# דו"ח עבודה

מגישים: אריה גרוס

אבי ווסרברגר

## הרצת המערכת:

### הקדמה

להוריד את הפרוייקט מgithub לeclipse

לוודא שמותקן gradle על eclipse (אם לא אז צריך להתקין)

בgradle task להריץ את build

### הרצה

בgradle task

כדי להפעיל את הפרוייקט מריצים את application -> run

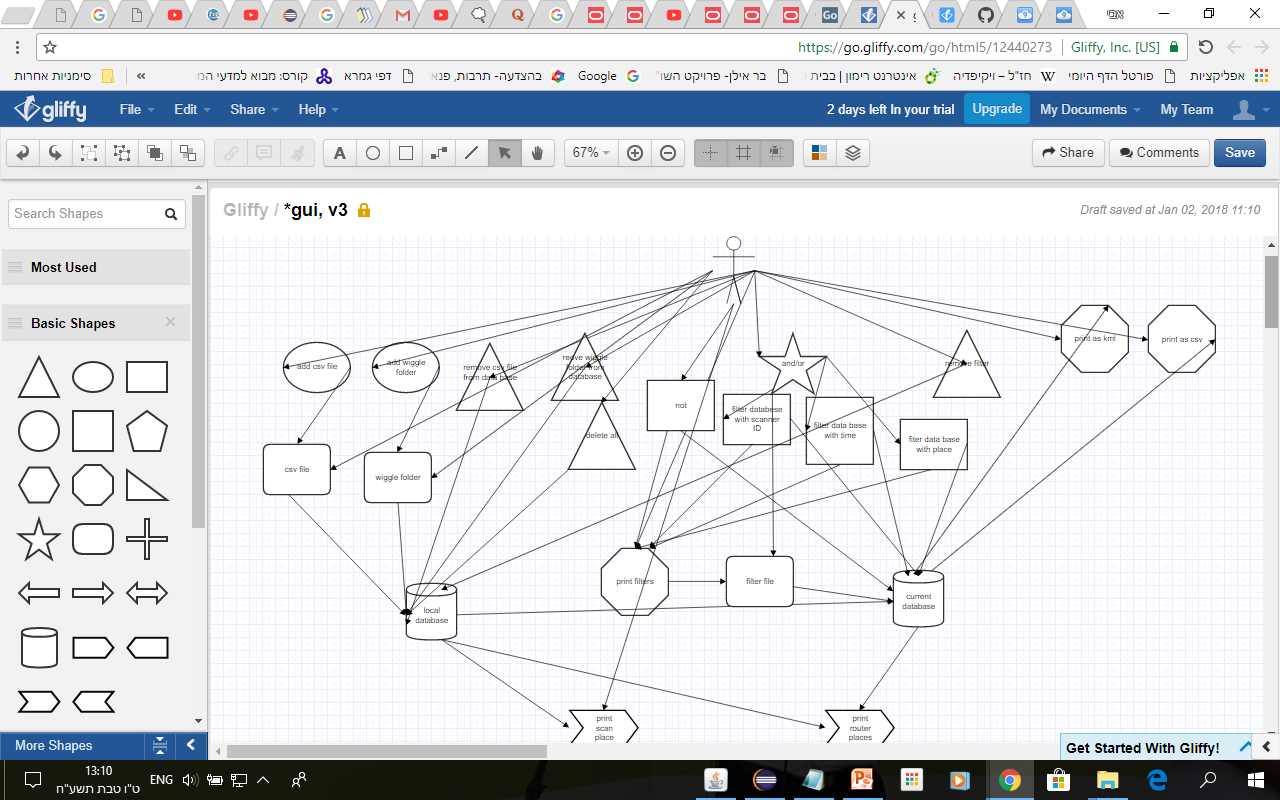
כדי להריץ את הטסטים מריצים את verification -> test

כדי להפוך ל jar מריצים את build -> jar (הוא יימצא בתקייה של המערכת ב build -> libs)

כדי להפוך לzip distribution -> distZip

כדי לנקות קבצים זמניים build -> claen

### מודל כללי



הסבר כללי במילים :בהרצת המערכת המשתמש יוכל להעלות קבצים מסוגים שונים (wiggle,csv,sql) למאגר הנתונים, הוא גם יכול לשנות את הקבצים עצמם ומאגר הנתונים יתעדכן, הוא יוכל לבקש להדפיס את מאגר הנתונים כ csv/kml, הוא יכול לפלטר מכלל מאגר הנתונים למאגר הנתונים המוצג כעת לפי ID ולפי זמן ולפי מיקום, (היחס לפילטר הקודם ביחס OR/AND), לשמור פילטר ולהעלות אותו. כמו כן הוא יכול להדפיס את כל הראוטרים עם המיקומים המשוערכים שלהם ולבקש מיקום משוער של סריקה לפי עוצמת הקליטה שלה של ראוטרים שונים [[1]](#footnote-1)

