ספר פרויקט

שם התלמיד: נדב כרמונה

ת.ז זהות: 324187301

שם בית ספר: הדרים הוד השרון

שם המורה: אלי גולדשטיין

תוכן עניינים

[מבוא 3](#_Toc104458967)

[המטר והיעדים הפרויקט 3](#_Toc104458968)

[מהות המערכת 3](#_Toc104458969)

[אתגרים מרכזיים 3](#_Toc104458970)

[מגבלות ואילוצים 3](#_Toc104458971)

[מבנה של הפרויקט 4](#_Toc104458972)

[מערכת השרתים ותקשורת עם ה GUI 4](#_Toc104458973)

[מבנה נתונים שבשימוש 5](#_Toc104458974)

[תקשורת בין כל השרתים למשתמש 6](#_Toc104458975)

[התחברות למערכת 7](#_Toc104458976)

[מהלך משחק 7](#_Toc104458977)

[Client מתנתק מהמערכת 7](#_Toc104458978)

[Database\_server 8](#_Toc104458979)

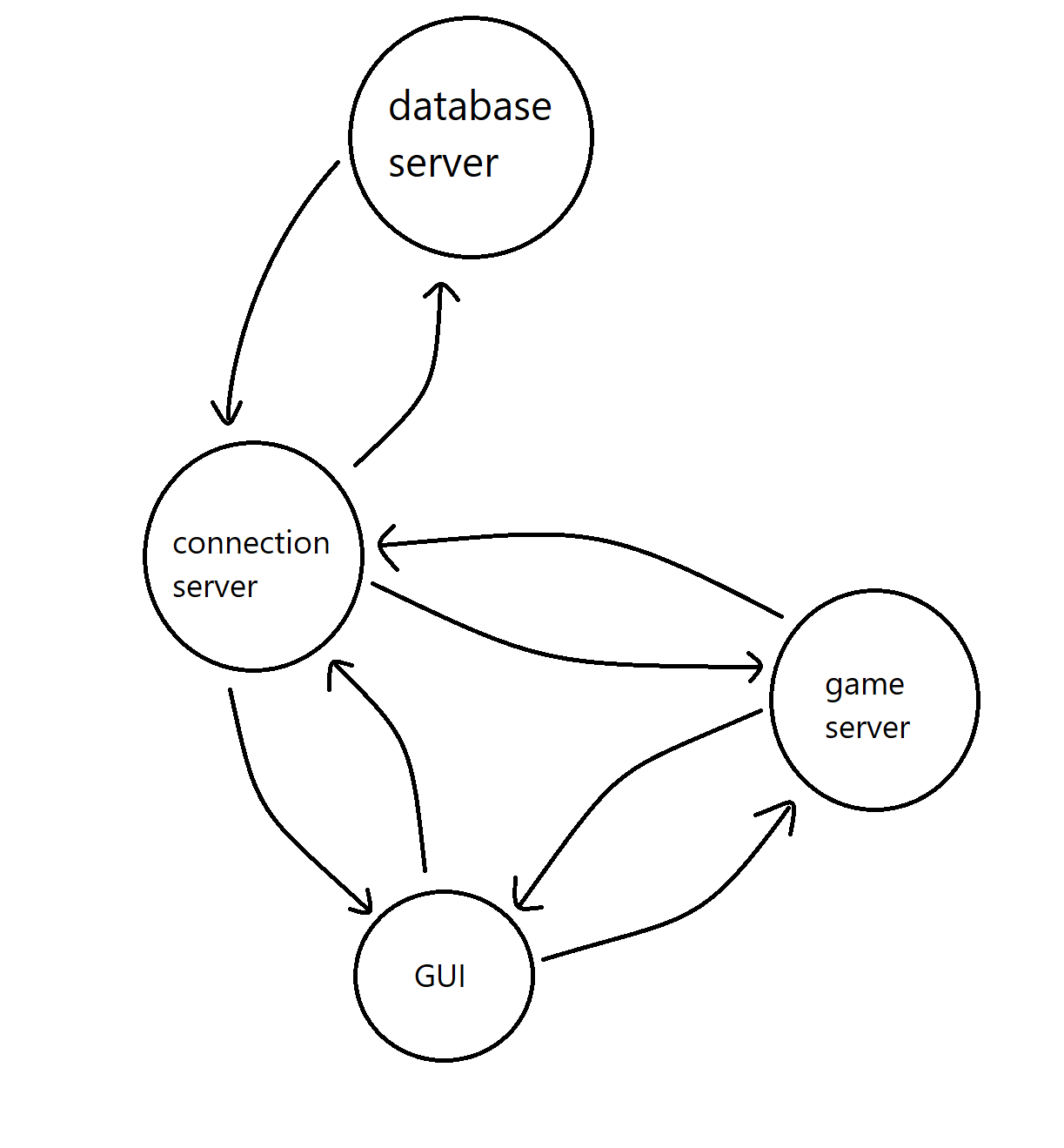
[Connection\_server 9](#_Toc104458980)

