GenerCube

(משחק יריות דו ממדי רב משתמשים)

שם התלמיד: נדב כרמונה

ת.ז זהות: 324187301

שם בית ספר: הדרים הוד השרון

שם המורה: אלי גולדשטיין



תוכן עניינים

[מבוא 4](#_Toc105611325)

[מערכת השרתים ותקשורת עם השחקנים(GUI) 5](#_Toc105611326)

[מטרת הפרויקט 6](#_Toc105611327)

[היעדים של הפרויקט 6](#_Toc105611328)

[תכנון מול ביצוע 6](#_Toc105611329)

[מגבלות ואילוצים 6](#_Toc105611330)

[הפעלת המערכת 7](#_Toc105611331)

[טכנולוגיות ששומשו בפרויקט 7](#_Toc105611332)

[בסיס נתונים שבשימוש 8](#_Toc105611333)

[תקשורת בין כל השרתים למשתמש 9](#_Toc105611334)

[התחברות למערכת 9](#_Toc105611335)

[מהלך משחק 10](#_Toc105611336)

[Client מתנתק מהמערכת 11](#_Toc105611337)

[מדריך למשתמש 12](#_Toc105611338)

[פתיחת וכניסה למערכת 12](#_Toc105611339)

[המשחק עצמו 16](#_Toc105611340)

[מבנה של הפרויקט 17](#_Toc105611341)

[Database\_server 17](#_Toc105611342)

[Connection\_server 18](#_Toc105611343)

[Game\_server 19](#_Toc105611344)

[GUI 20](#_Toc105611345)

[קוד הפרויקט 22](#_Toc105611346)

[Database\_server 22](#_Toc105611347)

[connection\_server 35](#_Toc105611348)

[game\_server 44](#_Toc105611349)

[GUI 64](#_Toc105611350)

[רפלקציה 94](#_Toc105611351)

[ביבליוגרפיה 95](#_Toc105611352)

[נספחים 96](#_Toc105611353)

# מבוא

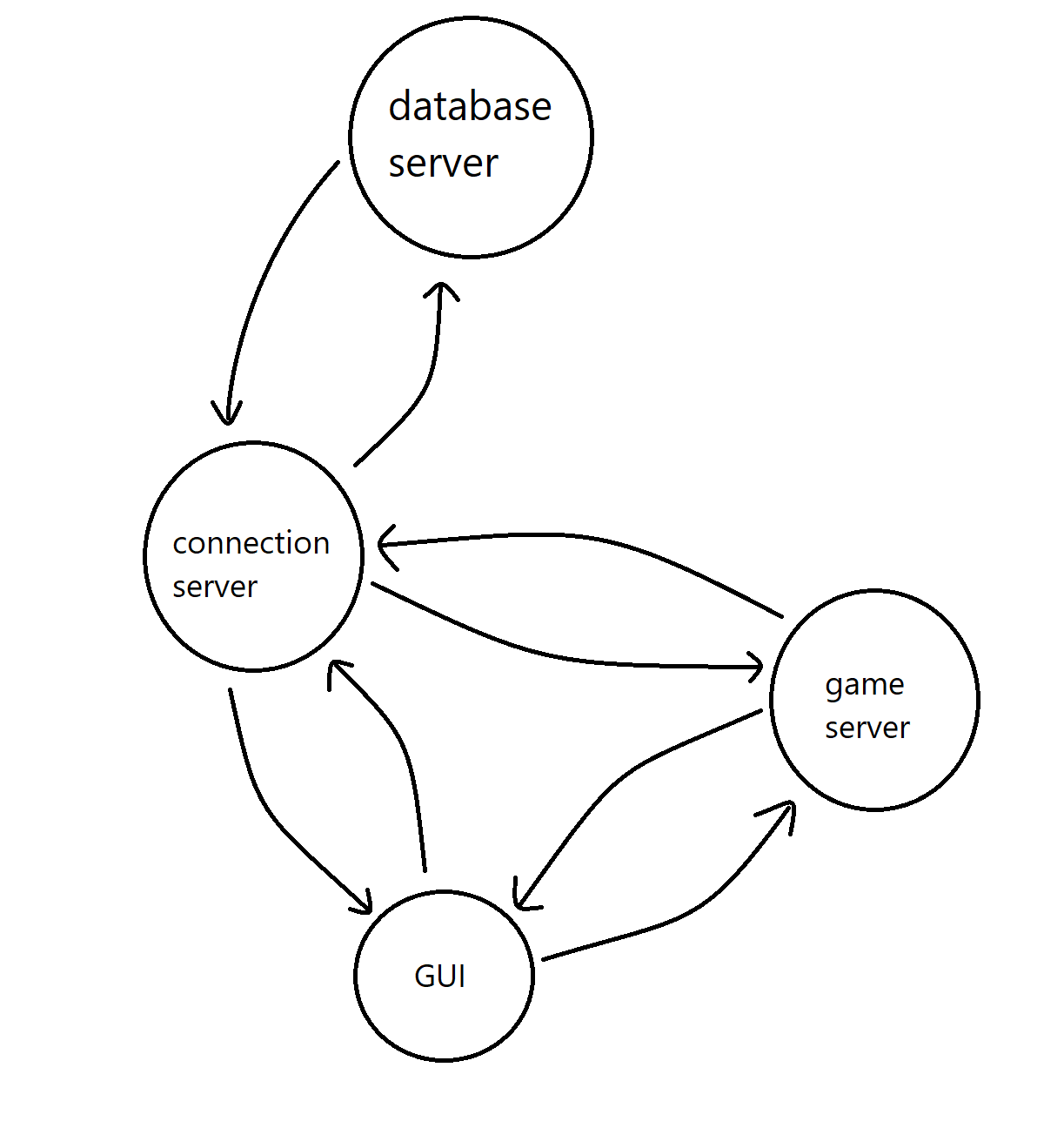
ספר פרויקט זה מכיל הסבר כללי על הפרויקט "GenerCube", השימוש בו והמה הוא מכיל. הספר מכיל תיאור של הארכיטקטורה של הפרויקט ומדריך למשתמש ולמפתח. הפרויקט הגיע מרקע של אהבה למשחקי יריות דו ממדיים רבי משתמשי.

מסקירת המצב כיום של משחקי היריות הדו ממדיים מעלה שישנם משחקים אינטרנטיים כאלו, אך אין בהם את הפשרות להירשם עם משתמש ולהשוות בין מצבך לשאר השחקנים.

הסיבה המרכזית שבחרתי לעשות את הפרויקט הזה היא משום אהבתי למשחקי היריות הדו ממדיים מרובי המשתמשים, תמיד אהבתי לשחק במשחקי יריות עם חברים, בשלל סוגים ורמות של משחקים. ורציתי ללמוד איך להכין משחק יריות דומה למה שאוהב ולתת לי ולחברים לשחק ביחד והשוות בין הניקוד שלנו.

האתגרים המרכזיים בפרויקט היו הבנת תחומים השימוש בMySQL , מציאת דרך לתקשר בין שרתים רבים, להכיל הרבה שחקנים במקביל, ולחשב ולפתוח שרתי משחק לפי הצורך ולעקוב אחריהם.

## מערכת השרתים ותקשורת עם השחקנים(GUI)

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. כל הסיסמאות יהיו מוצפנות בעזרת hashlib.md5 במנה הנתונים. כל GUI שמתחבר לgame server יצטרך לשלוח את הזמן בדיוק של דקות כדי ששרת המשחק יאשר את ההתחברות שלו.

## 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.

## תכנון מול ביצוע

בסוף, לא היה באפשרותי להכניס את כל מה שרציתי לפרויקט, כמו שכתבתי במטרות הפרויקט. לכן, אין אתר אינטרנט שבו אפשר לראות את בו יראו את ממוצע הניקוד שלהם למשחק ומיקומם ביחס לשחקנים האחרים בעזרת התחברות למערכת, אך במקום האתר ישנו מסך בית שנפתח עם התחברות למערכת שבו אפשר לראות את הדברים הללו. בנוסף אין את האופציה שבמקרה של שכחת סיסמא יוכל השחקן לבקש שהמערכת תשלח לו מייל עם סיסמא חדשה להתחברות. ואין את דרישת האבטחה של כניסת GUI ל game server שבו כל GUI שמתחבר לgame server צריך לשלוח את הזמן בדיוק של דקות כדי ששרת המשחק יאשר את ההתחברות שלו.

## מגבלות ואילוצים

GUI יכול לרוץ רק על windows. בסיס הנתונים ב MySQL חייב להיות באותו המחשב עם שרת הdatabase . ב MySQL חייב להיות Local instance MySQL80 connection כדי ששרת הdatabase יוכל להתחבר לבסיס הנתונים.

## 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.

## טכנולוגיות ששומשו בפרויקט

* Socket + select – שומש בשביל תקשורת בין ה GUI לgame server , ל connection server ולשרת ה database.
* Hashlib – שומש בשביל הצפנת הסיסמאות של השחקנים בכתיבתם במבנה הנתונים.
* Pickle – שומש בשביל העברת המידע בין ה GUI, הgame server , ה connection server וה database server.
* Tkinter + page – שומשו בשביל ליצור את מסכי הכניסה למערכת של ה GUI.
* שפת התכנות ששומשה: python.

