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 Бобина Д.А.

Проверил Морозов Н.С.

Нижний Новгород – 2022 г.

Содержание

[Введение 3](#_Toc125378133)

[Задачи 3](#_Toc125378134)

[Теоретическая часть 4](#_Toc125378135)

[Маршрутизация 4](#_Toc125378136)

[Стек TCP/IP 5](#_Toc125378137)

[Транспортный уровень (Transport Layer) 5](#_Toc125378138)

[Порты и сокеты 6](#_Toc125378139)

[Преобразование IP-адресов в символьные адреса 6](#_Toc125378140)

[IP 7](#_Toc125378141)

[Маска подсети 8](#_Toc125378142)

[Код программы 9](#_Toc125378143)

[Пример работы программы 9](#_Toc125378144)

[Заключение 10](#_Toc125378145)

[Список литературы 11](#_Toc125378146)

# Введение

**Цель работы**: написать простую онлайн-игру на Python для закрепления и развития знаний, полученных в ходе изучение данной дисциплины.

В итоге необходимо получить онлайн игру «Камень, ножницы, бумага», в которой оппоненты отправляют друг-другу выбранные значения и могут общаться в чате, программа получает и считает результат.

# 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

**Маршрутизатором**, или **шлюзом**, называется узел сети с несколькими IP-интерфейсами (содержащими свой MAC-адрес и IP-адрес), подключенными к разным IP-сетям, осуществляющий на основе решения задачи маршрутизации перенаправление дейтаграмм из одной сети в другую для доставки от отправителя к получателю.

**Маршрутизаторы** представляют собой либо специализированные вычислительные машины, либо компьютеры с несколькими IP-интерфейсами, работа которых управляется специальным программным обеспечением.

Маршрутизация состоит из двух этапов:

1. На первом этапе происходит изучение сети, какие подсети есть в этой составной сети, какие маршрутизаторы и как эти маршрутизаторы объединены между собой.
2. Второй этап маршрутизации выполняется когда сеть уже изучена и на маршрутизатор поступил пакет, для этого пакета нужно определить куда именно его отправить. Иногда для второго этапа маршрутизации используется отдельный термин “продвижение” (forwarding).

В современных аппаратных маршрутизаторах для построения таблиц маршрутизации используется специализированное ПО («прошивка»), для обработки же IP-пакетов используется коммутационная матрица (или другая технология аппаратной коммутации), расширенная фильтрами адресов в заголовке IP-пакета.

Передача информации осуществляется с помощью IP-пакетов, заголовок каждого IP-пакета содержит IP-адреса получателя и отправителя пакета. Каждый пакет обрабатывается маршрутизатором в соответствии с его таблицей маршрутизации. Таблица, в свою очередь, содержит информацию, компьютеру с каким адресом направлять пакеты с тем или иным диапазоном адресов. Например, все пакеты определенного диапазона могут направляться другому маршрутизатору, который «отвечает» за этот сегмент.

В ряде случаев маршрутизатор может преобразовывать заголовок пакета, заменяя адреса отправителя и/или получателя пакета. В частности, это происходит при взаимодействии локальной сети (имеющей свои адреса) с глобальной сетью Интернет. В этом случае локальная сеть может быть видна извне по одному глобальному IP-адресу.

Для того чтобы маршрутизатор мог направлять пакеты с одним глобальным адресом тем или иным получателям в локальной сети, используется таблица NAT, где помимо IP-адресов указываются порты, идентифицирующие приложения, устанавливающие соединение. При этом номера портов указаны не в заголовке IP-пакета, а в заголовке сегмента TCP либо UDP (сегменты инкапсулируются в поле данных IP-пакетов). Это позволяет осуществлять взаимно-однозначную идентификацию получателя и отправителя в тех случаях, когда за одним глобальным адресом находится множество компьютеров локальных сетей.

