ФГБОУ ВО

ННГАСУ

Кафедра информационных систем и технологий

Курсовая работа

по виртуальным технологиям

**Разработка онлайн игры**

Выполнила студентка 3 курса группы ИС-29

Алексеева В.А.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

Проверил преподаватель

Морозов Н.С. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

Нижний Новгород

2022 год

**Содержание**

[Введение 3](#_Toc124994612)

[Задачи 3](#_Toc124994613)

[Теоретическая часть 4](#_Toc124994614)

[1. IP — Internet Protocol 5](#_Toc124994615)

[2. TCP/IP — Transmission Control Protocol/Internet Protocol 6](#_Toc124994616)

[3. UDP — UserDatagramProtocol 6](#_Toc124994617)

[4. FTP — File Transfer Protocol 6](#_Toc124994618)

[5. DNS 7](#_Toc124994619)

[6. HTTP — HyperTextTransferProtocol 7](#_Toc124994620)

[Интерфейс приложения 9](#_Toc124994621)

[Заключение 10](#_Toc124994622)

[Список литературы 11](#_Toc124994623)

[Приложение 12](#_Toc124994624)

[I. Серверная часть приложения 12](#_Toc124994625)

[II. Клиентская часть приложения 14](#_Toc124994626)

# Введение

Целью данной работы является написание сетевой игры, для закрепления и развития навыков полученных в ходе изучение по данной дисциплине.

Необходимо создать онлайн игру «Камень ножницы бумага», каждый отправляет сопернику результат своего выбора, результат подсчитывается после получения выбора оппонента и после собственного, добавить небольшое окно чата, сообщения не должны мешать друг другу.

# 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 – это процесс выбора маршрута в любой сети. Компьютерная сеть состоит из множества компьютеров, называемых узлами, и маршрутов или связей, соединяющих эти узлы. Связь между двумя узлами во взаимосвязанной сети может осуществляться по разным маршрутами. Маршрутизация – это процесс выбора наилучшего маршрута с использованием некоторых заранее установленных правил.

Маршрутизация повышает эффективность сетевой коммуникации. Сбои сетевой связи приводят к длительному ожиданию загрузки страниц веб-сайта для пользователей. Это также может привести к сбою серверов веб-сайтов из-за того, что они не могут обслуживать большое количество пользователей. Маршрутизация помогает минимизировать сбои сети, управляя трафиком данных, чтобы сеть могла использовать как можно больше своей пропускной способности без перегрузки.

Маршрутизатор – это сетевое устройство, которое соединяет вычислительные устройства и сети с другими сетями. Маршрутизаторы в основном выполняют три основные функции.

Маршрутизатор определяет путь данных при перемещении из источника в место назначения. Он пытается найти наилучший путь, анализируя сетевые метрики, такие как задержка, пропускная способность и скорость.

Маршрутизатор пересылает данные следующему устройству по выбранному пути, чтобы в конечном итоге доставить их в место назначения. Устройство и маршрутизатор могут находиться в одной или в разных сетях. Иногда маршрутизатор может отправлять копии одного и того же пакета данных по нескольким разным путям. Это позволяет уменьшить количество ошибок из-за потери данных, создать избыточность и управлять объемом трафика.

Данные перемещаются по любой сети в виде пакетов данных. Каждый пакет данных имеет заголовок, содержащий информацию о предполагаемом адресате пакета. По мере доставки пакета к месту назначения несколько маршрутизаторов могут перенаправить его несколько раз. Маршрутизаторы выполняют этот процесс миллионы раз в секунду с миллионами пакетов.

Когда приходит пакет данных, маршрутизатор сначала ищет свой адрес в таблице маршрутизации. Это похоже на то, как пассажир просматривает расписание автобусов, чтобы найти лучший автобусный маршрут до места назначения. Затем маршрутизатор пересылает или перемещает пакет в следующую точку сети.

Например, когда вы посещаете веб-сайт с компьютера в офисной сети, пакеты данных сначала отправляются на маршрутизатор офисной сети. Маршрутизатор ищет пакет заголовка и определяет место назначения пакета. Затем он просматривает свою внутреннюю таблицу и пересылает пакет – следующему маршрутизатору или другому устройству, например принтеру – внутри самой сети.

Протокол маршрутизации – это набор правил, определяющих, как маршрутизаторы идентифицируют и пересылают пакеты по сетевому пути. Протоколы маршрутизации сгруппированы в две отдельные категории: протоколы внутренних шлюзов и протоколы внешних шлюзов.

