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)

2. 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.

3. Тому хто ввів ім’я необхідно під’єднатися до серверу якщо той працює. Дане повідомлення бачить лише один учасник (хто малює).
4. Коли гравець під’єднаний до серверу і вікторина не почалась, гравець маж мати можливість переглянути уже існуючих гравців, а також мати можливість розпочати тест.

5. 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.

6. 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.

7. 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\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