МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАВЧАЛЬНО-НАУКОВИЙ КОМПЛЕКС

«ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ»

НАЦІОНАЛЬНОГО ТЕХНІЧНОГО УНІВЕРСИТЕТУ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

КАФЕДРА МАТЕМАТИЧНИХ МЕТОДІВ СИСТЕМНОГО АНАЛІЗУ

Проект

З дисципліни «Комп’ютерні мережі»

**Виконав:** *студент 3-го курсу*

*гр. КА-71*

*Гульчук М. В.*

**Прийняв:** *Кухарєв С.О*

Київ 2020р.

Постановка задачі

Мета:

Розробити програмний продукт що дозволить проводити вікторину, в якій може приймати участь деяка кількість користувачів.

Логіка вікторини:  
Вікторина є доволі простою: Кожен гравець має ввести ім’я себе або своєї команнди. Коли всі зайшли, хтось розпочинає вікторину і на меті – найпершими надати правильну відповідь. Хто з раунд набрав найбільшу кількість балів, той переміг. Далі можна знов почати вікторину

Реалізація

Засоби:

Надамо перелік технологій, що будуть використані для написання продукту.

Клієнт. Для клієнта будемо використовувати: kivy– розмітка, інтерфейс, python – інтерпретатор, sockets – засіб зв’язку з сервером.

Сервер. Для серверу використовуватимемо мову програмування python. Для того, щоб реалізувати спілкування між клієнтами на сервером використаємо sockets.

Перелік підзадач:

Розіб’ємо завдання на підзадачі, та до деяких надамо можливі особливостями реалізації.

1. Створити діагностику серверу.

У будь-який момент вікторини якщо сервер вимикається, вікторина має повідомити про це користувача та вжити необхідних заходів згідно з інструкцією користувача(Вийти або спробувати ще раз)

1. Введення ім’я гравця/команди та передача його на сервер

Кожен клієнт має ввести своє ім’я та передати йогона сервер. При чому бажано зробити це один раз при запуску програми задля того, щоб не перевводити ім’я. випадку невдач серверу.

1. Тому хто ввів ім’я необхідно під’єднатися до серверу якщо той працює.

Дане повідомлення бачить лише один учасник ( хто малює ).

1. Коли гравець під’єднаний до серверу і вікторина не почалась, гравець маж мати можливість переглянути уже існуючих гравців, а також мати можливість розпочати тест.
2. Коли тест розпочато кожен гравець має мати можливість відповісти на питання рівно один раз

Усі учасники мають змогу бачити, хто ввійшов, чи вийшов з кімнати. Дане повідомлення буде відображатись у чаті.

1. Після закінчення вікторини кожен гравець бачить повідомлення про підсумок вікторини

Право на малювання має лише той, кому відомо загадане слово. Усі інші учасник не можуть малювати.

1. У разі коли гравець заходить до вікторини коли вона почалась, він має мати можливість підключитися до вікторини.

Якщо учасник має декілька відкритих вкладок з кімнатою, то на кожну вкладку надходять повідомлення.

Опис класів та функцій:

Клієнт.

Клі’єнта розбито на три частини: два класи і розмітку.

- Клас логіки клієнту – взаємодіє з сервером

- Клас логіки інтерфейсу – змінює інтерфейс згідно з команами логіки

- Розмітка інтерфейсу – розмітка інтерфейсу клієнту

Логіка:

clientlogic.py

Містить клас ClientLogic, та багато допоміжних функцій всередині для зв’язку з сервером та обміну даними з ним

А саме:

try\_to\_connect – під’єднується до серверу  
cr\_msg – створює повідомлення на відправку

check\_socket – перевіряє підключеність до сокету

receive\_msg – отримує повідомлення

assert\_type – перевіряж чи повідомлення потрібного типу

end\_session – закриває коннекшн з сервером

check\_if\_started – перевіряє чи стартанув квіз

decode\_list – декодує список рядків

check\_question – перевіряє питання

decode\_quest – декодує питання

get\_winner – отримує переможця

get\_income – отримує будь-яке вхідне повідомлення і перенаправляє його до інтерфейсу