[Game\_server 10](#_Toc104458981)

[GUI 11](#_Toc104458982)

# מבוא

## מערכת השרתים ותקשורת עם השחקנים

database server הינו השרת הראשי, בנוסף הוא מתקשר עם ה-MySQL .

\*החיצים מראים את התקשורת בין הצדדים השונים

connection server הוא השרת המגשר בין כל הפניות לשרת הdatabase, הוא בודק את תקינות ההודעות ומוודא שתהיה סביבת עבודה נקייה לשרת הdatabase.

game server הוא שרת המשחק עצמו, הוא מריץ את המשחק למשתמשים.

GUI הוא השחקן עצמו, כל משתמש מריץ GUI ובעזרתו מתחבר למערכת.

## מטרת הפרויקט

ניהול משתמשים במערכת. שחקנים יכולים להתחבר או ליצור משתמשים במערכת ולהיכנס למשחק. המשחק עצמו הוא משחק יריות דו ממדי שבו כל אחד משחק ריבוע שיכול לזוז ולירות, המטרה היא לפגוע בכמה שיותר שחקנים אחרים וכך לצבור נקודות. השחקנים יכולו בנוסף להתחבר לאתר אינטרנט מיוחד שבו בעזרת התחברות למערכת יראו את ממוצע הניקוד שלהם למשחק ומיקומם ביחס לשחקנים האחרים. במקרה של שכחת סיסמא יוכל השחקן לבקש שהמערכת תשלח לו מייל עם סיסמא חדשה להתחברות.

## היעדים של הפרויקט

על הפרויקט לנהל משתמשים, עד כ-20, ללא תקיעות והאטה בזמן הריצה, ללא לאבד מידע על השחקנים. על שרת הconnection לסננן את כל ההודעות הלא תקינות מהמערכת. הgame server יוכל להריץ את המשחק לעד 5 שחקנים בו זמנית ללא האטות ותקיעות.

## מהות המערכת

ניהול משתמשים במערכת. שחקנים יכולים להתחבר או ליצור משתמשים במערכת ולהיכנס למשחק. המשחק עצמו הוא משחק יריות דו ממדי שבו כל אחד משחק ריבוע שיכול לזוז ולירות, כל השחקנים נמצאים במסך שחור ובו קירות אפורים המשמשים קקירות ומחסה, המטרה היא לפגוע בכמה שיותר שחקנים אחרים וכך לצבור נקודות. כששחקן נכנס למערת מוצג בפניו home screen אשר בו יכול השחקן לראות את ממוצע הניקוד שלהם למשחק, ממתי המשתמש שלהם קיים ומהו מיקומם ביחס לשחקנים האחרים. מה-home screen השחקנים יכולים להתחבר למשחק או להתנתק מהמערכת.

## הפעלת המערכת

ה- database server וה- connection server נפתחים ידנית במחשב אחד. ה-GUI נפתח על ידי השחקנים מכל מחשב שהם נמצאים. ה- game server נפתח ונסגר על ידי המערכת לפי הצורך.

## מבנה נתונים שבשימוש

השתמשתי במנה הנותנים של MySQL. בחרתי דווקא ב MySQL כיוון שהוא פשוט ונוח לשימוש והיה לי רק את הצורך הבסיסי לפעולות פשוטות של קריאה כתיבה ומחיקה ממבנה נתונים. לא התעמקתי בחיפוש אחר מבנה נתונים בגלל הצורך הבסיסי שלי בו.

