SongSkill: A System for Continuous, Emotionally-Adaptive Music Generation

B.T. Franklin, BACS
Chief Research Engineer, Dunesailer Research Initiative, Phoenix, AZ, USA
www.dunesailer.science
e-mail: brandon.franklin@gmail.com

Abstract

The generation of music using software is not an especially new concept. There are many existing engines and approaches that are able to generate music at varying levels of quality, and in many different styles. At the same time, there are various tools and websites that assist in the selection of existing music based on mood or emotional content, and there has been a great deal of research into understanding how music impacts human emotions. However, until now, there has not been a sophisticated piece of software that can generate music based on the desired emotional impact.

Additionally, though much software has been written, and many techniques developed, to generate music in various forms, most of these do not produce results that would necessarily be appropriate for uses such as background music in games, at parties, etc.

The SongSkill music generation framework was created for this purpose. The SongSkill software generates music in a continuous manner, with the ability to adaptively respond to desired emotional parameters. This paper describes, in detail, the methodology and software structures used to construct SongSkill. Some of the especially difficult challenges faced (including, when possible, their solutions) are also described, along with future directions for further research.

1. Emotion Evocation vs. Emotion Conveyance

Understanding the relationship of a music generation system to a set of emotional parameters requires understanding the distinction between “emotion evocation” and “emotion conveyance”. Both of these are potential target goals of a music generator, and both have been demonstrated to have an observable (though, unfortunately, not always predictable) relationship to particular musical structures and techniques. The SongSkill framework is designed to run with configurations that utilize either of these approaches.

1.1 Emotion Evocation

Emotion evocation refers to the process of evoking a particular emotion in the listener. In other words, the music causes the listener to feel a certain way. Research into the details of this process has revealed that there is a core set of emotions that are most commonly evoked by music[1]. Similarly, there are some emotions, such as anger, that are difficult to directly and intentionally evoke in a listener.

Zentner et al., through the use of their Geneva Emotional Music Scales (GEMS) tests and associated research, have demonstrated a hierarchical grouping of the emotions that are most commonly evoked by music. They identify a collection of specific emotions which can be grouped into “factors”. Those factors can then be grouped into “superfactors”, of which there are only three. See Figure 1.

Emotional evocation is especially challenging because the process depends upon psychological factors that vary greatly among listeners[2]. For example, a song might remind a listener of a particular sad time in his or her life, thereby evoking the emotion “sadness”. However, a different listener might not have the same type of association with the music being produced, and therefore may not emotionally respond in the same way. It is nearly impossible for a composer (or music generator) to predict with any degree of certainty how a listener will respond to any given piece of music.

This is mitigated somewhat by the presence of various cultural expectations and cues that can be reasonably expected to be shared among the listening audience for a particular work. This is where the process of emotion evocation begins to cross over into emotion conveyance.

1.2 Emotion Conveyance

Emotion conveyance refers to the process of communicating a particular emotion (or set of emotions) in a musical piece. The emotional response of the listener is not important. The process of emotion conveyance is successful if the listener is able to correctly identify and understand the emotion that the composer was attempting to convey.

Because emotion conveyance does not depend upon any particular response in the listener, the full array of possible emotions are available in the composer’s palette. Emotions that cannot easily be evoked, such as despair, terror, and rage, can be directly communicated.

Any communication requires a common frame of reference for both parties, and this is where similarities can be found between emotion evocation and emotion conveyance. In both cases, the composer is utilizing a particular set of cultural expectations to represent emotional information, whether that information is to be perceived intellectually or viscerally by the listener. For example, Western culture

almost universally accepts the use of slow, quiet, minor keys as “sounding sad”.

