МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

«Нижегородский государственный архитектурно-строительный университет»

*Кафедра информационных систем и технологий*

**Курсовая работа по дисциплине**

**«Инфокоммуникационные системы и сети»**

**на тему:**

**«Игра»**

Выполнил студент группы ИС-29: Лицов Р. А.

Проверил старший преподаватель: Морозов Н. С.

Содержание

[Задание 3](#_Toc124962078)

[Теоретическая часть 4](#_Toc124962079)

[Программный код 7](#_Toc124962080)

[Пример работы приложения 17](#_Toc124962081)

[Список литературы 18](#_Toc124962082)

# **Задание**

Для игры написать сетевой код:

* для игры вдвоем (один - "сервер", второй - "клиент"),
* каждый отправляет сопернику результат своего выбора,
* результат подсчитывается только после и получения выбора оппонента и после собственного,
* добавить справа небольшое окно чата,
* игровые сообщения и сообщения чата не должны мешать друг другу.

# **Теоретическая часть**

CP/IP - это аббревиатура термина Transmission Control Protocol/Internet Protocol (Протокол управления передачей/Протокол Internet). В терминологии вычислительных сетей протокол — это заранее согласованный стандарт, который позволяет двум компьютерам обмениваться данными. Фактически TCP/IP не один протокол, а несколько. Именно поэтому вы часто слышите, как его называют набором, или комплектом протоколов, среди которых TCP и IP - два основных.

TCP/IP дает решение проблемы данными между двумя компьютерами, подключенными к одной и той же интрасети, но принадлежащими различным физическим сетям. Решение состоит из нескольких частей, причем каждый член семейства протоколов TCP/IP вносит свою лепту в общее дело. IP - самый фундаментальный протокол из комплекта TCP/IP - передает IP-дейтаграммы по интрасети и выполняет важную функцию, называемую маршрутизацией, по сути дела это выбор маршрута, по которому дейтаграмма будет следовать из пункта А в пункт B, и использование маршрутизаторов для "прыжков" между сетями.

Описание протоколов семейства TCP/IP с расшифровкой аббревиатур

* ARP (Address Resolution Protocol, протокол определения адресов): конвертирует 32-разрядные IP-адреса в физические адреса вычислительной сети, например, в 48-разрядные адреса Ethernet.
* FTP (File Transfer Protocol, протокол передачи файлов): позволяет передавать файлы с одного компьютера на другой с использованием TCP-соединений. В родственном ему, но менее распространенном протоколе передачи файлов - Trivial File Transfer Protocol (TFTP) - для пересылки файлов применяется UDP, а не TCP.
* ICMP (Internet Control Message Protocol, протокол управляющих сообщений Internet): позволяет IP-маршрутизаторам посылать сообщения об ошибках и управляющую информацию другим IP-маршрутизаторам и главным компьютерам сети. ICMP-сообщения "путешествуют" в виде полей данных IP-дейтаграмм и обязательно должны реализовываться во всех вариантах IP.
* IGMP (Internet Group Management Protocol, протокол управления группами Internet): позволяет IP-дейтаграммам распространяться в циркулярном режиме (multicast) среди компьютеров, которые принадлежат к соответствующим группам.
* IP (Internet Protocol, протокол Internet): низкоуровневый протокол, который направляет пакеты данных по отдельным сетям, связанным вместе с помощью маршрутизаторов для формирования Internet или интрасети. Данные "путешествуют" в форме пакетов, называемых IP-дейтаграммами.
* RARP (Reverse Address Resolution Protocol, протокол обратного преобразования адресов): преобразует физические сетевые адреса в IP-адреса.
* SMTP (Simple Mail Transfer Protocol, простой протокол обмена электронной почтой): определяет формат сообщений, которые SMTP-клиент, работающий на одном компьютере, может использовать для пересылки электронной почты на SMTP-сервер, запущенный на другом компьютере.
* TCP (Transmission Control Protocol, протокол управления передачей): протокол ориентирован на работу с подключениями и передает данные в виде потоков байтов. Данные пересылаются пакетами - TCP-сегментами, - которые состоят из заголовков TCP и данных. TCP - "надежный" протокол, потому что в нем используются контрольные суммы для проверки целостности данных и отправка подтверждений, чтобы гарантировать, что переданные данные приняты без искажений.
* UDP (User Datagram Protocol, протокол пользовательских дейтаграмм): протокол, не зависящий от подключений, который передает данные пакетами, называемыми UDP-дейтаграммами. UDP - "ненадежный" протокол, поскольку отправитель не получает информацию, показывающую, была ли в действительности принята дейтаграмма.

