



# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



עבודה בתכנון ותכנות מערכות

5 יחידות



**Scribble**

מערכת שיתוף ציורים

מגיש: אמיר וולברג

ת"ז: 212939631

מורים מנחים: מוטי מתיתיהו

בית ספר: אורט אבין

שנה: תש"ף 2020



# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



## תוכן עניינים

3	מסמך ייזום-שער
4	רקע
6	מדריך למשתמש
8	הגדרות
8	אתגרי ושלבי בניית הפרויקט
8	סיקור מצב השוק כיום
9	מסמך אפיון-שער
10	פונקציונליות המערכת
12	דרישות המערכת
16	מסמך עיצוב-שער
17	תיאור הלקוח
22	תיאור השרת
25	תיאור בסיס הנתונים של המערכת
26	טבלאות RFC
28	מסכי הפרויקט
35	יומן רפלקציה-שער
36	תיעוד מחקר
40	רפלקציה אישית
41	ביבליוגרפיה
42	נספחים
45	Scribble_client.py
66	Scribble_server.py



תיכון אורט אביר

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



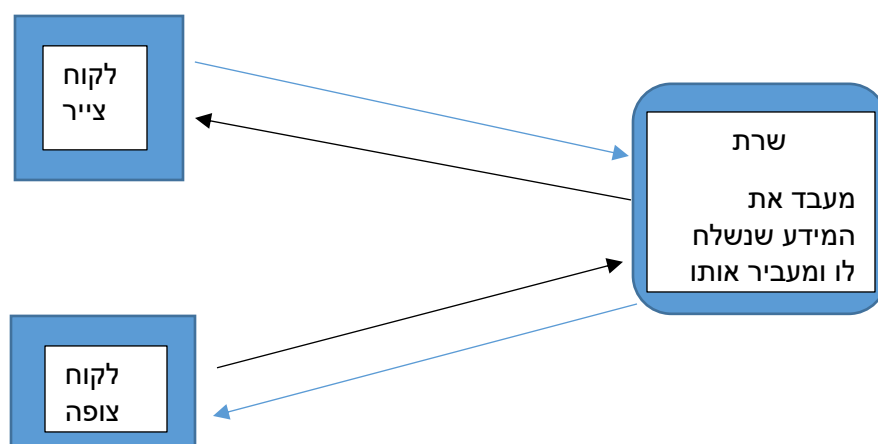
# מסמר ייזום-שער

## מבוא

בחרתי לעשות את הפרויקט הזה מכיוון שלדעתי הוא משלב בצורה טובה את הנושאים שלמדתי במגמת הנדסת תוכנה בהתמחות הגנת סייבר: רשתות, יצירת ממשקים גרפיים ועבודה עם בסיסי נתונים. בנוסף לכך גם התלהבתי מהאפשרות של יצירת משחק כלשהו מכיוון שאני אוהב לשחק במשחקי מחשב ורציתי את ההזדמנות לבנות אחד בעצמי כבר זמן מה.

## תיאור פרויקט

מערכת בארכיטקטורת שרת לקוח המאפשרת את שיתוף תהליך ופרטי הציור במסך הצייר למסך הצופה ושליחת ניחוסים של הצופה בנוגע למהותו של הציור לשרת.





# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



## הסבר על המשחק

משחק הסקריבל הוא משחק בו יש צייר המצייר דבר כלשהו ויש צופים המנסים לנחש מה הוא מצייר כמה שיותר מהר, הצופה שמנחש ראשון מה הצייר מצייר הוא המנצח.



## תיאור מהלך המשחק

מארח המשחק מפעיל את השרת, בממשק השרת הוא יכול לראות את כל הפרטים של המשחקים הקודמים השמורים במאגר הנתונים, למחוק משחקים ממאגר הנתונים, לבחור מה תהיה הכמות המקסימלית של צופים במשחק הבא ולהתחיל משחק חדש. כאשר מתחיל השרת משחק חדש יכולים משתמשים להתחבר אליו ולבחור להיות ציירים או צופים, יכולים להיות צייר אחד ובין 1 ל 9 צופים מחוברים בו זמנית (תלוי בבחירת מנהל השרת), הצייר בוחר מילה אותה הוא מצייר והצופים מנסים לנחש את המילה לפי הציור, הראשון שמנחש את הציור מנצח.

לאחר שנגמר משחק (לאחר שצופה כלשהו מנחשב את המילה הנכונה) השרת חוזר לממשק.

## הוראות להתקנה

התקן python 3.7 על המחשב

הורד את קוד הפרויקט יחד עם התמונות ההכרחיות ושים אותם באותה תיקיה  
הרץ את הלקוח או השרת (תלוי אם אתה מנהל השרת או לקוח שרוצה לשחק) דרך ה-CMD  
עם הפקודה `python scribble_client.py` והמשחק יופעל (לפני `scribble_client.py` צריך גם לרשום את הדרך אליו מהמיקום שבו אתה מפעיל את ה-CMD).  
תוכל ליצור גם Batch files שבהם הפקודות הנוכחיות בכדי לא לחזור על אותו תהליך שוב ושוב  
(הפקודות ב batch files צריכות להיות מותאמות למיקום של הקבצים במחשב שלך)



# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



## מדריך למשתמש

### מנהל שרת

#### ממשק השרת

- בכדי לראות את ה-ID של המשחקים הקיימים לוחצים על התפריט של Available game IDs
- בכדי להציג משחקים קודמים מכניסים את ה-ID שלהם ב- Display entry box ולוחצים על כפתור ה- Display או אנטר או שולוחצים על ה-ID בתפריט של Available game IDs
- בכדי למחוק משחקים קודמים מכניסים את ה-ID שלהם ב- Delete entry box ולוחצים על כפתור ה- Delete או אנטר
- בכדי לקבוע מה יהיה המספר המקסימלי של צופים במשחק הבא מכניסים את המספר הרצוי ל- Enter entry box ולוחצים על כפתור ה- Enter או אנטר
- בכדי לראות את הניחושים של כל משחק במשחק שכבר הוצג לוחצים על התפריט של Guesses of displayed games:
- בכדי לסגור את השרת לוחצים על הכפתור האדום (Power off) או על X
- בכדי להתחיל משחק חדש לוחצים על הכפתור הירוק (Start a new game)
- בכדי לרוקן את בסיס הנתונים לוחצים על הכפתור הכתום (Empty Database)



# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



## לקוח

### מסך הכניסה

- בכדי להצטרף למשחק מכניסים את השם משתמש הרצוי ובחרים תפקיד צופה או צייר (על ידי לחיצה על אחד הכפתורים watcher או painter)

- בכדי להתנתק לוחצים על X

### צופה

- בכדי לשלוח ניחוש רושמים את הניחוש ב – Guess entry box ולוחצים על הכפתור >>>> או על אנטר

- בכדי להתנתק לוחצים על X

### צייר

- בכדי לבחור מילה לצייר מכניסים את המילה ל – word entry box ולוחצים על כפתור ה – Enter או על אנטר

- בכדי להתנתק לוחצים על X או על התפריט Options ואז על Exit

- בכדי לצייר לוחצים על הכפתור השמאלי בעכבר ומזיזים אותו על הקנבס

- בכדי לשנות את עובי המכחול מזיזים את הגלגלת הכחולה בצד ימין למטה בתחתית המסך

- בכדי לשנות את צבע המכחול לוחצים על התפריט Colors בוחרים Brush color ואז בוחרים צבע ולוחצים ok

- בכדי לשנות את צבע הרקע לוחצים על התפריט Colors בוחרים Background color ואז בוחרים צבע ולוחצים ok

- בכדי למחוק את הציור לוחצים על התפריט Options ואז על Clear canvas



# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



## הגדרות

שרת: המחשב שעליו מותקנת המערכת שמאפשרת לנהל את מאגר הנתונים ולהתחיל משחקים.

מפעיל המערכת: האדם אשר אחראי על ניהול השרת דרך הממשק שלו (שמאפשר גישה ושליטה על מאגר הנתונים שבו נשמרים כל הציורים/משחקים הקודמים, שינוי המספר המקסימלי של הצופים במשחק הבא והתחלת משחק חדש).

לקוח צייר: הלקוח שמצייר את המילה שאותה צריכים הלקוחות הצופים לנחש.

לקוח צופה: הלקוח שצופה בצייר מצייר ומנסה לנחש מה הוא מצייר, הצופה הראשון שינחש נכון ינצח במשחק.

## אתגרי ושלבי בניית הפרויקט

- בניית צייר עם GUI באמצעות מודול tkinter
- שליחת כל המידע בזמן אמת לשרת והחזרתו לצופים
- עבודה עם threads בצד השרת והלקוח
- בניית הממשק הגרפי של הצופים
- שמירת כל משחק במאגר הנתונים
- בניית ממשק לשרת שבאמצעותו הוא יוכל לנהל את מאגר הנתונים

## סיקור מצב השוק כיום

את משחק ה scribble יש בהרבה גרסאות, אפשר גם לשחק לבד עם חברים מחוץ למחשב עם דף ונייר.

התוכנה שלי תאפשר גישה פשוטה יותר למשחק ואפשרות למארח המשחק (השרת) לנהל את המשחקים ולהציג משחקים ישנים.

דוגמה למשחק הנמצא בשוק כיום: <https://skribbl.io/>





תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



# מסמר אפיון-שער



# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



## פונקציונליות המערכת

(1) בצד השרת יופעל ממשק גרפי שיאפשר להסתכל על המשחקים הנמצאים במאגר הנתונים, למחוק משחקים ממאגר הנתונים ולשינוי המספר המקסימלי של הצופים במשחק הבא.

(2) הפעלת משחק - בממשק הגרפי של השרת ישנו כפתור להתחלת משחק חדש אשר סוגר את הממשק הגרפי של השרת ופותח את השרת למשתמשים אשר מתחברים לשרת:  
(1 א) הצייר מזין מילה או 2 שתאוחסן בשרת (מילה זו היא המילה אשר הצופים צריכים לנחש).  
(ב) אם הצייר משחק במצב מקוון הצייר יצייר את המילה על הקנבס השרת ישדר את הציור לצופים.

(2 א) הציור יוצג על קנבס במחשבי הלקוחות הצופים.  
(ב) המערכת בצד הלקוח הצופה תאפשר לו להזין ולשדר לשרת את הניחוש לשם הציור של הלקוח הצייר  
(ג) השרת יודיע ללקוח הצופה אם הוא ניחש את המילה הנכונה או אם הוא טעה.

(3) במקרה שהצייר עוזב טרם סיום המשחק יוכל צייר נוסף להתחבר והוא גם יאתחל את מסך הצופים, המערכת מתריעה כשצייר עוזב/מצטרף

(3) סיום המשחק: במקרה שאחד הצופים ניחש את המילה הנכונה, השרת ישלח הודעה מי ניצח לכל הלקוחות המחוברים (מפעיל וצופים).

(4) השרת ישמור רשומה חדשה בבסיס הנתונים שכוללת את: פרטי המשחק, הציור, המילה, הניחוש והזמן מתחילת המשחק שבו נשלח כל ניחוש.



# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



## (1) מסך כניסה:

- הכנסת "כינוי" שם השחקן במשחק
- בחירת תפקיד , צייר או צופה

## (2) פונקציות הצייר:

- הזנת מילה (שם, נושא הציור) שצריך הצופה לנחש
- בחירת רוחב העיפרון בין 5 ל 100
- בחירת צבע העיפרון
- בחירת צבע הרקע
- אפשרות למחיקת הציור
- אפשרות להתנתק

## (3) פונקציות הצופה:

- אפשרות להתנתק
- אפשרות לנחש את המילה הנבחרת

## (4) פונקציות מנהל השרת:

- אפשרות להתחיל משחק חדש (פותח את החיבור של השרת ללקוחות חדשים)
- אפשרות לרוקן את מאגר הנתונים
- אפשרות לכבות את השרת
- אפשרות לשנות את מספר הצופים המקסימלי במשחק הבא (בין 1 ל 9)
- אפשרות להציג משחק ממאגר הנתונים לפי ID שלו
- אפשרות למחוק משחק ספציפי ממאגר הנתונים לפי ID שלו

## אילוצים עיקריים

המערכת דורשת חיבור לאינטרנט  
ושימוש ב windows 7 ומעלה.

## דרישות המערכת

#	דרישה	קלט	תהליך	פלט	טיפול בשגיאות
1.1	הצגת משחק בממשק השרת	ה-ID של משחק	השרת ניגש למאגר הנתונים ומוציא ממנו את המידע של המשחק שאליו שייך ה-ID שהוכנס	הצגת המשחק – הציור, שם הציור והניחושים של כל משתתפי המשחק	אם מוכנס ID של משחק שלא קיים השרת יודיע למשתמש שה-ID איננו תקין ולא יציג כלום
1.2	מחיקת משחק דרך ממשק השרת	ה-ID של משחק	השרת ניגש למאגר הנתונים ומוחק את המשחק שאליו שייך ה-ID שהוכנס	המשחק שאליו שייך ה-ID שהוכנס נמחק ממאגר הנתונים	אם מוכנס ID של משחק שלא קיים השרת לא ימחק כלום
1.3	בחירת כמות הצופים המקסימלית שיוכלו להשתתף המשחק הבא	כמות הצופית המקסימלית הרצויה	השרת משנה את המשתנה הגלובלי שאחראי על כמות הצופים המקסימלית	כמות הצופים המקסימלית משתנה לפי המספר שהוכנס	אם מוכנס מספר גדול מ-9 או קטן מ-1 או אם מוכנס תו שאינו מספר יש הודעת שגיאה
1.4	ניקוי מאגר הנתונים	לחיצה על כפתור "Empty database"	השרת ניגש למאגר הנתונים ומרוקן אותו	כל המשחקים במאגר הנתונים נמחקים והוא מתרוקן	
1.5	כיבוי השרת	לחיצה על כפתור "Power off"	מבוצעת פעולת exit() והקוד מפסיק לרוץ	השרת נכבה	
1.6	פתיחת השרת למשחק חדש	לחיצה על כפתור "Start a new game"	ממשק השרת נסגר ומתחילה לרוץ המחלקה שאחראית על ניהול משחק פעיל	השרת נפתח למשתמשים להתחבר ולהתחיל משחק חדש	

#	דרישה	קלט	תהליך	פלט	טיפול בשגיאות
2	הצטרפות משתמש חדש למשחק	כינוי המשתמש ובחירת תפקיד	העברת הנתונים לשרת ושמירת הנתונים בשרת.	הלקוח מחובר לשרת	אם כבר יש צייר אז המשתמש מתבקש לחכות או לבחור תפקיד שונה . אם כבר יש מקסימום צופים אז המשתמש מתבקש לחכות או לבחור תפקיד שונה. אם הכינוי תפוס , ארוך מדי , ריק , יש בו רווח (" ") או מכיל את התו ';' מתבקש הלקוח להכניס כינוי חדש
3.1	צייר מצטרף למערכת וצריך לבחור מילה לניחוש הצופה	המילה שהצייר יצייר	העברת נתונים לשרת ושמירת הנתונים בשרת.	הצייר עובר למסך הציור	אם המילה היא 9- או ; או ריקה או מעל 20 תווים אז מתבקש הצייר להכניס מילה שונה.
3.2	צייר מחובר למערכת ומצייר	הציור שמצויר	העברת הנתונים והפרטים של הציור בזמן אמת לשרת וממנו ללקוח הצופה.	שליחת קואורדינטות, שינוי צבעים , ומחיקה לשרת וממנו ללקוח הצופה	אם הקואורדינטות הנצבעות הן מעל 999 או מתחת ל 99- אז הן לא נשלחות (כי הן מחוץ לתחום הלוח ואסור לשלוח קואורדינטות שהן מעל 3 ספרות או שההודעה ארוכה מדי)

#	דרישה	קלט	תהליך	פלט	טיפול בשגיאות
4	צופה מצטרף למערכת ומתחיל לנחש	הניחושים של הצופה	הצופה מקבל מהשרת את עדכוני הציור בזמן אמת	הניחושים של הצופה עוברים לשרת ונבדק אם הוא צודק או לא	אם המילה לא נכונה השרת משווה אותה למילה ורואה זאת ואז מודיע ללקוח שהוא טעה ושינסה שוב  אי אפשר לשלוח ; בתור ניחוש  אם הצופה נכנס באמצע המשחק השרת שולח לו את כל מה שצויר עד כה + מידע ממה שצויר בזמן אמת
5	לקוח (צייר או צופה) מתנתק באיזשהו שלב מהשרת	לקוח התנתק	הלקוח סוגר את חלון המשחק מה שמודיע לשרת שהוא התנתק	השרת מיידע את שאר הלקוחות שלקוח עזב (מציין אם זה צייר או צופה)	למקרה שנשלח מידע ללקוח שהתנתק יש שימוש ב except ו לכל תהליך שבו נשלח מידע ללקוחות
6	צופה ניחש את המילה הנכונה והמשחק נגמר	ניחוש של המילה הנכונה מהצופה	המילה הנכונה עוברת לשרת שמודיע לכולם מי ניחש אותה וכמה זמן לקח לו	כל המשתתפים עוברים למסך בסיום שבו רשום את שם המנצח ואת הזמן שלקח לו לנחש את המילה	אם 2 או יותר צופים מנחשים נכון בדיוק באותו הזמן המנצח נבחר ברנדומליות בתור אחד משניהם
7	השרת מאותחל וכל התהליך חוזר	נגמר משחק	הצופים מודיעים לשרת שהם במסך סיום ואפשר להכניס את נתוני המשחק למאגר הנתונים ולאתחל	הנתונים של המשחק נשמרו במאגר הנתונים והשרת מאותחל	בכדי שהשרת יצא מהלולאה בפונקציה "Incoming connections" הוא צריך שיתחבר לפחות לקוח חדש אחד לאחר שנגמר המשחק , לכן הצופים כולם מתנתקים ומתחברים מחדש כדי לשלוח את ההודעה לשרת לאתחל את עצמו



# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



## דרישות בסיס נתונים

בסיס הנתונים ימומש בשפת python 3.7 בעזרת יבוא המודל sqlite3 (שתמיר את המחרוזות בפיתון ל-sql) וישמרו בו פרטי הציור, הניחושים של כל צופה ובאיזה זמן הם נשלחו מתחילת המשחק ושם הציור אותו צריכים הצופים לנחש.