Figure 1 - Emotions, Factors, and Superfactors from GEMS[1]

Superfactor	Factor	Emotion
sublimity	wonder	happy
		amazed
		dazzled
		allured
		moved
	transcendence	inspired
		feeling of transcendence
		feeling of spirituality
		thrills
	tenderness	in love
		affectionate
		sensual
		softened-up
	nostalgia	sentimental
		dreamy
		nostalgic
		melancholic
	peacefulness	calm
		relaxed
		serene
		soothed
		meditative
vitality	energy	energetic
		triumphant
		fiery
		strong
		heroic
	joyful activation	stimulated
		joyful
		animated
		dancing
		amused
unease	tension	agitated
		nervous
		tense
		impatient
		irritated
	sadness	sad
		sorrowful

This is where one interesting trick for emotion evocation comes into play: It is often the case that the emotion being conveyed by a song will be evoked, through a type of sympathetic response, in the listener[3]. Therefore, a potential approach to addressing the challenge of unpredictable emotion evocation is simply to focus on very effective emotion conveyance, and hope that the listener has a sympathetic response that evokes that goal emotion in him or her.

There are various structural features of music which are specifically associated with the conveyance of particular emotions[4]. These features are *tempo*, *mode*, *loudness*, *melody*, and *rhythm*. By combining these features with the emotions identified using GEMS, one can produce a matrix of specific attribute values that can be used to convey goal emotions, at least at the Factor level. See Figure 2.

Figure 2 - Emotion Conveyance Attributes

Emotion Factor	Tempo	Mode	Loudness	Melody	Rhythm
Wonder	slow/mid	major	mid	harmonious	smooth/varied
Transcendence	mid	major	mid	harmonious	smooth/varied
Tenderness	slow/mid	major	quiet	harmonious	smooth
Nostalgic	slow	any	mid/quiet	harmonious	smooth
Peacefulness	slow	major	quiet	harmonious	smooth
Energy	fast	any	loud/mid	any	varied
Joyful Activation	fast	major	loud/mid	any	varied
Tension	any	minor	varied	clashing	irregular
Sadness	slow	minor	mid	any	smooth/varied

Notably, I have chosen to focus on emotion conveyance and evocation at the Factor level instead of the specific emotion because the distinction between the required attributes at the more granular level is not very clear. For example, the specific differences between the attributes that evoke “calm” vs. “relaxed” vs. “serene” are unclear. Additionally, the large number of possible emotions makes the matrix quickly become unmanageable in addition to containing dubious assumptions. I have found that focusing music generation at the Factor level produces much more practical results due to clearer distinctions.

2. Implementation Language

The SongSkill framework is implemented entirely in the recently-introduced Swift programming language, version 3, and runs natively on macOS. Swift was selected for this purpose because it is an expressive, concise language with a heavy focus on

good object-oriented programming practices, code correctness, and performance. It also allows simple implementation of flexible UIs both now and in the future, and has built-in support for many front-end technologies such as sound production.

3. Playback Structure and Engine

The SongSkill framework includes an underlying MIDI-based playback engine called *MODIPlayer*. This engine uses MIDI for output, but organizes the structures to be played back into “tracks” and “rows”, following the approach popularized by the many module tracker programs in the 1980’s and 1990’s demoscene[5]. This is where the name “MODIPlayer” comes from; it is a combination of the terms “MOD” and “MIDI”.

The MODIPlayer supports multiple simultaneous *MODIChannels*, each of which is mapped to exactly one MIDI channel, and has exactly one active instrument from the standard set of MIDI instruments. There is a special MODIChannel dedicated to percussion, which maps to MIDI channel 10 as a special case (guided by the special role of this channel in the MIDI specification).

Each MODIChannel is provided at any point in time with at least one *NoteStrip* structure, which represents exactly one measure of music for only that channel. The NoteStrip is made up of a series of *Rows*, with each beat of music being represented by four Rows. In this way, each Row can be thought of as representing a sixteenth note of positional information.

Rows are used to store various *Operations*, and multiple Operations can exist within the same Row. At the time of this writing, the following Operations are supported:

- Do nothing
- Start note with specific pitch, velocity, and lifespan (in rows)
- Stop note of specific pitch
- Set tempo to a specific beats-per-minute rate

Additional instructions will probably be added in the future, such as an instruction to change the MIDI instrument that is currently associated with the channel. The presence of the “Do nothing” Operation allows a simple requirement to be met: Every Row must contain at least one Operation.

Because beats are represented by a specific, known number of Rows, tempo is implemented in a straightforward way by simply controlling the rate at which Rows are read and executed from their containing NoteStrip.

