# 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

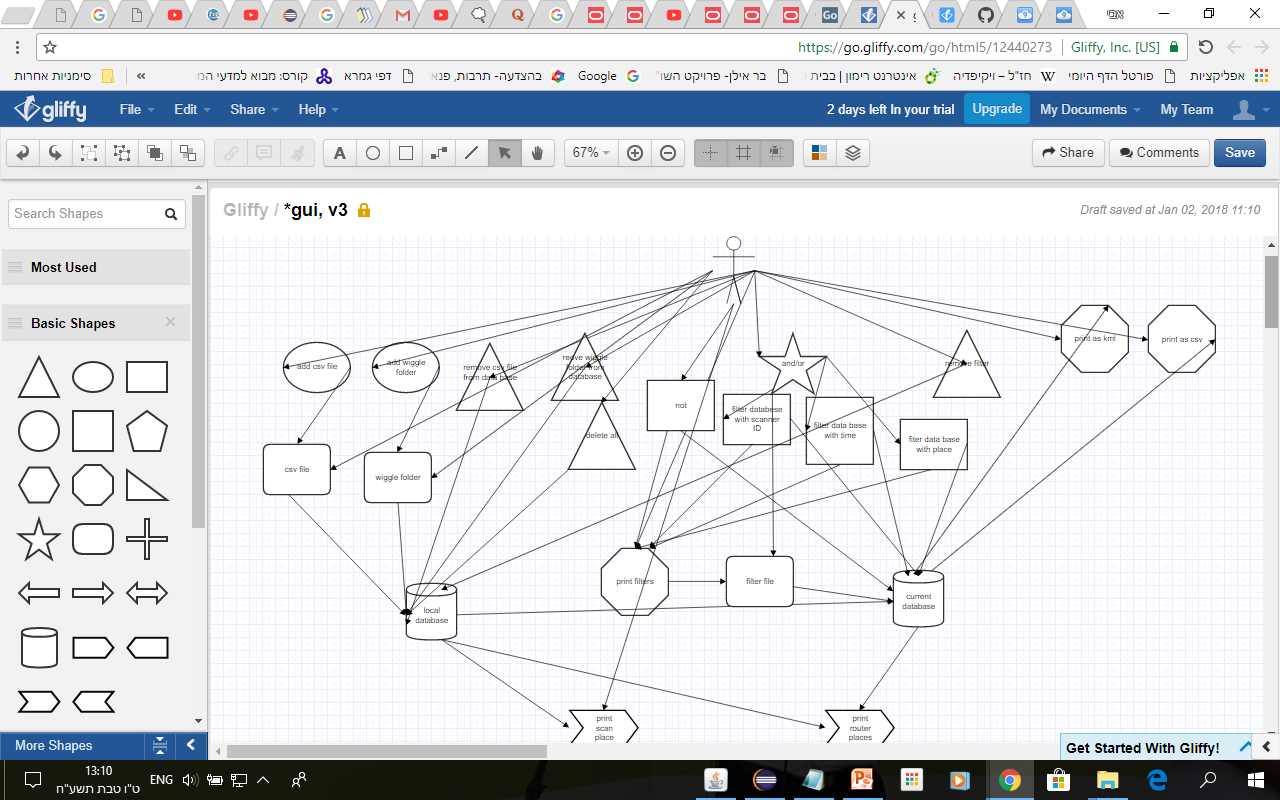
כדי להריץ את הטסטים מריצים את verification -> test

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 (Wwiggle,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

### המערכת כוללת ארבע חבילות:

#### Main progect

מכילה מחלקה ראשית menues שמתפעלת database מתוך gui כמו שתואר לעיל

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

### המערכת כוללת שלושה מחלקות מרכזיות:

**המחלקה WiggleWifi** שמעבירה מקבצי WiggleWifi למסד נתונים, הפונקציה המרכזית מקבלת נתיב של תקיה עם קבצים כאלו בתוכה

**המחלקה CSV** שמעבירה ממסד נתונים לקובץ CSV ומקובץ CSV למסד נתונים, המחלקה מכילה שתי פונקציות מרכזיות, אחת מקבלת מסד נתונים ונתיב של קובץ ויוצרת csv, מחזירה שגיאה אם יש בעיה עם הקובץ. ושנייה מקבלת קובץ ויוצרת מסד נתונים