## בסיס 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 מבסיס נתונים. לא התעמקתי בחיפוש אחר בסיס נתונים בגלל הצורך הבסיסי שלי בו.

מבנה הנתונים בנוי מכמה עמודות:

PlayerName- string המשמש כשם השחקן.

PlayerPassword- string המשמש בסיסמא של השחקן, מוצפנת בהצפנת md5 שעוברת פעולת digest() אשר הופך את הסיסמה המוצפנת בצורה בינארית.

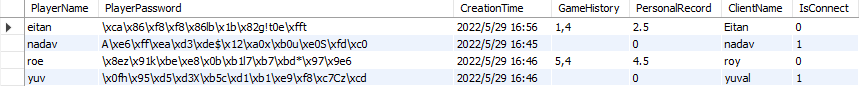
CreationTime- string שמתאר את זמן יצירת השחקן, מוצב במסך הבית של המשתמש.

GameHistory- string המתאר את היסטורית הניקוד של השחקן, משמש למען חישוב ממוצע הנקודות בPersonalRecord .

PersonalRecord- double המתאר את ממוצע הנקודות של השחקן, מוצג במסך הבית של המשתמש. בנוסף משמש לחישוב מיקום השחקן מבחינת טבלת הניקוד.

ClientName- string שמתאר את שם של הלקוח שיצר את השחקן, מוצג במסך הבית.

IsConnect- boolean שמתאר האם השחקן מחובר למערכת או לא, 0 אומר שהשחקן אינו מחובר ו-1 אומר שכן מחובר.



## תקשורת בין כל השרתים למשתמש

### התחברות למערכת

ה (client(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 וה client. 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 מה connection\_server ומתחבר ל game\_server.

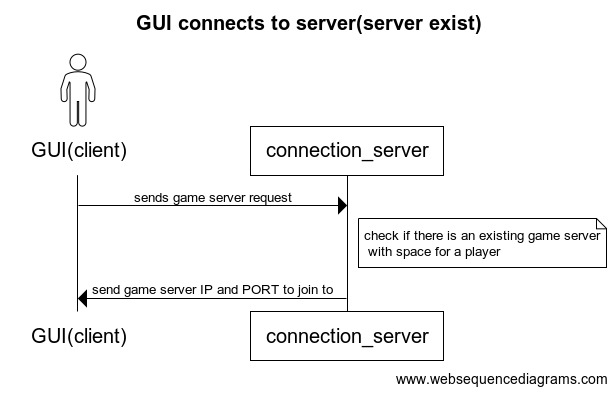
### מהלך משחק

במשך המחשק ה client 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 ל database\_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.

### Client מתחבר ל game\_server(קיים game server)



GUI שולח בקשת התחברות לשרת משחק ל connection\_server, הוא בודק שאכן קיים game\_server ושולח את ה IP של שרת המשחק ואת ה PORT שלו.

### Client מתחבר ל game\_server(לא קיים game server)

GUI שולח בקשת התחברות לשרת משחק ל connection\_server, הוא בודק האם קיים game\_server אשר לא קיים. אז ה connection\_server שולח לclient בקשה לפתוח game\_server חדש. לאחר פתיחת game\_server חדש והירשמותו ב connection\_server ה client מנסה להתחבר שוב.

## מדריך למשתמש

### פתיחת וכניסה למערכת

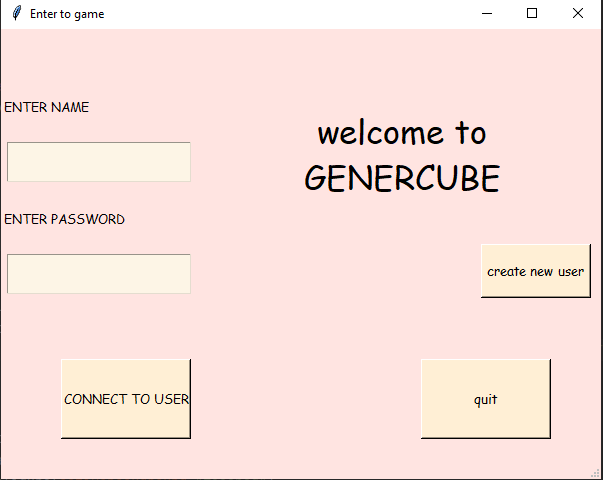
 -מתקינים את הקובץ GUI.exe ופותחים אותו.

-נפתח חלון להכנת IP: תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

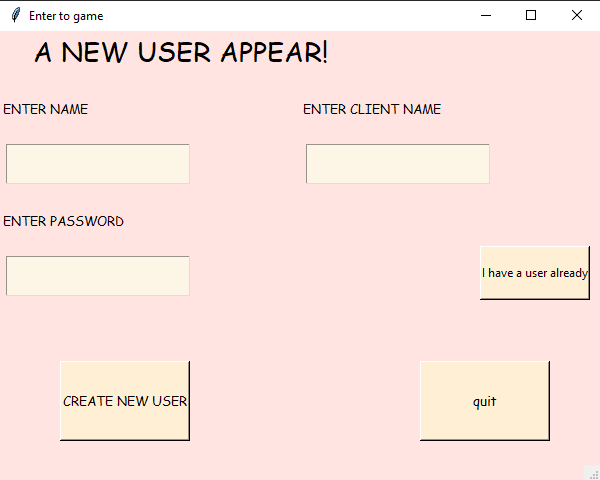
כותבים שם את הIP של הconnection server ואז לוחצים על כפתור ה"ENTER" עם העכבר.

<-הconnection server כותב את הIP שלו כשנפתח.

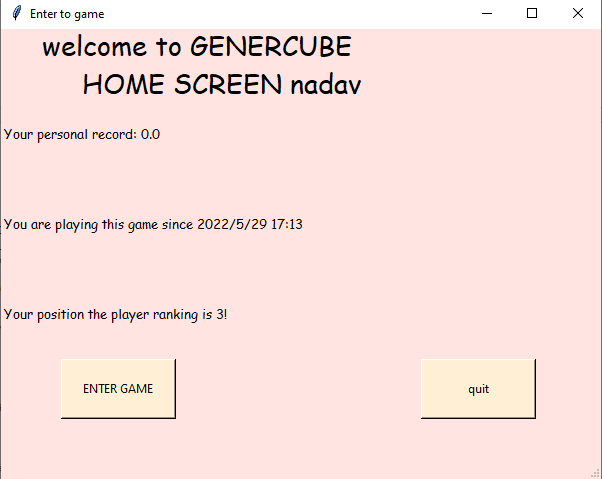
-נפתח חלון התחברות :

מכניסים שם משתמש וסיסמא ולוחצים על הכפתור "ENTER GAME". בלחיצה על הכפתור "quit" המסך ייסגר וריצת הקוד תסתיים.

במקרה ורוצים ליצור משתמש חדש לוחצים על הכפתור "create new user".

-כשלוחצים על הכפתור "create new user" נפתח החלון הבא:

מכניסים שם משתמש(שם שלא קיים כבר במערכת), סיסמא ואת שם השחקן ולוחצים על הכפתור "CREATE NEW USER". בלחיצה על הכפתור "quit" המסך ייסגר וריצת הקוד תסתיים.

-אם מכניסים את כל הפרטים נכון והמערכת מאשר את החיבור נפתח מסך בית:

מסך הבית מראה את שם השחקן שהוכנס, זמן יצירת השחקן, ממוצע הנקודות שלו למשחק ואת מיקומו בין שאר השחקנים מבחינת ממוצע הנקודות.

כדי להיכנס למשחק צריך ללחוץ עם העכבר על הכפתור "ENTER GAME", בלחיצה על הכפתור "quit" המסך ייסגר וריצת הקוד תסתיים.

### המשחק עצמו

כשנכנסים למשחק ממסך הבית נפתח המסך של המשחק:

הריבוע הכחולה היא הדמות של השחקן, שאר השחקנים הם ריבועים בצבע אדום.

כאשר במשחק הכפתורים שמשתמשים בהם הם:

W- להזיז את הדמות למעלה.

A- להזיז את הדמות שמאלה.

S- להזיז את הדמות למטה.

D- להזיז את הדמות ימינה.

עם העכבר ניתן לכוון את הדמות, ועם לחיצה אחת הלחצן השמאלי של העכבר ניתן לירות עם הדמות.

Esc- יציאה מהמשחק בחזרה למסך הבית.



# מבנה של הפרויקט

## Database\_server

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

השרת מכיל שתי מחלקות:

ה server side אחראי על כל התקשורת עם הconnection server והעברת המידע.

מחלקת ה Database אחראית על מבנה הנתונים של MySQL.

השרת קורא בקשות של clients, מעביר מידע מבסיס הנתונים ומעדכן אותו לפי הצורך. אם לא קיים בסיס נתונים או שאינו תקין הוא יוצא בסיס נתונים חדש. בעל פעולות קריאה כתיבה ומחיקה ממבנה הנתונים של ה MySQL.