**import** socket  
**import** select  
  
**class** ClientLogic:  
 **def** \_\_init\_\_(self, ):  
 self.HEADER\_LENGTH = 10  
 self.exit\_commands = (**"close"**, **"exit"**, **"quit"**)  
 self.msg\_types = {  
 **"j"**: **"Username"**, *# Information about the username immediately after connecting to the server* **"c"**: **"command"**, *# cammand for the server. Available commands: start* **"i"**: **"inform"**, *# inform clients about quiz start or end* **"q"**: **"question"**, *# ask question during the quiz* **"a"**: **"answer"**, *# send answer for the question* **"w"**: **"winner"**, *# announce the winner* **"e"**: **"exit"**, *# cancel the connection* **"o"**: **"other"** *# other type. Temporary type to adopt previous version* }  
 self.client\_sates = {  
 **"connecting"**: 0,  
 **"identification"**: 1,  
 **"waiting\_for\_quiz"**: 2,  
 **"answering\_the\_question"**: 3,  
 **"waiting\_for\_the\_next\_question"**: 4,  
  
 **"watching\_the\_results"**: 5,  
 }  
 self.IP = **"213.133.161.33"***#"127.0.0.1"* self.PORT = 1234  
 self.username=**"Noname"** *# defining vars for clients logic* roles = {**"w"**, *# to wait for the quiz* **"c"**, *# to write command for the server* **"o"** *# to just observe all the process* }  
  
 **def** try\_to\_connect(self):  
 **try**:  
 self.client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 self.client\_socket.connect((self.IP, self.PORT))  
 self.client\_socket.setblocking(**False**)  
 self.send\_msg(self.username, **"j"**)  
 print(**"Successfully connected to the server!"**)  
 **return True  
 except** Exception **as** ex:  
 print(**"Failed connecting to the server!"**)  
 print(str(ex))  
 **return False  
  
 def** cr\_header(self, str, msg\_type):  
 **assert** len(msg\_type) == 1  
 **return f"{**len(str)**:<{**self.HEADER\_LENGTH**}}"**.encode() + msg\_type.encode()  
  
 **def** cr\_msg(self, msg, msg\_type=**"o"**):  
 **return** self.cr\_header(msg, msg\_type) + msg.encode()  
  
 **def** receive\_msg(self):  
 **try**:  
 msg\_header = self.client\_socket.recv(self.HEADER\_LENGTH).decode()  
 **if** msg\_header == **""**:  
 **return** (**"closed"**, **"e"**) *# connection is closed* msg\_len = int(msg\_header)  
 msg\_type = self.client\_socket.recv(1).decode()  
 msg = self.client\_socket.recv(msg\_len).decode()  
 answ = (msg, msg\_type)  
 print(**f"\t\t\tReceived msg: {**msg**}. type: {**msg\_type**}"**)  
 **return** answ  
 **except**:  
 **return** (**""**, **"continue"**)  
  
 **def** check\_socket(self):  
 **try**:  
 socket, \*\_ = select.select([self.client\_socket], [], [self.client\_socket], 0)  
 **if** socket == []:  
 **return False  
 return True  
 except**:  
 **return False  
  
  
 def** assert\_type(self, expected, real, msg):  
 **if** real != expected:  
 print(**f"Unexpected msg type {**real**}('{**expected**}' expected) \n"  
 f"from {**self.username**} with payload {**msg**}. "**)  
 **return False  
 return True  
  
 def** assert\_types(self, expected, real, msg):  
 **if** real **not in** expected:  
 print(**f"Unexpected msg type {**real**}('{**expected**}' expected) \n"  
 f"from {**self.username**} with payload {**msg**}. "**)  
 **return False  
 return True  
  
 def** end\_session(self, send\_msg=**False**):  
 **if** send\_msg:  
 self.client\_socket.send(self.cr\_msg(**"Closing connection"**, **"e"**))  
 self.client\_socket.close()  
  
