Toward a Context Sensitive Music Generator for Affective State Expression

Marco Scirea[1] and **Peter Eklund**[2]
[1] Center for Computer Games Research
[2]Robotics, Evolution and Art Laboratory
IT University of Copenhagen, Denmark
{msci,petw}@itu.dk

Julian Togelius
Department of Computer Science and Engineering
New York University, NY, USA
julian@togelius.com

Abstract

We describe the architecture of a a music generator for games and the current state of its implementation. The objectives of the generator are: to develop the capability of expressing different affective states using a variety of AI techniques, being able to create music in real-time and finally to react in real-time to external stimuli. The project will be used in connection to research on Experience Driven Procedural Content Generation (*EDPCG*)(Yannakakis and Togelius 2011), especially in games. This project is the continuation of a previous project to construct a *Moody Music Generator* (Scirea, Cheong, and Bae 2014): a real-time procedural music generator developed using PD (Pure Data). While previous work could also express affective states(Scirea, Nelson, and Togelius 2015), the music generation did not take the temporal structure of music–such as chord sequences, leitmotifs, or improvisation–into account. With the new generator we aim at creating more interesting and expressive music, while still retaining the affective expressiveness of previous work.

Introduction

Music has the power to evoke moods and emotions–even music generated algorithmically. In fact, in many cases the whole purpose of a music generation algorithm is to evoke a particular mood. This is particularly true of music generators that form part of highly interactive systems such as computer games, where a common goal of dynamic music systems is to elicit a particular mood from the user on demand, as suits the current state of the game play. To take the example of a computer game, music generation can be seen as content within the experience-driven procedural content generation framework (Yannakakis and Togelius 2011), where the game adaptation mechanism generates music with a particular mood in response to player actions.

In games, unlike in traditional sequential media such as novels or movies, events unfold in response to player input. Therefore, the music composer in an interactive environment needs to create music that is dynamic while also being non-repetitive. Procedural generation of music content is a field that has received much attention over the last decade. While many games use some sort of procedural music structure, there are different approaches (or degrees), as suggested by Wooller *et al.*: *transformational* algorithms and *generative* algorithms (Wooller et al. 2005) In order to generate mood-based music, we use four musical features – **intensity**, **timbre**, **rhythm**, and **dissonances**, mainly inspired by Liu *et al.* (Liu, Lu, and Zhang 2003); see (Scirea, Nelson, and Togelius 2015) for a detailed explanation.

Architecture

The system is composed of three main parts: composition generator, real-time affective music composer and an archive of previous compositions. The archive maintains a database of all the previous compositions connected to the respective levels/scenes of the game. The archive allows persistence of compositions to be reused at a later time, but also allows us to calculate a measure of novelty of future compositions compared with what has already been heard. This database could also be expanded to connect compositions to specific characters, events, levels, etc.

The real-time affective music composer is the component that transforms a composition (see the following section for a detailed description of what we mean by composition) in the final score according to a specific mood or affective state that we want to express. In this paper we will focus on the composition generator, as it is already implemented.

Let's describe the functioning of the system when we enter a level/scene in a connected game: the system will first check if the current level appears in the archive, so if we already have a composition connected to the level. If so, the composition is passed to the real-time affective music composer, which creates a score out of the composition, with possible simple variations (to avoid monotone looping music, variations could be alterations, extensions or inversion of chords, small variations in the melody and so on) and using the mood altering music features to express the desired affective state. In the event we don't possess a composition connected to the level the composition generator is called, this will produce a composition that can be tested for novelty/similarity against one or all archived compositions. In this way we create associations between similar compositions and create compositions that sound significantly different from others the game player is already familiar with.

The system will also be able to react to game events, these events depending on the effect desired for example, a simple change in the affective state, a variation of the current composition or an entirely new composition.

Late-breaking abstract presented at The Sixth International Conference on Computational Creativity, June 2015

Composition Generation Currently, only the generation of *compositions* is implemented. By *composition* we mean a *chord sequence*, a *melody* and an *accompaniment*. It is noteworthy that accompaniment is only an abstraction and not a complete score of a possible accompaniment, we will describe it in detail in **Accompaniment Generation** below.