## Connection\_server

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

מכיל שני מחלקות:

server side אחראי על בדיקת ההודעות והתקשורת עם הGUI והgame server .

client side אחראי עם התקשורת עם ה database server.

מקבל הודעות מGUI ומ game server, בודק אם ההודעות תקינות ואם כן מעביר אותן לdatabase server לפי הצורך. ואת ההודעות של הdatabase server מחזיר ל GUI או game server המתאימים.

השרת עוקב אחרי שרתי המשחק (game server) ויכול לשלוח שחקנים לפי השרת הכי פנוי, ואם אין שרת משחק פנוי או שאין שרת משחק בכלל שולח הודעה לפתוח אחד חדש.

## Game\_server

השרת שמריץ את המשחק עצמו. מתחברים אליו שחקנים , עד 5. ישנה לולאה שחוזרת על עצמה של קבלת input מהשחקנים, חישוב מה שקורה במשחק עצמו, ושליחת המסך להדפסה לכל שחקן. במקרה של התנתקות גם אם אינה רצויה.

השרת בנוי מארבע מחלקות:

ServerSide הוא השרת עצמו, מנהל את כל הלקוחות במשחק, מקבל מהם את הקלטים שלהם ומחזיר להם מה להדפיס על המסך.

ClientSide אחראי על התקשורת עם הconnection server .

Game הינה המחלקה אשר מחשבת את כל מה שקורה במשחק (כל הקירות, היריות, השחקנים), היא מקבל את הקלטים של השחקנים ומחשבת ומזיזה את הדמויות.

המחלקה Orientation משמשת אך ורק לשליחת המידע להדפסת המשחק ללקוחות.

השרת משמש גם כ client לשרת הconnection, אשר אליו הוא שולח עדכון מצב הניקוד של כל שחקן בסוף משחק או כאשר כל השחקנים מתנתקים מהמשחק והשרת נסגר. שליחת המידע להדפסה ל GUI נעשית בעזרת ביצוע פעולת pickle שעושה serialize לאובייקט Orientation אשר נמצא בשניהם. אובייקט ה Orientation מכיל את כל התכונות של צורה שצריך להדפיס.

## GUI

השחקן עצמו, כקובץ executable אשר הלקוח מריץ, דרכו הוא מתחבר לשרת הconnection להתחברות וממנו לשרת המשחק. הקוד של ה GUI מכיל חלונות tkinter אשר מוצגים בכניסה למערכת וחלון pygame למסך של המשחק עצמו.

ה GUI מורכב משמונה מחלקות:

TopLevelMother משמש כמחלקת האם של TopLevel1 ו- TopLevel2.

TopLevel1 אחראי על מסך ה tkinter שנפתח כשמתחבר למשתמש, מתקשר כלקוח עם הconnection server .

TopLevel2 אחראי על מסך ה tkinter שנפתח כאשר יוצר משתמש חדש מתקשר כלקוח עם הconnection server .

IpCatcher אחראי על מסך ה tkinter שנפתח בתחילת ריצת הקוד כשמתבקש להכניס את ה IP של הconnection server .

HomeScreen אחראי על מסך הבית (tkinter) שנפתח כשמתחבר למשתמש ולמערכת בהצלחה מתקשר כלקוח עם הconnection server .

ClientSide משמש כלקוח שמתקשר עם ה game server, שולח לו את הקלטים של השחקן, ומדפיס את לוח המשחק בעזרת pygame.

Orientation משמש אך ורק לקבלת המידע להדפסת המשחק ללקוחות מgame server .

screen\_manager הוא האחראי על כל המסכים שיכולים להיפתח לשחקן, הוא גם דואג שכאשר השחקן מתנתק מהמשחק הוא חוזר למסך הבית.

קבלת המידע להדפסה מה game server נעשית בעזרת ביצוע פעולת pickle שעושה deserialize לאובייקט Orientation אשר נמצא בשניהם. אובייקט ה Orientation מכיל את כל התכונות של צורה שצריך להדפיס, כך הוא עובר על כל האובייקטים ומדפיס את כל הצורות על המסך.

# קוד הפרויקט

## Database\_server

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.update\_all(client\_mov[1])

players\_movement.append((current\_socket, "Every ting ok..."))

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.HOMESCREEN\_CONNECTS: # home screen

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

position = self.find\_position(player)

players\_movement.append((current\_socket, (player, 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))

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)

return True

except Exception as e:

print(e)

return False

else:

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

return False

else:

player = self.db.read(name)

print(password, player[1])

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

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

return True

else:

return False

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

print(client\_mov)

try:

for player in client\_mov:

print(player)

self.update\_individual(player)

return True

except Exception as e:

logging.error(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]

print(client\_player, database\_player)

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:

logging.error(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()

finally:

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

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:

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()

## connection\_server

import select

import logging

import socket

import pickle

from Constants import constant

import hashlib

logging.basicConfig(level=logging.DEBUG)

class ServerSide:

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

self.SERVER\_PORT = 7777

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.client\_side = ClientSide()

self.game\_servers = {}

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.append(self.client\_requests(current\_socket))

self.sending(players\_movement)

del players\_movement

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

players\_movement = []

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

if client\_mov == constant.QUITING:

self.client\_quit(current\_socket)

return False

else:

if client\_mov[0] == constant.USER\_CONNECTING: # user connect / new user

player = self.chek\_user\_request(client\_mov[1], client\_mov[2], client\_mov[3], client\_mov[4])

if not player:

players\_movement = (current\_socket, False)

else:

self.client\_side.send(pickle.dumps(player))

is\_ok = self.client\_side.read()

players\_movement = (current\_socket, is\_ok)

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

self.client\_side.send(pickle.dumps(client\_mov))

is\_ok = self.client\_side.read()

players\_movement = (current\_socket, is\_ok)

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

self.client\_side.send(pickle.dumps(client\_mov))

self.client\_side.read()

self.client\_quit(current\_socket)

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

logging.info(client\_mov[1:])

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

players\_movement = (current\_socket, constant.CHECK\_LIVE)

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

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

self.client\_side.send(pickle.dumps((constant.GAMESERVER\_UPDATE, client\_mov[2:])))

is\_ok = self.client\_side.read()

players\_movement = (current\_socket, is\_ok)

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

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

self.client\_quit(current\_socket)

elif client\_mov[0] == constant.SERVER\_REQUEST:

server = self.pick\_server()

players\_movement = (current\_socket, server)

return players\_movement

def chek\_user\_request(self, name, password, date, client\_name):

if name != "" and password != "" and len(name) < 10 and client\_name != "":

print(str(hashlib.md5(password.encode())))

return [constant.USER\_CONNECTING, name, str(hashlib.md5(password.encode()).digest())[2:-1], date, client\_name]

else:

logging.debug("not sending")

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(self.game\_servers[server])

try:

if self.game\_servers[server][2] < minimum and self.game\_servers[server][2] != 0:

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]])

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(f"no selected server x2")

return False

else:

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(f"no selected server x2")

return False

else:

return selected\_server[:-1]

def newclient(self, current\_socket):

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

logging.info(f"New client joined: {client\_address}")

self.client\_sockets.append(connection)

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

for client\_data in players\_movement:

if client\_data:

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)

# print(rsv)

except:

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

rsv = constant.QUITING

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):

logging.debug(f"{current\_socket} left")

try:

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

current\_socket.close()

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

finally:

return

class ClientSide:

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

self.my\_socket = socket.socket()

ip = socket.gethostname()

port = 6666

self.my\_socket.connect((ip, port))

logging.debug("client side connected...")

def send(self, data):

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

def read(self):

try:

lenoflen = int(self.my\_socket.recv(4).decode())

lenght = int(self.my\_socket.recv(lenoflen).decode())

data = self.my\_socket.recv(lenght)

data = pickle.loads(data)

return data

except Exception as e:

logging.error(e)

def main():

ds = ServerSide()

ds.run()

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

main()

## game\_server

import time

import select

import logging

import socket

import pickle

import sys

from Constants import constant

import hashlib

import pygame

import random

import math

logging.basicConfig(level=logging.DEBUG)

class ClientSide:

def \_\_init\_\_(self, server\_port, server\_ip, max\_clients, database\_ip):

self.my\_socket = socket.socket()

ip = database\_ip

port = 7777

logging.debug("connected ot " + str(ip) + "," + str(port))

self.my\_socket.connect((ip, port))

self.send(pickle.dumps((constant.NEW\_GAMESERVER, server\_port, server\_ip.encode(), max\_clients)))

self.read()

def send(self, data):

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

def read(self):

try:

lenoflen = int(self.my\_socket.recv(4).decode())

lenght = int(self.my\_socket.recv(lenoflen).decode())

data = self.my\_socket.recv(lenght)

data = pickle.loads(data)

return data

except Exception as e:

logging.error(e)

def make\_message(self, players, space):

stats = [constant.GAMESERVER\_UPDATE, space]

print("sending")

for player in players:

stats.append((player.name, player.score))

return stats

def run(self, players, space):

stats = self.make\_message(players, space)

self.send(pickle.dumps(stats))

self.read()

def close(self, players, space):

stats = self.make\_message(players, space)

self.send(pickle.dumps(stats))

self.read()

self.send(pickle.dumps((constant.SERVER\_QUIT, constant.QUITING)))

class ServerSide:

def \_\_init\_\_(self, database\_ip):

self.game = Game()

self.SERVER\_PORT = 55555

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

logging.debug("Setting up server...")