 **def** send\_msg(self, msg, type):  
 self.client\_socket.send(self.cr\_msg(msg, type))  
 print(**f"\t\tSent message {**self.cr\_msg(msg, type)**}"**)  
 **def** start(self):  
 self.send\_msg(**"start"**, **"c"**)  
 **def** check\_if\_started(self):  
 msg, type = self.receive\_msg()  
 print(**f"Checking if it is start {**msg**} {**type**}"**)  
 **if** self.assert\_types(**"i"**, type, msg):  
 **if** msg == **"start"**:  
 **return True  
 if** msg == **"already"**:  
 **return "already"  
 return False  
 def** decode\_list(self, msg):  
 users = []  
 **while**(len(msg)>0):  
 h = self.HEADER\_LENGTH  
 msg\_len = int(msg[:h])  
 msg = msg[h + 1:]  
 user = msg[:msg\_len]  
 msg = msg[msg\_len:]  
 users.append(user)  
 **return** users  
  
 **def** check\_question(self):  
 quest, type = self.receive\_msg()  
 **if** type == **"e"**:  
 print(**"Connection closed by the server"**)  
 self.end\_session()  
 **return ""**,**"e"  
 if** type == **"continue"**:  
 **return ""**, **"continue"  
 if not** self.assert\_types((**"q"**, **"i"**, **"w"**), type, quest):  
 print(**f"Closing with {**quest**} {**type**}"**)  
 self.end\_session(send\_msg=**True**)  
 **return ""**, **"e"  
 if** quest == **"end" and** type == **"i"**:  
 print(**"The quiz is ended. Now, lets wait for the next round"**)  
 **return False**, **False  
 if** type == **"q"**:  
 quest = self.decode\_quest(quest)  
 **return** quest, type  
 **def** decode\_quest(self, q):  
 h = self.HEADER\_LENGTH  
 quest\_len = int(q[:h])  
 q = q[h+1:]  
 quest = q[:quest\_len]  
 q = q[quest\_len:]  
 answs = []  
 **for** i **in** range(4):  
 answ\_len = int(q[:h])  
 q = q[h + 1:]  
 answ = q[:answ\_len]  
 q = q[answ\_len:]  
 answs.append(answ)  
 **return** (quest, answs)  
 **def** check\_winner(self):  
 quest, type = self.receive\_msg()  
 **if** type == **"e"**:  
 print(**"Connection closed by the server"**)  
 self.end\_session()  
 **return ""**, **"e"  
 if not** self.assert\_type(**"w"**, type, quest):  
 self.end\_session(send\_msg=**True**)  
 **return ""**, **"e"  
 if** type == **"w"**:  
 **return** quest, type  
 **def** get\_income(self):  
 if\_there = self.check\_socket()  
 **if** if\_there **is False**:  
 **return "empty"**, **False  
 else**:  
 msg, type = self.receive\_msg()  
 **if** type == **"e"**:  
 **return** msg, **"e"  
 if** type == **"continue"**:  
 **return "continue"**, **False  
 if** type == **"i"**:  
 **if** msg == **"start"**:  
 **return "start"**, **"i"  
 if** msg == **"already"**:  
 **return "already"**, **"i"  
 if** msg == **"end"**:  
 **return "end"**, **"i"  
 if** type == **"w"**:  
 **return** msg, type  
 **if** type == **"W"**:  
 **return** msg, type  
 **if** type == **"q"**:  
 **return** self.decode\_quest(msg), **"q"  
 if** type == **"o"**:  
 **return** msg, **"o"  
 if** type == **"u"**:  
 **return** self.decode\_list(msg), **"u"  
 return "Unrecognized type"**, **"e"**

інтерфейс:

**from** kivy.app **import** App  
**from** kivy.graphics.context **import** Clock  
**from** kivy.uix.screenmanager **import** ScreenManager, Screen  
**from** kivy.properties **import** ObjectProperty  
**from** kivy.lang **import** Builder  
**from** clientlogic **import** ClientLogic  
**import** sys  
  
