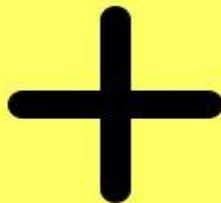




AshiAsh Prodcutin inc Presents:



Quick, Skribble!

מאת: גיל אשר
ת.ז: 214005746
שם המורה: ניר סליקטר
תאריך הגשה: 03.06.2021

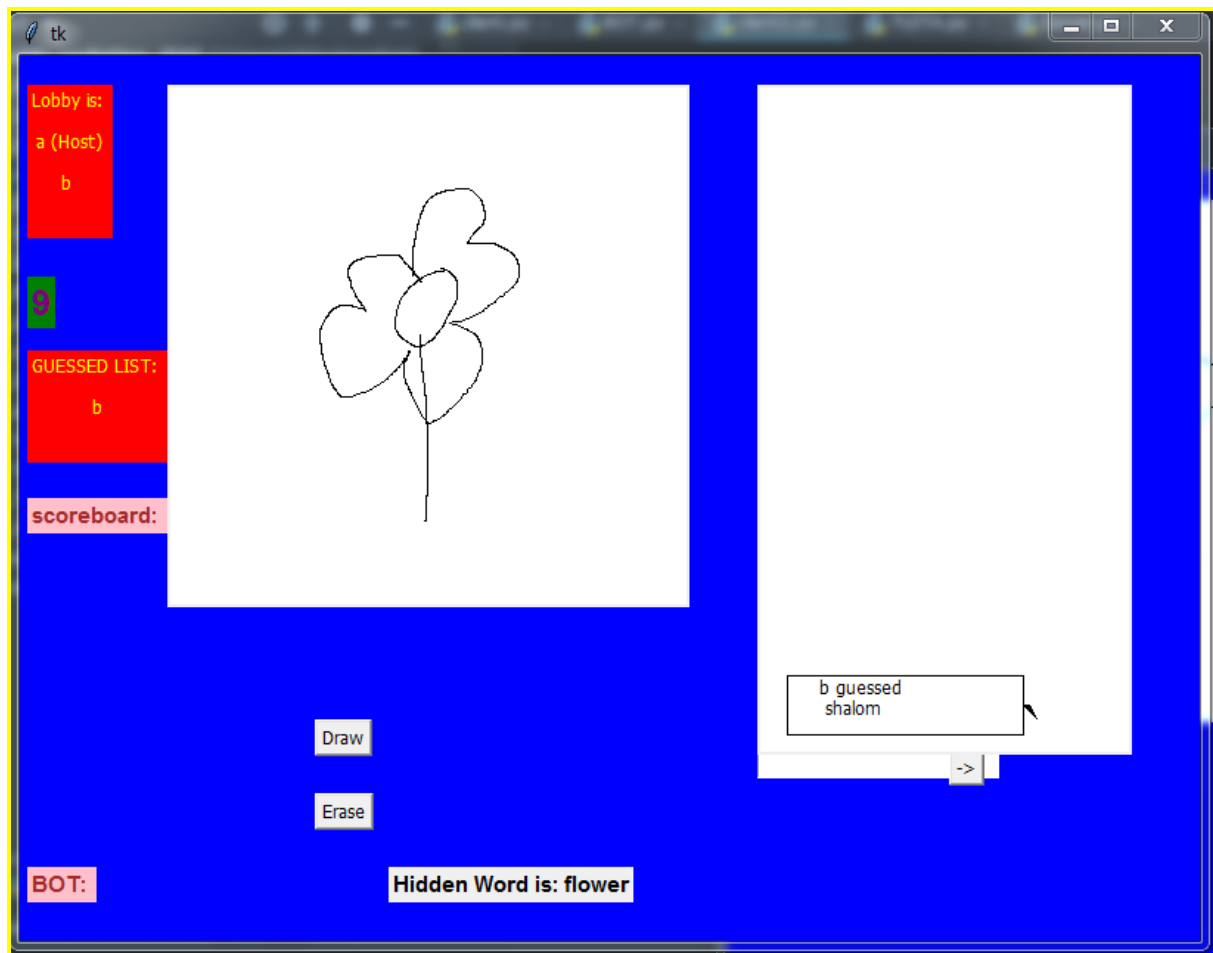
תוכן עניינים:

5	מבוא
5	קישור לקוד הפרויקט ב-GitHub
5	תיאור תכולת ספר הפרויקט
6	הרקע לפרויקט
6	תהליך המחקר
6	סקירת המצב הקיים בשוק
6	החידושים בפרויקט
6	למידה לפרויקט
7	אתגרים מרכזיים
7	הבעיות אשר נאלצתי להתמודד עימן
7	מוטיבציה לעבודה
7	הצורך עליו הפרויקט עונה
7	הסבר על machine learning
8	ארכיטקטורה של הפרויקט
9	הסבר על כל יחידה
9	שרת Server (הערה: פירוט על ההודעות נמצא במדריך למפתח)
9	לקוח Client
9	בוט BOT
12	General Use Case Diagram
13	הצגת המקרים (Use cases) עבור הפונקציות העיקריות בפרויקט
13	בחירת כינוי - Nickname picking
13	הצטרפות ללובי - Lobby joining
13	התחלת המשחק - Start the game
14	ציור ומחיקה - Draw and Erase
14	צפייה במצויר על הלוח וניחוש - Guess and watch the doodle
14	הכרזת מנצח/ים - Announce winner
15	הצגת הפתרון המוצע והסיבות לבחירתו
15	טכנולוגיות בהן נעשה שימוש בפרויקט

15	sockets סוקטים
15	מודל Machine Learning
16	מדריך למשתמש
16	הוראות התקנה
17	ספריות בפרויקט
19	היררכיית מסכים
20	תרשים מסכים
20	מסך חוקים
21	מסך בחירת הכינוי
21	מסך הכניסה ללובי - Join Lobby Page
21	לובי המנהל - Host Lobby Page
22	לובי המתנה רגיל - Regular Lobby Page
23	מסך המשחק - Game Page
24	מסך הכרזת המנצח - Winner Announcement Page
25	מדריך למפתח
25	הסבר כללי על קוד הפרויקט
25	הודעות במערכת
28	Threading
30	תרשים Threads של Servern
30	תרשים Threads של הBOT
31	קישור לקוד הפרוייקט בGitHub
31	קובץ הBOT
36	קובץ הServern
43	קובץ הClientn
56	קובץ class_names
56	תיקיית model
56	קובץ הML
56	תיקיית Data
56	קובץ הDB_installern
57	רפלקציה

57	תחושתי מהעבודה על הפרויקט
57	כלים שקיבלתי ואקח איתי להמשך
57	אתגרים שעמדו בפניי במהלך הפרויקט
57	מסקנות מהפרויקט
57	מה הייתי עושה אחרת אילו הייתי מתחיל היום
58	מה היה יכול להפוך את עבודתי ליעילה יותר
58	תכונות שהייתי רוצה להוסיף לפרויקט

מבוא:



קישור לקוד הפרויקט בGitHub:

<https://github.com/gilasher5103/Quick-Skribble->

תיאור תכולת ספר הפרויקט:

ספר הפרויקט שלפניכם יציג את פרויקט הסיום שלי – skribble.io. בשילוב עם Machine learning הספר כולל את הרקע לפרויקט, מטרתו וקהל היעד שלו. כמו כן, נמצאים בו תרשים הפרויקט, מדריך למשתמש במערכת זו וצילומי מסך של המתרחש בפרויקט. בנוסף לכך, קיימים בו מדריך למפתח ולבסוף גם רפלקציה אישית על הנעשה.

הרקע לפרויקט:

המשחק המקורי skribble.io תמיד היה בילוי מרגש עבורי, בנוסף המשחק Quick, draw! גם הוא היה מאוד חוויתי בשבילי לכן קיבלתי החלטה שהפרויקט שלי במגמה יהיה שילוב של השניים.

לאחר שעשיתי את הפרויקטים ב-java (black jack) ובאסמבלי (משחק חידתי) הבנתי שאני נהנה מאוד מתכנות של משחקים ועשייה של הדברים שאני אוהב, לכן מיד היה לי ברור שהפרויקט הבא שלי יהיה שילוב של שני המשחקים שאני אוהב.

הפרויקט כולל בחירה של שם על ידי כל אחד מן המשתמשים ולאחריו כניסה ללובי המשחק, כאשר מנהל הלובי (HOST) מחליט להתחיל את המשחק (עם או בלי BOT) מתחיל במקביל המשחק אצל כל אחד מן השחקנים, ומתחילה רוטציה בה כל אחד מן השחקנים מצייר משהו בתורו ושאר השחקנים רואים את הציור ומנחשים מה מצייר, תוך שBOT אשר יודע לזהות תמונה באמצעות Machine Learning מתחרה גם ומשחק תפקיד של מנחש בלבד. בסוף שלושה סבבים שבכל אחד מהם כל אחד מהשחקנים צייר משהו לשאר הקבוצה, נקבע מנצח על פי מי שצבר הכי הרבה נקודות לאורך הסבבים, (נקודות נצברות על ידי ניחושים נכונים של הדבר המצייר ועל פי הזמן בו לקח לשחקן לנחש).

תהליך המחקר:

סקירת המצב הקיים בשוק:

מחוויתי האישית, שני המשחקים (Quick, draw, skribble.io). שניהם מאוד מוצלחים אך לא ראיתי דוגמה שמשלבת בין השניים כאשר המשחק מבוסס על skribble.io אך משולב בו BOT שמנחש את המצייר בדומה ל Quick, draw! כמובן שעם המשאבים שברשותי אין לי כוונה להתחרות בחברות האלו אך הרעיון של לעשות משהו משלי סקרן אותי מאוד.

החידושים בפרויקט:

למדתי המון בשביל פיתוח הפרויקט. הדבר היחיד אשר ידעתי לפני שהתחלתי היה כיצד לשלוח הודעות מהלקוח לשרת ובחזרה. במהלך הפרויקט, יצא לי להשתמש בthreads על מנת להפעיל תהליך בפרויקט במקביל כגון טיימר, והפעלת שני sockets במקביל אשר נמצאים אצל השרת ואצל הלקוח (לכל לקוח יש שני sockets ולכל שרת יש שני sockets) בנוסף יצא לי לעבוד עם GUI בפייתון לראשונה והיה עליי ללמוד איך להשתמש בcanvas ולצייר לפי העכבר, כמובן שעל מנת ליצור את הBOT היה עליי ללמוד גם בתחום Machine learning.

למידה לפרויקט:

לפרויקט שלי הייתה כרוכה קודם כל בהרחבת הידע שלי ברשתות שכן הפרויקט שלי מתבסס ברובו על רשתות, כמו כן הייתי צריך להרחיב את הידע שלי בתחום threading, על מנת ליצור את המצב בו שני sockets רצים במקביל. בנוסף היה עליי ללמוד המון בתחום הGUI ועל הספריות והעצמים המוכללים בתוכו, ובנוסף להכל היה עליי ללמוד את תחום ה Machine Learning שכן לא היה ניתן ליצור BOT מזהה תמונה בלעדיו.

אתגרים מרכזיים:

הבעיות אשר נאלצתי להתמודד עימן:

להפעיל את מודל Machine learning שיצרתי היה אחד האתגרים הגדולים שיצא לי להתמודד עמם בפרוייקט, ביחד עם הקושי שלקח לי בשמירת התוכן של הקנבס עליו מציירים בכל רגע נתון, והמרתו מהפורמט PS (הפורמט ברירת מחדל בו הוא נשמר) ל JPEG (הפורמט אשר מתאים למודל), בנוסף שימוש בשני sockets במקביל (שניים בכל שרת ושניים בכל לקוח) היה אתגר נוסף איתו הייתי צריך להתמודד.

מוטיבציה לעבודה:

רוב המוטיבציה שלי לעבוד התקיימה בזכות שיעורי הסייבר המהנים עם חברי לכיתה ובזכות החלוקה להגשות קטנות פעם בכמה זמן, שבזכותה ידעתי שעליי להתחיל לעבוד בכדי להספיק לעמוד בכל המטרות שהצבתי לעצמי בין הגשה להגשה.










הצורך עליו הפרוייקט עונה:

הפרוייקט עונה בעיקר על הצורך להעביר את הזמן במשחקים קלילים אך מאתגרים נגד חברים או מול אנשים, בכדי לצבור את מספר הנקודות הרב ביותר. קהל היעד של הפרוייקט הינו ילדים, נוער ומבוגרים תחרותיים אשר מבליים את זמנם במשחקי מחשב.

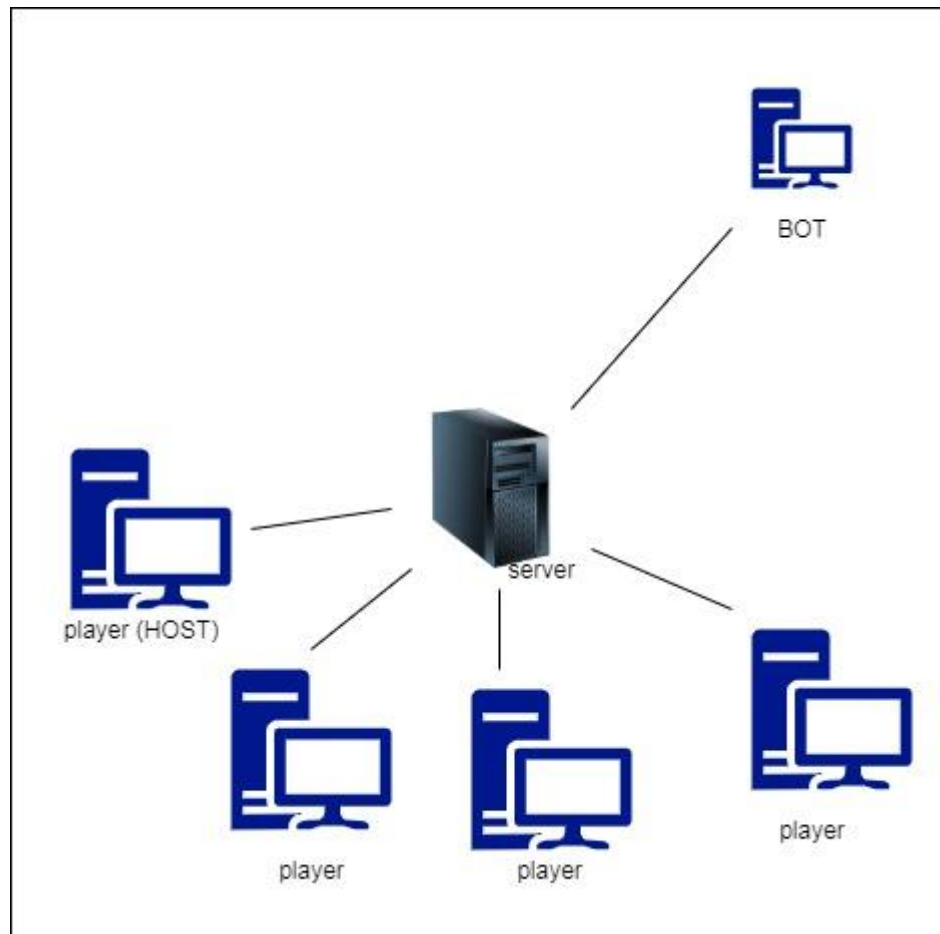
הסבר על machine learning:

למידה מכונה היא תת תחום במדעי המחשב ובבינה מלאכותית אשר עוסק בפיתוח אלגוריתמים המיועדים לאפשר למחשב ללמוד מתוך דוגמאות ופועל במגוון משימות חישוביות בהן התכנות הקלאסי הוא לא אפשרי.