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()

self.max\_clients = 5

self.client\_side = ClientSide(self.SERVER\_PORT, self.SERVER\_IP, self.max\_clients, database\_ip)

self.client\_sockets = []

self.messages\_to\_send = []

self.number\_of\_client = 0

self.players\_conection = {}

self.active = False

def game\_maker(self):

while True:

logging.info("new game start")

self.gamerun()

self.game.restart()

def close\_server(self):

self.client\_side.close(self.game.players, self.max\_clients - self.number\_of\_client)

sys.exit()

def print\_client\_sockets(self, client\_sockets):

for i in range(len(client\_sockets)):

logging.debug(""" """ + str(client\_sockets[i]))

def newclient(self, current\_socket, client\_sockets):

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

logging.info("New client joined:")

client\_sockets.append(connection)

self.print\_client\_sockets(client\_sockets)

player = Game.Player()

self.game.players.add(player)

self.players\_conection[player] = connection

self.players\_conection[connection] = player

self.number\_of\_client += 1

self.active = True

player.rect.center = self.game.teleport(self.game.all\_sprites)

self.game.leaderboard.change\_places(self.game.players)

def change\_client\_name(self, current\_socket, name):

player = self.players\_conection[current\_socket]

player.change\_name(name)

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)

# print(rsv)

except:

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

rsv = constant.QUITING

finally:

return rsv

def get\_from\_clients(self, rlist):

players\_movement = []

for current\_socket in rlist:

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

if self.max\_clients - self.number\_of\_client > 0:

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

logging.info(" players left to join: " + str(self.max\_clients - self.number\_of\_client))

else:

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

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

self.player\_quit(current\_socket)

else: # what to do with client

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

if move == constant.QUITING:

self.player\_quit(current\_socket)

elif move[0] == constant.USER\_ACTION: # user input

self.change\_client\_name(current\_socket, move[1])

players\_movement.append((move[2], current\_socket))

self.players\_conection[move[2]] = current\_socket

return players\_movement

def make\_messeges(self, rlist, players, enemies, all\_sprites, LeaderBoard):

for current\_socket in rlist:

bit\_mesege = []

for player in players:

bit\_mesege.append(player.Serialize())

for bullet in enemies:

bit\_mesege.append(bullet.Serialize())

for wall in all\_sprites:

bit\_mesege.append(wall.Serialize())

for i in range(len(LeaderBoard.txts)):

bit\_mesege.append(LeaderBoard.Serialize(i))

self.messages\_to\_send.append((current\_socket[1], pickle.dumps(bit\_mesege)))

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

logging.info(str(current\_socket) + " left")

self.game.players.remove(self.players\_conection[current\_socket])

self.players\_conection.pop(current\_socket)

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

current\_socket.close()

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

self.number\_of\_client -= 1

if self.active and self.number\_of\_client == 0:

self.close\_server()

def sending(self, mov\_makers):

self.make\_messeges(mov\_makers, self.game.players, self.game.enemies, self.game.all\_sprites,

self.game.leaderboard)

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))

def gamerun(self):

running = True

while running:

rlist, wlist, xlist = select.select([self.server\_socket] + self.client\_sockets, [], [], 0.1)

player\_movement = self.get\_from\_clients(rlist)

if self.game.game\_time > 0:

for movement in player\_movement:

try:

current\_sucket = movement[1]

the\_move = movement[0]

current\_player = self.players\_conection[current\_sucket]

look = the\_move[2]

fire = the\_move[1]

move = the\_move[0]

self.game.Player.set\_directangleion(current\_player, look)

val = self.game.Player.mov(current\_player, self.game.all\_sprites, move, fire)

if val != "successful":

self.game.enemies.add(val)

except Exception as e:

print(str(e) + " <---error")

self.game.colisions()

pygame.time.delay(10)

self.game.game\_time -= 1

if self.game.game\_time == 0:

self.client\_side.run(self.game.players, self.max\_clients - self.number\_of\_client)

if self.game.game\_time <= 0:

self.game.leaderboard.winner(self.game.players)

if self.game.game\_time == -300:

running = False

self.sending(player\_movement)

class Game:

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

self.game\_time = 2000

self.SCREEN\_WIDTH = 1100

self.SCREEN\_HEIGHT = 600

self.players = pygame.sprite.Group()

self.enemies = pygame.sprite.Group()

self.all\_sprites = pygame.sprite.Group()

for i in range(random.randint(10, 15)):

wall = Game.Walls()

self.all\_sprites.add(wall)

self.leaderboard = Game.LeaderBoard()

self.leaderboard.change\_places(self.players)

bord, line = self.leaderboard.set\_place()

self.all\_sprites.add(bord)

self.all\_sprites.add(line)

class Demo(pygame.sprite.Sprite):

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

super(Game.Demo, self).\_\_init\_\_()

self.rectangle = pygame.Surface((24, 24), pygame.SRCALPHA)

self.rect = self.rectangle.get\_rect()

def restart(self):

self.game\_time = 2000

self.all\_sprites.empty()

self.enemies.empty()

for i in range(random.randint(10, 15)):

wall = Game.Walls()

self.all\_sprites.add(wall)

for player in self.players:

player.score = 0

player.rect.center = self.teleport(self.all\_sprites)

self.leaderboard = Game.LeaderBoard()

self.leaderboard.change\_places(self.players)

bord, line = self.leaderboard.set\_place()

self.all\_sprites.add(bord)

self.all\_sprites.add(line)

class Enemy(pygame.sprite.Sprite):

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

super(Game.Enemy, self).\_\_init\_\_()

self.rectangle = pygame.Surface((10, 10), pygame.SRCALPHA)

self.rectangle.fill(pygame.Color('white'))

self.rect = self.rectangle.get\_rect()

self.angle = math.radians(player.angle - 90)

self.rect.center = (int(player.rect.centerx + 10 \* math.cos(self.angle)),

int(player.rect.centery - 10 \* math.sin(self.angle)))

# ^^^in case I make bullet not move:

# bullet will hit the shooter, because I usually move the bullet before checking for coalition!

self.rot\_image = self.rectangle

self.rot\_image\_rect = self.rot\_image.get\_rect(center=self.rect.center)

self.owner = player

self.bullet\_speed = 15

self.orientation = Orientation(self.rect.x, self.rect.y, self.rect.width, self.rect.height, self.angle,

'white', "")

def mov(self):

self.rect.move\_ip(math.cos(self.angle) \* self.bullet\_speed, math.sin(self.angle) \* -self.bullet\_speed)

self.rot\_image\_rect = self.rot\_image.get\_rect(center=self.rect.center)

self.orientation.x = self.rect.x

self.orientation.y = self.rect.y

def Serialize(self):

return pickle.dumps(self.orientation)

def teleport(self, walls):

demo = Game.Demo()

t = False

while not t:

demo.rect.center = (random.randint(1, 801), random.randint(1, 601))

if not pygame.sprite.spritecollideany(demo, walls):

t = True

new\_center = demo.rect.center

demo.kill()

return new\_center

class Player(pygame.sprite.Sprite):

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

super(Game.Player, self).\_\_init\_\_()

self.mouse = pygame.mouse

self.rectangle = pygame.Surface((24, 24), pygame.SRCALPHA)

self.rectangle.fill(pygame.Color('red'))

self.rectangle.fill(pygame.Color('white'), (5, 6, 14, 5))

self.rect = self.rectangle.get\_rect()

self.rot\_image = self.rectangle

self.rot\_image\_rect = self.rot\_image.get\_rect(center=self.rect.center)

self.angle = 0

self.speed = 8

self.firing\_speed = 6

self.fire\_wait = 120

self.t\_wait = 120

self.name = "none"

self.score = 0

self.orientation = Orientation(self.rect.x, self.rect.y, self.rect.width, self.rect.height, self.angle,