  
  
**class** WelcomeWindow(Screen):  
 username = ObjectProperty(**None**)  
 **def** submit(self):  
 print(self.username.text.strip() + **" was written when submitting"**)  
 logic.username=self.username.text.strip()  
 **if** (len(self.username.text.strip()) == 0):  
 **return** app.title = **"QUIZ IT! ("** + logic.username + **")"  
 if not** logic.try\_to\_connect():  
 wm.current = **"connError"  
 else**:  
 wm.current = **"wait"  
  
class** WaitWindow(Screen):  
 info = ObjectProperty(**None**)  
 users = ObjectProperty(**None**)  
 **def** \_\_init\_\_(self, \*\*kwargs):  
 super().\_\_init\_\_(\*\*kwargs)  
 **def** start(self):  
 logic.start()  
 **def** process(self, msg):  
 **pass  
 def** setUsers(self, usrs):  
 self.users.text = **"List of users\n"  
 for** usr **in** usrs:  
 self.users.text += usr + **"\n"  
  
  
  
  
class** QuizWindow(Screen):  
 question = ObjectProperty(**None**)  
  
 b1 = ObjectProperty(**None**)  
 b2 = ObjectProperty(**None**)  
 b3 = ObjectProperty(**None**)  
 b4 = ObjectProperty(**None**)  
  
 **def** \_\_init\_\_(self, \*\*kwargs):  
 super().\_\_init\_\_(\*\*kwargs)  
  
 **def** button\_pressed(self, answer):  
 print(**"button "** + answer + **" pressed"**)  
 **if not** self.answered:  
 logic.send\_msg(answer, **"a"**)  
 self.answered = **True  
 if not** self.question.text.endswith(**"Wait for result now..."**):  
 self.question.text = self.question.text + **"\nWait for result now..."  
  
 def** redraw\_quest(self, q):  
 self.answered = **False** self.question.text = q[0]  
 self.b1.text = q[1][0]  
 self.b2.text = q[1][1]  
 self.b3.text = q[1][2]  
 self.b4.text = q[1][3]  
  
 **def** process\_quest(self, q):  
 self.redraw\_quest(q)  
 print(**"Draw a question!"**)  
 **return  
  
 def** process\_win(self, w):  
 **if** w == logic.username:  
 w = **"Correct!"  
 else**:  
 w = w + **" gave the correct answer"** self.question.text = w  
  
**class** ResultWindow(Screen):  
 winner = ObjectProperty(**None**)  
 **def** \_\_init\_\_(self, winnerName, \*\*kwargs):  
 super().\_\_init\_\_(\*\*kwargs)  
 self.winnerName = winnerName  
 winner = ObjectProperty(**None**)  
 **def** return\_start(self, dt):  
 wm.current = **"wait"  
 def** process(self, msg):  
 print(**f"ResultWindow received something {**msg**}"**)  
 **if** msg.split(**" "**)[0] != logic.username:  
 self.winner.text = msg + **" Won\n"** self.winner.text += logic.username + **", "** + **"keep trying\n"  
 if** msg.split(**" "**)[0] == logic.username:  
 self.winner.text = **f"Congrats! You Won!\n{**msg**}"** Clock.schedule\_once(self.return\_start, 5)  
  
**class** ConnErrWindow(Screen):  
 **def** try\_again(self):  
 print(**"trying again..."**)  
 **if** logic.try\_to\_connect():  
 print(**"succeed"**)  
 wm.current = **"wait"  
 else**:  
 print(**"failed"**)  
 wm.current = **"connError"  
  
class** WindowManager(ScreenManager):  
 **pass**Builder.load\_file(**"client.kv"**)  
  
  
wm = WindowManager()  
logic = ClientLogic()  
  
  
waitWindow = WaitWindow(name=**"wait"**)  
screens = [ConnErrWindow(name = **"connError"**), WelcomeWindow(name = **"welcome"**), waitWindow, QuizWindow(name=**"quiz"**), ResultWindow(winnerName=**"Nobody"**, name=**"result"**)]  
**for** screen **in** screens:  
 wm.add\_widget(screen)  
  
wm.current = **"welcome"  
  
