

‏26/4/2020

Flying Squirrel

שירות סטרימינג של משחקים

מחבר - איתי בר 213100944

מורה מנחה – עידן פלדברג וקובי שוצמן

תוכן

[**א. מבוא:** 2](#_Toc38838723)

[א.1. הגדרת הפרויקט: 2](#_Toc38838724)

[א.2. מטרת הפרויקט 2](#_Toc38838725)

[א.3. היסטוריה 2](#_Toc38838726)

[א.4. מושגים ובעיות 2](#_Toc38838727)

[**ב. מבנה / ארכיטקטורה** 3](#_Toc38838728)

[ב.1.מדוע הכול עובד איך שהוא עובד: 3](#_Toc38838729)

[ב.2.פרוטוקולים לפרוייקט: 5](#_Toc38838730)

[ב.3.תרשימי זרימה 5](#_Toc38838731)

[**ג. מדריך למשתמש** 7](#_Toc38838732)

[ג.1.התקנת הפרוייקט 7](#_Toc38838733)

[ג.2.אייך משחקים סנייק 8](#_Toc38838734)

[**ד. מדריך למפתח** 8](#_Toc38838735)

[ד.1.מחלקות 8](#_Toc38838736)

[4.ב.קבצים 9](#_Toc38838737)

[**ה. רפלקציה / סיכום אישי** 9](#_Toc38838738)

[**ו. ביבליוגרפיה** 9](#_Toc38838739)

# **א. מבוא:**

במסגרת לימודיי בתיכון אלון רמת השרון, השתתפתי בפרוייקט "גבהים" שבו לומדים מחשבים וסייבר ברמה של עשר יחידות בשיתוף עם הצבא. במסגרת זו, היה עלי לבצע פרוייקט גמר. במסמך זה אני אציג את הפרוייקט. פרוייקט הגמר שלי הוא שירות סטרימינג של משחקים המיועד למצב בו הלקוחות הינם לקוחות "רזים" Thin client)) עם יכולת מיחשוב ורשת נמוכים. קראתי לפרוייקט בשם סנאי מעופף, Flying squirrel. הפרויקט נכתב ב-Python.

א.1. הגדרת הפרויקט:

לפרויקט "סנאי מעופף" שני חלקים עיקריים:

1. שרת מרובה משתמשים – הנחת העבודה שלי היא שיש רק שרת אחד.
2. לקוחות רזים (Thin client) - הלקוחות (שהם בעצם משתמשי השרת). הלקוח מוריד תוכנה.

הפרויקט מורכב משרת מרובה משתמשים (שנקראים גם בפרויקט בשם לקוחות). כאשר לקוח מריץ את המשחק, בפועל השרת מריץ את המשחק בשבילו, ושולח לו רק תמונות. זאת כדי שלא להסתמך על יכולת המחשוב המוגבלת של הלקוח. בעיקרון, בניתי פלטפורמה כללית להרצת משחקים כלשהם על השרת, כאשר בשלב זה יש בפלטפורמה רק יישום של המשחק סנייק.

א.2. מטרת הפרויקט

המטרה של "סנאי מעופף" היא בניית שירות סטרימינג של משחקים. **שירות סטרימינג** של משחקים היא שיטה להרצת משחקים על לקוחות חלשים (בעיקר מבחינה חישובית ומבחינת יכולות גרפיות, ולפעמים גם ברוחב פס). הרעיון הינו שהשרת הרחוק עושה את כל העבודה הקשה, ולכן הלקוח יכול להריץ גם משחקים שדורשים גרפיקה טובה (ברמה גבוהה), וזאת גם כאשר הלקוח משתמש בחומרה חלשה, לדוגמא טלפונים ניידים).

א.3. היסטוריה

שירות סטרימינג של משחקים הוא רעיון ישן, כאשר הניסיונות הראשונים נעשו כבר בשנת 2000, אך הם לא נחלו הצלחה רבה. בשנים האחרונות, צצים חברות רבות שמנסות לממש את הרעיון, בעיקר כי אם חברה תצליח לעשות את זה כמו שצריך דבר זה יחליף את קונסולות המשחקים. עם זאת אף חברה לא הצליחה להתחרות עדין בקונסולות. לאחרונה, הניסיון של גוגל עם גוגל סטדיה, קיבל ביקורות מעורבות, ואולם זה נראה כמו פתרון שאולי מסמן את האפשרות שהשיטה יכולה לרקום עור וגידים ולהיות פופולרית בעתיד הקרוב.