'red', "")

def Serialize(self):

return pickle.dumps(self.orientation)

def fire(self):

new\_enemy = Game.Enemy(self)

return new\_enemy

def set\_directangleion(self, look):

mx, my = look

dx, dy = mx - self.rect.centerx, my - self.rect.centery

self.angle = math.degrees(math.atan2(-dy, dx)) + 90

self.rot\_image = pygame.transform.rotate(self.rectangle, self.angle)

self.rot\_image\_rect = self.rot\_image.get\_rect(center=self.rect.center)

def mov(self, walls, player\_movement, fire):

val = "successful"

if player\_movement == 'w' and not self.rect.top - self.rect.height / 3 < 0:

self.rect.move\_ip(0, -self.speed)

if pygame.sprite.spritecollideany(self, walls):

self.rect.move\_ip(0, self.speed)

if player\_movement == 's' and not self.rect.bottom + self.rect.height / 3 > 600:

self.rect.move\_ip(0, self.speed)

if pygame.sprite.spritecollideany(self, walls):

self.rect.move\_ip(0, -self.speed)

if player\_movement == 'a' and not (self.rect.left - self.rect.width / 3 < 0):

self.rect.move\_ip(-self.speed, 0)

if pygame.sprite.spritecollideany(self, walls):

self.rect.move\_ip(self.speed, 0)

if player\_movement == 'd' and not (self.rect.right + self.rect.width / 3 > 800):

self.rect.move\_ip(self.speed, 0)

if pygame.sprite.spritecollideany(self, walls):

self.rect.move\_ip(-self.speed, 0)

if fire == 'fire' and self.fire\_wait <= 0:

val = self.fire()

self.fire\_wait = 60

if self.fire\_wait > 0:

self.fire\_wait -= self.firing\_speed

self.orientation.x = self.rect.x

self.orientation.y = self.rect.y

self.orientation.angle = self.angle

return val

def change\_name(self, name):

self.name = name

self.orientation.name = "0," + str(name)

def colisions(self):

self.leaderboard.change\_places(self.players)

hit = pygame.sprite.Group()

for bullet in self.enemies:

bullet.mov()

if pygame.sprite.spritecollideany(bullet,

self.all\_sprites) or 1000 < bullet.rect.x or bullet.rect.x < 0 or 800 < bullet.rect.y or bullet.rect.y < 0:

bullet.kill()

elif pygame.sprite.spritecollideany(bullet, self.players):

bullet.owner.score += 1

hit.add(bullet)

for player in self.players:

if pygame.sprite.spritecollideany(player, hit):

player.rect.center = self.teleport(self.all\_sprites)

for bullet in hit:

bullet.kill()

del hit

class Walls(pygame.sprite.Sprite):

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

super(Game.Walls, self).\_\_init\_\_()

self.center = (random.randint(1, 801), random.randint(1, 601))

self.size = (random.randint(50, 201), random.randint(50, 201))

self.rectangle = pygame.Surface(self.size, pygame.SRCALPHA)

self.rectangle.fill(pygame.Color('dark gray'))

self.rect = self.rectangle.get\_rect()

self.rect.move\_ip(self.center)

self.rect = self.rectangle.get\_rect(center=self.rect.center)

self.orientation = Orientation(self.rect.x, self.rect.y, self.rect.width, self.rect.height, 0,

'gray', "")

def Serialize(self):

return pickle.dumps(self.orientation)

class LeaderBoard:

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

self.bord = self.Block((800, 0), (200, 600), 'black')

self.line = self.Block((800, 0), (8, 600), 'white')

self.txts = []

self.font = pygame.font.Font('freesansbold.ttf', 32)

def set\_place(self):

place = [self.bord, self.line]

return place

def change\_places(self, players):

self.txts.clear()

text = self.font.render('leaderboard', True, (255, 0, 0), (0, 0, 0))

text\_name = "leaderboard"

textRect = text.get\_rect()

textRect.center = (920, 15)

self.txts.append((text\_name, textRect, 40, 'white'))

leader\_place = 75

for player in players:

space = " "

for i in range(16 - len(str(player.name)) - len(str(player.score))):

space += " "

text\_name = str(player.name) + space + str(player.score)

text = self.font.render(text\_name, True, (255, 0, 0),

(0, 0, 0))

textRect = text.get\_rect()

textRect.center = (300 // 2 + 800, leader\_place)

self.txts.append((text\_name, textRect, 30, 'white'))

leader\_place += 50

def Serialize(self, num):

orientation = Orientation(self.txts[num][1].x, self.txts[num][1].y, self.txts[num][1].width,

self.txts[num][1].height, self.txts[num][2], self.txts[num][3], self.txts[num][0])

return pickle.dumps(orientation)

class Block(pygame.sprite.Sprite):

def \_\_init\_\_(self, place, size, color):

super(Game.LeaderBoard.Block, self).\_\_init\_\_()

self.rectangle = pygame.Surface(size, pygame.SRCALPHA)

self.rectangle.fill(pygame.Color(color))

self.rect = self.rectangle.get\_rect()

self.rect.move\_ip(place)

self.orientation = Orientation(self.rect.x, self.rect.y, self.rect.width,

self.rect.height, 0, color, "")

def Serialize(self):

return pickle.dumps(self.orientation)

def winner(self, players):

if len(players) != 0:

winner = players.sprites()[0]

for player in players:

if winner.score < player.score:

winner = player

color = (255, 215, 0)

size = 60

font = pygame.font.Font('freesansbold.ttf', size)

text = font.render(str(winner.name) + " is the winner!!!", True, color)

text\_name = str(winner.name) + " is the winner!!!"

textRect = text.get\_rect()

textRect.center = (450, 300)

self.txts.append((text\_name, textRect, size, color))

class Orientation:

def \_\_init\_\_(self, x, y, width, height, angle, color, name):

self.x = x

self.y = y

self.width = width

self.height = height

self.angle = angle

self.color = color

self.name = name

def starting(database\_ip, \*args):

print("extras=> ")

for arg in args:

print(arg)

pygame.init()

try:

me = ServerSide(database\_ip)

me.game\_maker()

except Exception as e:

logging.error(e)

pygame.quit()

def main():

starting(str(socket.gethostname()))

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

main()

## GUI

import logging

import pickle

import socket

import tkinter as tk

import tkinter.ttk as ttk

import sys

import time

from Constants import constant

import game\_server

import multiprocessing

from multiprocessing import freeze\_support

import hashlib

import pygame

from pygame.locals import (

K\_w,

K\_s,

K\_a,

K\_d,

K\_ESCAPE,

KEYDOWN,

QUIT,

)

class ClientSide:

def \_\_init\_\_(self, ip, port, name):

logging.debug("client begin")

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

print("GUI- " + str(ip) + "," + str(port))

self.my\_socket.connect((ip, port))

logging.info("connect to server at {0} with port {1}".format(ip, port))

pygame.init()

self.SCREEN\_WIDTH = 1100

self.SCREEN\_HEIGHT = 600

pygame.display.set\_mode((self.SCREEN\_WIDTH, self.SCREEN\_HEIGHT))

self.display\_surface = pygame.display.set\_mode((1000, 20))

pygame.display.set\_caption('Show Text')

self.screen = pygame.display.set\_mode((self.SCREEN\_WIDTH, self.SCREEN\_HEIGHT))

self.game = ""

self.name = name

class Demo\_print(pygame.sprite.Sprite):

def \_\_init\_\_(self, x, y, width, height, angle, color, name):

super(ClientSide.Demo\_print, self).\_\_init\_\_()

self.rectangle = pygame.Surface((width, height), pygame.SRCALPHA)

self.rect = self.rectangle.get\_rect()

self.rect.move\_ip(x, y)

self.color = color

self.angle = angle

self.rectangle.fill(pygame.Color(self.color))

self.rot\_image = pygame.transform.rotate(self.rectangle, self.angle)

self.rot\_image\_rect = self.rot\_image.get\_rect(center=self.rect.center)

self.name = name

self.font = pygame.font.Font('freesansbold.ttf', int(self.angle))

try:

obj\_name = self.name.split(",")

self.text = self.font.render(obj\_name[1], True, self.color, (0, 0, 0))

except:

self.text = self.font.render(self.name, True, self.color, (0, 0, 0))

self.text.set\_colorkey((0, 0, 0))

def change\_color(self, color):

self.color = color

self.rectangle.fill(pygame.Color(self.color))

self.rot\_image = pygame.transform.rotate(self.rectangle, self.angle)

self.rot\_image\_rect = self.rot\_image.get\_rect(center=self.rect.center)

try:

obj\_name = self.name.split(",")

self.text = self.font.render(obj\_name[1], True, self.color, (0, 0, 0))

except:

self.text = self.font.render(self.name, True, self.color, (0, 0, 0))

def game\_run(self):

running = True

while running:

for event in pygame.event.get():

if event.type == KEYDOWN:

if event.key == K\_ESCAPE:

self.my\_socket.send(pickle.dumps("quit"))

running = self.close(True)

elif event.type == QUIT:

self.my\_socket.send(pickle.dumps("quit"))

running = self.close(True)

movement = self.mov()

self.send(pickle.dumps((constant.USER\_ACTION, self.name, movement)))

self.screen.fill((0, 0, 0))

self.read()

self.built\_all(self.game)

pygame.display.flip()

pygame.quit()

def built\_all(self, game\_obj):

objects = []

for obj in game\_obj:

objects.append(pickle.loads(obj))

sprits = []

for obj in objects:

sprit = self.Demo\_print(obj.x, obj.y, obj.width, obj.height, obj.angle, obj.color, obj.name)

sprits.append(sprit)

for sprit in sprits:

if sprit.name == "":

self.screen.blit(sprit.rot\_image, sprit.rot\_image\_rect.topleft)

else:

obj\_name = sprit.name.split(",")

if len(obj\_name) == 1:

self.screen.blit(sprit.text, sprit.rect)

else:

if str(obj\_name[1]) == str(self.name):

sprit.change\_color('blue')

self.screen.blit(sprit.rot\_image, sprit.rot\_image\_rect.topleft)

def mov(self):

mx, my = pygame.mouse.get\_pos()

mouse = pygame.mouse.get\_pressed(3)

move = "no input"

val = 'no input'

pressed\_keys = pygame.key.get\_pressed()

if pressed\_keys[K\_w]:

move = 'w'

if pressed\_keys[K\_s]:

move = 's'

if pressed\_keys[K\_a]:

move = 'a'

if pressed\_keys[K\_d]:

move = 'd'

if mouse[0]:

val = "fire"

return move, val, (mx, my)

def read(self):

try:

lenoflen = int(self.my\_socket.recv(4).decode())

lenght = int(self.my\_socket.recv(lenoflen).decode())

print(str(lenght))

self.game = self.my\_socket.recv(lenght)

self.game = pickle.loads(self.game)

if self.game == "close":

print("exit")

self.close(False)

except Exception as e:

logging.error(f"reading error in client\_side in GUI: {e}")

self.close(True)

def send(self, data):