  
  
class** ClientApp(App):  
 **def** build(self):  
 self.title = **"QUIZ IT!"** Clock.schedule\_interval(self.my\_callback, 0.2)  
 **return** wm  
 **def** exit(self):  
 logic.end\_session()  
 sys.exit()  
 **def** my\_callback(self, dt):  
 msg, msg\_type = logic.get\_income()  
 **if** msg\_type **is False**:  
 **return  
 if** msg\_type == **"e"**:  
 **if** wm.current == **"connError"**:  
 **return  
 if** msg == **"closed"**:  
 wm.current = **"connError"  
 return** logic.end\_session()  
 app.stop()  
 sys.exit()  
 **if** msg\_type == **"i"**:  
 **if** msg == **"start"**:  
 wm.current = **"quiz"  
 return  
 if** msg == **"already"**:  
 wm.current = **"quiz"  
 return  
 if** msg == **"end"**:  
 wm.current = **"result"  
 return  
 if** msg\_type == **"W"**:  
 wm.current = **"result"** wm.current\_screen.process(msg)  
 **return  
 if** msg\_type == **"q"**:  
 wm.current = **"quiz"** wm.current\_screen.process\_quest(msg)  
 **return  
 if** msg\_type == **"w"**:  
 wm.current = **"quiz"** wm.current\_screen.process\_win(msg)  
 **return  
 if** msg\_type == **"o"**:  
 **if** type(msg) == type(123):  
 **if** msg == **"gotowait"**:  
 wm.current = **"wait"  
 return** print(**"Got msg\_type 'o' but unknown instructions"**)  
 print(**"Got msg\_type 'o' but unknown instructions"**)  
 **if** msg\_type == **"u"**:  
 waitWindow.setUsers(msg)  
 **return** print(**f"Got unrecognized message {**msg**} of msg\_type {**msg\_type**}"** )  
 *#pdb.set\_trace()  
 #wm.current\_screen.my\_callback(dt)***if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 app = ClientApp()  
 app.run()

Розмітка:

Складається з 5ти вікон:

-вікно помидки

-вікно вводу імені

-вікно очікування

-вікно вікторини

-вікно оголошення переможця

<Label>:  
 bcolor: [0, 0, 0, 0]  
  
 canvas.before:  
 Color:  
 rgba: self.bcolor  
 font\_size: 20  
 halign: 'center'  
 valign: 'middle'  
<Button>:  
 background\_color: (0, 1, 0, .3)  
  
<ConnErrWindow>:  
 name: "connError"  
 GridLayout:  
 cols:1  
 Label:  
 text: "It seems like server shut down.\n It is likely server is shut down\nTry to Reconnect or exit?"  
 GridLayout:  
 cols: 2  
 Button:  
 text: "Reconnect"  
 on\_release: root.try\_again()  
 Button:  
 text: "Exit program"  
 on\_release: app.exit()  
<WelcomeWindow>:  
 name: "welcome"  
 username: username  
 GridLayout:  
 cols:1  
 Label:  
 text: "Welcome to the quiz application!\n Please, enter you nickname to continue"  
 TextInput:  
 halign: "center"  
 on\_text\_validate: root.submit()  
 font\_size: 40  
 multiline: False  
 background\_color: [0.5, 0.5, 0.5, 1]  
 id: username  
 Button:  
 text: "Enter"  
 on\_release: root.submit()  
<WaitWindow>:  
 name: "wait"  
 users: users  
 info:info  
 GridLayout:  
 cols:2  
 Label:  
 id: info  
 text: "Waiting for the quiz!\n Press 'Start' to start or join lasting quiz"  
 Label:  
 id: users  
 text: "List of users\n"  
 Button:  
 text: "Start"  
 on\_release: root.start()  
 Label:  
 text: "Name of the quiz:\nStandard"  
<QuizWindow>:  
 name: "quiz"  
 question: question  
 b1: b1  
 b2: b2  
 b3: b3  
 b4: b4  
 GridLayout:  
 cols:1  
 Label:  
 id: question  
 text: "Loading..."  
 GridLayout:  
 cols:2  
 Button:  
 id: b1  
 text: "Loading..."  
 on\_release: root.button\_pressed(b1.text)  
 Button:  
 id:b2  
 text: "Loading..."  
 on\_release: root.button\_pressed(b2.text)  
 Button:  
 id: b3  
 text: "Loading..."  
 on\_release: root.button\_pressed( b3.text )  
 Button:  
 id: b4  
 text: "Loading..."  
 on\_release: root.button\_pressed(b4.text)  
<ResultWindow>  
 name: "result"  
 winner: winner  
 GridLayout:  
 cols:1  
 Label:  
 font\_size: 30  
 id: winner  
 text: "Winner is "

