

**Find Best Route Game**











שם התלמיד: צחי עקד

ת.ז: 326543584

שם המנחה: יוסי זהבי

שם החלופה: הגנת סייבר

תאריך הגשה: 12.5.2022

שם בית הספר: תיכון ע"ש חיים הרצוג כפר סבא

תוכן עניינים

[מבוא 3](#_Toc103451339)

[מבנה/ארכיטקטורה של הפרויקט 9](#_Toc103451340)

[מדריך למשתמש 18](#_Toc103451341)

[בסיס הנתונים 31](#_Toc103451342)

[מדריך למפתח 33](#_Toc103451343)

[סיכום אישי / רפלקציה 53](#_Toc103451344)

[ביבליוגרפיה 54](#_Toc103451345)

[נספחים 55](#_Toc103451346)

# מבוא

תיאור תכולת הספר: הספר יסביר על הפרויקט שלי שמתבסס על הרעיון של ניהול מידע וזיכרון על פני כמה מחשבים, שממומש במשחק שיצרתי. בספר זה אסביר על הרקע לפרויקט, הסיבות לבחירתו, האתגרים בביצועו, מבנה/ארכיטקטורת פרויקט, מדריך למשתמש, בסיס נתונים, מדריך למפתח, סיכום אישי/רפלקציה, ביבליוגרפיה ונספחים.

הרקע לפרויקט:

עשינו תרגיל כיתתי שנקרא עיבוד מבוזר ובמסגרתו היינו צריכים למצוא סיסמה בהינתן hash שלה. כמות האפשרויות הייתה ,(אורך מקסימלי 10 והסיסמה מספרות 0-9 בלבד). כמות אפשרויות זו לא מספיקה למחשב אחד/תיקח יחסית הרבה זמן אז עשינו את התרגיל על הרבה מחשבים, בכל אחד מהם מורץ לקוח שמבצע משימות(בודק תחום מסוים של מספרים) ומחשב אחד שהוא שרת ששולח לכל לקוח משימות בטווח מספרים שונה, עד שהוא מקבל את הסיסמה. תרגיל זה נתן לי מוטיבציה להשתמש באותו רעיון של שימוש בכמה מחשבים בשביל הפרויקט שלי. לאחר מחקר מצאתי שב-waze משתמשים גם ברעיון של שימוש בכמה מחשבים, אבל למטרת ניהול זיכרון ולא משאבים. החלטתי לעשות פרויקט דומה ולשלב בין הרעיונות של ניהול הזיכרון והעיבוד המבוזר אך גם שונה ולא לעשות waze משל עצמי אלא לשלב רעיון של משחק. המצאתי משחק שבו בכל שלב יש כמות של נקודות שמיוצגות באמצעות בתים והמטרה לבחור סדר מסוים שיעבור בכולם בדרך היעילה ביותר. במשחק יהיה מידע רחב שיש לשמור על כמה מחשבים וגם חלוקת של משימה לשנות אותו לכמה מחשבים, דברים שמתאים לרעיונות גם של לזיכרון מבוזר וגם לעיבוד מבוזר.

מוטיבציה- הרעיון של משחק יתן לי מוטיבציה לעבוד על הפרויקט, מלהיב אותי לראות את המשחק עובד כמו שצריך במיוחד כשיש שימוש בכמה מחשבים כדי לשמור את המידע/זיכרון בשביל האפשרויות השונות למפת המשחק ולשנות אותו. גם המחשבה שאני יוצר משחק מיוחד משל עצמי שאף אחד לא עשה דבר כזה לפני נותנת מוטיבציה לעשות את הפרויקט הכי טוב שאני יכול. גם הרעיון שהפרויקט שלי ישמור מידע רגיש(סיסמאות של משתמשים) באמצעות אבטחת מידע מתאימה נותן מוטיבציה כי אף פעם לא מימשתי אבטחה כזו בפרויקט רחב.

הפרויקט עונה על הצורך שאי אפשר לנהל מידע שמתעדכן בזמן אמת בכמות גדולה מידי על מחשב אחד. חובה להשתמש בכמה מחשבים, כך שכל אחד ישמור חלק אחר של המידע ויבצע משימה אחרת. הפרויקט בא לתת פתרון של שימוש בזיכרון ומשאבים של כמה מחשבים. אממש פתרון זה במשחק שלי, אבל זו רק דוגמה לאיך לעשות זאת ואפשר להשתמש בפתרון בכל צורה כלשהי.

תהליך המחקר:

מצאתי שני מוצרים שמשתמשים ברעיון דומה לזה של הפרויקט שלי, כלומר שומרים זיכרון משתנה שגדול מידי בשביל מחשב אחד באופן מבוזר(בכמה מחשבים).

תמונה שמכילה מקורה

התיאור נוצר באופן אוטומטי

הצגת הפתרונות שכבר קיימים לבעיה:

פתרון של Waze:

ניהול הזיכרון במספר רב של מחשבים, כל מחשב אחראי על חלק אחר בעולם ועדכוני התנועה שבחלק זה נשלחים אליו. כאשר נוסעים ממקום מסוים למקום אחר ועוברים לחלק/אזור אחר בעולם הלקוח יעבור לתקשר עם המחשב שמתאים לאזור שלו, כל שרת ב-Waze יודע מהם השרתים של כל אזור שקרוב אליו, וכך הוא מעביר את המשתמש לשרת המתאים.



פתרון במשחק פורטל (2):

גם במשחק זה יש את הבעיה שאי אפשר לשמור את כל הנתונים של כל השלבים והשחקנים שבהם במחשב אחד, הבעיה נפתרת באמצעות ניהול זיכרון מבוזר, בכל פעם שעוברים מפורטל אחד לאחר או שעוברים שלב אנחנו עוברים לתקשר עם מחשב אחר שיש בו את הנתונים לשלב הבא(השרת שהלקוח מתקשר איתו מעביר אותו לשרת הבא).

תמונה שמכילה טקסט, מקורה

התיאור נוצר באופן אוטומטי

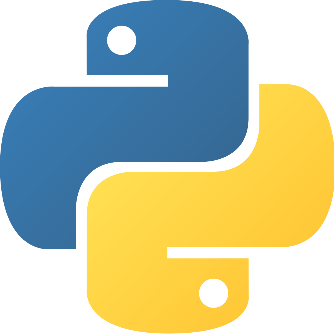
השוותי בין הפרויקט שלי למוצרים האחרים. הנתונים מסוכמים בטבלה, אפשר לראות את החידושים ואת הדברים שכבר יש גם במוצרים אחרים.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| מוצר- קריטריון | שמירת זיכרון משתנה בכמה מחשבים עם שרת ראשי שמנהל ביניהם | שמירת מידע אישי והתקדמות של כל משתמש | אבטחת מידע | מבנה נתונים שמציג שיאים ונגיש לכל המשתמשים | שימוש באקראיות ובכך יצירה של מספר בסדר גודל רב של מצבים אפשריים |
| משחק מבוזר  Find Best Route Game |  |  |  |  |  |
| Waze  תמונה שמכילה טקסט, אוסף תמונות  התיאור נוצר באופן אוטומטי |  | תמונה שמכילה טקסט, שלט  התיאור נוצר באופן אוטומטי |  | תמונה שמכילה טקסט, שלט  התיאור נוצר באופן אוטומטי | תמונה שמכילה טקסט, שלט  התיאור נוצר באופן אוטומטי |
| פורטל  Portal 2 - משחקים וקונסולות – כללי - YourGame |  |  |  | תמונה שמכילה טקסט, שלט  התיאור נוצר באופן אוטומטי | תמונה שמכילה טקסט, שלט  התיאור נוצר באופן אוטומטי |

ניתן לראות בטבלה שבפרויקט שלי כמו במוצרים קיימים יש שימוש בזיכרון מבוזר וגם אבטחת מידע לכל משתמש- שימוש בפרוטוקולי אבטחה כדי לוודא שגורמים שונים לא יוכלו להגיע למידע רגיש- הסיסמאות של המשתמשים(וגם אם הם מצליחים להגיע למבנה הנתונים של הסיסמאות זה לא לגמרי יספיק להם כי השרת שומר את ה-hash של הסיסמה עם salt, יפורט בהמשך). החידושים העיקריים הם שבפרויקט שלי מבנה נתונים ששומר שיאי משתמשים ואקראיות- תוכנת המשחק יכולה לייצר שלבים אקראיים בעצמה דבר שלא מסתמך על מבנה נתונים קבוע ומוחלט ונותן מרחב גדול של אפשרויות למשחק.

סקירת ספרות:

ישנם כמה כלים חשובים בעולם התוכנה שבאמצעותם מתקיים הפרויקט, אסביר על כל אחד ומדוע הוא מתאים לפרויקט שלי.



השפה פיתון: פיתון היא שפה בעלת קיצורים רבים ונוחות משתמש על חשבון זהירות שהמשתמש צריך לנקוט בעצמו. בנוסף, קל לייבא קוד מספריות חיצוניות- לא אממש אותן בעצמי כי הן לא עיקר הפרויקט אלא רק אשתמש בהן בהתאם. שתי התכונות האלו הופכות את פיתון לשפה יעילה ומהירה מאוד למתכנתים בעלי ידע ונסיון, דבר שמתאים לעבודה על פרויקט כה רחב.



גרפיקה- הפרויקט כאמור נכתב בפיתון והגרפיקה שלו מיושמת על ידי pygame, ממשק גרפי נוח של השפה שבעזרתו אפשר להציג מידע בקלות על המסך, בין אם המידע הוא תמונה ספציפית או צבעים שרוצים להציג על פיקסלים מסוימים במסך. בנוסף בעזרת ממשק זה אפשר להגיב לאירועים של המשתמש כמו לחיצה על העכבר.

תקשורת: סוקטים ופרוטוקול TCP- סוקט הוא נקודת קצה של תקשורת מידע. לכל סוקט כתובת הכוללת כתובת ip של המחשב ומספר פורט. במקרה שלנו אנחנו עובדים עם הפרוטוקול TCP שנועד ליצור חיבור אמין בין שני מחשבים וזקוק לכתובת סוקט המקור וכתובת סוקט היעד בכל הודעה. הפרוטקול מיישם את מטרתו בעזרת three way handshake, לחיצת יד משולשת כדי לפתוח חיבור מקושר בין שני מחשבים. היא מתבצעת בכך שמחשב אחד שולח הודעת synchronize (בקשת פתיחת תקשורת) למחשב השני, המחשב השני ישלח synchronize, acknowledgement, בקשה לתקשר עם המחשב הראשון ואישור בקשתו, ואז המחשב הראשון ישלח הודעת acknowledgement, אישור בקשת המחשב השני.

תמונה שמכילה טקסט, אלקטרוניקה, צילום מסך, מחשב

התיאור נוצר באופן אוטומטי

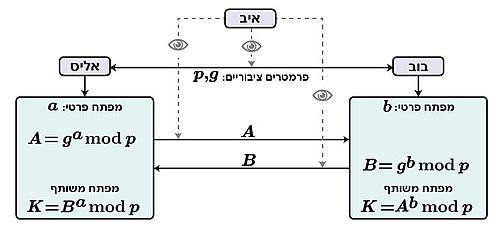
תמונה שמכילה שולחן

התיאור נוצר באופן אוטומטי

עם זאת, כדי להמשיך את העברת המידע בשלמותו, בסדרו הנכון ובאמינות יש לציין בכל הודעה מספר סידורי sequence number כדי שהצד המקבל יבין אם הוא קיבל את המידע בסדר הנכון, acknowleement כדי לוודא שהחבילה אכן הגיעה ליעד(אחרת שולחים שוב), ו-checksum פונקציה המחושבת לפי גוף ההודעה ובעזרת התוצאה שלה ניתן לדעת אם החבילה הגיעה בשלמותה.

בתמונה שדות הפרוטוקול TCP, אפשר לראות את החשובים שציינו.

אבטחת מידע:

פרוטוקול דיפי הלמן- מאפשר לשתי יחידות ברשת, גם לאלו שלא דברו קודם לעולם, להעביר ברשת סוד כלשהו בלי שגורם שלישי יצליח להבין מה. הפרוטוקול עובד בצורה הבא: כל אחד מהצדדים בוחר מפתח פרטי a בצד אחד, b בצד שני, ואחד הצדדים בוחר שני פרמטרים ציבוריים p,g שיעביר לשני. צד א' שולח לצד ב' את התוצאה של שהיא A, וצד ב' שולח לא' את התוצאה של שהיא B. לאחר מכן לשני הצדדים יש סוד משותף. צד א' מחשב אותו באמצעות- וצד ב': . 

במקרה שלנו אשתמש בסוד המשותף כדי ליצור מפתח סימטרי בשביל המשך תקשורת סימטרית בעזרת פרוטוקול הצפנה סימטרי, לדוגמה AES.

הצפנת AES(Advanced Encryption Method)- הצפנה סימטרית שכדי לבצע אותה צריך מפתח סימטרי שסוכם לפני כן אצל שני הצדדים (אותו אפשר להשיג בעזרת דיפי הלמן). בעזרת אותו מפתח ניתן להצפין הודעות וגם לפענח אותן.

פונקציות גיבוב(hash)- פונקציית גיבוב מקבלת קלט של מחרוזת כלשהי(ללא אורך או תווים מסוימים קבועים) ומחזירה מחרוזת באורך קבוע. לדוגמה הפונקציות: md5, sha1, sha256. אחד השימושים החשובים ביותר של פונקציות אלו(אתו איישם גם בפרויקט) הוא שמירת ה-hash של מידע רגיש בצד השרת. הסיבה לכך היא שברגע שגורם צד שלישי מגיע לקובץ/מבנה הנתונים שבו יש את המידע הרגיש של המשתמשים, אם המידע יהיה כתוב באופן ברור הפורץ יוכל בקלות למצוא אותו ולזהות אותו. לעומת זאת, ברגע ששמור הגיבוב של כל יחידת מידע ומספר האפשרויות למידע הוא גדול מאוד, הפורץ לא יוכל לזהות את המידע, אפילו שהצליח לחדור למבנה הנתונים של השרת. בחרתי בפונקציה sha256()- sha) זה קיצור של (secure hash algorithm המקבלת מחרוזת בינארית ומחזירה גיבוב של 256 ביטים שאפשר להפוך למחרוזת של 64 ספרות הקסדצימליות ואותה לשמור בבסיס הנתונים.

Salted hash- שיטת גיבוב שמקשה עוד יותר על האקרים שפרצו לבסיס הנתונים. לפני שעושים hash למחרוזת הבינארית של הסיסמה מוסיפים לה salt אקראי- מחרוזת של 32 ספרות הקסדצימליות במקרה שלי. בנוסף לשמירת תוצאת הגיבוב שומרים גם את ה-salt בבסיס הנתונים, כדי שיהיה אפשר להתאים סיסמה ו-salt לתוצאת הגיבוב. השימוש ב-salted hash מונע מאותו גיבוב סיסמה להופיע כמה פעמים בבסיס הנתונים. אם היינו עושים רק hash, היינו מקבלים תוצאות זהות בשביל סיסמאות זהות, פרצה להאקרים. לדוגמה: האקר יכול להכניס סיסמה פשוטה שהרבה אנשים משתמשים בה ואז להסתכל בבסיס הנתונים. ברגע שיש רק hash הוא יודע שכל מי שעם אותה תוצאת hash בבסיס הנתונים הוא גם בעל אותה סיסמה כמו שלו. לעומת זאת, כאשר משתמשים ב-salted hash מוסיפים לכל סיסמה עוד תווים אקראיים לפני ה-hash שלה, דבר שגורם לכך שלסיסמאות זהות יהיו תוצאות salted hash שונות. במקרה זה האקר לא יכול לראות בקלות למי יש סיסמה כמו שלו לדוגמה.