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

def close(self, cause):

if cause:

logging.debug("client close, client side")

else:

logging.debug("client close, server side")

return False

class Orientation:

def \_\_init\_\_(self, x, y, width, height, angle, color, name):

self.x = x

self.y = y

self.width = width

self.height = height

self.angle = angle

self.name = name

self.color = color

class IpCatcher:

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

\_bgcolor = '#c1cdcd' # X11 color: 'gray85'

\_fgcolor = '#000000' # X11 color: 'black'

\_compcolor = '#d9d9d9' # X11 color: 'gray85'

\_ana1color = '#d9d9d9' # X11 color: 'gray85'

\_ana2color = '#ececec' # Closest X11 color: 'gray92'

self.ip = ""

self.top = top

top.geometry("600x450+504+171")

top.minsize(360, 220)

top.maxsize(500, 380)

top.resizable(1, 1)

top.title("Enter to game")

top.configure(background=\_bgcolor)

top.configure(highlightbackground="#d9d9d9")

top.configure(highlightcolor="black")

self.style = ttk.Style()

self.Label1 = tk.Label(self.top)

self.Label1.place(relx=0.05, rely=0.01, height=50, width=280)

self.Label1.configure(activebackground="#f9f9f9")

self.Label1.configure(activeforeground="black")

self.Label1.configure(anchor='w')

self.Label1.configure(background=\_bgcolor)

self.Label1.configure(compound='center')

self.Label1.configure(disabledforeground="#a3a3a3")

self.Label1.configure(foreground="#000000")

self.Label1.configure(highlightbackground="#d9d9d9")

self.Label1.configure(highlightcolor="black")

self.Label1.configure(text="ENTER SERVER IP")

self.Label1.config(font=('Comic Sans MS', 20))

self.Entry1 = tk.Entry(self.top)

self.Entry1.place(relx=0.01, rely=0.3, height=40, relwidth=0.45)

self.Entry1.configure(background="#e0eeee")

self.Entry1.configure(disabledforeground="#ff3030")

self.Entry1.configure(font="TkFixedFont")

self.Entry1.configure(foreground="#000000")

self.Entry1.configure(highlightbackground="#d9d9d9")

self.Entry1.configure(highlightcolor="black")

self.Entry1.configure(insertbackground="black")

self.Entry1.configure(selectbackground="blue")

self.Entry1.configure(selectforeground="white")

self.Button1 = tk.Button(self.top)

self.Button1.place(relx=0.1, rely=0.733, height=60, width=115)

self.Button1.configure(activebackground="#838b8b")

self.Button1.configure(activeforeground="#000000")

self.Button1.configure(background="#e0eeee")

self.Button1.configure(compound='left')

self.Button1.configure(disabledforeground="#a3a3a3")

self.Button1.configure(foreground="#000000")

self.Button1.configure(highlightbackground="#d9d9d9")

self.Button1.configure(highlightcolor="black")

self.Button1.configure(pady="0")

self.Button1.configure(text="ENTER")

self.Button1.configure(command=self.enter\_ip)

self.style.configure('TSizegrip', background=\_bgcolor)

self.TSizegrip1 = ttk.Sizegrip(self.top)

self.TSizegrip1.place(anchor='se', relx=1.0, rely=1.0)

def enter\_ip(self):

self.ip = str(self.Entry1.get())

self.top.destroy()

class HomeScreen:

def \_\_init\_\_(self, top, connectir\_ip, connectir\_port, name):

\_bgcolor = '#ffe4e1' # X11 color: 'gray85'

\_fgcolor = '#000000' # X11 color: 'black'

\_compcolor = '#d9d9d9' # X11 color: 'gray85'

\_ana1color = '#d9d9d9' # X11 color: 'gray85'

\_ana2color = '#ececec' # Closest X11 color: 'gray92'

logging.debug("connection begin")

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

self.my\_socket.connect((connectir\_ip, connectir\_port))

self.name = name

self.gui\_run = True

self.send(pickle.dumps([constant.HOMESCREEN\_CONNECTS, self.name.encode()]))

data = self.read()

self.player = data[0]

self.position = data[1]

self.send(pickle.dumps([constant.SERVER\_REQUEST, self.name.encode()]))

data = self.read()

logging.debug(f"server to connect: {data}")

if not data:

self.open\_server((connectir\_ip, "helo"))

self.send(pickle.dumps([constant.SERVER\_REQUEST, self.name.encode()]))

data = self.read()

logging.debug(f"server to connect: {data}")

ip, port = data

self.ip = ip

self.port = port

self.top = top

top.geometry("600x450+504+171")

top.minsize(120, 1)

top.maxsize(1604, 881)

top.resizable(1, 1)

top.title("Enter to game")

top.configure(background=\_bgcolor)

top.configure(highlightbackground="#d9d9d9")

top.configure(highlightcolor="black")

self.style = ttk.Style()

self.Label4 = tk.Label(self.top)

self.Label1 = tk.Label(self.top)

self.Label1.place(relx=0.05, rely=0.01, height=60, width=600)

self.Label1.configure(activebackground="#f9f9f9")

self.Label1.configure(activeforeground="black")

self.Label1.configure(anchor='w')

self.Label1.configure(background=\_bgcolor)

self.Label1.configure(compound='center')

self.Label1.configure(disabledforeground="#a3a3a3")

self.Label1.configure(foreground="#000000")

self.Label1.configure(highlightbackground="#d9d9d9")

self.Label1.configure(highlightcolor="black")

self.Label1.configure(text='''welcome to GENERCUBE\n HOME SCREEN ''' + str(self.player[5]))

self.Label1.config(font=('Comic Sans MS', 20))

self.Label2 = tk.Label(self.top)

self.Label2.place(relx=0, rely=0.2, height=30, width=200)

self.Label2.configure(activebackground="#f9f9f9")

self.Label2.configure(activeforeground="black")

self.Label2.configure(anchor='w')

self.Label2.configure(background=\_bgcolor)

self.Label2.configure(compound='left')

self.Label2.configure(disabledforeground="#a3a3a3")

self.Label2.configure(foreground="#000000")

self.Label2.configure(highlightbackground="#d9d9d9")

self.Label2.configure(highlightcolor="black")

self.Label2.configure(text='''Your personal record: ''' + str(self.player[4]))

self.Label2.config(font=('Comic Sans MS', 10))

self.Label3 = tk.Label(self.top)

self.Label3.place(relx=0, rely=0.4, height=30, width=360)

self.Label3.configure(activebackground="#f9f9f9")

self.Label3.configure(activeforeground="black")

self.Label3.configure(anchor='w')

self.Label3.configure(background=\_bgcolor)

self.Label3.configure(compound='left')

self.Label3.configure(disabledforeground="#a3a3a3")

self.Label3.configure(foreground="#000000")

self.Label3.configure(highlightbackground="#d9d9d9")

self.Label3.configure(highlightcolor="black")

self.Label3.configure(text='''You are playing this game since ''' + self.player[2])

self.Label3.config(font=('Comic Sans MS', 10))

self.Label4 = tk.Label(self.top)

self.Label4.place(relx=0, rely=0.6, height=30, width=250)

self.Label4.configure(activebackground="#f9f9f9")

self.Label4.configure(activeforeground="black")

self.Label4.configure(anchor='w')

self.Label4.configure(background=\_bgcolor)

self.Label4.configure(compound='left')

self.Label4.configure(disabledforeground="#a3a3a3")

self.Label4.configure(foreground="#000000")

self.Label4.configure(highlightbackground="#d9d9d9")

self.Label4.configure(highlightcolor="black")

self.Label4.configure(text=self.position\_placer(self.position))

self.Label4.config(font=('Comic Sans MS', 10))

self.Button1 = tk.Button(self.top)

self.Button1.place(relx=0.1, rely=0.733, height=60, width=115)

self.Button1.configure(activebackground="#eecbad")

self.Button1.configure(activeforeground="#000000")

self.Button1.configure(background="#ffefd5")

self.Button1.configure(compound='left')

self.Button1.configure(disabledforeground="#a3a3a3")

self.Button1.configure(foreground="#000000")

self.Button1.configure(highlightbackground="#d9d9d9")

self.Button1.configure(highlightcolor="black")

self.Button1.configure(pady="0")

self.Button1.configure(text='''ENTER GAME''')

self.Button1.configure(command=self.Enter\_game)

