**פרויקט רשתות ואינטרנט 1 – המרדף**

**מגישים:**

# יהונתן מזרחי 3228999675

# אביאל דרעי 206299729

**סביבת עבודה:**

# מערכת הפעלה: windows 10 pro

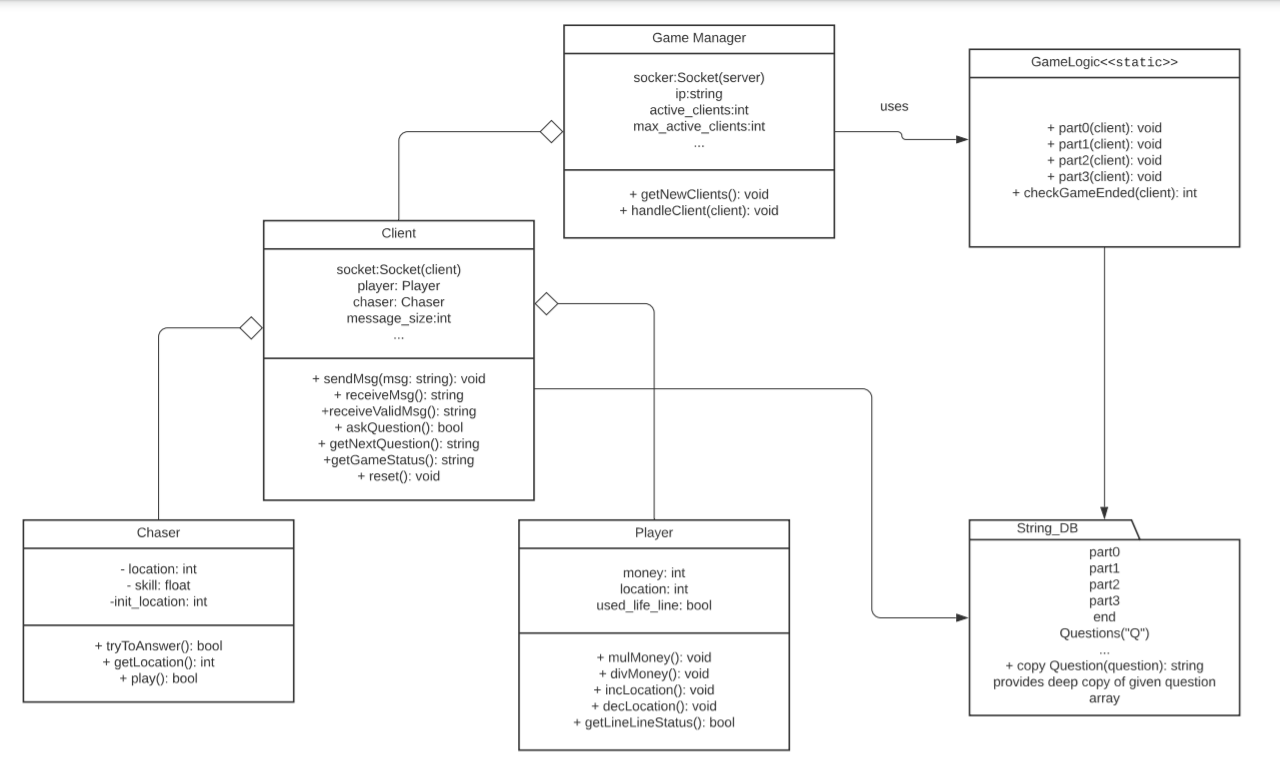
# שפת תכנות: python 3.9.0

**רקע תיאורטי:**

**הסבר לכל קובץ קוד:**

**:Backend**

תרשים UML:



צ'ייסר(Chaser):

מחלקה שתפקידה לייצג את הרודף במשחק, מסוגל לענות תשובה נכונה בהסתברות המיוצגת בתכונה skill(מקרה פרטי אצלנו בקוד: skill = 0.75) ולהתקדם בהתאם.

שחקן(Player):

מחלקה שתפקידה לייצג את השחקן, בעלת תכונות המציגות את כמות הכסף, המיקום הנוכחי והאם השחקן השתמש כבר בגלגל ההצלה. בנוסף, כתבנו במחלקה פונקציות שמאפשרות לגשת ולנהל את אותן תכונות.

לקוח(Client):

מחלקה המייצגת את כל מה שקשור ללקוח(בfrontend).