א.4. מושגים ובעיות

**גיימינג בענן** – שירות סטרימינג נכנס תחת המושג "גיימינג בענן" Cloud gaming) או Gaming on demand או Gaming-as-a-Service). ב"גיימינג בענן" - שרת בענן מריץ את המשחק והוא משודר ללקוח. בנוסף ל***שירות סטרימינג***, אפשר להשתמש בשתי אופציות אחרות: דרך אחת היא ***הורדה פרוגרסיבית***, שבה המשחק רץ אצל הלקוח, אבל מה שנמצא אצל הלקוח הוא רק חלק קטן מהמשחק, ועדיין משתמשים בשרת לחלק מהחישובים. דרך שנייה היא להשתמש ב-***גיימינג P2P (Peer to Peer)****,* בשיטה זו השרת שמריץ את המשחק נמצא אצל חלק מהלקוחות בעלי יכולות חישוב חזקה יחסית. הרעיון בשיטה זו הינה למצוא עמית עם מחשב חזק שקרוב ללקוח ובכך להקטין את הלאג שמוגדר בסעיף הבא.

**לאג (Lag)** - עיכוב תקשורתי בין מספר מחשבים, בתקשורת נתונים, ובפרט באינטרנט. מונח יסודי ב- online gaming, ובפרויקט שלי היה מאד חשוב למזער אותו. הלאג מחושב מהרגע שלקוח לוחץ על input (לדוגמא מקש או עכבר) ועד שהוא רואה את המשחק מגיב.

הלאג מורכב מחמישה שלבים עיקריים:

1. הזמן מהרגע שלקוח לוחץ על מקש וההודעה נשלחת
2. הזמן שלוקח להודעה לעבור מהלקוח לשרת
3. הזמן שלוקח לשרת לעבד את ההודעה ולשלוח הודעה בחזרה
4. הזמן שלוקח להודעה להגיע ללקוח
5. זמן שלוקח ללקוח לעבד את ההודעה ולהציג אותה.

בפרויקט אנחנו מתמקדים בלמזער את שלב 3 (עיבוד בשרת) ושלב 5 (עיבוד אצל הלקוח). שלב-1 הוא זניח ודי קבוע. נשים לב שהשלבים 2 ו-4 תלויים בתנאי הרשת, והדבר הדומיננטי בהם הוא המרחק מהשרת (שבדרך כלל מתואם עם המרחק הגיאוגרפי עם השרת). ניתן להקטין את הזמן של שלבים 2 ו-4 ע"י מיקום השרת קרוב ללקוח (וזה מחוץ להגדרת הפרויקט).

**רוחב פס** - כמות הנתונים שניתן להעביר בקו תקשורת בזמן נתון. מכיוון שאנחנו רוצים שהמערכת תעבד גם עם רוחב הפס קטן (מערכות סלולאריות ו-wi-fi עמוסות) אזי נשתדל להקטין את דרישות רוחב הפס של התוכנית. אחרת התעבורה יכולה לגרום לאיבודי חבילות. מכיוון שהפרויקט תוכנן ב- TCP איבוד פקטות יגרום ל- latency של הרשת.

**אובדן פקטות** – אובדן פקטות מתרחש כאשר פקטות המועברות ברחבי הרשת אינן מגיעות ליעדן הסופי. מכיוון שהשתמשתי ב- TCP איבוד פקטות, מטופל ע"י שכבת ה- TCP והנזק שיגרם הוא הגדלת הלאג.

**הסיבה לבחירת הפרוייקט** - היא כי אני משחק לא מעט במשחקים ונראה לי מעניין להבין כיצד משחקים כבדים יכולים לעבוד על מערכות חלשות יחסית (כמו הסלולארי שלי). נראה לי גם מעניין לחקור נושא זה וללמוד אותו. דבר זה גם נתן לי מוטיבציה להמשיך לעבוד על הפרוייקט.

