תמונה שמכילה ציור

התיאור נוצר באופן אוטומטי

מנחה: אלי גולדשטיין

תיכון הדרים הוד השרון | 27.05.2020

Shazam clone

גיא ון-דם 325133148

תוכן עניינים

[מבוא 3](#_Toc41922843)

[תיאור תכולת הספר 3](#_Toc41922844)

[הרקע לפרויקט 3](#_Toc41922845)

[תהליך המחקר 3](#_Toc41922846)

[סקירת ספרות 3](#_Toc41922847)

[אתגרים מרכזיים : 3](#_Toc41922848)

[מבנה – ארכיטקטורה של הפרויקט 4](#_Toc41922849)

[הפתרון המוצע 4](#_Toc41922850)

[הסבר על הרכיבים השונים 5](#_Toc41922851)

[Fingerprint algorithm 6](#_Toc41922852)

[Searching algorithm 7](#_Toc41922853)

[מדריך למשתמש 10](#_Toc41922854)

[הוראות התקנה 11](#_Toc41922855)

[Server/Database 11](#_Toc41922856)

[Running the Server 12](#_Toc41922857)

[Client/android application 12](#_Toc41922858)

[Running the Client/app 12](#_Toc41922859)

[מדריך למפתח - קבצי פייתון 13](#_Toc41922860)

[DBServer.py 14](#_Toc41922861)

[DatabaseItemsFunctions.py 17](#_Toc41922862)

[FingerprintDatabase.py 18](#_Toc41922863)

[LoadToDatabase.py 23](#_Toc41922864)

[Recording.py 24](#_Toc41922865)

[Song.py 27](#_Toc41922866)

[TargetZone.py 29](#_Toc41922867)

[FFTPreparationFunctions.py 30](#_Toc41922868)

[SpectrogramFilteringFunctions.py 32](#_Toc41922869)

[SpectrogramFunctions.py 35](#_Toc41922870)

[GeneralFunctions.py 36](#_Toc41922871)

[GlobalValues.py 37](#_Toc41922872)

[ImportsFile.py 38](#_Toc41922873)

[מדריך למפתח – קבצי ג'אווה/אנדרואיד 39](#_Toc41922874)

[AndroidManifest.xml 39](#_Toc41922875)

[acitivity\_main.xml 41](#_Toc41922876)

[MainActivity.java 43](#_Toc41922877)

[MsgSender.java 51](#_Toc41922878)

[SocketInfo.java 53](#_Toc41922879)

[WavRecorder.java 54](#_Toc41922880)

[StringObj.java 59](#_Toc41922881)

[רפלקציה 60](#_Toc41922882)

[ביבליוגרפיה 61](#_Toc41922883)

[נספחים 62](#_Toc41922884)

[קישורים ל GitHub repositories 62](#_Toc41922885)

[תוצאות הבדיקה 62](#_Toc41922886)

[Creating a new Database 63](#_Toc41922887)

[יומן שיקוף 64](#_Toc41922888)

# מבוא

תיאור תכולת הספר

* + הרקע לבחירתי בפרויקט הארכיטקטורה של הפרויקט
  + תהליך בנייתו
  + נספחים
  + קישורים למקורות מידע
  + הקוד שכתבתי..

## הרקע לפרויקט

חיפשתי נושא מעניין לפרויקט שלי, ניסיתי להמציא תכנים מקוריים אבל הם לא צלחו עקב חוסר ידע ומידע. הופיע לי סרטון שמסביר על איך Shazam עובד וישר ידעתי שזה רעיון מצוין לפרויקט, הנושא מעניין, יש לנו תקשורת בין שרת ללקוח ולפי כל מיני מאמרים באינטרנט הבנתי שאני באמת אוכל לממש תוכנה כזאת.

## תהליך המחקר

כמובן שזה לא רעיון חדש או מקורי, ואין שום חידושים בפרויקט שלי כי אני עובד עם האלגוריתם של 2003.

## סקירת ספרות

לפני שהתחלתי רציתי לוודא שיש לי חומר ומידע על איך האלגוריתם עובד. מצאתי מאמר מצוין שמסביר בצורה ברורה ונכנס לפרטים איך האלגוריתם עובד ואיך הוא ממומש.   
כמובן שהמאמר הזה לא מכסה את כל מה שהייתי צריך בשביל לממש את האלגוריתם, היו בו קצת חורי מידע, והייתי צריך למצוא מאמרי עזר שנכנסו ליותר פרטים בנושאים מסוימים. כלומר, עם קריאת המאמר וההתקדמות בו, הייתי צריך לחפש מאמרים ומידע באינטרנט על נושאים לא ברורים, חורי מידע, דרכי מימוש וכו..

## אתגרים מרכזיים :

## 

* הבעיה המרכזית שהתמודדתי איתה הייתה רמת הידע. Shazam נוצרה ע"י חוקרי signal processing ו computer science מאוניברסיטאות מתקדמות בארה"ב. זה חומר ברמת תואר ראשון-שני שאיתו הייתי צריך להתמודד. זאת הייתה בעיה מרכזית עבורי בגלל שלא רק שלא ידעתי את החומר, קשה למצוא מידע באינטרנט שמסביר **בדיוק** את מה שאני צריך.
* הסיבות שלי לבחירת הנושא היו התעניינות באיך האלגוריתם עובד ובעולם של ניתוח צליל, האתגר לנסות לממש תוכנה מסובכת ומרשימה כמו Shazam וכמובן בגלל שהנושא עמד בכל התנאים עבור פרויקט ברשתות.
* המוטיבציה שלי לעבודה הייתה לסיים את הפרויקט בהצלחה, העשרת הידע שלי, הן בפייתון והן בנושא של הפרויקט, וידיעה שהפרויקט יכול לפתוח לי דלתות בצבא ובנוסף 5 יחידות הלימוד.
* הפרויקט לא עונה על שום צורך. קיימת מערכת כזאת, ומראש ידעתי שהיא תעבוד יותר טוב ממה שאני אעשה. לא רק שאני מתחיל בפיגור של 17 שנה, אני תלמיד יחיד בתיכון עם זמן וידע מוגבל.

# מבנה – ארכיטקטורה של הפרויקט

הפתרון המוצע

הפתרון במילה אחת הוא audio fingerprint, טביעות אבצע כאלה נותנות לך המון אינפורמציה במעט זיכרון. חישוב טביעת האצבע לא לוקחת המון זמן, היא לא יותר מדי תובענית בצורך של משאבי חישוב (דבר חשוב עבור חישוב מהיר וחיפוש מהיר במאגר הנתונים).

תמונה שמכילה צילום מסך

התיאור נוצר באופן אוטומטי

כפי שניתן לראות, לאחר שהטלפון מקליט את השיר שאנו רוצים לחפש, הוא מחשב את טביעת האצבע של ההקלטה, שולח אותה לשרת לחיפוש מהיר במאגר הנתונים של טביעות האצבע. אם הכל הולך כשורה, השרת ישלח תשובה עם המידע על השיר המבוקש. זו ארכיטקטורה יחסית פשוט, היא מקטינה את כמות המידע שעובר ברשת, ומשתמשת בתכונות של מעבדי הטלפון החזקים, עבור חישוב יחסית פשוט ומהיר של טביעת האצבע של ההקלטה.

## הסבר על הרכיבים השונים

* טלפון המשתמש :
  + תפקידה להקליט את השיר המבוקש, ולהציג את המידע למשתמש. זה ה UI שלנו.
  + הקלט הוא הקול שמוקלט מהסביבה.
  + הפלט הוא המידע על השיר המבוקש (אם האופרציה עבדה כמו שמתכונן) או הודעת שגיאה אם לא.
* השרת :
  + תפקידו הוא להחזיק את מאגר הנתונים (הן של השירים והן של טביעות האצבע). לבצע את החיפוש במאגר הנתונים ולשלוח את התוצאות ללקוח.
  + הקלט הוא טביעות האצבע מהלקוחות.
  + הפלט הוא תוצאות החיפוש במאגר הנתונים, אשר ישלחו ללקוח.
* השרת משתמש ב 3 hash tables :
  + אחד בשביל ה fingerprint database עם מפתח - address (הסבר בהמשך) וערך – רשימת couple ים (הסבר בהמשך).
  + השני בשביל ה address-Couple Dictionary של כל Song object ב database.
  + השלישי בשביל ה song information, השם של השיר והאמן.
* זרימת מידע : קל לראות שמה- song database עוברים ל fingerprint table דרך ה fingerprint algorithm וכמובן שיש זרימת מידע בין השרת ללקוח – הלקוח מעבירה שאילתה והשרת מחזיר את התשובה.
* ארכיטקטורת רשת :
  + אנחנו נרצה שרת מרובה לקוחות שיוכלו לדבר עם השרת בו זמנית
  + .
  + כמובן שיש לנו תקשורת של שרת-לקוח.
  + אזורים הדורשים אבטחה יהיו :
    - השרת, שבתוכו נצטרך להגן על מאגרי הנתונים ואלגוריתם החיפוש.
    - כדאי שהחיבור בין השרת ללקוח יהיה מאובטח כדי שלא יהיה מעבר שגוי של נתונים..
* האלגוריתם הראשי : עקרונית, האלגוריתם הראשי כבר מוצג למעלה, עכשיו נצטרך לעבור לאלגוריתמים המשנים שהם ה fingerprint algorithm וה- searching algorithm.

## Fingerprint algorithm

תמונה שמכילה צילום מסך

התיאור נוצר באופן אוטומטי

## Searching algorithm

תמונה שמכילה טקסט, מפה

התיאור נוצר באופן אוטומטי

לפי שנתסכל על אלגוריתם החיפוש נעשה קצת סדר בטרמינולוגיה בעזרת דוגמא:

אחרי אלגוריתם טביעת האצבע אנחנו מקבלים time-frequency peak points. תמונה זו מייצגת נקודות אלה.

נמספר את הנקודות לפי זמן ואח"כ לפי גודל תדירות כפי שאפשר לראות בתמונה.

אנחנו נקרא **לכל** 5 נקודות עוקבות **target zone**. כמו 5 הנקודות שמסביבן ציור אדום.  **anchor point** של ה target zone הזה תהיה נקודה מס' 1. הנקודה השלישית לפני ה target zone.

עבור כל נק' בתמונה נוכל ליצור ,**address** שלשה המתוארת ככה :

(frequency of the point, frequency of the anchor point, time of point – time of anchor point)

ועבור כל target zone נוכל ליצור **couple** שהוא זוג שמתואר כך :

(time of anchor point, song id)

נעשה את התהליך הזה עבור כל נקודה. לכל נקודה (שהיא לא ההתחלה או הסוף) יהיה 5 addresses (כי היא מופיעה ב 5 target zones).

לכל שיר והקלטה ניצור dictionary עם

* מפתח – address
* ערך – רשימה של couple ים שמתאימים ל address. (כי address יוצר couple)

**אבל,** אנחנו יודעים שלהקלטה אין song id אז פשוט נשמיט אותו ונסתכל רק על ה time of anchor point. ונקבל dictianary עם

* מפתח – address
* ערך – time of anchor point

נקרא למילון של השיר addressCoupleDict  
ולמילון של ההקלטה נקרא addressTimeOfAnchorDict  
  
ועכשיו נוכל להבין איך נראה ה fingerprint database שלנו.  
הוא בעצם dictionary/hash table עם

* מפתח - address
* ערך - רשימה של couples שמתאימים ל address הזה. שיכול להכילsong id מכל שיר ממאגר השירים.

**אבל** אמרנו שיש לנו 3 hash tables במאגר הנתונים. המאגר השני הוא גם hash table כפי שנאמר, עם

* מפתח – song id.
* ערך – addressCoupleDict של ה Song object associated with the song id.

מאגר הנתונים השלישי הוא hash table עם

* מפתח – song id.
* ערך – song Information המידע על השיר שיופיע למשתמש, השם של השיר והאמן.

עכשיו נוכל להסתכל על אלגוריתם החיפוש :

תמונה שמכילה צילום מסך

התיאור נוצר באופן אוטומטי

# מדריך למשתמש

תמונה שמכילה צילום מסך

התיאור נוצר באופן אוטומטי

האפליקציה די פשוטה. מוצגות לנו הוראות לגבי כמה זמן כדאי להקליט ואיפה אפשר לצפות לשם השיר המבוקש.