## עבודת המערכת

המערכת יוצרת עצם של מבנה נתונים ויוצרת תצוגה גרפית, בשלב זה מתפצלים מתוך התכנית הראשית ארבעה חוטים:

אחד שכל שנייה מעדכן בתצוגה הגראפית את מספר הסריקות, מספר הראוטרים ותיאור הפילטר של מבנה הנתונים.

שני שבודק האם קבצי הcsv שהוטענו השתנו.

שלישי שבודק האם נמחק או הוסף קובץ בתקיות הwiggle שהוטענו.

ורביעי שבודק האם השתנה טבלת SQL שהוטענה.

(בשלושה האחרונים אם נוצר שינוי הוא מעדכן את מבנה הנתונים)

במקביל התוכנית הראשית מתפעלת את שאר הפעולות על מבנה הנתונים דרך התצוגה הגראפית, מאפשרת לפלטר את מבנה הנתונים, להטעין אותו בעוד קבצי csv ותיקיות wiggle, אפשרות לשמור את הפילטר בקובץ ולהטעין אותו משם, *לקבל פירוט* של כלל מיקומי הראוטרים ולמצוא מיקום של סריקה מסויימת וכו

## תיאור המערכת ומה פונה למה

### המערכת כוללת ארבע חבילות:

#### Main progect

מכילה **מחלקה ראשית menues** שמתפעלת database מתוך gui כמו שתואר לעיל (פונה רק לעצם database) וניגשת רק אליו (חוץ מהפעלת jtextPrint)

**מחלקה jtextPrint** שמייצרת חלון טקסט שאליו מודפס הoutput של התוכנית (כגון מיקומי ראוטרים, עידכונים לגבי בקשות משתמש שנכשלו וכדומה)

ושתי מחלקות לא פעילות ישנות ex0,ex2 שעובדות עם גירסאות ישנות יותר שהיו עושות את מה שביקשו במטלות אלו ונשארו כדי להראות שאכן עשינו אותן

### inputOutput

**המחלקה WiggleWifi** מכילה מתודה שמקבלת נתיב לתקייה ומחזירה מאגר נתונים מקבצי ה wiggle שבתוכו

**המחלקה CSV** מכילה מתודה שמקבלת נתיב לקובץ ומחזירה מאגר נתונים שבתוכו, וכן מתודה שמקבלת מאגר נתונים ונתיב ומדפיסה את מאגר הנתונים לcsv שם

**המחלקה KML** מכילה מתודה שמקבלת מאגר נתונים ונתיב ומדפיסה את מאגר הנתונים לkml שם

**המחלקה SQL** מכילה בנאי לעצם שמחזיק פרטי התקשרות לטבלת ,SQL מכילה מתודה שמקבלת פרטי התחברות ל SQL ומחזירה מאגר נתונים מקבצי ה SQL שבתוכו, וכן מתודה שמקבלת פרטי התחברות ל SQL ומחזירה את זמן השינוי האחרון שלה

המחלקות שלעיל מקבלות פניות רק מdatabase

**המחלקה generic functions** עם מתודה שמקבלת נתיב לקובץ ומחזירה את מספר השורות שבו (מקבלת פניות משאר המחלקות שבחבילה שלה ומהjunit

### Filters

**abstract: filter** מחלקה עם בנאי של עצם בעל ארבע פרמטרים (לפילטור בעזרתם), ומתודה אבסטרקטית לפילטור באמצעות שני מסדי נתונים (הנוכחי להצגה והכללי, מפלטר את הנוכחי להצגה)

מfilter יורשים **המחלקות and gate, orgate, notgate, filterbycoordinates,filterbytime, filterbyid**

כאשר שלושת הראשונים נועדים לחבר בין פילטרים והאחרונים לפלטר, כל המחלקות מכילות בנאי, פונקציית פילטור, toString. מחלקות אלו מקבלות פניות בעיקר מ filterTreeחוץ מהבנאים שנבנים ישירות מתוך database