תחום מערכות הפעלה- ניהול מספר מרובה של לקוחות multiclient בצד השרת באמצעות multithreading. -Thread הוא תהליך קטן שנועד לבצע משימה מסוימת. כל תהליך יכול להרים כמה threads ולתת להם לעבוד "במקביל". הדבר עוזר מאוד בשביל לשפר את יעילות העבודה והזמן שיקח להשלים משימות שונות, שכן הthreads יכולים לעבוד באופן אסינכרוני- ברגע שאחד מחכה למידע/קלט thread אחר יכול לעבוד בזמן זה וככה יש ניצול הרבה יותר טוב של המעבד והפחתה של ה"זמן המת" שלו- הזמן בו הוא לא מריץ שום תהליך.

ניהול כמות רחבה של זיכרון משתנה בכמה מחשבים, כך שכל מחשב אחראי על חלק אחר של הזיכרון. ניצול משאבים של כמה מחשבים, לכל אחד ניתנת משימה אחרת. כמובן שיש מחשב ראשי שמנהל את המחשבים הנוספים.

בסיס הנתונים:

שפת SQL- שפה נוחה לעבודה עם מבני נתונים שכוללים טבלאות, מאפשרת מגוון פעולות על הטבלאות, לדוגמה שינוי הטבלאות או הבאת מידע לפי תנאים ספציפיים מהטבלה. אשתמש בשפה זו כדי לעבוד עם הטבלה של המשתמשים והטבלה של השיאים בפרויקט שלי.

Sqlite3- ממשק פשוט למימוש השפה sql ושמירת נתונים בהתאם לצורך, אפשר להשתמש בו גם בשפת פיתון- דבר שמתאים לפרויקט שלי.

אתגרים מרכזיים- קושי בניהול מידע על כמה מחשבים במקום על מחשב אחד, קושי במימוש פרוטוקולי אבטחת המידע בעצמי, יצור אלגוריתם נכון של התוכנה המשלב אקראיות אך עדיין מתאים למאפייני כל שלב, שבעזרתו תוכל לייצר עוד ועוד שלבים שיהיו שונים זה מזה, עבודה על פרויקט גדול מאוד, קושי בתכנון פרויקט בצורה יעילה שבאמת עוזרת.



# מבנה/ארכיטקטורה של הפרויקט

הפתרון המוצע בפרויקט שלי הוא כאמור שימוש בכמה מחשבים שרתים שכל אחד מהם ישמור חלק אחר של זיכרון דינאמי(במקרה שלי כל אחד שומר אזור שונה של המפה). יש שרת אחד מרכזי שינהל את השרתים המשניים וידע להעביר את המידע הרצוי ללקוח בהתאם.

פתרון זה בא לתת המחשה למקרים שבהם אי אפשר לשמור את כל המידע/זיכרון המשתנה על מחשב אחד ולכן צריך להשתמש בכמה מחשבים.

הסבר כללי על תהליך ההתקשרות:

הלקוחות מתקשרים באופן ישיר רק עם השרת הראשי. שרת זה יכול לספק ללקוח את כל המידע שיצטרך, ללא צורך שהלקוח יתקשר עם גורם אחר(השרת הראשי עושה זאת עבורו). שרת זה בעל גישה לבסיס הנתונים של המשתמשים ושל השיאים, אך המידע על מפת המשחק המשתנה לא נמצא אצלו אלא אצל השרתים המשניים. השרת יבקש את המידע הנדרש בהתאם לשלב, יקבל אותו מהשרתים המשניים ויעביר אותו ללקוח. כך נוצר מצב שהלקוח מקבל מידע על תמונה מבוזרת על פני כמה מחשבים, אך הוא מתקשר רק עם יחידה אחת- השרת הראשי.

היחידות שבפרויקט:

יחידת הלקוח- מטרתה להציג את המידע באופן ידידותי למשתמש כדי שיוכל להשתמש במוצר בנוחות. בשביל לעשות זאת הלקוח חייב כמובן לתקשר עם השרת.

הלקוח יקבל כמה קלטים מהשרת כגון: התחברות או יצירת משתמש הצליחה/כשלה, טבלת השיאים במשחק ומפת כל כל שלב.

הפלט של הלקוח- המידע שהוא שולח לשרת בהתאם למה שהמשתמש מבקש, וגם פלט מסך, יציג את המסכים השונים למשתמש.

שרת ראשי- תפקידו לנהל את כל מהלך התקשורת עם הלקוח ולהעביר לו את המידע שהוא צריך. בשביל להתחבר/ליצור משתמש חדש או לראות את השיאים במשחק השרת יגש לבסיס הנתונים ויביא את הנתון הרלוונטי לכל בקשה. בשביל שהמשתמש יתחיל משחק תחרותי השרת יקבל מתת השרתים את החלקים של מפת המשחק ויעביר אותם ללקוח.

הקלטים שלו יהיו:

1. ממבנה הנתונים- יקבל את ה-hash של הסיסמאות ושמות המשתמשים או את טבלת השיאים.

2. מתתי השרתים יקבל את האזורים של מפת המשחק.

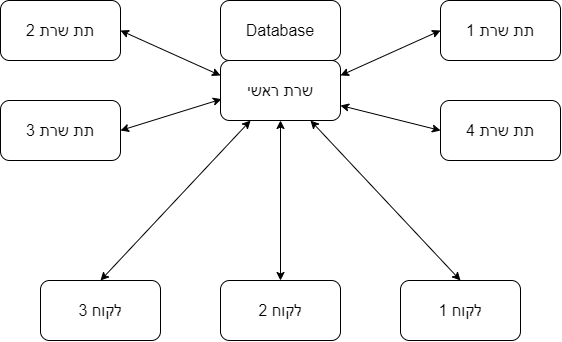
3. בקשות הלקוח.

הפלטים שלו יהיו התשובות לבקשות והשינויים בבסיס הנתונים.

שרתי משנה/תת שרתים- ישנם 4 תתי שרתים. כל תת שרת אחראי על כמה אפשרויות לאזורים במפת המשחק. בכל אפשרות יהיה מספר שונה של בתים. תת השרת ישנה את האזורים באקראיות כל הזמן. ברגע שהשרת הראשי יבקש מתת השרת אזור מסוים הוא ישלח לו אותו ויעדכן אותו שוב(ישנה אותו באקראיות).

קלט- בקשה לאזור מפה מהשרת הראשי. פלט – שליחת האזור עצמו לשרת הראשי.

תרשים היחידות בפרויקט, הקישור בין השרתים ללקוחות ובסיס הנתונים:

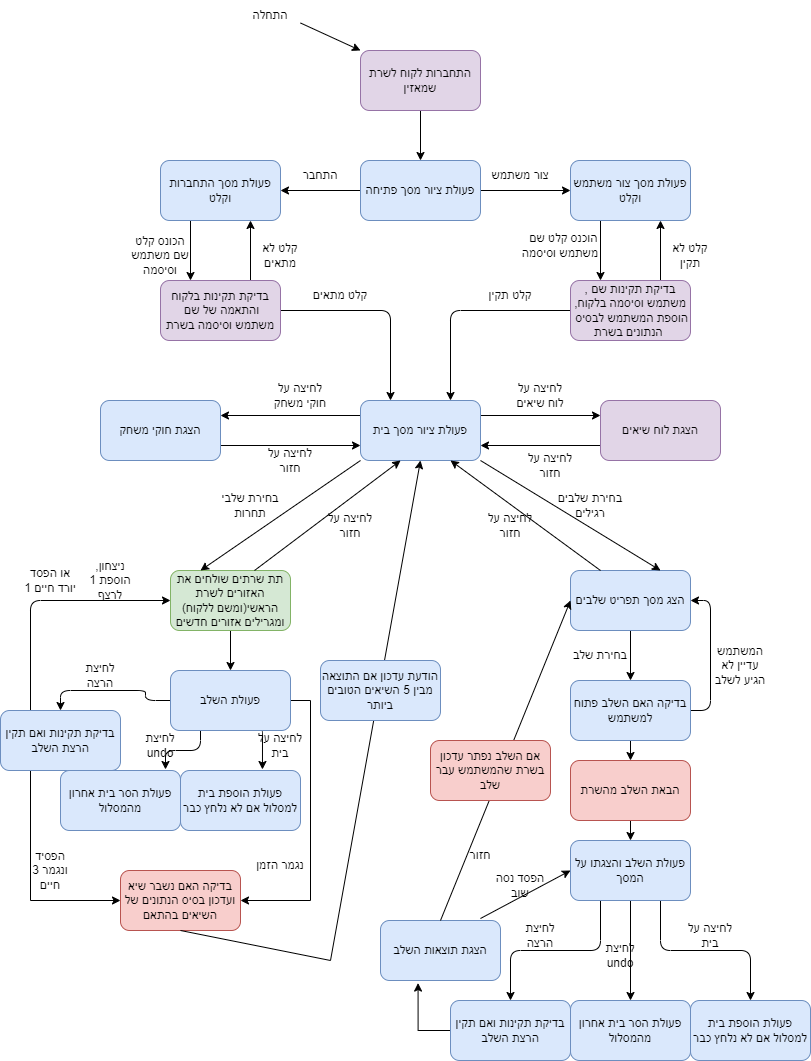


אפשר לראות שהלקוחות מתקשרים רק עם השרת הראשי.

השרת הראשי מתקשר בו זמנית עם מספר לקוחות ועם 4 תתי שרתים.

השרת הראשי בעל גישה לבסיס הנתונים.

כל אחד מתת השרתים מתקשר עם השרת הראשי בלבד.



ארכיטקטורת Top Down Level Design:

מקרא צבעים:

כחול: לקוח

אדום: שרת

סגול: שרת + לקוח

ירוק: תת שרתים->שרת->ללקוח

בארכיטקטורת Top Down Level Design אפשר לראות את הקריאות לפעולות העיקריות שיהיו בפרויקט ולהבין איך הוא עובד, היא כוללת תיאור הלולאה הראשית וגם שלב ההתחברות/יצירת משתמש שקודם לה.

ארכיטקטורת רשת:

אשתמש בפרוטוקול TCP- by size.

בשיטה זו משתמשים בפרוטוקול TCP שעליו הסברתי בסקירת הספרות ומוסיפים בקוד שדה שיכלול את גודל ההודעה (מכאן השם by size). לשדה זה יש גודל קבוע, אם גודל ההודעה קטן ממספר הספרות הקבוע של שדה זה, נוסיף ריפוד של אפסים לפני המספר. דוגמה פשוטה: נניח שגודל השדה הקבוע הוא 4 אבל גודל ההודעה הוא 26 שדה זה יהיה 0026.

שדות ההודעה(חוץ מהשדה של גודל ההודעה):

1. יהיה שדה שיציין את סוג ההודעה והוא יהיה בגודל קבוע.

2. שדה של גוף ההודעה בגודל משתנה(אבל יהיה אפשר לקבל אותו באופן נוח בעזרת שדה גודל ההודעה). כל שדה בגוף זה יהיה מופרד באמצעות התו "\n"(ירידת שורה).

אזורים הדורשים אבטחת מידע:

בתחילת התקשורת ייווצר המפתח הסימטרי באמצעות פרוטוקול דיפי הלמן.

כל אחת מההודעות שכוללת שם משתמש וסיסמה תהיה מוצפנת באמצעותAES עם המפתח הסימטרי שנוצר בתחילת התקשורת.

מאגרי המידע של חשבונות המשתמשים ישמרו בטבלת SQL(יפורט בחלק של בסיס הנתונים). בכל פעם שהשרת יקבל הודעה שבגופה שם משתמש וסיסמה מהלקוח הוא יפענח את המידע על פי הצופן ולאחר מכן יבדוק אם הגיבוב של המידע תואם לתוצאת גיבוב בטבלת ה-hash של המשתמשים(במקרה של התחברות), או יוסיף אותו לטבלה אם שם משתמש לא קיים כבר (במקרה של יצירת משתמש).

ישנו עוד פרוטוקול קטן בשביל שליחה וקבלה של הודעות מוצפנות. כאשר לוקחים הודעה ומצפינים אותה לפי הפקטה של AES מקבלים את ה-nonce משתנה מהודעה להודעה, cipher\_text- ההודעה עצמה מוצפנת ו-tag- קוד הודעה בכדי לוודא שההודעה אמיתית ולא נשלחה ממקור אחר. כדי לשלוח את שלושת המשתנים האלו שבגדלים לא קבועים החלטתי להוסיף לפני כל אחד מהם 4 תווים שייצגו את גודל השדה שבא אחריהם בדומה לפרוטוקול by\_size של tcp, רק שכאן הגודל מצוין לפני כל חלק של הודעה במקום לפני כל ההודעה עצמה, כך שיהיה אפשר להבדיל בין החלקים שלה.

טבלה המסכמת את ההודעות הקיימות בפרוטוקול.

בכל פעם שכתוב מלקוח לשרת או מהשרת ללקוח הכוונה לשרת הראשי כמובן, הלקוח לא מתקשר עם תת השרתים אלא שכל המידע עובר דרך השרת הראשי.

