

Generative Music Platform

עבודת גמר 5 יח"ל במגמת מדעי המחשב



נושא הפרוייקט – Generative Music

שם התלמיד – ערן מחלב

ת"ז – 323103150

שם המנחה – תומר טלגם

מורה מלווה – ירון יצחקי

שם בית ספר – תיכון אהל שם

שנה – 2019

<https://github.com/emachlev/GenerativeMusicPlatform>

תוכן עניינים

3	תקציר כולל ורקע.....
4	שפת התכנות וסביבת העבודה.....
5	מודולים וספריות שהשתמשתי בהם בפרוייקט.....
7	מבנה המערכת ואופן פעולתה.....
10	דוגמה ליצירת מנגינה.....
11	הרכבת הבתים.....
14	הרכבת השרשרת.....
18 schedule
19	גיבוי ובקרת גירסאות.....
20	צעדים להמשך.....
21	רפלקציה.....
21	ביבליוגרפיה.....

תקציר כולל ורקע

שמעתם פעם שיר שכל כך אהבתם שרציתם שהוא יימשך לנצח?

כולנו היינו שם: בין אם אנחנו מקשיבים למוזיקה סתם לכיף או כדי שהיא תעזור לנו להתרכז, השירים שאנחנו אוהבים תמיד קצרים מדי. לנגן אותם על repeat זה לא דבר טריוויאלי, כי חזרה מדויקת הופכת למשעממת דיי מהר.

Generative Music (מוזיקה ג'נרטיבית) הוא מונח המתאר מוזיקה המתנגנת על ידי תוכנת מחשב, ועונה על הקריטריונים הבאים:

- היא צריכה להיות שונה בכל פעם שמנגנים אותה, ללא חזרות ניכרות.
- היא צריכה להתנגן כל עוד מישהו מאזין לה.

מוזיקה ג'נרטיבית מציעה הזדמנות ליהנות כל הזמן מיצירות מוזיקליות שאנחנו אוהבים ללא החסרונות של מוזיקה מוקלטת, שהיא סופית ונמשכת זמן קבוע.

העיקרון העומד במרכז הפרוייקט שלי הוא יצירת המשכים למנגינות בעזרת אלגוריתמים מורכבים. המוצר הוא פלטפורמה היוצרת ומנגנת גרסאות אינסופיות של יצירות מוזיקליות שונות, והיא נגישה דרך הדפדפן, כך שמשתמש במערכת יוכל להאזין לה בקלות ומכל מכשיר.

שפת התכנות וסביבת העבודה

הפרוייקט הוא אפליקציית Web שצד השרת שלה כתוב ב-**Node.js**, סביבה המריצה קוד JavaScript מחוץ לדפדפן. Node.js מאפשרת יצירה של שרתי Web וכלים מבוססי רשת בעזרת JavaScript ואוסף של "מודולים" שמטפלים במימוש של פונקציות שונות. ל-Node.js תמיכה רשמית ברוב מערכות ההפעלה, והקוד שלה [פתוח](#) לחלוטין. ההבדל המרכזי בינה לתכנות בשפות אחרות כמו PHP הוא שרוב הפונקציות ב-PHP סינכרוניות, כלומר חוסמות את ריצת התוכנית עד שפעולתן נגמרת, בעוד פונקציות ב-Node.js הן **א-סינכרוניות**. הן רצות במקביל ומשתמשות ב-Callbacks כדי להודיע על הצלחתן או כשלונן.



השתמשתי ב-**PyCharm** בתור סביבת העבודה. היא ידועה בתור סביבת העבודה הכי נוחה לפייתון אשר מאפשרת כתיבה נוחה של הפרוייקט. גירסת ה-Pro שלה תומכת בכתיבה והרצה בשפות ופלטפורמות אחרות הכוללות את Node.js, ומספקת יכולת להוריד מודולים בלחיצה אחת ופירמוט אוטומטי של קוד.



מודולים וספריות שהשתמשתי בהם בפרוייקט

- **Express** – framework לאפליקציות Web ו-APIים עבור Node.js. בחרתי בו כי הוא מודול מאוד דומיננטי בקהילת המפתחים ב-Node ומשתמשים בו גם באתרים פופולאריים כמו PayPal ו-Uber.



- **EJS** – מנוע templates הכתוב ב-JavaScript ואחראי על רינדור ה-HTML באפליקציה. מנועי templates מסייעים בהפרדת קוד צד השרת המיועד לעיבוד של המידע מהקוד המציג אותו למשתמש בפועל. במקום להוסיף תוכן דינאמי לאתר בקוד צד השרת, בסיום העיבוד השרת מעביר את המידע למנוע ה-templates, והוא מציב אותו במקום המיועד לכך ב-HTML. בחרתי ב-EJS כי הוא פשוט ונוח, ולא היה לי צורך במנועים עוצמתיים יותר התומכים במגוון רב של אופציות כמו ירושה של אלמנטים וטמפלייטים.