המשתמש מתבקש להכניס את ה IP וה Port של ה socket שאליו נרצה להתחבר אצל השרת.

לא יהיה ניתן להכניס IP או Port לא נכונים.

יש לנו מעיין שעון שמורה לנו כמה שניות עברו מתחילת ההקלטה כדי שנוכל לדעת מתי הגענו לטווח השניות המבוקש.

ממשק המשתמש די פשוט. כשרוצים להתחיל להקליט לוחצים על START RECORDING וכשרוצים לסיים להקליט לוחצים על STOP RECORDING.

אם לא הצלחנו למצוא את השם של השיר מופיעה הודעה שאומרת שהשיר לא נמצא.

"didn't find anything :("

אם כן הצלנו למצוא משהו, השם והאמן של השיר יופיעו (כפי שנאמר) בתחתית המסך.

זה המסך היחיד שלנו.

## הוראות התקנה

### Server/Database

נתחיל בהורדת פייתון 3 למחשב:

<https://www.python.org/downloads/>

אח"כ נוריד את MongoDB ואת ה connector של פייתון למחשב כפי שמתואר במאמר הזה:

<https://www.geeksforgeeks.org/guide-install-mongodb-python-windows/>

נצטרך להוריד גם את התיקיות של פייתון שנשתמש בהם. ניתן לעשות זאת ע"י pip install או דרך ה IDE שאיתו נשתמש. נוריד את:

numpy  
pymongo  
scipy  
wavefile  
stft  
np

נסדר את הקבצי הפייתון לפי התיקיות המוצגות בעמוד הבא.

את השירים שנרצה להעלות למאגר הנתונים נצטרך לשמור בתור קבצי wave (.wav) בלי הסיומות שמוספות כשאתה מוריד את הקובץ. ניתן לעשות את השלב הזה ע"י מציאת השירים ב youtube ושימוש ב:

<https://ytmp3.cc/en13/>

אח"כ נשתמש ב Audacity כדי להעביר את הקבצים לפורמט הרצוי ומחיקת הסיומות שעלולות להפריע בקריאת הקבצים ע"י פייתון.

<https://www.audacityteam.org/download/>

נשים אותם בתיקיית DatabaseSongs.  
  
אח"כ ניכס לקובץ FingerprintDatabase.py ניצור פונקציית main ובתוכה מופע של FingerprintDatabase.

נקרא לפונקציה createNewDatabase

createNewDatabase(self)

אם נרצה להכניס שיר אחד:   
נשתמש בפונצקיה load של האובייקט שמקבלת את ה path, songId, songInfo

load(self, *path*, *songId*, *songInfo*)

אם נרצה להכניס כמה שירים בבת אחת:  
ניצור 2 dictionaries, הראשון עם

* מפתח – songId
* ערך – songInfo

והשני עם

* מפתח – songId
* ערך – path/location to the wavfile

נשתמש בפונקציה loadMany של האובייקט ונכנס את ה dictionaries לעיל בתור קלט.

loadMany(self, *songIdPathDict*, *songIdInfoDict*)

זה מה שקורה מאחורי הקלעים.  
אנחנו נשתמש בקובץ LoadToDatabase.py כדי להכניס כקלט למערכת את מיקומי הקבצים, ה song id שלהם והמידע עליהם (שם השיר והאמן). נריץ את הקובץ ב cmd או ב IDE שלנו וההוראות ברורות משם.

### Running the Server

ניצור מופע של DBServer ונריץ אותו ע"י קריאה לפונקציה run של האובייקט.

run(self)

### Client/android application

עבור הקבצים של ה אנדרואיד, נוסיף את המחלקות:

MsgSender.java  
SocketInfo.java  
StringObj.java  
WavRecorder.java

עם התוכן שמוצג בדפים הבאים. ונעתיק את קצבי ה MainActivity.xml/.java ואת ה AndroidManifest שמופיעים בכל פרויקט אנדרואיד. עדיף לקרוא לפרויקט ProjectApp בשביל העתקה חלקה. אחרת צריך לשנות שורה ב AndroidManifest וב MainActivity.xml/.java כדי לתקן את ה package name.

### Running the Client/app

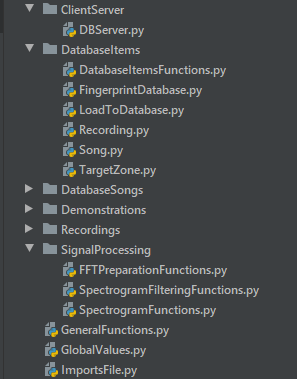
נבחן את הסרטון הבא:

<https://www.youtube.com/watch?v=13DPpfuP1Zs>

הוא יסביר לנו איך להריץ את הפרויקט על ה emulator ועל טלפון פיזי.

**חשוב להדגיש** הפרויקט יעבוד רק על טלפון פיזי בגלל שהקלטות ב emulator לא עובדות.

# מדריך למפתח - קבצי פייתון



הקבצים והתיקיות שלהם:

## DBServer.py

import ImportsFile  
import socket  
import select  
from DatabaseItems.FingerprintDatabase import FingerprintDatabase  
from DatabaseItems.Recording import Recording  
  
  
class DatabaseServer:  
 def \_\_init\_\_(self, *port*):  
 self.serverSocket = socket.socket()  
 self.serverSocket.bind(("0.0.0.0", *port*))  
  
 self.fingerprintDatabase = FingerprintDatabase()  
  
 self.serverSocket.listen(5)  
  
 self.openClientSockets = []  
 self.messagesToSend = []  
  
 self.recordingPath = r'C:\PythonProject2\Recordings\clientRecording.wav'  
  
 """  
 function name: sendWaitingMessages.  
 input: wlist - a list of writeable socket.  
 output: N/A  
 operation: loops through the input list and sends the messages in the 'messagesToSend' list if the socket is   
 writable  
 """  
  
 def sendWaitingMessages(self, *wlist*):  
  
 for *message* in self.messagesToSend:  
 (*client\_socket*, *data*) = *message* if *client\_socket* in *wlist*:  
 print("data is" + *data*)  
 *data* = *data*.encode('latin-1')  
 *client\_socket*.send(*data*);  
 print("sent message")  
 self.messagesToSend.remove(*message*)  
  
 """  
 function name: updateTheWavFile.  
 input: currentSocket, the current socket that needs attention.  
 output: the result of the search.  
 operation: updates the 'clientRecording.wav' file, and searches the database to find the information about the   
 recording.   
 """  
  
 def updateTheWavFile(self, *currentSocket*):  
  
 with open(self.recordingPath, 'wb') as *f*:  
 while True:  
  
 *m* = *currentSocket*.recv(1024) # try to receive 100 bytes  
 *f*.write(*m*)  
 if not *m*:  
 print("finished because m is empty")  
 break  
 if "finish" in *m*.decode('latin-1'):  
 break  
  
 *searchResult* = self.searchInDatabase() + "\n"  
 *currentSocket*.send(*searchResult*.encode('latin-1'))  
  
 """  
 function name: run.  
 input: N/A  
 output: N/A  
 operation: runs the server, channels the clients.  
 """  
  
 def run(self):  
 print("server is up")  
 while True:  
 *client\_data\_exist* = False  
  
 *rlist*, *wlist*, *xlist* = select.select([self.serverSocket] + self.openClientSockets, [], [])  
  
 for *current\_socket* in *rlist*:  
  
 if *current\_socket* is self.serverSocket:  
  
 (*new\_socket*, *address*) = self.serverSocket.accept()  
  
 self.openClientSockets.append(*new\_socket*)  
  
 else:  
 *client\_data\_exist* = True  
 *data* = *current\_socket*.recv(1024)  
 *data* = *data*.decode('latin-1')  
  
 if *data* == "":  
 self.openClientSockets.remove(*current\_socket*)  
 elif str(*data*[:4]) == "file":  
 self.updateTheWavFile(*current\_socket*)  
 if *client\_data\_exist*:  
 *dummyrlist*, *wlist*, *xlist* = select.select([], *rlist*, [], 0.1)  
 self.sendWaitingMessages(*wlist*)  
  
 """  
 function name: showCollection.  
 input: N/A  
 output: N/A  
 operation: prints the MongoDB collection, our database.  
 """  
  
 def showCollection(self):  
 self.fingerprintDatabase.showCollection()  
  
 """  
 function name: searchInDatabase.  
 input: N/A  
 output: the result of the search in the database.  
 operation: calls the 'searchInDatabase' function of the FingerprintDatabase object with a recording associated   
 with the 'clientRecording.wav' file"""  
  
 def searchInDatabase(self):  
 *r* = Recording(self.recordingPath)  
 *r*.initializeAll()  
 return self.fingerprintDatabase.searchInDatabase(*r*)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 *port* = 8888  
 *server* = DatabaseServer(*port*)  
 *server*.run()

## DatabaseItemsFunctions.py

import GlobalValues  
  
"""  
function name: targetZonesCreate.  
input: a list of time-frequency points.  
output: a list of lists.  
operation: creates a list of 5 consecutive points, until it runs out of points. adds that list to the "targetZone" list.  
"""  
  
  
def createTargetZones(*timeFrequencyPoints*):  
 *targetSize* = GlobalValues.targetSize  
 *targetZones* = []  
 for *i* in range(len(*timeFrequencyPoints*) - *targetSize* + 1):  
 *targetZones*.append([*timeFrequencyPoints*[*i* + *j*] for *j* in range(*targetSize*)])  
  
 return *targetZones*"""  
 function name: createAnchorPoints  
 input: timeFrequencyPoints - as the name suggest, a list of tuples. targetZones - a list of point lists.   
 output: a dictionary. where the key is an anchor point and the value is a list of points - the target Zone.   
 (5 following points, 3 points to the right [where the positive time axis is to the right] of the anchor point).  
 operation: initializes the 'anchorPointTargetZoneDict' variable for the Song and Recording objects.  
 """  
  
  
def createAnchorPoints(*timeFrequencyPoints*, *targetZones*):  
 return {*timeFrequencyPoints*[*i* - 3]: *targetZones*[*i*] for *i* in range(3, len(*targetZones*))}

## FingerprintDatabase.py

from ImportsFile import \*  
from DatabaseItems.Song import Song  
import pymongo  
  
  
def stringToTuple(*s*):  
 try:  
 return tuple(map(float, *s*.split(',')))  
 except Exception:  
 return *s*def addressToString(*tup*):  
 return str(*tup*[0]) + ',' + str(*tup*[1]) + ',' + str(*tup*[2])  
  
  
class FingerprintDatabase:  
 def \_\_init\_\_(self):  
 *myClient* = pymongo.MongoClient("mongodb://localhost:27017/")  
  
 *myDataBase* = *myClient*["mydatabase"]  
 self.collection = *myDataBase*["Database collection"]  
  
 # key - an address, value - a list of couples matching the key.  
 self.database = {}  
  
 self.storageDatabase = {}  
  
 self.songIdSongInfoDict = {}  
  
 self.songIdAddressCoupleDict = {}  
  
 '''  
 function name: loadMany.  
 input: a dictionary of paths and songId to be loaded to the fingerprint database.  
 output: N/A  
 operation: self explanatory. calls the load function for every item in the database.  
 '''  
  
 def loadMany(self, *songIdPathDict*, *songIdInfoDict*):  
 if not *songIdPathDict* or not *songIdInfoDict*:  
 return  
 self.pullAll()  
  
 for *songId*, *path* in *songIdPathDict*.items():  
 print("loading:", *songIdInfoDict*[*songId*])  
 *temp* = Song(*path*, *songId*)  
 try:  
 *temp*.initializeAll()  
 self.databaseUpdate(*temp*)  
 self.songIdAddressCoupleDict[*songId*] = *temp*.addressCoupleDict  
 self.songIdSongInfoDict[*songId*] = *songIdInfoDict*[*songId*]  
 except:  
 pass  
 self.pushAll()  
  
 '''   
 function name: load.  
 input: path, the song id.  
 output: N/A  
 operation: creates the address for the point in the address object, load them in the "database" dictionary,  
 if a key already exist, adds the match couple to the value (a list) of the key, if not, creates a new key and a list  
 of one object, so it can be expanded in the future.  
 '''  
  
 def load(self, *path*, *songId*, *songInfo*):  
 self.pullAll()  
  
 print("loading: ", *songInfo*)  
 *temp* = Song(*path*, *songId*)  
 *temp*.initializeAll()  
  
 self.databaseUpdate(*temp*)  
 self.songIdAddressCoupleDict[*songId*] = *temp*.addressCoupleDict  
 self.songIdSongInfoDict[*songId*] = *songInfo* self.pushAll()  
  