We decided to design the composition in such a way because we want to abstract what makes a music recognizable and gives it identity. By generating these parts, which lack some information that one would include in a classically composed piece of music (tempo, dynamics, etc.), we allow our system to modify the music played in real-time depending on the affective state to convey. The generation of compositions is a process with multiple steps: (i) create a chord sequence, (ii) evolve a melody fitting this chord sequence, and (iii) create an accompaniment for the melody/chord sequence combination.

Chord Sequence Generation Our method for generating a chord sequence is straightforward: we use a directed graph of common chord sequences and do random trajectories in this graph. We can specify various parameters of this sequence as: sequence length, first element, last element, chord to which the last element can resolve properly (e.g., if we specify that we want the last chord to be able to resolve in the V degree, the last element might be a I or a ii).

The interesting aspect of this graph is that it also shows us common resolutions to chords outside of the current key, this gives us a simple way of dealing with chord changes. Each chord can be interpreted as different degrees depending on which key we consider, so if we want a key change we can simply consider which degree the last chord in the sequence will be in the new key and follow the graph to return to the new key. This gives us good sounding key changes which do not sound too abrupt.

Melody Generation For melody generation we use an evolutionary approach. We defined a number of features to include and avoid in our melodies based on classical music composition guidelines and personal experience. We divided these features into constraints and objective functions. Accordingly, we use a Feasible/Infeasible two-population method (*FI-2POP*) with multi-objective optimization for the Feasible population. Given a chord sequence, a variable number of notes is generated for each chord, which will evolve without duration information. Once the sequence of notes is created we semi-randomly generate the duration of the notes.

FI-2POP The FI-2POP method consists in having two populations evolving in parallel, where feasible solutions are selected and bred to improve their objective function values while infeasible solutions are selected and bred to reduce their constraint violations. When breeding the new generation individuals are tested to see if they violate the constraints, if so they are moved to the Infeasible population, otherwise they are moved to the Feasible one. We have two constraints: a melody should *not have leaps between notes bigger than a fifth* and should *contain a minimum amount of leaps of a second* (50% in the current implementation).

Genome Representation The genome consists of a number of values (the number of notes we have to generate) which can express the notes belonging to two octaves of a generic key (so 0-13). We do not consider in this stage introducing notes not belonging to the key, as this will appear in later stages, when introducing variations of the composition to express affective states or chord variations.

Multi-Objective Optimization Fitness Function Three objectives compose the fitness functions: the melody should *approach big leaps (larger than a second) in a counter stepwise motion*, where the melody presents big leaps the *leap notes should belong to the underlying chord* and finally *the first note played on a chord should be part of the chord*.

Accompaniment Generation We include accompaniment in the composition because it is not only chords and melody that gives identity to music and expresses affect. We can generate accompaniment either from combinations of elements in a small archive or semi-randomly. The accompaniment is divided into two parts: a basic rhythm (a collection of note durations) and a basic note progression (for example an *arpeggio*). We can progress from the accompaniment representation to a score of the accompaniment by creating notes with durations from the basic rhythm and pitches from the progressions (offset on the current underlying chord). This basic score can be modified and slightly randomized for variety or for affect expression, while still maintaining a rhythmic and harmonic identity that will be characteristic of the composition.

Conclusions and Future (Ongoing) Work

We have described ongoing work on a context sensitive affect-expressing music generator designed to be used in games but that also could be used in other contexts. Compared to our previous generator, this development creates more complex music which might influence our current affective state expression theory. We intend to run a series of studies to explore these relationships.

References

Liu, D.; Lu, L.; and Zhang, H.-J. 2003. Automatic mood detection from acoustic music data. In *Proceedings of the International Symposium on Music Information Retrieval*, 81–7.

Scirea, M.; Cheong, Y.-G.; and Bae, B. C. 2014. Mood expression in real-time computer generated music using pure data. In *Proceedings of the International Conference on Music Perception and Cognition*.

Scirea, M.; Nelson, M. J.; and Togelius, J. 2015. Moody music generator: Characterising control parameters using crowdsourcing. In *Evolutionary and Biologically Inspired Music, Sound, Art and Design*. Springer. 200–211.

Wooller, R.; Brown, A. R.; Miranda, E.; Diederich, J.; and Berry, R. 2005. A framework for comparison of process in algorithmic music systems. In *Generative Arts Practice 2005 — A Creativity & Cognition Symposium*.

Yannakakis, G. N., and Togelius, J. 2011. Experience-driven procedural content generation. *IEEE Transactions on Affective Computing* 2(3):147–161.