# **ב. מבנה / ארכיטקטורה**

## ב.1.מדוע הכול עובד איך שהוא עובד:

ההחלטה הראשונה שהייתי צריך לקבל היא באיזה מהסוגים של גיימינג בענן אני רוצה לבחור. בעבודה זאת אני לא שם את השרת בענן (זה לא הפוקוס של העבודה, וזה לא האתגר שלה), למרות שאין בעיה גם לקחת את השרת שתכננתי ולשים אותו בענן. במסגרת עבודה זו התרכזתי בחלק של שירות סטרימינג משרת. החלטתי לא להשתמש ב-***הורדה פרוגרסיבית***, שבה חלק קטן מהמשחק רץ אצל הלקוח. אני רציתי לטפל בלקוח רזה, שהורדה פרוגרסיבית לא מספיק טובה לו, כי היא עדין דורשת מהלקוח יכולת חישוב. כמוכן העדפתי לטפל בפתרון של שרת שמשתמש בשרות סטרימינג, ולא להשתמש ***בגיימינג P2P*** כדי להתמקד על הבעיות של תכנון השרת בצורה היעילה ביותר, הבעיה שהכי ענינה אותי.

בחירת פרוטוקול תקשורת: אני בחרתי בTCP בתור פרוטוקול משום שחשוב לי שהמשחקים יהיו באיכות תמונה גבוהה ולכן פרוטוקול התקשורת לא יכול להרוס את התמונה או לאבד פקטות ולכן פרוטוקול התקשורת לא יכול להיות UDP.

נשים לב שבארכיטקטורה מי שיוצר את הקשר הוא הלקוח לשרת. נשים לב שסיטואציה שבה השרת מאחורי NAT ולא באותה רשת עם הלקוחות, ידרוש פתרון (כדי שהשרת יכול לקבל הודעות נכנסות, לדוגמא ליצור בשבילו חור ב- NAT, דבר שמחליש את האבטחה עם אין מנגנוני הגנה).

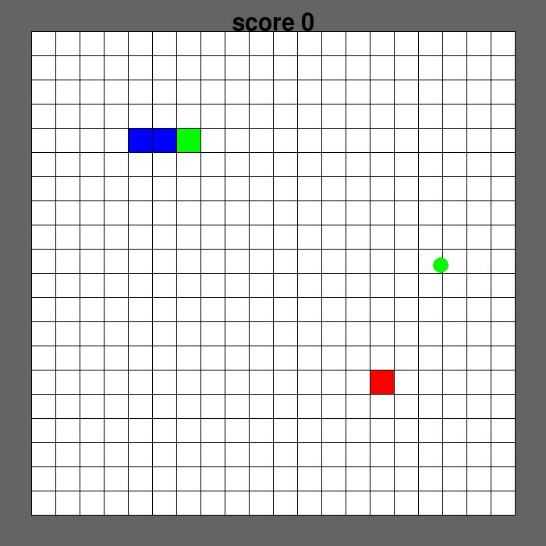
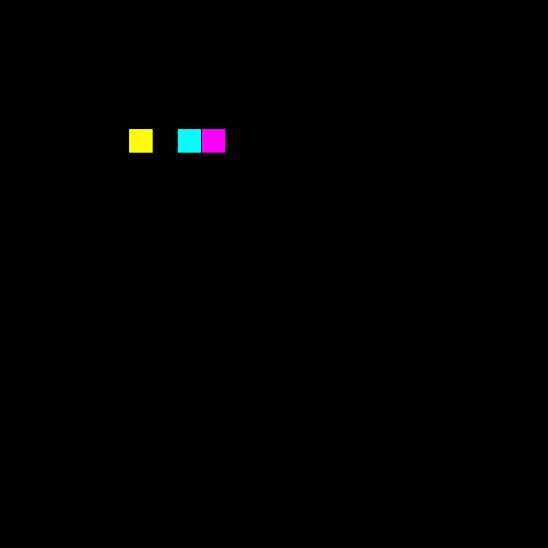
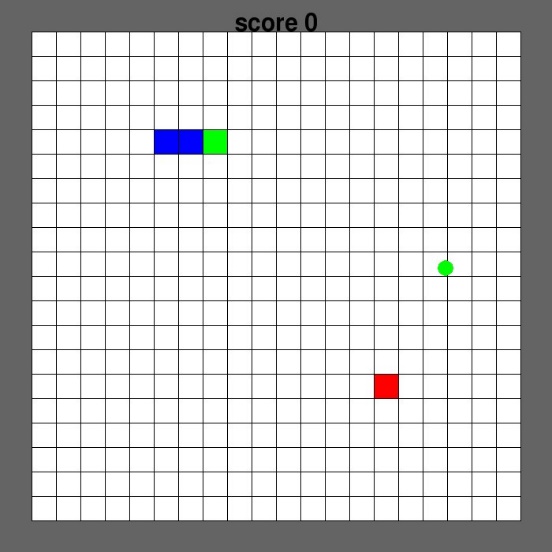
אני השתמשתי במצב שבו השרת והלקוחות נמצאים באותה רשת ביתית ולכן אני לא נדרש לטפח ב-NAT. אפשרות אחרת היא לקנות שרת (לדוגמא בענן) ולדאוג שהוא לא יהיה מאחורי NAT. לא רציתי להכנס לזה בפרויקט, אבל הפרויקט אמור לעבוד באופן זהה בסביבה כזאת.

