Mosab Mohamed – B20-04 – AI Assignment #2 Report

Task:

- Given a monophonic midi file of a melody. Generate an accompaniment for the melody by using an evolutionary algorithm.
- The accompaniment should be represented by a sequence of chords, each chord should contain three notes.

Solution details:

- We set a fixed length of the chords, so we can have a smaller search space.
  - The fixed length of each chord in the individual is one quarter.
- We have the following representation:
  - Population: list of individuals.
  - Individual: list of chords.
  - Chord: list that consists of 3 notes.
  - Note: an integer number that represents the midi value of a note.
- We want an accompaniment that compliments the original melody. And to achieve that we need to:
  - Be similar to the original melody in terms of flow.
  - Not be in the same octave as the original melody to avoid dissonance.
  - Have the chords follow the original melody's key.
  - Avoid having the same chord played consecutively for too long.
- To ensure progress in each generation:
  - We keep the best 50% of the previous generation.
  - We produce the other 50% of the population from the best 50% of the previous generation.
- To ensure variation in each generation:
  - We have a relatively high mutation probability which is 5% for each chord in the individual.
  - We mutate by replacing a chord in the individual with an entirely new random chord.

Algorithm Description:

- The program takes an input midi file (*.mid) that consists of melody and applies genetic programming(evolutionary algorithm) to generate an accompaniment consisting of several three note chords(triads) that would sound pleasing while played with the input melody.

Dependencies

- Python 3.8
- Python library mido
  - Used to parse the input midi file to readable values of notes and times.
  - Used to write the output file using the tracks of the input and the generated accompaniment.
- Python library music21
  - Used to detect the key of the input melody to help us in the fitness function.
- Python library numpy and random
  - Used to help us execute various random operations.

Genetic algorithm components:

- **Representation:**
  - The **population** is represented as a list of possible **individuals**(accompaniments).
  - Each **individual** is a list of N **chords**, the individual represents a possible solution to the problem.
  - Each **chord** is a list of 3 **notes,** the chord represents a chord from the following set of chords:
    - Major triad, Minor triad.
    - First inverted major triad, First inverted minor triad.
    - Second inverted major triad, Second inverted minor triad.
    - Diminished chord.
    - Suspended second chord.
    - Suspended fourth chord.
    - Rest.
  - Each **note** is an integer number that represents the midi value of a specific note.
  - Each **chord** object is played for one quarter duration which equals 384 in the mido library time units.

- **Fitness function:**
  - The fitness value of an individual depends on three aspects:
    - The similarity of each chord in the individual to the average note played in the same quarter of the original melody in a lower octave.
      - While for empty quarters we calculate the similarity to the average note played in the next quarter that does play some notes.
      - To avoid dissonance we change the target average we want to reach to the same melody but shifted 2 octaves below the original melody.
        - For example if we have an average note played in a quarter of the original melody that is equal to C5.
          We make the targeted average be C3.
    - The existence of each chord of the individual in the possible chords we can compose out of the original melody key.
      - Based on the tables presented in the assignment, and that a valid chord of some key is three notes that start from a root note in the table and between a pair of consecutive notes we skip a note.
        e.g : I-iii-V in major, i-III-v in minor.

Minor Keys	i	ii°	III	iv	v	VI	VII
Cm	Cm	D°	E♭	Fm	Gm	A♭	B♭
C♯m	C♯m	D♯°	E	F♯m	G♯m	A	B
Dm	Dm	E°	F	Gm	Am	B♭	C
D♯m	D♯m	E♯°	F♯	G♯m	A♯m	B	C♯
E♭m	E♭m	F°	G♭	A♭m	B♭m	C♭	D♭
Em	Em	F♯°	G	Am	Bm	C	D
Fm	Fm	G°	A♭	B♭m	Cm	D♭	E♭
F♯m	F♯m	G♯°	A	Bm	C♯m	D	E
Gm	Gm	A°	B♭	Cm	Dm	E♭	F
G♯m	G♯m	A♯°	B	C♯m	D♯m	E	F♯
A♭m	A♭m	B♭°	C♭	D♭m	E♭m	F♭	G♭
Am	Am	B°	C	Dm	Em	F	G
A♯m	A♯m	B♯°	C♯	D♯m	E♯m	F♯	G♯
B♭m	B♭m	C°	D♭	E♭m	Fm	G♭	A♭
Bm	Bm	C♯°	D	Em	F♯m	G	A