## תקשורת בין כל השרתים למשתמש

### התחברות למערכת

הclient(GUI) שולח הודעה לconnection server שבוקר את תקינות ההודעה ומעביר אותה לdatabase server שבודק ברשימה בMySQL אם המשתמש יכול להתחבר. אם הוא מנסה להיכנס למשתמש שכבר קיים שרת הdatabase בודק אם הסיסמא שהclient שלח תואמת את זה של המשתמש והאם המשתמש לא מחובר כרגע, ואם הclient מנסה ליצור משתמש חדש שרת הdatabase בודק אם לא קיים משתמש אחר עם שם דומה. אז שרת הdatabase שולח הודעה לconnection server שמעביר את ההודעה לclient שניסה להתחבר והclient. אם הclient יכול להתחבר הוא פותח home screen אשר ממנו יכול לראות את ממוצע הניקוד שלו ומיקומו ביחד למשתמשים אחרים, בנוסף יכול הclient להיכנס למשחק או להתנתק. אם הclient נכנס למשחק הוא מתנתק מהconnection\_server ומתחבר לgame\_server.

### מהלך משחק

במשך המחשק הclient שולח את הinput שלו לgame\_server והgame\_server מחזיר את מה על הclient להדפיס על המסך שלו. בסוף כל משחק או כאשר הgame\_server מתרוקן לגמרי משחקנים הוא מעביר את מצב הניקוד של כל שחקן לconnection\_server שבודק את ההודעה ומעביר אותה הלאה לdatabase\_server שיעדכן את היסטוריית הניקוד וממוצע הניקוד של כל משתמש.

### Client מתנתק מהמערכת

Client שולח הודעת התנתקות לgame\_server, מתנתק ממנו ומתחבר לconnection\_server בחזרה לhome screen. משם הclient יכול להתחבר למשחק בחזרה או להתנתק סופית. אם מתנתק סופית שולח הודעה על כך לconnection\_server שמעביר את ההודעה על כך לdatabase\_server שמעדכן את ההתנתקות של המשתמש מהמערכת.

## מדריך למשתמש

# מבנה של הפרויקט

## Database\_server

השרת הראשי האחראי על תקשורת עם mysql וניהול ובקרה אחר כל השרתי משחק שקיימים.

## Connection\_server

שרת הכניסות משמש כגשר תקשורת בין שרתי המשחק לבין שרת הdatabase.

## Game\_server

השרת שמריץ את המשחק עצמו.

## GUI

הלקוח עצמו.

# קוד הפרויקט

## Database server code

import mysql.connector

import select

import logging

import socket

import pickle

from Constants import constant

logging.basicConfig(level=logging.INFO)

class ServerSide:

def \_\_init\_\_(self):

self.SERVER\_PORT = 6666

self.SERVER\_IP = str(socket.gethostname())

logging.info("Setting up server at-> " + self.SERVER\_IP)

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

self.server\_socket.bind((self.SERVER\_IP, self.SERVER\_PORT))

self.server\_socket.listen()

logging.info("Listening for clients...")

self.client\_sockets = []

self.number\_of\_client = 0

self.messages\_to\_send = []

self.game\_servers = {}

self.db = Database()

def run(self):

while True:

rlist, wlist, xlist = select.select([self.server\_socket] + self.client\_sockets, [], [])

players\_movement = []

for current\_socket in rlist:

if current\_socket is self.server\_socket: # new client joins

try:

self.newclient(current\_socket) # create new client

self.number\_of\_client += 1

except Exception as e:

connection, client\_address = current\_socket.accept()

connection.send("cant connect".encode())

self.client\_quit(current\_socket)

else: # what to do with client

players\_movement = self.client\_actions(current\_socket)

self.sending(players\_movement)

def client\_actions(self, current\_socket):

players\_movement = []

client\_mov = self.client\_mesege(current\_socket)

# print(client\_mov)

if client\_mov == constant.QUITING:

self.client\_quit(current\_socket)

elif client\_mov[0] == constant.GAMESERVER\_UPDATE: # game\_server

self.game\_servers[current\_socket][2] = client\_mov[1]

self.update\_all(client\_mov[2:])

players\_movement.append((current\_socket, "I love..."))