לא ציינתי את הטיפוסים של גוף ההודעה כי הוא תמיד מחרוזת בינארית(bytes).

|  |  |  |  |
| --- | --- | --- | --- |
| סוג ההודעה | כיוון | מטרה | גוף ההודעה(אם יש) |
| PUBLICP | מהשרת ללקוח | לשתף את הלקוח בפרמטרים הפומביים בשביל פרוטוקול דיפי הלמן. | הפרמטרים הציבוריים p, g |
| RESULTA | מהלקוח לשרת | לשתף את השרת בתוצאה שיצאה ללקוח בדיפי הלמן. | A, התוצאה של  . |
| RESULTB | מהשרת ללקוח | לשתף את הלקוח בתוצאה שיצאה לשרת בדיפי הלמן. | B, התוצאה של  . |
| CLIENT\_ | מהלקוח לשרת | להודיע לשרת הראשי שהתחבר לקוח. | ריק |
| HELPER\_ | מתת שרת לשרת הראשי. | להודיע לשרת הראשי שהתחבר שרת משני(ולא עוד לקוח). | ריק |
| YOUR\_ID | מהשרת הראשי לתת שרת. | להודיע לשרת המשני על המספר הסידורי שלו. | המספר הסידורי |
| EXSUSER | מהלקוח לשרת. | לשלוח בקשת התחברות למשתמש קיים. | שם משתמש וסיסמה של משתמש(המידע מוצפן). |
| EXSUSER | מהשרת ללקוח. | לשלוח אישור התחברות למשתמש קיים ואת התקדמות המשתמש. | מספר השלב שהמשתמש הגיע אליו. |
| ERREXS\_ | מהשרת ללקוח. | להודיע על שגיאה, שם המשתמש לא קיים או שהסיסמה לא נכונה. | ריק. |
| NEWUSER | מהלקוח לשרת. | בקשת יצירת משתמש חדש. | שם משתמש וסיסמה חדשים(המידע מוצפן). |
| NEWUSER | מהשרת ללקוח. | אישור יצירת משתמש חדש. | ריק. |
| ERRNEW\_ | מהשרת ללקוח. | שגיאה- שם המשתמש כבר קיים. | ריק. |
| RECORDS | מהלקוח לשרת. | בקשת הטבלה של השיאים מקטגורית זמן מסוימות. | קטגורית הזמן של השיאים(3 או 5 או 10 דקות) |
| RECORDS | מהשרת ללקוח. | החזרת טבלת השיאים של קטגורית זמן מסוימת. | טבלת השיאים של קטגוריה מסוימת, כל שיא בתוך סוגריים, בין כל שיא רווח שורה '/n' ובין כל רשומה בשיא פסיק ורווח ' ,'. |
| GETSTAG | מהלקוח לשרת. | בקשת המפה של שלב מסוים. | מספר השלב המבוקש. |
| RETSTAG | מהשרת ללקוח. | לשלוח מפת שלב רגיל. | מפת השלב המבוקש. |
| GETCOMP | מהלקוח לשרת. | בקשת המפה לשלב תחרותי. | רמת קושי של השלב. |
| RETCOMP | מהשרת ללקוח. | לשלוח מפת שלב תחרותי. | מפת השלב המבוקש. |
| BEATSTA | מהלקוח לשרת. | יידוע השרת שהלקוח עבר שלב מסוים, כדי שהשרת יוכל לעדכן את התקדמות המשתמש. | מספר השלב שהמשתמש עבר. |
| STATIME | מהלקוח לשרת. | שליחת תוצאה שהמשתמש עשה, כדי שהשרת יבדוק האם היא מבין 5 השיאים הטובים ביותר. | כמות השלבים שעבר וקטגורית הזמן. |
| GETAREA | מהשרת הראשי לתת שרת. | לבקש מתת שרת אזור של המפה. | כמה בתים באזור. |
| RETAREA | מתת שרת לשרת הראשי. | לשלוח לשרת הראשי אזור של המפה. | האזור עצמו. |
| STATIME | מהלקוח לשרת. | הודעה כמה שלבים המשתמש הצליח לפתור ובאיזו קטגורית זמן. | כמות השלבים שהמשתמש פתר והזמן שלקח לו. |
| NEWRECO | מהשרת ללקוח. | להודיע ללקוח שהתוצאה שהוא עשה מבין 5 השיאים הכי טובים לקטגורית הזמן שלו. | ריק. |
| NOTRECO | מהשרת ללקוח. | להודיע ללקוח שהתוצאה שהוא עשה לא מבין 5 השיאים הטובים ביותר לקטגורית הזמן שלו. | ריק |
| DISCONN | מהלקוח לשרת. | להודיע לשרת שהלקוח מתנתק(במקרה שהלקוח יוצא מחלון ה-pygame).  (\*הערה: אפשר גם לסגור את התהליך של הלקוח client.py והשרת יתפוס את השגיאה שהלקוח התנתק) | ריק. |

מודלים/ספריות חיצוניות:

socket:

נקודת קצה התקשורת. כדי לקיים את התקשורת נצטרך אחת כזו בכל אחד מהרכיבים שמשתתף בתקשורת.

socket.socket()- מגדיר נקודת קצה חדשה.

socket.bind((ip, port))- הגדרת כתובת ip ופורט של החיבור.

socket.listen()- פתיחת החיבור- האזנה לחיבורים חדשים.

socket.connect((ip, port))- התחברות לנקודת קצה שמאזינה.

socket.accept()- אישור חיבור חדש, מחזיר את החיבור וכתובת היעד.

Socket.recv()- קבלת מידע מהחיבור (תחזיר את המידע שהתקבל).

Socket.send()- שליחת מידע לחיבור (מקבלת את המידע שרוצים לשלוח).

:threading

ספריית תת תהליך/תהליך קטן/תהליך בן הנועדה כדי שנוכל לבצע כמה פעולות במקביל. לכל הבנים יש גישה למרחב הכתובות של התהליך הראשי האבא. בעזרת ספרייה זו ניתן להקים שרת של multiclient, בשביל כל לקוח מרימים thread שיתקשר איתו וכך אפשר לתקשר עם כמה לקוחות במקביל. ברגע ש-thread אחד לא עושה כלום thread אחר יכול לרוץ וברגע לסיים את מה שהוא צריך לעשות, אחר כך הthread הבא פועל וכך ניתנת הרגשה שקורות שתי פעולות בו זמנית(אם זה על מעבד אחד אז זה לא באמת מקביליות מוחלטת אבל זה לא מורגש כי המעבד מהיר מאוד ואדם לא יכול להרגיש בכך).

השתמשתי בספרייה:

1. בשרת הראשי: לכל לקוח שמתחבר מורם thread שמתקשר איתו, שרת multiclient.

2. בלקוח: הרמתי thread אחד למסך ו-thread נוסף לתקשורת כך שהמסך לא יתקע וירוץ כל הזמן גם כשהthread השני מתקשר עם השרת.

3. בתת שרת: באופן דומה עם הלקוח, גם כאן יש thread שמתקשר עם השרת רק שכאן במקום thread מסך יש thread זיכרון שאחראי לשנות את קבצי האזורים בכל שנייה.

Super(Thread, self).\_\_init\_\_() או threading.Thread.\_\_init\_\_(self) קורא לפעולה הבונה של האב. פעולה זו מגדירה thread חדש. (בתוך הפעולה הבונה של המחלקה היורשת מ-threading).

Threading.Lock()- מגדירה אובייקט נעילה חדש. מוגדר כגלובלי כך שכל thread יכול לגשת אליו(או פשוט להגדיר אותו כתכונה נעילה שזה מה שאני עשיתי).

Lock.aquire()- נסיון לתפוס את הנעילה. אם הנעילה נתפסה כבר אצל מישהו אחר הפעולה תחכה עד שהיא תשוחרר(ואם לא היא תתפוס אותה בעצמה).

Lock.release()- שחרור הנעילה, חשוב מאוד לא לשכוח אחרי acquire כי אז הנעילה תישאר תמיד אצל אותו thread ושאר הthreads לא יוכלו לרוץ איתה.

הרעיון של הנעילה היא לאפשר רק לאחד הthreads לפעול איתה בו זמנית. משתמשים בה בדרך כלל כשthread אחד צריך לחכות שthread אחר יסיים לבצע פעולה מסוימת לפני שהוא ממשיך לרוץ. או כשכמה threads רוצים לגשת למשאב משותף בו זמנית, באמצעות נעילה אפשר לוודא שרק אחד ניגש בכל פעם ולמנוע התנגשויות בין threads.

את המטרה הראשונה של הנעילה מימשתי בלקוח באמצעות משתנים גלובליים והפונקציה time.sleep() (ראו ספריית time כמה שורות מתחת) כי זה היה לי יותר נוח אבל בסך הכל זה עושה אותו דבר, thread אחד מחכה עד שthread אחר יסיים פעולה מסוימת.

את המטרה השנייה של הנעילה מימשתי בתת שרת, גם הthread של התקשורת קורא מקובץ(כדי לשלוח את המידע לשרת) וגם הthread של הזיכרון ניגש לקובץ כדי לשנות אותו. כדי למנוע התנגשויות בין השניים השתמשתי בנעילה, בכל פעם שthread רוצה לגשת לקובץ מסוים הוא צריך לתפוס את הנעילה בהתחלה וכשהוא מסיים הוא משחרר אותה.

time:

ספריית זמן. הפעולה היחידה שהשתמשתי בה היא:

time.sleep()- התכנית תחכה את מספר השניות שהפעולה קיבלה לפני שתעבור לשורה הבאה(אבל זה לא אומר שthread אחד לא יכול לרוץ בזמן ש-thread אחר נמצא בפעולה של הsleep()- אלא להפך).

datetime:

פעולת תאריך/זמן עדכני. השתמשתי בה בעיקר כדי לקבל את הזמן הנוכחי.

datetime.datetime.now()- מחזירה את הזמן הנוכחי, ניתן להדפיסו וגם ניתן לחשב הפרשי זמנים עם זמן נוכחי אחרון פחות זמן נוכחי שנמדד לפני כן, השתמשתי בזה בשביל עדכון הטיימר וגם בתת שרת כדי לדעת מתי עברה שנייה.

hashlib:

ספרייה הכוללת פונקציות גיבוב. השתמשתי בפונקציה sha256 בעזרת הפעולה:

hashlib.sha256()- מקבלת מחרוזת בינארית/תווים בינאריים ומחזירה אובייקט גיבוב של התווים. אם נקרא לאובייקט obj לשם הדוגמה ניתן לאחר מכן להשתמש ב-obj.hexdigest() כדי לקבל את מחרוזת של ספרות הקסדצימליות (בסיס 16) של הגיבוב ואותה לשמור בבסיס הנתונים.

uuid:

יכולה ליצור מחרוזת בינארית unique ייחודית- השתמשתי בפעולה:

uuid.uuid4().hex - ייצור מחרוזת של 16 תווים בינאריים. השתמשתי במחרוזת הזו בתור salt בשביל ה- salted hash.

math:

ספריית פעולות מתמטיות.

math.sqrt()- מחזירה את השורש של מספר כלשהו (מקבלת את המספר). השתמשתי בה בכדי לחשב מרחק בין שתי נקודות(פשוט משפט פיתגורס/נוסחה).

string:

ספריית מחרוזות.

string.ascii\_letters- כל האותיות באנגלית קטנות וגדולות a-z , A-Z.

string.digits- כל הספרות 0-9.

string.punctuatio- כל סימני הפיסוק.

חיברתי את שלושתם (string.ascii\_letters + string.digits + string.punctuation) כדי לקבל את כל התווים שניתן להדפיס. לאחר מכן הוצאתי תווים כמו פסיק ומרכאות כדי ליצור מחרוזת שתכלול את כל התווים שמותר להכניס בשם משתמש וסיסמה.

itertools:

ספריית כלי עזר למשימות שנדרשות לולאה בשבילן(iteration=איטרציה של לולאה).

Itertools.permutations()- מקבלת רשימה של טווח מספרים וגודל קבוצה, מחזירה את כל הקבוצות האפשריות המקיימות את התנאים. לדוגמה אם הטווח הוא [0,1,2](אפשר גם להכניס מחרוזת "123") וגודל הקבוצה הוא 3 יוחזר:

נעזרתי בפעולה בכדי למצוא את כל הדרכים האפשריות בין הנקודות במפה כדי למצוא את הדרך הכי יעילה(מצאתי את סכום המרחקים הכי קטן וזה קורה בפעולה find\_best\_route).

pygame:

תיאור הספרייה כבר צוין באפיון הפרויקט: ממשק גרפי נוח של השפה(פיתון) שבעזרתו אפשר להציג מידע בקלות על המסך, בין אם המידע הוא תמונה ספציפית או צבעים שרוצים להציג על פיקסלים מסוימים במסך. בנוסף בעזרת ממשק זה אפשר להגיב לאירועים של המשתמש כמו לחיצה על העכבר.

random:

ספריית פעולות של אקראיות.

random.randint()- מקבלת שני מספרים שלמים ומגרילה מספר בטווח שבניהם(כולל).

השתמשתי בפעולה זו גם בשרת בכדי להגריל כמה בתים לבקש מכל תת שרת(מתוך סך כל הבתים) וגם בתת שרתים הגרלתי מיקומי בתים כדי לעדכן את האזורים שהם שומרים.

sqlite3:

ספרייה שמאפשרת לבצע משפטי sql בשפת פיתון, ובכך לגשת ולשנות את טבלאות בסיס הנתונים מתוך קבצי הפיתון(במקרה שלנו יש קובץ מיוחד שנכתב לשם כך והשימוש בו הוא בשרת).

sqlite3.connect()- מקבלת את קובץ בסיס הנתונים ומחזירה חיבור אליו- אותו שמרתי בתכונה conn.

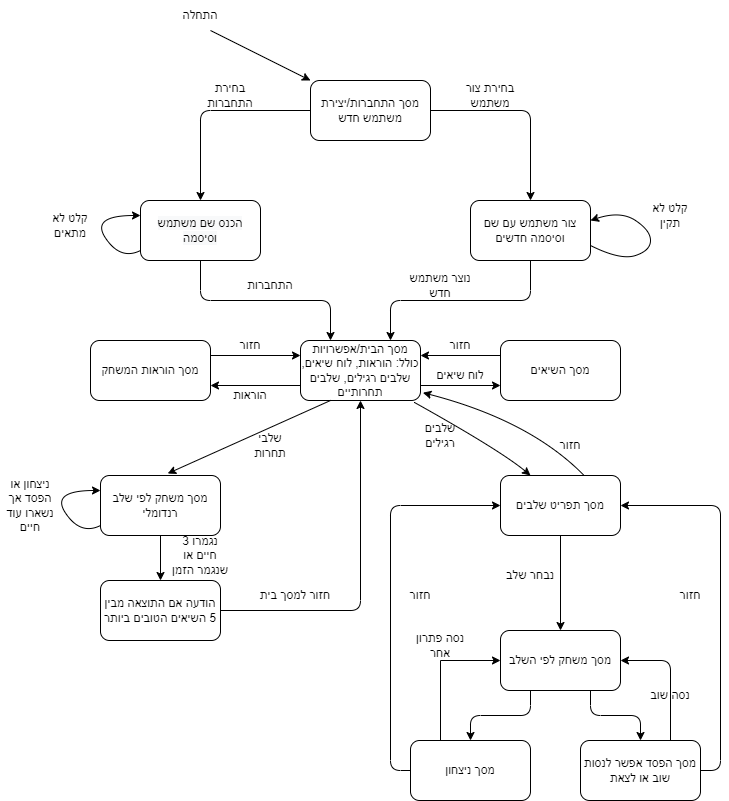
conn.cursor()- יוצר מצביע לבסיס הנתונים, דרכו אפשר לבצע שאילתות sql.

conn.close()- סוגרת את החיבור לבסיס הנתונים.

cursor.execute()- מקבלת שאילתת sql ומחזירה את הפלט שלה.

commit()- מבצעת את כל השאילתות שעשינו להן execute() ומשנות את בסיס הנתונים/כותבות לבסיס הנתונים (חובה לבצע פעולה זו לפני שסוגרים את החיבור ולא לאחר מכן).

# מדריך למשתמש

תרשים היררכיית מסכים:

הסבר כללי על פעולות המשתמש בדומה לתרשים המצורף:

תחילה יש לבחור אם רוצים ליצור משתמש חדש או להתחבר למשתמש קיים. כאשר יוצרים משתמש חדש שימו לב ששם המשתמש עומד באורך 4-16 תווים והסיסמה באורך 8-16 תווים. אם תנסו להתחבר עם שם משתמש או סיסמה לא נכונים, תנסו ליצור משתמש ששם המשתמש שלו כבר קיים במערכת או שערכי שם המשתמש או הסיסמה שלו לא עומדים בתנאים תקפוץ הודעת שגיאה בהתאם. אם הכל תקין תוכלו להתחבר/ליצור משתמש. לאחר מכן תגיעו למסך הבית. זהו המסך הראשי, ממנו אפשר להגיע למסך הוראות המשחק או למסך השיאים במשחק או לשחק בשלבים רגילים או תחרותיים. מכל אחד מהמסכים האלו אפשר לחזור כמובן למסך הבית.