 '''  
 function name: saveDatabase.  
 input: N/A  
 output: N/A  
 operation: saves the database changes to memory.  
 '''  
  
 def pushFingerprintDatabase(self):  
 self.collection.delete\_one({"\_id": 1})  
 try:  
 self.storageDatabase["\_id"] = 1  
 self.collection.insert\_one(self.storageDatabase)  
 except pymongo.errors.DuplicateKeyError:  
 print('===== ERROR === DuplicateKeyError for dict 1 === ERROR =====')  
 exit(1)  
 else:  
 print('===== changes to fingerprintDatabase SAVED SUCCESSFULLY =====')  
  
 '''  
 function name: pullDatabase  
 input: N/A  
 output: N/A  
 operation: pulls the database from memory so we can alter it.  
 '''  
  
 def pullFingerprintDatabase(self):  
 self.storageDatabase = self.collection.find\_one({"\_id": 1})  
 assert self.storageDatabase, "===== ERROR === error at pullFingerprintDatabase === ERROR ===== "  
 self.database = {stringToTuple(*key*): [tuple(*v*) for *v* in *value*] for *key*, *value* in self.storageDatabase.items() if  
 not *key* == '\_id'}  
  
 '''  
 function name: saveSongIdAddressCoupleDict  
 input: N/A  
 output: N/A  
 operation: saves the songIdDict to changes to memory.  
 '''  
  
 def pushSongIdAddressCoupleDict(self):  
 self.collection.delete\_one({"\_id": 2})  
 try:  
 self.collection.insert\_one(self.songIdAddressCoupleDict)  
 except pymongo.errors.DuplicateKeyError:  
 print('===== ERROR === DuplicateKeyError for dict 2 === ERROR =====')  
 else:  
 print('===== changes to songIdAddressCoupleDict SAVED SUCCESSFULLY =====')  
  
 '''  
 function name: pullSongIdAddressCoupleDict  
 input: N/A  
 output: N/A  
 operation: pulls the songIdAddressCoupleDict from memory  
 '''  
  
 def pullSongIdAddressCoupleDict(self):  
 self.songIdAddressCoupleDict = self.collection.find\_one({"\_id": 2})  
 assert self.songIdAddressCoupleDict, "===== ERROR === error at pullFingerprintDatabase === ERROR ===== "  
  
 """  
 function name: pushSongIdSongInfoDict.  
 input: N/A  
 output: N/A.  
 operation: saves the songIdSongInfoDict to memory.  
 """  
  
 def pushSongIdSongInfoDict(self):  
 self.collection.delete\_one({"\_id": 3})  
 try:  
 self.collection.insert\_one(self.songIdSongInfoDict)  
 except pymongo.errors.DuplicateKeyError:  
 print('===== ERROR === DuplicateKeyError for dict 3 === ERROR =====')  
 else:  
 print('===== changes to songIdSongInfoDict SAVED SUCCESSFULLY =====')  
  
 """  
 function name: pullSongIdSongInfoDict.  
 input: N/A  
 output: N/A.  
 operation: pulls the songIdSongInfoDict from memory.  
 """  
  
 def pullSongIdSongInfoDict(self):  
 self.songIdSongInfoDict = self.collection.find\_one({"\_id": 3})  
 assert self.songIdSongInfoDict, "===== ERROR === error at pullSongIdSongInfoDict === ERROR " \  
 "===== "  
  
 """  
 function name: pullAll.  
 input: N/A  
 output: N/A.  
 operation: pulls the fingerprintDatabase, SongIdAddressCoupleDict and songIdSongInfoDict from memory.  
 """  
  
 def pullAll(self):  
 self.pullFingerprintDatabase()  
 self.pullSongIdAddressCoupleDict()  
 self.pullSongIdSongInfoDict()  
  
 """  
 function name: pushAll.  
 input: N/A  
 output: N/A.  
 operation: saves the fingerprintDatabase, SongIdAddressCoupleDict and songIdSongInfoDict to memory.  
 """  
  
 def pushAll(self):  
 self.pushFingerprintDatabase()  
 self.pushSongIdAddressCoupleDict()  
 self.pushSongIdSongInfoDict()  
  
 '''  
 function name: databaseUpdateDECIMAL  
 input: a song object  
 output: N/A  
 operation: ==========a regular "decimal" method for checking and demonstration==========  
 creates the address for the point in the address object, load them in the "database" dictionary,  
 if a key already exist, adds the match couple to the value (a list) of the key, if not, creates a new key and a list  
 of one object, so it can be expanded in the future.  
 '''  
  
 def databaseUpdate(self, *song*):  
 *addressCoupleDict* = *song*.addressCoupleDict  
 for *key*, *value* in *addressCoupleDict*.items(): # the key is an address  
 if *key* in self.storageDatabase:  
 self.storageDatabase[*key*] += *value* else:  
 self.storageDatabase[*key*] = *value* '''  
 function name: searchInDatabase  
 input: the recoding list of addresses and couples  
 output: N/A  
 operation: if an address is in the database, the function updates the songIdTable.  
 '''  
  
 def searchInDatabase(self, *recording*):  
 self.pullAll()  
 for *address* in *recording*.addressAnchorTimeDict.keys():  
 if *address* in self.database.keys():  
 *recording*.songIdTableUpdate(self.database[*address*])  
  
 *recording*.songIdTableFilter()  
 return self.filterResults(list(*recording*.songIdNumOfKeysTable.keys()), *recording*)  
  
 """  
 function name: filterResults.  
 input: songIdList - self explnetory, recording - a Recording object, the recording in the search.  
 output: the information for the song we predict the recording matches to. (if we can make such prediction).  
 operation: now for each song id (the key) we'll find the delta that appears most in that list. we'll save the time   
 of appearances, and the song with the most appearances we'll be our prediction.   
 """  
  
 def filterResults(self, *songIdList*, *recording*):  
 if len(*songIdList*) == 0:  
 return "didn't find anything :("  
 elif len(*songIdList*) == 1:  
 return self.songIdSongInfoDict[*songIdList*[0]]  
  
 # initialize the songIdDeltaDict for future use.  
 *recording*.songIdDeltaDict = {*songId*: [] for *songId* in *songIdList*}  
  
 for *songId* in *songIdList*:  
 for *address*, *anchorTimeList* in *recording*.addressAnchorTimeDict.items():  
 for *anchorTime* in *anchorTimeList*:  
  
 *tempAddressCoupleDict* = self.songIdAddressCoupleDict[*songId*]  
 *stringAddress* = addressToString(*address*)  
 if *stringAddress* in *tempAddressCoupleDict*.keys():  
 *deltaList* = [abs(*anchorTime* - *couple*[0]) for *couple* in *tempAddressCoupleDict*[*stringAddress*]]  
 *recording*.songIdDeltaDict[*songId*] += *deltaList* # finds the delta that appear the most in the list, set the number of appearance to be the new value of the dict  
 *temp* = {*key*: max(*listOfDeltas*, key=*listOfDeltas*.count) for *key*, *listOfDeltas* in  
 *recording*.songIdDeltaDict.items()}  
  
 *recording*.songIdDeltaDict = {*key*: *listOfDeltas*.count(*temp*[*key*]) for *key*, *listOfDeltas* in  
 *recording*.songIdDeltaDict.items()}  
 *prediction* = self.songIdSongInfoDict[max(*recording*.songIdDeltaDict.items(), key=lambda *x*: *x*[1])[0]]  
 return *prediction* """  
 function name: showCollection.  
 input: N/A  
 output: N/A  
 operation: prints the 2 dictionaries in the collection.  
 """  
  
 def showCollection(self):  
 for *x* in self.collection.find():  
 print(*x*)  
 print(len(*x*))  
  
 """  
 function name: createNewDatabase.  
 input: N/A  
 output: N/A  
 operation: deletes the old database (the mongoDB collection) and inserts new dictionaries for the future database.  
 """  
  
 def createNewDatabase(self):  
 self.collection.drop()  
 self.collection.insert\_one({"\_id": 1})  
 self.collection.insert\_one({"\_id": 2})  
 self.collection.insert\_one({"\_id": 3})

## LoadToDatabase.py

*"""  
program for loading songs into the database.  
"""*from DatabaseItems.FingerprintDatabase import FingerprintDatabase  
  
*Database* = FingerprintDatabase()  
  
*songIdSongInfoDict* = {}  
*songIdPathDict* = {}  
  
print("to stop the loop enter 0 for any of the input boxes\n")  
*loopCondition* = 1  
while *loopCondition*:  
 *path* = input("enter the path to the wav file\n")  
 if *path* == '0':  
 break  
  
 *songId* = input("enter the song Id\n")  
 if *songId* == '0':  
 break  
  
 *songInfo* = input("enter the song Information, this will be shown to the user as the song metadata\n")  
 if *songInfo* == '0':  
 break  
  
 *songIdPathDict*[*songId*] = *path  
 songIdSongInfoDict*[*songId*] = *songInfo  
  
Database*.loadMany(*songIdPathDict*, *songIdSongInfoDict*)

## Recording.py

from ImportsFile import \*  
  
*decimalPoints* = GlobalValues.decimalPoints  
*coefficient* = GlobalValues.songIdFilterCoefficient  
  
  
class Recording:  
 def \_\_init\_\_(self, *path*):  
 self.timeFrequencyPoints = []  
  
 # a list of lists of points.  
 self.targetZones = []  
  
 # key - anchor point, value - target zone  
 self.anchorPointTargetZoneDict = {}  
  
 # self.addressCouplesList = []  
  
 self.addressAnchorTimeDict = {}  
  
 # key - couple, value - the number of time it appears in the recording address.  
 self.songIdTable = {}  
  
 # key - songId, value - the number of time is part of a key in the songIdTable  
 self.songIdNumOfKeysTable = {}  
  
 # key - songId, value - a list of deltas of the anchor time for each songId.  
 self.songIdDeltaDict = {}  
  
 # the path to the wav file of that recording.  
 self.dataPath = *path* """  
 function name: initializeAll.  
 input: N/A  
 output: N/A  
 operation: initializes the timeFrequencyPoints, targetZones, anchorPointTargetZoneDict, addressAnchorTimeDict   
 variables. also initializes the songIdDeltaDict to be an empty dictionary with lists as values for future use.  
 """  
  
 def initializeAll(self):  
 self.createConstellationMap()  
 self.targetZones = createTargetZones(self.timeFrequencyPoints)  
 self.anchorPointTargetZoneDict = createAnchorPoints(self.timeFrequencyPoints, self.targetZones)  
 self.createAddresses()  
  
 """  
 function name: createConstellationMap.  
 input: N/A  
 output: N/A  
 operation: initializes the timeFrequencyPoints variable.  
 """  
  
 def createConstellationMap(self):  
 try:  
 *sampleRate*, *data* = scipy.io.wavfile.read(self.dataPath)  
 self.timeFrequencyPoints = createFilteredSpectrogramPoints(list(*data*))  
 except:  
 print("error in reading the wav file...")  
  