**המחלקה KML** מעבירה ממסד נתונים לקובץ KML, המחלקה מכילה פונקציה מרכזית אחת שמקבלת מסד נתונים ונתיב של קובץ ויוצרת קובץ, הקובץ יממש timeline

### מחלקות עזר:

**המחלקה databaseFunctions** מכילה פונקציה שמוחקת ממסד הנתונים mac כפולים ומשאירה את הגבוה ביניהם, פונקציה שמחזירה את הMAC הכפולים יחד, פונקציה שמכניסה מאגר נתונים אחד לתוך השני, פונקציה ששואלת את המשתמש באיזה פילטר להשתמש ומחזירה אותו

**המחלקה genericFunctions** נועד בשביל פונקציות בודדות, מכילה פונקציה שסופרת מספר שורות בקובץ ומחזירה (הורדנו מהרשת ושינינו, כתוב בקוד עצמו בהערות)

המחלקה find places מכילה פונקציה פומבית שבהנתן מסד נתונים של סריקות משערכת את מיקום הראוטרים (בעזרת מספר שמייצג לפי כמה סריקות לבדוק) ומכילה פונקציה שבהנתן שני מסדי נתונים משערכת באמצעות הראשון את מיקום הסריקות של השני (נעזרת במספר שמייצג בכמה סריקות להשתמש כדי לשערך)

**המחלקה test** מהווה מחלקת בדיקות לכלל הפרוייקט, משתמשת בקבצים שנמצאים בתקיה test

**המחלקה ex0** מהווה את הפונקציות שבקשו מאיתנו בex0

**המחלקה ex2** מהווה את האלגוריתמים שבקשו מאיתנו בex2

### עצמים:

**העצם wifi** שמכיל פרטים על wifi

**העצם** **sameScanWifi** שמכיל עד 10 wifi ומיקום איפה נמדד ות.ז. של המכשיר המודד וזמן המדידה

**העצם wifiWithCoordinates** יורש מwifi וכן מכיל קואורדינטות (נועד למציאת מיקום wifi בודד)

מאגר מידע הוא בעצם ווקטור של sameScanWifi

### ממשקים:

**ממשק ומימוש של סינונים**, הממשק מכיל נתונים לסינון ופונקציה שנועדה למימוש של הסינון עצמו

הפונקציות משתמשות בפונקציה שמצאנו ברשת שסופרת מספר שורות בקובץ (המקור רשום שם).

## כלי תוכנה

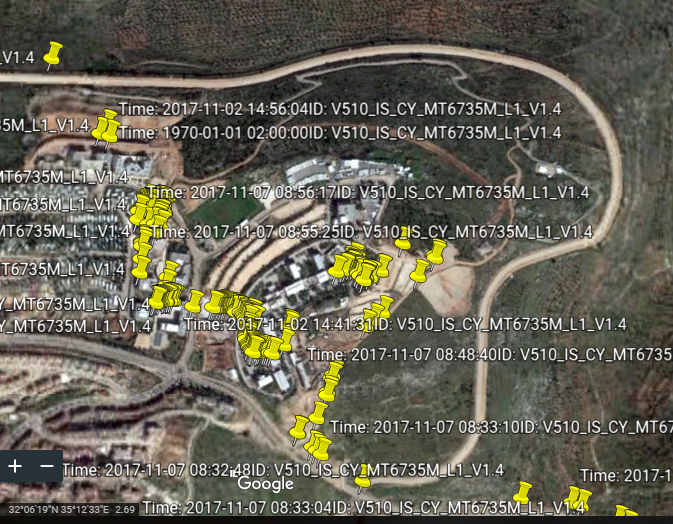
השתמשנו בספריות הרגילות בג'אווה, בממשק, בלכידת שגיאות [try-catch](על מנת לסנן קלט שגוי) וב java api for kml (jak) ובאיטרטור ובווקטור ןב hashmap ובירושה

