Description of chosen generation strategy with pros and cons

I aim to create a random pattern that slowly turns into a recognisable rhythm and then slowly becomes random again.

Because of time constraints I chose to hand in the minimum viable product. Highlighted in yellow are the parts implemented in the code.

A Euclidean algorithm generates a pattern and stores it in a dictionary. Random deviation values are generated. A new random pattern is generated by adding a deviation (a positive or negative value) to every stored timestamp in the event dictionary. Every time the full cycle is played the deviation decreases. The number of cycles (generations) it takes before the random pattern becomes the same as the pattern generated by the Euclidean algorithm is decided by the user. The result can be saved as a MIDI file, and the settings configured by the user can be saved as a preset in a .txt file. Multiple Euclidean patterns can be played at once. There is no constraint on how many layers can play at once. The user can drag new samples into the soundFile directory, and these samples will show up as pickable options in the command line.

Pros:

- Because the generated pattern is not entirely random it won't have to be fixed to a grid to deliver an interesting sound experience for the user.
- Can easily be transformed into a MIDI file

Cons:

- Not very intuitive -> numbers entered aren't very self-explanatory (hard to predict what it will sound like)
- Configuration can be tedious -> A lot of user input is required before the pattern generation can begin.