 """  
 function name: createAddresses   
 input: N/A   
 output: N/A   
 operation: initializes the addressAnchorTimeDict,   
 for every point in a target zone, calculates the address according to the anchor point. rounds the number to 3   
 decimal points and saves the result as string for the search in the database.   
 """  
  
 def createAddresses(self):  
 for *anchorPoint*, *targetZone* in self.anchorPointTargetZoneDict.items():  
 *timeOfAnchor* = round(*anchorPoint*[0], *decimalPoints*)  
 for *p* in *targetZone*:  
 *delta* = *p*[0] - *anchorPoint*[0]  
 *tempAddress* = (*anchorPoint*[1], *p*[1], int(round(*delta*, *decimalPoints*) \* 10))  
  
 if *tempAddress* in self.addressAnchorTimeDict:  
 self.addressAnchorTimeDict[*tempAddress*].append(*timeOfAnchor*)  
 else:  
 self.addressAnchorTimeDict[*tempAddress*] = [*timeOfAnchor*]  
  
 '''  
 function name: songIdTableUpdate  
 input: the value of the database - a list of couples  
 output: N/A  
 operation: goes through the list, counting the number of couples the song have in common with the address value.  
 '''  
  
 def songIdTableUpdate(self, *List*):  
 for *couple* in *List*:  
 if *couple* in self.songIdTable:  
 self.songIdTable[*couple*] += 1  
 else:  
 self.songIdTable[*couple*] = 1  
  
 '''  
 function name: songIdTableFilter   
 input: N/A   
 output: N/A   
 operation: we go through the songIdTable, removing   
 all the couples who's value is less than 2, than we go through the   
 songIdTable again, counting the number of time each songId appeared in a key (a couple) of the hash table (  
 putting it in a new hash table). we remove the song whose number is below 20\*coefficient, because 300 is the   
 number of target zones in the recording.   
 '''  
  
 def songIdTableFilter(self):  
  
 if self.songIdTable:  
 self.songIdTable = dict(filter(lambda *element*: *element*[1] > 1, self.songIdTable.items()))  
  
 if not self.songIdTable:  
 return  
  
 for *couple* in self.songIdTable.keys():  
 if *couple*[1] in self.songIdNumOfKeysTable.keys():  
 self.songIdNumOfKeysTable[*couple*[1]] += 1  
 else:  
 self.songIdNumOfKeysTable[*couple*[1]] = 1  
  
 self.songIdNumOfKeysTable = dict(  
 filter(lambda *element*: *element*[1] >= 20 \* *coefficient*, self.songIdNumOfKeysTable.items()))

## Song.py

from ImportsFile import \*  
  
*decimalPoints* = GlobalValues.decimalPoints  
  
  
class Song:  
  
 def \_\_init\_\_(self, *path*, *songID*):  
 self.path = *path* self.songID = *songID* # a list of 2-item-tuples  
 self.timeFrequencyPoints = []  
 # list of lists of points.  
 self.targetZones = []  
 # key - anchor point, value - target zone  
 self.anchorPointTargetZoneDict = {}  
 # key - address, value - a list of couple associated with this address.  
 self.addressCoupleDict = {}  
  
 """  
 function name: initializeAll.  
 input: N/A  
 output: N/A  
 operation: initializes the timeFrequencyPoints, targetZones, anchorPointTargetZoneDict, addressCoupleDict   
 variables.   
 """  
  
 def initializeAll(self):  
 self.createConstellationMap()  
 self.targetZones = createTargetZones(self.timeFrequencyPoints)  
 self.anchorPointTargetZoneDict = createAnchorPoints(self.timeFrequencyPoints, self.targetZones)  
 self.createAddresses()  
  
 """  
 function name: createConstellationMap.  
 input: N/A  
 output: N/A  
 operation: initializes the timeFrequencyPoints variable. it reads the wave file associated with the song   
 instance and prepares the data for the fft.   
 """  
  
 def createConstellationMap(self):  
  
 try:  
 *sampleRate*, *data* = wavefile.read(self.path)  
 *sampleRate*, *data* = prepareForSpectrogram(*sampleRate*, *data*)  
 self.timeFrequencyPoints = createFilteredSpectrogramPoints(*data*)  
 except Exception as *wavFileException*:  
 print("error in reading the wav file for path:", self.path)  
 raise *wavFileException* """  
 function name: createAddresses   
 input: N/A   
 output: N/A   
 operation: initializes the addressAnchorTimeDict,   
 for every point in a target zone, calculates the address according to the anchor point. rounds the number to 3   
 decimal points and saves the result as a String for storage in the database.   
 """  
  
 def createAddresses(self):  
 for *anchorPoint*, *targetZone* in self.anchorPointTargetZoneDict.items():  
 *couple* = (round(*anchorPoint*[0], *decimalPoints*), self.songID)  
 for *p* in *targetZone*:  
 *delta* = *p*[0] - *anchorPoint*[0]  
 *tempAddress* = str(int(*anchorPoint*[1])) + ',' + str(int(*p*[1])) + ',' + str(  
 int(round(*delta*, *decimalPoints*) \* 10))  
  
 if *tempAddress* in self.addressCoupleDict:  
 self.addressCoupleDict[*tempAddress*].append(*couple*)  
 else:  
 self.addressCoupleDict[*tempAddress*] = [*couple*]

## TargetZone.py

import GlobalValues  
  
"""  
function name: targetZonesCreate.  
input: a list of time-frequency points.  
output: a list of lists.  
operation: creates a list of 5 consecutive points, until it runs out of points.   
adds that list to the "targetZone" list.  
"""  
  
  
def createTargetZones(*timeFrequencyPoints*):  
 *targetSize* = GlobalValues.targetSize  
 *targetZones* = []  
 for *i* in range(len(*timeFrequencyPoints*) - *targetSize* + 1):  
 *targetZones*.append([*timeFrequencyPoints*[*i* + *j*] for *j* in range(*targetSize*)])  
  
 return *targetZones*

## FFTPreparationFunctions.py

from ImportsFile import \*  
import numpy  
import scipy  
import scipy.signal  
from SignalProcessing.SpectrogramFilteringFunctions import \*  
  
'''  
 function mame: stereoToMonoConvert  
 output: N/A  
 input: N/A  
 operation: for every point takes the average of left speaker and the right speaker.   
'''  
  
  
def stereoToMonoConvert(*data*):  
 # instead of int16 type. takes care of the overflow that occurred.  
 *data* = np.array(*data*, dtype=numpy.float64)  
 return list(map(lambda *tup*: (*tup*[0] + *tup*[1]) / 2, *data*))  
  
  
'''  
 function name: lowPassFilter  
 input: N/A  
 output:N/A  
 operation: uses butterWorth filter, cutoff frequency of 5kHz,   
'''  
  
  
def lowPassFilter(*data*, *sampleRate*):  
 *order* = 2  
 *cutOffFrequency* = GlobalValues.lowPassFilterFrequency  
 *nyq* = 0.5 \* *sampleRate* # Nyquist Frequency  
 *normal\_cutoff* = *cutOffFrequency* / *nyq* # Get the filter coefficients  
 *b*, *a* = scipy.signal.butter(*order*, *normal\_cutoff*, btype='low', analog=False)  
 return scipy.signal.filtfilt(*b*, *a*, *data*)  
  
  
'''  
function name: hammingWindow.  
input: data, list of samples from a wav file.  
output: result, list of sub-lists, 1024 in length (if possible).  
operation: divides the data into 1024 sublist, multiplies with the window function, return the result in a list, takes   
care of the reminders.  
  
'''  
  
  
def hammingWindow(*data*):  
 *windowSize* = GlobalValues.hammingWindowSize  
 *window* = np.hamming(*windowSize*)  
 *length* = len(*data*)  
 *subLists* = [*data*[*i*: *i* + *windowSize*] for *i* in range(0, *length* - *length* % *windowSize*, *windowSize*)]  
 return list(map(lambda *subList*: [*subList*[*i*] \* *window*[*i*] for *i* in range(len(*subList*))], *subLists*))  
  
  
'''  
function name:   
downSample input:   
N/A output: N/A   
operation: for every 4 sample points we take the average of them, reducing our sample rate to 10.025kHz, a quarter of the original sampling frequency.  
takes care of reminders.  
'''  
  
  
def downSample(*data*):  
 *length* = len(*data*)  
 *temp* = list(map(lambda *subList*: sum(*subList*) / 4, [*data*[*i*:*i* + 4] for *i* in range(0, (*length* - *length* % 4), 4)]))  
  
 if not *length* % 4 == 0: *temp*.append(sum(*data*[-(*length* % 4):]) / (*length* % 4))  
  
 return *temp*"""  
function name: prepareForSpectrogram.  
input: sampleRate - the sampling rate of the input data. data - the audio file from wavfile.read. a numpy array.  
output: a tuple - sampleRate, data. sample rate is the input sample rate divided by 4 (after the down sampling). data is   
the processed data.  
operation: checks the data to be a mono or a stereo file. if it's a mono file it passes the data through the stereoToMonoConvert function. otherwise   
continues as usual with the low pass filter and the down sampling.  
"""  
  
  
def prepareForSpectrogram(*sampleRate*, *data*):  
 if len(*data*.shape) == 2:  
 print('preparing stereo file...')  
 *data* = stereoToMonoConvert(*data*)  
 else:  
 print('preparing mono file...')  
  
 *data* = lowPassFilter(*data*, *sampleRate*)  
 *data* = downSample(*data*)  
 *sampleRate* /= 4  
 return *sampleRate*, *data*

## SpectrogramFilteringFunctions.py

import GlobalValues  
from ImportsFile import \*  
import scipy.fftpack as *fftpk*from pylab import \*  
  
# frequencies bands ranges.  
*RANGE* = [102.315, 210.015, 425.415, 856.215, 1717.815, 5508.855]  
# the bands values.  
*bandsValue* = [9, 19, 39, 79, 159, 511]  
*windowSize* = 1024  
*sampleRate* = 11025  
  
*frequencies* = fftpk.fftfreq(*windowSize*, d=(1.0 / *sampleRate*))  
*frequencies* = *frequencies*[0:len(*frequencies*) // 2]  
  
"""  
function name: updateRange.  
input: N/A  
output: N/A  
operation: update the frequency RANGE variable.   
"""  
  
  
def updateRange():  
 global *RANGE  
 RANGE* = list(map(getNthBin, *bandsValue*))  
  
  
"""  
function name: getNthBin.  
input: n, a natural number.   
output: returns the nth bin upper limit with a bin size of 10.77Hz.  
operation: a recursive function. the first bin is (0,binSize/2) and the following bins are the intervals where the bin   
Size is added to a and b [(a,b)].   
"""  
  
  
def getNthBin(*n*):  
 *binSize* = 10.77  
 if 0 == *n*:  
 return *binSize* / 2  
 else:  
 return getNthBin(*n* - 1) + *binSize*"""  
function name: pointIntoBand.  
input: bands - a dictionary of lists. (key - the number of band, value - a list of points with frequency in that band)  
 point - a frequency-amplitude point.  
output: N/A  
operation: goes with a loop until the point is no longer in that band and stops.  
"""  
  
  
def pointIntoBand(*bands*, *point*):  
 *i* = 0  
 *length* = len(*bands*)  
 while *RANGE*[*i*] <= *frequencies*[*point*[0]] and *i* < *length* - 1:  
 *i* += 1  
 *bands*[*i*].append(*point*)  
  
  
"""  
function name: FFTPointsIntoBands.  
input: frequencyAmplitudePoints - a list of the frequency-amplitude points returned by the fft.  
output: bands - a dictionary (key - the number of band, value - a list of points with frequency in that band).   
operation: loops through the input list and calls the pointIntoBand function.  
"""  
  
  
def FFTPointsIntoBands(*frequencyAmplitudePoints*):  
 *bands* = {*i*: [] for *i* in range(6)}  
 for *p* in *frequencyAmplitudePoints*:  
 pointIntoBand(*bands*, *p*)  
  
 return *bands*'''  
function name: maxOfBand  
input: the dictionary of points after the distribution to bands.  
output: the same dictionary, with only the bigger amplitude points left. the strongest point of each band.  
operation: uses the max function with the key set to the amplitude (the second value of the pair)  
'''  
  
  
def maxOfBand(*bands*):  
 for *key* in *bands*:  
 try:  
 *bands*[*key*] = max(*bands*[*key*], key=lambda *x*: *x*[1]) # amplitude.  
 except:  
 print(str(*key*) + "is empty")  
 return *bands*'''  
function name: averageValueOfBins  
input: the dictionary of strongest point of each band  
output: the average value of the frequency of these bins  
operation: average of all the value in the dictionary.  
'''  
  
  
def averageValueOfBins(*bands*):  
 try:  
 return sum([*v*[1] for *v* in *bands*.values()]) / len(*bands*) # amplitude.  
 except Exception:  
 print("error in summing the values in averageValueOfBins")  
 exit(1)  
  
'''  
function name: passAboveTheMean.  
input: the bans dictionary as seen before.  
output: a list of the points whose amplitude is above the averageValue of the max of each band, multiplied by a coefficient.  
operation: self explanatory.  
'''  
  
  
def passAboveTheMean(*bands*):  
 *averageValue* = averageValueOfBins(*bands*)  
 *meanCoefficient* = GlobalValues.meanCoefficient  
  
 return [*v* for *v* in *bands*.values() if *v*[1] >= *averageValue* \* *meanCoefficient*]  
  