## סביבת עבודה

התוכנית נכתבה ב-python 3.7

הממשק הגרפי נבנה באמצעות tkinter

בסיס הנתונים נבנה על ידי שימוש ב-sqlite



תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



# מסמר עיצוב- שער





# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



## תיאור הלקוח

### Scribble – Client.py

#### Imports

```
from tkinter import *  
from tkinter import ttk, colorchooser  
ספרייה השמשה לבניית הממשק הגרפי  
  
import socket  
ספרייה השמשה להקמת חיבור  
  
import threading  
ספרייה השמשה להרצת כמה תהליכונים במקביל  
  
import time  
ספרייה השמשה למדידת הזמן  
  
from PIL import ImageTk, Image  
ספרייה השמשה להצגת תמונות
```



# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



## Classes

### UserLogin

אחראי על פונקציות מסך הכניסה

```
▼ UserLogin
  m __init__(self, master)
  m add_watcher(self)
  m addPainter(self)
  m retrieve_input(self, role)
  f Role
  f User
  f master
  f painter_button
  f user_name
  f user_role
  f watcher_button
  f welcome
```

### PainterChoose

אחראי על פונקציות מסך בחירת המילה של הצייר

```
▼ PainterChoose
  m __init__(self, master)
  m add_text(self)
  m enter_press_log(self, event)
  m retrieve_input(self)
  f chosen_word
  f error1
  f error2
  f master
```



# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



Painter

אחראי על פונקציות הצייר

```
▼ C Painter
  m __init__(self, master)
  m painter_threading(self)
  m receive_data(self)
  m paint(self, e)
  m reset(self, w)
  m change_width(self, e)
  m clear(self)
  m change_fg(self)
  m change_bg(self)
  m draw_widgets(self)
  f c
  f color_bg
  f color_fg
  f color_menu
  f controls
  f game_over
  f master
  f menu
  f old_x
  f old_y
  f pen_width
  f resetting_mouse
  f slider
  f time_of_game
  f watcher_label
  f watchers_logged
  f winner_name
```



# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



Watcher

אחראי על פונקציות הצופה

```
▼ Watcher
  m __init__(self, master, w)
  m watcher_info(self)
  m end_screen(self)
  m add_text(self)
  m retrieve_input(self)
  m enter_send(self, event)
  m drawn_so_far(self)
  f Background_color
  f Guess_sent
  f Painted_Coordinates
  f Pen_color
  f Winner
  f action_list
  f enter_button
  f enter_guess
  f game_time
  f guess
  f guess_word
  f master
  f old_x
  f old_y
  f painter_logged
  f sent_guess
  f still_redrawing
  f w
```



# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



## Main code functions

### On\_closing\_watcher

מטפל במה שקורה כשצופה מתנתק

```
f on_closing_watcher(socket_)
```

### painter\_chosen

מטפל במה שקורה כשלקוח הוא הצייר

```
f painter_chosen()
```

### watcher\_chosen

מטפל במה שקורה כשלקוח הוא הצופה

```
f watcher_chosen()
```



# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



## תיאור השרת

### Scribble – Server.py

#### Imports

```
import socket
ספרייה השמשה להקמת חיבור

import threading
ספרייה השמשה להרצת כמה תהליכים במקביל

import select
ספרייה השמשה לקבלת socket ממשתמשים מרובים בזמנים שונים

import time
ספרייה השמשה למדידת הזמן

import sqlite3
ספרייה השמשה ליצירת תקשורת עם בסיס הנתונים

from tkinter import *
ספרייה השמשה לבניית הממשק הגרפי

from functools import partial
ספרייה השמשה להכנסת פונקציה עם משתנים בתור פקודה ללחיצת כפתור
```



# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



## Classes

### PaintServer

אחראי על מהלך המשחק מתקשורת בין לקוחות לשמירת פרטי המשחק במאגר הנתונים

```
▼ C PaintServer
  m __init__(self)
  m incoming_connection(self)
  m painter_client(self, painter_socket)
  m watcher_client_receive(self, watcher_socket, watcher_name)
  m game_ended(self)
  m game_over_socket(self, socket_)
  f Background_color
  f Painted_Coordinates
  f Pen_color
  f Previous_Coordinates
  f Watchers_logged
  f Winner
  f database_information
  f game_over
  f game_started
  f game_time
  f guess_list
  f guess_word
  f painter_chosen
  f painter_socket
  f start_of_game
  f watcher_list
  f watcher_name_list
  f watchers_enter_database
```



# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



Manager

אחראי על ממשק השרת ועל הפונקציות שכוללות ניהול מאגר נתונים והתחלת משחק חדש

```
▼ C Manager
  m __init__(self)
  m enter_press_log_watcher(self, event)
  m choose_watcher_num(self)
  m insert_game(self, id_of_game)
  m show_game(self, game_id)
  m enter_press_log_show(self, event)
  m enter_press_log_delete(self, event)
  m delete_all_games(self)
  m delete_a_game(self, id_delete)
  m reset_game_guesses_menu(self, game_id_renew)
  f action_label
  f background_color
  f close_server
  f command_list
  f delete_button
  f delete_id
  f delete_id_button
  f enter_button
  f file_menu
  f game_number
  f guess_menu
  f label_watcher_error
  f master
  f menu_bar
  f old_x
  f old_y
  f painting_name
  f pen_color
  f shown_games_list
  f start_button
  f w
  f watcher_button
  f watcher_numb
```



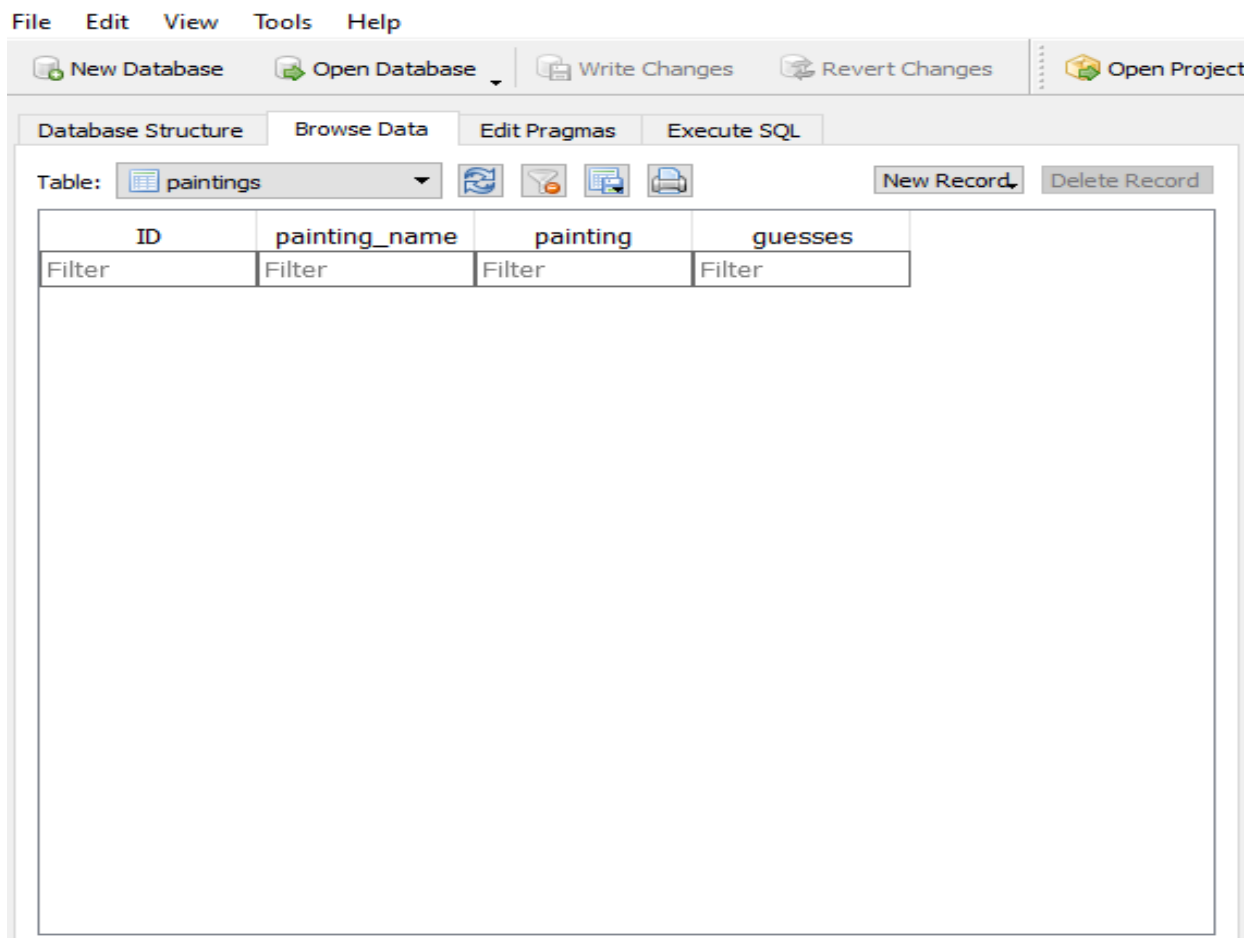


# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



## תיאור בסיס הנתונים של המערכת



**ID** - מיוצר ברנדומליות לכל משחק נועד לזיהוי המשחק

**Painting\_name** - שם הציור (המילה שבחר הצייר שאותה צריכים הצופים לנחש)

**Painting** - פרטי הציור כפי שהוא נראה בסוף המשחק (קואורדינטות צבועות וצבעים)

**Guesses** - הניחושים של כל משתמש אשר מוצגים בתבנית הבאה

(שם הצופה => ניחוש נכון/מוטעה : הניחוש, זמן הניחוש)

לדוגמה : 0:34, wrong guess : cat => Bob



# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



טבלאות RFC

שליחת מידע בין הלקוחות והשרת

## צייר לשרת

קלט	משמעות הקלט
the chosen word is: *the word*	המילה או 2 שבחר הצייר לצייר
3-	הצייר התנתק
9-	צייר חדש נכנס – לאפס את מסכי הצופים משמש גם לבדוק אם הצייר התנתק בתהליך בחירת המילה , אם הוא התנתק בתהליך בחירת המילה אז 9- ישלח בתור המילה והשרת ידע שהוא התנתק
1-	הצייר מחק את הציור
15 עד 26	להתכונן לקבלת הקואורדינטות שהצייר צבע (15 עד 26 אורך ההודעה שתשלח) (עם עובי המכחול)
coordinates: *coordinates painted and brush width + whether the line is reset or not*	הקואורדינטות שנצבעו ועובי המכחול + האם צובעים המשך של קו קודם או שמתחילים אחד חדש
14	הצייר שינה את צבע המכחול להתכונן לקבלת צבע המכחול החדש
color: *new brush color*	הצבע שאליו שינה הצייר את המכחול
28	הצייר שינה את צבע הרקע להתכונן לקבלת צבע הרקע החדש
background_color_is: *new background color*	הצבע שאליו שינה הצייר את הרקע

## צופה לשרת

קלט	משמעות הקלט
end	נגמר המשחק והצופה עבר בהצלחה למסך הסיום
/close	הצופה התנתק מהמשחק
guessed: *the guess*	הניחוש ששלח הצופה בנוגע לציור
reset server	הצופה עבר בהצלחה למסך הסיום השרת יכול לאתחל



# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



## השרת לצייר

קלט	משמעות הקלט
num : *number of watchers online*	מספר הצופים שמחוברים ברגע שמתחבר הצייר
+watcher	צופה התחבר
-watcher	צופה התנתק
game over	צופה ניחש את המילה ונגמר המשחק , להתכונן לשליחת שם המנצח והזמן שלקח לו

## השרת לצופה

קלט	משמעות הקלט
Game information incoming	להתכונן לקבלת פרטי הציור והמשחק עד לרגע שבו הצטרף הצופה
wrong	הניחוש שנשלח לא נכון
/close	השרת מאשר שהוא קיבל את הודעות ההתנתקות של הצופה ומאפשר לצופה להתנתק לגמרי
painter logged on	הצייר מחובר
painter logged off	הצייר מנותק
game over	המשחק נגמר לעבור למסך הסיום ולהתכונן לקבלת פרטי המנצח וזמן המשחק
Game full	המשחק מלא
Pen: *new brush color*	הצבע שאליו שינה הצייר את המכחול
Background: *new background color*	הצבע שאליו שינה הצייר את הרקע
Delete	הצייר מחק את הציור
reset_screen	צייר חדש נכנס לאפס את המסך
Coordinates: *newly painted coordinates and brush width + whether the line is reset or not*	הקואורדינטות שהצייר צבע ועובי המכחול + האם צובעים המשך של קו קודם או שמתחילים אחד חדש

## השרת ללקוח לפני שהוא מאושר בתור צייר או צופה

קלט	משמעות הקלט
Game full	המשחק מלא (אין מקום לעוד צופים)
you are now the painter	הלקוח הוא עכשיו הצייר
Painter already chosen	יש כבר צייר
Name Taken	השם שבחר הצופה כבר תפוס
You joined	הלקוח מחובר בתור צופה



# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



## מסכי הפרויקט

### ממשק השרת

Scribble server

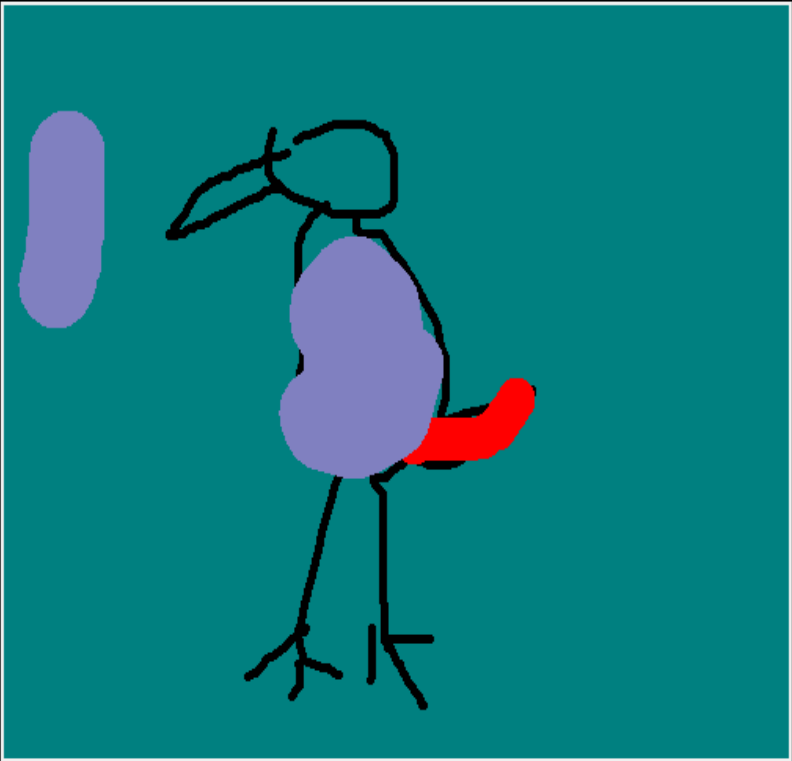
Available game IDs:    Guesses of displayed games:

Enter the ID of the game you want to display:

Enter the ID of the game you want to delete:

Choose how many watchers can play next game (default 5)

watcher number must be between 1 and 9



parrot

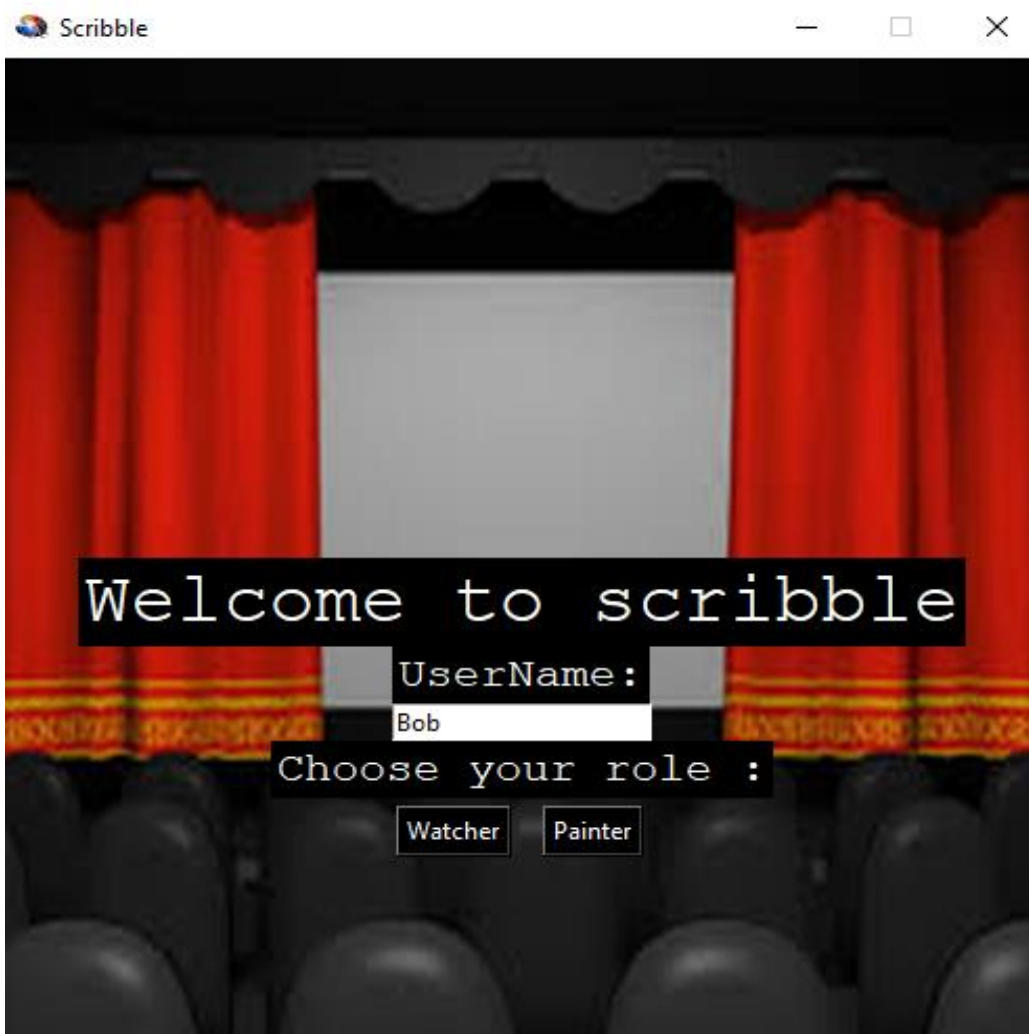


# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



## מסך פתיחה



## ממשק הצופה





# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



## מסך סיום הצופה



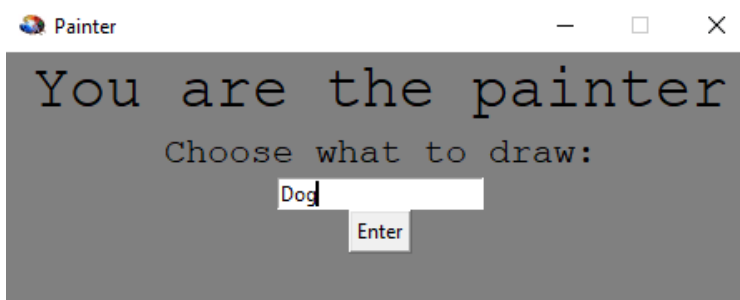


# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



מסך בחירת הציור של הצייר





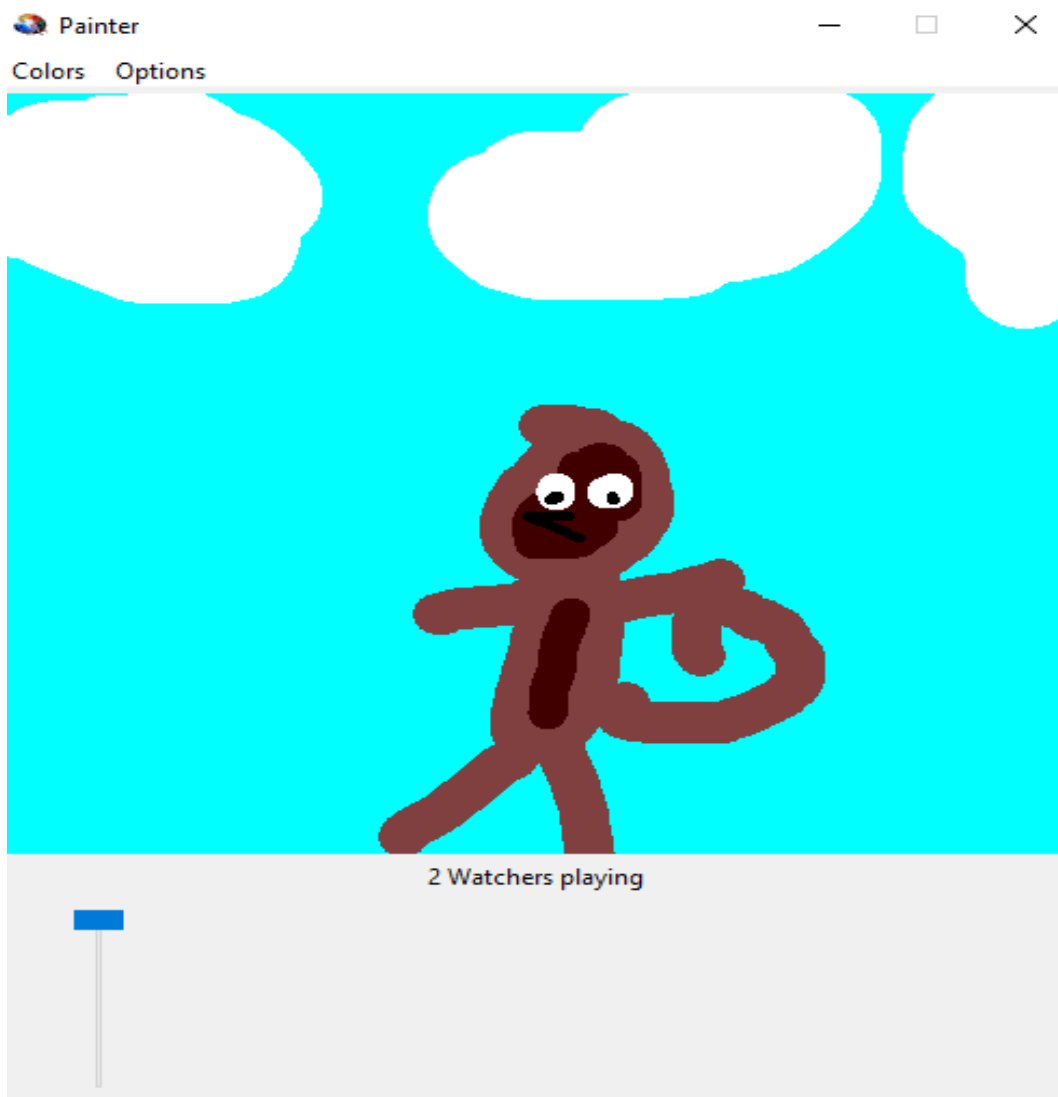


# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



## ממשק הצייר



## מסך סיום הצייר





תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



# יומן רפלקציה- שער



# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



## תיעוד מחקר

### בחירת ספריה לממשק הגרפי

#### תיאור הבעיה למחקר-

מציאת ספריה שמממשת ממשק גרפי שעליו יהיה ניתן לצייר בקלות ולשלוח ממנו מידע (הודעות ופקודות) לשרת.

#### מקורות מידע רלוונטים-

[https://www.python-course.eu/tkinter\\_canvas.php](https://www.python-course.eu/tkinter_canvas.php)

<https://stackoverflow.com/questions/35662844/python-tkinter-entry-get>

#### מסקנה-

נמצאה ספריה אשר מאפשרת קבלת מידע מהמשתמש לשליחה בקלות ויש לה יישומון של קאנבאס המאפשר ציור על פניו בקלות.

#### יישום בפרויקט-

`from tkinter import *`



# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



## שליחת וקבלת מידע לשרת בו זמנית מהצייר

### תיאור הבעיה למחקר-

הצייר צריך לשלוח את הציור לשרת ותוך כדי לקבל מידע מהשרת בנוגע למהלך המשחק.

### מקורות מידע רלוונטיים-

<https://stackoverflow.com/questions/2957116/make-2-functions-run-at-the-same-time>

### מסקנה-

באמצעות ספריית התהליכונים זה אפשרי להריץ 2 פונקציות במקביל (כשאחת לא עושה כלום השניה מופעלת ולהפך זה לא לגמרי במקביל אבל עובד למטרה שלי).

### יישום בפרויקט-

`import threading`



# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



## האטת השרת עקב כמות גבוהה מדי של תהליכונים שרצים במקביל

תיאור הבעיה למחקר-

השרת הריץ יותר מדי threads במקביל מה שגרם להאטה רצינית של שליחת המידע והמשחק.

מקורות מידע רלוונטים-

<https://stackoverflow.com/questions/2957116/make-2-functions-run-at-the-same-time>

<https://steelkiwi.com/blog/working-tcp-sockets/>

מסקנה-

השתמשתי ביותר מדי תהליכונים בלי סיבה, במקום להריץ 2 תהליכונים אחד שאחראי על שליחת מידע לצופים כל פעם שמתקבל מידע מהצייר ואחד שאחראי על קבלת מידע מהצייר אני יכול להריץ תהליכון אחד שמקבל מידע מהצייר ומיד מעביר אותו לכל הלקוחות הצופים.

יישום בפרויקט-

```
thread_transmit = threading.Thread(target=self.painter_client, args=(client_socket,))  
thread_transmit.start()
```

תהליכון שאחראי על קבלת המידע מהצייר והעברתו לצופים, מעביר לצופים בעזרת רשימה שבה כל sockets של כל הצופים כך:

```
for watcher in self.watcher_list:  
    watcher.send(data.encode())
```



# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



## הצגת משחקים ישנים בממשק השרת

### תיאור הבעיה למחקר-

בכדי להציג משחקים ישנים מממשק השרת צריך לשמור את המידע של המשחקים גם כשהוא לא פועל (כלומר אי אפשר לשמור את המידע במשתנים כגון רשימה).

### מקורות מידע רלוונטים-

<https://realpython.com/python-sql-libraries/#sqlite>

<https://sqlitebrowser.org/>

### מסקנה-

אפשר לשמור את המשחקים הישנים במאגר נתונים שבו הם ישמרו גם לאחר כיבוי השרת ושאליו ניתן לגשת מהשרת.

### יישום בפרויקט-

```
import sqlite3
```



# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



## רפלקציה אישית

תהליך יצירת הפרויקט היה מאתגר ומהנה עבורי. החל ממצאת רעיון למה לעשות ועד לתיקון הבאגים האחרונים הפרויקט היווה עבורי אתגר שכלי מה שגרם לכך שכל פעם שהצלחתי לעשות משהו והתגברתי על מכשול הרגשתי תחושת סיפוק רבה.

החלק הכי קשה עבורי היה ההתחלה, לבנות בסיס לכל הפרויקט שממנו אני אתקדם ואבנה את כל השאר לא היה צעד טריוויאלי ולקח לי הרבה זמן עד שהצלחתי להחליט ממה להתחיל. בסוף החלטתי להתחיל עם בניית צייר רגיל (ללא רשתות) ולהתקדם משם וזה עבד.

הכי אהבתי לבנות את ממשק השרת ולחשוב כיצד לעצב אותו בצורה הטובה ביותר ואילו פונקציות אוכל להוסיף לממשק. הסיבה שנהנתי כל כך מחלק זה היא שבניתי את ממשק השרת לאחר שכבר התנסיתי הרבה עם בניית ממשקים אצל הלקוח ככה שידעתי מה אני עושה ולא הייתי צריך לחפש כל כמה דקות פקודות שרירותיות. מה שנתן לי את האפשרות להתרכז בלפתח ולעצב את הממשק ובלחקור נושא חדש שעניין אותי בזמנו מאגר הנתונים שגם אותו הייתי צריך לשלב בממשק מנהל השרת.

לסיכום הפרויקט תרם לי רבות ברמה המקצועית וברמה האישית ולבנות אותו הייתה חוויה לימודית מעולה.

## תודות

הייתי רוצה להודות קודם כל למורה שלי מוטי מתיתיהו שעזר לי הרבה עם באגים שהופיעו לי במהלך כתיבת הקוד, עם הרעיון של הפרויקט ועם בניית הספר פרויקט.

בנוסף לכך הייתי רוצה גם להודות לחבריי לכיתה שעזרו לי עם חומרים שפחות הבנתי והם כבר שילבו בפרויקטים שלהם ועם באגים שהם כבר התגברו עליהם בעבר.

ולסיום הייתי רוצה להודות לאנשים עלומי שם רבים באינטרנט שחוו בעיות דומות לבעיות שאני חוויתי במהלך בניית הפרויקט ורשמו כיצד להתמודד איתן.





# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



## ביבליוגרפיה

שם הכותב: Dan XXX

נושא: Sqlite3 library

מקור: <https://realpython.com/python-sql-libraries/#sqlite>

שם הכותב:

נושא: שאלה על threads ב-

chrissygormley

stack overflow

מקור: <https://stackoverflow.com/questions/2957116/make-2-functions-run-at-the-same-time>

שם הכותב:

נושא: הורדה של db browser של

Sqlite

מקור: <https://sqlitebrowser.org/>

שם הכותב: python tkinter  
course

נושא: tkinter canvas widget

מקור: [https://www.python-course.eu/tkinter\\_canvas.php](https://www.python-course.eu/tkinter_canvas.php)

שם הכותב: nbro

נושא: שאלה על איך לגשת למידע

שנקלט על ידי ה - entry widget ב

tkinter מ - stack overflow

מקור: <https://stackoverflow.com/questions/35662844/python-tkinter-entry-get>

שם הכותב: geeksforgeeks

נושא: הסבר על sockets בפיתון

מקור: <https://www.geeksforgeeks.org/socket-programming-python/>

שם הכותב: tobias\_k

נושא: שאלה על ציור קווים על

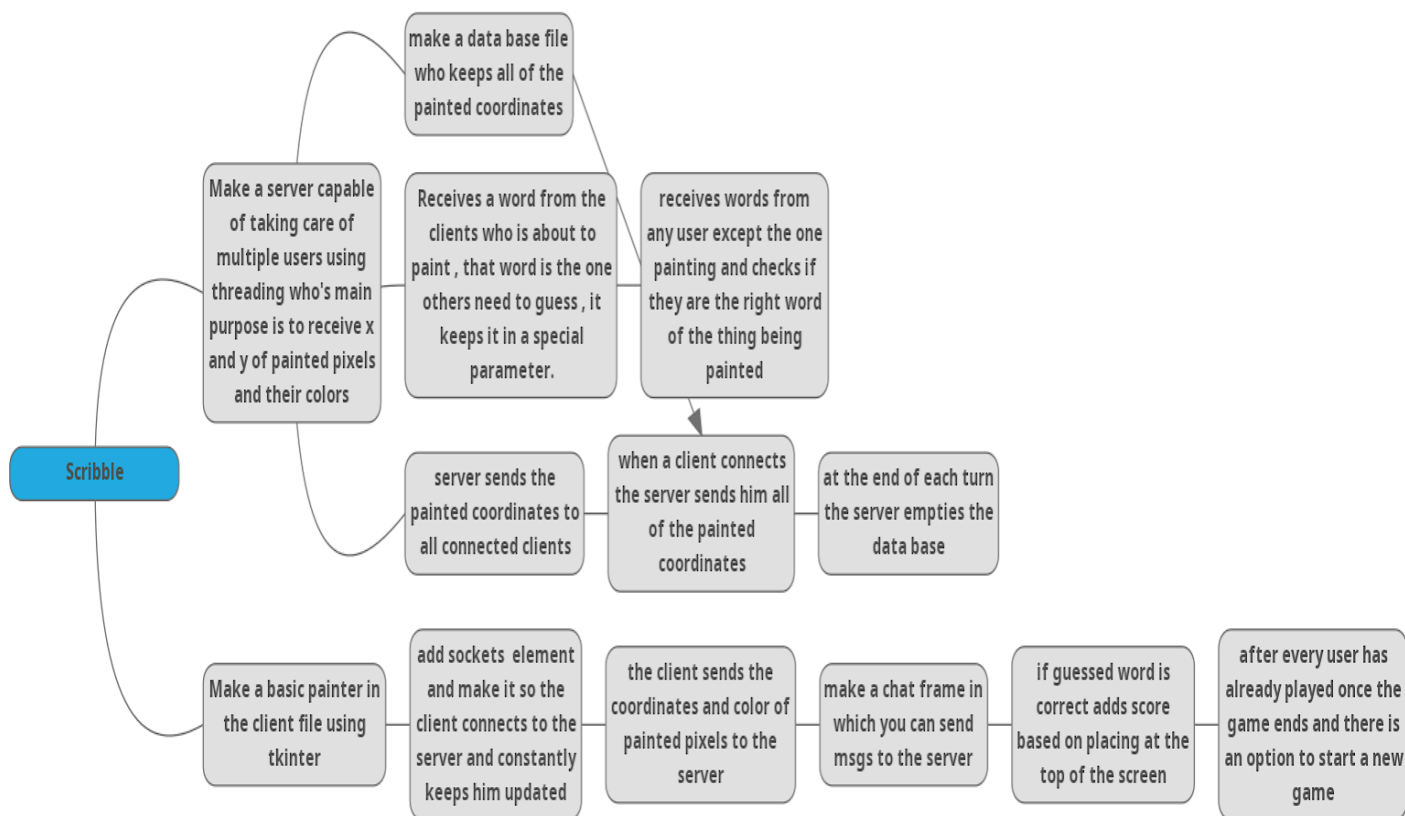
canvas ב - tkinter מ - stack

overflow

מקור: <https://stackoverflow.com/questions/25701347/how-to-draw-a-line-on-a-canvas>

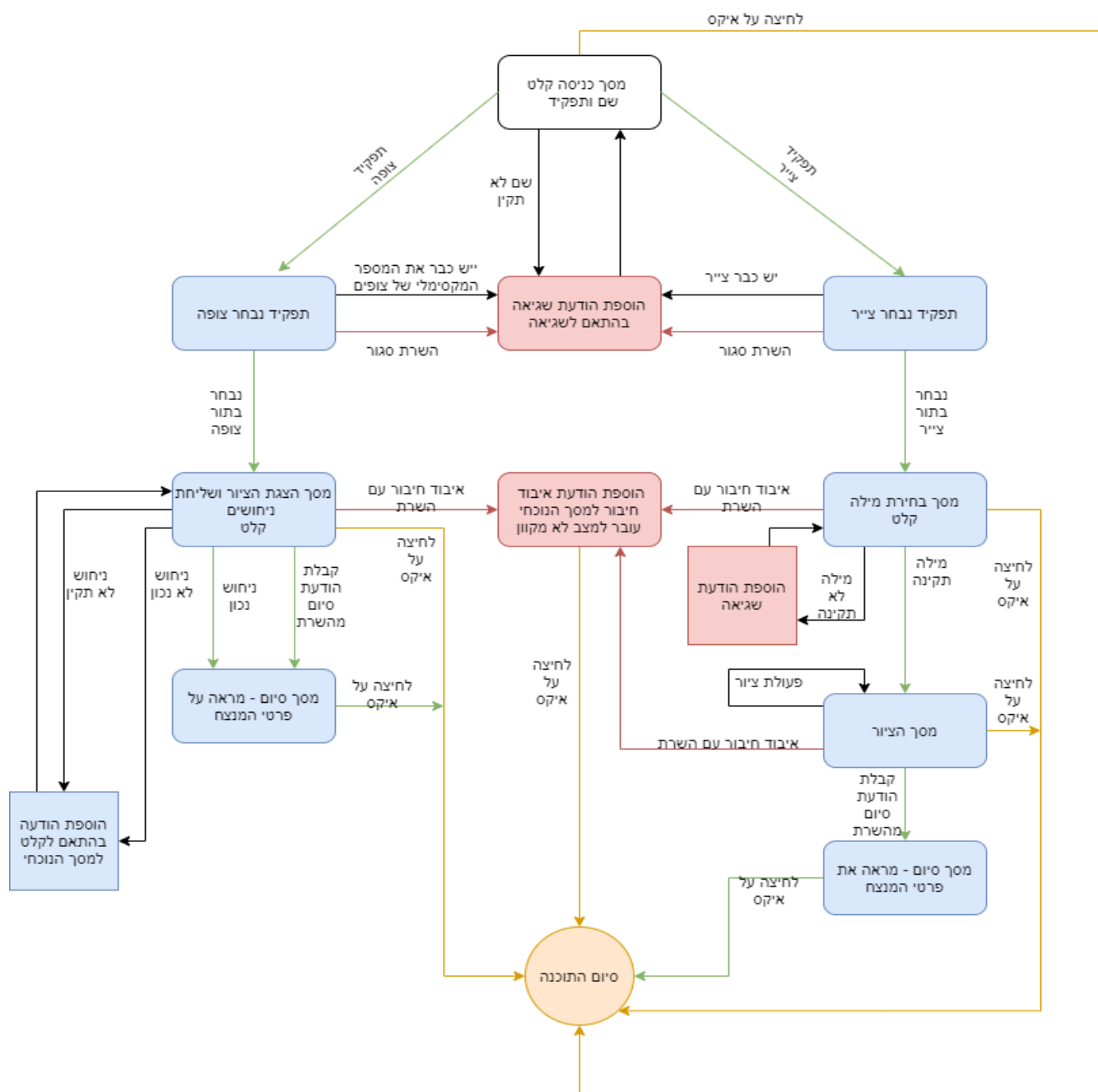
## נספחים

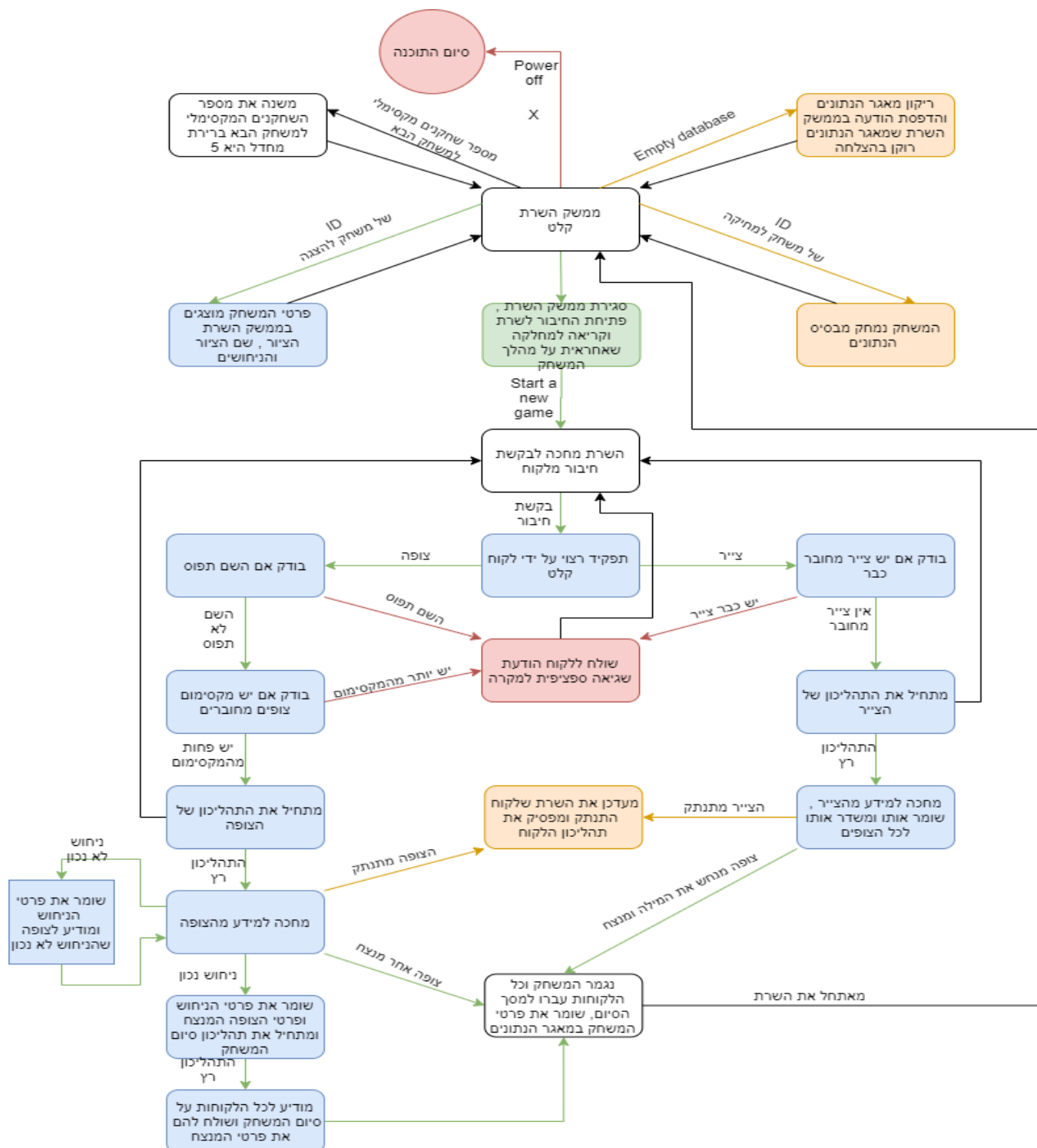
### מפת חשיבה



## תרשימי זרימה

### לקוח







# תיכון אורט אביר

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



[Scribble\\_client.py](#)

```
"""
Made by Amir Wolberg
Classes used: socket, thread, time, PIL and tkinter
"""

from tkinter import *
from tkinter import ttk, colorchooser
import socket
import threading
import time
from PIL import ImageTk, Image

"""
No known bugs - QA required
add transparency
"""

"""
classes - each a different interface ((login screen) , (painter word choose) , (painter canvas)
and (watcher canvas+guessing))
"""

# User code

class UserLogin:
    """
    takes care of the login screen widgets and information sending , keeps the information of the user name
    and user role in global variables
    """

    def __init__(self, master): # master is the root , the window
        """
        defining class variables and building the screen widgets of the login screen
        :param master:
        """
        self.master = master
        self.User = ""
        self.Role = ""

        self.welcome = Label(self.master, text='Welcome to scribble', bg='black', fg='white', font=("Courier", 28))
        self.welcome.pack()
```



# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



```
self.user_name = Label(self.master, text="UserName:", bg='black', fg='white', font=("Courier", 16))
self.user_name.pack()

self.user_name = Entry(self.master, bd=1)
self.user_name.pack()

self.user_role = Label(self.master,
                        text='Choose your role :', bg='black', fg='white', font=("Courier", 16))
self.user_role.pack()

self.watcher_button = Button(self.master, text="Watcher", command=self.add_watcher, bg='black',
fg='white')
self.watcher_button.pack()
self.watcher_button.place(x=190, y=380)

self.painter_button = Button(self.master, text="Painter", command=self.addPainter, bg='black',
fg='white')
self.painter_button.pack()
self.painter_button.place(x=260, y=380)

def add_watcher(self):
    """
    puts the username chosen and the role watcher into the global variables for name and role
    destroys the login window root
    """
    self.retrieve_input('watcher')
    self.master.destroy()

def addPainter(self):
    """
    puts the username chosen and the role painter into the global variables for name and role
    destroys the login window root
    """
    self.retrieve_input('painter')
    self.master.destroy()

def retrieve_input(self, role):
    """
    puts the information entered by the user into global variables that are then sent to the server
    in the main code
    """
    global Role
    global User
    User = self.user_name.get()
    Role = role

# Painter code
```



# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



```
class PainterChoose: # Handles the painter choosing a word to guess
    """
    takes care of the word choosing screen of the painter
    """

    def __init__(self, master): # master is the root , the window
        """
        defining class variables and building the screen widgets of the word choosing screen
        :param master:
        """

        self.master = master
        self.welcome = Label(self.master, text='You are the painter', bg='Grey', font=("Courier", 20))
        self.welcome.pack()

        self.chosen_word = Message(self.master, text="Choose what to draw:",
                                    bg='Grey', font=("Courier", 14), width=450)
        self.chosen_word.pack()

        self.chosen_word = Entry(self.master, bd=1)
        self.chosen_word.pack()
        self.chosen_word.bind('<KeyPress>', self.enter_press_log)

        enter_button = Button(self.master, text="Enter", command=self.add_text)
        enter_button.pack()

        self.error1 = Label(self.master, text="", bg='Grey', font=("Courier", 14))
        self.lost_connection = Label(self.master, text="", fg='red', font=("Courier", 12))

        self.master.protocol("WM_DELETE_WINDOW", lambda: exit())

    def add_text(self):
        """
        checks if the chosen word is valid and if it is calls the function that sends it to the server
        """

        if len(str(self.chosen_word.get())) > 20: # if the word is longer than 20 characters
            self.error1.destroy()
            self.error1 = Label(self.master, text='The word must be 20 characters or less', font=("Courier", 14))
            self.error1.pack()

        elif len(str(self.chosen_word.get())) == 0: # if the word is blank
            self.error1.destroy()
            self.error1 = Label(self.master, text='Word must not be blank', font=("Courier", 14))
            self.error1.pack()

        elif str(self.chosen_word.get()) == '-9':
            # if the word is -9 which is the code for logging off and resetting watcher screen
            self.error1.destroy()
```



# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



```
self.error1 = Label(self.master, text='Word must not be -9', font=("Courier", 14))
self.error1.pack()

elif ';' in str(self.chosen_word.get()):
    # if the word has ; which is used to separate guesses in the server
    self.error1.destroy()
    self.error1 = Label(self.master, text='Word must not contain ;', font=("Courier", 14))
    self.error1.pack()

else: # if the word is valid
    self.retrieve_input()

def enter_press_log(self, event):
    """
    if the user presses enter calls the add_text function
    :param event:
    """
    if event.keycode == 13:
        self.add_text()

def retrieve_input(self):
    """
    sends the chosen word (to draw) to the server
    """
    print("the chosen word is: " + self.chosen_word.get())
    word = self.chosen_word.get()
    # noinspection PyBroadException
    try:
        s.send((str(word)).encode())
        self.master.destroy()
    except Exception:
        print('connection lost...')
        self.lost_connection.destroy()
        self.lost_connection = Label(self.master, text="Connection lost...", fg='red', font=("Courier", 12))
        self.lost_connection.pack()

class Painter:
    """
    takes care of the painter screen , widgets and painting
    and also sends all of the painting information to the server
    """

    def __init__(self, master): # master is the root , the window
        """
        defining class variables and building the screen widgets of the painter
        :param master:
        """
        self.master = master
```





# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



```
# values for parameters used to paint
self.color_fg = 'black'
self.color_bg = 'white'
self.pen_width = 5
self.old_x = None
self.old_y = None

# values used in drawing the widgets later
self.controls = None
self.slider = None
self.menu = Menu(self.master)
self.color_menu = Menu(self.menu)

# values received from or sent to the server
self.winner_name = '' # the name of the winner of the game
self.time_of_game = '' # the length of time it took for the game to end
self.resetting_mouse = '1' # 0 meaning the mouse is not reset, 1 meaning it is
self.game_over = False # if the game is over turns to True
self.watchers_logged = 0 # number of watchers playing the game

# setting up the canvas and label that shows the number of watchers watching
self.c = Canvas(self.master, width=500, height=420, bg=self.color_bg, )
self.c.pack(fill=BOTH, expand=False)
self.watcher_label = Label(self.master, text=str(self.watchers_logged) + ' Watchers playing')
self.watcher_label.pack()

# calls the function responsible for the threads of the painter class
self.painter_threading()

# binding the mouse as the drawing tool
self.c.bind('<B1-Motion>', self.paint) # drawing the line
self.c.bind('<ButtonRelease-1>', self.reset) # when you release the button it stops painting

def painter_threading(self):
    """
    handles the threads of the painter class
    """
    thread_gui = threading.Thread(target=self.draw_widgets) # takes care of drawing the painting and
    # sending it
    thread_end_game = threading.Thread(target=self.receive_data) # takes care of receiving data from the
    # server
    thread_gui.start()
    thread_end_game.start()

def receive_data(self):
    """
    takes care of any data sent from the server to the painter
    also builds new widgets or changes old ones according to what the server instructs
    """
```



# תיכון אורט אב"ח

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



```
while True:
    # noinspection PyBroadException
    try:
        dataPainter = s.recv(10)
        dataPainter = dataPainter.decode()
        print(dataPainter)

        if dataPainter.startswith('num:'): # tells the painter how many watchers logged before he did
            self.watchers_logged = int(dataPainter[5:])
            print(str(self.watchers_logged) + ' Watchers playing')
            self.watcher_label.configure(text=str(self.watchers_logged) + ' Watchers playing')

        if dataPainter.startswith('+watcher'): # informs the painter a watcher joined
            self.watchers_logged += 1
            self.watcher_label.configure(text=str(self.watchers_logged) + ' Watchers playing')

        if dataPainter.startswith('-watcher'): # informs the painter a watcher left
            self.watchers_logged -= 1
            self.watcher_label.configure(text=str(self.watchers_logged) + ' Watchers playing')

        if dataPainter.startswith('game over'): # tells the painter someone guessed the word and game over
            dataWinner = s.recv(4096)
            dataWinner = dataWinner.decode()
            print('the winner is ' + dataWinner)
            dataTimeOfGame = s.recv(4096).decode()
            self.time_of_game = dataTimeOfGame # the time it took from the moment the painter joined until
            # someone guessed the word and won the game
            self.game_over = True
            self.winner_name = dataWinner # the name of the winner

        # Building the leader boards screen of the painter
        self.master.configure(background='blue')
        self.c.delete(ALL)
        self.watcher_label.destroy()
        self.slider.destroy()
        self.controls.destroy()
        self.menu.destroy()
        self.color_menu.destroy()
        self.master.geometry("500x500")

        # takes care of the leader boards image background
        pic_2 = Image.open("leader_boards.png")
        img_2 = pic_2.resize((500, 500), Image.ANTIALIAS)
        img_2.save("leader_boards.png")
        pic_2 = Image.open("leader_boards.png")
        tk_pic_2 = ImageTk.PhotoImage(pic_2)
        end_screen_label = Label(self.master, image=tk_pic_2)
        end_screen_label.place(x=0, y=0, relwidth=1, relheight=1)
        self.c.pack()
```



# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



```
# takes care of the leader boards labels showing the winner's name and time
label1 = Label(self.master, text="Game Over!", bg='White', fg='Black', font=("Courier", 35))
label1.pack()
label2 = Label(self.master, text="The winner is:", bg='White', fg='Black', font=("Courier", 16))
label2.pack()
label2.place(x=163, y=50)
label3 = Message(self.master, text=self.winner_name, bg='Yellow', fg='Black', font=("Courier", 14),
                  width=150)
label3.pack()
label3.place(x=195, y=80)
label4 = Label(self.master, text=self.time_of_game, font=("Courier", 12), bg='Black', fg='White')
label4.pack()
label4.place(x=228, y=279)

s.close() # game is over disconnecting from the server
break
except Exception:
    print('connection with server lost...')
    lost_connection = Label(self.master, text="Connection lost...", fg='red', font=("Courier", 12))
    lost_connection.pack()
    break

def paint(self, e):
    """
    draws the lines and sends the coordinates and width of the drawn lines to the server
    :param e:
    """

    # Drawing the lines
    if self.old_x and self.old_y:
        self.c.create_line(self.old_x, self.old_y, e.x, e.y, width=self.pen_width, fill=self.color_fg,
                           capstyle=ROUND, smooth=True)
    self.old_x = e.x
    self.old_y = e.y

    # Sending the painted Coordinates(x y) + pen width (5-100) + whether the mouse is reset(1) or not(0)

    if -99 < e.x < 999 and -99 < e.y < 999: # The coordinates must be no longer than triple digit numbers or
        # the server will kick the painter and have an error (because of the header size set for coordinates)

        pen_list_width = str(self.pen_width).split('.')
        pen_send_width = pen_list_width[0] # Doesnt take all of the numbers (only the ones before the dot(5-
100))
        line_info = ('Coordinates: ' + str(e.x) + ' ' + str(e.y) + ' ' + pen_send_width + ' ' +
                     self.resetting_mouse)
        # Coordinates = x coordinate y coordinate pen width (5-100) and whether the mouse is reset(1) or
        not(0)
        print(line_info)
```



# תיכון אורט אביר

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



```
s.send(str(len(line_info)).encode()) # Header sending the length of the incoming message
s.send(line_info.encode()) # Sends the server the line_info
self.resetting_mouse = '0'

else: # if the coordinates went past their limit - reset the mouse
    self.resetting_mouse = '1'

def reset(self, w): # Resetting x and y when you stop clicking
    """
    when the painter stops clicking the mouse (painting) resets the old_x and old_y coordinates and changes
    the resetting_mouse parameter to 1 (meaning the mouse is reset)
    :param w:
    """
    self.old_x = None
    self.old_y = None
    print(w)
    print('resetting')
    self.resetting_mouse = '1'

def change_width(self, e): # changes the pen width
    """
    change Width of the pen through slider
    :param e:
    """
    self.pen_width = e

def clear(self):
    """
    deletes all lines drawn on the canvas and tels the server it did so
    """
    self.c.delete(ALL)
    print('delete')
    s.send('-1'.encode()) # CODE FOR DELETING - -1

def change_fg(self): # changing the pen color
    """
    changes the color of the pen and sends the new pen color to the server
    """
    self.color_fg = colorchooser.askcolor(color=self.color_fg)[1]

    # sending the pen color
    pen_color = ('color: ' + str(self.color_fg))
    print(pen_color)
    s.send(str(len(pen_color)).encode()) # header , length of pen_color , must be '14'
    s.send(pen_color.encode()) # sends the new pen color

def change_bg(self): # changing the background color canvas
```



# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



```
"""
changes the background color or the canvas and sends the new background color to the server
"""

self.color_bg = colorchooser.askcolor(color=self.color_bg)[1]
self.c['bg'] = self.color_bg

# sends the new background color to the server
back_color = ('background_color_is: ' + str(self.color_bg))
print(back_color)
s.send(str(len(back_color)).encode()) # sends length , header of msg , its '28'
s.send(back_color.encode()) # sends the new background color to the server

def draw_widgets(self):
    """
    responsible for drawing the painter's widgets and menus and linking them to the right commands
    """

    self.controls = Frame(self.master, padx=5, pady=1)
    self.slider = ttk.Scale(self.controls, from_=5, to=100, command=self.change_width, orient=VERTICAL)
    self.slider.set(self.pen_width)
    self.slider.grid(row=0, column=1, ipadx=30)
    self.controls.pack(side=LEFT)

    self.master.config(menu=self.menu)
    self.menu.add_cascade(label='Colors', menu=self.color_menu)
    self.color_menu.add_command(label='Brush Color', command=self.change_fg)
    self.color_menu.add_command(label='Background Color', command=self.change_bg)
    option_menu = Menu(self.menu)
    self.menu.add_cascade(label='Options', menu=option_menu)
    option_menu.add_command(label='Clear Canvas', command=self.clear)
    option_menu.add_command(label='Exit', command=self.master.destroy)

# Watcher Code

class Watcher: # class that only allows to guess and watch the drawing
    """
    Takes care of the watcher client , sends info , receives info and presents it to the player.
    """

    global ip # ip of the server
    global port # port number to connect to

    def __init__(self, master, w): # master is the root , the window
        """
        defining class variables and building the screen widgets of the watcher
        :param master:
        """
```



# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



```
:param w:
"""

self.w = w
self.master = master
self.old_x = '0'
self.old_y = '0'
self.Background_color = '#ffffff' # current background color
self.Pen_color = '#000000' # current Pen_color
self.Painted_Coordinates = '000 000 5 1' # x y pen_width and mouse state - 1 for reset 0 for not
self.Guess_sent = True # if its true no guess has been sent yet, otherwise the user has started guessing
self.Winner = '' # name of the winner
self.game_time = '' # the length of the game
self.action_list = [] # paints the game if you joined late
self.still_redrawing = False # False when the action list is done being read and True when its still being
read
self.game_over = False # Turns True when the game is over and some has won

self.guess = Label(self.master, text='Guess', bg='khaki4', font=("Courier", 28))
self.guess.pack()

self.enter_guess = Label(self.master, text="Enter your guess here:", bg='khaki4', font=("Courier", 12))
self.enter_guess.pack()

self.guess_word = Entry(self.master, bd=1)
self.guess_word.pack()
self.guess_word.bind('<KeyPress>', self.enter_send)

self.enter_button = Button(self.master, text=">>>>", command=self.add_text, bg='black', fg='white')
self.enter_button.pack()

self.sent_guess = Label(self.master, text="Guess sent", bg='khaki4', fg='white', font=("Courier", 10))

self.wrong_guess = Label(self.master, text="Guess sent", bg='khaki4', fg='white', font=("Courier", 10))

self.painter_logged = Label(self.master, text="", bg='khaki4', font=("Courier", 14))
self.painter_logged.pack()

thread_watcher = threading.Thread(target=self.watcher_info,)
thread_watcher.start()

def watcher_info(self):
    """
    takes care of data sent to the watcher by the server
    """
    while True:
        # noinspection PyBroadException
        try:
            data_watcher = s.recv(40) # receives info
            data_watcher = data_watcher.decode("utf-8")
```



# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



```
print(data_watcher)

if data_watcher.startswith('Game information incoming'):
    pen_color_info = s.recv(7)
    pen_color_info = pen_color_info.decode()
    print(pen_color_info)
    self.Pen_color = pen_color_info

    self.still_redrawing = True # now entering the redrawing phase
    game_info = s.recv(1000000).decode() # receives info
    while 'stop;' not in game_info: # takes the information of the entire game so far , can be large
        game_info += s.recv(1000000).decode() # receives info
    print(game_info)
    self.action_list = game_info.split(';')
    thread_drawn_so_far = threading.Thread(target=self.drawn_so_far)
    thread_drawn_so_far.start()

if data_watcher.startswith('wrong'):
    self.wrong_guess.destroy()
    self.wrong_guess = Label(self.master, text="Wrong guess try again", bg='khaki4', fg='white',
                             font=("Courier", 10))
    self.wrong_guess.pack()

if data_watcher.startswith('/close'): # once u close the tab the server tells u to log out as well ,
    # UNLESS the game is already over and then tab closes on its own and u go to leader boards
    print('window closed')
    break

if data_watcher.startswith('painter logged on'): # the painter is online
    self.painter_logged.destroy()
    self.painter_logged = Label(self.master, text='Painter Online', bg='khaki4', fg='green',
                                font=("Courier", 14))
    self.painter_logged.pack()

if data_watcher.startswith('painter logged off'): # the painter is offline
    self.painter_logged.destroy()
    self.painter_logged = Label(self.master, text='Painter Offline', bg='khaki4', fg='red',
                                font=("Courier", 14))
    self.painter_logged.pack()

if data_watcher == 'game over': # game is over someone won
    print(data_watcher)
    time.sleep(0.01)
    data_winner_name = s.recv(4096) # receives info
    data_winner_name = data_winner_name.decode("utf-8")
    self.Winner = data_winner_name # name of the winner is saved
    print(self.Winner)
    data_game_time = s.recv(4096).decode()
    self.game_time = str(data_game_time)
```



# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



```
self.game_over = True # Game is over
self.end_screen() # goes into the leader boards
break

if data_watcher == 'Game full': # the game is full
    print('| The game is full try again later |')
    break

if data_watcher.startswith('Pen:'): # pen color
    print(data_watcher[4:]) # takes only the color
    self.Pen_color = data_watcher[4:]
    if self.still_redrawing:
        self.action_list.append('Pen:' + self.Pen_color)
    else:
        print('changed pen color')

if data_watcher.startswith('Background:'): # background color
    print(data_watcher[11:]) # self.change_bg(data[11:]) # takes only the color
    self.Background_color = data_watcher[11:]
    if self.still_redrawing:
        self.action_list.append('Background:' + self.Background_color)
    else:
        self.w['bg'] = self.Background_color
        print('background changed to' + self.Background_color)

if data_watcher.startswith('Delete'): # Delete or not
    print('delete') # self.clear() # YOU Need to clear right away and then change it back to false
    print('deleting stuff')
    if self.still_redrawing:
        self.action_list.append('delete')
    else:
        self.w.delete(ALL)

if data_watcher.startswith('reset_screen'): # a new painter joined , resetting the watcher's screen
    if self.still_redrawing:
        self.action_list.append('reset_screen')
    else:
        print('resetting screen')
        self.Background_color = '#ffffff'
        self.w['bg'] = self.Background_color
        self.w.delete(ALL)
        self.Pen_color = '#000000'

if data_watcher.startswith('Coordinates:') and len(data_watcher) < 27: # Len shorter than 27 in case
    it
    # sends extra info
    print(data_watcher[12:]) # takes only the coordinates + pen width
    self.Painted_Coordinates = data_watcher[12:]
    if self.still_redrawing:
```