- **Material Design for Bootstrap 4** – ערכת ה-UI הטובה ביותר (לפחות לדעתי...) לבניית אתרים ואפליקציות רספונסיביים ומותאמים למובייל. MDB מציעה יותר מ-500 אלמנטים מעוצבים, אייקונים, אנימציות CSS, טמפלייטים ומדריכים לשימוש אישי ומסחרי, והכל לפי הקונבנציות של העיצוב המטריאלי של Google.



- **Tone.js** – ספרייה לניגון מוזיקה בדפדפן באמצעות ה-Web Audio API. בהינתן קבצי השמע המתאימים, Tone.js מתנהגת למעשה כפסנתר הנשלט על ידי קוד. לדוגמה:

```
piano.triggerAttackRelease("C3", 1);
```

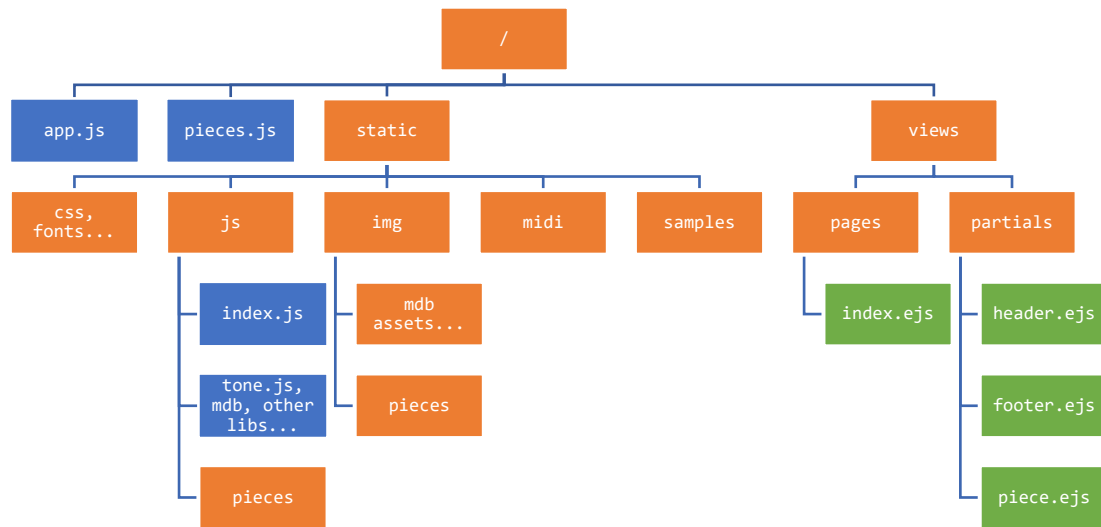
שורה זו מדמה לחיצה על התו "דו" (יפורט בהמשך) במשך שנייה, ושחרור שלו לאחר מכן. בנוסף השמשתי בספריית tonejs-instruments שדואגת לטעינת קבצי השמע (samples) של כל התווים למחשב הלקוח.



- **jQuery** – ספריית JavaScript שמטרתה להקל על כתיבת סקריפטים לצד הלקוח. העבודה עם ספרייה זו פשוטה ונוחה, ועושה פעולות כמו הסתרת אלמנטים בדף קלות וקצרות יותר על מנת שהקוד יהיה קצר ונוח לקריאה.



מבנה המערכת ואופן פעולתה



קובץ **pieces.js** – אינדקס של מנגינות מוכנות שהמערכת מציעה למשתמשים. מכיל רשימה של אובייקטים מסוג **piece**, לשימוש ע"י **app.js** (בהמשך).

• **piece** – אובייקט המייצג מנגינה ג'נרטיבית ומידע עליה. מנגינה היא מילון הכולל את המאפיינים הבאים:

- **name** – שם היצירה שעליה מתבססת המנגינה
- **by** – כותב היצירה
- **description** – תיאור המנגינה
- **file** – מיקום של סקריפט JavaScript היוצר את המנגינה (ביחס ל-
(/static/js/pieces
- **image** – מיקום של קובץ תמונה עבור המנגינה (ביחס ל-
(/static/img/pieces

```
const pieces = [
  assemblePiece('Aisatsana', 'Aphex Twin', 'Aisatsana [102]' is the closing track from Aphex Twin • Syro' released 22
  September 2014', 'aisatsana.js', 'aisatsana.jpg'),
  assemblePiece('Not Yet Remembered', 'Harold Budd and Brian Eno', 'The Plateaux of Mirror, 1980',
  'not_yet_remembered.js', 'not_yet_remembered.jpg'),
];

function assemblePiece(name, by, description, file, image='') {
  return { 'name': name, 'by': by, 'description': description, 'file': file, 'image': image }
}

module.exports = pieces;
```