חוקי המשחק:

בכל שלב יוצג למשתמש מספר בתים במיקומים שונים על המסך. יהיה על המשתמש לבחור את הבתים בסדר כלשהו. כדי לבחור בית צריך פשוט ללחוץ עליו. אם נבחר בית לא רצוי אפשר לבטל אותו בעזרת לחיצה על undo.

מטרת המשחק- בכל שלב על המשתמש למצוא את המסלול הטוב ביותר(יתכן שיהיו כמה כאלו). בסיום כל שלב המשתמש יוכל המשתמש לראות אם הצליח את אותו או לא ולנסות לעשות אותו שוב.

בסוג המשחק של שלבים רגילים:

תחילה במסך תפריט השלבים יש לבחור שלב מבין השלבים שצבועים בירוק(שהם השלבים שהמשתמש ניצח) או את השלב הצהוב- כלומר השלב שהמשתמש נמצא בו כרגע. לאחר הפסד בשלב ניתן לנסות אותו שוב ע"י לחיצה על try again. בכל מצב אפשר לחזור לתפריט השלבים.

בסוג המשחק של שלבים תחרותיים:

על המשתמש לפתור כמה שיותר שלבים מבלי להיפסל 3 פעמים ולפני שנגמר הזמן. על כל שלב יתווסף 1 לכמות השלבים שהמשתמש פותר. ברגע שהמשתמש נפסל 3 פעמים או שנגמר הזמן יקבל הודעה אם נכנס ל-5 השיאים הטובים ביותר ויחזור למסך הבית.

מסך הפתיחה:

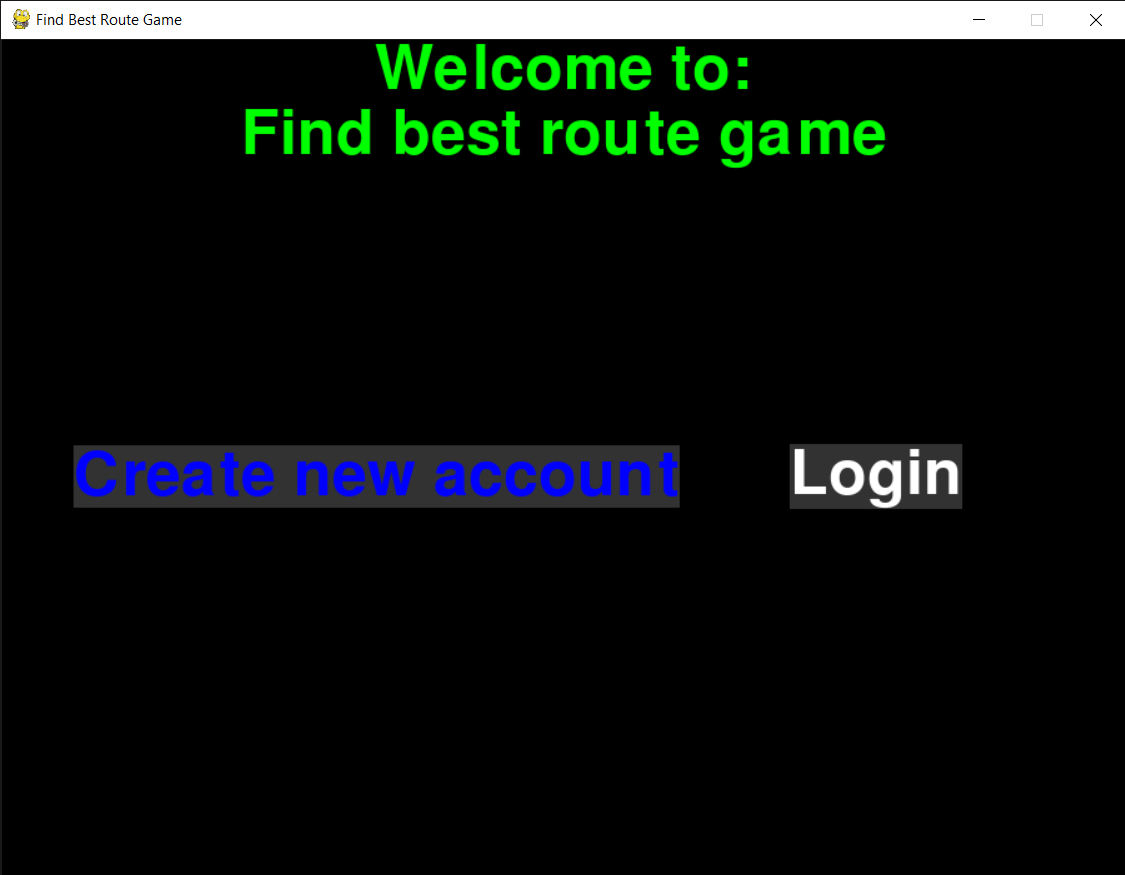
לכל מסך אפשרויות(כמו מסך הפתיחה) יש כותרת ואחריה מוצגות האפשרוית.

יש ללחוץ על אפשרות כדי לעבור למסך.

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

ברגע שעומדים עם העכבר על אפשרות כלשהי הצבע שלה ישתנה(לא רואים את העכבר בתמונה אך כשצילמתי הוא היה על האפשרות שבכחול):



לאחר לחיצה על Create new account ממסך הפתיחה נקבל:

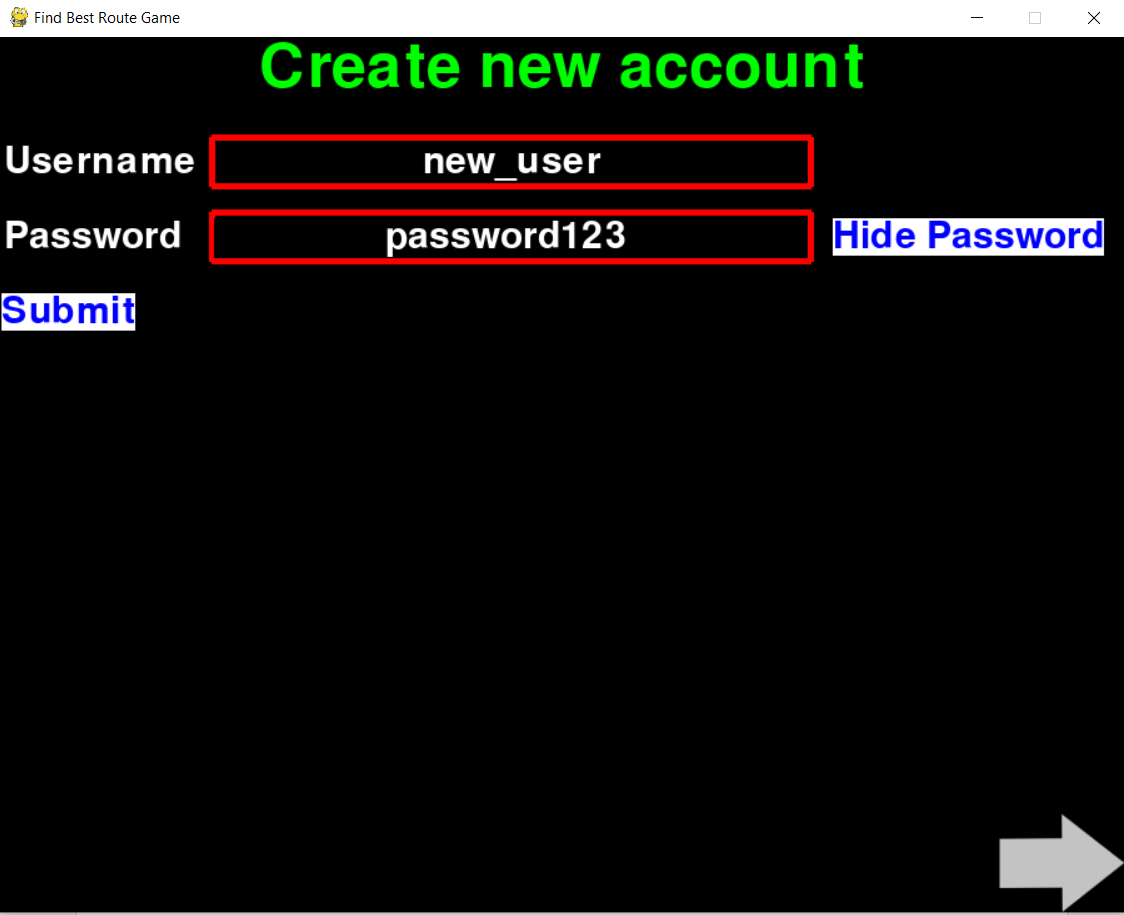
מסך יצירת משתמש חדש:

יש ללחוץ עם העכבר בתוך כל שדה לפני שממלאים אותו.

לחיצה על 'sumbit' תכניס את הפרטים(שימו לב שעל שם המשתמש להיות בעל לפחות 4 תווים ועל הסיסמה להיות בעלת לפחות 8 תווים). אם תכניסו שם משתמש שכבר קיים או פרטים לא תקינים תקבלו שגיאה ותוכלו להכניס מחדש.

ניתן להסתיר ולהראות את הסיסמה ע"י לחיצה על hide password(או show\_password).

לחיצה על החץ למטה תחזיר אותנו למסך הפתיחה.



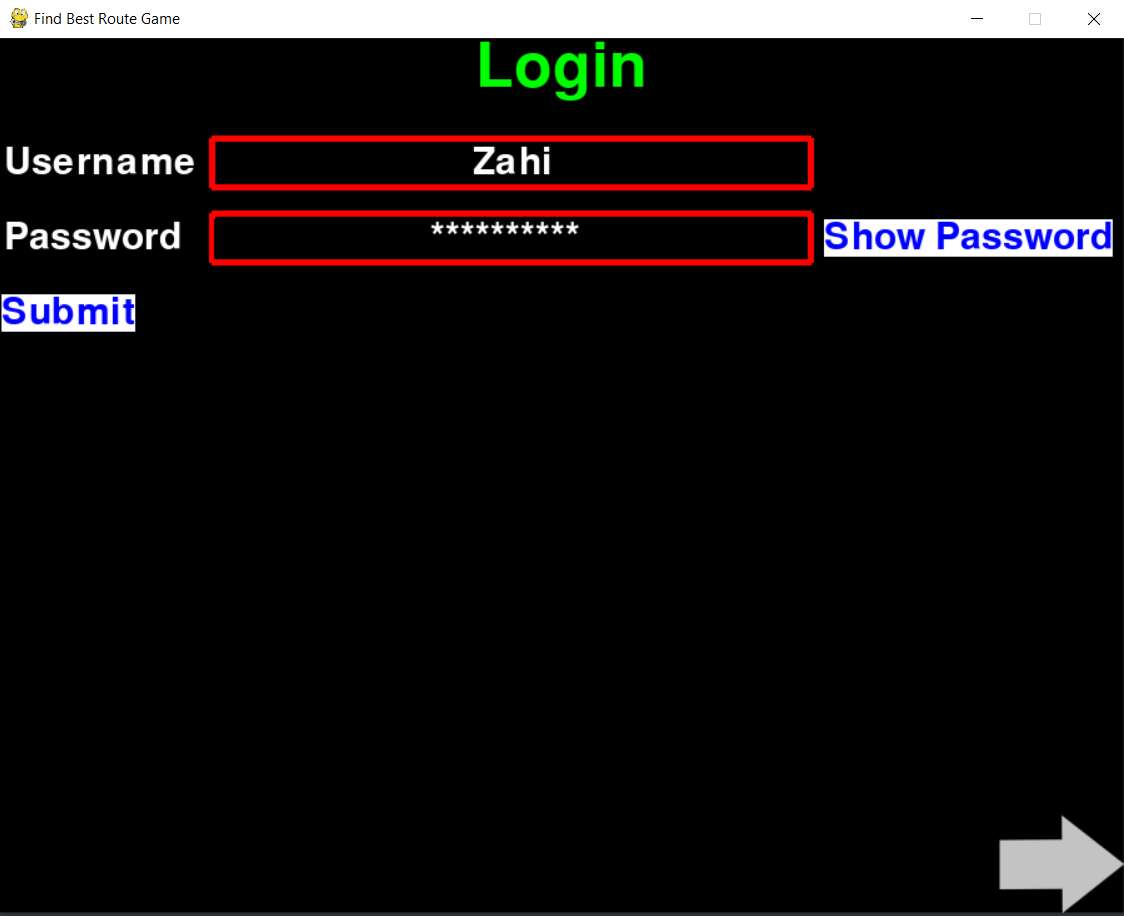
לאחר לחיצה על login במסך הפתיחה נקבל:

מסך התחברות:

דומה כמעט לחלוטין למסך יצירת המשתמש, (אך משמש כדי להתחבר למשתמש במקום ליצור)

גם כאן אם תכניסו שם משתמש או סיסמה לא תקינים או שלא קיימים במערכת תקפוץ שגיאה ותוכלו להכניס מחדש.

כאן הדוגמה עם סיסמה מוסתרת(ניתן ללחוץ show\_password כדי לראות אותה).



לאחר שיצרנו משתמש חדש או שהתחברנו למשתמש קיים נקבל:

מסך בית:

מסך אפשרויות ממנו אפשר לעבור לשלבים רגילים, שלבים תחרותיים, שיאים או הוראות המשחק.

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

אם בחרנו במסך הבית באפשרות Campaign נקבל:

מסך תפריט שלבים:

בירוק אפשר לראות את כל השלבים שהושלמו.

בצהוב השלב שהמשתמש נמצא בו.

באדום השלבים שעוד לא הושלמו ויש להשלים שלבים קודמים כדי לנסות שלבים אלו(אם תנסו תקפוץ שגיאה שלא סיימתם את השלבים שלפני כן ותוכלו לבחור שלב אחר).

(לחיצה על החץ למטה תחזיר אותנו למסך הבית).

