self.choise = choise  
 for btn in self.game\_btns:  
 btn['state'] = tk.DISABLED  
 self.lbl3.configure(text=f'Оппонент: {self.opponent\_name}')  
 self.client.send('action', str(choise))  
 root.wait\_variable(self.opponent\_choise)  
 self.calc\_result(choise, self.get\_opponent\_choise())  
 for btn in self.game\_btns:  
 btn['state'] = tk.NORMAL  
  
 def set\_my\_name(self, name):  
 self.my\_name = name  
  
 def set\_opponent\_name(self, name):  
 self.opponent\_name = name  
  
 def get\_opponent\_choise(self):  
 return self.opponent\_choise.get()  
  
class SocketClient:  
   
 def \_\_init\_\_(self, name: str):  
 self.client = None  
 self.name = name  
 self.game = None  
 self.series=0  
  
 def result\_handler(self, message: str):  
   
 if message == 'draw':  
 self.game.draw += 1  
 self.series=0  
 self.game.lbl['fg']='#FF8000'  
 self.game.lbl.configure(text='Ничья')  
 if message == 'win':  
 self.game.win += 1  
 self.series+=1  
 self.game.lbl['fg']='#008000'  
 self.game.lbl.configure(text=f'Победа\nСерия побед: {self.series}')  
 if message == 'lose':  
 self.game.lose += 1  
 self.series=0  
 self.game.lbl.configure(text=f'Проигрыш')  
 self.game.lbl['fg']='#FF0000'  
 self.game.lbl2.configure(text=f'Побед: {self.game.win}\nПроигрышей: {self.game.lose}\nНичей: {self.game.draw}')  
  
 for btn in self.game.game\_btns:  
 btn['state'] = tk.NORMAL  
  
 def socket\_start(self, host: str, port: int):  
 self.client = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 self.client.connect((host, port))  
 while True:  
 data = self.client.recv(1024)  
 if not data:  
 continue  
 data = json.loads(data.decode())  
 command = data['command']  
 nickname = data['nickname']  
 message = data['message']  
  
 self.game.lbl3.configure(text=f'Оппонент: {nickname}')  
 if command == 'result':  
 self.result\_handler(message)  
 if command == 'chat':  
 self.game.txt.configure(state='normal')  
 self.game.txt.insert(END, f'{datetime.datetime.now().hour}:{datetime.datetime.now().minute}:{datetime.datetime.now().second} {nickname} -> {message}\n')  
 self.game.txt.see('end')  
 self.game.txt.configure(state='disabled')  
  
 def send(self, command: str, message: str):  
 data = json.dumps(  
 {'command': command, 'nickname': self.name, 'message': message}  
 )  
 self.client.sendall(data.encode())  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 print(f'Игра запущена!')  
 print('Введите ваше имя:')  
 nick=input()  
 root = Tk()  
 root.geometry('430x430')  
 root.title('Камень, ножницы, бумага')  
 root.resizable(False, False)  
 root['bg'] = '#FFF'  
 client = SocketClient(name=nick)  
 app = Main(root, client)  
 app.pack()  
 game\_thread = threading.Thread(target=root.mainloop)  
 socket\_thread = threading.Thread(  
 target=client.socket\_start, args=('192.168.1.207', 8081)  
 )  
 socket\_thread.start()  
 game\_thread.run()