קובץ **app.js** – הסקריפט הראשי שמופעל עם הרצת האפליקציה. מייבא את רשימת המנגינות מ-pieces.js, קובע את המודולים והפרמטרים הדרושים לריצת השרת, ה-templating engine ואת תיקיית הקבצים הסטטיים, ומתחיל להקשיב לחיבורים מלקוחות:

```
const pieces = require('./pieces.js');

const PORT = 8081;

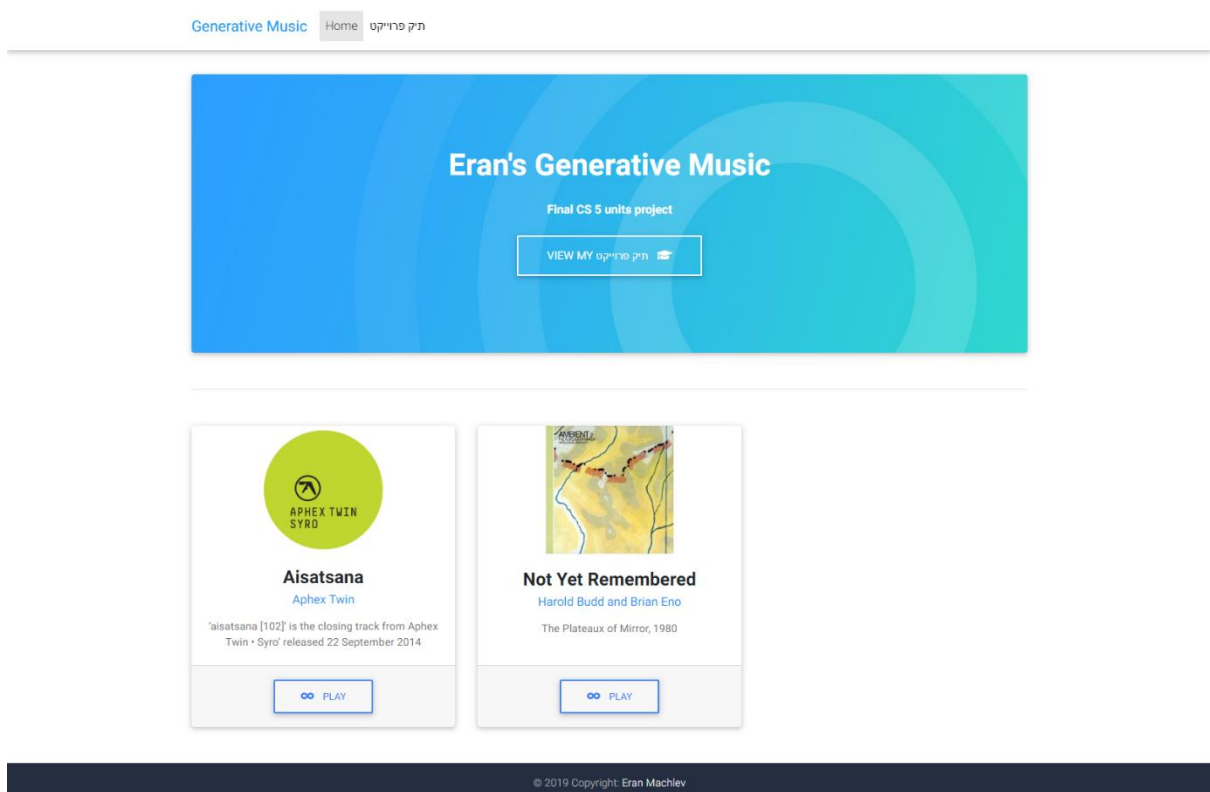
// server.js
// load the things we need
const express = require('express');
const app = express();

// set the view engine to ejs
app.set('view engine', 'ejs');
// set the static path
app.use('/static', express.static('static'));

// index page
app.get('/', function (req, res) {
  res.render('pages/index', {
    pieces: pieces
  });
});

app.listen(PORT);
console.log('Listening on: localhost:' + PORT);
```

כאשר לקוח נכנס לעמוד הראשי באתר, השרת מחזיר לו את עמוד האינדקס עם כל המנגינות שהוא יכול להקשיב להן. כשהוא בוחר מנגינה כפתור ה-PLAY הופך לכפתור STOP וקובץ ה-JS המקושר למנגינה מורד ומתחיל לרוץ מקומית במחשב שלו. כשהלקוח מעוניין להפסיק להאזין למנגינה, הוא לוחץ על כפתור ה-STOP וריצת הקובץ מפסיקה.



קובץ `/static/js/index.js` – סקריפט הלקוח הראשי. רץ במחשב הלקוח עם טעינת האתר, ולאחר טעינת כל המודולים הדרושים לפעולת המערכת. סקריפט זה מגדיר את האובייקטים והפעולות שהמנגינות צריכות כדי לרוץ כהלכה.

```
let schedule = null; // The schedule of the currently playing piece
let playingCaller = null; // The Play button that was pressed to play the currently playing piece

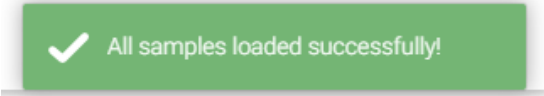
function playPiece(file, caller) {
  if (Tone.context.state !== "running") {
    Tone.context.resume();
  }
  let tempPlayingCaller = playingCaller;
  if (schedule) {
    stopPiece(playingCaller)
  }
  if (caller !== tempPlayingCaller) {
    $.getScript(file);
    $(caller).html('<i class="fas fa-stop mr-2 piece-play-icon"></i> Stop');
    $(caller).removeClass('btn-outline-primary');
    $(caller).addClass('btn-primary');
    playingCaller = caller;
  }
}

function stopPiece(caller) {
  Tone.Transport.cancel();
  schedule = null;
  $(caller).html('<i class="fas fa-infinity mr-2 piece-play-icon"></i> Play');
  $(caller).removeClass('btn-primary');
  $(caller).addClass('btn-outline-primary');
  playingCaller = null;
}

let piano = SampleLibrary.load({
  instruments: "piano"
});

Tone.Buffer.on('load', function () {
  $('.piece-play-btn').removeAttr('disabled');
  toastr.success('All samples loaded successfully!');
  piano.toMaster();
  Tone.Transport.start();
});
```