self.Button2 = tk.Button(self.top)

self.Button2.place(relx=0.7, rely=0.733, height=60, width=115)

self.Button2.configure(activebackground="#eecbad")

self.Button2.configure(activeforeground="#000000")

self.Button2.configure(background="#ffefd5")

self.Button2.configure(compound='left')

self.Button2.configure(disabledforeground="#a3a3a3")

self.Button2.configure(foreground="#000000")

self.Button2.configure(highlightbackground="#d9d9d9")

self.Button2.configure(highlightcolor="black")

self.Button2.configure(pady="0")

self.Button2.configure(text='''quit''')

self.Button2.configure(command=self.quit)

self.style.configure('TSizegrip', background=\_bgcolor)

self.TSizegrip1 = ttk.Sizegrip(self.top)

self.TSizegrip1.place(anchor='se', relx=1.0, rely=1.0)

def open\_server(self, database\_ip):

server = multiprocessing.Process(target=game\_server.starting, args=database\_ip)

server.start()

time.sleep(2)

def quit(self):

self.send(pickle.dumps([constant.HOMESCREEN\_QUITING, self.name.encode()]))

print('---------------quit----------------')

sys.stdout.flush()

self.my\_socket.close()

self.top.destroy()

self.gui\_run = False

# sys.exit()

def read(self):

lenoflen = int(self.my\_socket.recv(4).decode())

lenght = int(self.my\_socket.recv(lenoflen).decode())

data = self.my\_socket.recv(lenght)

data = pickle.loads(data)

return data

def send(self, data):

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

def position\_placer(self, position):

if position == 1:

return "You are the best in the game!"

else:

return "Your position the player ranking is " + str(position) + "!"

def Enter\_game(self):

self.my\_socket.close()

self.top.destroy()

class TopLevelMother:

def \_\_init\_\_(self, top, headline, botton1, error\_msg, ip, is\_connect):

\_bgcolor = '#ffe4e1' # X11 color: 'gray85'

\_fgcolor = '#000000' # X11 color: 'black'

\_compcolor = '#d9d9d9' # X11 color: 'gray85'

\_ana1color = '#d9d9d9' # X11 color: 'gray85'

\_ana2color = '#ececec' # Closest X11 color: 'gray92'

logging.debug("connection begin")

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

if is\_connect != 0:

self.database\_ip = ip

self.connector\_ip = ip

self.connector\_port = 7777

try:

self.my\_socket.connect((self.connector\_ip, self.connector\_port))

except Exception as e:

print(e)

sys.exit()

logging.info("connect to server at {0} with port {1}".format(self.connector\_ip, self.connector\_port))

else:

self.my\_socket = ip

self.database\_ip = self.my\_socket.getsockname()

self.top = top

top.geometry("600x450+504+171")

top.minsize(120, 1)

top.maxsize(1604, 881)

top.resizable(1, 1)

top.title("Enter to game")

top.configure(background=\_bgcolor)

top.configure(highlightbackground="#d9d9d9")

top.configure(highlightcolor="black")

self.style = ttk.Style()

self.Label4 = tk.Label(self.top)

self.error\_msg = error\_msg

self.name = ""

headline = headline.split(",")

self.Label1 = tk.Label(self.top)

self.Label1.place(relx=headline[1], rely=headline[2], height=headline[3], width=600)

self.Label1.configure(activebackground=\_bgcolor)

self.Label1.configure(activeforeground="black")

self.Label1.configure(anchor='w')

self.Label1.configure(background=\_bgcolor)

self.Label1.configure(compound='center')

self.Label1.configure(disabledforeground="#a3a3a3")

self.Label1.configure(foreground="#000000")

self.Label1.configure(highlightbackground="#d9d9d9")

self.Label1.configure(highlightcolor="black")

self.Label1.configure(text=headline[0])

self.Label1.config(font=('Comic Sans MS', 25))

self.Entry1 = tk.Entry(self.top)

self.Entry1.place(relx=0.01, rely=0.25, height=40, relwidth=0.307)

self.Entry1.configure(background="#fdf5e6")

self.Entry1.configure(disabledforeground="#eee0e5")

self.Entry1.configure(font="TkFixedFont")

self.Entry1.configure(foreground="#000000")

self.Entry1.configure(highlightbackground="#d9d9d9")

self.Entry1.configure(highlightcolor="black")

self.Entry1.configure(insertbackground="black")

self.Entry1.configure(selectbackground="blue")

self.Entry1.configure(selectforeground="white")

self.Label2 = tk.Label(self.top)

self.Label2.place(relx=0, rely=0.15, height=20, width=114)

self.Label2.configure(activebackground=\_bgcolor)

self.Label2.configure(activeforeground="black")

self.Label2.configure(anchor='w')

self.Label2.configure(background=\_bgcolor)

self.Label2.configure(compound='left')

self.Label2.configure(disabledforeground="#a3a3a3")

self.Label2.configure(foreground="#000000")

self.Label2.configure(highlightbackground="#d9d9d9")

self.Label2.configure(highlightcolor="black")

self.Label2.configure(text='''ENTER NAME''')

self.Label2.config(font=('Comic Sans MS', 10))

self.Entry2 = tk.Entry(self.top)

self.Entry2.place(relx=0.01, rely=0.5, height=40, relwidth=0.307)

self.Entry2.configure(background="#fdf5e6")

self.Entry2.configure(disabledforeground="#eee0e5")

self.Entry2.configure(font="TkFixedFont")

self.Entry2.configure(foreground="#000000")

self.Entry2.configure(highlightbackground="#d9d9d9")

self.Entry2.configure(highlightcolor="black")

self.Entry2.configure(insertbackground="black")

self.Entry2.configure(selectbackground="blue")

self.Entry2.configure(show="\*")

self.Entry2.configure(selectforeground="white")

self.Label3 = tk.Label(self.top)

self.Label3.place(relx=0, rely=0.4, height=20, width=130)

self.Label3.configure(activebackground=\_bgcolor)

self.Label3.configure(activeforeground="black")

self.Label3.configure(anchor='w')

self.Label3.configure(background=\_bgcolor)

self.Label3.configure(compound='left')

self.Label3.configure(disabledforeground="#a3a3a3")

self.Label3.configure(foreground="#000000")

self.Label3.configure(highlightbackground="#d9d9d9")

self.Label3.configure(highlightcolor="black")

self.Label3.configure(text='''ENTER PASSWORD''')

self.Label3.config(font=('Comic Sans MS', 10))

self.Button1 = tk.Button(self.top)

self.Button1.place(relx=0.1, rely=0.733, height=80, width=130)

self.Button1.configure(activebackground="#eecbad")

self.Button1.configure(activeforeground="#000000")

self.Button1.configure(background="#ffefd5")

self.Button1.configure(compound='left')

self.Button1.configure(disabledforeground="#a3a3a3")

self.Button1.configure(foreground="#000000")

self.Button1.configure(highlightbackground="#d9d9d9")

self.Button1.configure(highlightcolor="black")

self.Button1.configure(pady="0")

self.Button1.configure(text=botton1)

self.Button1.configure(command=self.entername)

self.Button1.config(font=('Comic Sans MS', 10))

self.Button2 = tk.Button(self.top)

self.Button2.place(relx=0.7, rely=0.733, height=80, width=130)

self.Button2.configure(activebackground="#eecbad")

self.Button2.configure(activeforeground="#000000")

self.Button2.configure(background="#ffefd5")

self.Button2.configure(compound='left')

self.Button2.configure(disabledforeground="#a3a3a3")

self.Button2.configure(foreground="#000000")

self.Button2.configure(highlightbackground="#d9d9d9")

self.Button2.configure(highlightcolor="black")

self.Button2.configure(pady="0")

self.Button2.configure(text='''quit''')

self.Button2.configure(command=self.quit)

self.Button2.config(font=('Comic Sans MS', 10))

self.style.configure('TSizegrip', background=\_bgcolor)

self.TSizegrip1 = ttk.Sizegrip(self.top)

self.TSizegrip1.place(anchor='se', relx=1.0, rely=1.0)

def print\_error(self):

self.Label4 = tk.Label(self.top)

self.Label4.place(relx=0, rely=0.6, height=20, width=120)

self.Label4.configure(activebackground='#ffe4e1')

self.Label4.configure(activeforeground="black")

self.Label4.configure(anchor='w')

self.Label4.configure(background='#ffe4e1')

self.Label4.configure(compound='left')

self.Label4.configure(disabledforeground="#a3a3a3")

self.Label4.configure(foreground="red")

self.Label4.configure(highlightbackground="#d9d9d9")

self.Label4.configure(highlightcolor="black")

self.Label4.configure(text=self.error\_msg)

self.Label4.config(font=('Comic Sans MS', 10))

def delete\_error(self):

self.Label4.destroy()

def quit(self, \*args):

try:

self.send(pickle.dumps(constant.QUITING))

finally:

print('---------------quit----------------')

for arg in args:

print('another arg:', arg)

sys.stdout.flush()

sys.exit()

def send(self, data):