\* TCP/IP - это набор протоколов, которые позволяют физическим сетям объединяться вместе для образования Internet. TCP/IP соединяет индивидуальные сети для образования виртуальной вычислительной сети, в которой отдельные главные компьютеры идентифицируются не физическими адресами сетей, а IP-адресами.  
\* В TCP/IP используется многоуровневая архитектура, которая четко описывает, за что отвечает каждый протокол. TCP и UDP обеспечивают высокоуровневые служебные функции передачи данных для сетевых программ, и оба опираются на IP при передаче пакетов данных. IP отвечает за маршрутизацию пакетов до их пункта назначения.  
\* Данные, перемещающиеся между двумя прикладными программами, работающими на главных компьютерах Internet, "путешествуют" вверх и вниз по стекам TCP/IP на этих компьютерах. Информация, добавленная модулями TCP/IP на стороне отправителя, "разрезается" соответствующими TCP/IP-модулями на принимающем конце и используется для воссоздания исходных данных.

# **Программный код**

**Game.py**

import json

import random

from enum import IntEnum

from tkinter import Tk, Frame, Button, Label, END

import tkinter as tk

import threading

import socket

from typing import Optional

class Action(IntEnum):

Kamen = 0

Nojnici = 1

Bumaga = 2

class GameCommand:

def \_\_init\_\_(self, game: "Main", choice: Action):

self.choice = choice

self.game = game

def process\_button(self):

self.game.client.send("action", str(self.choice.value))

def \_\_call\_\_(self):

for btn in self.game.game\_buttons:

if btn["state"] == tk.DISABLED:

return

btn["state"] = tk.DISABLED

threading.Thread(target=self.process\_button).start()

class Main(Frame):

def \_\_init\_\_(self, root, client\_: "SocketClient"):

super(Main, self).\_\_init\_\_(root)

self.client = client\_

client\_.game = self

self.root = root

self.opponent\_name = ""

self.game\_buttons = []

self.game\_start\_label: Optional[Label] = None

self.game\_data\_label: Optional[Label] = None

self.opponent\_label: Optional[Label] = None

self.entry: Optional[tk.Entry] = None

self.txt: Optional[tk.Text] = None

self.button\_font = ("Times New Roman", 15)

self.mini\_button\_font = ("Times New Roman", 13)

self.win = self.draw = self.lose = 0

self.start\_iu()

def game\_data\_text(self):

return f"Побед: {self.win}\nПроигрышей:" f" {self.lose}\nНичей: {self.draw}"

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"Я -> {input\_text}\n")

self.txt.see("end")

self.txt.configure(state="disabled")

self.entry.delete(0, END)

self.client.send("chat", input\_text)

def start\_iu(self):

self.game\_buttons = [

Button(

self.root,

text="Камень",

font=self.button\_font,

command=GameCommand(self, Action.Kamen),

),

Button(

self.root,

text="Ножницы",

font=self.button\_font,

command=GameCommand(self, Action.Nojnici),

),

Button(

self.root,

text="Бумага",

font=self.button\_font,

command=GameCommand(self, Action.Bumaga),

),

]

self.game\_buttons[0].place(x=10, y=100, width=120, height=50)

self.game\_buttons[1].place(x=155, y=100, width=120, height=50)

self.game\_buttons[2].place(x=300, y=100, width=120, height=50)

self.master.bind("<Return>", self.send\_button)

self.game\_start\_label = Label(

self.root,

text="Начало игры!",

bg="#FFF",

font=("Times New Roman", 18, "bold"),

)

self.game\_data\_label = Label(

self.root,

justify="left",

font=self.mini\_button\_font,

text=self.game\_data\_text(),

bg="#FFF",

)

self.opponent\_label = Label(

self.root,

justify="right",

font=self.mini\_button\_font,

text=f"Оппонент: Нет",

bg="#FFF",

)

self.game\_start\_label.place(x=150, y=5)

self.game\_data\_label.place(x=5, y=5)

self.opponent\_label.place(x=145, y=55)

self.txt = tk.Text(

self.root, font=self.mini\_button\_font, width=47, height=8, bg="#ebd7ca"