בהתחלה, הסקריפט מאתחל את אובייקט הפסנתר ע"י טעינת ה-samples הדרושים, ולאחר מכן הוא מאפשר למשתמש לנגן כל אחת מהיצירות הזמינות באתר ומודיע לו על כך באמצעות התראת toast:



✓ All samples loaded successfully!

הפעולות `playPiece` ו-`stopPiece` מקבלות מה-HTML את הפרמטרים לזיהוי היצירה, והן מטפלות בטעינת הסקריפט המתאים ובהפעלה או ההפסקה של ה-schedule שלו (יפורט בהמשך). בנוסף, הן מקבלות את הכפתור שהמשתמש לחץ עליו כדי לשנות את המצב שלו מ-PLAY ל-STOP ולהפך.

דוגמה ליצירת מנגינה

פרק זה הוא החלק המרכזי בפרוייקט וכאן בא לידי ביטוי המחקר שביצעתי. המשימה הייתה לקחת יצירה מוזיקלית מוכנה, ולבנות תהליך שמנגן אותה במבנה ובחוקיות שלה בלי סוף וללא חזרות (כלומר: לחולל לה המשך התואם את מאפייניה הייחודיים). תחילה חיפשתי מנגינה מתאימה בעלת מבנה קבוע שיהיה נוח לבנות ממנה גרסה ג'נרטיבית, ובחרתי ב-[aisatsana](#) מאת Aphex Twin. כדי להסביר למה [aisatsana](#) היא בחירה מעולה למטרה הזאת, אצטרך להגדיר כמה מושגים:

- **תו** – צליל בעל שם מוגדר, כמו "דו", "רה", "מי"...
- **פעימה** – יחידת הבסיס של הזמן במנגינה. המהירות שבה יצירה מנוגנת יכולה להימדד בפעימות לדקה, או **BPM** (קצב). הקצב של [aisatsana](#) הוא 102 BPM. זאת אומרת שאורך פעימה אחת ביצירה הזאת הוא קצת פחות מ-0.6 שניות. כל תו יכול להיות באורך של פעימה אחת או חצי פעימה.

ל-[aisatsana](#) מבנה פשוט יחסית, והיא עוקבת אחרי דפוס קבוע: מהתו הראשון, כל 16 פעימות מכילות סדרה עם כמות מסויימת של תווים, שאותה נכנה "בית".

לדוגמה, בין השנייה השלישית לשנייה ה-12 נמצא הבית הראשון. בין השנייה ה-12 לשנייה ה-21 נמצא הבית השני, וכך הלאה עד סוף היצירה, שכוללת 32 בתים בסך הכל.

יצירה זו פשוטה בהרבה מהשירים הפופולאריים המקובלים כיום, שמורכבים מחלקים שקיים ביניהם שוני מהותי, כמו בית ופזמון.

מנקודת המבט הזאת, למעשה ניתן לתאר את [aisatsana](#) כאלגוריתם:

- כל 16 פעימות בצע את התהליך הבא {
 - נגן בית באורך 16 פעימות}
- {

מערכת שתייצר מנגינה בסגנון של aisatsana תצטרך להתבסס על האלגוריתם הנ"ל. פעימה היא יחידת זמן שניתן להמיר לשניות בהינתן BPM של השיר, אז נציג זאת בשפה שהמחשב יבין:

- $102 = \text{BPM}$
- $\text{זמן_פעימה} = 60 * (1/102)$
- כל ($16 * \text{זמן_פעימה}$) שניות בצע את התהליך הבא {
 - נגן בית באורך $16 * \text{זמן_פעימה}$ שניות}

כעת אסביר על האופן בו המערכת מרכיבה בכל פעם את הבית שעליה לנגן.

הרכבת הבתים

דרך אחת ולגמרי לגיטימית לבנות גירסה אינסופית של aisatsana היא להפריד את כל ה-32 בתים במנגינה המקורית, וכל 16 פעימות לבחור בית אקראי ולנגן אותו. התוצאה תהיה שונה מהמקור, והיא תהיה גם אינסופית, אך משתמש שמאזין לה הרבה זמן יזכור בסוף את כל 32 הבתים, וכאמור, חזרה מדוייקת נהיית משעממת דיי מהר.

חשוב שהמערכת תדע לבנות ולנגן בתים חדשים בנוסף לבתים הקיימים ביצירה, ושהם יישמעו דומה לבתים המקוריים.