  
'''  
function name: filterFFT.  
input: the time-frequency points were getting from the fft.  
output: the peak points of that fft.  
operation: puts the points in the bands, keeps only the max of these bands and pass the points that are above the mean..  
'''  
  
  
def filterFFT(*frequencyAmplitudePoints*):  
 *bands* = FFTPointsIntoBands(*frequencyAmplitudePoints*)  
 *bands* = maxOfBand(*bands*)  
  
 return passAboveTheMean(*bands*)

## SpectrogramFunctions.py

from ImportsFile import \*  
import scipy  
import scipy.signal  
from pylab import \*  
from GeneralFunctions import \*  
  
'''  
function name: FFT.  
input: data, sampleRate.  
output: a tuple, (frequency, amplitudes)  
operation: uses the scipy built-in function, cuts the result in half because of the symmetry, takes care of the  
amplitude units.  
'''  
  
  
def FFT(*data*):  
 *amplitude* = abs(scipy.fft.fft(*data*))  
 *amplitude* = 2 \* *amplitude*[0:len(*amplitude*) // 2]  
 *amplitude* = *amplitude* / len(*data*)  
  
 return *amplitude*'''  
function name: createFilteredSpectrogramPoints.  
input: data - the audio data after been pass through the preparation stages.   
output: a list of time - frequency PEAK points. (the peaks are chosen according to their amplitude and by the filtering algorithm.  
operation: calculates the fft for every 1024 points (after going through the hamming window function) in the input data. and passes the result   
through the filtering algorithm.   
'''  
  
  
def createFilteredSpectrogramPoints(*data*):  
 *spectrogramPoints* = []  
 *result* = hammingWindow(*data*)  
  
 for *i* in range(len(*result*)):  
 *amp* = FFT(*result*[*i*])  
 *frequencyAmplitudePoints* = createPoints([\*range(0, 512)], *amp*)  
 *filteredPoints* = filterFFT(*frequencyAmplitudePoints*)  
 for *f* in *filteredPoints*:  
 *spectrogramPoints*.append((*i* / 10, round(*f*[0], 3)))  
  
 *spectrogramPoints* = sorted(*spectrogramPoints*, key=lambda *x*: *x*[0]) # filter the points by time.  
 return *spectrogramPoints*

## GeneralFunctions.py

*"""  
function name: createPoints.  
input: x - a list, y - a list. 2 list of the same object and size.  
output: a list of tuples.  
operation: N/A  
"""*def createPoints(*x*, *y*):  
 return [(*x*[*i*], *y*[*i*]) for *i* in range(0, len(*x*))]  
  
  
"""  
function name: createLists.  
input: points - a list of tuples.  
output: 2 lists. a list of the first item in the tuples and a list of the second item in the tuples.  
operation: N/A.  
"""  
  
  
def createLists(*points*):  
 return [*p*[0] for *p* in *points*], [*p*[1] for *p* in *points*]

## GlobalValues.py

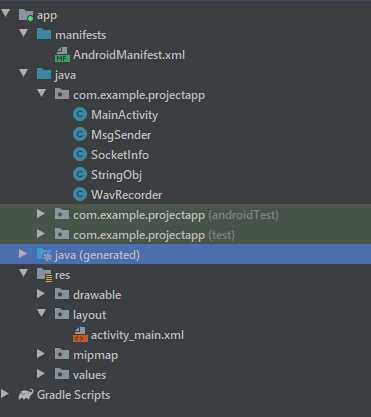
global *decimalPoints*, *songIdFilterCoefficient*, *meanCoefficient*, *gs*, *targetSize*, *hammingWindowSize  
  
hammingWindowSize* = 1024  
*lowPassFilterFrequency* = 5000  
*decimalPoints* = 3  
*songIdFilterCoefficient* = 1  
*meanCoefficient* = 1  
  
# targetZone size.  
*targetSize* = 5

## ImportsFile.py

import GlobalValues  
from DatabaseItems.DatabaseItemsFunctions import \*  
from scipy.io.wavfile import \*  
  
from DatabaseItems.TargetZone import createTargetZones  
from SignalProcessing import \*  
from SignalProcessing.FFTPreparationFunctions import \*  
from SignalProcessing.SpectrogramFunctions import \*  
from SignalProcessing.SpectrogramFilteringFunctions import \*  
  
import scipy.signal  
from scipy.io import wavfile  
  
import GlobalValues  
  
import numpy  
import scipy  
import scipy.fftpack as *fftpk*import scipy.signal  
from pylab import \*  
  
import GlobalValues  
from scipy import signal  
import scipy.io.wavfile as *wavefile*from GeneralFunctions import \*  
  
from scipy import signal  
import numpy as *np*import scipy.io.wavfile as *wavefile*import pymongo

a file to keep all the imports in. it didn’t work as I would’ve hoped so but its ok.

# מדריך למפתח – קבצי ג'אווה/אנדרואיד



קבצים ותיקיות:

## AndroidManifest.xml

<?xml version="1.0" encoding="utf-8"?>  
<manifest xmlns:android="http://schemas.android.com/apk/res/android"  
 package="com.example.projectapp">  
  
 <uses-permission android:name="android.permission.RECORD\_AUDIO" />  
 <uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE" />  
 <uses-permission android:name="android.permission.STORAGE" />  
  
 <uses-permission android:name="android.permission.INTERNET"/>  
 <uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE"/>  
 <uses-permission android:name="android.permission.ACCESS\_WIFI\_STATE"/>  
  
 <application  
 android:allowBackup="true"  
 android:icon="@mipmap/ic\_launcher"  
 android:label="@string/app\_name"  
 android:roundIcon="@mipmap/ic\_launcher\_round"  
 android:supportsRtl="true"  
 android:theme="@style/AppTheme">  
 <activity android:name=".MainActivity">  
 <intent-filter>  
 <action android:name="android.intent.action.MAIN" />  
  
 <category android:name="android.intent.category.LAUNCHER" />  
 </intent-filter>  
 </activity>  
 </application>  
  
</manifest>

## acitivity\_main.xml

<?xml version="1.0" encoding="utf-8"?>  
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:gravity="center"  
 android:orientation="vertical"  
 tools:context=".MainActivity">  
  
 <TextView  
 android:id="@+id/ipInputText"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:width="350dp"  
 android:text="please enter the IP address and the Port of the network you're connected to."  
 android:textSize="19sp" />  
  
  
 <EditText  
 android:id="@+id/ipInput"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:width="150dp"  
 android:digits="0123456789."  
 android:hint="IP"  
 android:imeOptions="actionNext"  
 android:inputType="numberDecimal"  
 android:maxLength="15" />  
  
 <EditText  
 android:id="@+id/portInput"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_marginBottom="30dp"  
 android:width="70dp"  
 android:digits="0123456789"  
 android:hint="Port"  
 android:imeOptions="actionDone"  
 android:inputType="numberDecimal"  
 android:maxLength="5" />  
  
 <TextView  
 android:id="@+id/text"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:width="350dp"  
 android:text="record between 10-15 seconds for the most accurate result.."  
 android:textSize="20sp" />  
  
  
 <Chronometer  
 android:id="@+id/chronometer"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_marginTop="40dp"  
 android:layout\_marginBottom="100dp"  
 android:textSize="30sp" />  
  
 <TextView  
 android:id="@+id/text2"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_marginBottom="30dp"  
 android:width="350dp"  
 android:text="The song name will appear on the button of the screen.."  
 android:textSize="20sp" />  
  
 <Button  
 android:id="@+id/btnRecord"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="Start Recording" />  
  
 <Button  
 android:id="@+id/btnStop"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="Stop Recording" />  
  
</LinearLayout>

## MainActivity.java

package com.example.projectapp;  
  
import android.Manifest;  
import android.content.Context;  
import android.content.pm.PackageManager;  
import android.media.MediaRecorder;  
import android.os.Bundle;  
import android.os.Environment;  
import android.os.StatFs;  
import android.os.SystemClock;  
import android.text.Editable;  
import android.text.TextWatcher;  
import android.util.Log;  
import android.view.KeyEvent;  
import android.view.View;  
import android.view.inputmethod.EditorInfo;  
import android.widget.Button;  
import android.widget.Chronometer;  
import android.widget.EditText;  
import android.widget.TextView;  
import android.widget.Toast;  
  
import androidx.appcompat.app.AppCompatActivity;  
import androidx.core.app.ActivityCompat;  
import androidx.core.content.ContextCompat;  
  
import java.io.File;  
import java.util.Random;  
import java.util.concurrent.ExecutionException;  
import java.util.concurrent.TimeUnit;  
import java.util.concurrent.TimeoutException;  
  
import static android.Manifest.permission.*RECORD\_AUDIO*;  
import static android.Manifest.permission.*WRITE\_EXTERNAL\_STORAGE*;  
  
public class MainActivity extends AppCompatActivity {  
 private Chronometer chronometer;  
 private boolean running; // is the chronometer running.  
 private EditText ipInput, portInput;  
 Button buttonStart, buttonStop;  
 MediaRecorder mediaRecorder;  
 WavRecorder wavRecorder;  
 Random random;  
 MsgSender MsgSender;  
 StringObj str;  
 private SocketInfo socketInfo = new SocketInfo("", -1);  
 private boolean isValidIP = false;  
  
  
 public static final int *RequestPermissionCode* = 1;  
  
 */\*\*  
 \* the 'run' function.  
 \*/* @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_main*);  
  
 buttonStart = findViewById(R.id.*btnRecord*);  
 buttonStop = findViewById(R.id.*btnStop*);  
 chronometer = findViewById(R.id.*chronometer*);  
 ipInput = findViewById(R.id.*ipInput*);  
 portInput = findViewById(R.id.*portInput*);  
  
 random = new Random();  
  
 ipInput.setOnEditorActionListener(editorActionListener);  
 portInput.setOnEditorActionListener(editorActionListener);  
  
 buttonStart.setEnabled(false);  
 buttonStop.setEnabled(false);  
  
 buttonStart.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 if (checkPermission()) {  
 MediaRecorderReady();  
  
 wavRecorder.startRecording();  
 buttonStart.setEnabled(false);  
 buttonStop.setEnabled(true);  
 startChronometer(view);  
  
 Toast.*makeText*(MainActivity.this, "Recording started",  
 Toast.*LENGTH\_LONG*).show();  
 } else {  
 requestPermission();  
 }  
 }  
 });  
  
 buttonStop.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 wavRecorder.stopRecording();  
 buttonStop.setEnabled(false);  
 buttonStart.setEnabled(true);  
 pauseChronometer(view);  
 Toast.*makeText*(MainActivity.this, "Recording Completed",  
 Toast.*LENGTH\_LONG*).show();  
  
 send("file");  
 }  
 });  
 }  
  
 private TextView.OnEditorActionListener editorActionListener = new TextView.OnEditorActionListener() {  
 @Override  
 public boolean onEditorAction(TextView v, int actionId, KeyEvent event) {  
 switch (actionId) {  
 case EditorInfo.*IME\_ACTION\_NEXT*:  
 isIpValid();  
 break;  
 case EditorInfo.*IME\_ACTION\_DONE*:  
 isPortValid();  
 break;  
 }  
 return false;  
 }  
 };  
  
 */\*\*  
 \*  
 \*/* public void MediaRecorderReady() {  
 mediaRecorder = new MediaRecorder();  
 mediaRecorder.setAudioSource(MediaRecorder.AudioSource.*MIC*);  
  
 String outputFile = Environment.*getExternalStorageDirectory*().getAbsolutePath() + "/recording\_.wav";  
 if (getApplicationContext() != null &&  
 *hasPermission*(Manifest.permission.*WRITE\_EXTERNAL\_STORAGE*, getApplicationContext()) &&  
 *isExternalStorageWritable*() &&  
 *isExternalStorageReadable*() &&  
 *hasSufficientFreeSpaceAvailable*(getApplicationContext().getExternalFilesDir(Environment.*DIRECTORY\_MUSIC*).getPath())) {  
 outputFile = new File(getApplicationContext().getExternalFilesDir(Environment.*DIRECTORY\_MUSIC*), "recording\_.wav").getPath();  
 } else if (*hasSufficientFreeSpaceAvailable*(getApplicationContext().getFilesDir().getPath())) {  
 outputFile = new File(getApplicationContext().getExternalFilesDir(Environment.*DIRECTORY\_MUSIC*), "recording\_.wav").getPath();  
 }  
 wavRecorder = new WavRecorder(outputFile);  
 }  
  