# תיכון אורט אביר

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



```
self.action_list.append('Coordinates:' + self.Painted_Coordinates)
else:
    painted_list = self.Painted_Coordinates.split(' ')
    x_ = painted_list[0]
    y_ = painted_list[1]
    pen_width = painted_list[2] # pen_width is the pen width
    reset = painted_list[3]
    if reset == '0':
        self.w.create_line(self.old_x, self.old_y, x_, y_, fill=self.Pen_color, width=pen_width,
                           capstyle=ROUND, smooth=True)
    if reset == '1':
        self.w.create_line(x_, y_, str(int(x_) - 0.00001), str(int(y_) - 0.00001),
                           fill=self.Pen_color, width=pen_width, capstyle=ROUND, smooth=True)

        self.old_x = x_
        self.old_y = y_
except Exception:
    if not self.game_over:
        print('connection lost...')
        lost_connection = Label(self.master, text="Connection lost...", bg='khaki4', fg='red', font=(
            "Courier", 12))
        lost_connection.pack()
    else:
        print('exception invalid , game already over')
        break

def end_screen(self):
    """
    builds the watcher's leader boards screen
    +
    tells the server that the watcher has gone into the leader boards successfully and that it can now reset
    """
    self.w.delete(ALL)
    self.sent_guess.destroy()
    self.enter_guess.destroy()
    self.wrong_guess.destroy()
    self.guess.destroy()
    self.guess_word.destroy()
    self.enter_button.destroy()
    self.painter_logged.destroy()
    self.master.geometry("500x500")

    self.w.configure(bg='Grey')
    self.w.create_line(200, 120, 200, 220, fill='yellow', width=5,
                       capstyle=ROUND, smooth=True)
    self.w.create_line(200, 120, 300, 120, fill='yellow', width=5,
                       capstyle=ROUND, smooth=True)
    self.w.create_line(300, 120, 300, 220, fill='yellow', width=5,
                       capstyle=ROUND, smooth=True)
    label0 = Label(self.master, text="1", bg='Grey', fg='Yellow', font=("Courier", 32))
```



# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



```
label0.pack()
label0.place(x=235, y=150)

label1 = Label(self.master, text="Game Over!", bg='Khaki4', fg='Black', font=("Courier", 35))
label1.pack()
label2 = Label(self.master, text="The winner is:", bg='Grey', fg='Black', font=("Courier", 16))
label2.pack()
label2.place(x=163, y=50)
label3 = Message(self.master, text=self.Winner, bg='Grey', fg='Black', font=("Courier", 14), width=150)
label3.pack()
label3.place(x=200, y=85)
label4 = Label(self.master, text="Time of guess: " + self.game_time, font=("Courier", 12), bg='Grey',
               fg='White')
label4.pack()
label4.place(x=180, y=280)

s.send('end'.encode()) # tells the server it went smoothly into the leader boards
data_database_updated = s.recv(1024).decode() # waiting to hear from the server , when the client hears
# from the server it knows that the data base has finished updating and the server can be told to reset
now
print(data_database_updated)
s.close() # game over disconnecting

s_reset = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # Af_inet - ipv4 , sock_stream - tcp
s_reset.connect((ip, port)) # new socket connected to tell the server to reset
s_reset.send('reset server'.encode())
s_reset.close()

def add_text(self):
    """
    tells the watcher the guess was sent
    calls the function that sends the guess to the server
    """
    self.wrong_guess.destroy()
    self.sent_guess.destroy()
    self.sent_guess = Label(self.master, text="Sending guess...", bg='khaki4', fg='white', font=("Courier", 10))
    self.sent_guess.pack()
    self.retrieve_input()

def retrieve_input(self):
    """
    sends the watcher's guess to the server
    """
    print(self.guess_word.get())
    if ';' not in self.guess_word.get():
        # noinspection PyBroadException
        try:
            s.send(('guessed: ' + str(self.guess_word.get())).encode())
            self.sent_guess.destroy()
```



# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



```
self.sent_guess = Label(self.master, text="Guess sent", bg='khaki4', fg='white',
                        font=("Courier", 10))
self.sent_guess.pack()
except Exception:
    print('guess did not make it to the server')
    self.sent_guess.destroy()
    self.sent_guess = Label(self.master, text="try again, guess not sent", bg='khaki4', fg='white', font=(
        "Courier", 10))
    self.sent_guess.pack()
else:
    self.sent_guess.destroy()
    self.sent_guess = Label(self.master, text="guess must not contain ;", bg='khaki4', fg='white', font=(
        "Courier", 10))
    self.sent_guess.pack()

def enter_send(self, event):
    """
    sends guess when you press enter
    :param event:
    """
    if event.keycode == 13:
        self.add_text()

def drawn_so_far(self):
    """
    Draws the game up until the watcher catches on and gets the game
    broadcast to him live
    """
    first_line = True # is True for the first line drawn
    old_x_drawn = '0'
    old_y_drawn = '0'
    pen_color = '#000000'
    for act in self.action_list:
        if act.startswith('Coordinates:'): # paints the coordinates
            draw = act[12:]
            draw_list = draw.split(' ')
            x = draw_list[0]
            y = draw_list[1]
            pen_width = draw_list[2] # pen_width is the pen width
            reset = draw_list[3]
            if reset == '0' and not first_line: # meaning the mouse has not reset
                self.w.create_line(old_x_drawn, old_y_drawn,
                                   x, y, fill=pen_color, width=pen_width, capstyle=ROUND, smooth=True)
            if reset == '1' or first_line: # meaning the mouse has reset
                self.w.create_line(x, y, str(int(x) - 0.00001), str(int(y) - 0.00001),
                                   fill=pen_color, width=pen_width, capstyle=ROUND, smooth=True)
            first_line = False # we are past the first line drawn
            old_x_drawn = x
            old_y_drawn = y
```



# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



```
if act.startswith('Pen:'): # changes the pen color
    pen_color = act[4:]
    print('pen color changed to' + pen_color)

if act.startswith('delete'): # deletes coordinates painted thus far
    print('delete')
    self.w.delete(ALL)

if act.startswith('reset_screen'):
    print('resetting screen')
    self.Background_color = '#ffffff'
    self.w['bg'] = self.Background_color
    self.w.delete(ALL)
    pen_color = '#000000'

if act.startswith('Background:'): # changes background color
    self.w['bg'] = act[11:]
    print('background changed to' + act[11:])

if self.game_over: # the game is over stop drawing
    break

self.Pen_color = pen_color
self.old_x = old_x_drawn
self.old_y = old_y_drawn
self.still_redrawing = False

'''
Main code
'''
if __name__ == '__main__':
    '''
    Globals
    '''
    global Role # the role chosen by the user
    global User # the user name
    ip = '127.0.0.1' # ip of the server
    port = 5001 # port number to connect to

    '''
    Functions
    '''
    def on_closing_watcher(socket_: socket.socket): # Informs the server a watcher has left
        '''
        takes care of what happens when the watcher closes the window and randomly disconnects
        :param socket_:
        '''
```



# תיכון אורט אביר

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



```
# noinspection PyBroadException
try:
    socket_send('/close'.encode())
    print('window closed')
except Exception:
    print('game already over , server reset/server crashed')

def painter_chosen(): # takes care of what happens when you are the painter
    """
    takes care of what happens when you choose to be the painter
    first enters the painter choose class in which the painter chooses what to draw
    and then enters the drawing board class of the painter
    after the game is over or if the painter disconnects closes the socket and informs the server that the painter
    left
    """
    # WORD CHOOSING ROOT
    rootPainterWord = Tk()
    rootPainterWord.title("Painter")
    rootPainterWord.wm_iconbitmap('scribble_logo.ico')
    rootPainterWord.geometry("450x150")
    rootPainterWord.resizable(width=False, height=False)
    rootPainterWord.configure(background='Grey')
    PainterChoose(rootPainterWord) # the window in which you choose what to draw
    rootPainterWord.mainloop()
    # resetting the watcher's screen whenever a painter is chosen
    # noinspection PyBroadException
    try:
        s.send('-9'.encode()) # -9 is the code for resetting the settings of the watcher or if its sent instead of
        # a word it lets the server know the painter logged out
    except Exception:
        print('connection lost...')

    # whenever a new painter logs - DRAWING ROOT

    rootPainter = Tk()
    Painter(rootPainter) # the root in which you draw
    rootPainter.title('Painter')
    rootPainter.geometry("500x580")
    rootPainter.resizable(width=False, height=False)
    rootPainter.wm_iconbitmap('scribble_logo.ico')

    rootPainter.mainloop()
    # noinspection PyBroadException
    try:
        s.send('-3'.encode()) # -3 code for logging off tells the server the painter has logged out
    except Exception:
        print('game already ended so no need to inform the server that the painter has logged out')
    s.close()
```



# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



```
def watcher_chosen(): # takes care of what happens if you're a watcher
    """
    builds the watcher tk window and canvas and then runs the watcher class
    when the watcher disconnects class the function that tells the server the watcher has disconnected
    """

    # WATCHING ROOT
    root_watch = Tk()
    root_watch.title('Watcher')
    root_watch.wm_iconbitmap('watcher_logo.ico')
    root_watch.configure(bg='khaki4')
    root_watch.resizable(width=False, height=False)
    w = Canvas(root_watch, width=500, height=420, bg="white", )
    w.pack(fill=BOTH, expand=False)
    Watcher(root_watch, w)
    root_watch.mainloop()
    on_closing_watcher(s)

    """
    Main
    """

    restartPainter = False # if there is already a painter turns True
    restartWatcher = False # if there are already 5 watchers turns True
    restartUser = False # if user name is invalid turns True
    restartName = False # if the name is already taken turns True
    serverClosed = False # if the server is not open turns True

    while True:
        root = Tk() # builds the base root , the logging window
        root.title("Scribble")
        root.wm_iconbitmap('scribble_logo.ico')
        root.geometry("500x500")
        root.resizable(width=False, height=False)
        root.configure(background='Black')
        C = Canvas(root, bg="black", height=250, width=300)

        pic = Image.open("login_screen.png")
        img = pic.resize((500, 500), Image.ANTIALIAS)
        img.save("login_screen.png")
        pic = Image.open("login_screen.png")
        tk_pic = ImageTk.PhotoImage(pic)

        background_label = Label(root, image=tk_pic)
        background_label.place(x=0, y=0, relwidth=1, relheight=1)
        C.pack()

        UserLogin(root) # calls the class taking care of the login window

    if serverClosed is True: # if the server is closed
```



# תיכון אורט אביר

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



```
closed_server = Message(root, text="The server is closed at the moment , please try another time...",
font=(
                                "Courier", 7), width=200)

closed_server.pack()
closed_server.place(x=150, y=430)
server_closed = False
Role = None # resets the role
User = None # resets the nickname

if restartPainter is True: # if there is already a painter
    chosenPainter = Message(root, text="there is already a painter please choose a different role", font=(
                                "Courier", 7), width=200)

    chosenPainter.pack()
    chosenPainter.place(x=150, y=430)
    restartPainter = False
    Role = None # resets the role
    User = None # resets the nickname

if restartWatcher is True: # if there are already 5 watchers
    chosenWatcher = Message(root, text="the max amount of watchers was reached please choose a
different role",
                                font=("Courier", 7), width=200)
    chosenWatcher.pack()
    chosenWatcher.place(x=150, y=430)
    restartWatcher = False
    Role = None # resets the role
    User = None # resets the nickname

if restartName is True: # if the watcher's user name is already taken
    takenName = Message(root, text="This user name is already taken please choose a different one",
                                font=("Courier", 7), width=200)
    takenName.pack()
    takenName.place(x=150, y=430)
    restartName = False
    Role = None # resets the role
    User = None # resets the nickname

if restartUser is True: # if the username entered is invalid
    invalidName = Message(root,
                                text="user name is invalid (contains ';'/' ', blank or is longer than 10 characters"
                                "), please enter a different user name", font=("Courier", 7), width=200)
    invalidName.pack()
    invalidName.place(x=150, y=430)
    restartUser = False
    Role = None # resets the role
    User = None # resets the nickname

root.mainloop()
# noinspection PyBroadException
```



# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



```
try:
    if Role is None: # role must be either watcher or painter
        break
except Exception:
    print('role is not defined yet , window closed')
    break

else:
    client_role = Role # either painter or watcher
    user_name = User # name of the user
    user_name_checker = False # checks the validity of the user name entered

    if ' ' in user_name or ';' in user_name: # checks if the user name contains spaces ( ' ') or ';'
        user_name_checker = True

    if not type(user_name) is str: # making sure there isn't a syntax error when checking length of
user_name
        user_name = str(user_name)
        print(user_name)

    if not type(client_role) is str: # making sure there isn't a syntax error when adding to client_role
        client_role = str(client_role)
        print(client_role)

    if len(user_name) > 10 or len(user_name) == 0 or user_name_checker:
        # if the user name is too long or is 0 letters or contains ';'/' ' restarts
        restart_user = True
        continue

    else: # user name is valid
        user_info = client_role + ';' + user_name # separated by a ';'

        # sockets - establishing connection
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # Af_inet - ipv4 ,sock_stream - tcp
        # noinspection PyBroadException
        try:
            s.connect((ip, port))
            s.send(user_info.encode()) # sends the user info to the server (user name and desired role)
        except Exception: # server is closed
            server_closed = True
            continue

    if client_role == 'painter': # what happens when you choose painter as your role
        # noinspection PyBroadException
        try:
            data = s.recv(100)
        except Exception: # if the server is closed/crashed
            print('server closed/crashed')
            server_closed = True
```





# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



```
continue

data = data.decode()
if data == 'you are now the painter': # meaning there is no other painter logged to the server atm
    painter_chosen() # you are now the painter , runs the painter function
    break
else: # there is already another painter logged to the server , restarts
    print('there is already a painter, restart and choose a different role')
    restartPainter = True
    continue

else: # what happens when you choose watcher as your role
    # noinspection PyBroadException
    try:
        data = s.recv(10) # either game full , name taken or you joined
    except Exception: # if the server is closed/crashed
        print('server closed/crashed')
        server_closed = True
        continue

data = data.decode()
print(data)

if data == 'Game full': # meaning there are 5 watchers logged in , restarts.
    restartWatcher = True
    continue
if data == 'Name Taken': # the user name you chose is already taken by another watcher , restarts.
    restartName = True
    continue
else: # meaning there are less than the max number of watchers logged in and the name is not
    taken
    watcher_chosen() # you are now a watcher , runs the watcher function
    break
```



# תיכון אורט אביר

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



[Scribble\\_server.py](#)

```
"""
Made by Amir Wolberg
Classes used: socket , select , functools, time , sqlite3, thread and tkinter
"""

import socket
import threading
import select
import time
import sqlite3
from tkinter import *
from functools import partial

"""
No bugs known - need to QA
add option to choose how many watchers can play
"""

Classes
"""

class PaintServer:
    """
    class that takes care of receiving and sending information from the painter to the watchers
    after the manager
    starts a new game
    """

    def __init__(self):
        """
        defines all of the class variables and starting the incoming_connections function
        """

        self.Background_color = '#ffffff' # current background color
        self.Pen_color = '#000000' # current Pen_color
        self.Painted_Coordinates = '000 000 5 1' # x , y , pen width and mouse state(reset(1) or not
        reset(0))
        self.Previous_Coordinates = '000 000 5 1' # the coordinates previously painted
        self.game_time = '0' # the time from the moment the painter entered a word until someone
        won
```



# תיכון אורט אביר

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



```
self.watchers_logged = 0 # watchers currently connected to the server , max is 5
self.watchers_were_logged = 0 # number of watchers who were present in the game at any
point
self.watchers_enter_database = 0 # tells us how many watchers had their guesses put in
self.guess_list

self.watcher_list = [] # list containing the sockets of all of the logged watchers
self.watcher_name_list = [] # list containing the names of all of the logged watchers
self.guess_list = [] # keeps all of the guesses of different clients , has lists in it , in those lists
each
# belonging to a different watcher are the guesses

self.painter_chosen = False # True if a painter is online , False if there is no painter
self.game_over = False # when a client guesses the right word this changes to True and the
game is over

self.database_information = "" # keeps all of the details of the painting for the database
self.Winner = "" # will contain the name of the winner

self.start_of_game = None # contains the time of the game's start , when the latest painter
chose a word
self.guess_word = None # contains the word that needs to be guessed , the name of the
painting
self.painter_socket = None # contains the current painter's socket

self.incoming_connection()

def incoming_connection(self): # waits for incoming connections and adds them to the list
    """
    handles each connection from a client to the server using sockets
    starts the threads of either the watcher or painter depending on the information sent to it by
    the client
    """

    while not self.game_over:
        print('still in incoming_connection')

        read_list = select.select([s], [], [])[0] # waits for a socket connection

        if self.game_over:
            break

        if read_list: # if a socket connection was received , takes care of the specific socket
```