בנוסף השתמשתי בסוקט ( socket) שעליו הגדרתי פרוטוקול הודעות שלי, דבר המאפשר לי גמישות וחופש פעולה.

לפלטפורמה בו ישוחק המשחק בחרתי בPYGAME כי זו הפלטפורמה הפופולרית ביותר לפיתון למשחקים. בנוסף, המשחק של כל לקוח ירוץ בתהליך משלו על השרת, על מנת לדאוג ליעילות מרבית ולהקטנת הזמן לביצוע פעולות בשרת וגם כי PYGAME לא מאפשר מספר PYGAME במקביל באותו תהליך.

על מנת להקטין את השימוש ברוחב הפס במקום להעביר מהשרת ללקוח את התמונות באופן רציף אני שולח רק את הפרש התמונה. בהפרש התמונה חלק גדול מאוד הינו שחור דבר שמאפשר כיווץ משמעותי של גודל הפקטה. עם זאת, תהליך זה יכול להגדיל את הלאג אם הוא לא אידאלי, לכן אפרט בהמשך על השינוי שערכתי בפרוייקט בנוגע לתהליך זה – וזאת עד למצב הנוכחי.

**התהליך**:



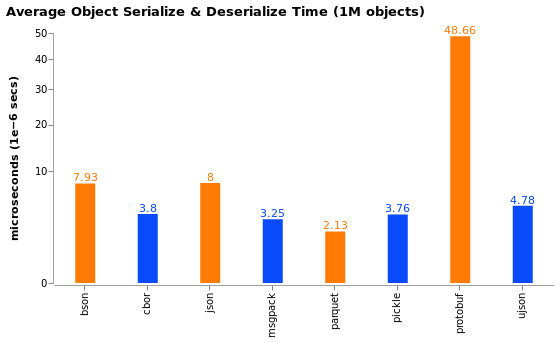
**תמונה ישנה**

**תמונה חדשה**

**הפרש בין התמונות**

הלאג המתקבל בפרוייקט הוא עד 0.1 שניה לכל פעולה – וזאת לפי המחקר שלי.