 */\*\*  
 \* requests permission from the user to access the phone storage and record audio.  
 \*/* private void requestPermission() {  
 ActivityCompat.*requestPermissions*(MainActivity.this, new  
 String[]{*WRITE\_EXTERNAL\_STORAGE*, *RECORD\_AUDIO*}, *RequestPermissionCode*);  
 }  
  
 */\*\*  
 \* an android function as seen by the override, presents whether the permission was granted  
 \* with the Toast prints.  
 \*/* @Override  
 public void onRequestPermissionsResult(int requestCode,  
 String permissions[], int[] grantResults) {  
 switch (requestCode) {  
 case *RequestPermissionCode*:  
 if (grantResults.length > 0) {  
 boolean StoragePermission = grantResults[0] ==  
 PackageManager.*PERMISSION\_GRANTED*;  
 boolean RecordPermission = grantResults[1] ==  
 PackageManager.*PERMISSION\_GRANTED*;  
  
 if (StoragePermission && RecordPermission) {  
 Toast.*makeText*(MainActivity.this, "Permission Granted",  
 Toast.*LENGTH\_LONG*).show();  
 } else {  
 Toast.*makeText*(MainActivity.this, "Permission denied", Toast.*LENGTH\_LONG*).show();  
 }  
 }  
 break;  
 }  
 }  
  
 */\*\*  
 \** ***@return*** *if the access to the data is possible.  
 \*/* public static boolean isExternalStorageWritable() {  
 String state = Environment.*getExternalStorageState*();  
 return Environment.*MEDIA\_MOUNTED*.equals(state);  
 }  
  
 */\*\*  
 \** ***@return*** *if the permission to record audio is granted.  
 \*/* public static boolean isExternalStorageReadable() {  
 String state = Environment.*getExternalStorageState*();  
  
 return Environment.*MEDIA\_MOUNTED*.equals(state)  
 || Environment.*MEDIA\_MOUNTED\_READ\_ONLY*.equals(state);  
 }  
  
 */\*\*  
 \** ***@return*** *if we have permission to write to users calender data.  
 \*/* public static boolean hasPermission(String permission, Context context) {  
 if (ContextCompat.*checkSelfPermission*(context, Manifest.permission.*WRITE\_CALENDAR*)  
 != PackageManager.*PERMISSION\_GRANTED*) {  
 return false;  
 }  
 return true;  
 }  
  
 */\*\*  
 \** ***@return*** *if we have enough free space to store our temporary wav file.  
 \*/* public static boolean hasSufficientFreeSpaceAvailable(String path) {  
 StatFs stat = new StatFs(path);  
 long bytesAvailable = (long) stat.getBlockSize() \* (long) stat.getBlockCount();  
 long megAvailable = bytesAvailable / 1048576;  
 if (megAvailable > 2) {  
 return true;  
 }  
 return false;  
 }  
  
 */\*\*  
 \** ***@return*** *if we have permission to write to the data and record audio.  
 \*/* public boolean checkPermission() {  
 int result = ContextCompat.*checkSelfPermission*(getApplicationContext(),  
 *WRITE\_EXTERNAL\_STORAGE*);  
 int result1 = ContextCompat.*checkSelfPermission*(getApplicationContext(),  
 *RECORD\_AUDIO*);  
 return result == PackageManager.*PERMISSION\_GRANTED* &&  
 result1 == PackageManager.*PERMISSION\_GRANTED*;  
 }  
  
 */\*\*  
 \* sends the message to the server followed by the bytes of the wav file.  
 \*  
 \** ***@param*** *val  
 \*/* public void send(String val) {  
 String ans = val;  
 str = new StringObj(ans);  
 MsgSender = new MsgSender(str, socketInfo);  
 try {  
 MsgSender.execute(ans).get(10000, TimeUnit.*MILLISECONDS*);  
 } catch (ExecutionException e) {  
 e.printStackTrace();  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 } catch (TimeoutException e) {  
 Toast.*makeText*(MainActivity.this, "something is wrong with the network, " +  
 "try a different IP address or a different port...",  
 Toast.*LENGTH\_LONG*).show();  
 }  
 if (!str.getStr().equals("file")) {  
 Toast.*makeText*(MainActivity.this, str.getStr(), Toast.*LENGTH\_LONG*).show();  
 }  
 }  
  
 */\*\*  
 \* start the chronometer, our 'stopwatch'  
 \*/* public void startChronometer(View v) {  
 if (!running) {  
 chronometer.setBase(SystemClock.*elapsedRealtime*());  
 chronometer.start();  
 running = true;  
 }  
 }  
  
 */\*\*  
 \* stops and resets the chronometer, our 'stopwatch'  
 \*/* public void pauseChronometer(View v) {  
 if (running) {  
 chronometer.setBase(SystemClock.*elapsedRealtime*());  
 chronometer.stop();  
 running = false;  
 }  
 }  
  
 */\*\*  
 \* checks if the input IP is valid.  
 \*/* public void isIpValid() {  
 String ipText = ipInput.getText().toString();  
 String[] ipNumbers = ipText.split("\\.");  
 if (ipNumbers.length != 4) {  
 Toast.*makeText*(MainActivity.this, "invalid ip", Toast.*LENGTH\_LONG*).show();  
 return;  
 }  
  
 for (String s : ipNumbers) {  
 try {  
 int temp = Integer.*parseInt*(s);  
 if (temp >= 256) {  
 Toast.*makeText*(MainActivity.this, "invalid ip ", Toast.*LENGTH\_LONG*).show();  
 return;  
 }  
 } catch (Exception e) {  
 Toast.*makeText*(MainActivity.this, "invalid ip ", Toast.*LENGTH\_LONG*).show();  
 return;  
 }  
  
 }  
 Toast.*makeText*(MainActivity.this, "ip is valid",  
 Toast.*LENGTH\_LONG*).show();  
  
 socketInfo.setIp(ipText);  
 isValidIP = true;  
 return;  
 }  
  
 */\*\*  
 \* checks if the input Port is valid.  
 \*/* public void isPortValid() {  
 if (!isValidIP) {  
 Toast.*makeText*(MainActivity.this, "enter valid ip first",  
 Toast.*LENGTH\_LONG*).show();  
 return;  
 }  
 try {  
 int port = Integer.*parseInt*(portInput.getText().toString());  
 if (port > 65535) {  
 Toast.*makeText*(MainActivity.this, "invalid port",  
 Toast.*LENGTH\_LONG*).show();  
 return;  
 }  
 Toast.*makeText*(MainActivity.this, "everything is ok, connecting to the server",  
 Toast.*LENGTH\_LONG*).show();  
 socketInfo.setPort(port);  
 portInput.setText("");  
 ipInput.setText("");  
 buttonStart.setEnabled(true);  
 buttonStop.setEnabled(false);  
  
 } catch (Exception e) {  
 Toast.*makeText*(MainActivity.this, "invalid port",  
 Toast.*LENGTH\_LONG*).show();  
 return;  
 }  
 }  
  
  
}

## MsgSender.java

package com.example.projectapp;  
  
  
import android.os.AsyncTask;  
import android.os.Build;  
  
import androidx.annotation.RequiresApi;  
  
import java.io.BufferedReader;  
import java.io.File;  
import java.io.IOException;  
import java.io.InputStream;  
import java.io.InputStreamReader;  
import java.io.OutputStream;  
import java.io.PrintWriter;  
import java.net.Socket;  
import java.nio.file.Files;  
  
public class MsgSender extends AsyncTask<String, SocketInfo, Void> {  
 Socket s;  
 SocketInfo socketInfo;  
 PrintWriter pw;  
 InputStreamReader isr;  
 BufferedReader bf;  
 StringObj str;  
  
 */\*\*  
 \* constructor.  
 \*/* public MsgSender(StringObj string, SocketInfo sI) {  
 this.str = string;  
 this.socketInfo = sI;  
 }  
  
 */\*\*  
 \* does the work for sending the information to the server and splitting down the wav file.  
 \*/* @RequiresApi(api = Build.VERSION\_CODES.*O*)  
 @Override  
 protected Void doInBackground(String... voids) {  
 String message = voids[0];  
 try {  
  
 try {  
 s = new Socket(socketInfo.getIp(), socketInfo.getPort());  
 } catch (Exception e) {  
  
 }  
 OutputStream output = s.getOutputStream();  
 pw = new PrintWriter(output);  
  
  
 if (str.getStr().startsWith("file")) {  
 InputStream input = null;  
 input = Files.*newInputStream*(new File("/storage/emulated/0/Android/data/com.example.projectapp/files/Music/recording\_.wav").toPath());  
  
 //send file  
 pw.write(message);  
 pw.flush();  
 byte[] buffer = new byte[4096];  
  
 for (  
 int bytesRead = input.read(buffer);  
 bytesRead != -1;  
 bytesRead = input.read(buffer)) {  
 output.write(buffer, 0, bytesRead);  
 output.flush();  
 }  
 pw.write("finish");  
 pw.flush();  
 }  
  
  
 isr = new InputStreamReader(s.getInputStream());  
 bf = new BufferedReader(isr);  
 voids[0] = bf.readLine();  
  
 this.str.setStr(voids[0]);  
  
 } catch (IOException ioe) {  
 ioe.printStackTrace();  
 }  
 return null;  
  
  
 }  
  
  
}

## SocketInfo.java

package com.example.projectapp;  
  
public class SocketInfo {  
 private String ip;  
 private int port;  
  
 public SocketInfo(String ip, int port) {  
 this.ip = ip;  
 this.port = port;  
 }  
  
 public String getIp() {  
 return ip;  
 }  
  
 public void setIp(String ip) {  
 this.ip = ip;  
 }  
  
 public int getPort() {  
 return port;  
 }  
  
 public void setPort(int port) {  
 this.port = port;  
 }  
}

## WavRecorder.java

package com.example.projectapp;  
  
  
import java.io.File;  
import java.io.FileInputStream;  
import java.io.FileNotFoundException;  
import java.io.FileOutputStream;  
import java.io.IOException;  
  
import android.media.AudioFormat;  
import android.media.AudioRecord;  
import android.media.MediaRecorder;  
import android.os.Environment;  
  
public class WavRecorder {  
 private static final int *RECORDER\_BPP* = 16;  
 private static final String *AUDIO\_RECORDER\_FOLDER* = "AudioRecorder";  
 private static final String *AUDIO\_RECORDER\_TEMP\_FILE* = "record\_temp.raw";  
 private static final int *RECORDER\_SAMPLERATE* = 11025;  
 private static final int *RECORDER\_CHANNELS* = AudioFormat.*CHANNEL\_IN\_MONO*;  
 private static final int *RECORDER\_AUDIO\_ENCODING* = AudioFormat.*ENCODING\_PCM\_16BIT*;  
 short[] audioData;  
  
 private AudioRecord recorder = null;  
 private int bufferSize = 0;  
 private Thread recordingThread = null;  
 private boolean isRecording = false;  
 int[] bufferData;  
 int bytesRecorded;  
  
 private String output;  
  
 public WavRecorder(String path) {  
 bufferSize = AudioRecord.*getMinBufferSize*(*RECORDER\_SAMPLERATE*,  
 *RECORDER\_CHANNELS*, *RECORDER\_AUDIO\_ENCODING*) \* 3;  
  
 audioData = new short[bufferSize]; // short array that pcm data is put  
 // into.  
 output = path;  
  