בפרוייקט שלי התחום בא לידי ביטוי תחילה ביצירת המודל מזהה התמונה דבר אשר נעשה על ידי מציאת database מתאים באינטרנט והורדתו למחשב, שמירתו בתיקיית data (קורה באמצעות הקובץ DB_installer2) ולימוד המכונה לזהות את התוכן המצויר בדוגמאות המוזנות לו ב databases בסיום תהליך זה נשמר המודל בתוך תיקיית model עליה ארחיב בהמשך והתיקייה הזאת שומשה בקובץ ה BOT על מנת לזהות בהמשך את המצויר בקנבס ולשלב את ה BOT מזהה התמונה בתוכנית.

	gilasher5103 Update BOT.py	451d4e0 39 minutes ago	9 commits
	BOT.py	Update BOT.py	39 minutes ago
	DB_installer2.py	Add files via upload	1 hour ago
	ML2.py	Add files via upload	2 days ago
	Server.py	Update Server.py	42 minutes ago
	client.py	Update client.py	41 minutes ago
	model.zip	Add files via upload	2 days ago
	trophy.jpg	Add files via upload	2 days ago
	ת.תיק פרויקט - גיל אשר	Add files via upload	yesterday

ארכיטקטורה של הפרויקט:



אופי התקשורת: כל שחקן שולח בקשה להתחבר לסרבר כאשר הוא מופעל והסרבר מאשר את החיבור, וכאשר שחקן מבצע פעולה מסויימת במשחק, נשלחת הודעה מתאימה אל הסרבר, שבהתאם לצורך שולח הודעות לשאר השחקנים המחוברים, השחקן הראשון שמצטרף ללובי המשחק מוגדר כמנהל (HOST) בנוסף לשחקן המנהל יש האפשרות להחליט האם יהיה BOT במשחק הקרוב, ובמידה והוא מחליט להוסיף את ה-BOT למשחק, נשלחת הודעה לסרבר שמעדכן את כל השחקנים שיש BOT במשחק, ומריץ באופן אוטומטי את קובץ ה-BOT, שני הsockets של BOT מתחברים באופן אוטומטי לשני הsockets של השרת, ומהשלב הזה התקשורת בין ה-BOT לשרת קוראת באופן אוטומטי לחלוטין ללא התערבות גורם חיצוני.

הסבר על כל יחידה:

שרת Server: (הערה: פירוט על ההודעות נמצא במדריך למפתח)

השרת מקבל את כל ההודעות מכל הלקוחות, ושומר ומנהל את הנתונים הרלוונטים למשחק, אחראי על הרצת הBOT והתקשורת איתו.

לקוח Client:

הלקוח הוא הplayer בתרשים ומייצג כל משתמש של האפליקציה. הוא שולח בקשות, ומקבל תשובות מהשרת. כל לקוח יכול לראות את כל הניחושים בקנבס שמציג את הניחושים, וכל לקוח יכול לראות מה מצוייר ולצייר בתורו. אחד הלקוחות יקבל את התואר HOST ולו תהיה שליטה על מתי מתחיל המשחק עבור כל השחקנים בלובי, והבחירה אם יתווסף BOT למשחק הינה בידי.

בוט BOT:

יחידת הבוט היא יחידה אופציונלית אשר מכילה את מודל machine learning ורצה אך ורק אם נשלחה בקשה לשרת על ידי הHOST, וכאשר מתקבלת הבקשה היא מתחילה תקשורת עם השרת אשר מתרחשת באופן אוטומטית, תפקידה במשחק הינו לקבל כל פרק זמן קבוע בקשה מן השרת לשמור את המצויר באותו רגע על הקנבס כתמונה ולשלוח ניחוש של הBOT בנוגע למה מצויר בה.

אלגוריתמים עיקריים:

במשחק שלי קיימים שני אלגוריתמים עיקריים- משחק עם BOT משחק ללא BOT.

שני המשחקים הינם דומים בכל המאפיינים שלהם מלבד העובדה שאופן הציור על הקנבס במשחק עם הBOT שונה מאופן הציור על הקנבס בלעדיו, (יורחב על כך בהמשך) בשני המשחקים כאשר מתחיל המשחק מופיע מסך משחק אשר מכיל את הקנבס, את שעון המשחק, את לוח התוצאות, את הרשימה המייצגת מי ניחש נכון עד כה, את הרשימה המציגה את הלובי (שמות השחקנים), קנבס נוסף עליו נרשמים הנחושים של המנחשים ותיבת טקסט וכפתור אליה מוזנים הניחושים, בנוסף נמצאים שני כפתורים המחליפים בין מצב ציור למצב מחיקה עבור השחקן המצייר.

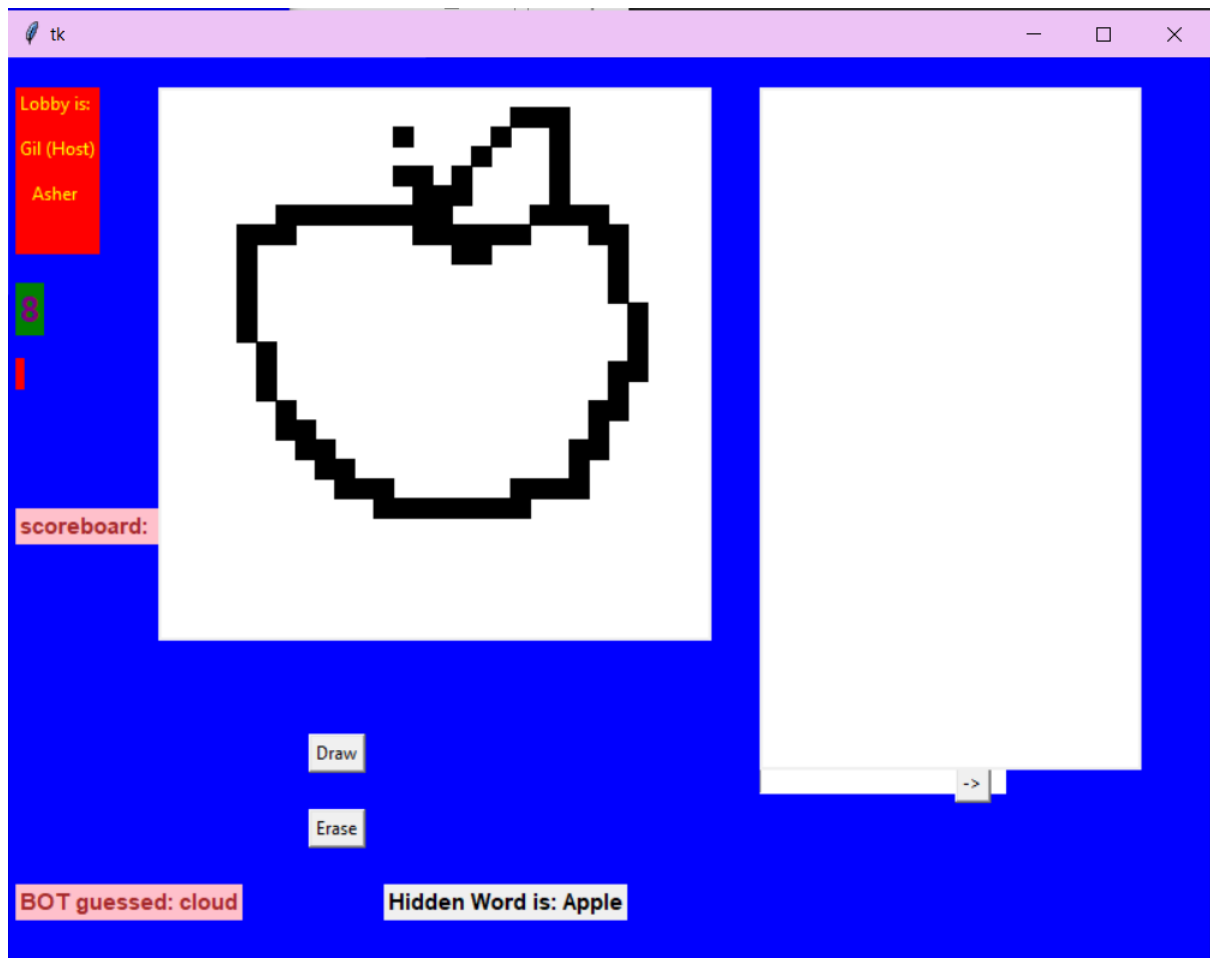
לאחר מכן, הסרבר מעדכן את השחקן כשמגיע תורו- התורות הם לפי סדר הכניסה ללובי. כאשר מגיע תורו של שחקן, נוצרים עבורו כפתורי הציור והמחיקה, ומתחילים במיידית את זמן תורו, וכמובן מופיעה לו במסך המילה אותה הוא צריך להעביר אל המנחשים. השחקן לוחץ על הקנבס בנקודה מסוימת, והיא מיד מועברת אל הסרבר שמפיץ אותה לשאר השחקנים ולבוט (אם הוא במשחק) והנקודה מצויירת בהתאם על הקנבס של השחקנים המנחשים. בתום הזמן של התור נעלמים למצייר הקודם כפתורי הציור והמחיקה והם מופיעים אצל השחקן הבא שמצייר לכולם וכך הלאה, עד שכל שחקן מצייר שלוש פעמים. בזמן הציור של כל שחקן ישנה אפשרות לשאר השחקנים לשלוח ניחושים דרך תיבת הטקסט, שנשלחים אל הסרבר ודרכו מופצים לכל שאר השחקנים (אם הסרבר מגלה שהניחוש הוא נכון, אז הוא מוסיף אותם לרשימת האנשים שניחשו ללא חשיפה של הניחוש עצמו).

כאשר השחקן האחרון מסיים לצייר את סיבובו השלישי, מוכרז\מוכרזים המנצחים לפי מי שצברו הכי הרבה נקודות לאורך המשחק.

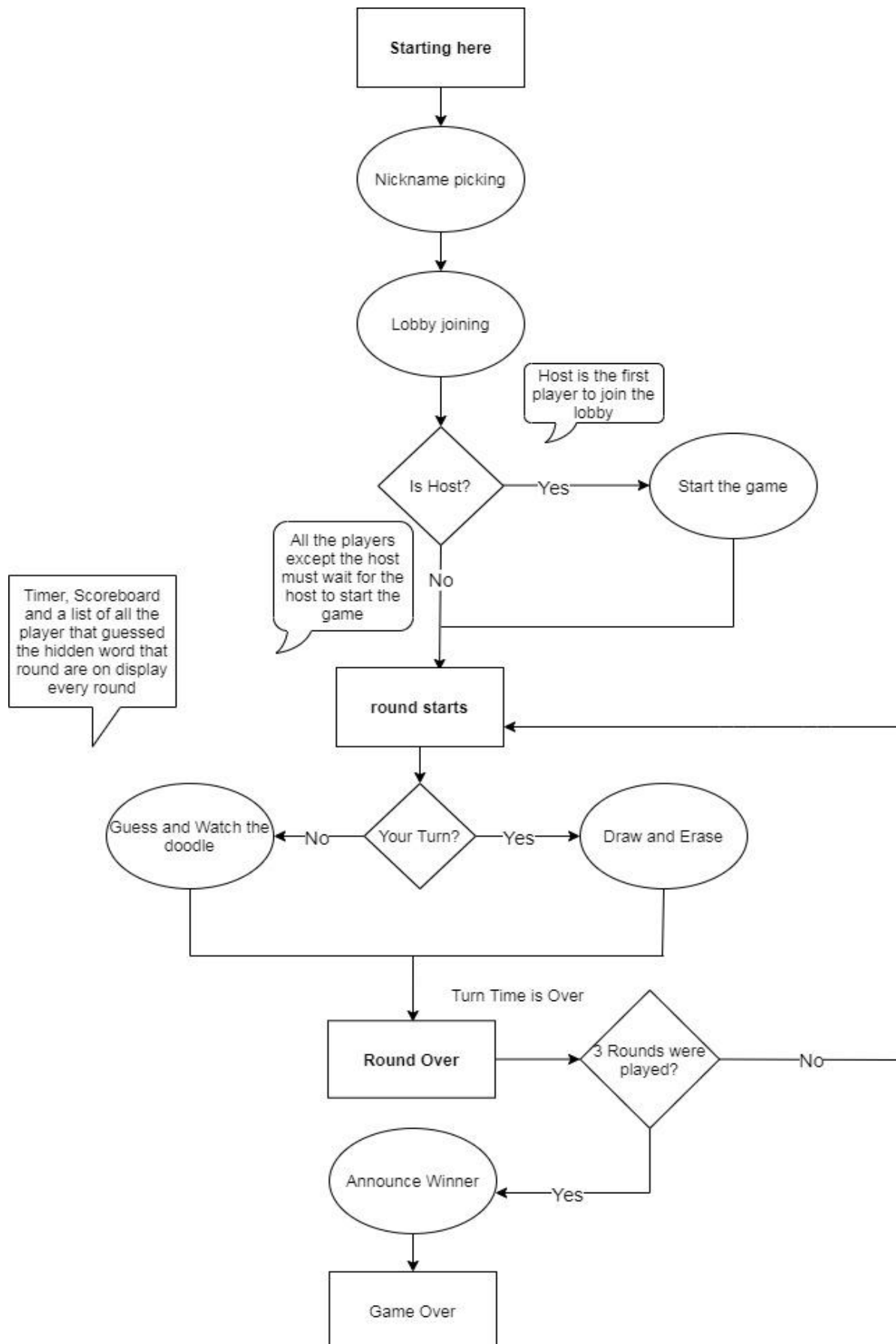
הבדלי הציור בין המשחק עם הBOT והמשחק ללא הBOT: כאשר אחד השחקנים משחק את תפקיד המצייר ולא המנחש במשחק ללא הBOT ולוחץ על נקודה בלוח במטרה לצבוע או למחוק אותה נשלח הפיקסל עליו הוא לחץ לשאר השחקנים והנקודה מצויירת או נמחקת (מצויירת בלבן) גם אצלם, אך כאשר משחקים את המשחק עם הבוט עלינו להתאים את הציור המצויר לתמונות איתן עובד מודל הmachine learning לכן עלינו ליצור תמונה המורכבת מ28 על 28 פיקסלים (איכות פחות טובה), על מנת לעשות זאת חילקנו את הקנבס ל28 על 28 ריבועים, וכאשר מצייר או נמחק אחד הריבועים נשלח המקום בו צריך להיות מצויר ריבוע בשאר הקנבסים ולא המקום של הנקודה בה לחץ העכבר, לאחר מכן אצל כל אחד מן השחקנים ואצל הבוט מצויר המלבן על פי ההנחיות שנשלחו אליו מן הסרבר.