תמונה שמכילה טקסט, אלקטרוניקה, לוח מקשים

התיאור נוצר באופן אוטומטי

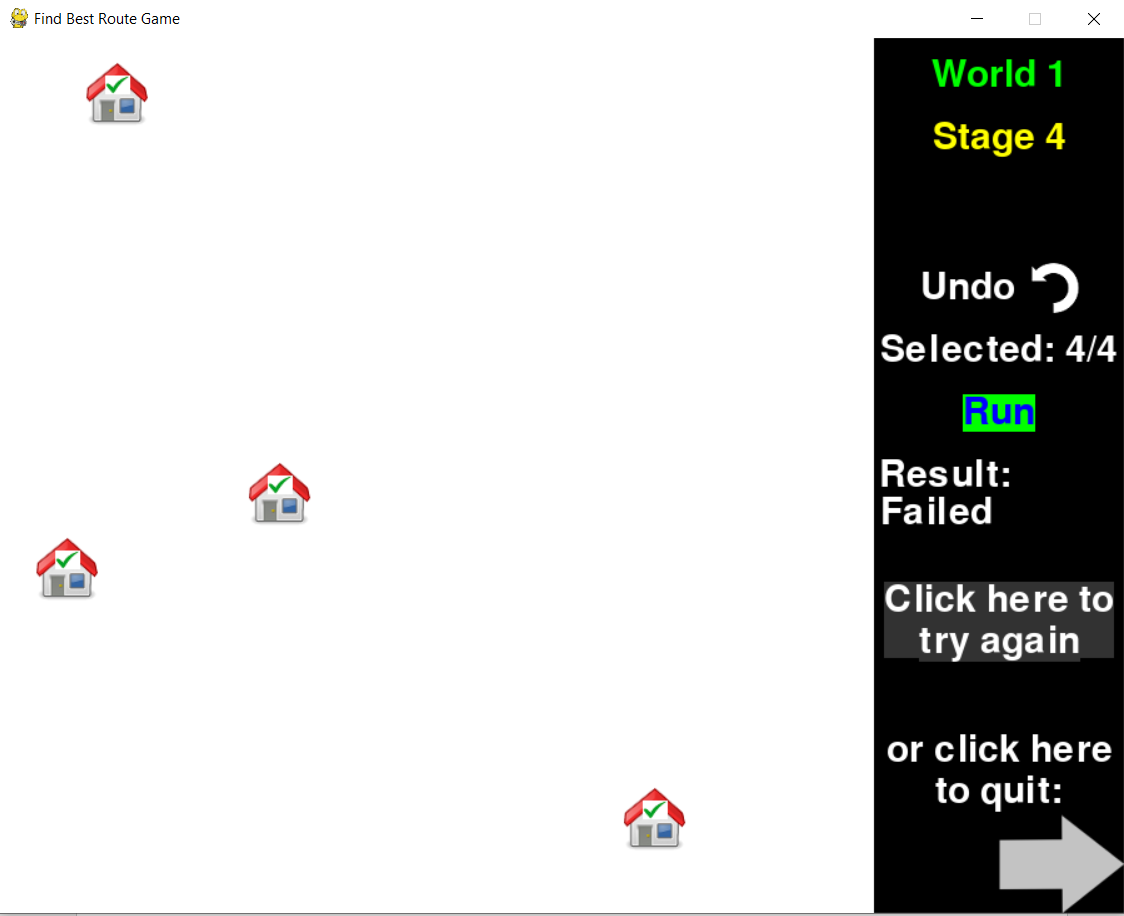
לאחר בחירת שלב נקבל: (התמונה הבאה היא אחרי שברחנו בשלב 1-4)

מסך שלב רגיל:

בצד ימין: אפשר לראות את מספר העולם, מספר השלב, יש אפשרות undo- נלחץ כדי לבטל את הבית האחרון שבחרנו, אפשר לראות גם כמה בתים בחרו מתוך המקסימום בשלב.



נחבר את כל הבתים בסדר מסוים ונלחץ Run:



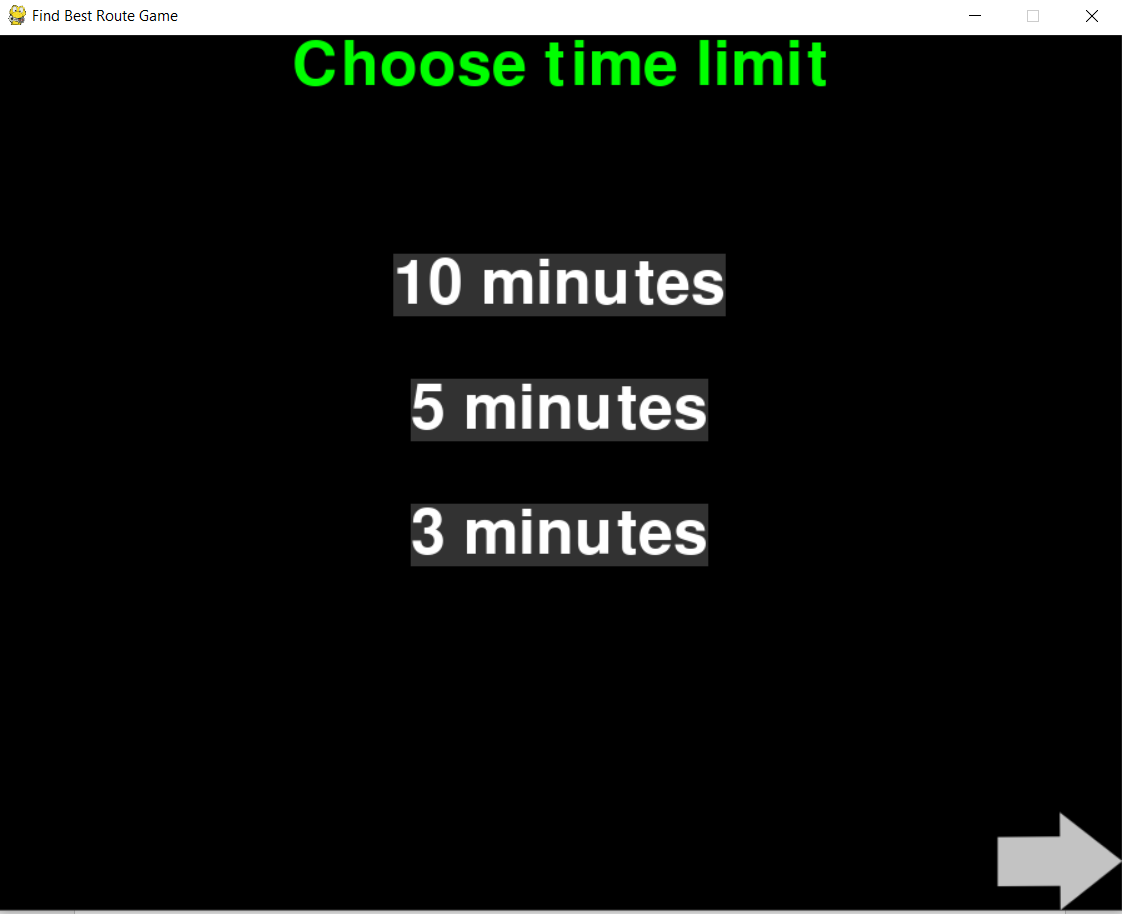
קיבלנו הודעה שלא הצלחנו את השלב. ננסה שוב.



הצלחנו את השלב ועכשיו אפשר לעבור לשלב הבא.

אם בחרנו באופציה'competitive' במסך הבית נקבל:

מסך בחירת זמן:



לאחר שבחרנו את הזמן הרצוי נקבל:

מסך שלב תחרותי:

בצד ימין: אפשר לראות את הזמן שנשאר לנו, כמה חיים/פסילות נשארו לנו וכמה שלבים עשינו.

כמו בשלב רגיל לוחצים undo כדי לבטל בחירה של בית, אפשר לראות כמה בתים בחרנו מתוך סך כל הבתים בשלב וכשמסיימים לוחצים Run.

המטרה היא לעשות כמה שיותר שלבים בלי להיפסל 3 פעמים ולפני שנגמר הזמן.

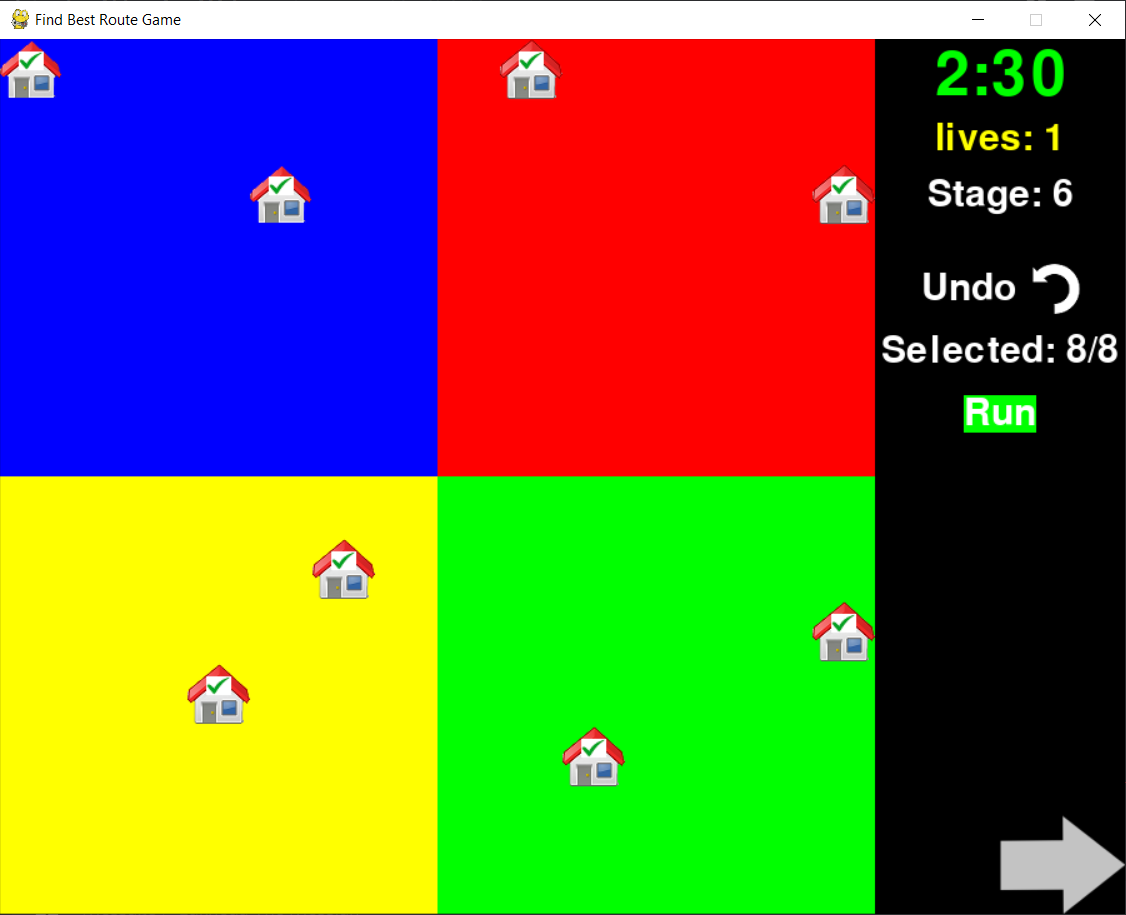
מספר הבתים בכל שלב קבוע אבל המיקומים של הבתים אקראיים(בניגוד לשלב רגיל שבו המיקומים קבועים).

מפת שלב תחרותי מעוצבת עם 4 צבעים שונים כדי להציג לנו את החלוקה של הזיכרון למחשבים- כל צבע מציג אזור שהובא מתת שרת אחר(תת השרת כל הזמן מגריל מיקומים וברגע שהשרת הראשי מבקש ממנו מיקום הוא שולח אותו ומגריל שוב). במקרה הזה זהו השלב הראשון ולכן הוא הכי קל מבחינת כמות בתים ויש בו רק 3 בתים, אז מתת השרת שאחראי על האזור שצבוע בכחול לא ביקשנו אזור כלל.

תמונה שמכילה ריבוע

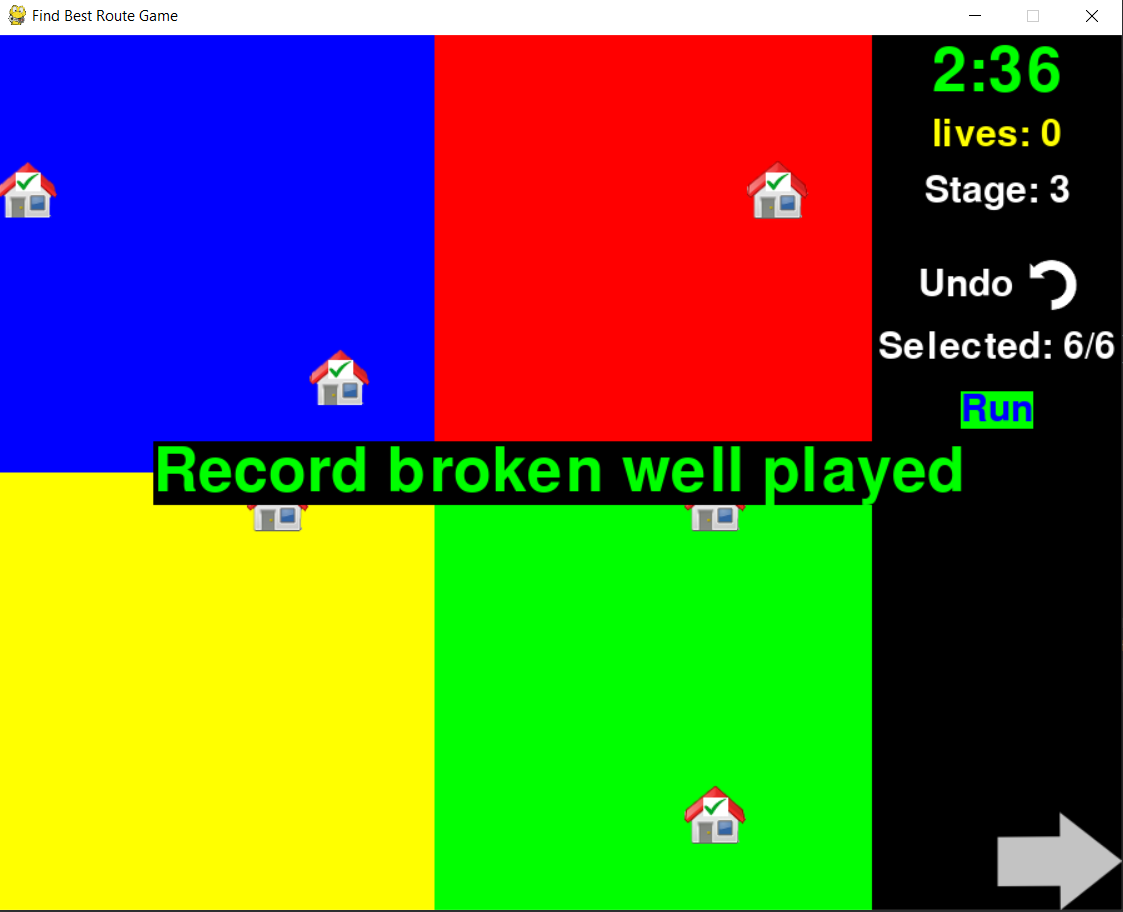
התיאור נוצר באופן אוטומטי

לאחר כמה שלבים נוספים בתים ולכן רמת הקושי עולה(בממוצע):



אחרי שסיימנו נקבל הודעה למסך ל-2 שניות אם השיא שלנו מבין חמשת השיאים הכי טובים לקטגורית הזמן.