**המחלקה** **filterTree** מכילה עצם שמייצג עץ פילטרים, מכיל מתודת הדפסה, מתודת פילטור, בנאי עם ענף ימני ענף שמאלי וצומת נוכחית, בנאי רק עם צומת נוכחית וseters לענפים ולצומת, מקבל פניות מdatabase ומעצמו (כל ענף הוא עץ בפני עצמו), וקורא לפילטרים (שהם הצמתים והעלים עצמם)

### Data

**המחלקה check** מכילה מתודה שמקבלת מחרוזת ומעיפה שגיאה אם זה לא בתבנית זמן תקינה (מקבלת פניות מכלל העצמים שמכילים זמן)

**המחלקה findPlaces** מכילה מתודהשממלאת מיקומים של סריקות במאגר נתונים, ומתודה שמחזירה מיקומים של ראוטרים במאגר נתונים (מקבלת פניה מdatabase)

**המחלקה wifi** מממשת עצם של wifi בודד בסריקה (חלק מהעצמים שלקמן)

**המחלקה sameScanWifi** מממשת עצם של סריקה (שאוסף שלהם מהווה בסיס נתונים)

**המחלקה wifiWithCoordinates** מממשת ראוטר (בשביל מציאת מיקום של ראורים וכדו')

**המחלקה databaseFunctions** מכילה מתודה שמוחקת ראוטרים כפולים מבסיס נתונים (היה בשימוש בex0), מתודה שמחזירה ראוטרים מבסיס נתונים מאוגדים כל ראוטר בפני עצמו (בשימוש כדי למצוא את המיקומים של הראוטרים ולדעת כמה יש), מתודה שמכניסה מאגר נתונים אחד לתוך השני, מתודה שמשבטת מאגר נתונים, מתודה שבודקת האם סריקה מופיעה במאגר נתונים

**המחלקה database** מכילה את מאגר הנתונים הכולל, מאגר הנתונים להצגה, הפילטר הנוכחי, מאגרי הנתונים שנקלט מהם מידע והמקורות שלהם והעדכון האחרון שלהם, היא פונה לכלל העצמים, מתפעלת אותם ויוצרת אותם

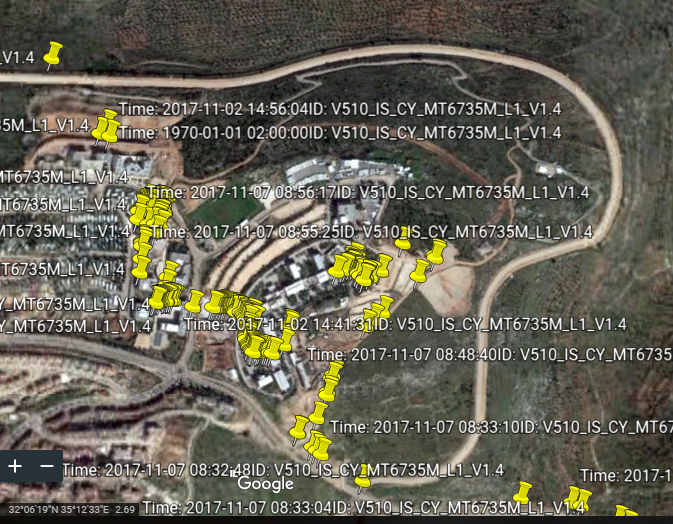
המחלקה מכילה פונקציות שמממשות את כל הבקשות מהgui שלעיל

## בדיקות

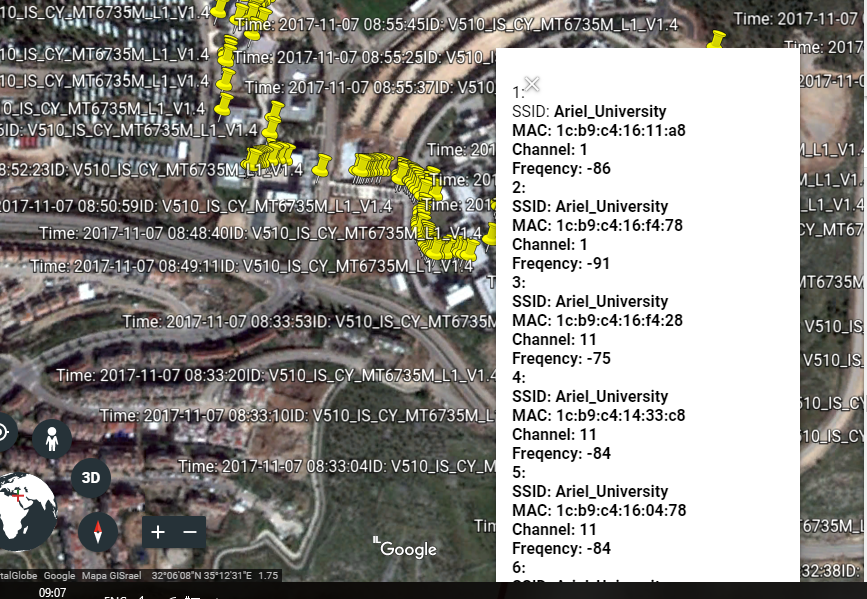
מצורפת תיקיה בשם test עם קבצי אפליקציה תקינים ולא תקינים וכן קבצי csv כנ"ל ומחלקת junit בשם test שבודקת את האפליקציה בעזרתם