כל שנייה שעוברת הסרבר מעדכן את שעוני המשחקים אצל כל אחד מן השחקנים, וכאשר שעון המשחק מגיע ל0 הסרבר מאפס אותו, מודיע לשחקן הנוכחי על סוף התור שלו (וכך גם נעלמים כפתורי הציור והמחיקה של המצייר), ומעדכן את השחקן הבא שתורו התחיל (וכך נוצרים במסכו כפתורי הציור והמחיקה), הוא מיד מגריל מילה חדשה שאותה יצטרכו השחקנים לנחש, ושולח להם הודעה על מנת לנקות את הקנבסים של השחקנים.

תמונה ממשחק עם BOT:



General Use Case Diagram:



הצגת המקרים (Use cases) עבור הפונקציות העיקריות בפרויקט:

בחירת כינוי - Nickname picking:

- תיאור הפעולה: המשתמש בוחר לעצמו את הכינוי שלו למשחק.
- תנאים מקדימים: המשתמש הפעיל את האפליקציה, יש לו חיבור פעיל לאינטרנט והוא נכנס למסך בחירת הכינוי.
- תנאי סיום: המשתמש שמור אצל השרת יחד עם הכינוי שלו.
- שלבי פעולה:
1. המשתמש נכנס לבחירת הכינוי.
 2. המשתמש ממלא את השדה הריק בהתאם לכתוב על הכפתור ליד השדה ("Enter your nickname:").
 3. לאחר לחיצה על כפתור ה – Enter your nickname נשלחת הודעה לשרת המעדכנת אותו בפרטי הלקוח החדש (כינוי socket) והמשתמש יעבור למסך Lobby.

הצטרפות ללובי - Lobby joining:

- תיאור הפעולה: המשתמש מנסה להצטרף ללובי המשחק.
- תנאים מקדימים: המשתמש הפעיל את האפליקציה, יש לו חיבור פעיל לאינטרנט והוא נכנס למסך ה-Lobby.
- תנאי סיום: המשתמש מצטרף ללובי.
- שלבי פעולה:
1. השחקן לוחץ על כפתור ההצטרפות ללובי ("join lobby").

התחלת המשחק - Start the game:

- תיאור הפעולה: השחקן המנהל (Host) מתחיל את המשחק עבור על השחקנים ומחליט האם להוסיף למשחק BOT.
- תנאים מקדימים: המשתמש הפעיל את האפליקציה, יש לו חיבור פעיל לאינטרנט והוא נכנס למסך ה-Lobby ראשון (כך הוא קיבל את תפקיד הHost).
- תנאי סיום: השחקן המנהל מתחיל את המשחק לאחר בחירה האם להוסיף BOT למשחק.
- שלבי פעולה:
1. השחקן מחליט עם לסמן או לא את תיבת הסימון (Checkbox) שאחראית על הוספת הבוט ("ADD BOT").
 2. השחקן לוחץ על כפתור התחלת המשחק ("start the game").

ציור ומחיקה - Draw and Erase:

תיאור הפעולה: השחקן מצייר ומוחק על הקנבס.
תנאים מקדימים: המשתמש הפעיל את האפליקציה, יש לו חיבור פעיל לאינטרנט והוא נכנס למסך ה-משחק ותורו להיות השחקן המצייר.
תנאי סיום: השחקן מצייר על הקנבס והדבר משותף עם כל השחקנים.
שלבי פעולה:

1. השחקן בוחר האם למחוק או לצייר בעזרת לחיצה על כפתורי Draw וErase.
2. השחקן לוחץ על הקנבס עם העכבר שלו ומצייר או מוחק על המסך.
3. השרת מעביר את המיקום בו צייר השחקן לשאר השחקנים והBOT במידה והוא במשחק והדבר מצוייר גם אצלם. (במידה ואין BOT במשחק השרת מעביר את הנקודה בה לחץ השחקן המצייר לשאר השחקנים, במידה ויש BOT במשחק השרת מעביר לשאר השחקנים את המיקום בו צריך להצבע מלבן על פי פרוטוקול קבוע מראש)

צפייה במצויר על הלוח וניחוש - Guess and watch the doodle:

תיאור הפעולה: השחקן צופה במצויר על הקנבס ומנחש מה מצויר בו.
תנאים מקדימים: המשתמש הפעיל את האפליקציה, יש לו חיבור פעיל לאינטרנט והוא נכנס למסך ה-משחק ולא תורו להיות השחקן המצייר.
תנאי סיום: השחקן שולח ניחוש אחרי צפייה בציור והניחוש נשלח לשאר השחקנים.
שלבי פעולה:

1. השחקן צופה במצויר על הקנבס במסך המשחק.
2. השחקן ממלא את השדה הריק בניחוש שלו.
3. השחקן לוחץ על הכפתור ליד תיבת הטקסט על מנת לשלוח את הניחוש שלו.
4. ניחוש השחקן מתפרסם על קנבס הניחושים במידה והניחוש שגוי ובמידה והניחוש נכון השחקן מצטרף לרשימת השחקנים שניחשו נכון באותו סיבוב (ותיבת הטקסט של הניחושים הופכת ללא זמינה עד תום הסיבוב).

הכרזת מנצח/ים - Announce winner:

תיאור הפעולה: מוצג על המסך מנצחי המשחק
תנאים מקדימים: המשתמש הפעיל את האפליקציה, יש לו חיבור פעיל לאינטרנט ועברו שלושה סיבובי משחק.
תנאי סיום: נפתח מסך המציג את השחקן/ים שצברו הכי הרבה נקודות לאורך הסיבובים.
שלבי פעולה:

1. נפתח המסך המציג את המנצח/ים.
2. התוכנה נסגרת והמשחק מסתיים.

הצגת הפתרון המוצע והסיבות לבחירתו:

רציתי ליצור משחק skribble.io אשר משולב עם המשחק quick, draw! לשם כך היה עליי ליצור משחק שעובד עם רשתות משתמש בsocket על מנת לתקשר בין השחקנים לסרבר ובין הסרבר לBOT, על מנת ליישם את עקרונות המשחק Quick, draw! במשחק היה עליי להשתמש במודל machine learning שילמד לזהות תמונות של כ-100 מילים אשר נמצאות במאגר.

בנוסף על מנת לשמור את התמונה היה עליי להשתמש בספריה ghostscript דבר שבו נאלצתי לעשות כי לאחר שעות של חיפושים הגעתי למסקנה שהיא הפתרון היחיד שבר יישום בחומרה שבה אני עובד.

על מנת ליצור את מודל machine learning היה עליי להוריד את הספריות tensorflow keras ו-PIL דבר אשר אילץ אותי להחליף את גרסת הפייתון איתה אני עובד ולעבור לעבודה על מחשב עם מערכת הפעלה של 64 bit. בחרתי בספריות אלו כי מצאתי אותן הכי נוחות לשימוש והן הוכיחו את עצמן כיעילות ושימושיות.

טכנולוגיות בהן נעשה שימוש בפרויקט:

סוקטים sockets:

העצם socket מהווה נקודת קצה לשליחת וקבלת נתונים ברחבי הרשת. socket קיים ופעיל רק בתוך process שבו הוא רץ. השימוש בטכנולוגיה זו היה נורא נוח והגיוני עבורי מכיוון שהתחום נלמד השנה ושנה שעברה במגמה.

מודל Machine Learning:

תחום Machine Learning הינו השיפור של תוכנה והלימוד שלה תוך ניסוי ואימון של התוכנה באמצעות מאגרי מידע.

השימוש בתחום הזה היה האפשרות היחידה על מנת ליצור ב-BOT מזהה תמונה, והשימוש בספריות tensorflow keras ו-PIL היו קלות לפיצוח ושימוש ולכן בחרתי להעזר בהן.

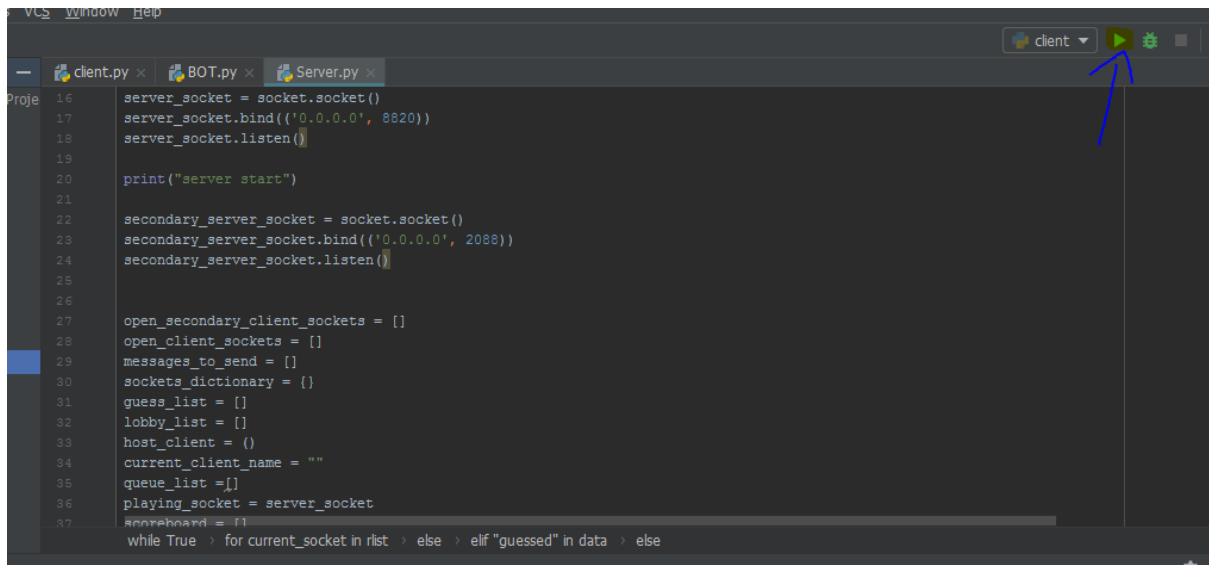
מדריך למשתמש:

את המשחק שלי ניתן להריץ על כל מחשב אשר בעל מערכת הפעלה של 64 סיביות.

הוראות התקנה:

1. להתקין python 3.8 x64 למחשב-
/https://www.python.org/downloads/release/python-386
2. להוריד את הספריה keras, tensorflow ו PIL באמצעות הפקודה PIP install + שם הספריה
3. להוריד את הספריה ghostscript מהאתר <https://www.ghostscript.com/download>
4. להוסיף את תיקיית הbin שירדה לpath מדרך:
<https://www.youtube.com/watch?v=gb9e3m98avk>
5. להוסיף לתיקייה את קבצי BOT.py, client.py, server.py, ואת תיקיית 'model'.
6. להריץ את קובץ Server.py (פתיחת סביבת העבודה ולחיצה על כפתור ההרצה)
7. להריץ את קובץ Client.py (פתיחת סביבת העבודה ולחיצה על כפתור ההרצה)

בתמונה ניתן לראות שהחץ הכחול מצביע על כפתור ההרצה:



ספריות בפרויקט:

ספריות הלקוח - Client.py:

ספריית socket - עם ספרייה זו ייבאנו את העצם socket דרכו מתרחשת התקשורת בין הלקוחות והשרת ובין השרת והBOT.

ספריית tkinter - באמצעות ספרייה זו יצרתי את הGUI בפרויקט שכן דרכה יצרתי את החלון (Tkinter) את הקנבס, את הכפתורים, ואת התוויות (labels)

ספריית select: באמצעות ספרייה זאת ניתן לקיים תקשורת מרובת משתתפים (יותר משרת ולקוח אחד) שכן דרכה ניתן לקבל רשימות של כל הsockets אשר מהם ניתן לקרוא מידע וכל הsockets אשר להם ניתן לשלוח הודעה דבר אשר הוכיח את עצמו מאוד שימושי וחשוב בפרויקט שלי.

ספריית threading - באמצעות ספרייה זו התאפשר ליצור threads שונים ולהריץ פעולות שונות במקביל, דבר שבלעדיו לא ניתן לקיים את פעולות הציור והמחיקה, שליחת ההודעות בין השרת והלקוח והשרת והבוט, ותפעול של שני sockets במקביל אצל כל אחד מן החלקים במערכת (client, server and BOT)

ספריות השרת - Server.py:

ספריית time - משומשת על מנת להפעיל את Timer של המשחק
ספריית random - משומשת על מנת לבחור מספר רנדומלי איתו נבחרת המילה הרנדומלית לתור.

ספריות הבוט - BOT.py:

ספריית numpy - ספרייה זו תומכת במערכים ומבני נתונים בעלי כמה מימדים, וכן ביכולתה לבצע פעולות מתמטיות מורכבות על מבני מידע אלו, השימוש בספרייה זו נעשה גם ביצירת המודל שכן קבצי האימון והבדיקה של המודל מכל קטגוריה הינם מסוג numpy

ספריית matplotlib.pyplot - ספרייה זו מאפשרת להציג בצורה ויזואלית ולייצר גרפים של מידע בשפת python ולהרחבה המתמטית שלה numpy.

ספריית keras - ספרייה זו הינה ספרייה ממשק בינה מלאכותית עבור python, ומשמשת כממשק עבור הספרייה tensorflow.

ספריית tensorflow - ספרייה זו הינה ספרייה של גוגל שמטרתה לבנות ולעצב רשתות עצביות (neural networks).