elif client\_mov[0] == constant.USER\_CONNECTING: # user connects

is\_ok = self.check\_connection(client\_mov[1], client\_mov[2], client\_mov[3], client\_mov[4])

print(f"is\_ok = {is\_ok}")

players\_movement.append((current\_socket, is\_ok))

elif client\_mov[0] == constant.NEW\_GAMESERVER: # new game server at wait

logging.info(client\_mov)

self.game\_servers[current\_socket] = [client\_mov[1], str(client\_mov[2])[2:-1], client\_mov[3]]

elif client\_mov[0] == constant.HOMESCREEN\_CONNECTS: # home screen

player = self.db.read(client\_mov[1].decode())

position = self.find\_position(player)

print(f"self.pick\_server returns {self.pick\_server()}")

players\_movement.append((current\_socket, (player, self.pick\_server(), position)))

elif client\_mov[0] == constant.HOMESCREEN\_QUITING: # home screen quit

player = self.db.read(client\_mov[1].decode())

self.db.add(player[0], player[1], player[5], player[2], player[3], player[4], False)

players\_movement.append((current\_socket, constant.QUITING))

elif client\_mov[0] == constant.SERVER\_QUIT: # game server quit

self.game\_servers.pop(current\_socket)

print(self.game\_servers)

self.client\_quit(current\_socket)

"""elif client\_mov[0] == constant.ENTER\_GAME:

players\_movement.append((current\_socket, self.pick\_server()))"""

return players\_movement

def find\_position(self, player):

all\_players = self.db.to\_string()

position = len(all\_players)

for other\_player in all\_players:

if player != other\_player:

if player[4] >= other\_player[4]:

position -= 1

return position

def check\_connection(self, name, password, date, client\_name):

if date != "" and client\_name != "":

if self.db.is\_exist(name):

return [False]

else:

try:

self.db.add(name, password, client\_name, date, "", 0, True)

print(f"self.pick\_server() returned {self.pick\_server()}")

if not self.pick\_server():

return [True, True]

else:

return [True, False]

except Exception as e:

print(e)

return [False]

else:

if not self.db.is\_exist(name):

return [False]

else:

player = self.db.read(name)

if password == player[1] and player[6] == 0:

self.db.add(player[0], player[1], player[5], player[2], player[3], player[4], True)

if not self.pick\_server():

return [True, True]

else:

return [True, False]

else:

return [False]

def pick\_server(self):

minimum = 10

selected\_server = None

if not self.game\_servers:

print(f"no selected server")

return False

for server in self.game\_servers:

# print(server)

if self.game\_servers[server][2] < minimum:

selected\_server = [self.game\_servers[server][0], self.game\_servers[server][1], server]

print(f"selected\_server = {selected\_server}")

minimum = self.game\_servers[server][2]

if selected\_server is None:

print(f"why did we get here?")

return False

else:

try:

data = pickle.dumps(constant.CHECK\_LIVE)

selected\_server[-1].send(str(len(str(len(data)))).zfill(4).encode() + str(len(data)).encode() + data)

return selected\_server[:-1]

except:

self.game\_servers.pop(selected\_server[-1])

print("servers: " + str(self.game\_servers))

self.client\_quit(selected\_server[-1])

selected\_server = self.pick\_server()

if not selected\_server:

print("helo")

return False

else:

return selected\_server[:-1]

def update\_all(self, client\_mov):

try:

for player in client\_mov:

self.update\_individual(player)

return True

except Exception as e:

print(e)

return False

def update\_individual(self, client\_player):

database\_player = self.db.read(client\_player[0])

name = database\_player[0]

password = database\_player[1]

date = database\_player[2]

client\_name = database\_player[5]

try:

history = database\_player[3]

if history == "":

self.db.add(name, password, client\_name, date, client\_player[1], client\_player[1], True)

else:

try:

former\_points = str(history).split(",")

points = 0

for former\_point in former\_points:

points += float(former\_point)

except:

points = float(history)

former\_points = [history]

self.db.add(name, password, client\_name, date,

str(history) + "," + str(client\_player[1]),

(client\_player[1] + points) / (len(former\_points) + 1), True)

except Exception as e:

print(e)

self.db.add(name, password, client\_name, date, client\_player[1], client\_player[1], True)

finally:

logging.info("update successful")

def newclient(self, current\_socket):

connection, client\_address = current\_socket.accept()

logging.info("New client joined!")