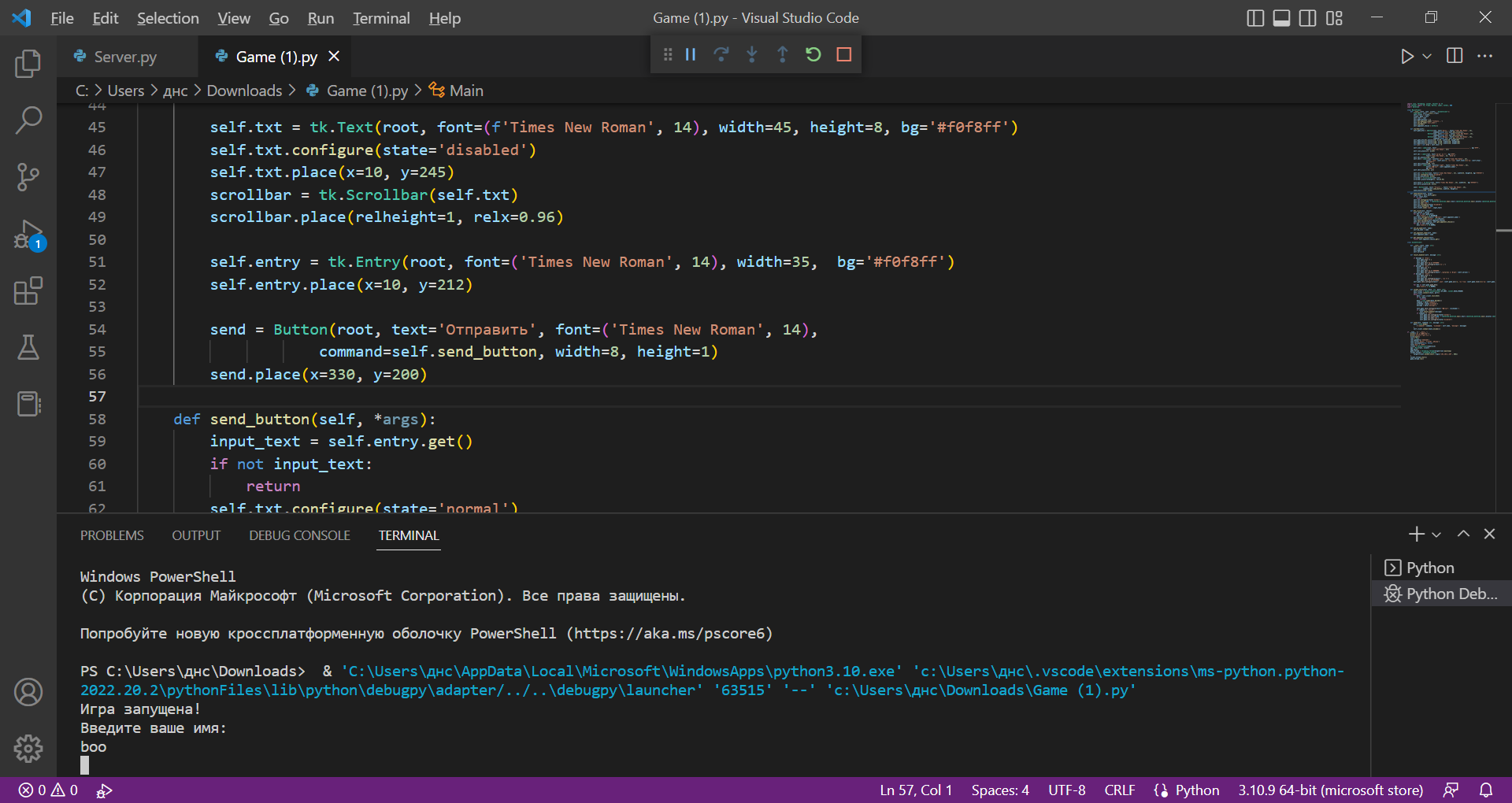
# Пример работы программы

## Работа сервера

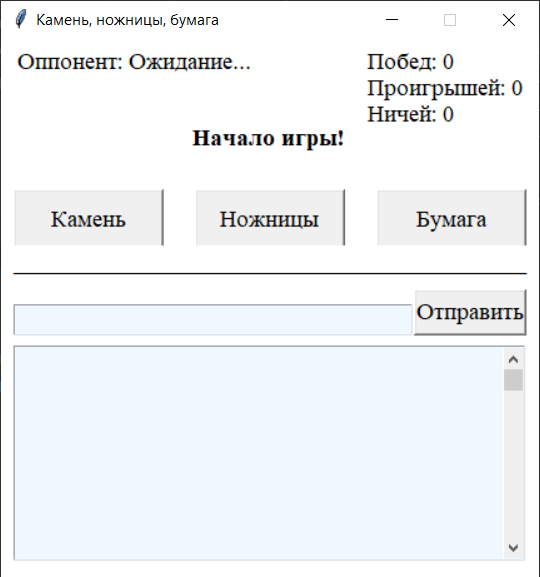


## Работа игры

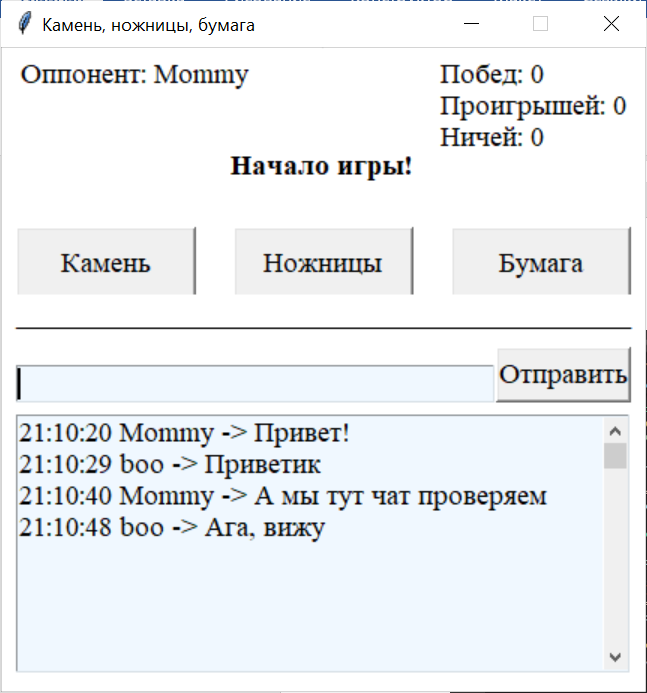
После запуска игры необходимо ввести свое имя, которое будет использовано в игре в качестве никнейма:



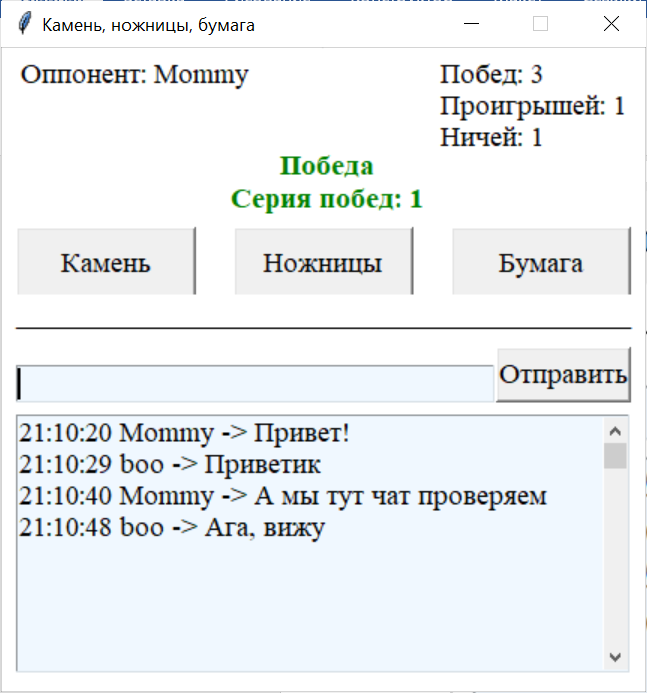
При запуске игры и ожидании оппонента игрок наблюдает:



Для коммуникации игроков сделан чат:



Пример работы игры:



# Заключение

В ходе выполнения курсовой работы была рассмотрена реализация передачи данных через протокол TCP. Для этого была создана онлайн игра «Камень, Ножницы, Бумага», которая связывает вычислительные машины по локальной сети и реализует клиент-серверное подключение через сокеты. Так же был создан сервер, обрабатывающий результаты и подключающий игроков.

Передача через сокеты надежна и удобна в использовании, так как сокет гарантирует исправление ошибок, обработку доставки и сохраняет последовательность отправляемых данных. Таким образом, использование сокетов автоматизирует передачу данных по сети.

# Список литературы

1. Библиотека программиста, «Компьютерные сети от А до Я: стек протоколов TCP/IP» [Электронный ресурс]. URL: <https://proglib.io/p/tcp-ip> (Дата обращения: 12.12.2022).
2. Екатерина @ketrin7, «Пишем графическую программу на Python с tkinter» [Электронный ресурс]. URL: <https://habr.com/ru/post/268531/> (Дата обращения: 12.12.2022).
3. Созыкин А. В, «Разбираемся с основными топологиями компьютерных (локальных) сетей» [Электронный ресурс]. URL: <https://zvondozvon.ru/tehnologii/topologiya-setei> (Дата обращения: 12.12.2022).
4. Халелова Е. Н, «Лекция №3. Стек протоколов TCP/IP» [Электронный ресурс]. URL:

<https://en-15.ucoz.ru/POKS_lekcii/lekcia1/stek_protokolov_ip1.pdf> (Дата обращения: 12.12.2022).

1. Черменых В. Н «Библиотека Tkinter: Графические примитивы» [Электронный ресурс]. URL: <https://it-black.ru/biblioteka-tkinter-graficheskie-primitivy/> (Дата обращения: 12.12.2022).
2. Черменых В. Н, «Понятия маршрутизации» [Электронный ресурс]. URL: <https://it-black.ru/ponyatiya-marshrutizatsii/> (Дата обращения: 12.12.2022).
3. Infinet wireless, «Понятие маршрутизации» [Электронный ресурс]. URL: <https://wiki.infinetwireless.com/pages/viewpage.action?pageId=76808266> (Дата обращения: 12.12.2022).