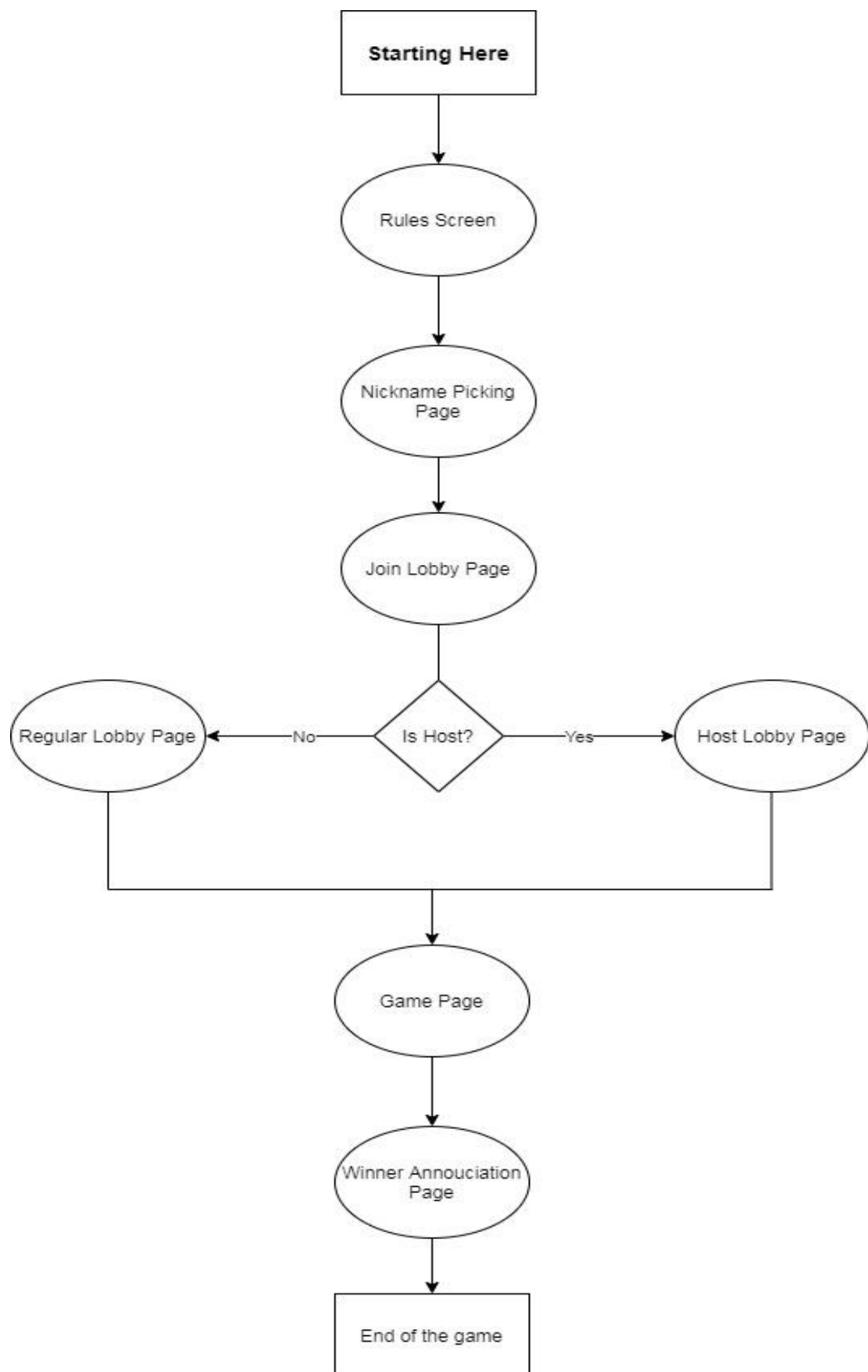
ספריית PIL - ספרייה אשר השימוש בה הוא לבצע פעולות ועריכה על קבצי תמונה ושמירת קבצי התמונה בפורמטים שונים.

ספריית OS - ספרייה זו מספקת פעולות אשר ניתן לפרנות למערכת ההפעלה ישירות והשתמש בה.

ספריית io - ספריית IO מאפשרת לנו לטפל בקלט הנוגע לקבצי המערכת.

ספריית glob - בספרייה זו ניתן להשתמש על מנת לגשת לקבצים או שמותיהם במערכת על פי דפוס קבוע.

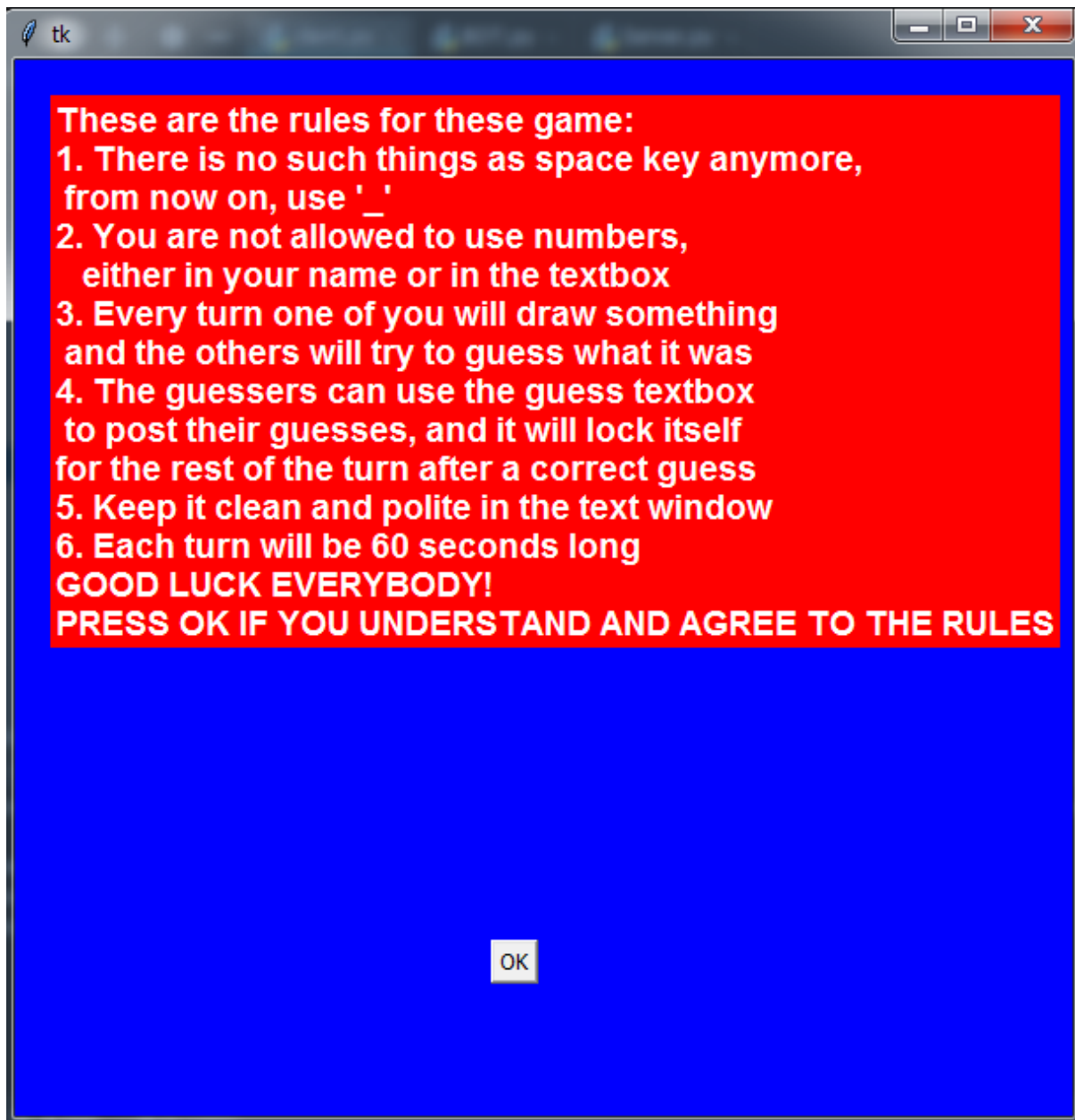
היררכיית מסכים:



תרשים מסכים:

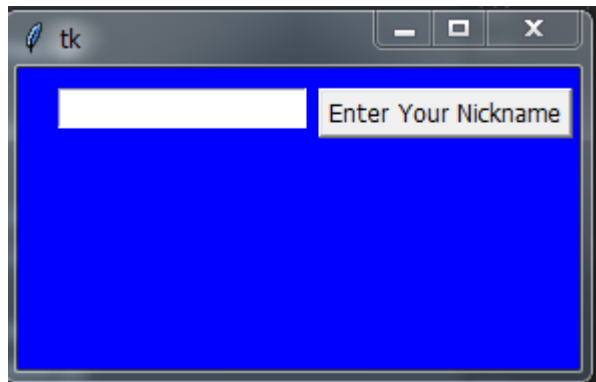
מסך חוקים:

המסך הראשון שהמשתמש רואה כאשר הוא פותח את האפליקציה בפעם הראשונה. מטרת המסך היא להסביר לשחקן את חוקי המשחק וממנו עוברים אל מסך הכנסת הכינוי.



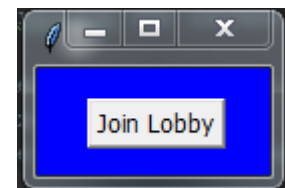
מסך בחירת כינוי:

מסך בו השחקן מזין את כינויו למערכת.
קלט: כינוי השחקן מתקבל כקלט ונשמר אצל השרת
ממסך זה עוברים למסך הכניסה ללובי.



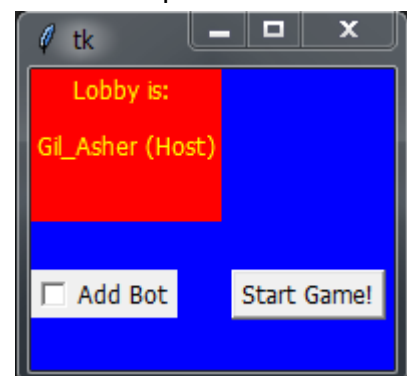
מסך הכניסה ללובי - Join Lobby Page:

במסך זה השחקן נכנס לתוך רשימת הלובי של המשחק
קלט: לחיצה על כפתור הכניסה ללובי ("Join Lobby")
ממסך זה עוברים למסך לובי ההמתנה הרגיל או למסך לובי המנהל (במידה והמשתמש שלחץ הינו הראשון
שלחץ על הכפתור "Join Lobby")



לובי המנהל - Host Lobby Page:

במסך זה ממתין מנהל המשחק (הראשון להכנס ללובי) לעוד שחקנים שיצטרפו תוך שלפניו נמצא רשימת
השחקנים שהצטרפו עד כה ופתוחות עבורו האפשרויות לקבוע האם יהיה BOT במשחק או לא ומתי להתחיל
את המשחק.
ממסך זה עוברים למסך המשחק.
קלט: לחיצה על כפתור "Start the game" להתחלת המשחק, ולחיצה על תיבת הסימון ("Add Bot")
להוספת BOT למשחק.



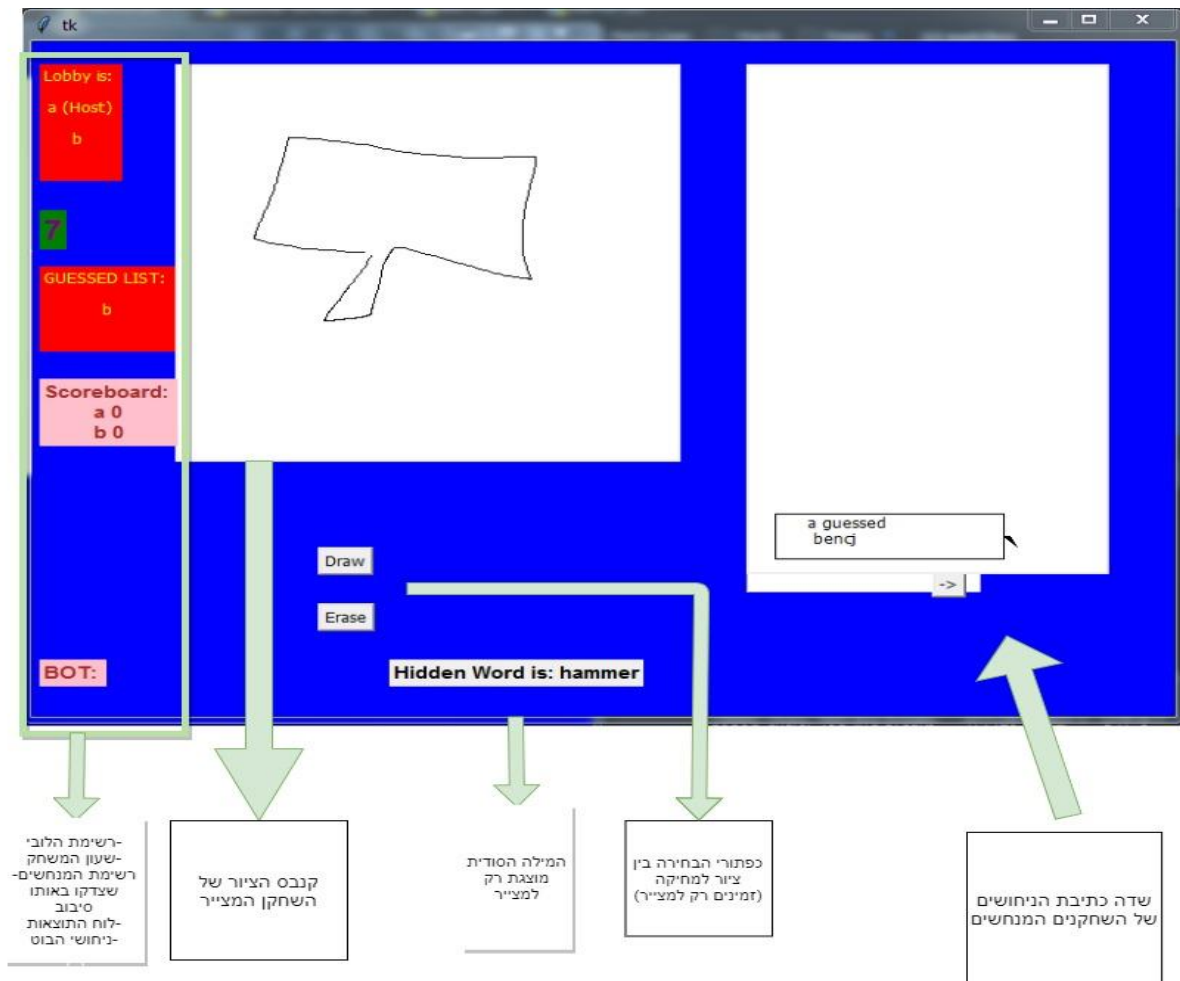
לובי המתנה רגיל - Regular Lobby Page:

במסך זה ממתינים כל השחקנים שאינם מנהל המשחק להתחלת המשחק על ידי המנהל תוך שלפניהם מוצגת רשימת השחקנים שהצטרפו ללובי עד כה. ממסך זה עוברים למסך המשחק באופן אוטומטי כאשר המנהל קובע.



מסך המשחק - Game Page

מסך בו מתרחש המשחק לכולם.
קלט: קואורדינטות הלחיצה על הקנבס של השחקן המצייר, בחירת השחקן המצייר האם לצייר או למחוק מן הקנבס, וניחושי השחקנים. ממסך זה עוברים למסך ההכרזה על המנצח.



מסך הכרזת המנצח - Winner Announcement Page :

מסך בו מוצג לכל השחקנים מנצח המשחק.