### IP — Internet Protocol

Протокол передачи, который первым объединил отдельные компьютеры в единую сеть. Самый примитивный в этом списке. Он является ненадёжным, т. е. не подтверждает доставку пакетов получателю и не контролирует целостность данных. По протоколу IP передача данных осуществляется без установки соединения.

Основная задача этого протокола — маршрутизация датаграмм, т. е. определение пути следования данных по узлам сети.

Популярная версия на текущий момент — IPv4 с 32-битными адресами. Это значит, что в интернете могут хранится 4.29 млрд адресов IPv4. Число большое, но не бесконечное. Поэтому существует версия IPv6, которая поможет решить проблему переполнения адресов, ведь уникальных IPv6 будет 2 ^ 128 адресов (число с 38 знаками).

### TCP/IP — Transmission Control Protocol/Internet Protocol

Это стек протоколов TCP и IP. Первый обеспечивает и контролирует надёжную передачу данных и следит за её целостностью. Второй же отвечает за маршрутизацию для отправки данных. Протокол TCP часто используется более комплексными протоколами.

### UDP — UserDatagramProtocol

Протокол, обеспечивающий передачу данных без предварительного создания соединения между ними. Этот протокол является ненадёжным. В нём пакеты могут не только не дойти, но и прийти не по порядку или вовсе продублироваться.

Основное преимущество UDP протокола заключается в скорости доставки данных. Именно поэтому чувствительные к сетевым задержкам приложения часто используют этот тип передачи данных.

### FTP — File Transfer Protocol

Протокол передачи файлов. Его использовали ещё в 1971 году — задолго до появления протокола IP. На текущий момент этим протоколом пользуются при удалённом доступе к хостингам. FTP является надёжным протоколом, поэтому гарантирует передачу данных.

Этот протокол работает по принципу клиент-серверной архитектуры. Пользователь проходит аутентификацию (хотя в отдельных случаях может подключаться анонимно) и получает доступ к файловой системе сервера.

### DNS

Это не только система доменных имён (DomainNameSystem), но и протокол, без которого эта система не смогла бы работать. Он позволяет клиентским компьютерам запрашивать у DNS-сервера IP-адрес какого-либо сайта, а также помогает обмениваться базами данных между серверами DNS. В работе этого протокола также используются TCP и UDP.

### HTTP — HyperTextTransferProtocol

Изначально протокол передачи HTML-документов. Сейчас же он используется для передачи произвольных данных в интернете. Он является протоколом клиент-серверного взаимодействия без сохранения промежуточного состояния. В роли клиента чаще всего выступает веб-браузер, хотя может быть и, например, поисковый робот. Для обмена информацией протокол HTTP в большинстве случаев использует TCP/IP.

HTTP имеет расширение HTTPS, которое поддерживает шифрование. Данные в нём передаются поверх криптографического протокола TLS.

В стеке TCP/IP используются три типа адресов: локальные (называемые также аппаратными), IP-адреса и символьные доменные имена.

В терминологии TCP/IP под локальным адресом понимается такой тип адреса, который используется средствами базовой технологии для доставки данных в пределах подсети, являющейся элементом составной интерсети. В разных подсетях допустимы разные сетевые технологии, разные стеки протоколов, поэтому при создании стека TCP/IP предполагалось наличие разных типов локальных адресов. Если подсетью интерсети является локальная сеть, то локальный адрес — это МАС - адрес. МАС - адрес назначается сетевым адаптерам и сетевым интерфейсам маршрутизаторов. МАС - адреса назначаются производителями оборудования и являются уникальными, так как управляются централизованно. Для всех существующих технологий локальных сетей МАС - адрес имеет формат 6 байт, например 11-AO-17-3D-BC-01.

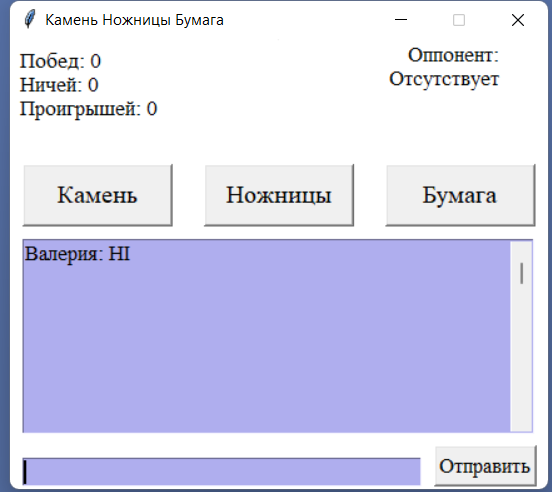
Однако протокол IP может работать и над протоколами более высокого уровня, например над протоколом IPX или Х.25. В этом случае локальными адресами для протокола IP соответственно будут адреса IPX и Х.25. Следует учесть, что компьютер в локальной сети может иметь несколько локальных адресов даже при одном сетевом адаптере. Некоторые сетевые устройства не имеют локальных адресов. Например, к таким устройствам относятся глобальные порты маршрутизаторов, предназначенные для соединений типа «точка-точка».