לדוגמה:



אפשר לצפות בשיאים ע"י לחיצה על records במסך הבית, אחריה נקבל:

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

מסך בחירת קטגורית זמן שיא:

(לחיצה על החץ חזור תחזיר אותנו למסך הבית)

לאחר שבחרנו קטגורית זמן נקבל:

מסך שיאים:

לכל שיא רואים: דירוג כל (אם יש שני שיאים עם אותו מספר שלבים השיא שבוצע יותר קודם יהיה בדירוג יותר טוב), מספר השלבים שבוצעו, (שם) המשתמש שביצע את השיא, התאריך והשעה שבו בוצע השיא.



התקנות:

אני עבדתי בפרויקט על פיתון 3.7, אבל כל גרסת פיתון 3.5-3.10 תעבוד(כנראה שגם שמעל 3.10 פשוט זו הגרסה הכי עדכנית שבדקתי). יש להתקין גם את כל הספריות הבסיסיות המוצעות בהתקנה(אי אפשר בלי pip).

יש להריץ ב-cmd (או בטרמינל ב(pycharm- את הפקודות הבאות כדי להתקין את הספריות החיצוניות הנדרשות:

pip install pygame

pip install pycryptodome

# בסיס הנתונים

השרת הראשי ישמור את טבלאות ה-sql שימומשו ב-Sqlite3.

הטבלאות שבמבנה הנתונים:

\*הערה: כל הרשומות שלהלן הן חובה ולא יכול להיות שדה ריק.

טבלת משתמשים- שתשמור את המידע לכל משתמש. בעזרתה נוכל לבדוק האם קיים צמד שם משתמש וסיסמה כלשהו(ע"י הsalted hash של הסיסמה), האם שם משתמש מסוים תפוס(בשביל יצירת משתמש חדש) ומה ההתקדמות של כל משתמש.

כל רשומה בטבלה תכלול 3 עמודות:

1. שם משתמש (טיפוס varchar)

2. Salt (טיפוס varchar)

3. Hash של סיסמה (טיפוס varchar).

4. מספר שלב שאליו המשתמש הגיע(טיפוס int).

העמודות ורשומה לדוגמה:

|  |  |  |  |
| --- | --- | --- | --- |
| UserName | Salt | SaltedPasswordHash | Stage |
| "Zahi" | "53cdd2a0853f46d79631daaf569314e7" | "12fe0c3805b84e0ce5b48117a90484c0d8861f8ae6805102d68cae91aa775183" | 3 |

טבלת שיאים- שתשמור את התוצאות הטובות ביותר לכל קטגורית זמן מסוימת, כך שכל משתמש יוכל לראות את השיאים שנקבעו וע"י מי הם נקבעו.

כל רשומה בטבלה תכלול 5 עמודות:

1. דירוג השיא (1=מקום ראשון/הכי טוב) (טיפוס int).

2. מספר השלבים שנפתרו(ללא 3 פסילות ולפני שנגמר הזמן) (טיפוס int).

3. הזמן שלקח לפתור את השלבים בדקות(טיפוס int).

4. שם המשתמש שקבע את השיא (טיפוס varchar).

5. התאריך שבו נקבע השיא (שעה-יום-חודש-שנה) (טיפוס varchar).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Rank | TimeInMinutes | Stage | UserName | Date |
| 2 | 3 | 5 | "Zahi" | "2021-11-21-14:46" |

השרת הראשי ישמור את המידע על כל שלב באמצעות קובץ .txt כך גם תתי השרתים ישמרו את המידע על האזורים השונים שלהם. שיטת שמירת המידע של השלב/אזור פשוטה, בכל שלב יש מספר של בתים במיקומים שונים, בין כל בית לבית יהיה רווח שורה '/n'(כל בית ישמר בשורה משל עצמו) ולכל בית ישמרו שני מספרים x ו-y, ביניהם יפריד פסיק.

לדוגמה שלב 2 שבו יש בית 1 במיקום (100, 100), בית 2 במיקום (500, 150) ובית 3 במיקום (400, 450) ישמר בצורה הבאה:

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

באופן דומה ישמור כל תת שרת את המידע על אזור. כשהשרת הראשי יצטרך מפת שלב שלמה הוא יבקש מכל תת שרת אזור עם מספר שונה של בתים בהתאם לשלב הנדרש ואז יחבר את כל האזורים לתמונה מלאה וישלח ללקוח.

לדוגמה אזור שבו יש 3 בתים, בית 1 ב- (100, 250), בית 2 ב-(0, 50), בית 3 ב-(400, 450) יקראarea3.txt (על שם מספר הבתים באזור) ויראה כך:

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

# מדריך למפתח

הערה: בכל פעם שנכתב str הכוונה למחרוזת (זה הטיפוס של מחרוזת בפיתון קיצור של string).

הקבצים למימוש הפרויקט:

client.py:

קובץ הלקוח. קובץ זה נועד לשמש כלקוח במשחק, הוא יתקשר עם השרת במידת הצורך וגם יציג את המשחק למשתמש וידע להגיב בהתאם לפעולות המשתמש. המטרה היא שבעזרת קובץ זה המשתמש יקבל חוויה נעימה ונוחה של המשחק, תוך כדי שהקוד מתקשר בשבילו עם השרת ללא הרגשת המשתמש.

משתנים:

קבועים:

-SERVER\_IPמחרוזת של כתובת ip של השרת הראשי.

PORT- הפורט (מספר שלם) שעליו יתקשרו השרת והלקוח.

גלובליים:

משתנים לתקשורת בין הthread של מסך המשחק לthread של התקשורת:

login ו-create\_account בוליאניים- אמת אם הלקוח התחבר/יצר משתמש ושקר אם עוד לא סיים להתחבר/ליצור משתמש.

account\_response- מחרוזת של תגובת השרת להתחברות/יצירת המשתמש (האם המשתמש הצליח להתחבר/ליצור משתמש).

bool\_selected\_stage- בוליאן האם נבחר שלב או שעוד לא.

passed\_stage\_campaign- בוליאן האם השלב הובס או לא.

stage\_reached- השלב אליו הגיע המשתמש.

stage\_selected- השלב שהמשתמש בחר.

stage\_map\_list- רשימה שכוללת את כל המיקומים של הנקודות/בתים בשלב הנוכחי.

מתוך פעולות:

route- אובייקט מסוג Dequeu (פירוט על המחלקה בהמשך) שישמור את המסלול הנוכחי של המשתמש.

screen\_state- (int) מספר שלם המייצג את מצב המסך הנוכחי. כל פעולת מסך מחזירה את מצב המסך בסיומה וכך בלולאה הראשית יודעים לאיזו פעולה לקרוא(לפי מצב המסך שקיבלנו).

פונקציות:

\*נקודה- רשימה שבה שני מספרים שלמים: x ו-y.

|  |  |  |  |
| --- | --- | --- | --- |
| חתימת הפעולה | תיאור | טענת כניסה | טענת יציאה |
| find\_best\_route(points) | מחשבת את המרחק המינימלי ואת הדרך היעילה ביותר. | מקבלת רשימה של כל הנקודות במפה. | מחזירה את המרחק המינימלי(מספר לא שלם) ואת הדרך היעילה ביותר (המתאימה למרחק המינמלי) (רשימה של מספרי הבתים(int)). |
| calc\_total\_dist(points, order) | מחשבת את סכום המרחקים בין נקודות לפי סדר מסוים. | מקבלת רשימה של הנקודות במפה ועוד רשימה שמייצגת את הסדר שבו נבחרו(מספרים שלמים). | ומחזירה את סכום המרחקים(מספר לא שלם). |
| get\_distance(p1, p2) | מחשבת מרחק בין שתי נקודות. | מקבלת שתי נקודות. | מחזירה מרחק בין נקודות(מספר לא שלם). |
| is\_mouse\_in\_rect(self, min\_x, min\_y, max\_x, max\_y) | בודקת אם העכבר במלבן בין הנקודות מינימום ומקסימום x ו-y. | מקבלת מינימום ומקסימום של x ו-y(4 מספרים שלמים int) | מחזירה True אם העכבר בתוך המלבן, אחרת False(bool). |

מחלקות:

מחלקת Dequeue שבשבילה כתבתי גם את מחלקת DoubleNode נועדה להוות מבנה נתונים נוח ויעיל בשביל לשמור את הנתיב/הדרך בכל שלב. המבנה הוא כמו תור וגם מחסנית ויכלתי לייבא אותו מספרייה, כך:

from collections import deque

אני החלטתי שאני רוצה לממש אותו בעצמי, המימוש נמצא בנספחים.

שם המחלקה: Thread

תיאור: יורשת מ- threading.Thread. מחלקת תהליכון שתשמור מידע על כל תהליכון באופן מסודר, דרכה כל ה-threads ירוצו.

תכונות:

tid – (קיצור של thread id) (int) מספר שלם שמיצג את מקומו הסידורי של התהליכון.

lock – נעילת התהליכון (טיפוס \_thread.lock).

מתודות:

|  |  |  |  |
| --- | --- | --- | --- |
| חתימת הפעולה | תיאור | טענת כניסה | טענת יציאה |
| \_\_init\_\_(self) | פעולה בונה, מריצה את הפעולה הבונה של מחלקת האב ומאתחלת את התכונות. | לא מקבלת כלום. | לא מחזירה כלום. |
| \_\_str\_\_( self) | יוצרת מחרוזת של תכונות התהליכון. | לא מקבלת כלום. | מחזירה מחרוזת של תכונות התהליכון. |
| run(self) | מריצה את הthread. | לא מקבלת כלום. | לא מחזירה כלום. |
| thread\_started(self) | מדפיסה את הזמן הנוכחי ו-id של הthread. | לא מקבלת כלום. | לא מחזירה כלום. |

שם המחלקה: CommunicationThread

תיאור: יורשת מ- Thread. מחלקת התהליכון של התקשורת.

תכונות:

sock- סוקט התקשורת של הthread (טיפוס socket.socket).

key- המפתח שאיתו מצפינים ומפענחים הודעת בעלות מידע רגיש(בינארי- טיפוס bytes).

מתודות:

\*comm- קיצור של communication (בפעולות התקשורת).

|  |  |  |  |
| --- | --- | --- | --- |
| חתימת הפעולה | תיאור | טענת כניסה | טענת יציאה |
| \_\_init\_\_(self) | פעולה בונה, מריצה את הפעולה הבונה של מחלקת האב ומאתחלת את התכונות. | לא מקבלת כלום. | לא מחזירה כלום. |
| run(self) | מריצה את הthread.-ממשת את האלגוריתם שלו: מייצרת מפתח לפי דיפי הלמן, קוראת לפעולות התקשורת של ההתחברות. לאחר ההתחברות רצה בלולאה אינסופית ולפי שינוי משתנים גלובליים קוראת לפעולות תקשורת עם השרת. | לא מקבלת כלום. | לא מחזירה כלום. |
| records\_comm(self) | מבקשת מהשרת את השיאים ושומרת אותם במשתנה גלובלי | לא מקבלת כלום. | לא מחזירה כלום. |
| check\_record\_comm() | בודקת האם נשבר שיא ומעדכנת משתנה גלובלי בהתאם. | לא מקבלת כלום. | לא מחזירה כלום. |
| passed\_campaign\_stage\_comm(self) | מעדכן את השרת שהמשתמש עבר שלב | לא מקבלת כלום. | לא מחזירה כלום. |
| stage\_map\_comm(self, is\_competitive) | מבקשת מהשרת את המפה המתאימה לשלב ושומרת אותה במשתנה גלובלי(רשימה). | מקבלת בוליאן-האם השלב תחרותי או רגיל(bool). | לא מחזירה כלום. |
| account\_comm(self) | שולחת לשרת את שם המשתמש והסיסמה שהוכנסו באופן מוצפן ושומרת את תגובתו במשתנה גלובלי. | לא מקבלת כלום. | לא מחזירה כלום. |
| send\_with\_size(self, data) | שולחת את ההודעה דרך סוקט המחלקה. | מקבלת מחרוזת הודעה לשלוח(str). | לא מחזירה כלום. |
| receive\_by\_protocol(self) | קולטת הודעה מסוקט המחלקה ומחזירה את הסוג ההודעה וגוף ההודעה. | לא מקבלת כלום. | לא מחזירה כלום. |
| receive\_by\_size(self) | קולטת הודעה מסוקט המחלקה ומחזירה אותה. | לא מקבלת כלום. | לא מחזירה כלום. |
| send\_encrypted\_msg(self, msg) | מצפינה הודעה לפי מפתח המחלקה ושולחת אותה דרך סוקט המחלקה.(בעזרת פעולה מקובץ ההצפנה dh\_aes) | מקבלת את ההודעה שרוצים לשלוח(str). | לא מחזירה כלום. |
| receive\_encrypted\_msg(self) | קולטת הודעה מסוקט המחלקה, מפענחת אותה לפי מפתח האובייקט ומחזירה את תוכן ההודעה המקורית(בעזרת פעולה מקובץ ההצפנה dh\_aes). | לא מקבלת כלום. | מחזירה את תוכן ההודעה המקורית (str). |

שם המחלקה: ScreenThread

תיאור: יורשת מ- Thread. מחלקת התהליכון של המסך.

תכונות:

screen- מסך pygame (טיפוס (pygame.Surface.