לא ניתן להגריל תווים אקראית ולנגן אותם כל 16 פעימות. אידיאלית, מישהו שאף פעם לא שמע את המנגינה המקורית יוכל להקשיב לגירסה האינסופית שלה בלי שהוא יוכל להבדיל בין הבתים המקוריים לבתים שהמערכת מרכיבה בעצמה.

קיימות לא מעט דרכים לייצר מוזיקה חדשה על בסיס קלט קיים. חלק גדול מהן מתבסס על עקרונות של deep learning, אבל דבר כזה ידרוש קלט הרבה יותר גדול מ-32 בתים. אחרי חיפוש קצר באינטרנט מצאתי פתרון הולם: **שרשראות מרקוב**.

שרשרת מרקוב היא מודל הסתברותי שמורכב מרשימה של מצבים והסתברויות לעבור ממצב אחד למצב אחר. על מנת לפשט את ההסבר, נניח שהיצירה מורכבת משני בתים, שכל בית הוא באורך 4 פעימות וכולל 4 תווים באורך פעימה אחת:

בית 1 – לה, פה, לה, פה

בית 2 – מי, דו, לה, דו

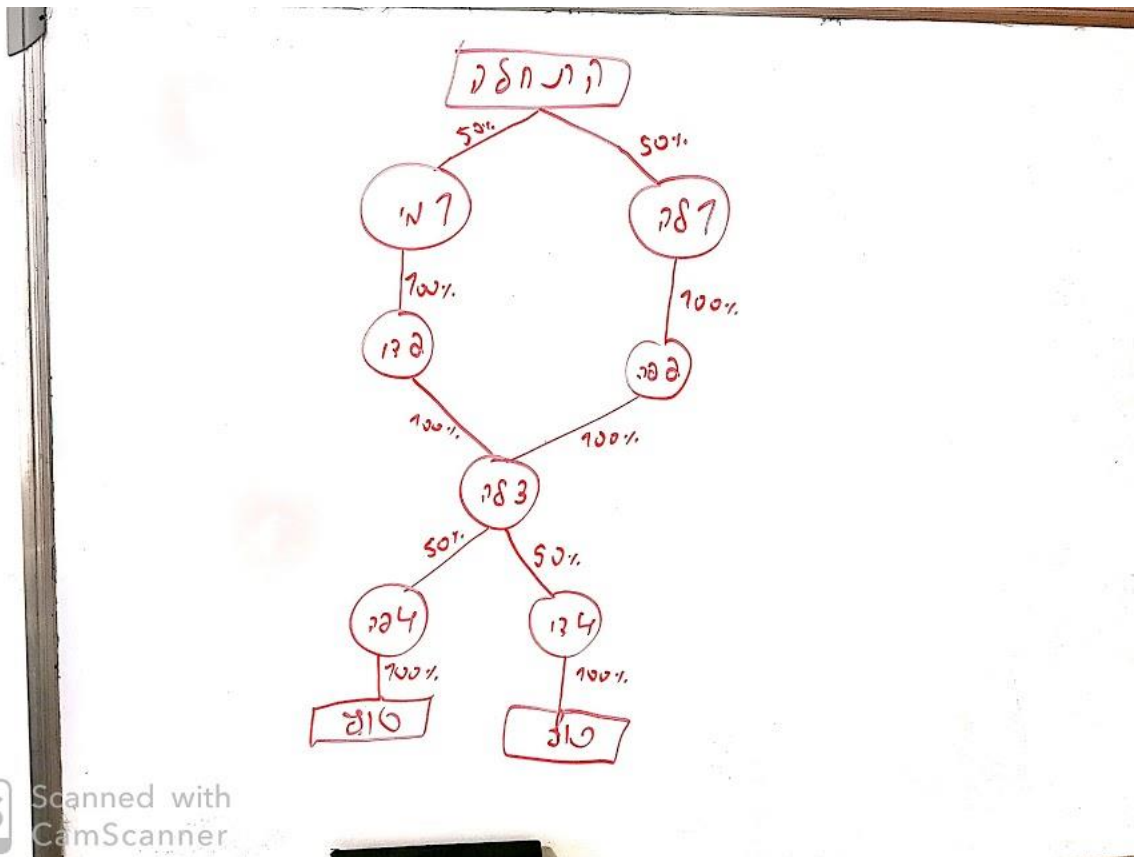
אדגים בנייה של שרשרת מרקוב על פי שני הבתים האלה. תחילה, נצטרך להגדיר את המצבים: תחילת הבית, סוף הבית, וכל פעימה בבית.