## Стек TCP/IP

Протокол TCP/IP – это целая сетевая модель, описывающая способ передачи данных в цифровом виде. На правилах, включенных в нее, базируется работа интернета и локальных сетей независимо от их назначения и структуры.

Протокол TCP/IP основан на OSI и так же, как предшественник, имеет несколько уровней, которые и составляют его архитектуру. Всего выделяют 4 уровня – канальный (интерфейсный), межсетевой, транспортный и прикладной.

## Порты и сокеты

Процессы, работающие на прикладном уровне, «общаются» с транспортным, но они видны ему как «черные ящики» с зашифрованной информацией. Зато он понимает, на какой IP-адрес адресованы данные и через какой порт надо их принимать. Этого достаточно для точного распределения пакетов по сети независимо от месторасположения хостов. Порты с 0 до 1023 зарезервированы операционными системами, остальные, в диапазоне от 1024 до 49151, условно свободны и могут использоваться сторонними приложениями.

Комбинация IP-адреса и порта называется сокетом и используется при идентификации компьютера. Если первый критерий уникален для каждого хоста, второй обычно фиксирован для определенного типа приложений. Так, получение электронной почты проходит через 110 порт, передача данных по [протоколу FTP](https://timeweb.com/ru/community/articles/chto-takoe-ftp-i-kak-rabotat-v-filezilla-1) – по 21, открытие сайтов – по 80.

## Преобразование IP-адресов в символьные адреса

Технология активно используется для назначения буквенно-цифровых названий веб-ресурсов. При вводе домена в адресной строке браузера сначала происходит обращение к специальному серверу [DNS](https://timeweb.com/ru/community/articles/vvedenie-v-terminologiyu-elementy-i-ponyatiya-dns-1). Он всегда прослушивает порт 53 у всех компьютеров, которые подключены к интернету, и по запросу преобразует введенное название в стандартный IP-адрес.

После определения точного местонахождения файлов сайта включается обычная схема работы – от прикладного уровня с кодированием данных до обращения к физическому оборудованию на уровне сетевых интерфейсов. Процесс называется инкапсуляцией информации. На принимающей стороне происходит обратная процедура – декапсуляция.

## IP

IP-адрес — уникальный сетевой адрес узла в компьютерной сети, построенной на основе стека протоколов TCP/IP. IP-адрес представляет собой серию из 32 двоичных бит (единиц и нулей).

Так как человек невосприимчив к большому однородному ряду чисел, такому как этот 11100010101000100010101110011110 (здесь, к слову, 32 бита информации, так как 32 числа в двоичной системе), было решено разделить ряд на четыре 8-битных байта и получилась следующая последовательность: 11100010.10100010.00101011.10011110.

Это не сильно облегчило жизнь и было решение перевести данную последовательность в, привычную нам, последовательность из четырёх чисел в десятичной системе, то есть 226.162.43.158. 4 разряда также называются октетами. Данный IP адрес определяется протоколом IPv4. По такой схеме адресации можно создать более 4 миллиардов IP-адресов.

Максимальным возможным числом в любом октете будет 255 (так как в двоичной системе это 8 единиц), а минимальным – 0.

IP-адреса делятся на 5 классов (A, B, C, D, E).

* **Класс A**. Старший бит в адресах такого формата всегда равен 0. За идентификацию сети отвечает начальный октет, позволяющий разместить 127 уникальных сетей. Оставшиеся 3 октета используются для обозначения узлов, максимальное количество которых составляет 17 млн. на каждую сеть.
* **Класс B**. Первые биты IP-адреса равны 10. Начальные два октета относятся к идентификатору сети, а последние два – к идентификатору узла. Возможно создание 16384 сетей, каждая из которых поддерживает размещение 65000 узлов.
* **Класс C**. Начальные биты IP-адреса равны 110. За идентификацию сети отвечают первые три октета, позволяющие создать 2 млн. сетей. Последний октет отводится для идентификации узлов, максимальное число которых составляет 254 на каждую сеть.
* **Класс D**. Запись IP-адреса начинается с битов 1110. В сетях подобного формата используется широковещательная рассылка сообщений нескольким узлам.
* **Класс E**. IP-адреса зарезервированы для использования в будущем. Первые биты всегда равны 11110.