Major Keys	I	ii	iii	IV	V	vi	vii°
C	C	Dm	Em	F	G	Am	B°
C♯	C♯	D♯m	E♯m	F♯	G♯	A♯m	B♯°
D♭	D♭	E♭m	Fm	G♭	A♭	B♭m	C°
D	D	Em	F♯m	G	A	Bm	C♯°
E♭	E♭	Fm	Gm	A♭	B♭	Cm	D°
E	E	F♯m	G♯m	A	B	C♯m	D♯°
F	F	Gm	Am	B♭	C	Dm	E°
F♯	F♯	G♯m	A♯m	B	C♯	D♯m	E♯°
G♭	G♭	A♭m	B♭m	C♭	D♭	E♭m	F°
G	G	Am	Bm	C	D	Em	F♯°
A♭	A♭	B♭m	Cm	D♭	E♭	Fm	G°
A	A	Bm	C♯m	D	E	F♯m	G♯°
B♭	B♭	Cm	Dm	E♭	F	Gm	A°
B	B	C♯m	D♯m	E	F♯	G♯m	A♯°

    - The similarity of each chord in the individual to the previous two chords in the individual. Where we:
      - Punish the individual for having more than two consecutive chords that are the same.
      - Reward the individual for having exactly two consecutive chords that are the same.

- **Selection technique:**
  - **Selection method:** Tournament.
  - The selection phase selects half of the population to remain, the other half we remove from the population.
  - **Selection process:**
    - We select half of the population based on a random tournament bracket, where random pairs of individuals are put ahead of each other.
      - The Winner gets added to the next population.
      - The Loser is removed fully.

- **Crossover technique:**
  - **Crossover method:** n-point crossover.
  - The crossover phase chooses two random parents from the selected individuals. And they mate and produce two children.
  - **Mating process:**
    - We iterate through the chords of the parents and do the following with a 50% probability:
      - Child1 gets the chord from Parent1,
Child2 gets the chord from Parent2.
      - Child1 gets the chord from Parent2,
Child2 gets the chord from Parent1.

- **Mutation technique:**
  - **Mutation method:** replacement of a chord with a random new one.
  - The mutation phase takes the new children produced by the crossover phase, mutates them, and then returns the mutated children.
  - **Mutation process:**
    - We iterate through the chords of the child and with a 5% probability:
      - we replace the current chord with a new random chord based on a random root and a random octave

- **Population Size:**
    - **Population size = 1024**
    - The population size is set before running the evolutionary algorithm, and at the end of each generation we get a new population with the same size.
    - The population size I found most suitable is 1024, because:
        - Less than that might lead to getting stuck in local maxima.
        - More than that would make the code very slow.
    - The population size needs to be a number that is divisible by two because:
        - The selection phase selects half of the population size
        - The crossover and mutation phases generates a new half of the population
    - If the population size is not an even number some errors might occur.

- **Stopping Criteria:**
    - Due to the fact that there is no clear definition of “perfect” music, I opted with the maximum number of generations method.
    - The maximum number of generation method makes the evolution run for a set number of generations and then terminates.