מדריך למפתח:

הסבר כללי על קוד הפרויקט:

הקוד שכתבתי בפרויקט מחולק לארבעה חלקים, הוא הקוד של הלקוח: GUI, והתקשורת עם השרת, הקוד של השרת: מנהל את הנתונים ואת המשחק, אחראי על תקשורת עם השחקנים הרצת הBOT והתקשורת איתו, הקוד של הBOT: מכיל את מודל זיהוי התמונה, מתקשר עם הסרבר ושולח ניחושים באשר למה מצויר בקנבס, הקוד שיוצר את מודל זיהוי התמונה: קוד שלומד על פי מאגר תמונות ליצור ניחושים ולזהות תמונה, ובסיומו שומר את המודל. (אין צורך להריץ את הקוד הזה יותר מפעם אחת).
את הפרויקט כתבתי בשפת python והשתמשתי בסביבת העבודה pycharm. והשתמשתי Packages כאלו ואחרים על מנת לקצר את תהליך כתיבת הקוד ולהפוך אותו לכמה שיותר יעיל.

הודעות במערכת:

הודעות מהלקוח לשרת בsockets הראשי:

<code>"client_Name + "n5103</code>	הודעת Name הלקוח שולח לשרת את שמו, השרת מוסיף את הלקוח וsocket שלו לdictionary
<code>client_Name + " guessed " + client_Guess</code>	הודעת Guess הלקוח שולח לשרת את הניחוש שלו, השרת בודק אם הניחוש נכון ומגיב בהתאם
<code>"client_Name + " enter5103</code>	הודעת Join_Request הלקוח מבקש מן השרת להצטרף ללובי השרת מאשר את הבקשה
<code>"client_Name + " STARTEDTHEGAME</code>	הודעת start the game נשלחת מהשחקן המנהל (HOST) לשרת כאשר הוא מעוניין להתחיל את המשחק עבור כולם, השרת מתחיל את המשחק

הודעות מהלקוח לשרת בsocket המשני (secondary socket):

<pre>LocToDraw: lastX- " + str(lastX) + "" lastY- " + str(lastY) + " X- " + str(X) (+ " Y- " + str(Y)</pre>	<p>הודעת מקום לציור (loctodraw) הודעה בה השחקן המצייר שולח לשרת את המקום בו הוא צייר במשחק ללא הBOT</p>
<pre>LocToErase: lastX- " + str(lastX) + "" lastY- " + str(lastY) + " X- " + str(X) (+ " Y- " + str(Y)</pre>	<p>הודעת מקום למחיקה (loctoerase) הודעה בה השחקן המצייר שולח לשרת את המקום בו הוא מוחק במשחק ללא הBOT</p>
<pre>RECTDRAWING- X: " + str(int(rectX_index)) + " Y: " + ((str(int(rectY_index</pre>	<p>הודעת מקום לציור מלבן (rectdrawing) שניהם rectX_index and rectY_index מייצגים את השורה והטור בהם צריכים להצבע המלבן בתוך 28 על 28 המלבנים על הקנבס הודעה הנשלחת מן השחקן המצייר אל השרת במשחק עם BOT</p>
<pre>RECTERASING- X: " + str(int(rectX_index)) + " Y: " + ((str(int(rectY_index</pre>	<p>הודעת מקום לציור מלבן (recterasing) שניהם rectX_index and rectY_index מייצגים את השורה והטור בהם צריכים להצבע המלבן בתוך 28 על 28 המלבנים על הקנבס הודעה הנשלחת מן השחקן המצייר אל השרת במשחק עם BOT</p>

הודעות מהשרת ללקוח בsocket הראשי:

<pre>guess.split(" ",1)[0]+" " + " ([:].join(guess.split(" ",1)[1</pre> <p>=</p> <p>CLIENT NAME + "GUESSED" + CLIENT GUESS</p>	<p>הודעת guess: הודעה המפיצה ניחוש של שחקן לשאר הלקוחות כאשר הוא לא נכון.</p>
<pre>(Lobby is: \r\n"+ "".join(lobby_list"</pre>	<p>הודעת Lobby is: הודעה זו נשלחת מהשרת לכל הלקוחות ומציגה מחרוזת המכילה את כל הנמצאים בLobby</p>
<pre>"Lobby is: \r\n"+ "".join(lobby_list)</pre>	<p>הודעת Host: הודעה הנשלחת מהשרת ללקוח הראשון שנכנס ללובי ומודיעה לו כי הוא המנהל</p>
<pre>"THE GAME IS ABOUT TO BEGIN WITH BOT" IN THE GAME IS ABOUT TO BEGIN WITHOUT" "BOT</pre>	<p>הודעות התחלת המשחק: נשלחות מן השרת לכל השחקנים ומעדכנות האם יש או אין BOT במשחק</p>

<code>message + "DS123 " #donescoreboard123</code>	הודעת scoreboard: שולחת בסיום כל תור מן השרת לכל הלקוחות במשחק את לוח התוצאות
<code>"message = "Your turn</code>	הודעת "התור שלך" הודעה הנשלחת בתחילת תור מן השרת אל השחקן שתורו לצייר
<code>(Timer: " + str(i"</code> מייצג את השניות i	הודעת Timer: הודעה הנשלחת מן השרת אל הלקוח כל שנייה ומעדכנת אותו לשנות את שעון המשחק הנוכחי.
<code>"message2 = "Turn Over</code>	הודעת Turn Over: הודעה הנשלחת מן השרת לכל השחקנים בסוף שעון המשחק ומודיעה להם על סוף התור.
<code>"message3 = "Clear Screen</code>	הודעת Clear Screen: הודעה הנשלחת מן השרת אל כל אחד מן הלקוחות ומודיעה להם לנקות את הקנבס לקראת התור הבא.
<code>!!!guesser_name + " Scored</code>	הודעת Correct guess: הודעה הנשלחת מן השרת אל כל אחד מן השחקנים במשחק כאשר אחד מן השחקנים מצליח לנחש את המילה החבויה, ההודעה מכילה בין היתר את שם השחקן
<code>"message = "BOT GOT IT RIGHT</code> או <code>message = "BOT guessed: " + bot_guess</code>	הודעות ניחוש הBOT: הודעה הנשלחת לכל אחד מן השחקנים במשחק עם הBOT כאשר הוא מנחש. ההודעה נשלחת בהתאם אם הBOT ניחש נכון או טעה

הודעות מהשרת ללקוח בsockets המשני:

<code>LocToDraw: lastX- " + str(lastX) + "</code> <code>lastY- " + str(lastY) + " X- " + str(X)</code> <code>(+ " Y- " + str(Y</code>	הודעת מקום לציור (LocToDraw) הודעה בה השרת מפיץ את ההודעת ציור שצויינה בעבר לכל השחקנים במשחק מלבד השחקן המצייר
<code>LocToErase: lastX- " + str(lastX) + "</code> <code>lastY- " + str(lastY) + " X- " + str(X)</code> <code>(+ " Y- " + str(Y</code>	הודעת מקום לציור (LocToErase) הודעה בה השרת מפיץ את ההודעת מחיקה שצויינה בעבר לכל השחקנים במשחק מלבד השחקן המצייר
<code>RECTDRAWING- X: " + "</code> <code>str(int(rectX_index)) + " Y: " + "</code> <code>((str(int(rectY_index</code>	הודעת מקום לציור (RectToDraw) הודעה בה השרת מפיץ את ההודעת ציור מלבן שצויינה בעבר לכל השחקנים במשחק מלבד השחקן המצייר במשחק עם הBOT

<pre>RECTERASING- X: " + str(int(rectX_index)) + " Y: " + (str(int(rectY_index</pre>	ההודעת מקום לציור (RectToErase) הודעה בה השרת מפיץ את ההודעת ציור שצויינה בעבר לכל השחקנים במשחק מלבד השחקן המצייר במשחק עם הBOT
--	---

הודעות מהשרת אל הBOT:

<pre>()Activate ML Model".encode"</pre>	הודעת בקשה לניחוש: השרת מבקש מן הBOT להשתמש במודל הML כדי לבצע ניחוש למה מצויר בקנבס
---	--

הודעות מהBOT אל השרת:

<pre>() IM THE BOT".encode"</pre>	הודעת "אני הוא הBOT" הודעה הנשלחת מן הBOT אל השרת במטרה לעדכן את השרת שהBOT התחבר ולאפשר לשרת לשמור את הsocket של הBOT
<pre>" " + BOTG: " + Bot_Answer"</pre>	הודעת ניחוש של הBOT: הודעה הנשלחת מן הBOT אל השרת ושולחת אל השרת את הניחוש של הBOT באמצעות ML

Threading:

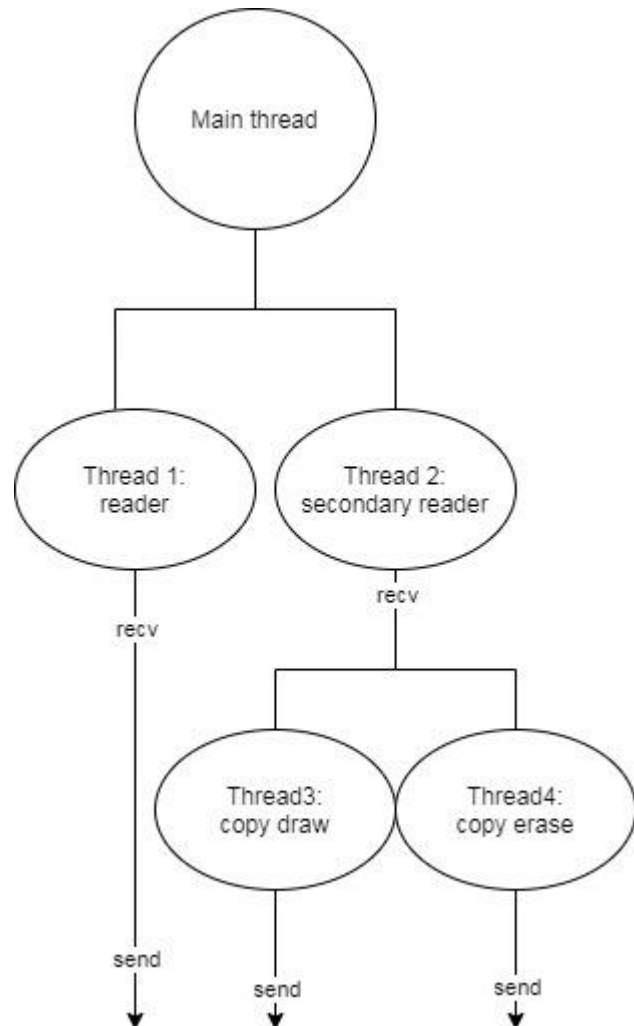
Threading מאפשר הפעלה של פעולה במקביל לקוד אשר ממשיך לרוץ ברקע.

מכיוון שעליי לשלוח כמות גדולה מאוד של הודעות (כמות גדולה כל שנייה) עליי היה להשתמש בthreading על מנת לפצל את מערכת קבלת ההודעות לשני sockets שרצים במקביל בכל אחד מן הקבצים (client.py, server.py, Bot.py), אחד אחראי על כמות ההודעות הגדולה המתקבלת בכל הנוגע לציור על הקנבס, והשני על ההודעות הנוגעות למשחק עצמו. בנוסף על מנת ליצור שעון משחק שירץ במקביל למשחק עליי היה ליצור thread נוסף גם בשבילו. כמובן שגם הציור והמחיקה על הקנבס מתרחשים בthreads שונים כיוון שאין אנו רוצים לתקוע את המשחק עד שתצבע הנקודה הבאה על המסך.

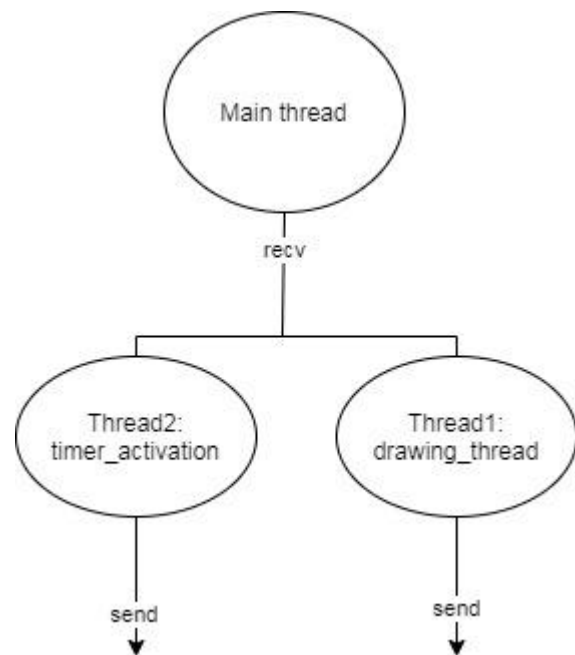
כלומר, המשחק לא נתקע אף פעם כאשר מבצעים פעולה, אלא היא רצה במקביל. כאשר מתקבלת תשובה על הצלחה או כישלון של הפעולה הא-סינכרונית ניתן לשנות את מצב המשחק בהתאם.

תרשימי הThreads במערכת: הערה: recv and send - פעולות אלה מייצגות את פעולות השליחה והקבלה של המידע דרך האובייקט socket.