הרעיון הראשון היה שמירת התמונה בקובץ ואז לעשות חיסור תמונה מתמונה על מנת לקבל את ההפרש שגם הוא תמונה. השלב הבא היה לכווץ את תוכן התמונה של ההפרש - עשיתי זאת באמצעות המרת הקובץ לפורמט תמונה (PNG) – אשר דחס אוטומטי את התמונה בעת ההמרה (רוב פורמטי התמונות עושים כיווץ). בחרתי ב PNG שכן הייתי צריך למצוא סוג תמונה שלא מאבדת מידע במהלך הכיווץ (אחרת התמונה המוצגת תראה מוזר), לכן אי אפשר להשתמש בJPEG למשל, והחלטתי להשתמש בPNG. הדרך הזאת הייתה איטית - בערך 0.2 שניות לפעולה [עיבוד שרת ולקוח], כאשר ביחס ללקוח רזה כנראה מדובר בלאג גדול יותר כי הפעולות הללו שוות בין הלקוח לשרת. דרך זו היתה איטית בעיקר מאחר והיא דורשת גישה לדיסק במקום ישירות לשמור על הזיכרון, פעולה מיותרת העולה בזמן רב. בנוסף, פעולות ישירות על התמונה (איך שעשיתי את החיסור) הן מאוד איטיות.

כתוצאה מכך עברתי לייצג את התמונה בתור מטריצה שלושה מיימדים (700X700X3) כאשר שני הפרמטרים הראשונים מייצגים את הפיקסלים (X ו-Y) והפרמטר השלישי הוא הצבע RGB : R אדום, G ירוק וB כחול. המטריצה היא מטריצה של NUMPY, שכתובה ב-C כך שכל פעולות המטריצה ממש מהירות. הבעיה היא שאני צריך להעביר מהר ממטריצה למחרוזת כדי לשלוח את המידע. הדרך הראשונה שניסיתי היא לעשות סריאליזציה (תהליך של תרגום מבני נתונים לפורמט שניתן לאחסן אותו). אני בחרתי ב-PICKLE בתור הכלי כי הוא הכלי הכי מהיר לפייטון (ראה טבלה משמאל) ובפייטון הכי קל להשתמש בכלי זה. בנוסף, לפני השליחה עשיתי כיווץ לתמונה ב zlib הפתרון הזה היה איטי ממש בערך 0.5 שניות לפעולה (לוקח יותר זמן לעשות סריאליזציה מאשר די- סריאליזציה אבל לא בהפרש מאוד גדול אז הלאג אמיתי גדול הרבה יותר). בנוסף, עשיתי גם ניסיון עם JSON בתור פורמט הסריאליזציה אך התוצאה היתה איטית יותר כפי שהטבלה מראה שאמור לקרות.

בסופו של דבר מצאתי פונקציה של NUMPY כדי להמיר ישירות בין מטריצה לטיפוס של בייטים. כל התהליך (לא כולל שליחה) לוקח עכשיו 0.05 שניות כאשר חצי ממנו זה הזמן שלוקח למחשב לקחת תמונה מPYGAME ורוב מה שנותר זה תהליך כיווץ התמונה.

כמו כן, שיניתי את תהליך ההפרש מחיסור לXOR משום שתהליך זה יותר מהיר ונותן תוצאות באותה רמה.

אחרי כל הפעולות שביצעתי לצורך ייעול תהליך שליחת התמונה, בדקתי בכמה יעיל בכלל הכיווץ והאם היה בו צורך. התהליך לוקח תמונה בגודל 1,470,000 בייט [700\*700\*3] והופך אותו לתמונה של 2,000 בייט. כלומר, התהליך מקטין תמונה פי 1000 בערך.

## ב.2.פרוטוקולים לפרוייקט:

פרוטוקול עיבוד התמונה מפורט בשני העמודים הקודמים.

פרוטוקול תקשורת בין שרת ולקוח: כאשר לקוח מתחבר לשרת דבר ראשון הוא שולח לשרת את השם שלו ומחכה עד שהשרת ישלח את התמונה הראשונה ואז הוא פותח את הPYGAME. לפקטות המשמשות לשליחת מידע מהלקוח לשרת צירפתי את המידע ואז הוספתי עוד מילה בסוף הפקטה שמתארת מה המידע שנשלח.

יש שלוש אפשרויות למילה המצויה בסוף הפקטה: אחת: היא שהמילה בסוף היא name, ואז המידע לפני זה, זה השם של הלקוח; אפשרות שניה: היא שהמילה היא mouse ואז הכוונה היא למידע על העכבר שיש בו שינוי וpos זה מידע מהחיצים של המקלדת; אפשרות שלישית: באותו אופן גם הודעות מהשרת נגמרות ב-endend על מנת לבדוק שכל ההודעה התקבלה.