Так же IP-адреса делятся на «цвета» - белые и серые (или публичные и частные). Публичным IP адресом называется IP адрес, который используется для выхода в Интернет. Адреса, используемые в локальных сетях, относят к частным. Частные IP не маршрутизируются в Интернете.

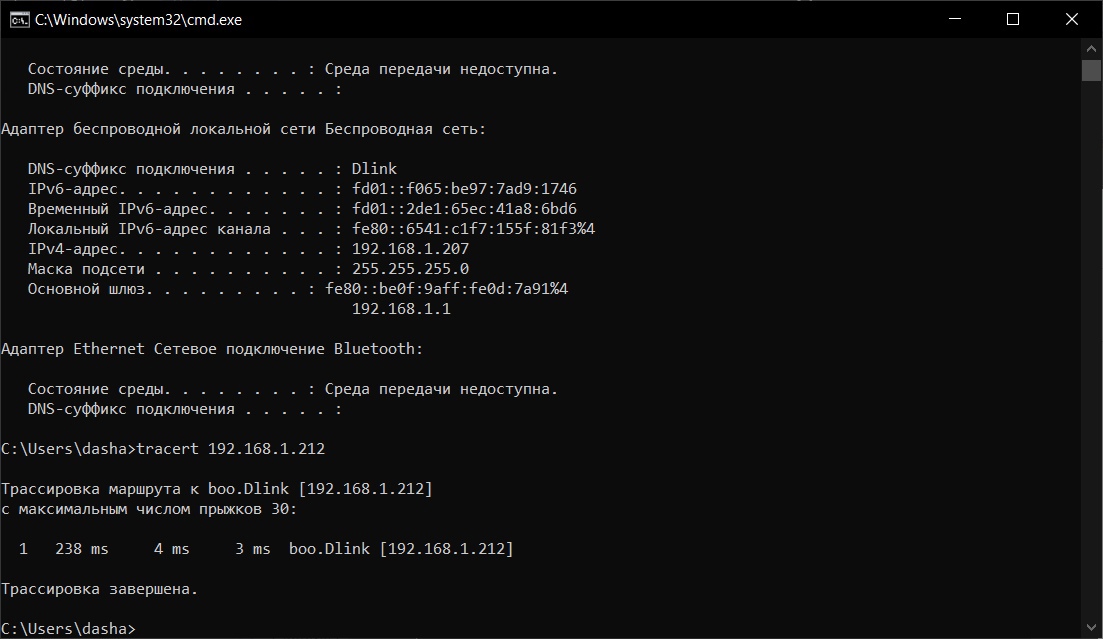
## Маска подсети

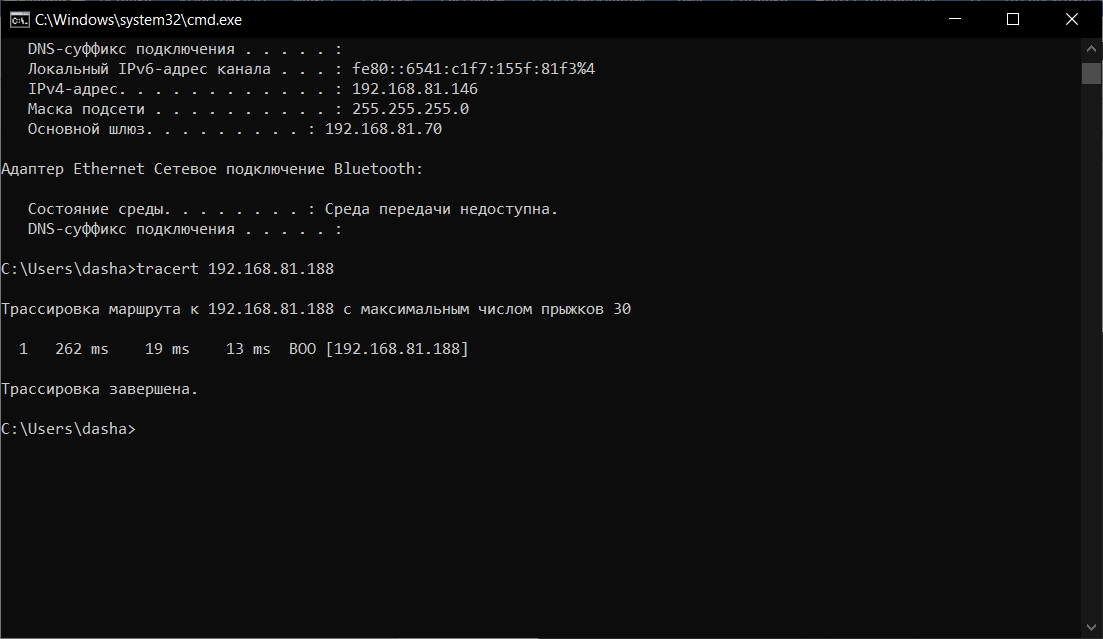
Устройства различают части IP-адреса при помощи **маски подсети** – 32-битной строки, разделённой на 4 октета, как и IP-адрес. При