This playback mechanism can most easily be envisioned as being similar to the paper strips of music used to control a pianola, where notes to be triggered simultaneously occupy the exact same horizontal row, and the movement of the paper is always at a constant rate.

The MIDI engine that is used to produce the actual sounds is the AVAudioEngine that is included with Apple's AVFoundation framework. At the time of this writing, the engine is using the FluidR3_GM soundfont, which is available for free online[6], but future versions may support alternate or additional soundfonts.

4. Compositional Structure and Engine

The SongSkill framework uses multiple stages and layers of algorithms during the generative process. Broadly speaking, these are modeled after the sequence of steps that a human composer might use in creating a new piece. However, the open-ended nature of the system, as well as the goal of being able to create music endlessly, means that the generative process must be very dynamic and self-reflective, with early steps sometimes "setting the stage" for the steps that follow. Other steps look at what has already been generated to use as a basis for new structures.

4.1. Represented Musical Elements

The data structures manipulated by SongSkill provide an insight into its function. While not every musical concept is (or can realistically be) represented, this core set is adequate for the production of the vast majority of relatable music. The data structures are:

- Pitch (between 0 and 127, similar to the MIDI specification)
- Duration (expressed in quarterbeats, and therefore MODIPlayer Rows)
- Dynamic (between 1 and 127, similar to the MIDI specification)
- Moment (which has a Duration)
    - Note (a type of Moment with a Pitch, Duration, and Dynamic)
    - Rest (a type of Moment with a only a Duration)
- Measure (contains a collection of Moments)
- Scale
- Chord (expressed by degree: I, II, III, IV, V, or VI)
- Tempo (expressed in beats per minute)

Of these, the only data structure containing substantial logic is Scale, which is able to return Pitches dynamically for any represented musical scale. Scale modes that can be represented at the time of this writing are: Major, Major Pentatonic, Minor, and Minor Pentatonic. Scales beginning in any key are supported.

4.2. Stochastic Models

Several pieces of the SongSkill framework rely on stochastic models. To create a consistent set of representations, and ease the coding process, two data structures were created as utility classes.

The *ProbabilityGroup* class is instantiated with a collection of “buckets”, each of which is any object that implements the *Hashable* protocol built into Swift. Each bucket is associated with a percentage value in the range 1-100. A ProbabilityGroup requires that all of its bucket probabilities add up to 100. The API of ProbabilityGroup allows for a randomly-selected bucket to be retrieved at any time. The probability that any given bucket will be returned is the percentage value that was associated with it when the ProbabilityGroup was created.

The *WeightedProbabilityPile* class also uses buckets, but each bucket is instead associated with a “weight” value rather than a percentage. Buckets can be added, removed, and retrieved (without removal) from the pile at any time. The chance that any given bucket will be retrieved is proportionate to the relative size of that bucket’s weight value to the total weight values currently in the pile.

Both of these classes use pseudorandom numbers provided by the *arc4random_uniform()* utility function that is part of Apple’s Foundation framework.

4.3. Aesthetic Guides

As mentioned earlier, the SongSkill framework is able to support both emotion conveyance and emotion evocation goals. In fact, the framework is even more flexible than this, externalizing the rules for music generation into *AestheticGuides*. An AestheticGuide implementation has subsections called “chapters” that are each associated with, and provide guidance for, a specific phase or component of the generation process.

Any particular AestheticGuide implementation is based upon a specific musical goal for the generator. For example, there is an AestheticGuide for emotion evocation called *EmotionEvocationAestheticGuide*. Other implementations can be created, such as one for emotion conveyance, or one for the generation of dance music, or doom metal.

When any element of the framework can be configured with probabilities, options, or any other modifiable aspect of the generation process, it consults its AestheticGuide chapter.

4.4. Flow Generation

It is common for popular music pieces to follow structured patterns of repetition and variation. The incorporation of musical fragments and sections that are recognizable to the listener can enhance the listener’s ability to feel a sense of predictability, and therefore familiarity and comfort with the piece. Repetition has also been found to enhance the intensity of the listener’s emotional response to the music.[7]

However, the SongSkill framework is designed to generate continuous music, which

does not lend itself to some of the same structures used by popular music, such as the following examples:

- verse | chorus | verse | chorus | bridge | solo | verse | chorus
- intro | verse | chorus | verse | chorus | solo | verse | chorus | outro

These structures generally assume a beginning and end to a song. The idea of an "intro" or an "outro" is effectively meaningless in a continuous-generation context.