# תיכון אורט אביר

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



```
# noinspection PyBroadException
try:
    client_socket, address = s.accept()
    data = client_socket.recv(4096).decode('utf-8')
    print(data) # either name and role of the client or a confirmation to reset the server

    if data.startswith('reset server'): # not a client but a msg confirming the game is
over        continue

    data_list = data.split(';')
    role = data_list[0]
    name = data_list[1]
    print('user role is:' + role)
    print('user name is:' + name)

    # If the role is painter
    if role == 'painter' and not self.painter_chosen: # only 1 painter can be online at a
time        print('the painter is chosen')

    # noinspection PyBroadException
    try:
        client_socket.send('you are now the painter'.encode())
        self.painter_chosen = True

        thread_transmit = threading.Thread(target=self.painter_client,
args=(client_socket,))
        thread_transmit.start()
        # thread that takes information from the painter and sends it to all of the
watchers

    except Exception: # when the painter disconnects can cause an error and jump to
the exception
        print('jumped error!')
        self.painter_chosen = False # the painter has disconnected
        continue

    elif role == 'painter' and self.painter_chosen: # a painter is already online
        print('reached the limit of painters')
        # noinspection PyBroadException
        try:
            client_socket.send('painter already chosen'.encode())
```



# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



```
except Exception:
    print('client disconnected before msg was sent')
    continue

# If the role is watcher ( takes the role as watcher automatically if its anything except
painter)
else:
    print(name)
    print(self.watcher_name_list)

    if name in self.watcher_name_list: # can't have multiple watchers with the same
name
        # noinspection PyBroadException
        try:
            client_socket.send('Name Taken'.encode())
        except Exception:
            print('watcher logged off before name taken was sent')

    else: # name is valid and the role is watcher
        global watcher_number # the chosen max number of watchers allowed to play
        if self.watchers_logged < int(watcher_number):
            # up to x watchers are allowed to be online at the same time

            # noinspection PyBroadException
            try:
                client_socket.send('You joined'.encode())

                if self.painter_chosen:
                    client_socket.send('painter logged on'.encode())
                    print('painter is on')
                    # tells the watchers a painter is logged

                if not self.painter_chosen:
                    client_socket.send('painter logged off'.encode())
                    print('painter is off')
                    # tells the watchers there is no painter
            except Exception:
                print('watcher crashed/logged off before starting its thread')
                continue

        self.watchers_logged += 1
        self.watchers_were_logged += 1
        self.watcher_name_list.append(name)
```



# תיכון אורט אביר

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



```
# noinspection PyBroadException
try:
    self.painter_socket.send('+watcher'.encode())
    # tells painter another watcher logged
except Exception:
    print('no painter logged yet')

print('number of watchers logged: ' + str(self.watchers_logged))
self.watcher_list.append(client_socket)

thread_watcher = threading.Thread(target=self.watcher_client_receive, args=(
    client_socket, name))
# thread handling the information sent by the watcher to the server
thread_watcher.start()

else: # max watchers already online
    print('reached the limit of watchers')
    # noinspection PyBroadException
    try:
        client_socket.send('Game full'.encode())
    except Exception:
        print('client logged off so Game full was not sent')
    continue
except Exception:
    print('an error occurred in incoming_connections , moving on to the next client')
    continue

def painter_client(self, painter_socket: socket.socket):
    """
    Handles receiving information from the painter client and broadcasting it to the watchers
    :param painter_socket:
    """
    for watcher in self.watcher_list:
        watcher.send('painter logged on'.encode()) # tells the watchers a painter has logged

    # noinspection PyBroadException
    try:
        data_word = painter_socket.recv(21) # word must not be longer than 20 characters
        data_word = data_word.decode()
        if data_word == '-9':
            # If The word entered was blank it sends '-9' and the painter is kicked or if painter
            logged off
```



# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



```
print('connection aborted , word entered was blank , painter kicked')
self.painter_chosen = False
self.painter_socket = None
for watcher in self.watcher_list:
    watcher.send('painter logged off'.encode()) # tells the watchers a painter has logged
off
else:
    self.guess_word = str(data_word) # what the painter is drawing
    print("the word to guess is: " + self.guess_word)

except Exception:
    self.painter_socket = None
    self.painter_chosen = False
    for watcher in self.watcher_list:
        watcher.send('painter logged off'.encode()) # tells the watchers a painter has logged
off
    print('jumped error! (1) painter crashed/logged off')

if self.painter_chosen: # if the painter has not been kicked out continue

    self.start_of_game = time.time() # when the painter logs the game has officially started
    print('game has started time is : ' + str(self.start_of_game))

    for watcher in self.watcher_list:
        watcher.send('painter logged on'.encode()) # tells the watchers a painter has logged
    self.painter_socket = painter_socket # painter has gone through the validation process,
its socket is kept

    print('num : ' + str(self.watchers_logged))
    # noinspection PyBroadException
    try:
        self.painter_socket.send(('num : ' + str(self.watchers_logged)).encode())
        # tells the painter how many watchers are logged
    except Exception:
        self.painter_socket = None
        self.painter_chosen = False
        for watcher in self.watcher_list:
            watcher.send('painter logged off'.encode()) # tells the watchers a painter has logged
off
        print('jumped error! (2) painter crashed/logged off')

if self.painter_chosen: # if the painter has not been kicked out start its loop
    while not self.game_over:
```



# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



```
print('still in painter_client')
# noinspection PyBroadException
try:
    """
    maximum length of the 2 digit number representing the msgs length
    each number is a header for a certain type of information:
    -3 for painter logging out, -9 for resetting watcher screen, -1 for deleting,
    15-26 for painted coordinates, 14 for pen color , 28 for background color
    """

    data = painter_socket.recv(2) # maximum length of the 2 digit number - header
    data = data.decode() # also checks if at any given moment the painter crashed

except Exception: # error occurred, the painter has timed out (disconnected)
    print('jumped error! (3) painter crashed/logged off')
    self.painter_chosen = False # the painter has disconnected
    self.painter_socket = None
    for watcher in self.watcher_list:
        watcher.send('painter logged off'.encode()) # tells the watchers the painter has
logged off
    break

# noinspection PyBroadException
try:
    if self.game_over:
        break

    if data == '-3': # meaning the current painter has closed the tab and logged out
        print('painter logged out')
        for watcher in self.watcher_list:
            watcher.send('painter logged off'.encode()) # tells the watchers painter has
logged off
        self.painter_chosen = False
        self.painter_socket = None
        break

    if data == '-9': # -9 is code for resetting the watchers screen happens when a new
painter logs
        print('resetting screen')
        self.database_information = '' # empties database painting information
        for client in self.watcher_list:
            client.send('reset_screen'.encode()) # reset_screen code for resetting
watchers screen
```





# תיכון אורט אביר

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



```
if data == '-1': # -1 is the code for deleting (clearing the screen)
    print('delete')
    self.database_information = 'Pen:' + self.Pen_color + ';' + 'Background:' + \
        self.Background_color + ';' # removes the painted coordinates
    for client in self.watcher_list:
        client.send('Delete'.encode())

if 27 > int(data) > 14: # meaning it has to be a coordinate (+ width of pen + mouse
state)
    # Coordinates: xxx yyy www m - the template in which its sent
    data = painter_socket.recv(int(data))
    data = data.decode()
    print(data[13:]) # return coordinates x,y and pen width + mouse state
    self.Painted_Coordinates = data[13:]
    self.database_information += 'Coordinates:' + self.Painted_Coordinates + ';'
    if not (self.Previous_Coordinates == self.Painted_Coordinates): # so it doesn't
resend
        for client in self.watcher_list:
            client.send(('Coordinates:' + self.Painted_Coordinates).encode())
        self.Previous_Coordinates = self.Painted_Coordinates

if data == '14': # 'color:'
    data = painter_socket.recv(14) # size of 14
    data = data.decode()
    print(data[7:]) # color changed to
    self.Pen_color = data[7:]
    self.database_information += 'Pen:' + self.Pen_color + ';'
    pen_send = ('Pen:' + self.Pen_color)
    for client in self.watcher_list:
        client.send(pen_send.encode())

if data == '28': # background_color:
    data = painter_socket.recv(28) # size of 28
    data = data.decode()
    print(data[21:]) # background changed to
    self.Background_color = data[21:]
    self.database_information += 'Background:' + self.Background_color + ';'
    background_send = ('Background:' + self.Background_color)
    for client in self.watcher_list:
        client.send(background_send.encode())

except Exception: # happens when the painter disconnects or times out
```



# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



```
print('jumped error!(4) painter crashed/logged off')
for watcher in self.watcher_list:
    watcher.send('painter logged off'.encode()) # tells the watchers the painter has
logged off
self.painter_chosen = False # the painter has disconnected
self.painter_socket = None
break

def watcher_client_receive(self, watcher_socket: socket.socket, watcher_name):
    """
    Handles information received from the watcher (mainly guesses)
    Handles information sent to the watcher by the server (not by the painter)
    also keeps watcher guesses to add to the database
    :param watcher_socket:
    :param watcher_name:
    """

    guess_word_list = [] # list of all of the watcher's guesses

    time.sleep(0.2) # cool down needed after the last packet sent to the watcher (if painter is
on or not)
    if len(self.database_information) != 0: # meaning the game has started before the watcher
joined
        # noinspection PyBroadException
        try:
            watcher_socket.send('Game information incoming'.encode()) # warning watcher
client to prepare for info

            watcher_socket.send(self.Pen_color.encode()) # sends the currently pen color to the
watcher

            self.database_information += 'stop;' # tells watcher when to stop receiving
information - header
            watcher_socket.send(self.database_information.encode())
        except Exception:
            print("watcher connection lost (1)")

    while not self.game_over:
        print('still in watcher_client_receive')
        # noinspection PyBroadException
        try:
            data = watcher_socket.recv(1024) # also checks at any given moment if the watcher
did not crash
            data = data.decode()
```



# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



```
if data.startswith('end'): # meaning the game has ended and the watcher entered the
leader boards
    break

if data.startswith('/close'): # meaning the watcher closed the tab
    self.watchers_logged -= 1 # a watcher has disconnected
    self.watcher_name_list.remove(watcher_name)
    self.watcher_list.remove(watcher_socket)
    watcher_socket.send('/close'.encode()) # the connection has been shut down
successfully
    # noinspection PyBroadException
    try:
        self.painter_socket.send('-watcher'.encode()) # informs the painter a watcher has
logged off
    except Exception:
        print('error occurred - the painter has not logged yet so we can not send it
information')
        break

if data.startswith('guessed:'): # a guess sent by the watcher
    word_guessed = str(data[8:])

    # noinspection PyBroadException
    try:
        if word_guessed == self.guess_word: # if the watcher guessed correctly
            current_time = time.time()
            time_of_guess = current_time - self.start_of_game
            time_format = time.strftime("%M:%S", time.gmtime(time_of_guess)) # the time
of guess
            print(time_format)

            guess_data = watcher_name + ' => correct guess : ' + str(data[8:]) + ', ' + \
                str(time_format)
            guess_word_list.append(guess_data) # enters guessed word

            print('the client has guessed the word')
            self.game_over = True # the client has guessed the correct word and the game is
over

            self.Winner = watcher_name
            self.game_time = str(time_format)
            print(self.Winner)
            thread_over = threading.Thread(target=self.game_ended)
```



# תיכון אורט אביר

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



```
thread_over.start()
break

else:
    current_time = time.time()
    time_of_guess = current_time - self.start_of_game
    time_format = time.strftime("%M:%S", time.gmtime(time_of_guess)) # the time
of guess
    print(time_format)

    guess_data = watcher_name + ' => wrong guess : ' + str(data[8:]) + ', ' +
str(time_format)
    guess_word_list.append(guess_data) # enters guessed word

    watcher_socket.send('wrong'.encode())
    print('guess again')

except Exception:
    print('error occurred - no painter has logged yet and the game did not start yet')
except Exception:
    print('watcher connection lost (2)')
    self.watchers_logged -= 1 # a watcher has disconnected or crashed
    self.watcher_name_list.remove(watcher_name)
    self.watcher_list.remove(watcher_socket)
    # noinspection PyBroadException
    try:
        self.painter_socket.send('-watcher'.encode()) # informs the painter a watcher has
logged off
    except Exception:
        print('error occurred - the painter has logged yet so we can not send it information')
        break

    print('appending guesses of ' + watcher_name)
    self.guess_list.append(guess_word_list)
    self.watchers_enter_database += 1 # another watcher's guesses were put into
self.guess_list
    print(self.watchers_enter_database)

def game_ended(self):
    """
    takes care of what happens when a watcher wins and the game
    is over
    """
```



# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



```
time.sleep(0.1)

print('Game has ended going into leader boards')
for client in self.watcher_list: # Tells all watchers the game is over
    # noinspection PyBroadException
    try:
        self.game_over_socket(client)
    except Exception:
        print('a watcher has crashed - game over information to it not sent')

if self.painter_chosen:
    # noinspection PyBroadException
    try:
        self.game_over_socket(self.painter_socket) # if a painter is logged tells him the game is
over
    except Exception:
        print('painter logged off/crashed , did not send winning screen')

print(self.database_information) # paintings details at the end of the game that go into the
database

'''
    this part waits 5 seconds for all of the watchers to confirm they have entered their guesses
into the
    self.guess_list variable , if 1 or more haven't done it in under 5 seconds the server does not
keep
    their guesses in the database
'''
time_counter = 0
game_watcher_guesses_data = ''
while True:
    time_counter += 1

    time.sleep(1) # this while loop can run up to 5 times , with this its about 5 seconds
    print(self.watchers_enter_database)
    print(self.watchers_logged)
    print(self.watchers_were_logged) # whenever a watcher starts the thread that at the end
of he gets
    # uploaded into the database self.watchers_were_logged grows by 1 , which is why we
use it to
    # compare to self.watchers_enter_database which at the end of said thread grows by 1

    if self.watchers_enter_database >= self.watchers_were_logged or time_counter > 4:
```



# תיכון אורט אביר

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



```
# makes sure all watchers have entered their information into the database
for user in self.guess_list:
    for guess in user:
        game_watcher_guesses_data += guess + ' ';
if time_counter > 4: # 1 or more users timed out and took more than 5 seconds to
confirm
    print('One or more of the users data has not entered the database because they
timed out')
    game_watcher_guesses_data += 'Note - 1 or more users have timed out of the game ;'
    conn = sqlite3.connect('scribble_database.db')
    conn.execute("INSERT INTO paintings VALUES (NULL, ?, ?, ?);",
        (self.guess_word, self.database_information, game_watcher_guesses_data))
    conn.commit()
    conn.close()
    print('Record created successfully')
    for sock in self.watcher_list:
        # noinspection PyBroadException
        try:
            sock.send('database updated'.encode()) # tells the watchers the database has
            # updated and that they can now send the server the reset signal
        except Exception:
            print('watcher crashed or logged off , database confirmation not sent to it')
            # the database had been updated with this game's information
            break

def game_over_socket(self, socket_: socket.socket):
    """
    receives a socket and tells it the game is over , then sends the winner and length of the game
    :param socket_:
    """
    socket_.send('game over'.encode()) # informs that the game is over
    time.sleep(0.01)
    print('== > sending the winner : ' + self.Winner)
    socket_.send(self.Winner.encode()) # tells players who won
    time.sleep(0.01)
    print('== > sending the length of the game: ' + self.game_time)
    socket_.send(self.game_time.encode()) # sends the length of the game

class Manager:
    """
    takes care of the manager window , handling the database information and presenting previous
```



# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



*games*

*in addition to being responsible for opening the server for a new game*

```
def __init__(self):
```

```
    """
```

```
    creates the widgets and parameters used in the Manager class
```

```
    """
```

```
# Parameters that change when the painting is drawn , used to present the painting
```

```
self.pen_color = 'Black'
```

```
self.background_color = 'White'
```

```
self.old_x = '0'
```

```
self.old_y = '0'
```

```
self.shown_games_list = []
```

```
# Defining the tk window and canvas
```

```
self.master = Tk()
```

```
self.master.title("Scribble server")
```

```
self.master.wm_iconbitmap('scribble_logo.ico')
```

```
self.master.geometry("850x500")
```

```
self.master.resizable(width=False, height=False)
```

```
self.master.configure(background='Black')
```

```
self.w = Canvas(self.master, width=500, height=420, bg="white", )
```

```
self.w.pack()
```

```
self.w.place(x=340, y=7)
```

```
# defining what happens if you close the window
```

```
self.master.protocol("WM_DELETE_WINDOW", lambda: exit())
```

```
# finding all available game ids in the data base
```

```
conn_0 = sqlite3.connect('scribble_database.db')
```

```
cur_0 = conn_0.cursor()
```

```
cur_0.execute("SELECT * FROM paintings")
```

```
rows_0 = cur_0.fetchall() # rows_0 holds all of the rows in the database table(paintings)
```

```
# Defining labels who are currently unused
```

```
self.painting_name = Label(self.master, text="", bg='Black', fg='White', font=20) # the name  
of shown painting
```

```
self.action_label = Label(self.master, text="", bg='Black', fg='Red') # label describing the  
action taken
```



# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



```
# Defining labels currently in use

# displays available game IDs in menu bars

self.menu_bar = Menu(self.master)
self.guess_menu = Menu(self.menu_bar)
self.file_menu = Menu(self.menu_bar, tearoff=1)
self.command_list = []
for row_0 in rows_0: # for each row in the table
    self.command_list.append(row_0[0]) # adds the Game id (the first item in every row of
the table)
    self.file_menu.add_command(label=str(row_0[0]),
                              command=partial(self.insert_game, str(row_0[0])))
    print(row_0[0])

self.menu_bar.add_cascade(label="Available game IDs: ", menu=self.file_menu) # shows
available game ids
self.menu_bar.add_cascade(label="Guesses of displayed games: ", menu=self.guess_menu)

self.master.config(menu=self.menu_bar) # puts the menu bar in the tk window

# displaying a new game
label_ = Label(self.master, text="Enter the ID of the game you want to display: ", bg='Black',
fg='White')
label_.pack()
label_.place(x=5, y=10)

self.game_number = Entry(self.master, bd=1) # here the id of the game the manager
wishes to present is written
self.game_number.pack()
self.game_number.place(x=55, y=35)
self.game_number.bind('<KeyPress>', self.enter_press_log_show)

self.enter_button = Button(self.master, text="Display", command=lambda:
self.show_game(self.game_number.get()))
self.enter_button.pack()
self.enter_button.place(x=95, y=60)

# deleting a specific game ID
label_1 = Label(self.master, text="Enter the ID of the game you want to delete: ", bg='Black',
fg='White')
label_1.pack()
label_1.place(x=5, y=120)
```





# תיכון אורט אביר

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



```
self.delete_id = Entry(self.master, bd=1) # here the id of the game the manager wishes to
delete is written
self.delete_id.pack()
self.delete_id.place(x=55, y=145)
self.delete_id.bind('<KeyPress>', self.enter_press_log_delete)

self.delete_id_button = Button(self.master, text="Delete",
                               command=lambda: self.delete_a_game(self.delete_id.get()))
self.delete_id_button.pack()
self.delete_id_button.place(x=95, y=170)

# choosing the number of watchers in the following game
label_watcher = Label(self.master, text="Choose how many watchers can play next game
(default 5)", bg='Black',
                      fg='White')
label_watcher.pack()
label_watcher.place(x=5, y=230)

self.watcher_numb = Entry(self.master, bd=1) # here the number of watchers
self.watcher_numb.pack()
self.watcher_numb.place(x=55, y=255)
self.watcher_numb.bind('<KeyPress>', self.enter_press_log_watcher)

self.watcher_button = Button(self.master, text="Enter",
                              command=self.choose_watcher_num)
self.watcher_button.pack()
self.watcher_button.place(x=95, y=280)

self.label_watcher_error = Label(self.master, text="", bg='Black', fg='White')

# start game button - opens the server for a new game
self.start_button = Button(self.master, text="Start a new game", bg='Black', fg='Green',
                           command=lambda: self.master.destroy())
self.start_button.pack()
self.start_button.place(x=5, y=350)

# delete button - empties database
self.delete_button = Button(self.master, text="Empty database", bg='Black', fg='Orange',
                             command=self.delete_all_games)
self.delete_button.place(x=5, y=400)

# power off button - turns off the server
```



# תיכון אורט אביר

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



```
self.close_server = Button(self.master, text="Power off", bg='Black', fg='Red',
command=lambda: exit())
self.close_server.pack()
self.close_server.place(x=5, y=450)

self.master.mainloop()

def enter_press_log_watcher(self, event): # makes it so you can use enter instead of having to
press the button
    """
    when the enter key is pressed calls the choose_watcher_num() function with the number of
    watchers entered
    :param event:
    """
    if event.keycode == 13:
        self.choose_watcher_num()

def choose_watcher_num(self):
    """
    takes care of making the number of watchers allowed in the next game what the manager
    chose
    """
    global watcher_number
    # noinspection PyBroadException
    try:
        if 10 > int(self.watcher_num.get()) > 0: # if the num is between 1-9
            self.label_watcher_error.destroy()
            watcher_number = self.watcher_num.get()
            print('watcher number is now ' + self.watcher_num.get())
        else:
            print('invalid size')
            self.label_watcher_error.destroy()
            self.label_watcher_error = Label(self.master, text="watcher number must be between 1
and 9", bg='Black',
                                             fg='red')
            self.label_watcher_error.pack()
            self.label_watcher_error.place(x=25, y=310)
    except Exception:
        print('invalid type not int')
        self.label_watcher_error.destroy()
        self.label_watcher_error = Label(self.master, text="watcher number must be an int type",
        bg='Black',
        fg='red')
```



# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



```
self.label_watcher_error.pack()
self.label_watcher_error.place(x=25, y=310)

def insert_game(self, id_of_game):
    """
    shows the game (number: id_of_game) and inserts its Id into the display entry
    :param id_of_game:
    """
    self.show_game(id_of_game)
    self.game_number.delete(0, 'end')
    self.game_number.insert(0, id_of_game)

def show_game(self, game_id): # takes care of showing the chosen game
    """
    takes care of showing the chosen game
    draws the painting of the game
    adds all of the game's guesses to the guesses menu bar
    and shows the name of the game
    :param game_id:
    """

    # resetting the screen
    self.action_label.destroy()
    self.painting_name.destroy()
    self.w['bg'] = 'White'
    self.w.delete(ALL)

    # Getting all of the chosen game's information from the database
    conn_1 = sqlite3.connect('scribble_database.db')
    cur_1 = conn_1.cursor()
    cur_1.execute("SELECT * FROM paintings")

    rows_1 = cur_1.fetchall()

    game_name = "" # the word the users have to guess in the game
    painting = "" # all of the painting information that is needed to redraw it
    guesses = "" # all of the users guesses
    correct_game_id = False # checks if the game id is correct , if its in the database
    shown_game_id = True # checks if the game was already shown before

    # column (0) is 'ID' column (1) is 'painting_name' column (2) is 'painting' and column (3) is
    'guesses'
    for row_1 in rows_1:
```



# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



```
if str(row_1[0]) == str(game_id):
    game_name = str(row_1[1])
    painting = str(row_1[2])
    guesses = str(row_1[3])
    correct_game_id = True

if correct_game_id: # if the game id is in the database
    print(painting)
    print(game_name)
    print(guesses)

if game_id not in self.shown_games_list:
    shown_game_id = False # the game was not shown before
    self.shown_games_list.append(game_id)

self.action_label.destroy()

painting_list = painting.split(';') # 'Coordinates:' or 'Pen:' or 'Background:' or 'stop'
(ignore 'stop')

# Draws the painting
for act in painting_list:
    print(act)

    if act.startswith('Coordinates:'): # paints the coordinates
        draw = act[12:]
        draw_list = draw.split(' ')
        x = draw_list[0]
        y = draw_list[1]
        pen_width = draw_list[2] # pen_width is the pen width
        reset = draw_list[3]
        if reset == '0': # meaning the mouse has not reset
            self.w.create_line(self.old_x, self.old_y,
                               x, y, fill=self.pen_color, width=pen_width, capstyle=ROUND,
smooth=True)
        if reset == '1': # meaning the mouse has reset
            self.w.create_line(x, y, str(int(x) - 0.00001), str(int(y) - 0.00001),
                               fill=self.pen_color, width=pen_width, capstyle=ROUND, smooth=True)
            self.old_x = x
            self.old_y = y

    if act.startswith('Pen:'): # changes the pen color
        self.pen_color = act[4:]
```



# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



```
print('pen color changed to' + self.pen_color)

if act.startswith('Background:'): # changes background color
    self.background_color = act[11:]
    self.w['bg'] = self.background_color
    print('background changed to' + self.background_color)

# resets the colors
self.pen_color = 'Black'
self.background_color = 'White'

# puts the painting's name on the screen
self.painting_name = Message(self.master, text=game_name, bg='Black', fg='White',
anchor='se', font=20,
                                width=300)
self.painting_name.pack()
self.painting_name.place(x=500, y=440)

# shows the guesses by each user and at what time since the start of the game they were
made
if not shown_game_id: # if it was already shown that means the guesses are already at
the guesses menu bar

    guess_menu = Menu(self.guess_menu, tearoff=1)

    guesses_list = guesses.split(';')
    for guess in guesses_list: # puts all guesses in guess menu
        guess_menu.add_command(label=str(guess), command=lambda: print('working')) #
shows guesses

    self.guess_menu.add_cascade(label='Guesses for the word "' + game_name + '" Game
ID: ' + game_id,
                                menu=guess_menu)
    self.master.config(menu=self.menu_bar)

else: # invalid game id (not found in the database)
    print('invalid game ID')
    self.action_label.destroy()
    self.action_label = Message(self.master, text='Error: Invalid game ID', bg='Black', fg='Red',
                                font=20, width=330)
    self.action_label.pack()
    self.action_label.place(x=500, y=440)
```



# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



```
def enter_press_log_show(self, event): # makes it so you can use enter instead of having to
press the button
    """
    when the enter key is pressed calls the show_game function with the game id entered
    :param event:
    """
    if event.keycode == 13:
        self.show_game(self.game_number.get())

def enter_press_log_delete(self, event): # makes it so you can use enter instead of having to
press the button
    """
    when enter is pressed calls the delete_a_game function with the game id entered
    :param event:
    """
    if event.keycode == 13:
        self.delete_a_game(self.delete_id.get())

def delete_all_games(self):
    """
    Delete all rows in the sql database table
    and deletes all guesses/available game ids from their menus
    conn: Connection to the SQLite database
    """

    # Deletes all rows from the database table
    conn_2 = sqlite3.connect('scribble_database.db')
    sql_2 = 'DELETE FROM paintings'
    cur_2 = conn_2.cursor()
    cur_2.execute(sql_2)
    conn_2.commit()

    # clears screen
    self.action_label.destroy()
    self.painting_name.destroy()
    self.w['bg'] = 'White'
    self.w.delete(ALL)

    # clears all guesses/available game ids from their menus
    self.menu_bar.destroy()
    self.action_label = Message(self.master, text='Database emptied', bg='Black', fg='orange',
font=20,
                                width=330)
```



# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



```
self.action_label.pack()
self.action_label.place(x=500, y=440)
self.menu_bar = Menu(self.master)
self.guess_menu = Menu(self.menu_bar)
self.file_menu = Menu(self.menu_bar, tearoff=1)
self.menu_bar.add_cascade(label="Available game IDs: ", menu=self.file_menu)
self.menu_bar.add_cascade(label="Guesses of displayed games: ", menu=self.guess_menu)
self.master.config(menu=self.menu_bar)
```

```
def delete_a_game(self, id_delete):
```

```
    """
```

```
    Delete a row in the sql database table
    and remakes the menus without the deleted game id/guesses
    conn: Connection to the SQLite database
    """
```

```
# deletes a row from the sql database table
conn_3 = sqlite3.connect('scribble_database.db')
sql_3 = ('DELETE FROM paintings WHERE ID = ' + "" + id_delete + "")
cur_3 = conn_3.cursor()
cur_3.execute(sql_3)
conn_3.commit()
print('deleting game number: ' + id_delete)

cur_3 = conn_3.cursor()
cur_3.execute("SELECT * FROM paintings")
rows_3 = cur_3.fetchall() # rows_3 holds all of the database table rows in it
```

```
# noinspection PyBroadException
try: # if the id was shown before
    self.shown_games_list.remove(id_delete)
    print('deleted the game id ' + id_delete + ' from the shown game list')

except Exception: # if the id was not shown yet
    print('Id not shown yet')
```

```
# remakes the menus without the deleted game id/guesses
self.menu_bar.destroy()
self.menu_bar = Menu(self.master)
self.guess_menu = Menu(self.menu_bar)
self.file_menu = Menu(self.menu_bar, tearoff=1)
```

```
for row_3 in rows_3: # for each row in the database table - remakes the shown game ids
```



# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



```
menu without deleted id
self.file_menu.add_command(label=str(row_3[0]),
                           command=partial(self.insert_game, str(row_3[0]))) # shows available
game ids
print(str(row_3[0])) # row_3[0] holds the game id

self.menu_bar.add_cascade(label="Available game IDs: ", menu=self.file_menu)
self.menu_bar.add_cascade(label="Guesses of displayed games: ", menu=self.guess_menu)

self.master.config(menu=self.menu_bar)

print(self.shown_games_list)
for restore_game_id in self.shown_games_list: # remakes the guesses menu without the
deleted game guesses
    if restore_game_id == id_delete:
        print('Id must not be restored because it was deleted')
    else:
        self.reset_game_guesses_menu(restore_game_id)
        print(restore_game_id)

def reset_game_guesses_menu(self, game_id_renew):
    """
    remakes each cascade in the shown game guesses menu
    puts all of the game_id_renew guesses in said cascade
    :param game_id_renew:
    """
    conn_4 = sqlite3.connect('scribble_database.db')
    cur_4 = conn_4.cursor()
    cur_4.execute("SELECT * FROM paintings")

    rows_4 = cur_4.fetchall() # rows_4 holds all of the database table rows in it

    guesses_1 = ""
    game_name_1 = ""

    for row_4 in rows_4: # runs on the database table to find the specific game who's id is
game_id_renew
        if str(row_4[0]) == str(game_id_renew):
            game_name_1 = str(row_4[1]) # the name of the game who's id is game_id_renew
            guesses_1 = str(row_4[3]) # the guesses of the game who's id is game_id_renew

    guess_menu = Menu(self.guess_menu, tearoff=1)
```





# תיכון אורט אבין

מגמת הנדסת תוכנה בהתמחות הגנת Cyber



```
guesses_list_1 = guesses_1.split(';')
for guess_1 in guesses_list_1: # goes over every guess of the game and puts it in the menu
    guess_menu.add_command(label=str(guess_1), command=lambda: print('working')) #
shows guesses

self.guess_menu.add_cascade(label='Guesses for the word ' + game_name_1 + ' Game ID: '
+ game_id_renew,
                             menu=guess_menu)
self.master.config(menu=self.menu_bar)

"""
Main
"""

global watcher_number # holds the max number of watchers allowed to play in the game

while True:

    watcher_number = '5' # by default 5 watchers can play
    print("default number of watchers " + watcher_number)
    Manager()
    print('Opened server for a new game')

    # Setting up sockets

    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # Af_inet - ipv4 ,sock_stream - tcp ,
sock_dgram - udp
    s.bind(('127.0.0.1', 5001)) # which ip and port the server will run from
    s.listen(100) # queue of 100 (up to 100 clients can wait in line to connect to the server)

    PaintServer()

    s.close()

    print('server reset successful')
```