המחלקה מכילה מופעים של המחלקות צ'ייסר ושחקן ובנוסף מטפלת בכל התקשורת מול השחקן בfrontend.

כדי לנהל את התקשורת הגדרנו את התכונה socket ואת הפונקציות reciveMsg, reciveValidMsg, getMsg וaskQuestion שאפשר בעזרתן לשלוח ולקבל הודעות מהשחקן.

בנוסף, הגדרנו כמה פונקציות נוספות שיעזרו לנו לדעת מה השאלה הבאה שצריך לשלוח ואת gameStatus שמחזירה מחרוזת המייצגת את מצב הלוח.

מנהל המשחק(GameManager):

זאת המחלקה שמחברת את הכל, ממנה הקוד מתחיל. יש בה socket שמייצג את הצד של השרת ותכונות שמאפשרות לעקוב אחר כמות הלקוחות שמחוברים בו זמנית לשרת.

המחלקה הזאת עובדת עם תהליכונים כך שתהליכון הראשי שרץ ברקע הוא של הפונקציה getNewClients שמטרתו הוא לבדוק האם יש לקוחות חדשים שרוצים להתחבר ולהגיב בהתאם. בנוסף נפתח לכל לקוח תהליכון נפרד handleClient, שמקבל את המופע החדש שיצרנו עבורו (מסוג (Client ומתחיל בשבילו את המשחק.

לוגיקת המשחק(GameLogic):

זאת מחלקה סטטית, שתפקידה הוא לנהל את זרימת המשחק, בהתאם לחוקי הפורמט.

היא מקבלת כפרמטר מופע של המחלקה Client ומריצה עבורו את כל שלבי המשחק.

בנק הודעות(StringDB):

קובץ שמכיל בתוכו dictionary שמכיל בתוכו את כל המלל שאנחנו צריכים עבור המשחק(שאלות, תשובות, הודעת פתיחה וכו').

**:FrontEnd**

חשוב לציין שהעברנו את כל הלוגיקה לצד של הbackend, כך שבfrontend נשאר רק לדאוג להצגה הוויזואלית ולתקשורת מול הserver.

**:CLI**

clientSocket:

ראשית, מתבצע ניסיון להתחבר לשרת במידה והוא נכשל(בין אם השרת לא פעיל או שהוא מלא) מודפסת הודעת שגיאה מתאימה ומתבצעת יציאה מהתוכנית.

במידה והלקוח כן הצליח להתחבר, המשחק מתחיל והצדדים מתקשרים ביניהם(אחד אחד כמו "פינג פונג"). במידה והשרת החזיר את הודעת הניתוק, הלקוח סוגר את הsocket ויוצא מהתוכנית.

**UI((extra:**

חשבנו שזה יהיה מגניב ליישם ממשק גרפי לפרויקט. הוספנו קובץ בשם gameUI.exe שמפעיל את אפליקציה. ממשנו את הממשק הגרפי בלי לשנות את פרוטוקול התקשורת מול השרת, כך שלקוחות יכולים להתחבר מאיזה ממשק שהם בוחרים במקביל(כל עוד לא עוקפים את הכמות המקסימלית בסה"כ).

**הסבר על פתיחת socket:**

ראשית, הגדרנו את המשתנים PORT וIP. את הPORT קבענו ל5050 ואת הIP חילצנו בעזרת הפקודות: socket.gethostbyname(socket.gethostname()).

לאחר מכן, יצרנו מופע של המחלקה socket ע"י הפקודה הבאה:

socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

הפרמטר הראשון socket.AF\_INET מגדיר את השימוש במשפחת הכתובות IPV4 והפרמטר השני socket.SOCK\_STREAM מגדיר את הפרוטוקול שבו אנו רוצים להשתמש – TCP.

לאחר מכן, נקשר את החיבור שהגדרנו ל(IP, PORT) על ידי הפקודה:

socket.bind((IP, PORT))

כדי לשלוח ולקבל הודעות השתמשנו בפקודות:

socket.send(MSG), socket.recv(MAX\_SIZE)