פרוטוקול נוסף הוא הגדרה של המצב מקלדת כאשר

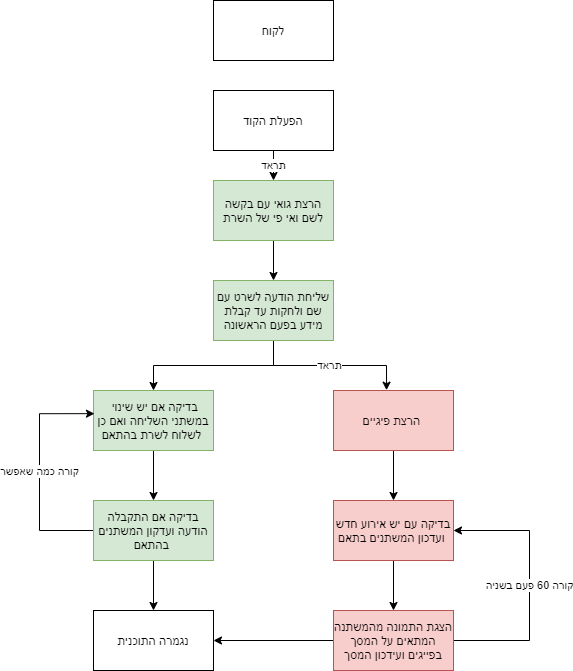
|  |  |
| --- | --- |
| מספר | מה הוא אומר |
| 0 | כלום |
| 1 | חץ ימין לחוץ |
| 2 | חץ למטה לחוץ |
| 3 | חץ שמאול לחוץ |
| 4 | חץ למעלה לחוץ |
| 1- | הלוקח לא מחובר יותר |

## ב.3.תרשימי זרימה

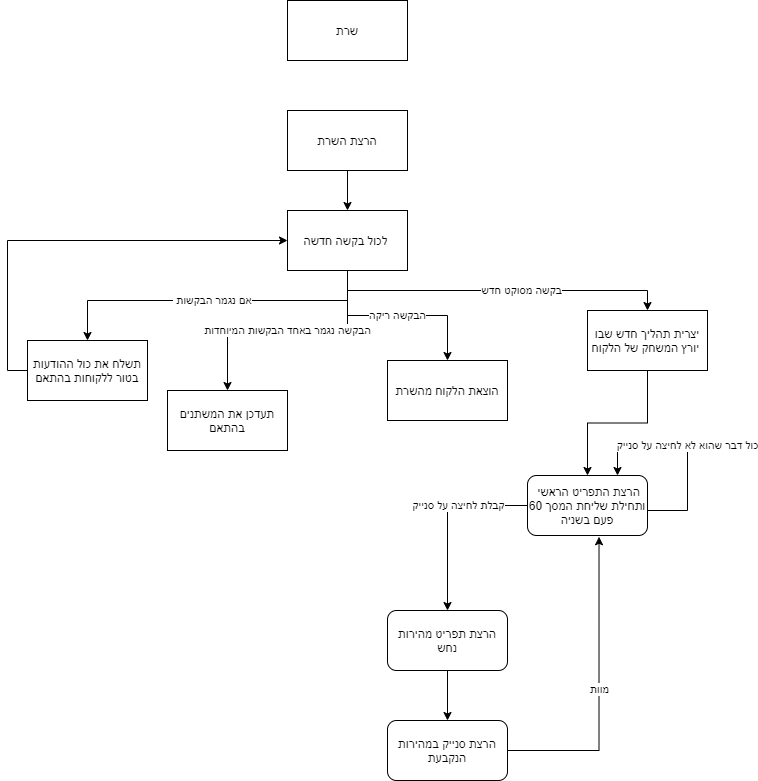
צבעים שונים מציינים טרדים שונים

צורות שונות מציינים תהליכים שונים

תרשים זרימה ללקוח:



תרשים זרימה לשרת:



# **ג. מדריך למשתמש**

## ג.1.התקנת הפרוייקט

התקנת לקוח - צריך להוריד את הפריטים client, gui\_for\_pro, config לאותה תיקיה. על מנת להריץ, יש להריץ את client. יש לכתוב שם בחלון הצץ את השם שלך ואז לרשום בחלון השחור את הIP של השרת ואז רק לבחור משחק ולשחק.