Сервер містить тільки логіку обробки запитів і відповіді на них:

(Немає класів)

Містять функцію перевірки вхідних повідомлень від серверу що обробляє його в залежності від стану.

Є два стану: очікування на старт опитування, та проведення самого опитування.

**import** operator  
**import** socket  
**import** select  
**import** time  
  
  
*#defining data for the quiz*questions = [(**"What is the world’s most heavy land mammal?"**,  
 **"Hippopotamus"**, [**"Hippopotamus"**, **"Elephant"**, **"Giraffe"**, **"Gaur"**]),  
 (**'Which Middle Eastern city is also the name of a type of artichoke'**,  
 **'Jerusalem'**, [**"Jerusalem"**, **"Istanbul"**, **"Tehran"**, **"Dubai"**]),  
 (**'The Velocipede was a nineteenth-century prototype of what?'**,  
 **'a Bicycle'**, [**"a Plane"**, **"a Boat"**, **"a Car"**, **"a Bicycle"**])]  
TIME\_FOR\_QUESTION = 10  
TIME\_FOR\_LOCAL\_WINNER = 3  
  
quiz\_started = **False***#defining protoclols parameters*HEADER\_LENGTH = 10  
exit\_commands = (**"close"**, **"exit"**, **"quit"**)  
msg\_types = {  
 **"j"**: **"Username"**, *#Information about the username immediately after connecting to the server* **"c"**: **"command"**, *#command for the server. Available commands: start* **"i"**: **"inform"**, *#inform clients about quiz start or end* **"q"**: **"question"**, *# ask question during the quiz* **"a"**: **"answer"**, *# send answer for the question* **"w"**: **"winner"**, *# announce the winner* **"e"**: **"exit"**, *# cancel the connection* **"o"**: **"other"**, *# other type. Temporary type to adopt previous version* **"W"**: **"Winner"**, *#announcing winner of the round* **"u"**: **"users"** }  
  
  
**def** cr\_header(str, msg\_type):  
 **assert** len(msg\_type) == 1  
 **return f"{**len(str)**:<{**HEADER\_LENGTH**}}"**.encode() + msg\_type.encode()  
**def** cr\_msg(msg, msg\_type):  
 **return** cr\_header(msg, msg\_type) + msg.encode()  
**def** receive\_msg(socket):  
 **try**:  
 msg\_header = socket.recv(HEADER\_LENGTH).decode()  
 **if** msg\_header == **""**:  
 **return** (**""**, **"e"**) *# connection is closed* msg\_len = int(msg\_header)  
 msg\_type = socket.recv(1).decode()  
 msg = socket.recv(msg\_len).decode()  
 answ = (msg, msg\_type)  
 print(**f"\t\t\tReceived msg: {**msg**}. type: {**msg\_type**}"**)  
 **return** answ  
 **except**:  
 **return** (**""**, **"continue"**)  
  