 }  
  
 private String getFilename() {  
 return (output);  
 }  
  
 private String getTempFilename() {  
 String filepath = Environment.*getExternalStorageDirectory*().getPath();  
 File file = new File(filepath, *AUDIO\_RECORDER\_FOLDER*);  
  
 if (!file.exists()) {  
 file.mkdirs();  
 }  
  
 File tempFile = new File(filepath, *AUDIO\_RECORDER\_TEMP\_FILE*);  
  
 if (tempFile.exists())  
 tempFile.delete();  
  
 return (file.getAbsolutePath() + "/" + *AUDIO\_RECORDER\_TEMP\_FILE*);  
 }  
  
 public void startRecording() {  
  
 recorder = new AudioRecord(MediaRecorder.AudioSource.*MIC*,  
 *RECORDER\_SAMPLERATE*, *RECORDER\_CHANNELS*,  
 *RECORDER\_AUDIO\_ENCODING*, bufferSize);  
 int i = recorder.getState();  
 if (i == 1)  
 recorder.startRecording();  
  
 isRecording = true;  
  
 recordingThread = new Thread(new Runnable() {  
 @Override  
 public void run() {  
 writeAudioDataToFile();  
 }  
 }, "AudioRecorder Thread");  
  
 recordingThread.start();  
 }  
  
 private void writeAudioDataToFile() {  
 byte data[] = new byte[bufferSize];  
 String filename = getTempFilename();  
 FileOutputStream os = null;  
  
 try {  
 os = new FileOutputStream(filename);  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 }  
  
 int read = 0;  
 if (null != os) {  
 while (isRecording) {  
 read = recorder.read(data, 0, bufferSize);  
 if (read > 0) {  
 }  
  
 if (AudioRecord.*ERROR\_INVALID\_OPERATION* != read) {  
 try {  
 os.write(data);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
  
 try {  
 os.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
  
 public void stopRecording() {  
 if (null != recorder) {  
 isRecording = false;  
  
 int i = recorder.getState();  
 if (i == 1)  
 recorder.stop();  
 recorder.release();  
  
 recorder = null;  
 recordingThread = null;  
 }  
  
 copyWaveFile(getTempFilename(), getFilename());  
 deleteTempFile();  
 }  
  
 private void deleteTempFile() {  
 File file = new File(getTempFilename());  
 file.delete();  
 }  
  
 private void copyWaveFile(String inFilename, String outFilename) {  
 FileInputStream in = null;  
 FileOutputStream out = null;  
 long totalAudioLen = 0;  
 long totalDataLen = totalAudioLen + 36;  
 long longSampleRate = *RECORDER\_SAMPLERATE*;  
 int channels = ((*RECORDER\_CHANNELS* == AudioFormat.*CHANNEL\_IN\_MONO*) ? 1  
 : 2);  
 long byteRate = *RECORDER\_BPP* \* *RECORDER\_SAMPLERATE* \* channels / 8;  
  
 byte[] data = new byte[bufferSize];  
  
 try {  
 in = new FileInputStream(inFilename);  
 out = new FileOutputStream(outFilename);  
 totalAudioLen = in.getChannel().size();  
 totalDataLen = totalAudioLen + 36;  
  
 WriteWaveFileHeader(out, totalAudioLen, totalDataLen,  
 longSampleRate, channels, byteRate);  
  
 while (in.read(data) != -1) {  
 out.write(data);  
 }  
  
 in.close();  
 out.close();  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 private void WriteWaveFileHeader(FileOutputStream out, long totalAudioLen,  
 long totalDataLen, long longSampleRate, int channels, long byteRate)  
 throws IOException {  
 byte[] header = new byte[44];  
  
 header[0] = 'R'; // RIFF/WAVE header  
 header[1] = 'I';  
 header[2] = 'F';  
 header[3] = 'F';  
 header[4] = (byte) (totalDataLen & 0xff);  
 header[5] = (byte) ((totalDataLen >> 8) & 0xff);  
 header[6] = (byte) ((totalDataLen >> 16) & 0xff);  
 header[7] = (byte) ((totalDataLen >> 24) & 0xff);  
 header[8] = 'W';  
 header[9] = 'A';  
 header[10] = 'V';  
 header[11] = 'E';  
 header[12] = 'f'; // 'fmt ' chunk  
 header[13] = 'm';  
 header[14] = 't';  
 header[15] = ' ';  
 header[16] = 16; // 4 bytes: size of 'fmt ' chunk  
 header[17] = 0;  
 header[18] = 0;  
 header[19] = 0;  
 header[20] = 1; // format = 1  
 header[21] = 0;  
 header[22] = (byte) channels;  
 header[23] = 0;  
 header[24] = (byte) (longSampleRate & 0xff);  
 header[25] = (byte) ((longSampleRate >> 8) & 0xff);  
 header[26] = (byte) ((longSampleRate >> 16) & 0xff);  
 header[27] = (byte) ((longSampleRate >> 24) & 0xff);  
 header[28] = (byte) (byteRate & 0xff);  
 header[29] = (byte) ((byteRate >> 8) & 0xff);  
 header[30] = (byte) ((byteRate >> 16) & 0xff);  
 header[31] = (byte) ((byteRate >> 24) & 0xff);  
 header[32] = (byte) (((*RECORDER\_CHANNELS* == AudioFormat.*CHANNEL\_IN\_MONO*) ? 1  
 : 2) \* 16 / 8); // block align  
 header[33] = 0;  
 header[34] = *RECORDER\_BPP*; // bits per sample  
 header[35] = 0;  
 header[36] = 'd';  
 header[37] = 'a';  
 header[38] = 't';  
 header[39] = 'a';  
 header[40] = (byte) (totalAudioLen & 0xff);  
 header[41] = (byte) ((totalAudioLen >> 8) & 0xff);  
 header[42] = (byte) ((totalAudioLen >> 16) & 0xff);  
 header[43] = (byte) ((totalAudioLen >> 24) & 0xff);  
  
 out.write(header, 0, 44);  
 }  
  
}

## StringObj.java

package com.example.projectapp;  
  
class StringObj {  
 private String str;  
  
 public StringObj(String string) {  
 this.str = string;  
 }  
  
 public String getStr() {  
 return str;  
 }  
  
 public void setStr(String string) {  
 this.str = string;  
 }  
}

# רפלקציה

* עבורי העבודה על הפרויקט הייתה מעשירה, מלחיצה, כיפית ומתסכלת מידי פעם. היא לקחה המון זמן ומאמץ. לנסות, לחפש, להיכשל ולהצליח מדי פעם כולם היו חלקים מרכזים בעבודה על הפרויקט.
* אני קיבלתי הרבה מהפרויקט, חוץ מהידע שלי על signal processing (שהוא נורא מוגבל חשוב להדגיש) והידע שלי בפייתון. הבנתי דברים חשובים על
  + תיעוד קוד
  + עבודה מסודרת ומתודית
  + חשיבות של תכנון עד הפרטים הכי קטנים ברמת שם משתנה.

עבור עבודה חלקה ויעילה.

* הייתי לוקח את הכלים שלמדתי על תכנון, הן של זמנים והן של התוכנה עצמה. כלים שהם כלים במובן המוחשי שעוזרים לך לייעל את העבודה כמו סביבת עבודה מותאמת אישית לדוגמא, הם דברים חשובים מאוד עבורי והם כן עזרו לי בעבודה.
* הקשיים שהתמודדתי איתם במהלך העבודה היו רמת החומר שאיתו הייתי צריך לעבוד, בלי יותר מדי ידע מקדים. גודל הפרויקט, חוסר מידע באינטרנט על נושאים מסוימים, לחץ לסיים בידיעה שיש לי פרויקט קשה וזמן מוגבל. שגרמו לי לפעול בצורה לא נכונה, ולהתרכז בדרכים הנכונים.
* המסקנות שלי מהפרויקט הם שתכנון הוא דבר חשוב מאוד, לבדוק יותר טוב מה אתה הולך לעשות לפני שאתה קופץ לעשות אותו, לבדוק אם זה באמת מה שאתה צריך ורוצה לעשות.
* אם הייתי מתחיל היום, קודם כל הייתי מתכנן מה אני צריך לעשות, טוב יותר. הייתי מבין עד רמת המימוש כל פרט ופרט של מה שאני צריך לעשות ולא פשוט לקפוץ פנימה ולנסות לכתוב קוד כמה שיותר מהר. הייתי מתכנן את המודלים, הפונקציות שלהם והמשתנים. לראות איך כל הפרויקט עובד כיחידה אחת ואז מתחיל לעבוד. בכנות, החלק הזה לא היה לוקח לי יותר מחודשיים-שלושה, כתיבת הקוד הייתה יכולה לקחת לי משהו כמו שבוע. תכנון ראשוני הוא דבר חשוב מאוד, אני מצטער ללמוד את זה בסוף הפרויקט ושמח שלמדתי את זה בכלל (better late than never).
* אני חושב שאם העבודה הייתה יותר יעילה עבורי הייתי מצליח לספק מוצר טוב יותר, שלם יותר ומרשים יותר...
* שאלת חקר עצמי היא: האם היה עדיף לי ללכת בכיוון של למידת מכונה ובינה מלאכותית (כי אני מניח שככה האלגוריתם שלהם עובד היום..)

# ביבליוגרפיה

המאמר הכי חשוב של הפרויקט הזה הוא המאמר הזה:

<http://coding-geek.com/how-shazam-works/>

עקבתי אחרי ההוראות שלו עד לפרט האחרון. אחרי שראיתי את הכלליות של המאמר האקדמאי ידעתי שלא אוכל להצליח לעקוב ולממש את הכתוב שם. לעומת זאת המאמר הזה היא ברור, מפורט, מובן ונוח לעבוד לפיו. בלעדיו לא הייתי מצליח.

מאמר אקדמאי:

<https://www.ee.columbia.edu/~dpwe/papers/Wang03-shazam.pdf>

MongoDB:

<https://www.w3schools.com/python/python_mongodb_getstarted.asp>

<https://www.geeksforgeeks.org/guide-install-mongodb-python-windows/>

fft:

<https://www.youtube.com/watch?v=mkGsMWi_j4Q>

<https://docs.scipy.org/doc/scipy/reference/generated/scipy.fft.fft.html>

<https://numpy.org/doc/stable/reference/generated/numpy.fft.fftfreq.html#numpy.fft.fftfreq>

hamming window function:

<https://numpy.org/doc/stable/reference/generated/numpy.hamming.html>

low pass filter:

<https://medium.com/analytics-vidhya/how-to-filter-noise-with-a-low-pass-filter-python-885223e5e9b7>

זה נראה שלא השתמשתי בהרבה מידע.. **ההפך הוא הנכון** אני בטוח שבזמן העבודה על הפרויקט נכנסתי וקראתי ממאות מקורות מידע שמהם אולי היה מספיק רק שורה אחת או שהם לא הועילו לי. כמו כן התחלתי לשמור את המקורות מידע שהשתמשתי בהם יחסית מאוחר לתוך העבודה לפרוייקט..

# נספחים

## קישורים ל GitHub repositories

קישורים ל GitHub repositories של קבצי הפייתון והאנדרואיד:

<https://github.com/guyvandam/PythonProjectFinal>

<https://github.com/guyvandam/ProjectApp>

## תוצאות הבדיקה

במאגר הנתונים שלי יש 6 שירים. כמובן שבנית את הפרויקט שלי בצורה שניתנת להרחבה.. השירים הם:

Treasure by Bruno Mars – Id 1  
<https://www.youtube.com/watch?v=nPvuNsRccVw>  
Adventure of a lifetime by Coldplay – Id 2  
<https://www.youtube.com/watch?v=nJtK14ffgEM>  
Hymn for the weekend by Coldplay – Id 3  
<https://www.youtube.com/watch?v=g8xIoipyjzo>  
Darlin’ by the beach boys – Id 4  
<https://www.youtube.com/watch?v=-uAK0Ws6TwY>  
Animals by Maroon 5 – Id 5  
<https://www.youtube.com/watch?v=7BJ3ZXpserc>  
See you again by Wiz Kahlifa – Id 6  
<https://www.youtube.com/watch?v=RgKAFK5djSk>

נבדוק קטעים של 10 שניוות עבור כל שיר, מהתחלתו עד סופו.