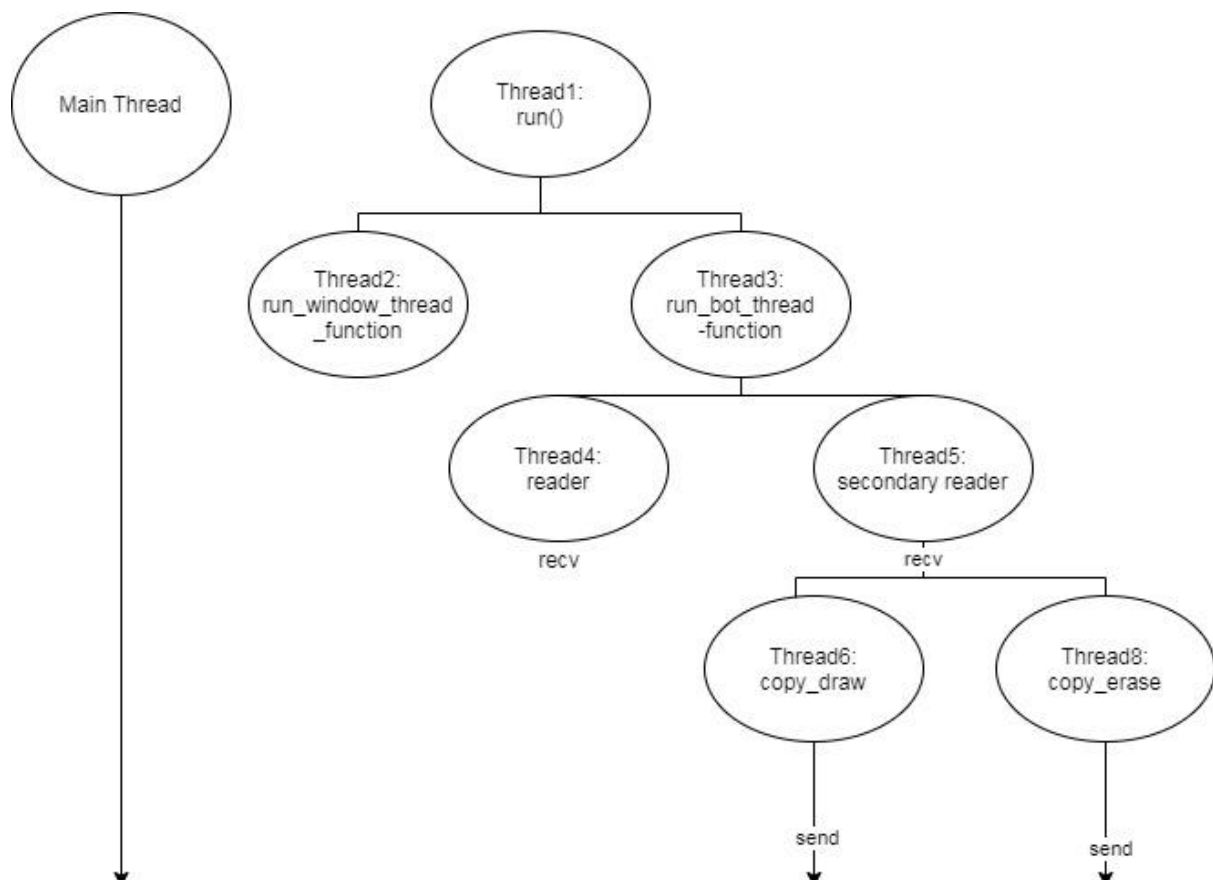
תרשים הThreads של Clinet:



תרשים הThreads של Server:



תרשים הThreads של BOT:



קישור לקוד הפרוייקט בGitHub:

<https://github.com/gilasher5103/Quick-Skribble->

הסברים על קבצים, פונקציות וחלקי קוד חשובים

קובץ הBOT:

שם הקובץ – BOT.py

תוכנו - בקובץ זה כתובות הפעולות אשר מתקשרות עם השרת, והפעולות האחראיות על הזיהוי של התמונה.

Run() Fuction:

פעולה זאת נקראת על ידי השרת מיד ביצירת האובייקט של sockets והיא אחראית על חיבור הsockets של הBOT לsockets של השרת והתחלת הthreads הבאים:

1. run_bot_thread_function
2. run_window_thread_function

```
bot_socket = socket.socket()
secondary_bot_socket = socket.socket()
my_canvas = Canvas

def run():
    global bot_socket
    global secondary_bot_socket

    print("BOT start")

    bot_socket.connect(("127.0.0.1", 8820))
    print("BOT connect to server")

    print("secondary BOT socket start")

    secondary_bot_socket.connect(("127.0.0.1", 2088))
    print("secondary BOT socket connected to server")
    run_window_thread = threading.Thread(target=run_window_thread_function)
    run_window_thread.start()
    run_bot_thread = threading.Thread(target=run_bot_thread_function)
    run_bot_thread.start()
```

run_bot_thread_function():

פעולה זאת נקראת מתוך הפעולה run() ותפקידה הינו להתחיל את הפעולה start_server_interaction() ואת התהליכים reader ו secondary_reader

run_window_thread_function():

פעולה זאת נקראת מתוך הפעולה run() ותפקידה הינו ליצור את הקנבס של החלון של הBOT ולהוסיף אליו את הקנבס.

```
def run_bot_thread_function():

    start_server_interaction()
    print("start_server_interaction() ")

    t = threading.Thread(target=reader)
    t.start()
    print("reader was called")
    t2 = threading.Thread(target=secondary_reader)
    t2.start()

def run_window_thread_function():
    global my_canvas
    print("run_thread_function")
    window = Tk()
    window.geometry("600x600")
    window.configure(bg='blue')
    my_canvas = Canvas(window, width=350, height=350, background='white')
    my_canvas.place(x=20, y=20)
    window.mainloop()
```

start_server_interaction():

הפעולה שולחת הודעה לsocket הראשי של השרת על מנת לאפשר לשרת לשמור את הsocket של הBOT בתוך משתנה והלקל את התקשורת העתידית.

```
def start_server_interaction():
    print("start_server_interaction")
    bot_socket.send("IM THE BOT".encode())
```

copy_draw() and copy_erase():

הפעולות נקראות מתוך פעולת secondary_reader() ומקבלות string המציין את הנקודות בהן צריך לצבוע מלבן על פי הפרוטוקול הבא: הקנבס אשר מורכב ל364X364 פיקסלים מחולק ל28X28 מלבנים באורך 13X13 פיקסלים, string המתקבל מכיל את המצביע על המלבן אותו צריך לצבוע, לדוגמה עבור המלבן בשורה השלישית בטור השני יתקבל string המציין שצריך לצבוע מלבן בX=1 וY=2 (הספירה מתחילה ב0)

ההמרה לנקודות בהן צריכים לצבוע את המלבן כתובה בתוך הפעולות. הפעולה `copy_draw` צובעת בצבע שחור ועוד שפעולת `copy_erase` צובעת מלבן בצבע לבן.

```
def copy_draw(datar): # draw what the playing player draws
    global my_canvas
    lista = datar.split("RECTDRAWING- ")
    lista.remove("")
    for coordinates in lista:
        l = coordinates.split(" ")
        x = int(l[1])
        y = int(l[3])
        my_canvas.create_rectangle(x*13,y*13,x*13+13,y*13+13,fill='black')

def copy_erase(datar): # erase what the playin player erase
    global my_canvas
    lista = datar.split("RECTERASING- ")
    lista.remove("")
    for coordinates in lista:
        l = coordinates.split(" ")
        x = int(l[1])
        y = int(l[3])
        my_canvas.create_rectangle(x * 13, y * 13, x * 13 + 13, y * 13 + 13, fill='white',outline="white")
```

```
def clear_screen(): # clears the canvas at the end of the turn
    global my_canvas
    my_canvas.delete("all")
```

reader():

הפעולה אחראית על התקשורת בין socket הראשי של הBOT לsocket הראשי של השרת. התקשורת ביניהם נוגעת לנושאים כגון ניקוי הקנבס בסופו של תור ושימוש במודל machine learning.

```
def reader():
    global private_lobby_list
    global guess_list
    global my_turn
    global guessed_list
    global bot_socket
    global secondary_bot_socket
    while True:
        rlist, wlist, xlist = select.select([bot_socket], [bot_socket], [])
        for current_socket in rlist:
            datar = current_socket.recv(1024).decode()
            print("server replied with: " + datar)
            # print("datar[0:4] is: " + datar[0:4])

            if "Clear Screen" in datar:
                clear_screen()
            if "Activate ML Model" in datar:
                make_a_guess()
```

secondary_reader():

הפעולה אחראית על התקשורת בין socket המשני של הBOT עם socket המשני של השרת, התקשורת שעוברת ביניהם הינה רק הודעות הנוגעות למיקום בו על הBOT לצייר מלבן בקונבס.

```
def secondary_reader():
    global bot_socket
    global secondary_bot_socket
    while True:
        rlist, wlist, xlist = select.select([secondary_bot_socket], [secondary_bot_socket], [])
        for current_socket in rlist:
            datar = current_socket.recv(1024).decode()
            #print("SECOND SERVER: " + datar)
            if "RECTDRAWING-" in datar:
                #print("calling copy draw")
                t3 = threading.Thread(target=copy_draw, args=(datar,))
                t3.start()
            if "RECTERASING-" in datar:
                print("calling copy erase")
                t4 = threading.Thread(target=copy_erase, args=(datar,))
                t4.start()
```

make_a_guess():

פעולה של הBOT אשר אחראית על שימוש במודל machine learning על מנת לזהות את התמונה אשר שמורה בקובץ "Image_To_Guess" ולשלוח לשרת את ניחושו של הBOT.

```
def make_a_guess():
    save_current_canvas()

    try:
        Bot_Answer = ""

        os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2' # !!!!!!!!!!!!!!!

        file_list = glob.glob("Image_To_GUESS.jpg")

        file = open("class_names.txt") # !!!!!!!!!!!!!!!

        line = file.read().replace("\n", " ")
        file.close()
        CATEGORIES = list(line.split(" "))

        x = []
        for each in file_list:
            img = image.load_img(each, color_mode="grayscale", target_size=(28, 28))
            img = image.img_to_array(img)
            img = img.reshape(28, 28, 1)
            img = img.astype('float32')
            img = (255 - img) / 255.0
            x.append(img)
            plt.figure()
            plt.imshow(img, cmap='Greys')
            plt.grid(False)
            plt.colorbar()
            # plt.show()
```

```

        # plt.show()
        x.append(img)

x = np.array(x)
model = tf.keras.models.load_model("keras.h5", compile=True) # !!!!!
result = model.predict_classes(x)
for i in range(len(result)):
    print(CATEGORIES[result[i]], result[i])
    Bot_Answer = str(CATEGORIES[result[i]])
print("RESULT IS: ")
print(Bot_Answer)

for i in range(len(result)):
    plt.figure()
    plt.imshow(x[i], cmap='Greys')
    plt.grid(False)
    plt.axis('off')
    plt.colorbar()
    plt.title(CATEGORIES[result[i]])
    # plt.show()
message = "BOTG: " + Bot_Answer + " "
bot_socket.send(message.encode())
except:
    print("BOT ERROR")

```

save_current_canvas():

הפעולה נקראת מתוך הפעולה make_a_guess() והיא אחראית על שמירת התוכן אשר נמצא באותו רגע על הקנבס והמרתו ל JPG בשם "Image_to_guess".

```

def save_current_canvas():
    my_canvas.update()
    ps = my_canvas.postscript(colormode='color')
    image = Image.open(io.BytesIO(ps.encode('utf-8')))
    image.save("Image_To_GUESS.jpg")

```

קובץ הServer:

שם הקובץ – server.py
 תוכנו - בקובץ זה כתובות הפעולות אשר מתקשרות עם הלקוח והBOT, והמשתנים אשר צריכים לשמור לאורך המשחק.

משתנים חשובים + הערות:

```
server_socket = socket.socket() #starts the server main socket
server_socket.bind(('0.0.0.0', 8820))
server_socket.listen()

print("server start")

secondary_server_socket = socket.socket() #starts the server secondary socket
secondary_server_socket.bind(('0.0.0.0', 2088))
secondary_server_socket.listen()

found_bot = False #bool that represents if the bot was found yet
open_secondary_client_sockets = [] #available client secondary sockets
open_client_sockets = [] #available client sockets
messages_to_send = [] #list of messages that need to be sent
sockets_dictionary = {} #dictionary that contains a socket and the name of the socket user
guess_list = [] #list that represents the player who guessed correctly the hidden word
lobby_list = [] #list that represents the players that are in the lobby
host_client = () #tuple that contains the name and socket of the host
current_client_name = "" #variable that contains the name of the current client in the reader func
queue_list = [] #list that represents the queue of the game
playing_socket = server_socket #variable that contains the socket of the current drawer
scoreboard = [] #list of tuples that contains the name of the player and their score
bot_socket = server_socket #variable that contains the socket of the bot
current_time = 30 #initial time for the timer

file = open("class_names.txt") #categories in the game
```

```

def send_waiting_messages(wlist): #sends the messages that are left to be sent

    counter = 0
    for guess in guess_list:
        for socket_to_send in wlist:
            message = guess.split(" ")[0]+" " + " ".join(guess.split(" ")[1:])
            # print("sending to: " + sockets_dictionary{})
            socket_to_send.send(message.encode())
            counter += 1
            if(counter == len(wlist)):#checks if every socket recieved the guess
                guess_list.remove(guess)

def update_lobby_list(wlist): #updates the lobby list when a player enters the lobby
    for socket_to_send in wlist:
        current_lobby_list = "Lobby is: \r\n"+ " ".join(lobby_list)
        socket_to_send.send(current_lobby_list.encode())

def send_host_message(wlist): #alerts the host that he is the host of the game
    if host_client[1] in wlist:
        message = "You are now the host"
        host_client[1].send(message.encode())

def send_start_message(wlist,add_bot_bool): #notify the cliets that the game is beginning
    for socket_to_send in wlist:
        for key, value in sockets_dictionary.items():
            if socket_to_send == value:
                if key in " ".join(lobby_list):
                    print("About to send something")
                    if add_bot_bool:
                        message = "THE GAME IS ABOUT TO BEGIN WITH BOT"
                    else:
                        message = "THE GAME IS ABOUT TO BEGIN WITHOUT BOT"
                    socket_to_send.send(message.encode())
        create_scoreboard()
        # print("something was sent")

def create_scoreboard(): #creates a list that contains tuples of player's name and score based on the lobby list
    global lobby_list
    global scoreboard
    print(lobby_list)
    for i in range(0, len(lobby_list)):
        scoreboard.append((lobby_list[i], 0))
    print(scoreboard)

```

```

def update_scoreboard(guesser_name): #update the score of the players based on their performance in the last round
    global scoreboard
    global current_time

    for i in range(0, len(scoreboard)):
        if scoreboard[i][0].split(" ")[0] == guesser_name:
            lst = list(scoreboard[i])
            lst[1] = current_time * 5 + lst[1]
            scoreboard[i] = tuple(lst)
            print("SCOREBOARD HAS BEEN UPDATED")
            print(scoreboard)

def send_scoreboard(): #sends the scoreboard to all the clients at the end of a round
    global scoreboard
    message = "SCOREBOARD: "
    for i in range(0, len(scoreboard)):
        message = message + scoreboard[i][0].split(" ")[0] + str(scoreboard[i][1])
    message = message + "DS123 " #donescoreboard123
    print("SCOREBOARD MESSAGE IS: " + message)
    for socket_to_send in wlist:
        socket_to_send.send(message.encode())

def random_hidden_word(): #picks the hidden word of the turn
    global hidden_word
    global CATEGORIES

    word_index = random.randint(0, 99)
    hidden_word = CATEGORIES[word_index]
    print("THE HIDDEN WORD IS: " + hidden_word)

```

```

def handle_queue(wlist): #alerts the next player that it is now his turn
    global sockets_dictionary
    global queue_list
    global playing_socket
    global round_number
    print("queue list is: ")
    print(queue_list)
    message = "Your turn" + str(random_hidden_word()) + " END OF 'YOUR TURN' Message"
    if queue_list == []:
        if round_number < 3:
            round_number = round_number + 1
            make_queue(wlist)
        else:
            if "Host" in queue_list[0]:
                if sockets_dictionary[queue_list[0][0:len(queue_list[0]) - 9]] in wlist:
                    sockets_dictionary[queue_list[0][0:len(queue_list[0]) - 9]].send(message.encode())
                    playing_socket = sockets_dictionary[queue_list[0][0:len(queue_list[0]) - 9]]
                    queue_list.remove(queue_list[0])
            else:
                if sockets_dictionary[queue_list[0][0:len(queue_list[0]) - 3]] in wlist:
                    sockets_dictionary[queue_list[0][0:len(queue_list[0]) - 3]].send(message.encode())
                    playing_socket = sockets_dictionary[queue_list[0][0:len(queue_list[0]) - 3]]
                    queue_list.remove(queue_list[0])

    t5 = threading.Thread(target=timer_activation, args=(wlist, playing_socket,))
    t5.start()