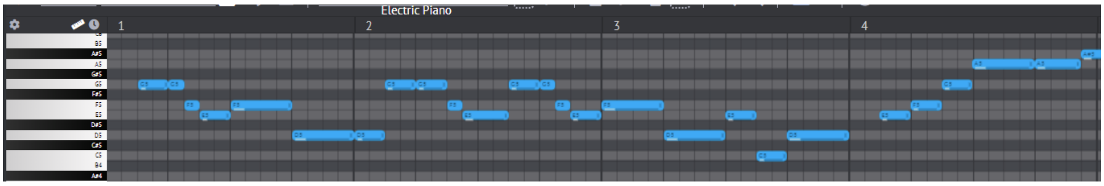
- **Number of generations:**
    - **Maximum number of generations = 5000**
    - The number of generations I found most suitable is 5000, because :
        - Less than that would not lead to a relatively good generation.
        - More than that would not lead to a significantly better generation.

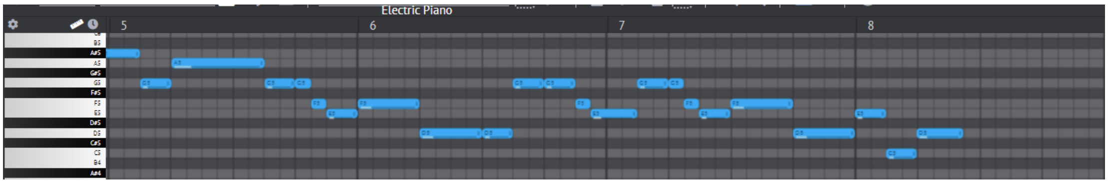
- **Generation Time:**
  - The generation time depends on:
    - Population size.
    - Length of individuals.
    - Number of generations.
  - On average the generation times for each input is:
    - Input 1:
      - Population size = 1024
      - Length of individual = 29 chord
      - Number of generations = 5000
      - **Generation time = 20 minutes**
    - Input 2:
      - Population size = 1024
      - Length of individual = 32 chord
      - Number of generations = 5000
      - **Generation time = 23 minutes**
    - Input 3:
      - Population size = 1024
      - Length of individual = 44 chord
      - Number of generations = 5000
      - **Generation time = 36 minutes**

Input - Output:

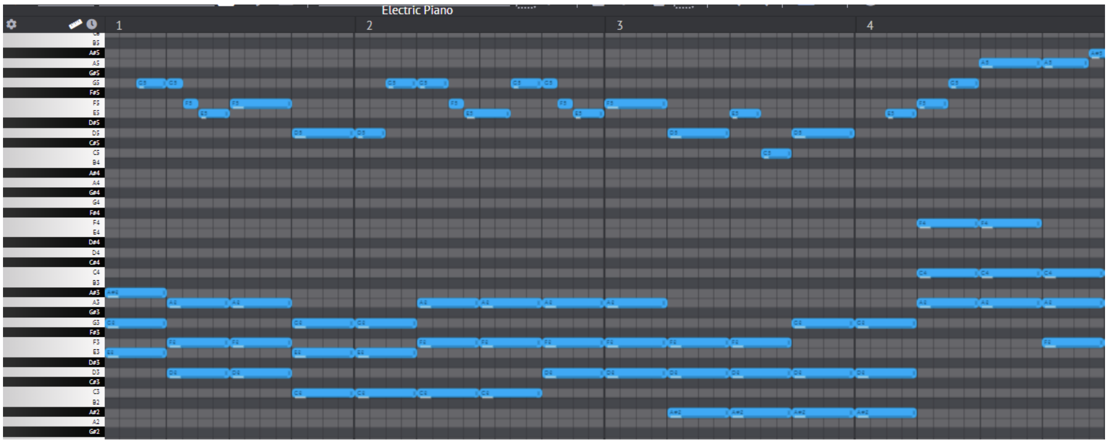
- **<u>Note: </u>** To listen to the output you can use [OnlineSequencer]