# מטלה 0

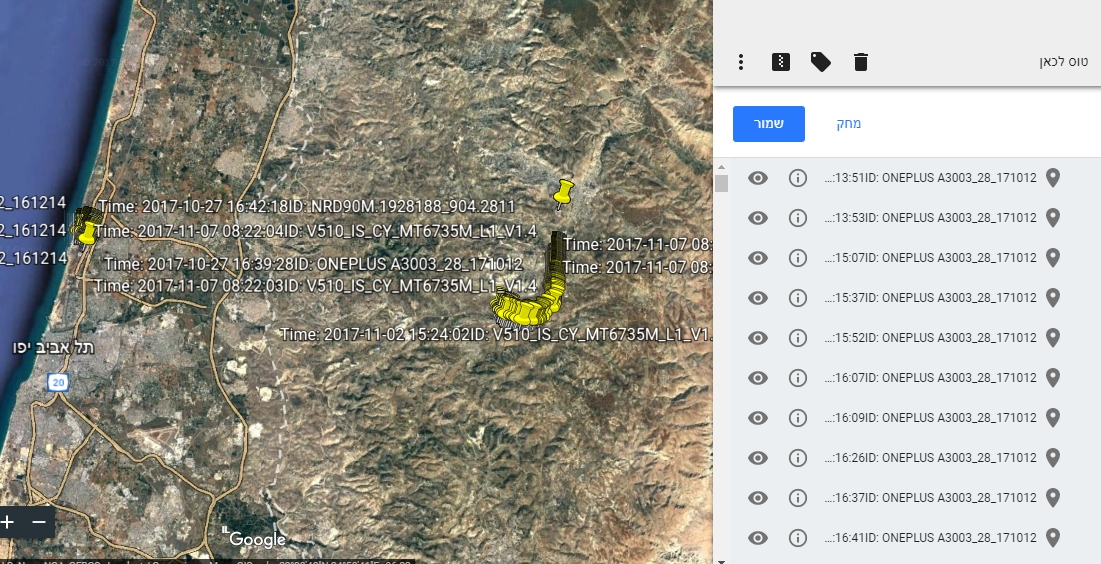
## ניסויים

ביצענו ניסוי באוניברסיטה

למעלה ניתן לראות את נקודות הדגימה עם זמן והID של הבודק

למטה לחצנו על אחת נקודות הדגימה כדי שתציג את התוכן

תמונה של כלל הדגימות כולל הדגימות שהועלו למודל ונסיעה מהר ברכה לאוניברסיטה (אפשר לראות שהgps של הפלאפון לא מדוייק בדגימה הבודדת שנמצאת בשכם בערך, עושה הרבה בעיות עם waze):



# מטלה 1

1. בוצע
2. <https://github.com/amgross/munche_prog>
3. ביצענו חלוקה למחלקות והוספנו עצמים
4. קיים, בתיקייה test
5. השתמשנו בjak, היא עושה רושם התוכנה הכי ידועה ורשמית שעושה את זה ולכן יש לה הרבה תוספות ופיצ'רים וכן הכי הרבה שאלות והסברים ברשת עליה
6. הוספנו
7. הוספנו

# מטלה 2

שינינו את המבנה שישמש למבנה נתונים כללי ולא כפונקציות שונות, שאר הפירוט נמצא בתקיה docs בקובץ word שם

1. בתמונה אין SQL כי נגמר לנו רשיון המשמש הזמני באתר שבו יצרנו את זה בין מטלה 3 ל4 [↑](#footnote-ref-1)