```

```

def ask_for_bot_guess(): #send a request to the bot for a guess that describes what is painted on the canvas
    print("BOT IS BEING CALLED")
    bot_socket.send("Activate ML Model".encode())

def timer_activation(wlist,playing_socket): # a thread that is resposnsible for the timer in the game
    global current_time
    global found_bot
    print("TIME ACTIVATION WAS CALLED")
    for i in range(30,-1,-1):
        for socket_to_send in wlist:
            if i % 5 == 0 and found_bot:
                ask_for_bot_guess()
            message = "Timer: " + str(i)
            current_time = i
            socket_to_send.send(message.encode())
        time.sleep(1)
    message2 = "Turn Over"
    playing_socket.send(message2.encode())
    for some_socket in wlist:
        message3 = "Clear Screen"
        some_socket.send(message3.encode())
        if found_bot:
            bot_socket.send(message3.encode())

    send_scoreboard()
    handle_queue(wlist)

```

```

def send_loc_broadcast(current_socket,data,wlist):#sends location to draw to everyone but the current socket which draws
    for socket_to_send in wlist:
        if socket_to_send != current_socket:
            socket_to_send.send(data.encode())

def correct_answer(wlist,guesser_name):#notify the clients when someone mennages to guess the hidden word
    for socket_to_send in wlist:
        message = guesser_name + " Scored!!"
        socket_to_send.send(message.encode())
    update_scoreborad(guesser_name)

```

```

def drawing_thread(): #responsible for broadcasting the location to draw and erase for all the player but the drawer
    while True:
        rlist, wlist, xlist = select.select([secondary_server_socket] + open_secondary_client_sockets, open_secondary_cl
        for current_socket in rlist:
            if current_socket is secondary_server_socket:
                (new_socket, address) = secondary_server_socket.accept()
                open_secondary_client_sockets.append(new_socket)
                print("new secondary client connected")
            else:
                data = current_socket.recv(1024)
                data = data.decode()
                # print("data is " + data)
                if data == "":
                    open_secondary_client_sockets.remove(current_socket)
                    print("connection with client lost")
                elif data[0:10] == "LocToDraw:" or "RECTDRAWING" in data:
                    #print("Entered the loc to draw if")
                    send_loc_broadcast(current_socket, data, wlist)
                elif data[0:11] == "LocToErase:" or "RECTERASING":
                    #print("Entered the loc to erase if")
                    send_loc_broadcast(current_socket, data, wlist)

def add_bot_to_scoreboard(): #adds the bot to the scoreboard list
    global scoreboard
    global current_time
    scoreboard.append(("BOT", 0))

def update_bot_scoreboard(): #updates the bot score when it guesses the hidden word
    print("ADDING BOT SCORE TO SCOREBOARD")
    global scoreboard
    global current_time
    for i in range(0, len(scoreboard)):
        if scoreboard[i][0].split(" ")[0] == "BOT":
            lst = list(scoreboard[i])
            lst[1] = current_time * 5 + lst[1]
            scoreboard[i] = tuple(lst)
            print("SCOREBOARD HAS BEEN UPDATED")
            print(scoreboard)

def send_bot_guesses(isCorrect, bot_guess): #sends the bot's guesses to every client in the game
    if isCorrect:
        message = "BOT GOT IT RIGHT"
        for socket_to_send in wlist:
            socket_to_send.send(message.encode())
            update_bot_scoreboard()
    else:
        message = "BOT guessed: " + bot_guess
        for socket_to_send in wlist:
            socket_to_send.send(message.encode())
    print(message)

```



```

counter = 0 #main reader: this loop is responsible for recieving and handling the data from the clients and bot
while True:
    rlist, wlist, xlist = select.select([server_socket] + open_client_sockets, open_client_sockets, [])
    if counter % 1000000 == 0:
        print("shalom oved", counter)
    counter += 1
    for current_socket in rlist:
        #print("new mashu hegia")
        if current_socket is server_socket:
            (new_socket, address) = server_socket.accept()
            open_client_sockets.append(new_socket)
            print("new client connected")
        else:
            data = current_socket.recv(1024)
            data = data.decode()
            print("data is " + data)
            if data[len(data)-5:] == "n5103":
                current_client_name = data[0:len(data) - 5]
                print(data[0:len(data) - 5])
                sockets_dictionary[current_client_name] = current_socket
            elif data == "":
                open_client_sockets.remove(current_socket)
                print("connection with client lost")
            elif data[len(data)-9:] == "enter5103":
                print(data[0:len(data) - 9] + " is being added to lobby")
                lobby_list.append(data[0:len(data) - 9]+"\r\n")
                if host_client == ():
                    print(data[0:len(data) - 9] + " is now the host")

```

```

if host_client == ():
    print(data[0:len(data) - 9] + " is now the host")
    host_client = (data[0:len(data) - 9], current_socket)
    send_host_message(wlist)
    lobby_list.remove(data[0:len(data) - 9] + "\r\n")
    lobby_list.append(data[0:len(data) - 9] + "(Host)\r\n")
    update_lobby_list(wlist)
elif "STARTEDTHEGAME" in data:
    if "STARTEDTHEGAMEWITHBOT" in data:
        send_start_message(wlist, True)
        BOT.run()
        add_bot_to_scoreboard()
        print("NICELY DONE")
    elif "STARTEDTHEGAMEWITHOUTBOT" in data:
        send_start_message(wlist, False)
    print("And so, it begins... (THE GAME)")
    make_queue(wlist)
elif "guessed" in data:
    print("data[data.find('guessed')+1:] " + data[data.find("guessed")+1:])
    if data[data.find("guessed")+8:] == hidden_word:
        print("CORRECT GUESS!!!!!!!!!!")
        correct_answer(wlist, data.split(" ")[0])
    else:
        guess_list.append(data)
elif "IM THE BOT" in data:
    print("BOT CONNECTED")
    bot_socket = current_socket

```

```

elif "IM THE BOT" in data:
    print("BOT CONNECTED")
    bot_socket = current_socket
    found_bot = True
elif "BOTG" in data:
    index = data.find("BOTG")
    index2 = data[index + 6:].find(" ")
    g = data[index + 6:index + 6 + index2]
    print("G IS: " + g)
    if g == hidden_word:
        print("BOT GUESSED CORRECTLY AND IM ABOUT TO SEND IT")
        send_bot_guesses(True, g)
    else:
        send_bot_guesses(False, g)

send_waiting_messages(wlist)

```

קובץ Client:

שם הקובץ – client.py

תוכנו - בקובץ זה כתובות הפעולות אשר מתקשרות עם השרת ויוצרות את canvas.

```

print("client start")
client_socket = socket.socket()
client_socket.connect(("127.0.0.1", 8820))
print("client connect to server")

print("secondary client socket start")
secondary_client_socket = socket.socket()
secondary_client_socket.connect(("127.0.0.1", 2088))
print("secondary client socket connected to server")

isHost = False #bool that is True only if the current player is the host
my_turn = False #bool that is True only if it is currently that player turn
client_Name = "" #string that saves the nickname of the player
client_Guess = "" #string of the current guess of the player
private_lobby_list = "" # string that describes the lobby
guessed_list = "" # string that describes the players who guessed the hidden word that turn
flag = 0
lastX = 0 # had to define the variables lastX and lastY in order to use them but their initial values are irrelevant
lastY = 0
drawing = 0

guess_list = [] #list of the guesses that were made in the game

add_bot_variable = 0 # variable is 0 if there is no bot in the game and 1 if there is

```

```

def recieve_name(): # recieve the player's name and send it
    global client_Name
    client_Name = textBox.get("1.0", 'end-1c') # recieves the current input from the textbox
    print("The name is " + client_Name)
    enter_Name_Button.destroy() # removes the name button
    textBox.delete(1.0, END) # clears the text box
    join_lobby_button.place(x=50, y=20) # adds the enter guess button
    textBox.config(state='disabled')
    textBox.destroy()
    name_message = client_Name + "\n5103" # informs the server that this is the first message from the client
    client_socket.send(name_message.encode())

def Guess(): # send guesses
    global client_Guess
    global client_Name
    client_Guess = textBoxNew.get("1.0", 'end-1c') # recieve input from the text box
    print(client_Name + " guessed " + client_Guess)
    print("client_Guess " + client_Guess)
    textBoxNew.delete(1.0, END) # clears the text box
    if len(client_Guess) < 15: # 15 is the length of the longest word
        message = client_Name + " guessed " + client_Guess
        client_socket.send(message.encode())
        print("sending: " + message)
    else:
        print("GUESS WAS TOO LONG TO SEND")

```

```

def joinLobby(): # responsible for joining the lobby
    join_lobby_button.destroy()
    join_request = client_Name + " enter5103"
    print("sending request to join by: " + join_request)
    client_socket.send(join_request.encode())
    lobby_list_label.grid(row=2, column=0)

def update_private_lobby_list(): # change the label of lobby list
    global private_lobby_list
    print("The Lobby list before adding is: " + private_lobby_list)
    lobby_list_label.config(text=private_lobby_list)

def add_start_button(): #adds the start button to the screen
    start_game_button.place(x=80, y=107)

```

```

def start(): #sends the server a message to start the game and if there is a bot in the game
    global add_bot_variable
    global add_bot_button
    message = ""
    if add_bot_variable == 1:
        message = client_Name + " STARTEDTHEGAMEWITHBOT"
    else:
        message = client_Name + " STARTEDTHEGAMEWITHOUTBOT"

    client_socket.send(message.encode())
    start_game_button.destroy()
    add_bot_button.destroy()

```

```
def start_the_actual_game(add_bot_bool): #adjust the screen to game screen
    global add_bot_variable
    window.geometry("800x600")
    lobby_list_label.place(x=5, y=20)
    my_canvas.place(x=100, y=20)
    # Guess_Button.grid(row=3, column=1)
    textBoxNew.place(x=500, y=470)
    textBoxNew.config(state='normal')
    Guess_Button.place(x=630, y=470)
    guess_Canvas.place(x=500, y=20)
    guessed_list_label.place(x=5, y=200)
    if add_bot_bool:
        add_bot_variable = 1
    else:
        add_bot_variable = 0
```

```
def fill_rect(rectX_index, rectY_index, color): #draws or erase on the canvas when there is a bot in the game
    global my_canvas
    if color == "black":
        my_canvas.create_rectangle(rectX_index * 13, rectY_index * 13, rectX_index * 13 + 13, rectY_index * 13 + 13,
                                   fill="black")
    elif color == "white":
        my_canvas.create_rectangle(rectX_index * 13, rectY_index * 13, rectX_index * 13 + 13, rectY_index * 13 + 13,
                                   fill="white", outline='white')
    # print("white rectangle is being painted")
```

```
def draw_and_erase_with_bot(event): #draws and erase on the canvas when there is a bot in the game
    global drawing
    # global my_turn
    # if my_turn:
    X = event.x
    Y = event.y
    global flag
    global lastX
    global lastY
    global color

    rectX_index = 0 # with those variables i will identify which rectangle im suppose to paint
    rectY_index = 0

    rectX_index = X / 13
    rectY_index = Y / 13

    if drawing == 0:
        fill_rect(int(rectX_index), int(rectY_index), "black")
        message1 = "RECTDRAWING- X: " + str(int(rectX_index)) + " Y: " + str(int(rectY_index))
        secondary_client_socket.send(message1.encode())
    else:
        fill_rect(int(rectX_index), int(rectY_index), "white")
        message2 = "RECTERASING- X: " + str(int(rectX_index)) + " Y: " + str(int(rectY_index))
        secondary_client_socket.send(message2.encode())
```

```
def drawOrErase(event):#calls the right function in corilation with the current state(with or without bot,draw or eras
    global drawing
    global add_bot_variable
    if drawing == 0:
        if add_bot_variable == 1:
            draw_and_erase_with_bot(event)
        else:
            draw(event)
    else:
        if add_bot_variable == 1:
            draw_and_erase_with_bot(event)
        else:
            erase(event)
```

```
def draw(event):_#draws in the game without the bot
    global my_turn
    if my_turn:
        X = event.x
        Y = event.y
        global flag
        global lastX
        global lastY
        global color

        if flag == 0:
            flag = 1
            lastX = X
            lastY = Y

        message = "LocToDraw: lastX- " + str(lastX) + " lastY- " + str(lastY) + " X- " + str(X) + " Y- " + str(Y)
        secondary_client_socket.send(message.encode())

        my_canvas.create_line(lastX, lastY, X, Y, fill="black")
        lastX = X
        lastY = Y
```

```
def erase(event):#erase from the canvas in the game without the bot
    global my_turn
    if my_turn:
        X = event.x
        Y = event.y
        global flag
        global lastX
        global lastY
        global color

        if flag == 0:
            flag = 1
            lastX = X
            lastY = Y

        message = "LocToErase: lastX- " + str(lastX) + " lastY- " + str(lastY) + " X- " + str(X) + " Y- " + str(Y)
        secondary_client_socket.send(message.encode())

        my_canvas.create_rectangle(lastX, lastY, lastX + 10, lastY + 10, fill="white", outline="")
        lastX = X
        lastY = Y
```

```

def copy_draw(datar): # draw what the playing player draws
    lista = datar.split("LocToDraw: ")
    lista.remove("")
    print("COPY DRAW WAS CALLED")
    print("LIST A = ")
    print(lista)
    for i in range(0, len(lista)):
        # client_socket.send("copied that".encode())
        lastX = int(lista[i].split(" ")[1])
        lastY = int(lista[i].split(" ")[3])
        X = int(lista[i].split(" ")[5])
        Y = int(lista[i].split(" ")[7])
        my_canvas.create_line(lastX, lastY, X, Y, fill="black")

def copy_erase(datar): # erase what the playin player erase
    lista = datar.split("LocToErase: ")
    lista.remove("")

    for i in range(0, len(lista)):
        # client_socket.send("copied that".encode())
        lastX = int(lista[i].split(" ")[1])
        lastY = int(lista[i].split(" ")[3])
        X = int(lista[i].split(" ")[5])
        Y = int(lista[i].split(" ")[7])
        my_canvas.create_rectangle(lastX, lastY, lastX + 10, lastY + 10, fill="white", outline="")

```

```

def start_turn(datar): # adds the button to draw
    global my_turn
    global drawButton
    global eraseButton
    global hidden_word_label

    drawButton = Button(window, text="Draw", command=drawInitiator)
    eraseButton = Button(window, text="Erase", command=eraseInitiator)
    print("start turn was called")
    clear_screen()
    drawButton.place(x=200, y=450)
    eraseButton.place(x=200, y=500)
    lst = datar.split(" ")
    i = 0
    for word in lst:
        if word == "Your":
            break
        i = i + 1

    print("Hidden word time")
    hidden_word_label.config(text="Hidden Word is: " + lst[i + 2])
    hidden_word_label.place(x=250, y=550)