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

def entername(self):

logging.debug(f"username = {self.Entry1.get()}")

logging.debug(f"password = {self.Entry2.get()}")

self.send(pickle.dumps((constant.USER\_CONNECTING, self.Entry1.get(), self.Entry2.get(), "", "player")))

try:

lenoflen = int(self.my\_socket.recv(4).decode())

lenght = int(self.my\_socket.recv(lenoflen).decode())

data = self.my\_socket.recv(lenght)

data = pickle.loads(data)

if not data:

self.print\_error()

self.Entry1.delete(0, 'end')

self.Entry2.delete(0, 'end')

else:

self.name = str(self.Entry1.get())

self.my\_socket.send(pickle.dumps(constant.QUITING))

self.my\_socket.close()

self.delete\_error()

self.top.destroy()

except Exception as e:

logging.error(f"TopLevelMother error occurred: {e}")

sys.exit()

class TopLevel1(TopLevelMother):

def \_\_init\_\_(self, top, ip):

super(TopLevel1, self).\_\_init\_\_(top, '''welcome to\nGENERCUBE,0.5,0.1, 160''', '''CONNECT TO USER''',

'''INCORRECT NAME OR PASSWORD''', ip, 1)

self.ip = ip

self.Button3 = tk.Button(self.top)

self.Button3.place(relx=0.8, rely=0.478, height=54, width=110)

self.Button3.configure(activebackground="#eecbad")

self.Button3.configure(activeforeground="#000000")

self.Button3.configure(background="#ffefd5")

self.Button3.configure(compound='left')

self.Button3.configure(disabledforeground="#a3a3a3")

self.Button3.configure(foreground="#000000")

self.Button3.configure(highlightbackground="#d9d9d9")

self.Button3.configure(highlightcolor="black")

self.Button3.configure(pady="0")

self.Button3.configure(text='''create new user''')

self.Button3.configure(command=self.move\_level2)

self.Button3.config(font=('Comic Sans MS', 10))

def move\_level2(self):

self.top.withdraw()

root = tk.Tk()

root.protocol('WM\_DELETE\_WINDOW', root.destroy)

\_w2 = TopLevel2(root, self, self.my\_socket)

self.name = \_w2.name

def level2\_got\_in(self, name):

self.top.deiconify()

self.name = name

if name != "":

self.my\_socket.close()

self.top.destroy()

class TopLevel2(TopLevelMother):

def \_\_init\_\_(self, top, level1, ip):

super(TopLevel2, self).\_\_init\_\_(top, '''A NEW USER APPEAR!,0.05,0.01, 60''', '''CREATE NEW USER''',

'''not possible''', ip, 0)

self.level1 = level1

self.Button3 = tk.Button(self.top)

self.Button3.place(relx=0.8, rely=0.478, height=54, width=110)

self.Button3.configure(activebackground="#ececec")

self.Button3.configure(activeforeground="#000000")

self.Button3.configure(background="#ffefd5")

self.Button3.configure(compound='left')

self.Button3.configure(disabledforeground="#a3a3a3")

self.Button3.configure(foreground="#000000")

self.Button3.configure(highlightbackground="#d9d9d9")

self.Button3.configure(highlightcolor="black")

self.Button3.configure(pady="0")

self.Button3.configure(text='''I have a user already''')

self.Button3.configure(command=self.back\_to\_level1)

self.Entry3 = tk.Entry(self.top)

self.Entry3.place(relx=0.51, rely=0.25, height=40, relwidth=0.307)

self.Entry3.configure(background="#fdf5e6")

self.Entry3.configure(disabledforeground="#a3a3a3")

self.Entry3.configure(font="TkFixedFont")

self.Entry3.configure(foreground="#000000")

self.Entry3.configure(highlightbackground="#d9d9d9")

self.Entry3.configure(highlightcolor="black")

self.Entry3.configure(insertbackground="black")

self.Entry3.configure(selectbackground="blue")

self.Entry3.configure(selectforeground="white")

self.Label5 = tk.Label(self.top)

self.Label5.place(relx=0.5, rely=0.15, height=20, width=144)

self.Label5.configure(activebackground="#f9f9f9")

self.Label5.configure(activeforeground="black")

self.Label5.configure(anchor='w')

self.Label5.configure(background="#ffe4e1")

self.Label5.configure(compound='left')

self.Label5.configure(disabledforeground="#a3a3a3")

self.Label5.configure(foreground="#000000")

self.Label5.configure(highlightbackground="#d9d9d9")

self.Label5.configure(highlightcolor="black")

self.Label5.configure(text='''ENTER CLIENT NAME''')

self.Label5.config(font=('Comic Sans MS', 10))

def back\_to\_level1(self):

self.top.destroy()

self.level1.level2\_got\_in(self.name)

def entername(self):

logging.debug(f"username = {self.Entry1.get()}")

logging.debug(f"password = {self.Entry2.get()}")

date = time.localtime()[0:-4]

update\_date = str(date[0]) + "/" + str(date[1]) + "/" + str(date[2]) + " " + str(date[3]) + ":" + str(

date[4]).zfill(2)

self.send(pickle.dumps(

(constant.USER\_CONNECTING, self.Entry1.get(), self.Entry2.get(), update\_date, self.Entry3.get())))

try:

lenoflen = int(self.my\_socket.recv(4).decode())

lenght = int(self.my\_socket.recv(lenoflen).decode())

data = self.my\_socket.recv(lenght)

data = pickle.loads(data)

if not data:

self.print\_error()

self.Entry1.delete(0, 'end')

self.Entry2.delete(0, 'end')

self.Entry3.delete(0, 'end')

else:

self.name = str(self.Entry1.get())

self.my\_socket.send(pickle.dumps(constant.QUITING))

self.my\_socket.close()

self.delete\_error()

self.back\_to\_level1()

except Exception as e:

logging.error(f"TopLevel2 error occurred: {e}")

self.back\_to\_level1()

class screen\_manager:

def get\_ip(self):

root = tk.Tk()

root.protocol('WM\_DELETE\_WINDOW', root.destroy)

start\_tk = IpCatcher(root)

root.mainloop()

return start\_tk.ip

def entering(self):

ip = self.get\_ip()

root = tk.Tk()

root.protocol('WM\_DELETE\_WINDOW', root.destroy)

\_w1 = TopLevel1(root, ip)

# root.after(1000, \_w1.loop, root)

root.mainloop()

return \_w1.connector\_ip, \_w1.connector\_port, \_w1.name

def stay\_screen(self, ip, port, name):

root = tk.Tk()

root.protocol('WM\_DELETE\_WINDOW', root.destroy)

home\_screen = HomeScreen(root, ip, port, name)

root.mainloop()

return home\_screen.ip, home\_screen.port, home\_screen.gui\_run

def screen\_control\_loop(self):

connector\_ip, connector\_port, name = self.entering()

gui\_run = True

while gui\_run:

port, ip, gui\_run = self.stay\_screen(connector\_ip, connector\_port, name)

if not gui\_run:

break

logging.basicConfig(level=logging.DEBUG)

try:

me = ClientSide(ip, port, name)

me.game\_run()

except Exception as e:

logging.error(e)

continue

def main():

freeze\_support()

SM = screen\_manager()

SM.screen\_control\_loop()

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

main()

# רפלקציה

אני למדתי הרבה במהלך הפרויקט, גם על תקשורת בעזרת פעולת הselect של פייטון גם על פייטון עצמו, וגם על הבנת המגבלות שיש בזמן עבודה על פרויקט. היו דברים שהורדתי מהפרויקט בגלל מגבלת הזמן, כגון שליחת סיסמה חדשה למשתמש בעזרת מייל בלחיצת כפתור במשך ההתחברות למערכת.

אני אזכור את הפרויקט הזה לעתיד כניסיון ראשון בהכנת פרויקט משמעותי הדורש הרבה זמן, השקעה ועבודה. אני אזכור שכדי להכין מוצר רציתי בתכנות יהיה עלי לדעת ולהבין את כל החלקים המרכיבים את המוצר ולדעת אילו חלקים להוסיף ועל אילו לוותר.

אם הייתי מתחיל פרויקט נוסף בתכנות הייתי מתחיל לעבוד קשה על הפרויקט מהיום הראשון ומלכתחילה מכניס פחות חלקים שאני מכיר בהתחשבות הזמן שיש לי להכנת הפרויקט. ועם ישיבה ממושכת מתחילת העבודה והכנת תוכנית עבודה יעילה וכנה ישנו סיכוי שהייתי גם יכול להכניס את החלקים שהורדתי מהפרויקט בסוף.

לבסוף, אני אקח איתי המון מידע שלמדתי במהלך הכנת הפרויקט, ולא רק מידע שלמדתי למען הכנת הפרויקט. למדתי הרבה על תפעול GitHub, tkinter, socket ועל העברת קבצי קוד לקבצי executable.

# ביבליוגרפיה

W3schools –

<https://www.w3schools.com/python/python_reference.asp>

pygame documentation-

<https://www.pygame.org/docs/>

# נספחים

בשביל ייצור מסכי הכניסה למערכת ואת מסך הבית של ה GUI, השתמשתי בתוכנה PAGE. תוכנת ה PAGE היא אפליקציה המאפשרת ליצור דפי tkinter בקלות.