установке соединения каждый октет IP-адреса сопоставляется с октетом маски подсети.

По умолчанию в стандартной домашней сети маска подсети имеет вид: 255.255.255.0.

В примере маска IP-адреса указана в десятичном представлении и содержит числа «255» и «0». Первое отвечает за идентификацию сети, а второе за обозначение конечного узла.

# Построение топологии





# Код

## Сервер

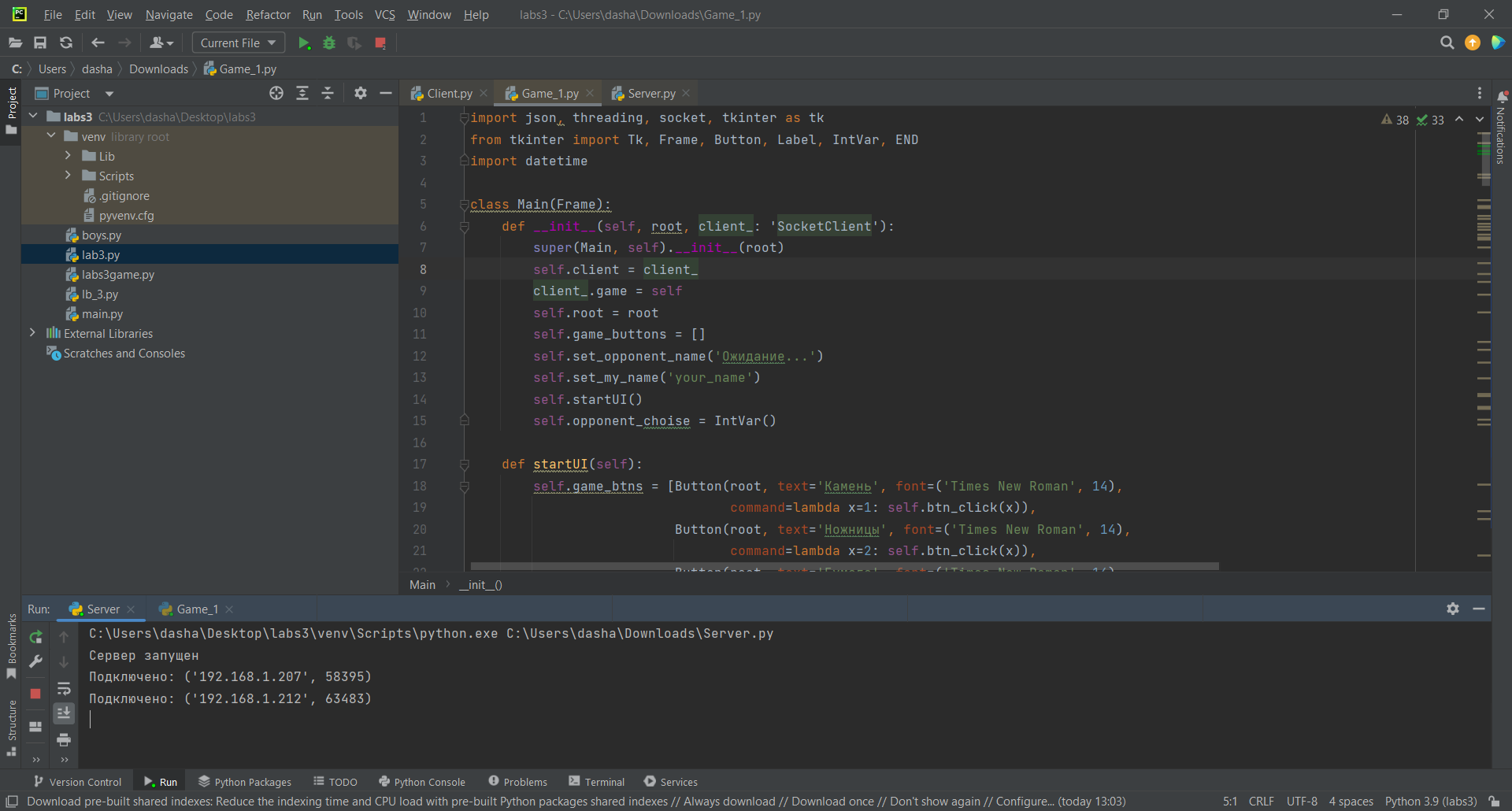
import json, socket, threading  
from typing import Optional, Union  
  