בית 1 – התחלה ← 1לה ← 2פה ← 3לה ← 4פה ← סוף

בית 2 – התחלה ← 1מי ← 2דו ← 3לה ← 4דו ← סוף

(המספר שלפני כל תו מייצג את מיקום הפעימה שלו בבית)

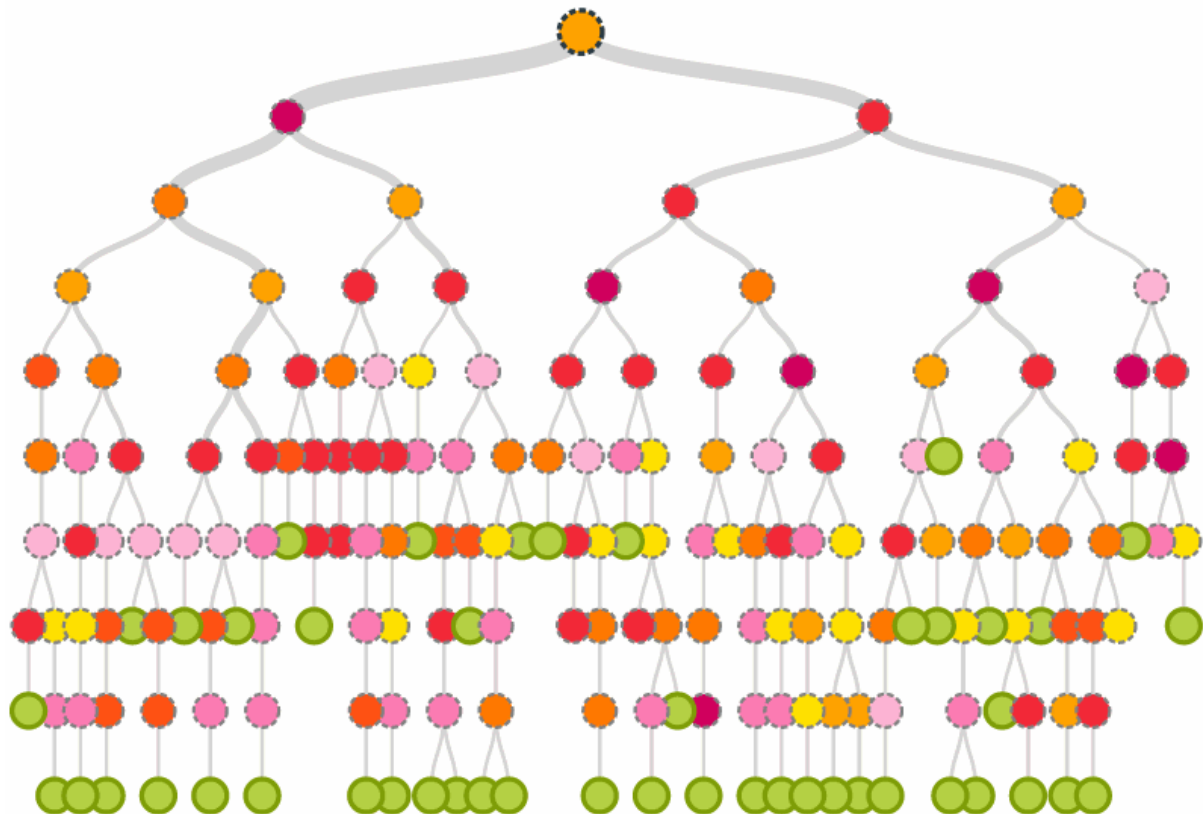
השרשרת מורכבת מתשעה מצבים: התחלה, 1לה, 1מי, 2פה, 2דו, 3לה, 4פה, 4דו, וסוף. הנה עץ ההסתברויות למעברים בין כל אחד מהמצבים:



עם המודל הנ"ל, אפשר ליצור בתים על ידי "הליכה" על העץ – התחלה ב"התחלה", ובחירה בכל המעברים לפי ההסתברויות עד שמגיעים לסוף. תוצאה אפשרית של הליכה על העץ הזה יכולה להיות **התחלה** ← 1לה ← 2פה ← 3לה ← 4ד ← סוף.

זה בית חדש לגמרי שלא מופיע ביצירה המקורית, והוא יישמע דומה לשניים האחרים כי שלושתם נוצרו מאותה שרשרת מרקוב.

אפשר להשתמש בדיוק באותה השיטה ל-aisatsana עם שרשרת מרקוב הרבה יותר גדולה. יהיו בה יותר מצבים ומעברים, יהיו בה מספר תווים שמתנגנים יחד (אקורדים) ויהיה אפשר להריץ עליה את אותו התהליך כדי לייצר בתים חדשים שהם לא חלק מהיצירה המקורית אבל תואמים את החוקיות שלה.



הרכבת השרשרת

בפועל, בבתים ב-aisatsana קיימים תווים שמתנגנים בזמן של חצי פעימה. אז בקוד נתייחס אליה כמנגינה של 204 BPM (הקצב יוכפל בשתיים) ונספור חצאי פעימות במקום פעימות שלמות.