## בדיקות

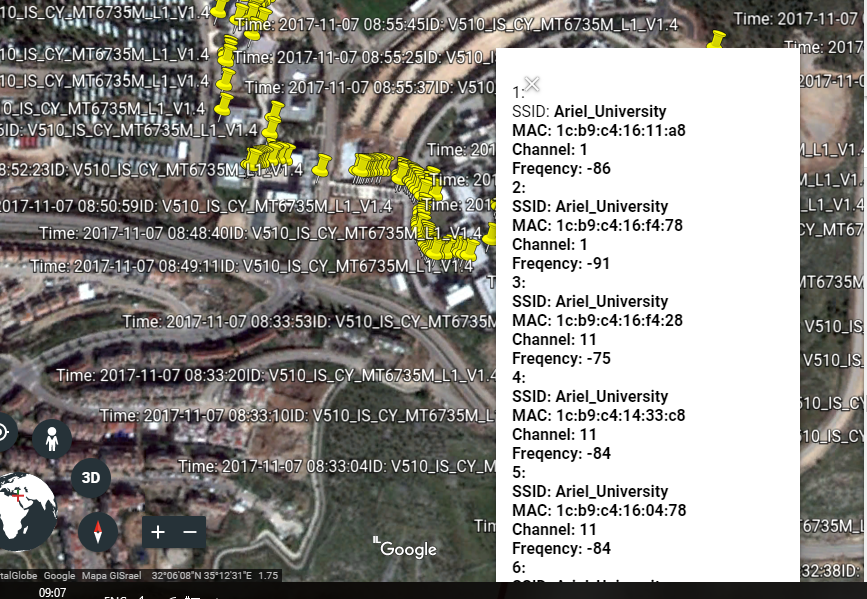
מצורפת תיקיה בשם test עם קבצי אפליקציה תקינים ולא תקינים וכן קבצי csv כנ"ל ומחלקת junit בשם test שבודקת את האפליקציה בעזרתם

# מטלה 0

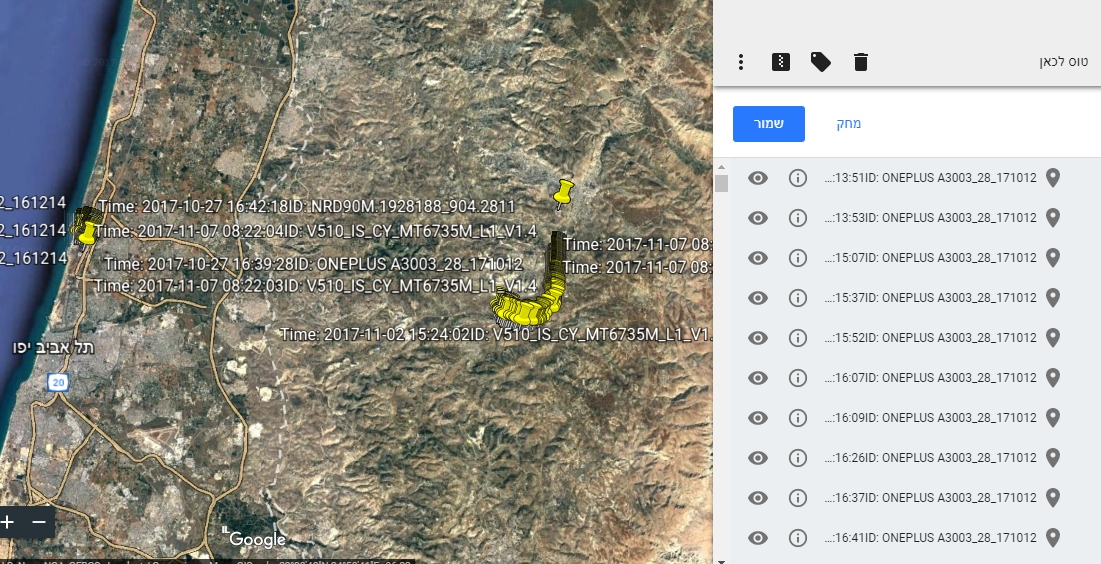
## ניסויים

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)