class Server:  
 def \_\_init\_\_(self, host, port):  
 self.sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 self.sock.bind((host, port))  
 self.clients: list[socket.socket] = []  
 self.actions: dict[socket.socket, 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()

## Клиент

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

## Сервер



## Клиент

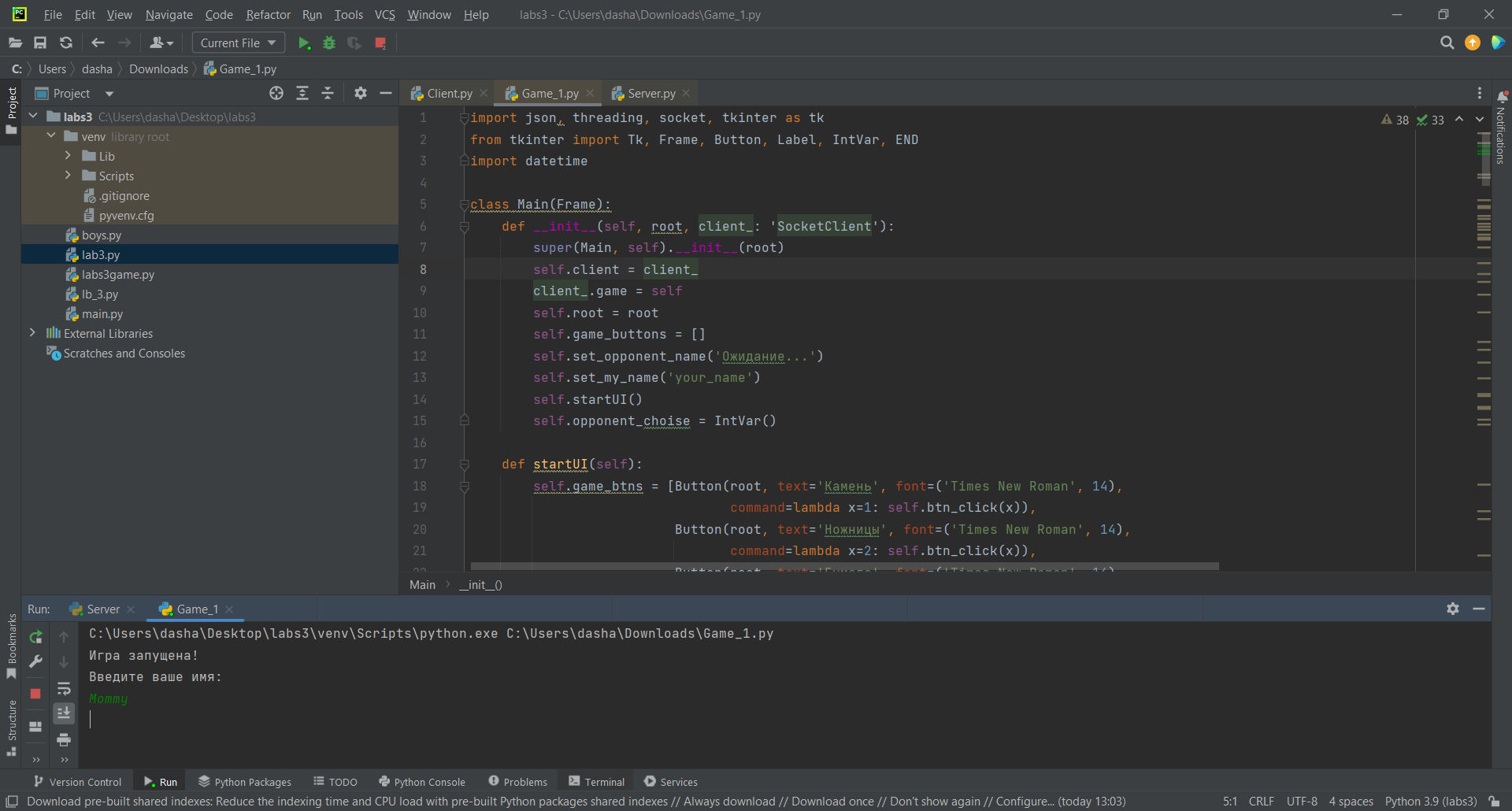


Рис.1 Запуск программы, ввод имени

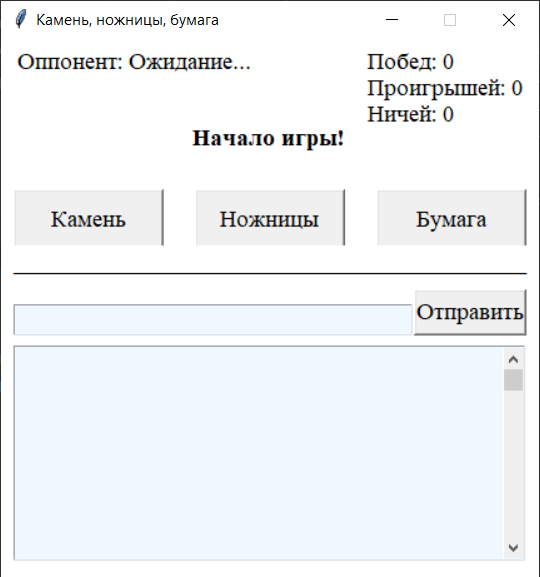


Рис. 2 Ожидание подключения оппонента

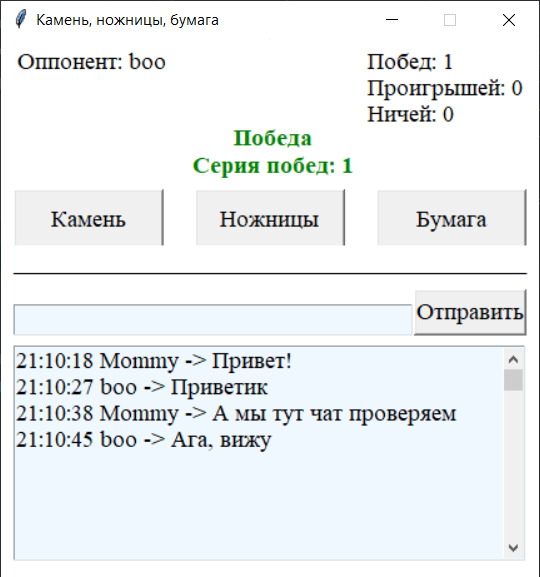


Рис. 3 Работа чата

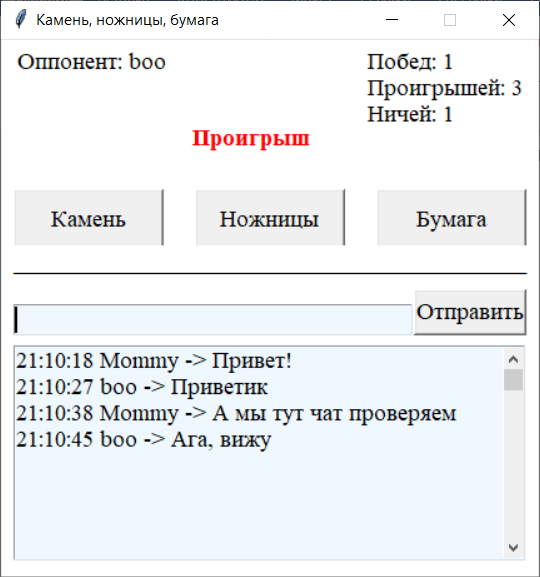


Рис. 4 Работа игры

# Заключение

В ходе проделанной работы была создана онлайн игра «Камень, Ножницы, Бумага», связывающая две вычислительных машины в локальной сети по протоколу