self.client\_sockets.append(connection)

def make\_messages(self, players\_movement):

for client\_data in players\_movement:

self.messages\_to\_send.append((client\_data[0], pickle.dumps(client\_data[1])))

def client\_mesege(self, current\_socket):

rsv = ""

try:

lenoflen = int(current\_socket.recv(4).decode())

lenght = int(current\_socket.recv(lenoflen).decode())

rsv = current\_socket.recv(lenght)

rsv = pickle.loads(rsv)

except:

logging.error("problem with resiving a message: " + str(current\_socket))

rsv = 99

finally:

return rsv

def sending(self, players\_movement):

self.make\_messages(players\_movement)

for message in self.messages\_to\_send:

current\_socket, data = message

try:

current\_socket.send(str(len(str(len(data)))).zfill(4).encode() + str(len(data)).encode() + data)

self.messages\_to\_send.remove(message)

except Exception as e:

logging.error("problem with sending a message: " + str(current\_socket))

self.client\_quit(current\_socket)

def client\_quit(self, current\_socket):

print(str(current\_socket) + " left")

try:

current\_socket.shutdown(socket.SHUT\_RDWR)

current\_socket.close()

self.client\_sockets.remove(current\_socket)

finally:

return

class Database:

def \_\_init\_\_(self):

self.mydb = ""

self.mycursor = ""

self.open\_sequence()

def open\_sequence(self):

try:

self.mydb = mysql.connector.connect(

host="localhost",

user="root",

password="root",

database="shootydb"

)

self.mycursor = self.mydb.cursor()

is\_exist = False

self.mycursor.execute("SHOW TABLES")

for x in self.mycursor:

# print(str(x)[2:-3])

if str(x)[2:-3] == "players":

is\_exist = True

if not is\_exist:

self.create\_db()

return

except:

if self.mydb == "":

self.mydb = mysql.connector.connect(

host="localhost",

user="root",

password="root"

)

if self.mycursor == "":

self.mycursor = self.mydb.cursor()

self.create\_db()

return

def create\_db(self):

self.mycursor.execute("CREATE DATABASE shootydb")

self.mycursor.execute("CREATE TABLE shootydb.players (PlayerName VARCHAR(255) primary key not null,"

" PlayerPassword VARCHAR(255) not null, CreationTime VARCHAR(255) not null,"

" GameHistory VARCHAR(255) not null, PersonalRecord double not null,"

" ClientName VARCHAR(255) not null, IsConnect boolean not null)")

def is\_exist(self, name):

sql = "SELECT \* FROM shootydb.players WHERE PlayerName = '" + name + "'"

self.mycursor.execute(sql)

place = self.mycursor.fetchall()

if len(place) == 0:

return False

else:

return True

def add(self, PlayerName, PlayerPassword, ClientName, CreationTime, GameHistory, PersonalRecord,

is\_connect): # add a value new, if already exist change old one to match the new input

try:

if self.is\_exist(PlayerName):

self.delete(PlayerName)

sql = "INSERT INTO shootydb.players " \

"(PlayerName, PlayerPassword, ClientName, CreationTime, GameHistory, PersonalRecord, IsConnect)" \

" VALUES (%s, %s, %s, %s, %s, %s, %s)"

val = (PlayerName, PlayerPassword, ClientName, CreationTime, GameHistory, PersonalRecord, is\_connect)

print(val)

self.mycursor.execute(sql, val)

self.mydb.commit()

return True # if add successful return True

except Exception as e:

print(e)

return False # if doesn't work return False

def delete(self, name):

try:

myresult = self.read(name)

sql = "DELETE FROM shootydb.players WHERE PlayerName = '" + name + "'"

self.mycursor.execute(sql)

self.mydb.commit()

return myresult # if delete successfully return what was deleted

except Exception as e:

print(e)

return False # if doesn't work return False

def read(self, name):

self.mycursor.execute("SELECT \* FROM shootydb.players where PlayerName = '" + name + "'")

myresult = self.mycursor.fetchall()

return myresult[0]

def to\_string(self):

self.mycursor.execute("SELECT \* FROM shootydb.players")

myresult = self.mycursor.fetchall()

return myresult

def update\_all(self, what, update):

sql = "UPDATE help SET " + what + " = '" + update + "'"

self.mycursor.execute(sql)

self.mydb.commit()

def get\_names(self):

names = []

self.mycursor.execute("SELECT \* FROM shootydb.players")

myresult = self.mycursor.fetchall()

for x in myresult:

names.append(x)

return myresult

def get\_winner(self):

top = ["", "", -1]

for player in self.get\_names():

if player[2] > top[2]:

top = player

return top

def main():

ds = ServerSide()

ds.run()

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

main()