The first part of this challenge is to find a way to easily represent the notes in the melody, luckily for us many years ago MIDI was created, so I used the numbers in MIDI to match every single note being, for example, 60 the central C. And at the beginning of the sequence of notes I set the key signature with the number of sharps or flats.

Will be rempresented as: **0# 57 59 60 62**

Lilypond

Going from music notation to this MIDI number representation with just one voice is quite easy (although I might work a hack for that also), but once I generate all the notes that form the chords the process ends up being a pain in the ass. Therefore, at the end I take the output and pass it through a little script that transform the numbers to Lilypond, generate a .ly file and an .jpg of the sheet.

The genetic algorithm

A genetic algorithm works pretty much like a species evolution in real life. You take a group of individuals, called population, in this population there are a few individuals that have the best attributes, those are the ones that will survive and carry on the genes to the next generation. This process continues generation over generation until there is an individual with the perfect attributes, or at least with the best so far. You can read more in Wikipedia.

The process can be structured in this step:

- Initialization
- Selection
- Crossover
- Mutation
- Termination

Let's tackle them one by one. But first let me explain the framework that I used.

Working enviroment

I used a library called DEAP, Distributed Evolutionary Algorithms in Python, it is a novel evolutionary computation framework for rapid prototyping and testing of ideas. It has all the basic tools to work with genetic algorithms you only have to create the functions to create, select, mate and mutate the individuals.

The file lily_template has the template to create the lilypond file and to give it format I used the string.Template class. Then I set some global variables that I'm going to use:

OPTIONS_* are the different roles that a note can have in a chord: The first note in the scale of the tonality can be the first note in the tonic chord, the third degree in the 6° chord or the fifth degree in the subdominant. So I represented this options as differences between the fundamental note of the possible chords and the note degree in the scale. This diferences change a little bit in a minor tonality.

MOD_* are just the grades of the chords in a major and a minor tonality.

Initialization

In the initialization part, you have to create the initial population with all the individuals. In this example an individual will be a chord progression, with all the chords beeing of four notes each. I represented each individual as a list of chords, and each chord a list of notes, for example:

This individual would be represented as [[48, 64, 67, 79], [53, 59, 65, 74], [55, 59, 62, 74], [48, 55, 67, 76]]

To do anything I need the tonality of the exercise, so I defined a function to find it given the bass voice and the key signature, it just looks at the key signature and it looks if the voice has any sight of being a minor key:

The function returns a tuple with the tonality represented as a number from 0 to 11 (C to B) and a 'M' if the tonality is major or 'm' if it's minor.

Then I wrote a function to create a single chord given the name of the chord and the fundamental note. The interesting part of this is that a chord of four notes can have multiple permutations, different set of notes and it can have a close or open harmony.

For example all these chords are D major and have the same bass:

After this function, I only need to select a chord for each of the notes in the bass. This process use the midi representation of the notes and making arithmetic operation with them and with a random choice between the possible options. At the end we have a complete chord progression. Now, DEAP requires a generator to create the individuals, so I just create a simple generator that yields each chord of the progression.

Selection

From Wikipedia: *"During each successive generation, a proportion of the existing population is selected to breed a new generation. Individual solutions are selected through a fitness-based process, where fitter solutions (as measured by a fitness function) are typically more likely to be selected."* So, I created a fitness function based on the classical harmony, it evaluate the progression to find "errors" like a distance between notes greater than an octave, or try to avoid two voices singing the same note:

Also between two chords, you have to avoid the consecutive fifths and octaves, and a normal person tends to make the intervals in a voice more *"natural"* making jumps not bigger than a fifth that often.

And for the evaluation of an individual I used this generator to access an element and the two neighbors.

Crossover

In the crossover section I used a simple One point crossover provided by DEAP in its set of tools.

Mutation

In the mutation section I just create a new chord for each one of the lasts chords that randomly pass a threshold.

The next function is just to create a lilypond file using the chords in the individual to see a nice sheet.

In the main function is where the actual algorithm's running, it's a simple evolutionary algorithm with a hall of fame where the best individuals will be saved. The little while loop is just if you want to run multiple times the algorithm until an individual gets an evaluation lower than, in this case, 15.

And at the end set up all the functions and the form of the individual in the toolbox that way DEAP can use them in the algorithm.

The program verbose each of the generations showing the number of individuals evaluated, the average evaluation value, the standard deviation, the minimum and the maximum. At the end it shows the three best individuals in all the evolution process, and it creates a lilypond file with the best of all.

However:

This is only if the input is ready but we have now midi files as an input in for this case I used some help from the following link to convert .mid files to arrays has length of the time of the midi file and width from 22 till 108 the piano keys.

So I convert it to plot then I wrote the notes with it’s period of time then choose the notes which I should represent the chord progression with.

I worked very hard for the code but it still has some bugs, so it will not work fully. However, I managed to get some input it is not that beauty but better than nothing.

The code is almost well constructed and readable but will not work in one time so I split it to get the 3 required outputs.

I repeat my apologies for late, please do not let me lose the hole grade.