מתודות:

|  |  |  |  |
| --- | --- | --- | --- |
| חתימת הפעולה | תיאור | טענת כניסה | טענת יציאה |
| \_\_init\_\_(self) | פעולה בונה, מריצה את הפעולה הבונה של מחלקת האב ומאתחלת את התכונות. | לא מקבלת כלום. | לא מחזירה כלום. |
| run(self) | מריצה את הthread. ממשת את האלגוריתם שלו: פותחת את חלון ה-pygame, רצה בלולאה(עד שנסגר חלון הpygame) וקוראת לפעולה לפי משתנה גלובלי של מצב המסך(int), לכל מצב מסך מספר משלו. | לא מקבלת כלום. | לא מחזירה כלום. |
| welcome\_screen(self) | מציירת את מסך הפתיחה ומגיבה לאירועיו. | לא מקבלת כלום. | לא מחזירה כלום, פלט מסך. |
| account\_screen(self, log\_in) | מציירת את מסך התחברות/יצירת משתמש ומגיבה לאירועיו. | האם זה מסך התחברות או מסך יצירת משתמש(bool). | לא מחזירה כלום, פלט מסך. |
| get\_screen\_input(self, x\_pos, y\_pos, max\_len, entered\_data, hide\_data) | מקבלת קלט מהמשתמש ותציג אותו על המסך. | מיקום אופקי ואנכי של הקלט על המסך(שני int), אורך מקסימלי של הקלט (int), קלט שהוכנס כבר ((str והאם להראות את הקלט או להסתיר(bool). | מחזירה את מצב המסך(מיוצג על ידי מספר int) והמידע שהוכנס((str. |
| home\_screen(self) | מציירת את מסך הבית ומגיבה לאירועיו(לחיצה על אפשרות). | לא מקבלת כלום. | לא מחזירה כלום, פלט מסך. |
| stages\_menu\_screen(self) | מציירת את מסך תפריט השלבים ומגיבה בהתאם ללחיצה על שלב. | לא מקבלת כלום. | לא מחזירה כלום, פלט מסך. |
| records\_options\_screen(self) | מציירת את מסך אפשרויות זמני שיאים ומגיבה לבחירת אפשרות. | לא מקבלת כלום. | לא מחזירה כלום, פלט מסך. |
| records\_screen(self) | מציירת את מסך השיאים של קטוגריה מסוימת לפי משתנה גלובלי. | לא מקבלת כלום. | לא מחזירה כלום, פלט מסך. |
| stage\_screen(self, is\_competitive) | תצייר מסך שלב ותגיב בהתאם לפעולות המשתמש. | האם השלב תחרותי או רגיל(bool). | לא מחזירה כלום, פלט מסך. |
| update\_time(self, time\_diff\_in\_seconds) | תחשב זמן חדש שנשאר ותעדכן את הסטופר. | הזמן שעבר מאז העדכון האחרון(int). | מחזירה האם נשאר זמן או שלא(כלומר נגמר) (bool) וגם פלט מסך סטופר. |
| competitive\_options\_screen(self) | מציירת את מסך האפשרויות של שלב תחרותי ומגיבה לבחירת אפשרויות. | לא מקבלת כלום. | לא מחזירה כלום, פלט מסך. |
| campaign\_succeeded\_stage(self) | מדפיסה למסך הודעה שהשלב הובס ומעדכנת משתנה גלובלי על כך. (נותנת אפשרות לעשות את השלב מחדש או לצאת) | לא מקבלת כלום. | לא מחזירה כלום, פלט מסך. |
| campaign\_failed\_stage(self) | מדפיסה למסך הודעה שהשלב לא הובס ומעדכנת משתנה גלובלי על כך (נותנת אפשרות לעשות את השלב מחדש או לצאת) | לא מקבלת כלום. | לא מחזירה כלום, פלט מסך. |
| how\_to\_play\_screen(self) | קוראת מקובץ את הוראות המשחק ומציגה אותן על המסך. | לא מקבלת כלום. | לא מחזירה כלום, פלט מסך. |
| show\_text\_colorful(self, text, x, y, size, font\_color, background\_color) | מציגה טקסט על המסך במיקום ובצבעים שמקבלת. | מקבלת טקסט(מחרוזת str), מיקום x, y (שני מספרים שלמים int), גודל כתב (int), צבע פונט(tuple של 3 מספרים שלמים), צבע רקע (גם tuple של 3 מספרים שלמים). | לא מחזירה כלום, פלט מסך. |
| show\_text(self, text, x, y) | מציגה טקסט על המסך במיקום שקיבלה. הצבע נקבע לפי משתנה גלובלי(כך שאם צריך אותו צבע כמה פעמים ברצף אפשר להשתמש בפעולה הזאת ולא צריך הרבה פרמטרים). | מקבלת טקסט(מחרוזת str), מיקום x, y (שני מספרים שלמים int) | לא מחזירה כלום, פלט מסך. |
| def show\_image(self, img\_dir, x, y) | מציגה תמונה על המסך במיקום מסוים. | מקבלת מחרוזת של מיקום התמונה, שני מספרים שלמים x ו-y של מיקום התמונה. | לא מחזירה כלום, פלט מסך. |

server.py:

קובץ השרת הראשי, בעל גישה למבנה הנתונים, יכול לשנות אותו או להביא ממנו מידע ולשלוח את המידע ללקוח. בנוסף אחראי על העברת הלקוח לשרת משנה- השרת הראשי ידע את כל הכתובות ip של כל השרתים המשניים.

משתנים:

קבועים:

LOCAL\_IP- "0.0.0.0".

PORT- הפורט שעליו יתקשרו השרת והלקוח.

DB\_FILE\_LOCATION- מיקום הקובץ של בסיס הנתונים.

גלובליים:

global\_lock- הנעילה של הthread, נועד כדי למנוע התנגשויות בין threads.

helpers\_available- רשימת בוליאנים, אחד לכל תת שרת שמייצג האם תת שרת מחובר או לא.

areas- רשימה של 4 אזורים של המפה, כל אזור מגיע מתת שרת אחר. אזור- רשימה של נקודות(נקודה היא רשימה של שני מספרים שלמים

מתוך ה-main():

threads- רשימה שתשמור אובייקטים מסוג Thread, רשימת התהליכונים של השרת.

פונקציות:

|  |  |  |  |
| --- | --- | --- | --- |
| חתימת הפעולה | תיאור | טענת כניסה | טענת יציאה |
| check\_dc() | לפי הרשימה הגלובלית helpers\_available בודקת אם אחד השרתים התנתק. | לא מקבלת כלום. | מחזירה True אם לפחות אחד מתת השרתים מנותק, False אחרת(bool). |

מחלקות:

מחלקת Thread (הוסבר כבר בלקוח).

מחלקת CommunicationThread דומה לזו שבלקוח אך עם שינויים כך שתתאים לשרת.

שם המחלקה: CommunicationThread

תיאור: יורשת מ- Thread. מחלקת התהליכון של התקשורת.

תכונות:

sock- סוקט התקשורת של הthread (טיפוס socket.socket).

address- tuple של כתובת היעד שאיתה מתשרים(str), פורט היעד(int).

user- המשתמש איתו הthread מתקשר (אם הוא מתקשר עם שרת משנה ישאר None).

key- המפתח שאיתו מצפינים ומפענחים הודעת בעלות מידע רגיש (בינארי-טיפוס bytes).

מתודות:

\*comm- קיצור של communication (בפעולות התקשורת).

|  |  |  |  |
| --- | --- | --- | --- |
| חתימת הפעולה | תיאור | טענת כניסה | טענת יציאה |
| \_\_init\_\_(self) | פעולה בונה, מריצה את הפעולה הבונה של מחלקת האב ומאתחלת את התכונות. | לא מקבלת כלום. | לא מחזירה כלום. |
| run(self) | בודקת האם היעד הוא לקוח או תת שרת ומריצה את הthread בהתאם. | לא מקבלת כלום. | לא מחזירה כלום. |
| פעולות תקשורת עם לקוח: |  |  |  |
| run\_thread\_client(self) | פעולת הרצת thread שמתקשר עם לקוח. מייצרת מפתח לפי דיפי הלמן, קוראת לפעולת תקשורת ההתחברות ולאחר מכן רצה בלולאה אינסופית שבה מקבלת את בקשת הלקוח וקוראת לפעולת תגובה בהתאם. | לא מקבלת כלום. | לא מחזירה כלום. |
| account\_comm(self) | מקבלת הודעה מוצפנת מהלקוח, בודקת האם הלקוח מבקש ליצור משתמש חדש או להתחבר, יוצרת משתמש אם תקין ושולחת הודעה ללקוח האם המשתמש נוצר/ האם ההתחברות הצליחה. | לא מקבלת כלום. | אם התחבר/נוצר משתמש מחזירה את האובייקט של משתמש זה, אחרת מחזירה None. |
| send\_campaign\_stage(self, stage\_wanted) | שולחת רשימה של מפת שלב ללקוח. | מקבלת את מספר השלב המבוקש(לפיו מחשבת מאיזה עולם הוא ואיזה מספר הוא בעולם)(int). | לא מחזירה כלום. |
| send\_comp\_stage(self, difficulty) | מחשבת ומגרילה כמה בתים צריך מכל תת שרת, מבקשת מתתי השרתים את האזורים בהתאם, מאחדת אותם לתמונה שלמה(מיוצג באמצעות רשימה של רשימות) ושולחת אותה ללקוח. | מקבלת את רמת הקושי של השלב(מספר int). | מחזירה True אם נשלח שלב, False אם לפחות אחד מתת השרתים לא מחובר/התנתק. |
| check\_record(self, stage, t) | בודקת האם נשבר שיא(האם השיא בטופ 5) ואם כן מעדכנת את טבלת השיאים לשיא החדש.  \*הערה: לפעולה יש את שם המשתמש ששבר את השיא מהתכונה user(אובייקט של המשתמש שאיתו הthread מתקשר). | מקבלת את מספר השלבים(int) והזמן בדקות(int) של השיא. | לא מחזירה כלום. |
| send\_all\_records\_of\_time(self, records\_time) | שולחת ללקוח את טבלת השיאים של קטגורית הזמן המבוקשת(5 השיאים הטובים ביותר). | מקבלת את הזמן של השיאים בדקות (int) (3 דקות, 5 דקות או 10 דקות). | לא מחזירה כלום. |
| פעולות תקשורת עם תת שרת: |  |  |  |
| run\_thread\_helper\_server(self) | שולחת לthread את הid שלו ואז נמצאת בלולאה אין סופית שבה בודקת אם צריך אזור מתת השרת ואם כן מבקשת ממנו. | לא מקבלת כלום. | לא מחזירה כלום. |
| get\_area\_from\_helper(self) | שולחת לתת שרת בקשה לאזור(עם מספר הבתים בו ממשתנה גלובלי) ושמה במשתנה גלובלי את הרשימה של האזור שקיבלה מתת השרת. | לא מקבלת כלום. | מחזירה True אם קיבלה הודעת תגובה תקינה, אחרת False. |
| פעולות כלליות של שליחה וקבלה (זהות לאלו שבלקוח): |  |  |  |
| send\_with\_size(self, data) | שולחת את ההודעה דרך סוקט המחלקה. | מקבלת מחרוזת הודעה לשלוח(str). | לא מחזירה כלום. |
| receive\_by\_protocol(self) | קולטת הודעה מסוקט המחלקה ומחזירה את הסוג ההודעה וגוף ההודעה. | לא מקבלת כלום. | לא מחזירה כלום. |
| receive\_by\_size(self) | קולטת הודעה מסוקט המחלקה ומחזירה אותה. | לא מקבלת כלום. | לא מחזירה כלום. |
| send\_encrypted\_msg(self, msg) | מצפינה הודעה לפי מפתח המחלקה ושולחת אותה דרך סוקט המחלקה.(בעזרת פעולה מקובץ ההצפנה dh\_aes) | מקבלת את ההודעה שרוצים לשלוח(str). | לא מחזירה כלום. |
| receive\_encrypted\_msg(self) | קולטת הודעה מסוקט המחלקה, מפענחת אותה לפי מפתח האובייקט ומחזירה את תוכן ההודעה המקורית(בעזרת פעולה מקובץ ההצפנה dh\_aes). | לא מקבלת כלום. | מחזירה את תוכן ההודעה המקורית (str). |

helper\_server.py:

תיאור: קובץ תת שרת/שרת עזר. ישמור מידע דינאמי(מתעדכן כל הזמן באופן אקראי) לאפשרויות רבות לאזורים במפה. כאשר השרת הראשי יבקש אפשרות מסוימת, ישלח את האזור המתאים ויעדכן אותו.

משתנים חשובים:

גלובליים:

my\_id- (int)- אם השרתים המשניים על אותו מחשב צריך משתנה זה כדי שכל אחד ישמור את הקבצים שלו במיקום אחר(וגם עוזר בשביל לדבג).

houses\_needed- (int)- כמות הבתים הנדרשים שיהיו באזור.

global\_lock- הנעילה של הthread, נועד כדי למנוע התנגשויות בין threads (טיפוס thread.lock\_).

מחלקות:

מחלקת Thread (הוסבר כבר בלקוח).

שם המחלקה: CommunicationThread

תיאור: יורשת מ- Thread. מחלקת התהליכון של התקשורת.

תכונות:

sock- סוקט התקשורת של ה-thread (טיפוס socket.socket).

|  |  |  |  |
| --- | --- | --- | --- |
| חתימת הפעולה | תיאור | טענת כניסה | טענת יציאה |
| \_\_init\_\_(self) | פעולה בונה, מריצה את הפעולה הבונה של מחלקת האב ומאתחלת את התכונות. | לא מקבלת כלום. | לא מחזירה כלום. |
| run(self) | מריצה את הthread. מקבלת את הid ואחר כך רצה בלולאה אין סופית- מקבלת בקשה לאזור מהשרת ושולחת אותו. | לא מקבלת כלום. | לא מחזירה כלום. |
| פעולות כלליות של שליחה וקבלה (זהות לאלו שבלקוח ובשרת): |  |  |  |
| send\_with\_size(self, data) | שולחת את ההודעה דרך סוקט המחלקה. | מקבלת מחרוזת הודעה לשלוח. | לא מחזירה כלום. |
| receive\_by\_protocol(self) | קולטת הודעה מסוקט המחלקה ומחזירה את הסוג ההודעה וגוף ההודעה. | לא מקבלת כלום. | לא מחזירה כלום. |
| receive\_by\_size(self) | קולטת הודעה מסוקט המחלקה ומחזירה אותה. | לא מקבלת כלום. | לא מחזירה כלום. |

MemoryThread:

תיאור: יורשת מ-Thread. מחלקת תהליכון שאחראי לעדכון המידע/הזיכרון של אזורי השלבים.

תכונות:

אין (אין תכונות נוספות לאלו שמורשות ממחלקת ה-Thread).

מתודות:

|  |  |  |  |
| --- | --- | --- | --- |
| חתימת הפעולה | תיאור | טענת כניסה | טענת יציאה |
| \_\_init\_\_(self) | פעולה בונה, קוראת לפעולה הבונה של מחלקת האב. | לא מקבלת כלום. | לא מחזירה כלום. |
| run(self) | מריצה את הthread. רצה בלולאה אינסופית ובכל שנייה מעדכנת את המיקומים. ברגע שמקבלת ממשתנה גלובלי שנעשה שימוש באחד הקבצים היא מעדכנת אותו ספציפית. | לא מקבלת כלום. | לא מחזירה כלום. |
| update\_file(self, num\_of\_houses) | מגרילה ערכים חדשים לקובץ לפי מספר הבתים שקיבלה. | מספר הבתים שבקובץ שיש לעדכן. | לא מחזירה כלום. |
| update\_all\_files(self) | מעדכנת את כל הקבצים(קורה כל שנייה). | לא מקבלת כלום. | לא מחזירה כלום. |

tcp\_by\_size.py:

תיאור: קובץ התקשרות בפרוטוקול tcp כשהשדה הראשון הוא גודל ההודעה.

משתנה- קבוע: TCP\_DEBUG- בוליאן- האם להדפיס הודעות דיבג.

פעולות:

|  |  |  |  |
| --- | --- | --- | --- |
| חתימת הפעולה | תיאור | טענת כניסה | טענת יציאה |
| send\_with\_size(sock, bdata) | שולחת הודעה בפרוטוקול tcp\_by\_size. | מקבלת את הסוקט עליו יש לשלוח את המידע ואת המידע הבינארי שיש לשלוח. | לא מחזירה כלום. |
| recv\_by\_size(sock) | מקבלת הודעה בפרוטוקול tcp\_by\_size. | מקבלת את הסוקט שממנו מקבלים את המידע. | מחזירה את המידע שהתקבל. |

dh\_aes.py:

תיאור: קובץ ההצפנה. Dh- קיצור של דיפי הלמן- הפרוטוקול שבעזרתו מייצרים מידע משותף וממנו מפתח. AES- פרוטוקול לתקשורת מוצפנת בעזרת מפתח משותף שנקבע לפני כן (במקרה שלנו באמצעות דיפי הלמן).