)

self.txt.configure(state="disabled")

self.txt.place(x=10, y=160)

scrollbar = tk.Scrollbar(self.txt)

scrollbar.place(relheight=1, relx=0.958)

self.entry = tk.Entry(

self.root, font=self.mini\_button\_font, width=45, bg="#6e645d"

)

self.entry.place(x=0, y=335)

send = Button(

self.root,

text="Отправить",

font=self.mini\_button\_font,

command=self.send\_button,

width=9,

height=1,

)

send.place(x=340, y=325)

class SocketClient:

def \_\_init\_\_(self, name: str):

self.client = None

self.name = name

self.game = None

def result\_handler(self, message: str):

if message == "draw":

self.game.draw += 1

self.game.game\_start\_label.configure(text="Ничья")

if message == "win":

self.game.win += 1

self.game.game\_start\_label.configure(text="Победа")

if message == "lose":

self.game.lose += 1

self.game.game\_start\_label.configure(text="Проигрыш")

self.game.game\_data\_label.configure(text=self.game.game\_data\_text())

for btn in self.game.game\_buttons:

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.opponent\_label.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"{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\_\_":

main\_root = Tk()

main\_root.geometry("430x360+200+200")

main\_root.title("Камень, ножницы, бумага")

main\_root.resizable(False, False)

main\_root["bg"] = "#FFF"

nick = f"Player{random.randint(0, 100000)}"

print(nick)

client = SocketClient(name=nick)

app = Main(main\_root, client)

app.pack()

game\_thread = threading.Thread(target=main\_root.mainloop)

socket\_thread = threading.Thread(

target=client.socket\_start, args=("127.0.0.1", 8081)

)

socket\_thread.start()

game\_thread.run()

**Server.py**

import json

import socket

import threading

from typing import Optional, Union

class Result:

WIN = "win"

LOSE = "lose"

DRAW = "draw"

class RPSServer:

def \_\_init\_\_(self, host, port):

self.host = host

self.port = port

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

self.sock.bind((self.host, self.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()

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 = Result.LOSE

opponent\_result = Result.WIN

if user\_choice == opponent\_choice:

user\_result = Result.DRAW

opponent\_result = Result.DRAW

if (user\_choice + 1) % 3 == opponent\_choice:

user\_result = Result.WIN

opponent\_result = 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"],

}

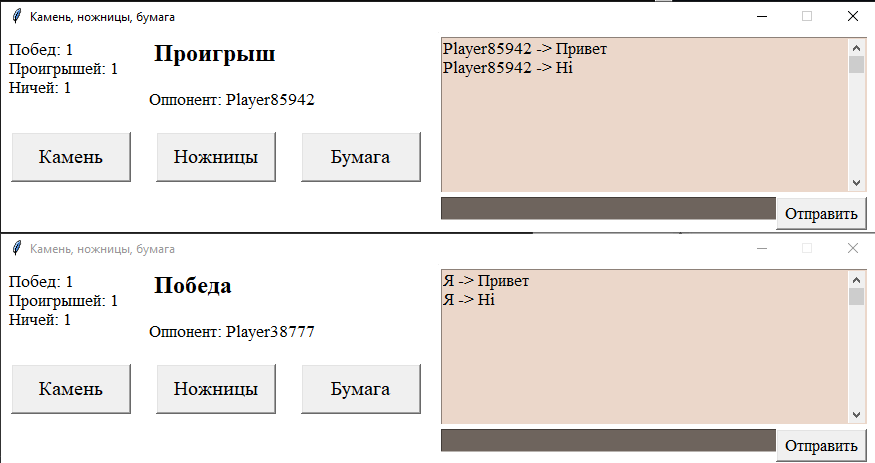
).encode()

)

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

RPSServer("127.0.0.1", 8081).listen()

# **Пример работы приложения**



# **Список литературы**

1. Руководство по TCP/IP для начинающих [Электронный ресурс] – URL: https://www.internet-technologies.ru/articles/rukovodstvo-po-tcp-ip-dlya-nachinayuschih.html