כדי לעבוד על aisatsana במערכת שלי, הייתי צריך ייצוג דיגיטלי שלה. בחרתי להשתמש בקבצי MIDI המומרים ל-JSON בגלל שפשוט לעבוד איתם ב-JS. קבצי MIDI כוללים את ההוראות המדויקות לנגן את היצירה – על איזה תווים ללחוץ ומתי. את קבצי ה-JSON שמתי בתיקיית midi בספרייה static. הנה חלק מהתוכן של :aisatsana.json

```
"notes": [
  {
    "name": "E3",
    "midi": 52,
    "time": 0,
    "velocity": 0.30708661417322836,
    "duration": 0.5882355
  },
  {
    "name": "G3",
    "midi": 55,
    "time": 0.5882355,
    "velocity": 0.31496062992125984,
    "duration": 0.5882355
  },
  {
    "name": "C3",
    "midi": 48,
    "time": 1.176471,
    "velocity": 0.2992125984251969,
    "duration": 0.5882354999999999
  },
  {
    "name": "C4",
    "midi": 60,
    "time": 1.7647065,
    "velocity": 0.33858267716535434,
    "duration": 7.6470615
  },
  {
    "name": "E3",
    "midi": 52,
    "time": 9.411768,
    "velocity": 0.30708661417322836,
    "duration": 0.5882354999999997
  },
  {
    "name": "G3",
    "midi": 55,
    "time": 10.0000035,
    "velocity": 0.31496062992125984,
    "duration": 0.5882354999999997
  }
],
```

הקובץ מורכב מרשימה של מבני נתונים מסוג תו, ולכל תו יש שם ואת הזמן ביחס לתחילת היצירה שהוא מנוגן בו. המספר ליד כל שם של תו מייצג את האוקטבה שלו. אין צורך להבין את המושג הזה, רק לדעת שהוא מזהה את הצליל הספציפי על הפסנתר שמייצג את התו ביצירה.

כאמור, כשהלקוח לוחץ על PLAY, הקובץ aisatsana.js מורד למחשב שלו ומורץ שם. כל קובץ מנגינה מורכב מקוד שמכין אותה, ופעולת schedule שחוזרת על עצמה ב-interval קבוע. תחילה מגדירים מספר קבועים:

```
BPM = 204;  
SECONDS_PER_MINUTE = 60;  
NOTES_IN_BEAT = 2;  
NOTE_INTERVAL_SECONDS = SECONDS_PER_MINUTE / (NOTES_IN_BEAT * BPM);  
SONG_LENGTH = 301;
```

מגדירים את הקצב לפעמיים הקצב המקורי של המנגינה, כדי לטפל בחצאי פעימות. המשתנה **NOTES_IN_BEAT** קובע את מספר התווים הבסיסיים המקסימלי שיש בפעימה אחת. כדי לפצות על ההכפלה של הקצב, אנחנו מציבים בו 2.

המשתנה **NOTE_INTERVAL_SECONDS** קובע את האורך של כל פעימה בשניות, וערכו מתבסס על הקבועים שמעליו.

המשתנה **SONG_LENGTH** הוא אורך היצירה בשניות.

```
$.getJSON('/static/midi/aisatsana.json', function (data) {

  notes = data.tracks[1].notes.slice(0); // The entire track
  pressedNotes = []; // Notes that are pressed at each beat

  for (time = 0; time <= SONG_LENGTH; time += NOTE_INTERVAL_SECONDS) { // Filling the list
    pressedNotesInCurrentBeat = notes.filter(note =>
      time <= note.time && note.time < time + NOTE_INTERVAL_SECONDS
    ).map(({name}) => name).sort();
    pressedNotes.push(pressedNotesInCurrentBeat.join(','));
  }

  verses = []; // List of verses
  verseLengthBeats = 32; // Every verse is 16 beats
  pressedNotesCopy = pressedNotes.slice(0);
  while (pressedNotesCopy.length > 0) { // Divide pressed notes to verses
    verses.push(pressedNotesCopy.splice(0, verseLengthBeats));
  }

  versesWithIndex = verses.map(verse =>
    verse.map((names, i) =>
      names.length === 0 ? `${i}` : `${i}${','}${names}`
    )
  );

  chain = new Chain(versesWithIndex);

  Tone.Transport.scheduleRepeat(
    schedule,
    verseLengthBeats * NOTE_INTERVAL_SECONDS
  );
});
```

טוענים את קובץ ה-JSON מהשרת. כשההורדה מסתיימת מתחילים בתהליך ההכנה – לוקחים את רשימת התווים ועושים עליה איטרציה. בכל פעם מתקדמים זמן של פעימה אחת (כלומר, חצי פעימה בפועל) ורושמים את התווים שלחוצים בכל פעימה. מה שמתקבל מהלולאה הראשונה הוא רשימה של מחרוזות. האינדקס של איבר ברשימה הוא מספר החצי-פעימה ביצירה, והמחרוזת המוחזרת עבורו היא רשימה של תווים המופרדת בפסיקים.

```
513: ""
514: "G4"
515: ""
516: "A4,E4,G3"
517: ""
518: "B4"
519: ""
520: "C3,C5,D4"
521: ""
522: ""
523: ""
524: "C4,E3,E5"
525: ""
526: ""
527: ""
528: ""
529: ""
```


בלולאה השנייה, לוקחים את הרשימה הזאת ומחלקים אותה ל-32 חלקים שווים. בית אחד הוא באורך 16 פעימות, ושתי פעימות בקוד מייצגות פעימה אחת ביצירה. הרשימה verses היא רשימה של רשימות באורך 32. כל רשימה מייצגת בית ביצירה, וכל איבר ברשימה מייצג פעימה (חצי פעימה), איבר הוא מחרוזת מופרדת בפסיקים שמגדירה את התווים הלחוצים בחצי הפעימה.