פעולות:

|  |  |  |  |
| --- | --- | --- | --- |
| חתימת הפעולה | תיאור | טענת כניסה | טענת יציאה |
| DiffieHelman\_send\_pg(sock) | פעולת פרוטוקול דיפי הלמן לצד א'. תחליט על פרמטרים ציבוריים ותשלח אותם לצד ב'. לאחר שנוצר מידע משותף תייצר ממנו מפתח משותף. | הסוקט דרכו מתבצעת התקשורת. | המפתח המשותף. |
| DiffieHelman\_recieve\_pg(sock) | פעולת פרוטוקול דיפי הלמן לצד ב'. תקבל את הפרמטרים הציבוריים מצד א'. לאחר שנוצר מידע משותף תייצר ממנו מפתח משותף. | הסוקט דרכו מתבצעת התקשורת. | המפתח המשותף. |
| add\_data\_len(data\_bytes) | מוסיפה את גודל המידע לפניו כדי שיוכלו לדעת כמה בתים לקרוא בכל חלק הודעה. | מקבלת מידע בינארי. | מחזירה את המידע כך שלפניו מופיעה מחרוזת בינארית בגודל קבוע של 4 תווים של גודל המידע שבא לאחר מכן. |
| send\_encrypted\_data(sock, key, msg) | מצפינה הודעה באמצעות AES ולאחר מכן שולחת אותה באמצעות tcp\_by\_size. | מקבלת את הסוקט עליו יש לשלוח את המידע, את המפתח ואת המידע הבינארי שיש לשלוח. | לא מחזירה כלום. |
| receive\_encrypted\_data(sock, key) | מקבלת הודעה באמצעות tcp\_by\_size, ומפענת אותה באמצעות AES. | מקבלת את הסוקט שממנו מקבלים את המידע. | מחזירה את תוכן ההודעה המקורית. |

sql\_orm.py:

תיאור: קובץ השימוש במבנה הנתונים. יכיל את כל הפעולות לגשת ולשנות את מבנה הנתונים ואת כל משפטי ה-sql הדרושים לכך.

משתנה קבוע:

AMOUNT\_OF\_RECORDS- (int) – כמות השיאים שיש לשמור לכל קטגורית זמן.

מחלקות:

שם המחלקה: User

תיאור: מחלקת המשתמש שמטרתה לשמור את כל הפרטים עליו בצורה מסודרת.

תכונות:

user\_name- שם המשתמש (str).

password\_hash- הגיבוב של הסיסמה (בינארי- טיפוס bytes).

stage- מספר השלב אליו הגיע המשתמש(int).

מתודות:

|  |  |  |  |
| --- | --- | --- | --- |
| חתימת הפעולה | תיאור | טענת כניסה | טענת יציאה |
| \_\_init\_\_( self, user\_name, password\_hash, stage) | פעולה בונה, מאתחלת את התכונות. | מקבלת פרמטרים לאתחול תכונות. | לא מחזירה כלום. |

שם המחלקה: Record

תיאור: מחלקת המשתמש שמטרתה לשמור את כל הפרטים עליו בצורה מסודרת.

תכונות:

rank- (int) דירוג השיא.

time- (int) הזמן של השיא בדקות.

stage - (int) כמות השלבים שנפתרו (בלי להיפסל 3 פעמים ולפני שנגמר הזמן).

user\_name- (str) שם המשתמש.

date- (str) התאריך של השיא בפורמט "דקות-שעות-יום-חודש-שנה".

מתודות:

|  |  |  |  |
| --- | --- | --- | --- |
| חתימת הפעולה | תיאור | טענת כניסה | טענת יציאה |
| \_\_init\_\_(self, rank, time, stage, user\_name, date) | פעולה בונה, מאתחלת את התכונות. | מקבלת פרמטרים לאתחול תכונות. | לא מחזירה כלום. |
| record\_broken(self, time, user, date) | תעדכן את התכונות הרלוונטיות לשיא החדש | מקבלת זמן, משתמש ותאריך של שיא חדש. | לא מחזירה כלום. |

שם המחלקה: UserRecordORM

תיאור: מחלקת בסיס הנתונים שמטרתה לגשת ולשנות את בסיס הנתונים.

תכונות:

conn - (קיצור של connection) החיבור לבסיס הנתונים (טיפוס (sqlite3.Connection.

cursor- מצביע החיבור לבסיס הנתונים (טיפוס (sqlite3.Cursor.

file\_location – (str) מחרוזת מיקום הקובץ של בסיס הנתונים.

מתודות:

|  |  |  |  |
| --- | --- | --- | --- |
| חתימת הפעולה | תיאור | טענת כניסה | טענת יציאה |
| \_\_init\_\_(self, file\_location) | פעולה בונה, תאתחל את conn ו-cursor ותאתחל את file\_location לפי המיקום שתקבל. | מקבלת מיקום קובץ בסיס הנתונים(str). | לא מחזירה כלום. |
| open\_db(self) | תפתח את החיבור למבנה הנתונים. | לא מקבלת כלום. | לא מחזירה כלום. |
| close\_db(self) | תסגור את החיבור למבנה הנתונים. | לא מקבלת כלום. | לא מחזירה כלום. |
| commit(self) | תבצע פעולות sql שמשנות את בסיס נתונים. | לא מקבלת כלום. | לא מחזירה כלום. |
| try\_insert\_user(self, user) | אם שם המשתמש לא קיים מכניסה את המשתמש לבסיס הנתונים. | מקבל אובייקט מסוג User של המשתמש שמנסים ליצור. | מחזירה True אם הוכנס משתמש, אחרת False (bool). |
| try\_get\_user(self, username, pass\_hash) | בודקת אם קיים משתמש עם הפרמטרים שהוכנסו. | מקבלת שם משתמש (str) וגיבוב של סיסמה(str). | תחזיר אובייקט User (אם אין משתמש מתאים לבסיס הנתונים ולפרמטרים תחזיר None). |
| increase\_stage(self, user) | בודקת האם המשתמש עבר כבר את השלב האחרון, אם לא מעלה את המשתמש שלב(מוסיפה 1 לעמודה של שלב המשתמש). | מקבלת אובייקט User של המשתמש. | לא מחזירה כלום. |
| get\_records\_of\_time(self, t) | מביאה את כל השיאים מקטגורית זמן מסוימת. | מקבלת זמן שיא בדקות(int). | מחזירה מחרוזת של הרשומות של כל השיאים של זמן מסוים (str). |
| check\_record\_broken(self, record) | בודקת אם שיא נמצא בטופ 5, ואם כן מעדכנת את טבלת השיאים. | מקבל אובייקט מסוג Records של השיא שיש לבדוק. | מחזירה True אם השיא בטופ 5, אחרת False(bool). |

stagen.txt:

stagen.txt – קובץ בצד השרת הראשי שישמור מידע על שלב. יהיו 9 שלבים כאלה(בכל עולם) כאשר n מייצג את מספר השלב 1-9.

תוכנו יהיה ה-x וה-y של כל בית בשלב.

arean.txt:

קובץ בצד תת שרת שישמור מידע על אזור משתנה. יהיו 4 אזורים כאלו בכל אחד מ4 תת השרתים כאשר n יהיה מספר משתנה בין 1-4. לדוגמה: לאזור שבו יש 2 בתים(n=2) יקרא area2.txt.

תוכנו יהיה ה-x וה-y של כל בית באזור.

how\_to\_play.txt:

קובץ בצד השרת הראשי שישמור את ההוראות ואת חוקי המשחק.

תוכנו יהיה הוראות המשחק.

# סיכום אישי / רפלקציה

העבודה על הפרויקט הייתה תהליך מצד אחד ארוך מייגע, מעייף וקשה אבל מצד שני הוא היה גם ובעיקר כיף, מספק ומלמד. קיבלתי הרבה בעבודה על הפרויקט גם מבחינת ניסיון וגם מבחינת למידת דברים חדשים.

כלים להמשך: צברתי ניסיון או/ו- ידע ב-: ניהול זמנים, חקר שוק קיים, עבודה עם הצפנה(את פרוטוקול דיפי הלמן מימשתי לבד וללא ספריות חיצוניות) והבנה איך היא עובדת, עבודה עם תמונה מבוזרת ועיבוד מבוזר-איך מנהלים כמה מחשבים באמצעות מחשב אחד ומשתמשים בזיכרון ובמשאבים שלהם, עבודה ממושכת על פרויקט במשך כמה חודשים, תכנון מוצר/פרויקט לפני כתיבת קוד וכמה התכנון חשוב, עבודה עם בסיס נתונים, טבלאות, כתיבת התחברות משתמשים הכוללת בה גם את הגרפיקה, התקשורת וההצפנה והקישור לבסיס הנתונים, עבודה עם גרפיקה בכללי, טיפול בבאגים בפרויקט גדול, עבודה על חלקים קטנים וחיבורם לתכנית גדולה, התמודדות עם קטעים/דברים שאני לא יודע לעשות באמצעות האינטרנט ובאמצעות ניסוי וטעייה.

קשיים: פעם ראשונה שאני ממש שני פרוטוקולי הצפנה שיוצרים הצפנה היברידית לגמרי בעצמי. הייתי צריך להבין איזה פרוטוקולים לבחור ואיך לממש אותם, לדוגמה בחלק של שליחת הודעה מוצפנת הייתי צריך לחקור ולהבין שיש שלושה חלקים כפלט של הפעולה של הencrypt והבנתי שאני צריך לשלוח אותם בפרוטוקול משל עצמי. היה מאתגר לכתוב מסמך כזה גדול לפרויקט, בעיקר בחלקים של התכנון, היו לי הרבה שינויים בהתחלה כי לא הייתי בטוח במאה אחוז מה אני רוצה לעשות בדיוק וגם היה קשה בחלק של העיצוב תכנה, שבו הייתי צריך לכתוב את תכנון של כל הקוד מבלי להתחיל לכתוב את הקוד עצמו, במיוחד כשיש שינויים ותקלות בדרך בפרויקט גדול. יש יחסית הרבה מסכים ומלל בפרויקט שלי אז היה קושי להדפיס את כל הפלטים למסך במקומות הנכונים וזה לקח יחסית הרבה זמן.

מסקנות: כשכותבים פרויקט או מוצר כזה יש להשקיע בתכנון כמה שיותר. החלקים שתכננתי טוב ושהתכנון בהם היה נכון נעשו בקלות יותר מהחלקים שבהם התכנון היה פחות טוב. עם זאת, חשוב להבין שזה לא פרקטי לתכנן הכל מושלם מההתחלה וזה בסדר שיש שינויים, אבל ברגע שיש שינוי בקוד יש לשנות את התכנון בהתאם. כדי לכתוב פרויקט גדול צריך להשקיע ביחידות / פונקציות / מחלקות בנפרד ולהריץ את הדברים ביחד רק אחרי שעובדים בנפרד, וגם אז חשוב לזכור שיכולים להיות באגים בשילוב בין יחידות ויש לטפל בהם.

אם הייתי מתחיל היום הייתי מנסה תוך כדי שאני חושב מה אני רוצה לעשות בתחילת הפרויקט כותב את האפיון כדי לא להיתקע וגם זה יכול לעזור להבין מה אני יכול ורוצה לעשות ומה פחות מתאים. לאחר שסיימתי את העיצוב תכנה כמעט ולא נגעתי בו עד שסיימתי כמעט את כל הקוד, מה שהיה טעות גדולה לדעתי, אם הייתי מתחיל מחדש הייתי משתדל לשנות את העיצוב תכנה לעיתים קרובות תוך כדי שאני כותב את הקוד, כך שאם יש חלק שהבנתי שתכננתי לא טוב אני יכול לתכנן אותו מחדש ואז לכתוב את הקוד שוב. זה גם היה מקל על השינויים בעיצוב תכנה כי כל שינוי בו היה מתבצע תוך כדי תהליך כתיבת הקוד של אותו חלק ולא לאחר שסיימתי לכתוב כמעט את כל הקוד של פרויקט גדול.

גיליתי על עצמי שכשיש תחום שמעניין אותי ויש לי מטרה אני לא מוכן לוותר ואני לא מפחד ללמוד ולחקור דברים חדשים שפחות מוכרים לי. תחושת הסיפוק שהדברים עובדים עצומה.

# ביבליוגרפיה

מימוש פרוטוקול AES בפיתון באמצעות פקטת pycryptodome:

<https://pycryptodome.readthedocs.io/en/latest/src/cipher/aes.html>

<https://www.youtube.com/watch?v=GYCVmMCRmTM&ab_channel=BasselTech>

עזר עם sqlite3 בפיתון:

<https://docs.python.org/3/library/sqlite3.html>

אתרים כלליים שעזרו לי עם הרבה תקלות ובעיות או דברים שלא ידעתי איך לעשות:

<https://stackoverflow.com/>

<https://www.geeksforgeeks.org/>

<https://www.programiz.com/>

<https://www.w3schools.com/>

# נספחים

פירוט על המחלקות DoubleNode ו-Dequeue שבלקוח client.py:

שם המחלקה: DoubleNode

תיאור: שרשרת חוליות דו כיוונית, מכל חוליה node אפשר לגשת לחוליה הבאה וגם לחוליה הקודמת.

תכונות:

value – ערך החוליה(טיפוס כללי/משתנה).

prev – החוליה הקודמת (אובייקט מסוג (DoubleNode.

next – החוליה הבאה (אובייקט מסוג (DoubleNode.

מתודות:

|  |  |  |  |
| --- | --- | --- | --- |
| חתימת הפעולה | תיאור | טענת כניסה | טענת יציאה |
| \_\_init\_\_(self) | פעולה הבונה שרשרת ריקה. | לא מקבלת פרמטרים. | לא מחזירה כלום. |
| \_\_str\_\_(self) | יוצרת מחרוזת של ערכי כל החוליות לפי הסדר. | לא מקבלת כלום. | מחזירה מחרוזת של כל החוליות לפי הסדר. |

שם המחלקה: Dequeue

תיאור: מבנה שיתנהג גם כתור וגם כמחסנית, כלומר יהיה אפשר להוציא ולהכניס איברים בראש המבנה וגם להוציא ולהכניס לסוף המבנה.

תכונות:

head – אובייקט מטיפוס DoubleNode שיהיה האיבר הראשון במבנה.

tail – אובייקט מטיפוס DoubleNode שיהיה האיבר האחרון במבנה.

מתודות:

|  |  |  |  |
| --- | --- | --- | --- |
| חתימת הפעולה | תיאור | טענת כניסה | טענת יציאה |
| \_\_init\_\_(self) | פעולה הבונה שרשרת ריקה. | לא מקבלת כלום. | לא מחזירה כלום. |
| \_\_str\_\_(self) | יוצרת מחרוזת של ערכי כל החוליות לפי הסדר. | לא מקבלת כלום. | מחזירה מחרוזת של כל החוליות לפי הסדר. |
| remove\_head(self) | אם התור לא ריק מוציא את האיבר הראשון במבנה. | לא מקבלת כלום. | מחזירה 'אמת' אם הוצא איבר, 'שקר' אחרת. |
| remove\_tail(self) | אם התור לא ריק מוציא את האיבר האחרון במבנה. | לא מקבלת כלום. | מחזירה 'אמת' אם הוצא איבר, 'שקר' אחרת. |
| add\_head(self) | מוסיף לראש התור איבר חדש. | לא מקבלת כלום. | לא מחזירה כלום. |
| add\_tail(self) | מוסיף לסוף התור איבר חדש. | לא מקבלת כלום. | לא מחזירה כלום. |