**def** accept\_client(socket, sockets\_list, clients):  
 client\_socket, client\_address = socket.accept()  
 user, type = receive\_msg(client\_socket)  
 **if** type != **"j"**:  
 print(**f"Connecting failed: type '{**type**}' insted of 'j' in header"** )  
 **return False  
 if** user **is False**:  
 **return False** sockets\_list.append(client\_socket)  
 clients[client\_socket] = user  
 print((**f"Accepted new connectinon from {**client\_address[0]**} : {**client\_address[1]**}"**,  
 **f'{**user**}'**))  
 send\_users()  
 **return True  
  
def** assert\_type(expected, real, user, msg):  
 **if** real != expected:  
 print(**f"Unexpected msg type '{**real**}' ('{**expected**}' expected) \n"  
 f"from {**user**} with payload {**msg**}. "**)  
 **return False  
 return True  
  
def** closed\_connection(notified\_socket):  
 **global** sockets\_list  
 **global** clients  
 print(**f'Closed connection from {**clients[notified\_socket]**}'**)  
 **try**:  
 sockets\_list.remove(notified\_socket)  
 **del** clients[notified\_socket]  
 send\_users()  
 **except**:  
 print()  
  
**def** broadcast(msg, msg\_type):  
 **global** clients  
 **global** sockets\_list  
 close = []  
 **for** client **in** clients.keys():  
 **try**:  
 client.send(cr\_msg(msg, msg\_type))  
 **except**:  
 close.append(client)  
 **for** i **in** range(len(close)-1, -1):  
 closed\_connection(close[i])  
  
 print(**f'Broadcasting message "{**msg**}" type "{**msg\_type**}"'** )  
**def** send(msg, client, msg\_type):  
 **global** clients  
 **global** sockets\_list  
 **try**:  
 client.send(cr\_msg(msg, msg\_type))  
 **except**:  
 print(**f'Failed to send message "{**msg**}" type "{**msg\_type**}" to {**clients[client]**}'**)  
 closed\_connection(client)  
 **return** print(**f'Sent message "{**msg**}" type "{**msg\_type**}" to {**clients[client]**}'** )  
  
**def** gen\_quest(q):  
 quest = cr\_msg(q[0], **"q"**).decode()  
 answs = [cr\_msg(i, **"a"**).decode() **for** i **in** q[2]]  
 res = quest  
 **for** i **in** answs:  
 res += i  
 **return** res  
  
**def** send\_users():  
 answ = **""  
 for** i **in** clients.values():  
 answ += cr\_msg(i, **"u"**).decode()  
 broadcast(answ, **"u"**)  
  
  
*#defining data for TCP and IP protocols*IP = **"127.0.0.1"**PORT = 1234  
  
server\_soket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
server\_soket.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1) *# allowing trying to connect to the port several times  
  
#connecting to the port and starting listening to it*server\_soket.bind((IP, PORT))  
server\_soket.listen()  
  
*#storing all the sockets*sockets\_list = [server\_soket]  
clients = {}  
  
  
**def** process\_income(q=**None**):  
 **global** status  
 **global** correct\_answer\_recieved  
 **global** winner  
 **global** sockets\_list  
 **global** clients  
 **global** quiz\_started  
 **global** read\_sockets, exception\_sockets  
 **global** winner  
 read\_sockets, exception\_sockets = **None**, **None** print(**f"Current status is {**status**}"**)  
 **if** status == **"wait"**:  
 read\_sockets, \_, exception\_sockets = select.select(sockets\_list, [], sockets\_list)  
 **elif** status == **"quiz"**:  
 print(**f"Current status is {**status**}"**)  
 read\_sockets, \_, exception\_sockets = select.select(sockets\_list, [], sockets\_list, 0.01)  
 **else**:  
 print(**"Status error!"**)  
  