[illegible]

לאחר מכן, לוקחים את רשימת הבתים ומוסיפים לכל האיברים את האינדקס שלהם. זאת על מנת להגדיר את המצבים השונים לבניית השרשרת. `versesWithIndex`:

[illegible]

על מנת להפוך את תהליך בניית השרשרת עצמה לפשוט ביותר, השתמשתי בספריית [markov-chains](#). היא מספקת אובייקט Chain המקבל רשימה של מחרוזות ומרכיב ממנה שרשרת מרקוב. כך אפשר לקבל תוצר של "הליכה" על עץ ההסתברויות בשורה אחת:

```
chain = new Chain(versesWithIndex);
generated verse = chain.walk();
```

לבסוף, משתמשים במחלקת `Tone.Transport` כדי לתזמן קריאה לפעולה `schedule` כל 32 חצאי פעימות.

```
schedule = () => { // For each generated verse (runs indefinitely)
  verse = [];
  while (verse.filter(ve=> ve.includes(',')).length < 5) {
    verse = chain.walk(); // Walk the markov chain and get a verse
  }
  verse.forEach(str => { // For each beat in the verse
    [t, ...names] = str.split(','); // returns [index, note1, note2...] or just [index] if current
    beat is a rest
    parsedT = Number.parseInt(t, 10); // Get the current beat's delay in the verse
    names.forEach(name => { // Play every beat in the specified delay (index*duration_of_note)
      waitTime = parsedT * NOTE_INTERVAL_SECONDS;
      piano.triggerAttack(name, `${waitTime + 1}`);
    });
  });
};
```

תחילה, הולכים על עץ ההסתברויות עד שמקבלים בית שיש בו לפחות 5 תווים. לאחר מכן, מפרקים כל פעימה בבית לאינדקס שלה ולשמות התווים שמנגנים בה. את שמות התווים מעבירים אחד אחרי השני לפעולה `triggerAttack` של `Tone.js` שתנגן אותם למשתמש. לאינדקס של הפעימה יש תפקיד חשוב כאן – הוא קובע את הדילי שמנגנים בו כל תו. זאת על מנת שלא ישמעו את כל התווים בבית באותו זמן.

הפעולה הנ"ל רצה כל 16 פעימות. כל הבתים נשמעים כאילו הם הגיעו מאותה היצירה, והניגון הוא אקראי לגמרי. הסתברותית, לא ייתכן מצב שבית אחד יחזור על עצמו. המנגינה תמשיך להתנגן כל עוד מישהו מקשיב לה ולכן היא עומדת בתנאים של מוזיקה ג'נרטיבית.

גיבוי ובקרת גירסאות

על מנת לעקוב אחרי שינויים בקוד שלי ולמנוע אובדן של מידע, השתמשתי ב-Git ואירחתי את הפרוייקט ב-GitHub:

<https://github.com/emachlev/GenerativeMusicPlatform>

Git סיפק לי את היכולת לחזור אחורה במקרה של בעיה בקוד ולחלק את השינויים שלי לבראנצ'ים נפרדים על מנת להפוך את העבודה לנוחה יותר.



צעדים להמשך

- בניית גרסאות ג'נרטיביות למספר מנגינות נוספות
- הוספת אפשרות לבניית מנגינה על בסיס קלט מהמשתמש (כנראה MIDI)
- שיפור הממשק הגרפי (הוספת אמצעים ויזואליים)

רפלקציה

תהליך העבודה היה מעניין ומלמד מאוד. אני אוהב מוזיקה ושמחתי לבנות את הפרוייקט בנושא שמלהיב אותי. אף פעם לא חשבתי שאפשר לבנות מנגינה שלמה בעזרת אלגוריתמים ממוחשבים ושהיא תישמע טוב.

אני מאמין שהתחום של מוזיקה ג'נרטיבית ימשיך להתפתח בעתיד, ושישתמשו בתוכנות כדי להלחין שירים חדשים.

במהלך הפיתוח והמחקר נתקלתי במספר אתגרים, כמו זה שהמנגינה כללה תווים שנכנסו בחצאי פעימות ולא בפעימות שלמות. אחרי חיפוש קצר באינטרנט מצאתי את הפתרון, שגם היה קל לממש.

אני שמח שבחרתי בנושא הזה לפרוייקט שלי כי למדתי ממנו המון והוא שינה את נקודת המבט שלי על כל מה שנוגע ליצירה של תוכן מקורי באמצעות תוכנה.

ביבליוגרפיה

How Generative Music Works – A Perspective: <https://teropa.info/loop>

Generative Music: https://en.wikipedia.org/wiki/Generative_music

Beat: [https://en.wikipedia.org/wiki/Beat_\(music\)](https://en.wikipedia.org/wiki/Beat_(music))

Aisatsana on Musescore: <https://musescore.com/eljest/scores/5179638>

Markov chain: https://en.wikipedia.org/wiki/Markov_chain