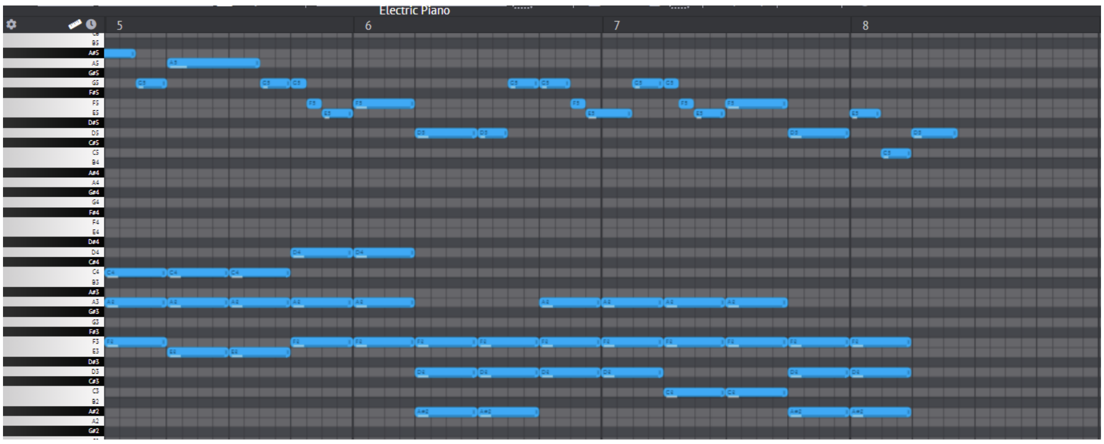
- **Input 1:**



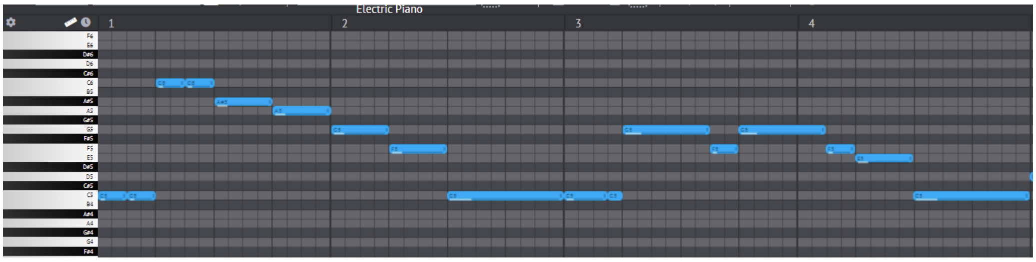


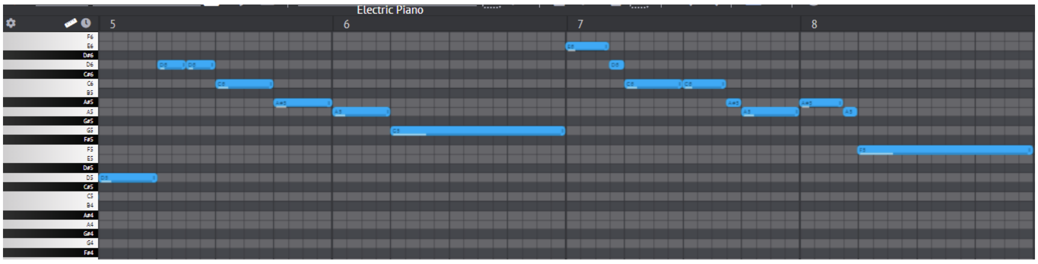
- **Output 1:**



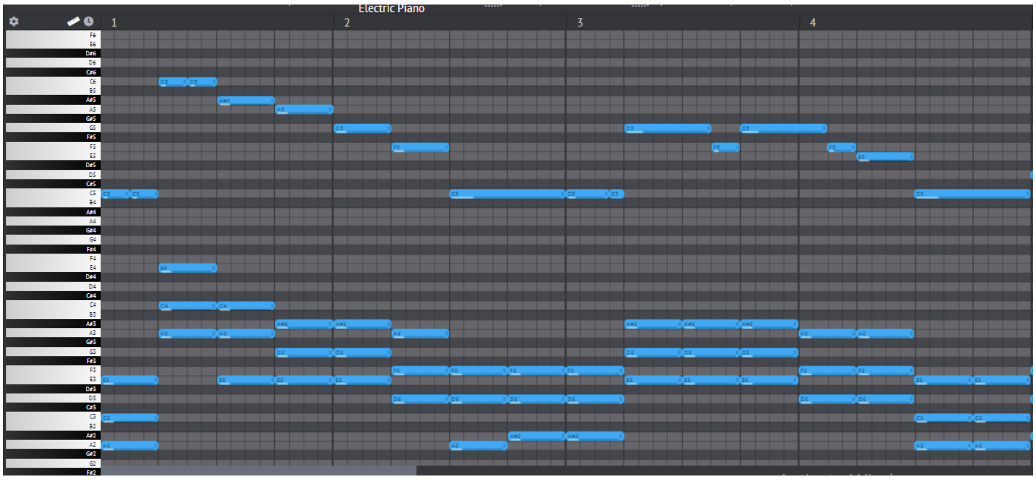