To accomplish the goal of a structured musical flow in a continuous manner, the SongSkill framework uses the concept of a *SongFlowPath*. This is a data structure that contains a *Pattern* that is made up of *Sections*. Sections each have a simple identifier (such as A, B, C, etc.) and a designation (none, prime, double-prime). Following musical tradition, the designation is a way of indicating that a Section is based very closely upon another Section, but with some modification. For example, A' (read "A prime") is very similar to, but not exactly the same as, A.

At the time of this writing, the following SongFlowPaths are supported by the generator:

- A B A B
- A A B A'
- A A' A A'
- A A B B A A B B
- A A B B A A C C
- A A' B B'

Each Section defines the identify of four measures of music. The generator creates each Section lazily, and only as needed, based on the current state of playback within the SongFlowPath. When playback has reached the end of the defined SongFlowPath, a new one is immediately generated, allowing the music to flow seamlessly into it and continue. This means that "A" in one SongFlowPath does not refer to the same "A" from the previous SongFlowPath, allowing the music to take advantage of repetition within a single SongFlow, but generate novel sections after the same sections have been reused adequately.

4.5. Compositional Memory

As mentioned above, when a new SongFlowPath is generated, the previously-generated Sections are discarded. However, there is value in maintaining and reusing some of the information generated during the creation of those Sections. As the music continues to be played and generated, the subtle (or not-so-subtle) use of some of the same note sequences and other musical characteristics that have already been used can create a sense of coherence, and deepen the sense of familiarity in the listener.

This is achieved through the use of the *SongSessionMemory*. There is a single instance of this class that persists throughout the entire music generation runtime. This works in a way philosophically similar to a Least-Recently Used (LRU) cache, in that musical elements which have been used the most recently are the most likely to be used again during the generation process, and elements that have not been used in the longest amount of time are the most likely to be "forgotten" by being discarded to make room for newer elements.

At the time of this writing, this is the area of SongSkill that is under the most intense research and development, and the exact elements that will be stored and reused are still being determined.

4.6. Part Generation Stages

The SongSkill framework generates music in a layered approach, with each layer being associated with one musical "part" in the formal sense. Each subsequent layer can then refer to the output of the previous layers to inform its own generation process. Relatedly, each layer can store and present specific metadata about the choices it made during the generation process, making analysis simpler for subsequent layers. This is analogous to musicians in a "jam session" who are each listening to one another and making adaptive choices as well as providing cues and hints to the other musicians indicating what will most likely follow.

As mentioned above, music is generated one Section at a time, so each of these layer generators is an implementation of the *PartGenerator* protocol, and the collective output is in the form of a *GeneratedSection* object which owns several data structures, each of which implements the *GeneratedPartSection* protocol.

4.6.1. Chord Progression Generation

At the time of this writing, section generation begins with the *ChordsPartGenerator*. This part is based on the concept of a section being driven by an underlying chord progression, which is an attribute shared by the majority of popular music.

Chord progressions are generated using stochastic models based upon the research into common chord progressions in popular music that has been done by Anderson et al.[8] in the production of the crowdsourced TheoryTabs website and associated Hooktheory book. By utilizing the public API provided on the site, I was able to produce a simple set of numeric probabilities indicating both how likely a given chord was to begin a progression and how likely any chord was to be followed by any other chord. Adjusting these values slightly based on my own knowledge to produce better results, I arrived at a straightforward stochastic model of chord transitions, which can be understood as a state machine. See Figure 3.