связи TCP/IP. Разработка приложения потребовала изучения таких аспектов, как: IP-адресация, маршрутизация стека TCP/IP, работа с сокетами клиентской и серверной части.

В результате было разработано клиент-серверное приложение на языке программирования Python 3, интерфейс которого был реализован с помощью библиотеки tkinter (кроссплатформенная библиотека для разработки графического интерфейса на языке Python).

# Список литературы

1. Дмитрий ([@f1af](https://habr.com/ru/users/f1af/)), «Основы TCP/IP для будущих дилетантов» [Электронный ресурс]. URL: <https://habr.com/ru/post/326574/> (Дата обращения: 13.12.2022).
2. Зверев Д. А, «Что такое IP-адрес и маска подсети и зачем они нужны» [Электронный ресурс]. [URL: https://skillbox.ru/media/code/chto-takoe-ipadres-i-maska-podseti-i-zachem-oni-nuzhny/](URL:%20https://skillbox.ru/media/code/chto-takoe-ipadres-i-maska-podseti-i-zachem-oni-nuzhny/) (Дата обращения: 13.12.2022).
3. Линков В. В, «Всё об IP адресах и о том, как с ними работать» [Электронный ресурс]. URL: <https://habr.com/ru/post/350878/> (Дата обращения: 13.12.2022).
4. Полянин М. Н, «Пишем игру на Python» [Электронный ресурс]. [URL: https://thecode.media/pygames/](URL:%20https://skillbox.ru/media/code/chto-takoe-ipadres-i-maska-podseti-i-zachem-oni-nuzhny/) (Дата обращения: 13.12.2022).
5. Bot, «Основные принципы работы сетевой маршрутизации» [Электронный ресурс]. URL: <https://pc.ru/docs/network/routing> (Дата обращения: 13.12.2022).
6. T-Rex, «Руководство по стеку протоколов TCP/IP для начинающих» [Электронный ресурс]. URL: <https://selectel.ru/blog/tcp-ip-for-beginners/> (Дата обращения: 13.12.2022).