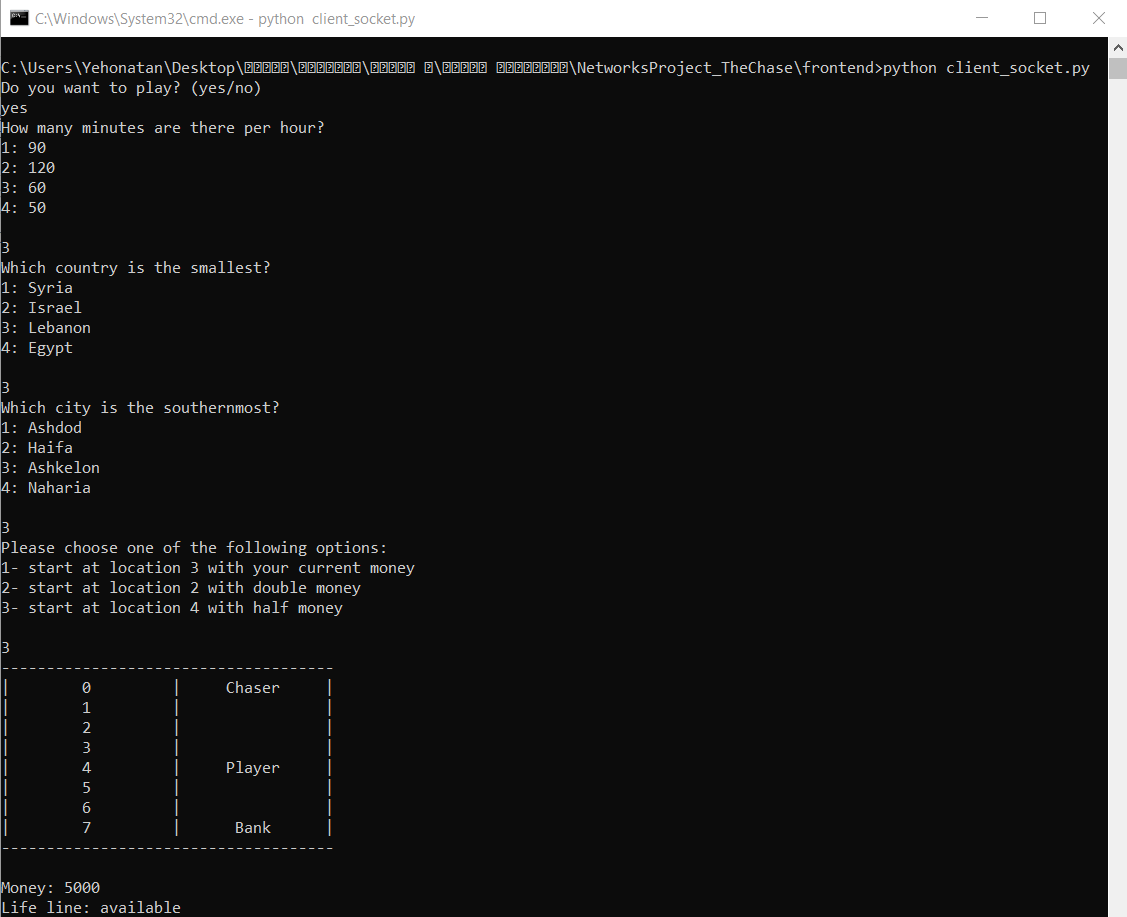
כדי לפענח את ההודעה בחזרה לstring השתמשנו במתודה .decode(‘utf-8’)