To determine the number and placement of the chords in a generated progression, the framework employs various implementations of *ChordsPartSectionDesigner*. By

doing an approximate visual analysis of the most common progression structures illustrated on the TheoryTabs website, I was able to determine five of the most common structures, and create implementations of ChordsPartSectionDesigner to build each of them. These structures are:

- One Chord per Measure
- Ahead of the One (where each chord begins slightly before the beginning of each subsequent measure)
- Split Chord (where only one measure has two chords, each occupying half the length of the measure, and the other measures each have one chord)
- Boosted Chord (where the chord of the last measure begins halfway through the previous measure, and all other measures have one chord)
- Single-Chord Dominant (where one chord extends through all the measures except for the last measure, which has a different chord)

Figure 3 - Chord Transition Probabilities

	I	II	III	IV	V	VI
I	0	5	5	30	45	15
II	15	0	10	20	25	30
III	5	10	0	40	10	35
IV	40	5	5	0	40	10
V	30	5	5	30	0	30
VI	15	5	5	40	35	0

The starting chord is shown on the left. Chords to transition to are shown along the top. Each value is expressed as a percentage chance of the transition occurring. Higher probabilities are indicated with darker color.

After the progression structure has been determined and the chords have been placed, an interpretation technique is selected. Techniques are represented using implementations of the *ChordTechnique* protocol. At the time of this writing, there are two completed implementations:

- Block Chord Technique (plays all the notes of a chord at once)
- Arpeggio Chord Technique (plays the notes of a chord in sequence, with various types of sequences available)

The selected technique is used to produce individual Note instances which are placed in the Measures of the *ChordsGeneratedPartSection*.

4.6.2. Melody Generation

Melody generation begins with the selection of a *MelodyPartSectionDesigner* implementation. At the time of this writing, the only two versions that have been implemented are a “freeform” designer and a “repetition” designer. The former creates a melody structure that spans the entire length of the section, whereas the

latter generates a shorter melodic structure and repeats it within the section design, allowing for small variations to add interest. No actual notes are generated during this design stage.

After a design has been finalized, an implementation of *MelodyTechnique* is selected to generate the actual notes of the melody. At the time of this writing, the only two technique implementations are “all scale degrees” and “pentatonic”, with the former allowing any note in the scale, and the latter restricting the note choices to the pentatonic scale within the current key.

The selected technique is used to generate the notes of the melody through the use of a melody-specific stochastic model. The starting note of the melody is selected using probabilities shown in Figure 4. The duration of the note is determined using the probabilities shown in Figure 5. Each subsequent note is selected based on a relative movement within the current scale, using the probabilities shown in Figure 6.

Figure 4 - Starting Notes of Melody

Starting Scale Degree	All Degrees Probability (%)	Pentatonic (Major) Probability (%)	Pentatonic (Minor) Probability (%)
1	30	30	35
2	5	5	--
3	20	20	20
4	5	--	20
5	30	5	5
6	5	30	--
7	5	--	--

Figure 5 - Duration of Notes in Melody

Duration (beats)	Probability (%)
3	10
2	20
1	30
1/2	25
1/4	15

Figure 6 - Note Transition Probabilities in Melody

Interval	All Degrees Probability (%)	Pentatonic Probability (%)
3	3	--
2	7	5
1	15	20
0	50	50
-1	15	20
-2	7	5
-3	3	--

After all the notes have been selected for the length of the Section, a final "enhancement" pass is made, adjusting notes that occur on a beat that are not on a stable degree to the nearest stable degree. This is a technique recommended by Anderson et al.[9] in the book, *Hooktheory I*, for producing much more pleasing melodic lines.

4.6.3. Percussion Generation

The percussion generation stage focuses on generating rhythmic accompaniment using sounds approximating a drum set. As described above, this takes advantage of the nature of MIDI channel 10 as a percussion-only channel, since this channel allows multiple types of instruments to be played in parallel. The pitch value provided for any given note determines what percussion instrument is played by that note.

The *PercussionPartGenerator* uses the current musical goal to select a protocol implementation of the *PercussionInterpretationGenerator* protocol. The generator selects an array of percussion techniques to apply to the channel, organized by the specific instrument that the technique represents. For example, here are some of the percussion technique implementations at the time of this writing:

- *SimpleKickDrumPercussionTechnique*
- *SimpleSnareDrumPercussionTechnique*
- *SimpleHiHatPercussionTechnique*
- *SimpleRideCymbalPercussionTechnique*

Only techniques that are appropriate for the current *GeneratedSection* are applied, and each implementation is interrogated to identify for itself whether or not it is appropriate. Each appropriate technique uses its own distinct algorithm to produce a resulting sequence of Notes, and adds them to the Measure in a layered manner, with each subsequent technique adding Notes for its specific instrument.