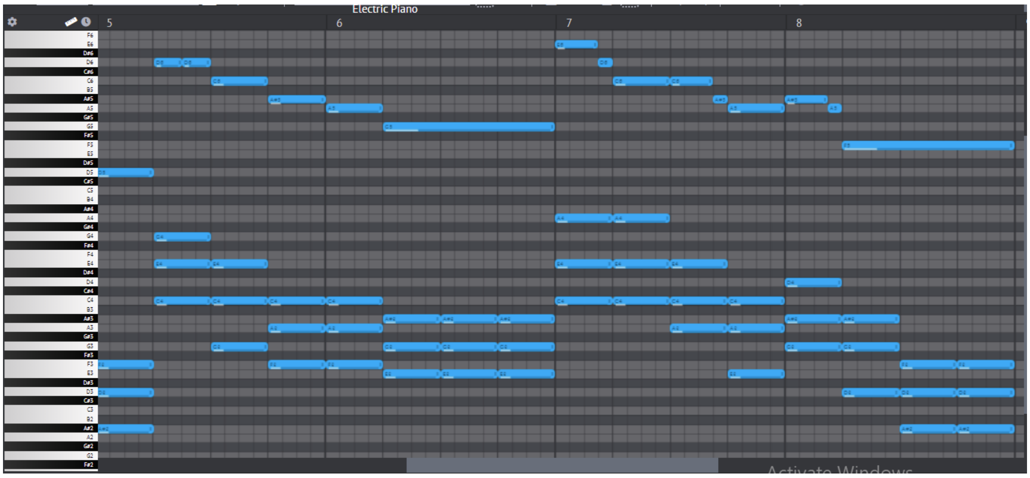
- **Input 2:**



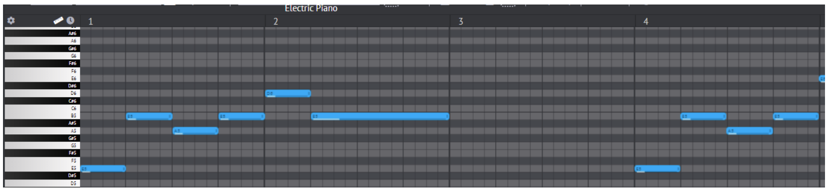


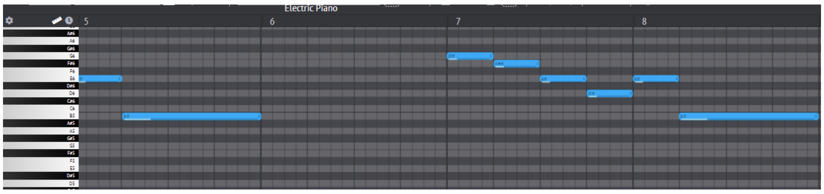
- **Output 2:**





- **Input 3:**





- **Output 3:**