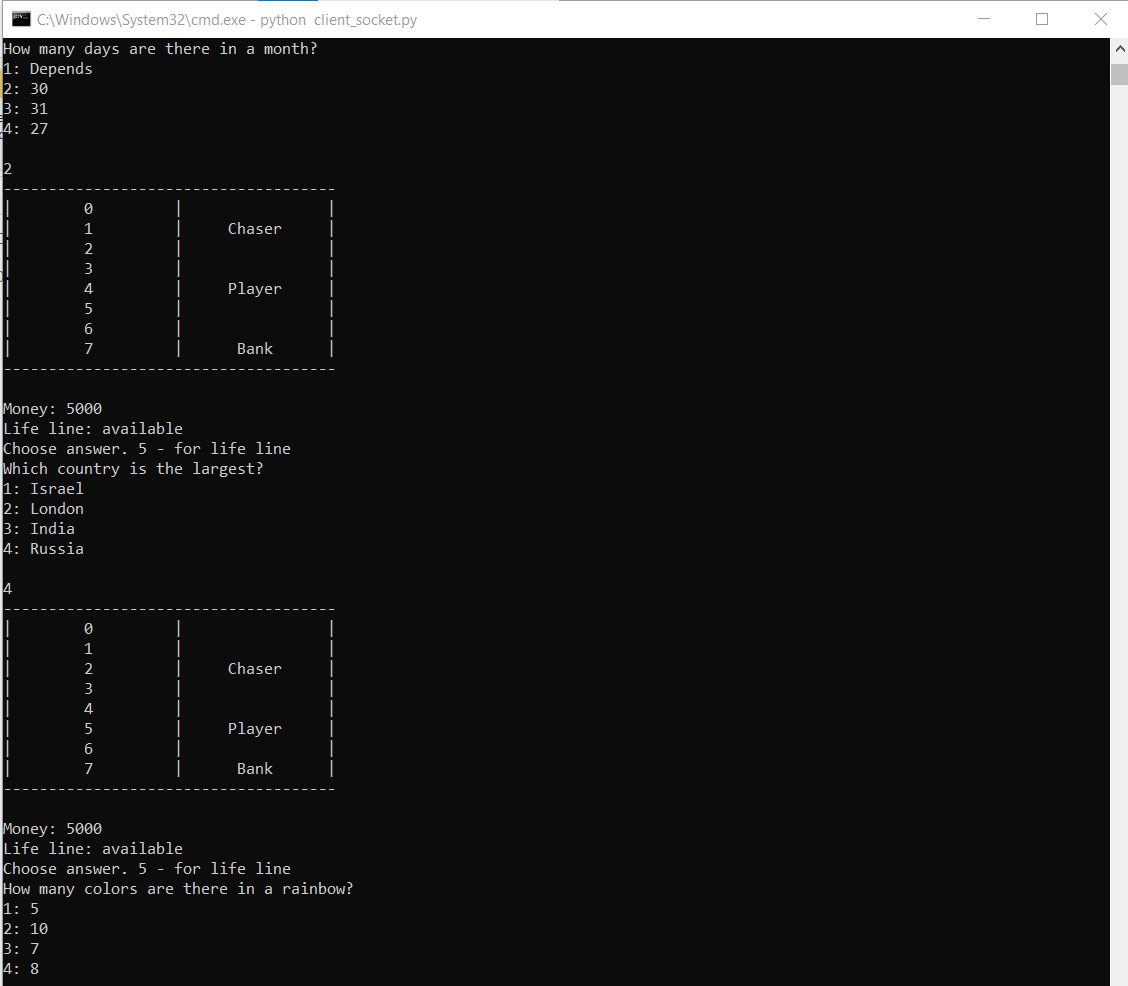
בסוף המשחק סגרנו את הsocket בעזרת המתודה close().