התקנת שרת - צריך להוריד את הפריטים עם המילה server וגם את IMPO וcolor לאותה תיקיה. על מנת להריץ, יש להריץ את .server ובזה סיימת את התפקיד. השרת יכול להריץ עד חמישה אנשים.

## ג.2.אייך משחקים סנייק

במשחק שולט השחקן על נחש ארוך ודק המשוטט במשטח של 20 על 20, אוסף אוכל (ריבועים אדומים), ומנסה להימנע מלהתנגש בגוף שלו. הנחש מתארך כשהוא אוכל, וכך קשה יותר להימנע מלהיתקע בגוף עצמו. השחקן שולט על כיוון תנועת ראש הנחש (למעלה, ימינה, למטה או שמאלה), ויתר גופו של הנחש עוקב אחרי ראשו. הנחש נמצא בתנועה מתמדת ולא ניתן לעצור אותו במהלך המשחק. עם השחקן פוגע בקיר עם הנחש, הנחש יוצא מהצד השני.

# **ד. מדריך למפתח**

כל הקבצים נשמרים באותה תיקיה.

## ד.1.מחלקות

color מחלקה לשרת שמטרתה היא לאכסן את הערכים של הצבעים השונים בפורייקט.

server\_prameters מחלקה לשרת ששומרת מספר משתנים כך שאני יכול לעביר אותם בין טרד לטרד. לכול לקוח יש בשרת את פרמטרים השרת שלו בתהליך שלו.

רשימת המשתנים במחלקה

|  |  |  |
| --- | --- | --- |
| משתנה | מטרה | סוג |
| Fr | מהו הפיריים רייט של המשחקים | מספר פשוט |
| Finish | אם הקוד סיים את פעולתו (נועד לסגור פעולה ב"נימוס") | בוליאני |
| Pos | המצב של המקלדת אצל הלקוח | משתנה לתהליכים |
| Mouse | המצב של העכבר אצל הלקוח | רשימה לתהליכים |
| Name | שם המשתמש של הלקוח | מחרוזת |
| Socket | הסוקט של אותו לקוח של התהליך | socket |
| Mts | טור שמחבר בין כול התהליכים שמכיל את ההודעות שצריך לשלוח. | טור לתהליכים |
| Screen | המסך של המשחק של אותו לקוח | screen |
| Lat | התמונה החדשה ביותר | מערך של פיינם מטיפוס של uint8 גודל 700 700 3 |
| Old | התמונה אחד לפני החדשה ביותר | מערך של פיינם מטיפוס של uint8 גודל 700 700 3 |

impo מחלקה לשרת ששומרת משפר משתנים בתהליך של השרת. הם כולם ממאוכסנים במילון על השרת כאשר המפתח הוא הסוקט.

|  |  |  |
| --- | --- | --- |
| משתנה | מטרה | סוג |
| Socket | הסוקט של הלקוח | socket |
| Ip | האי פי של הלקוח | מחרוזת |
| Name | השם של הלקוח | מחרוזת |
| Mouse | המצב של העכבר אצל הלקוח | רשימה לתהליכים |
| Pos | המצב של המקלדת אצל הלקוח | משתנה לתהליכים |
| Tread | התהליך של ההרצת המשחק של הלקוח | תהליך |

config מחלקה ללקוח ששומרת משפר משתנים כך שאני יכול להביר אותם בין טרד לטרד.

|  |  |  |
| --- | --- | --- |
| משתנה | מטרה | סוג |
| Pos | המצב של המקלדת | מספר שלם |
| Mouse | המצב של עכבר | רשימה של מספרים שלמים |
| Name | השם משתמש שלי | מחרוזת |
| Fin | אם הקוד סיים את פעולתו (נועד לסגור תרדים ב"נימוס") | בוליאני |
| Img | נועד ששני אנשים לא יגשו לתמונה באותו זמן | בוליאני |
| Old | התמונה הישנה | מערך של פיינם מטיפוס של uint8 גודל 700 700 3 |
| Let | התמונה החדשה | מערך של פיינם מטיפוס של uint8 גודל 700 700 3 |

## 4.ב.קבצים

client תפקידו הם להיות הלקוח. המשתנים החשובים שלו הם בconfig .

gui\_for\_pro הGUI לקבלת שם משתמש.