5. Noteworthy Challenges and Difficulties

Perhaps the most difficult challenge encountered overall is the process of specifically identifying and replicating the musical attributes that relate to an exact target emotion factor. It is, somewhat surprisingly, relatively easy to generate music that "sounds good" to most listeners, but shaping that process such that it evokes specific emotional responses in the same listeners is much less clear-cut. Frequently, the process involves a great deal of trial and error, and I am not aware of any research that provides exact musical parameters for the evocation of particular emotions across the entire spectrum. In fact, such exact parameters may not even exist, known or not.

Other challenges have been of a more technical nature. For example, even with a very well-defined Note and Measure API, the process of interrogating existing

musical information to build a different musical part upon is complex and laborious. This is exacerbated somewhat by the fact that there is no “one size fits all” solution for the generation of musical parts. Each part is generated in a different way, and as a result, employs unique data structures to represent output. This can, unfortunately, introduce implementation-specific dependencies in PartGenerators that follow and depend upon earlier ones.

One challenge that was encountered well into the development process was the ability to generate music for aesthetic goals other than the evocation of emotions. Other goals, such as generating only a specific genre of music, or generating music to convey (rather than evoke) a particular emotion, seem to be just as valuable, but the SongSkill framework was not originally imagined as being able to support them all. This problem was solved with the introduction of the AestheticGuide concept, which externalized the aesthetic goals and made the framework much more general.

Challenges on the horizon include, but are not limited to:

- Very robust musical memory
- Background threading, to allow music generation while heavy UI or graphics use is in progress
- Discovering a general style of open-ended interpretation for chords, rather than specific hard-coded note sequences

6. Future Directions

I expect SongSkill to be under development and growing in capability for years to come. There is a long list of areas of improvement and expansion, some of which are:

- Broadening of support for different emotion factors
- Introduction of genre-specific interpretations, and the ability to dynamically “remix” these with one another
- Support for switching instruments during the song generation process
- More PartGenerators, such as bass, decorative effects, and atmospheric pads
- Various commercial applications that use SongSkill to provide music for specific scenarios (yoga, fear of flying, house party, retail ambiance, etc.)

References

[1] M. Zentner et al., “*Emotions Evoked by the Sound of Music: Characterization, Classification, and Measurement*”, American Psychological Association, Emotion, 8(4), 2008

[2] Scherer, K. R.; Zentner, M. R. (2001). "Emotional effects of music: production rules". *Music and Emotion: Theory and Research*: 361–387.

[3] Hunter, P. G.; Schellenburg, E. G.; Schimmack, U. (2010). "Feelings and perceptions of happiness and sadness induced by music: Similarities, differences, and mixed emotions". *Psychology of Aesthetics, Creativity, and the Arts*. 4: 47–56.

[4] Gabrielle, A.; Stromboli, E. (2001). "The influence of musical structure on emotional expression". *Music and Emotion: Theory and Research*: 223–243.

[5] Ranjan Parekh (2006). *Principles of Multimedia*. Tata McGraw-Hill. p. 727. ISBN 978-0-070-58833-2.

[6] "SoundFonts to try." *SynthFont* www.synthfont.com/soundfonts.html. Accessed 5 Nov. 2016.

[7] Ali, S. O.; Peynircioglu, Z. F. (2010). "Intensity of emotions conveyed and elicited by familiar and unfamiliar music". *Music Perception: An Interdisciplinary Journal*. 27: 177–182.

[8] Miyakawa, Ryan, Dave Carlton, and Chris Anderson. "Famous Chord Progressions." *TheoryTabs*, Hooktheory LLC, https://www.hooktheory.com/theorytab/common-chord-progressions. Accessed 5 Nov. 2016.

[9] Miyakawa, Ryan, Dave Carlton, and Chris Anderson. "3.3 Stable vs. Unstable Degrees." *Hooktheory I*, Hooktheory LLC, 5 June 2012. Accessed 5 Nov. 2016.