 **for** notified\_socket **in** read\_sockets:  
 print(**f"Current status is {**status**}"** )  
 **if** notified\_socket == server\_soket: *# accepting new connection* **if not** accept\_client(server\_soket, sockets\_list, clients):  
 print(**f"Error with connecting client"**)  
 **else**:  
 answ, type = receive\_msg(notified\_socket)  
 user = clients[notified\_socket]  
 **if** type == **"e"**:  
 closed\_connection(notified\_socket)  
 **return  
 if** type == **"continue"**:  
 **return  
 if** type == **"c" and** answ == **"start"**:  
 **if** status == **"wait"**:  
 broadcast(**"start"**, **"i"**)  
 status = **"quiz"** quiz\_started = **True** print(**"THE QUIZ IS STARTED"**)  
 **return  
 if** status == **"quiz"**:  
 send(**"start"**, notified\_socket, **"i"**)  
 send(gen\_quest(q), notified\_socket, **"q"**)  
 quiz\_started = **True  
 return  
  
  
 if** type == **"a"**:  
 **if** status == **"wait"**:  
 send(**"gotowait"**, notified\_socket, **"o"**)  
 **return  
 elif** status == **"quiz"**:  
 print(**f"Received an answer from {**user**}: {**answ**} on {**int(time.time() - t)**} seconds"**)  
 **if** correct\_answer\_recieved:  
 print(**"But correct answer already received"**)  
 **return  
 if** answ == q[1]:  
 print(**"And that's right"**)  
 correct\_answer\_recieved = **True  
 if not** user **in** countScore.keys():  
 countScore[user] = 0  
 countScore[user] = countScore[user] + 1  
 winner = user  
 **return  
 else**:  
 print(**"And it's wrong!"**)  
 **return  
 else**:  
 print(**"Status error"**)  
 **return** print(**f"Unrecognized income type: {**type**}, msg: {**answ**}"**)  
  
  
  
 **for** notified\_socket **in** exception\_sockets:  
 closed\_connection(notified\_socket)  
  
  
status = **"wait"**run = **True  
while** run: *#main loop* print(**"Accepting all connections and listening to the commands..."**)  
 status = **"wait"** quiz\_started = **False  
 while not** quiz\_started: *#accepting all connections and listening to the commands* process\_income()  
  
 *# #print(f"Received command from {user}: {msg}")  
 # if msg == "start":  
 # quiz\_started = True  
 # broadcast("start", clients, "i")  
 # print("THE QUIZ IS STARTED")  
 # time.sleep(1)* countScore = {clientName:0 **for** clientName **in** clients.values()}  
  
 status = **"quiz"  
 for** q **in** questions:  
 winner = **"Friendship"** print(**f"Asking the question: {**q**}"**)  
 broadcast(gen\_quest(q), **"q"**)  
 t = time.time()  
 *#listening to the answers* correct\_answer\_recieved = **False  
 while** (time.time() - t) < TIME\_FOR\_QUESTION **and not** correct\_answer\_recieved:  
 print(**"Going to check the income..."**)  
 process\_income(q=q)  
  
  
 *#announcing the winner* broadcast(winner, **"w"**)  
  
 print(**f"time: {**time.time() - t**} "**)  
 time.sleep(TIME\_FOR\_LOCAL\_WINNER)  
  
 broadcast(**"end"**, **"i"**)  
 overall\_winner = max(countScore.items(), key=operator.itemgetter(1))[0]  
 max\_score = max(countScore.items(), key=operator.itemgetter(1))[1]  
 number\_of\_winners = sum([int(v==max\_score) **for** \_,v **in** countScore.items()])  
 **if** number\_of\_winners > 1 **or** max\_score == 0:  
 overall\_winner = **"Friendship"  
 else**:  
 overall\_winner += **" with "** + str(max\_score) + **" Scores "** broadcast(overall\_winner, **"W"**)

Результати роботи:

Було створено вікторину «QUIZ IT» що відповіла усім поставленим вимогам в задачі та допомогла розібратися з основними засадами комп’ютерних мереж.

Посилання на репозиторій: <https://github.com/sevagul/CN-2020-COURSEWORK-QUIZ>