* 1 = הצלחה.
* 0.5 = לא נמצאה תשובה && max{songIdNumOfKeysTable} נכון.
* 0 = לא נמצאה תשובה.
* 0.5- = תשובה לא נכונה && max{songIdNumOfKeysTable} נכון.
* 1- = תשובה לא נכונה.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| starting time of the recording in the song (seconds) | Treasure | adventure of a lifetime | Hymn for the weekend | Darlin' | Animals | See you again |
| 0 | 1 | 1 | 1 | 0 | 0.5 | -1 |
| 10 | 1 | 1 | 1 | 0.5 | 0.5 | 1 |
| 20 | 0 | 1 | 1 | 0 | 0.5 | 1 |
| 30 | 1 | 0 | 1 | 0.5 | 0.5 | 1 |
| 40 | 1 | 0 | 0.5 | 0 | 0 | -1 |
| 50 | 0 | 0 | 1 | 0 | 0.5 | -1 |
| 60 | 1 | 0 | 1 | 0 | 0 | 1 |
| 70 | 1 | 0.5 | 1 | 0 | 1 | 1 |
| 80 | 1 | 0.5 | 0.5 | 0 | 0.5 | -0.5 |
| 90 | 0 | 1 | 0.5 | 0.5 | -1 | 1 |
| 100 | 0 | 1 | 1 | 0.5 | 0 | 1 |
| 110 | 1 | -1 | 0 | 0.5 | 0 | 1 |
| 120 | 1 | 1 | -0.5 | -1 | 1 | -1 |
| 130 | -1 | 0 | 0 |  | -1 | -1 |
| 140 | -1 | 1 | 0.5 |  | 0 | -1 |
| 150 | -1 | 1 | 1 |  | -1 | 1 |
| 160 | 0 | 1 | 1 |  | 0 | 1 |
| 170 |  | 0.5 | 1 |  | 0 | 1 |
| 180 |  | 0 | 1 |  | 0 | 1 |
| 190 |  | 1 | 0.5 |  | 0 | 1 |
| 200 |  | 1 | 1 |  | 0.5 | 0 |
| 210 |  | 1 | 1 |  | 0 | -1 |
| 220 |  | 0 |  |  | 0 | -0.5 |
| 230 |  | 1 |  |  |  |  |
| 240 |  | 0 |  |  |  |  |
| 250 |  | 1 |  |  |  |  |
|  |  |  |  |  |  |  |
| Average | 0.352941 | 0.557692 | 0.727273 | 0.115385 | 0.108696 | 0.217391 |

אנחנו רואים תוצאות די גרועות עבור שירים 4 ו- 5. אנחנו רואים ששיר 4 קצר מאוד ביחס לאחרים, כמו כן, אני חושב שהבעיה הייתה בקריאת קובץ ה wav מהלקוח. פתרון יעיל היה לעשות את החישובים אצל הלקוח, כלומר ליצור את טביעת האצבע אצל הלקוח ולשלוח את המידע אל השרת, ככה נעביר פחות מידע ולא תהיה לנו בעיה של העברת וסגירת קבצים. בעיה שהייתה יכול לצוץ מפתרון זה, המימושים השונים של ה FFT עבור השפות השונות..

## Creating a new Database

if \_\_name\_\_ == '\_\_main\_\_':  
 *songIdSongInfoDict* = {'1': 'Treasure by Bruno Mars',  
 '2': 'Adventure of a lifetime by Coldplay',  
 '3': 'Hymn for the weekend by Coldplay',  
 '4': "darlin' by the Beach boys",  
 '5': 'Animals by Maroon 5',  
 '6': 'see you again by Wiz Khalifa'}  
  
 *songIdPath* = {'1': r"C:\PythonProject2\DatabaseSongs\BrunoMarsTreasure.wav",  
 '2': r"C:\PythonProject2\DatabaseSongs\ColdplayAdventureOfALifetime.wav",  
 '3': r"C:\PythonProject2\DatabaseSongs\ColdplayHymnForTheWeekend.wav",  
 '4': r"C:\PythonProject2\DatabaseSongs\Darlin.wav",  
 '5': r"C:\PythonProject2\DatabaseSongs\Maroon5Animals.wav",  
 '6': r"C:\PythonProject2\DatabaseSongs\WizKhalifaSeeYouAgain.wav"}  
  
 *db* = FingerprintDatabase()  
 *db*.createNewDatabase()  
 *db*.loadMany(*songIdPath*, *songIdSongInfoDict*)

## יומן שיקוף

**יומן שיקוף**

06.10.2019 : Process Explorer

למדתי על משתנים לוקאלים, לדוגמא על כך שאם אני יוצר משתנה לוקאלי במופע אחד למופע האחר אין גישה אליו, הוא יוצר משתנה לוקאלי שונה לגמרי.  
  
למדתי שאי אפשר למחוק קובצים מסוימים אלא אם כן ה handle שלהם מושמדים.

לא כ"כ הבנתי את המסך של הenvironment, מה זה handle ומה עוזר לי מסך ה strings.  
  
לסיכום למדתי דברים מעניינים על המחשב, על Process Explorer, ועל מערכת ההפעלה windows.

Thread synchronization : 06.12.2019  
למדתי על קטע קריטי – הקטע שבו התהילכון רוצה לגשת אל משתנה בזיכרון ולקרוא או לכתוב לו. מנעולים נועדו כדי לפתור את הבעיה שנוצרת אם שני תהליכונים נכנסים לקטע הקריטי באותו הזמן, המנעולים עושים זאת בכך שהם נותנים רק לתהליכון אחד להגיע לקטע הקריטי, לקבל גישה למשתנה, רק לתהליכון שלו יש את "המפתח" למנעול.   
למדתי על מצב של קיפאון – מצב שבו הקוד לא מתקדם בגלל שלאף תהליכון אין הרשאה להמשיך או הקוד מתוכנן כך שנוצר קיפאון בכל התהליכונים.  
לא הבנתי את הקטע של הmutex ואת סמפור.

WinAPI : 06.12.2019  
winapi מגדיר את כל הפעולות של windows מה הם עושות, איך לקרוא ולהשתמש בהם.  
COM – פונקציות של windows שניתנות להרצה ע"י כל שפת תכנות.  
לא הבנתי מה ה Mutex עושה ואת ההפעלה שלו עפ המצגת.

17.01.2019 מצב הפרויקט

* הגשתי מסמך איפיון.
* התחלתי לעבוד על הקלטת קול דרך אנדרואיד.

נספחים 24.01.2020

* <http://greenteapress.com/thinkdsp/thinkdsp.pdf> - ספר שמסביר על signal processing עם פייטון.

ואני ואלי עבדנו על "להתקין" את הספריות של thinkdsp בסוף הצלחנו. יש בספריות האלה כלים שיכולים לעזור לי להבין על signal processing ואולי אוכל למשמש חלק מהם בעצמי. קראתי גם על DFT ואני מתחיל להיות בקיא בכל המונחים והדרכים לקליטת אות והעיבוד שלה.

עדכון 01.03.2020

המשכתי לכיוון של ה matching algorithms לקחתי את הדוגמא של המסמך, יצרתי target zones, anchor pointes ו- addresses.  
גם הצלחתי ליצור spectrogram של קובץ.

עדכון 24.03.2020

תמונה שמכילה צילום מסך

התיאור נוצר באופן אוטומטיעכשיו עובד על העניין של הsectrogram עובד על החלק הזה:   
החלק של צד השרת.

עדכון 02.04.2020

בניתי כמה פונקציות הדגמה, כולל פונקציית הדגמה עבור ה lowpassFilter מעכשיו אני שם את כל פונקציות ה signalProcessing בקובץ אחד, ומשם אקרא לפונקציות שאצטרך. הוספתי פונקציה שעושה fft ומחזירה את התוצאות. השתמשתי בקבצים הבאים:

<https://docs.scipy.org/doc/numpy/reference/generated/numpy.fft.fftfreq.html#numpy.fft.fftfreq>

<https://docs.scipy.org/doc/scipy/reference/generated/scipy.fft.fft.html>

עדכון 10.04.2020

פונקציה שמחלקת את ה bins של ה fft ל bands מוצאת את המקס. בהמשך ל spectrogram filtering

עדכון 11.04.2020

חילקתי לפונקציות שעושות דבר אחד כל אחת, הוספתי הערות. לא הספקתי יותר מדי, ישבתי איזה חצי שעה... אבל בכל זאת התקדמתי.

13.04.2020

סיימתי את ה filtered spectrogram החישוב לקח הרבה זמן והסתבכתי קצת עם הקטע של היחידות, ראיתי סרטון ביוטיוב שמסביר על ה FFT ועכשיו יש לי יותר הבנה לגבי היחידות של הצירים ושל מה הפונקציות עושות.

https://www.youtube.com/watch?v=mkGsMWi\_j4Q‏

14.04.2020

המשכתי מאיפה שהפסקתי אתמול, עכשיו אני מודע לערכים של ה spectrogram סידרתי את זה כך שהפונקציה באמת פולטת spectrogram נכון. אני עדיין לא מרוצה מהתוצאות. קיבלתי נקודות צמודות אחת לשנייה לעומת נקודות פרוסות בדוגמא של המאמר שלי.  
החישוב עדיין לוקח לא מעט זמן...

תמונה שמכילה גדול

התיאור נוצר באופן אוטומטיתמונה שמכילה צילום מסך

התיאור נוצר באופן אוטומטי

עדכון 15.04.2020

עשיתי עוד קצת בדיקות, חיפשתי דרכים לחשב את ה spectrogram גם שאוכל לבדוק ולשנות דברים יותר מהר וגם בגלל שאנחנו רוצים שהתכונה תעבוד מהר. מצאתי כמה דרכים, להשתמש בתוכנות מובנות, ליצור משתנים לוקאלים וכו...  
סידרתי את הקוד, המעטתי בשורות ובסיבוכים.

עדכון 16.04.2020

שמתי לב לדבר נורא בקוד שלי, הרצתי את getNthBin פעמיים עבור כל נקודה, סתם העברתי משתנים (binSize), ושמרתי את ה dictionary שלי עם מפתחות של tuple בלי סיבה. שיניתי את זה היום, יצרתי משתנה RANGE במקום לחשב כל פעם מחדש באיזה band הנקודה נמצאת. בקיצור פעלתי בצורה מטומטמת לפני והקוד שלי היא פשוט לא טוב. עכשיו הקוד הרבה[, הרבה] יותר מהר, הוא נקי, מסודר וקריא יותר.

קטע סאונד של דקה היה לוקח לי משהו כמו דקה וחצי לחשב, עכשיו 3-4 שניות. [!!!!!!]

העמוד הזה עזר לי, ממשתי פונקציה דומה ל getIndex

<https://www.toptal.com/algorithms/shazam-it-music-processing-fingerprinting-and-recognition>

עדכון 18.04.2020

המשכתי הלאה מה signal processing לכיוון ה database שיניתי כמה דברים שעשיתי לפני, יש לי כבר את רוב הקוד כתוב מהפעם האחרונה שהתעסקתי בחלק הזה. אני צריך להחליט עכשיו על המבנה של החלק הזה מבחינת עצמים, האם אני רוצה הרבה פונקציות סטטיות או של אובקייטים, מה הם האובקייטים בכלל? בינתיים אני עם אובייקטים: song, recording, database

עדכון 19.04.2020

התקדמות לא רעה היום.. כמעט סיימתי את השלב של ה time coherency אמור לקחת לי כמה דקות לסיים. [עדכון לעדכון: עכשיו סיימתי את זה פחדתי שלא אסיים את זה היום, אז כתבתי את היומן שיקוף.] עכשיו נשאר רק להריץ ולבדוק שהכל עובד. צריך עוד ליצור פונקציה שטוענת השירים ל database אולי אתחיל ללמוד sql בקרוב. עד סוף השבוע אני מצפה שיהיה לי database (חלקי אפילו) ב sql ואוכל להתחיל לעשות בדיקות ולראות איזשהו מוצר גמור.

עדכון 24.04.2020

עד עכשיו עבדתי על sql, התקנתי והתחלתי לשחק עם התוכנה, הבנתי ש SQL יסבך לי את הקוד וניסיתי לעבור למשהו אחר, מצאתי את mongoDB שנראה מבטיח נכון לעכשיו, אמשיך לעבוד איתו בימים הקרובים...  
הבנתי שטבלאות של SQL לא נותנות לי שימוש. ההפך הם רק מקשות ומסבכות את מבנה הנתונים.

עדכון 25.04.2020

MongoDB עובד!! (בערך). מה שכתבתי ב GitHub:

added the whole part for the memory database. now my database is saved with monogDB. I have 2 dictioneris right now, the address - list of couple dictionary and the songId - addressCoupleDict dictionary.   
for now, it works good, instead of using 3 decimal points, I multiply all the values by 1000, it’s not ideal but due to the fact that mongoDB keys can’t contain any point characters, it’s the best solution for now.

עדכון 26.04.2020

checked the code today, it didn’t work. I started working with the built-in spectrogram functions of matplotlib and scipy, matplot worked best. I recon I have 2 problems with the code, the spectrogram peak finding and the fact that I’m not going to get the same exact value with recordings and songs, so I'll have to start working with epsilons. I have 3 methods for the peak finding, and I'll experiment with them in the coming days.

עדכון 29.04.2020

added 2 functions for filtering the peak points to stop bands of the same frequency, it worked to an extent. also added a function to filter the spectrogram points using regions (סביבות), this method is 1-4 seconds slower than before (depending on the configuration)