IP-адреса представляют собой основной тип адресов, на основании которых сетевой уровень передает пакеты между сетями. Эти адреса состоят из 4 байт, например 109.26.17.100. IP-адрес назначается администратором во время конфигурирования компьютеров и маршрутизаторов. IP-адрес состоит из двух частей: номера сети и номера узла. Номер сети может быть выбран администратором произвольно, либо назначен по рекомендации специального подразделения Internet (Internet Network Information Center, InterNIC), если сеть должна работать как составная часть Internet.

Обычно поставщики услуг Internet получают диапазоны адресов у подразделений InterNIC, а затем распределяют их между своими абонентами. Номер узла в протоколе IP назначается независимо от локального адреса узла. Маршрутизатор по определению входит сразу в несколько сетей. Поэтому каждый порт маршрутизатора имеет собственный IP-адрес. Конечный узел также может входить в несколько IP-сетей. В этом случае компьютер должен иметь несколько IP-адресов, по числу сетевых связей.

Таким образом, IP-адрес характеризует не отдельный компьютер или маршрутизатор, а одно сетевое соединение.

# Интерфейс приложения



# Заключение

Мы научились использовать локальную сеть для обмена данными,создали онлайн игру, изучили систему маршрутизации, а также систему стека TCP/IP. В работе мы провели настройку протокла транспортного уровня и настройку сокетов серверной и клиентской части игры. Таким образом мы выполнили цель нашей работы и все поставленные задачи.

# Список литературы

1. Курс «Компьютерные сети» [Электронныйресурс] URL: http://math.gsu.by/wp-content/uploads/courses/networks/построение\_больших\_сетей
2. Статья «Протоколы передачи данных»[Электронныйресурс] URL: https://tproger.ru/explain/protokoly-peredachi-dannyh-chto-jeto-kakie-byvajut-i-v-chjom-razlichija/
3. Документация IBM«Маршрутизация TCP/IP»[Электронныйресурс] URL:https://www.ibm.com/docs/ru/aix/7.1?topic=protocol-tcpip-routing
4. Очень простой чат(клиент/сервер) на Python.[Электронныйресурс] URL: https://xn--90aeniddllys.xn--p1ai/ochen-prostoj-chatklient-server-na-python/
5. Что такое маршрутизация? [Электронныйресурс] URL: <https://aws.amazon.com/ru/what-is/routing>

# Приложение

## Серверная часть приложения

**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"**, 55000).listen()

## Клиентская часть приложения

**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):  
 Rock = 0  
 Scissors = 1  
 Paper = 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\_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**}\nНичей: {**self.draw**}\nПроигрышей:" f" {**self.lose**}"  
  
 def** send\_button(self):  
 input\_text = self.entry.get()  
 **if not** input\_text:  
 **return** self.txt.configure(state=**"normal"**)  
 self.txt.insert(END, **f"Валерия: {**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.Rock),  
 ),  
 Button(  
 self.root,  
 text=**"Ножницы"**,  
 font=self.button\_font,  
 command=GameCommand(self, Action.Scissors),  
 ),  
 Button(  
 self.root,  
 text=**"Бумага"**,  
 font=self.button\_font,  
 command=GameCommand(self, Action.Paper),  
 ),  
 ]  
  
 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\_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:\nОтсутствует"**,  
 bg=**"#FFF"**,  
 )  
 self.game\_data\_label.place(x=5, y=5)  
 self.opponent\_label.place(x=300, y=0)  
  
 self.txt = tk.Text(  
 self.root, font=self.mini\_button\_font, width=45, height=8, bg=**"#AFAEEE"** )  
 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=35, bg=**"#AFAEEE"** )  
 self.entry.place(x=10, y=335)  
 send = Button(  
 self.root,  
 text=**"Отправить"**,  
 font=self.mini\_button\_font,  
 command=self.send\_button,  
 width=8,  
 height=1,  
 )  
 send.place(x=339, 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:\n {**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"my opponent"** 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"**, 55000)  
 )  
 socket\_thread.start()  
 game\_thread.run()