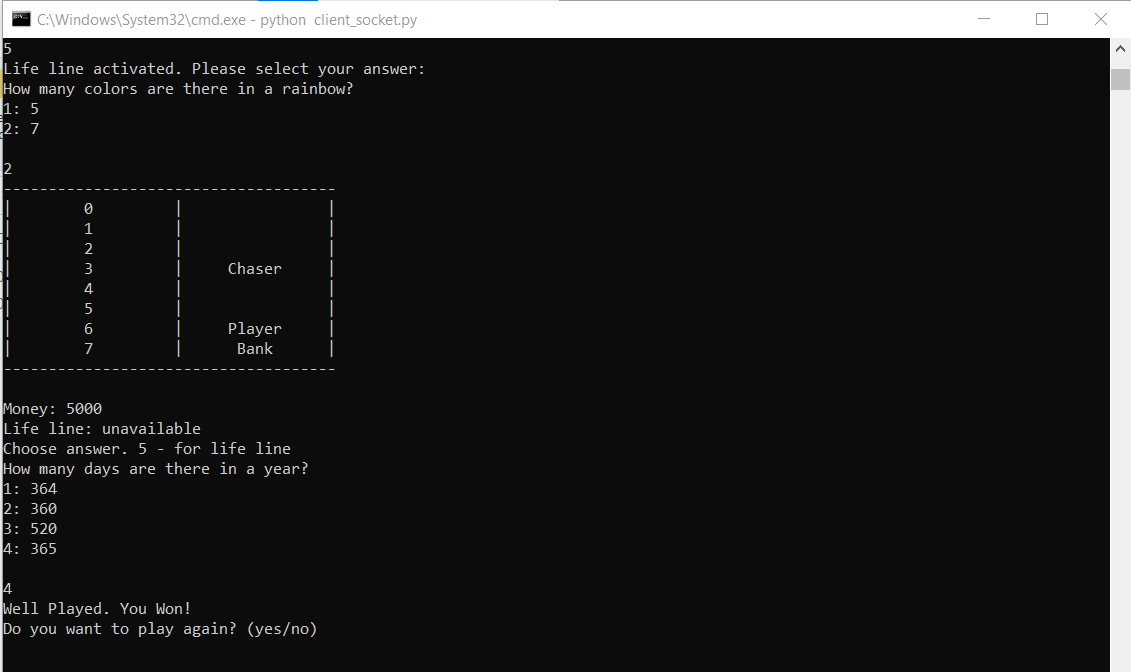
**צילומי מסך:**

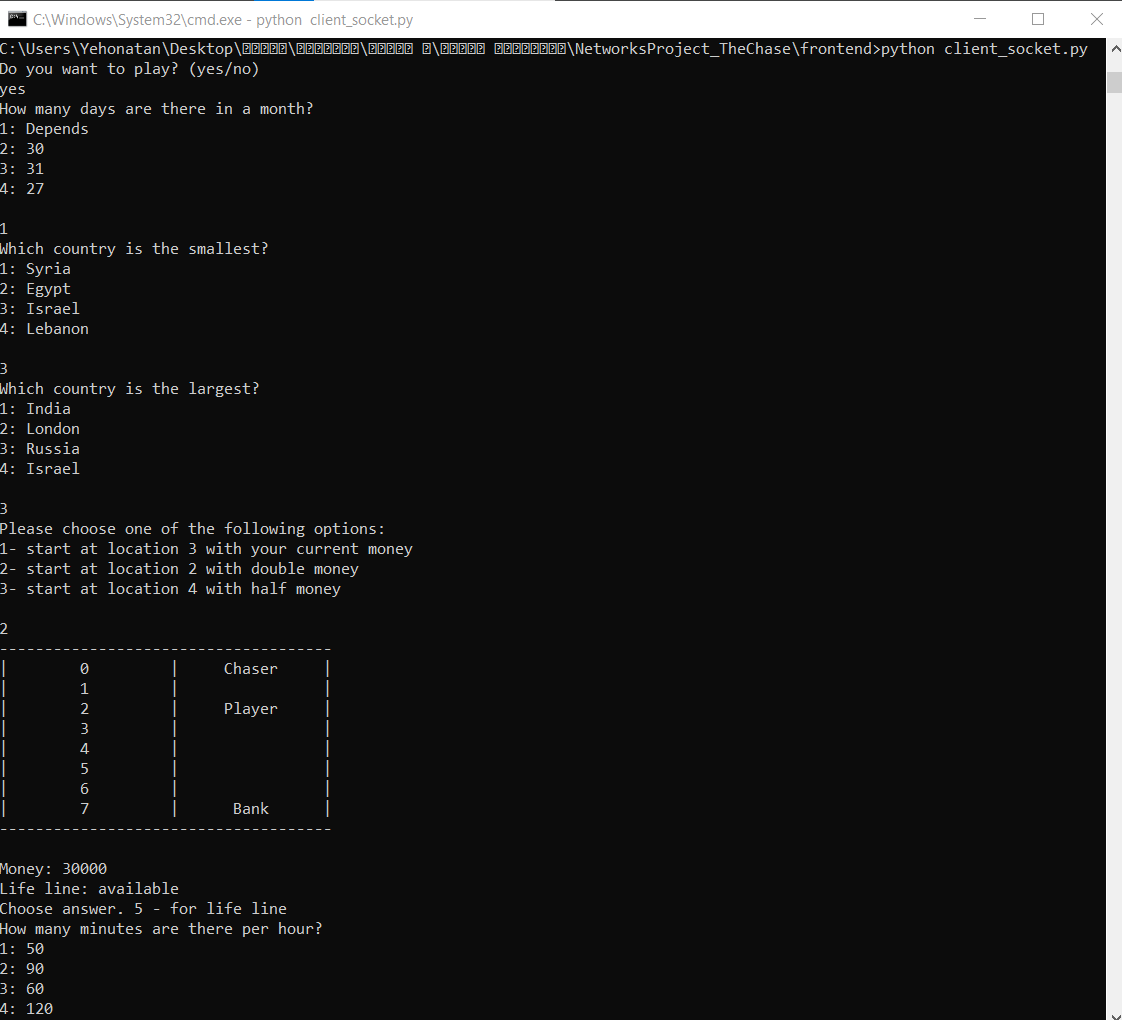
**CLI:**

ניצחון:







הפסד:

