autognomus' README

Germán Ruiz-Marcos

July 2021

0.1 Introduction

This is *autognomus*' repository. *autognomus* (as in **Au**tomatic **T**ension-**O**riented **Gen**erator **o**f **Mus**ic) is our publicly available music generation system capable of generating tonal music that has long-term structure and matches input tension profiles.

autognomus consists of four components or sub-systems, each of which tackles one of the following tasks:

- generating a main theme, consisting of a melody and an underlying harmony
- arranging the theme's harmony
- developing new material, similar to the theme, so that boredom is avoided and long-term structure is achieved
- morphing the theme and its developments, both their melody and harmony, so that they match changes in a given tension profile in real time

Notice that, in the tasks, the only one that concerns a real-time process is the last one.

Figure 1 presents *autognomus*' *TAnDeM* architecture (as in **T**hemer, **A**rra**n**ger, **De**veloper and **M**orpher).

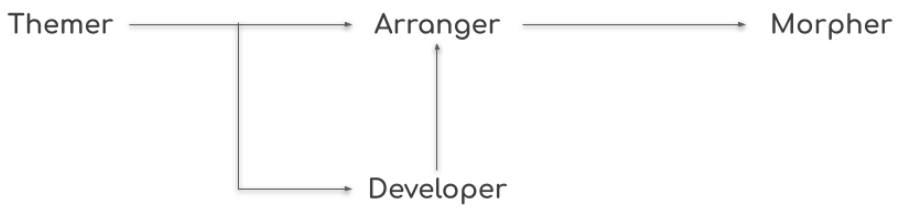

Figure 1: A diagram of *autognomus*' *TAnDeM* architecture.

As seen in Figure 1, *autognomus* consists of four sub-systems. The first sub-system, *Themer*, generates melodic and harmonic sequences which will be interpreted as the main theme in a given narrative. The second sub-system, *Arranger*, generates the appropriate arrangement of the voices in the harmony of the newly generated theme. The third sub-system, *Developer*, generates developments of the newly generated theme. The fourth and final sub-system, *Morpher*, transforms the generated theme so that it matches input tension profiles in real time.

In this repository, each sub-systems is implemented in a Jupyter Notebook.

0.2 How does *autognomus* work?

Let us imagine we want to use *autognomus* to automatically generate music for a real-time interactive experience, such as, for instance, a video game. How would we do this?

First, we would run *Themer*'s Jupyter Notebook. This sub-system would generate the main theme in our video game. To do so, we would have to feed the *Themer* with a collection of input parameters. These include the number of measures we want the theme to consist of, a tempo marking in beats per minute (BPM), the time-signature and key-signature to be used in the theme, and the character of the theme. *Themer* would generate a theme as a sequence of chord labels and a melody against them. In *Themer*'s interface, we can listen to the generated materials. We can also re-run some steps of the generation, in case we do not like what we are hearing. Once we are satisfied with the generated theme, *Themer* will save it so that it can be used by the rest of the sub-systems.

Second, we would run *Arranger*'s Jupyter Notebook. This sub-system would organise the voicing of the chords in the generated theme. We would have to feed the *Arragner* with *Themer*'s outputs. *Arranger* would then generate the corresponding arrangement of the chords in blocks and it will save it so that it can be used by the rest of the sub-systems. In the current version of *autognomus*, we cannot choose between different types of arrangements. This is something we want to implement in the future.

Third, we would run *Developer*'s Jupyter Notebook, but only if we want the generated theme to be developed. We would have to feed the *Developer* with *Themer*'s outputs. *Developer* would then generate a collection of developments (i.e. independent compositions) and it will save them so that they can be used by the rest of the sub-systems. The developments are twice as long as the input theme. That is to say, if the input theme lasts eight bars, each of the generated six developments would last sixteen bars. In order for the generated developments to be used in our video game, they would have to be arranged. That is to say, we would have to run each development through the *Arranger*, as described above in the second step.

Fourth, we would run *Morpher*'s Jupyter Notebook. We would have to feed the *Morpher* with *Themer*'s, *Arranger*'s and *Developer*'s outputs. *Morpher* would then loop the generated materials. It would play the generated theme twice, followed by one of the developments, and this process would repeat ad infinitum. In our video game, this is the music that would be played whenever we are in a safe place. Let us imagine that, in our video game, there are monsters who want to capture us. We could then quantify the degree of tension in the game as inversely related to the distance to the monsters. That is to say, the closer we get to a monster, the tenser, and vice versa. *Morpher* also considers a tension threshold. This threshold can be interpreted here as the minimum degree of tension for us to start to get worried about being captured by a monster. That is to say, if the current degree of tension is below the threshold, *Morpher* would interpret that we are in a safe place. And so it would continue playing the looped theme and its developments. However, if the current degree

of tension is above the threshold, *Morpher* would transform the theme or the development so that it matches the current degree of tension.

0.3 Where can you learn more about *autognomus*?

autognomus' implementation is partly based on our system *AuToTen*, which automatically calculates the degrees of tension in a piece of tonal music according to Fred Lerdahl's model of tonal tension. For more detail, see:

Ruiz-Marcos, Germán; Willis, Alistair and Laney, Robin (2020). Automatically calculating tonal tension. In: *The 2020 Joint Conference on AI Music Creativity* (Sturm, Bob and Elmsley, Andy eds.), 19-23 Oct 2020, Royal Institute of Technology (KTH), Stockholm, Sweden [On-line].
URL: `http://oro.open.ac.uk/72732/`

autognomus was developed as part of Germán Ruiz-Marcos' PhD research project. For more detail about what motivated *autognomus*, its implementation and evaluation, see:

Ruiz-Marcos, Germán (2021). Tension-driven Automatic Music Generation. PhD thesis The Open University.