Server השרת שתפקידו הם לנהל את כול מה שקשור תפקוד השרת. יש בו גם את הפונקציה send\_waiting\_messages שאחרי לשלוח את ההודעות ללקוחות. משתנים חשובים mts טור לשיחת הודעות בפונקציה send\_waiting\_messages. limpo מילון שמאחסן את המחלקה impo המפתח הוא הסוקט של אותו לקוח.

server\_screen הקוד שאחרי על המסך שבו רצים המשחקים. יש בו את הלולאה המרכזית של המשחק. יש בו את הפונקציה sending\_info שדואגת לקחת את המסך והופך אותו להודעה לשליחה ושם אותו בmts.

server\_snake הרצת המשחק סנייק. יש בו את הפונקציה לציור טקסט drowtext.

server\_menu אחרי על התפריטים השונים במשחק. יש בו את הפונקציות הבאות drowtext שאחרית על ציור טקסט, botten שאחרי על ציור כפתורים. first\_sean התפריט של כול המשחקים. snake\_speed התפריט של המהירות של הסנייק. black\_screen מסך שחור.

# **ה. רפלקציה / סיכום אישי**

העבודה על הפרוייקט לטעמי הייתה קשה ומתגרת אך כיפית באותו הזמן לעיתיים רבות.

למדתי רבות בפרוייקט בנושאים רבים ומגוונים למשל על תכנות מרובה תהליכים, עיבוד תמונה ועוד. נושאים שלא יצא להתעסק בהם קודם ולמדתי אותו כמאת מאפס בשביל הפורייקט.

הקשיים המשמעותיים ביותר שנתקלתי בהם בפרוייקט הם למידה של נושאים חדשים מאפס ומציאת חומר טוב לקרוא עליהם לרוב שהייתי צריך עזרה במציאת חומר נעזרתי באנשים מחבל הוראת הסייבר בבית הספר וחיפשתי באינטרנט. בעיה נוספת למשך כמה זמן היה לי בעיות עם הסביבה של הפייטון דבר שפגע במשך שבועיים ביכולת שלי להתקדם עם הפרוייקט. אתגר משמועותיי שהיה לי בפרוייקט היה מציאת הפרוטוקול הטוב ביותר להעברת התמונה בין השרת ללקוח כפי שפורט. אני חיפשתי מלא דרכים שונות לעשות עיבוד תמונה כך שיהיה לי יכולת למצוא את מה ששונה כמה שיותר מהר לקח בין מעבר למעבר בשינוי פרוטוקול בין 10 – 20 שעות של חיפוש מידע ומניעת בגים וטסטים רבים.

לא הייתי עושה כמעט שום דבר אחרת במחשבה לאחור אולי מתחיל לעבוד יותר מוקדם על הפרוייקט ואך כנראה הייתי מוסיף עוד פיצ'רים ומשחקים לפרוייקט.

אני רוצה גם להגיד תודה לצוות מגמת סייבר על עזרה שלהם עם הפרוייקט.

# **ו. ביבליוגרפיה**

<https://www.ynet.co.il/articles/0,7340,L-5287207,00.html>

<https://medium.com/@shmulikamar/python-serialization-benchmarks-8e5bb700530b>

<https://app.diagrams.net/>

<https://www.digitaltrends.com/gaming/cloud-gaming-vs-console-gaming/>

<https://www.youtube.com/watch?v=WOH7hDXrfwc>

# **נספחים**

אחד מהרעיונות שלי לפרויקט שלא בוצע מבחנת זמן הוא ליישם את המושג של לאג שלילי לחלק מהמשחקים.