```



```

def add_guess(): #adds a guess to the guess canvas
    global client_Name
    rectXa = 20
    rectYa = 400
    rectXb = 180
    rectYb = 440

    triangleYa = 430
    triangleYb = 420
    triangleYc = 420

    triangleXa = 190
    triangleXb = 185
    triangleXc = 180

    global guess_list

    if len(guess_list) < 10:
        for i in range(len(guess_list) - 1, -1, -1):
            guess_Canvas.create_rectangle(rectXa, rectYa, rectXb, rectYb, fill="white")
            points = [triangleXa, triangleYa, triangleXb, triangleYb, triangleXc, triangleYc]
            guess_Canvas.create_polygon(points)
            if len(guess_list[i]) > 12 and len(guess_list[i].split('\n')) < 2:
                guess_list[i] = guess_list[i][0:len(client_Name) + len(" guessed")] + "\n" + guess_list[i][
                    len(client_Name) + len(
                        " guessed"):]

            guess_Canvas.create_text(rectXa + 50, rectYa + 15, text=guess_list[i]) # todo
            rectYa = rectYa - 60
            rectYb = rectYb - 60

```

```

        triangleYa = triangleYa - 60
        triangleYb = triangleYb - 60
        triangleYc = triangleYc - 60

    else:
        for i in range(len(guess_list) - 1, len(guess_list) - 10, -1):
            guess_Canvas.create_rectangle(rectXa, rectYa, rectXb, rectYb, fill="white")
            points = [triangleXa, triangleYa, triangleXb, triangleYb, triangleXc, triangleYc]
            guess_Canvas.create_polygon(points)
            if len(guess_list[i]) > 12 and len(guess_list[i].split('\n')) < 2:
                guess_list[i] = guess_list[i][0:len(client_Name) + len(" guessed")] + "\n" + guess_list[i][len(client_Name) + len(" guessed"):]
            guess_Canvas.create_text(rectXa + 50, rectYa + 15, text=guess_list[i]) # todo
            rectYa = rectYa - 60
            rectYb = rectYb - 60

            triangleYa = triangleYa - 60
            triangleYb = triangleYb - 60
            triangleYc = triangleYc - 60

```

```
def clear_screen(): # removes the paintings
    global guessed_list
    guessed_list = ""
    print("Screen is clean")
    guessed_list_label.config(text="")
    my_canvas.delete("all")

def finish_turn(): #responsible for prepering the next turn
    global my_turn
    global drawButton
    global eraseButton
    # clear_screen()
    print("destroying the buttons")
    drawButton.destroy()
    eraseButton.destroy()
    hidden_word_label.config(text="")
    my_turn = False
```

```

def update_scoreboard(datar):#updates the scoreboard
    global scoreboard_label
    s = datar
    l = s.split(" ")
    i = 0
    for word in l:
        if "SCOREBOARD" in l[i]:
            i = i + 1
            break
        i = i + 1

    digit_or_string = False # true when its digits
    scoreboard_final_string = "Scoreboard: \n"
    for letter in l[i]:
        if letter.isdigit() == False and digit_or_string == True:
            scoreboard_final_string = scoreboard_final_string + "\n"
            digit_or_string = False
        if letter.isdigit() and digit_or_string == False:
            scoreboard_final_string = scoreboard_final_string + " "
            digit_or_string = True
        scoreboard_final_string = scoreboard_final_string + letter

    scoreboard_final_string = scoreboard_final_string[0:len(scoreboard_final_string) - 7]
    scoreboard_label.config(text=scoreboard_final_string)
    print(scoreboard_final_string)

```

```

def reader():#responsible for the main communication with the server
    global private_lobby_list
    global guess_list
    global my_turn
    global guessed_list
    while True:
        # print("I'm in a loop help me")
        rlist, wlist, xlist = select.select([client_socket], [client_socket], [])
        for current_socket in rlist:
            datar = current_socket.recv(1024).decode()
            print("server replied with: " + datar)
            # print("datar[0:4] is: " + datar[0:4])
            if datar[0:20] == "You are now the host":
                datar = datar[20:]
                print("datar iss: " + datar)
                isHost = True
                add_b_button()
                add_start_button()

            if datar[0:5] == "Lobby":
                print("private client lobby is being updated")
                private_lobby_list = datar
                update_private_lobby_list()

            if "THE GAME IS ABOUT TO BEGIN WITHOUT BOT" in datar:
                start_the_actual_game(False)
                datar = datar[38:]

```

```

if "THE GAME IS ABOUT TO BEGIN WITH BOT" in datar:
    start_the_actual_game(True)
    datar = datar[35:]
if "Your turn" in datar:
    print("my turn")
    my_turn = True
    textBoxNew.configure(state="disabled")
    start_turn(datar)
    datar = datar[9:]
if datar != "":
    if "guessed" in datar and "BOT" not in datar: # TODO: IF TIMER IS IN GUESS SHOULD UPGRADE THE IF
        print("someone guessed something")
        guess_list.append(datar)
        add_guess()
    elif "Timer:" in datar:
        print("calling the update_time_label function")
        update_time_label(datar)
    elif "Turn Over" in datar:
        print("finishfinish!!!!!!!!!!")
        finish_turn()
    if "Clear Screen" in datar:
        textBoxNew.configure(state="normal")
        bot_guess_label.configure(text="BOT:")
        clear_screen()
    if "Scored!!" in datar:
        textBoxNew.configure(state="disabled")
        datar = datar.replace(" Scored!!", "")
    if datar == client_Name:

```

```

        if datar == client_Name:
            textBoxNew.configure(state="disabled")
            guessed_list = guessed_list + datar + "\r\n"
            update_guessed_list()
    if "SCOREBOARD" in datar: |
        update_scoreboard(datar)
    if "BOT guessed:" in datar:
        update_bot_label(datar, True)
    if "BOT GOT IT RIGHT" in datar:
        update_bot_label(datar, False)

```

```
def update_bot_label(datar, bool): #adds the bot guesses to the screen
    global bot_guess_label

    print("datar is " + datar)
    if bool:
        l = datar.split(" ")
        i = 0
        for word in l:
            if word == "BOT":
                break
            i = i + 1
        bot_guess_label.config(text=l[i] + " " + l[i + 1] + " " + l[i + 2])
        print(l[i] + l[i + 1] + l[i + 2])
    else:
        bot_guess_label.config(text="BOT GOT IT RIGHT")
```

```
def copy_draw_with_bot(datar): #draws what the playing player draws in the game with the bot
    global my_canvas
    lista = datar.split("RECTDRAWING- ")
    lista.remove("")
    # print("COPY DRAW WAS CALLED")
    # print("LIST A = ")
    # print(lista)
    for coordinates in lista:
        l = coordinates.split(" ")
        x = int(l[1])
        y = int(l[3])
        my_canvas.create_rectangle(x * 13, y * 13, x * 13 + 13, y * 13 + 13, fill='black')

def copy_erase_with_bot(datar): #erases where the playing player erases in the game with the bot
    global my_canvas
    lista = datar.split("RECTERASING- ")
    lista.remove("")
    print("COPY ERASE WAS CALLED")
    # print("LIST A = ")
    # print(lista)
    for coordinates in lista:
        l = coordinates.split(" ")
        x = int(l[1])
        y = int(l[3])
        my_canvas.create_rectangle(x * 13, y * 13, x * 13 + 13, y * 13 + 13, fill='white', outline="white")
```

```

def secondary_reader():#responsible for handling the draw and erase messages
    while True:
        rlist, wlist, xlist = select.select([secondary_client_socket], [secondary_client_socket], [])
        for current_socket in rlist:
            datar = current_socket.recv(1024).decode()
            print(datar)
            if datar[0:10] == "LocToDraw:":
                print("calling copy draw")

                t3 = threading.Thread(target=copy_draw, args=(datar,))
                t3.start()
            if datar[0:11] == "LocToErase:":
                print("calling copy erase")
                t4 = threading.Thread(target=copy_erase, args=(datar,))
                t4.start()
            if "RECTDRAWING- X" in datar:
                t33 = threading.Thread(target=copy_draw_with_bot, args=(datar,))
                t33.start()
            if "RECTERASING- X:" in datar:
                t44 = threading.Thread(target=copy_erase_with_bot, args=(datar,))
                t44.start()

```

```

rules_label = Label(rules_window, text=rules, fg="white", bg="red", font='Helvetica 14 bold', justify=LEFT)
rules_label.place(x=20, y=20)

ok_button = Button(rules_window, text="OK", command=remove_rules_window)
ok_button.place(x=270, y=500)
rules_window.mainloop()

window = Tk() #THE NEXT LINE DESCRIBE THE GUI VARIABLES
window.geometry("270x150")
window.configure(bg='blue')
textBoxNew = Text(window, width=20, height=1)
textBox = Text(window, width=15, height=1)
textBox.place(x=20, y=10)
enter_Name_Button = Button(window, text="Enter Your Name", command=recieve_name)
enter_Name_Button.place(x=150, y=10)
Guess_Button = Button(window, text="->", command=Guess)
join_lobby_button = Button(window, text="Join Lobby", command=joinLobby)
lobby_list_label = Label(window, text=private_lobby_list, bg="red", fg="yellow")
guessed_list_label = Label(window, text="GUESS LIST: \r\n" + guessed_list, bg="red", fg="yellow")
start_game_button = Button(window, text="Start Game!", command=start)
my_canvas = Canvas(window, width=364, height=364, background='white')
guess_Canvas = Canvas(window, width=250, height=450, background='white')
drawButton = Button(window, text="Draw", command=drawInitiator)
eraseButton = Button(window, text="Erase", command=eraseInitiator)
my_canvas.bind('<B1-Motion>', drawOrErase)
my_canvas.bind('<ButtonRelease>', liftThePen)
time_label = Label(window, text="30", fg="purple", bg="green", font='Helvetica 18 bold')

time_label.place(x=5, y=150)
scoreboard_label = Label(window, text="scoreboard: ", fg="brown", bg="pink", font='Helvetica 11 bold')
scoreboard_label.place(x=5, y=300)
hidden_word_label = Label(window, text="", font='Helvetica 11 bold')
add_bot_button = Checkbutton(window, text="Add Bot", variable=add_bot_variable, command=add_bot)
bot_guess_label = Label(window, text="BOT: ", fg="brown", bg="pink", font='Helvetica 11 bold')
bot_guess_label.place(x=5, y=550)

t = threading.Thread(target=reader)
t.start()

t2 = threading.Thread(target=secondary_reader)
t2.start()

window.mainloop()

```

קובץ `class_names`:

קובץ הכרחי המכיל את השמות של המילים מהם נבחרות המילים אשר מועברות ומנוחשות במשחק.

תיקיית `model`:

תיקייה אשר מכילה בתוכה את מודל machine learning בו ה-BOT משתמש.

קובץ ה-`ML`:

קובץ אשר שומש בעבר על מנת ליצור ולשמור את מודל machine learning ואין צורך להריצו פעם נוספת.

תיקיית `Data`:

תיקייה המכילה את קבצי ה-NPY איתם יצרנו ואימנו את מודל machine learning.

קובץ ה-`DB_installer`:

קובץ זה אחראי על הורדת ה-DB איתו אימנו את מודל ה-MACHINE LEARNING ושמירתו בתוך תיקיית `.data`.

רפלקציה:

תחושותי מהעבודה על הפרויקט:

העבודה על הפרויקט הייתה קשה אך מפרה, אני מאמין שתמיד ליצור דבר בעצמך ולעבוד קשה עבורו זה דבר מספק ומלמד.

אני מרוצה מן התוצר הסופי וללא ספק מסופק מן העבודה הקשה על הפרויקט והתהליך הארוך.

כלים שקיבלתי ואקח איתי להמשך:

בעבודה על הפרויקט הפקתי המון ולמדתי הרבה שיעורים חשובים, שיעורים כגון כיצד לחפש מידע באינטרנט וללמוד לבד, ושיעורים מקצועיים בנושאי פיתוח machine learning. למדתי על עצמי שאני מסוגל לעבוד הרבה זמן ברצף וקשה ושיש לי מוטיבציה ומוסר עבודה גבוהים.

אתגרים שעמדו בפניי במהלך הפרויקט:

האתגר הגדול ביותר היה ללמוד בעצמי חומרים חדשים אך אתגר לא קטן נוסף היה האתגר אשר איתו מתמודד כל מתכנת והוא להתמודד עם בעיות בקוד, דבר שעלול להיות "מלחמה" של שעות בשביל שינוי שנפתר בשניות.

הפעלת הקוד שלי באופן בו הוא מתרחש (השרת מריץ בוט ולכל קובץ socket כפול) היווה אתגר שהייתי צריך לגרום לו לתפקד כראוי הדבר לא היה פשוט אך לאחר עבודה קשה הצלחתי ואין מרוצה ממני.

מסקנות מהפרויקט:

המסקנה הכי גדולה שהפקתי מן הפרויקט היא שיש לי חיבה לתחום, אין ספק שנהנתי מכתובת הקוד, גם מהחלקים אשר מוגדרים בעיני רוב כפחות כיפיים כגון התמודדות עם בעיות וחרישה של האינטרנט בשביל פתרון.

מה הייתי עושה אחרת אילו הייתי מתחיל היום

אילו הייתי מתחיל לעבוד על הפרויקט היום הדבר הכי חשוב שהייתי משנה הוא ההתנהלות בעניין ספר הפרויקט, את הדבר התחלתי רק בסיום העבודה על הקוד דבר שהיה פחות נוח מבחינת העבודה עליו ויותר לחוץ בזמן.

בנוסף כאשר נתקעתי בקוד בתהליך העבודה על דברים קטנים הייתי מעדיף בדיעבד לפנות לחבר או מורה שיהווה זוג עיניים חדשות ויעזור לי למצוא את הבעיה.

מה היה יכול להפוך את עבודתי ליעילה יותר:

קודם כל שילוב ספר הפרויקט בתהליך העבודה בשלב מוקדם יותר היה משמעותית מקל על תהליך העבודה ובנוסף אם הייתי מתייעץ יותר עם אנשים אשר מבינים את נושאי העבודה שלי בסביבתי הדבר היה מיעל את העבודה.

תכונות שהייתי רוצה להוסיף לפרויקט:

הייתי מעוניין להוסיף אופציה לBOT לשחק גם את תפקיד המצייר, בנוסף הייתי מעוניין להוסיף אופציה לכמה משחקים בכמה לובים (lobby) שונים.

לסיכום למרות כל הקשיים והאתגרים בדרך אני מאוד שמח שהלכתי בדרך בה הלכתי ובחרתי בפרויקט אותו בחרתי ואני מרוצה מאוד מהתהליך ומן התוצר הסופי!