Fall 12-14-2018

VARIATIONS ON A THEME: Using Amino Acid Sequences to Generate Music

Aaron Kosmatin
San Jose State University

VARIATIONS ON A THEME:

Using Amino Acid Sequences to Generate Music

A Thesis

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfilment

of the Requirements for the Degree

Master of Science

by

Aaron J Kosmatin

December 2018

The Designated Thesis Committee Approves the Thesis Titled

VARIATIONS ON A THEME:

Using Amino Acid Sequences to Generate Music

by

Aaron J Kosmatin

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSE STATE UNIVERSITY

December 2018

Dr. Sami Khuri, Department of Computer Science

Dr. Katerina Potika, Department of Computer Science

Dr. Mike Wu, Department of Computer Science

ABSTRACT

VARIATIONS ON A THEME:

Using Amino Acid Sequences to Generate Music

by Aaron J Kosmatin

In this project, we explore using a musical space to represent the properties of amino acids. We consider previous mappings and explore the limitations of these mappings. In this exploration, we will propose a new method of mapping into musical spaces that extends the properties that can be represented. For this work, we will use amino acid sequences as our example mapping. The amino acid properties we will use include mass, charge, structure, and hydrophobicity. Finally, we will show how the different musical properties can be compared for similarity.

Contents

List of Figures

List of Tables

1. Overview

Proteins are typically represented as strings of characters encoding amino acids. It is a well-known fact that humans process knowledge and information in a variety of different ways; one of them being auditory. Over the years, researchers have acknowledged the fact that there are significant similarities between the structures of proteins and music. Both structures are composed of phrases organized into themes

In this project, we create a bijective mapping between an amino acid space and a music space. The purpose for this mapping is threefold. We have created a music space, in which amino acid sequences can be compared and interesting subsequences can be found, that preserves the properties of the amino acids. By preserving the properties of amino acids in the mapping to a musical space, we create aurally similar music for similar amino acids. We have created a space that allows a person to hear the two amino acid sequences' convergences and divergences. The project is broadly appealing to persons of different academic backgrounds and interests. This project incorporates aspects of biology, music, chemistry, and computer science.

1.1 Limitation of Character Mappings

Amino acids are chemical compounds which have physical properties. Amino acids are typically represented as single letters. While this representation is com-

pact, it fails to preserve the physical properties of the amino acids. Isoleucine, represented as 'I', and Leucine, represented as 'L', are very closely related amino acids, but this relationship is lost when represented as 'I' and 'L'. It may be thought that 'I' and 'L' are alphabetically close, but Lysine, represented as 'K', sits alphabetically in the middle but shares few properties with Leucine and Isoleucine.

Our mapping will represent four amino acid properties. The first property is size: tiny, small, and regular. The second property is hydrophobicity: hydrophilic and hydrophobic. The third property is polarity: non-polar, polar, positive, and negative. The fourth property is structural: aromatic and aliphatic. In Figure 1.1, we can see T, Threonine, is a small, polar, hydrophobic amino acid, that is neither aliphatic nor aromatic.

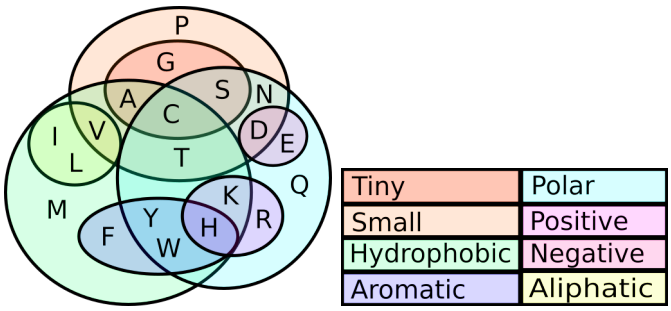


Figure 1.1: Amino Acid Properties

1.2 Previous Work

Several previous mappings from amino acids and DNA bases to music exist. Takahashi and Miller [1] map each amino acid into a chord instead of a single note. In an effort to preserve the chemical and physical properties of amino acids, they group similar amino acids into chord inversions. They make use of first inversions only. For example, Leucine is assigned to an A minor chord, and Isoleucine, a closely related amino acid, is assigned to the first inversion of the A minor chord. Similarly, Arginine is assigned the A minor an octave above, and Lysine is assigned

to its first inversion. The full mapping can be seen in Table 1.1. In this way, the authors are able to assign all 20 amino acids to 13 chords and 7 inverted chords. Additionally, Takahashi and Miller use the chord duration to represent each amino acid's natural frequency of occurrence.

Table 1.1: Takahashi and Miller Mapping

Chord Root	Major Triad	First Inversion
C	Trp	
D	Met	
E	Pro	
F	His	
G	Tyr	Phe
A	Leu	Ile
B	Val	Ala
C	Cys	
D	Gly	
E	Thr	Ser
F	Gln	Asn
G	Glu	Asp
A	Arg	Lys

Ohno and Ohno [2] use a cyclical assignment, such that each DNA base is assigned two notes in each octave. Cytosine maps to middle C; Adenine maps to either D or E; Guanine maps to either F or G; Thymine maps to either A or B; and Cytosine maps to either C or D in the next octave. In this manner, they are able to transcribe from DNA sequences to melodies and from melodies to DNA sequences. For each DNA sequence, there are many melodies that can be created, but each melody will have only one DNA sequence. While they make use of rhythm, it is arbitrarily assigned, and does not reflect any properties of the amino acids.

Gena and Strom [3] determine pitch through equations that use the amino acids' acidity, dissociation constant, base pair composition, molecular weight, and hydrogen bonds. The intensity (volume) of each note is determined by the melting temperature and hydrogen bond strength of the respective amino acid. The note

duration is determined by the dissociation constant and molecular weight of the amino acid.

To varying degrees, the prior mappings have similar musical limitations. We use the Takahashi and Miller[1] to explain them.

1.3 Inherent Musical Limitations of Previous Mappings

Musically, there are two problems with the Takahashi and Miller mapping. First, it fails to follow the beat distribution associated with the music's time signature. Second, it fails to distribute notes consistent with scale expectations.

1.3.1 Rhythm

Neither altering the note durations, without regard for the rhythm signature, nor using a single note duration and emphasis, create a sense of rhythm. Previous musical mappings map into 4/4 time. Time signature 4/4 time is the most common signature in western music, and has a very specific feel to it. The 4 beats are not equally important in the 4/4 time signature. Typically, in 4/4 time, the first beat of the measure is the most important, followed by the third. These two beats are called the down beats. The up beats, the second and fourth beats of the measure, get less emphasis than the down beats. While Takahashi and Miller wrote their music in 4/4 time, it does not follow the 4/4 convention.

By analyzing the note distribution in Takahashi and Miller's Human Thymidylate Synthase A composition, represented in Figure 1.2, we can see that there is some variance in where the notes land. The figure clearly shows that the rhythmic emphasis is unexpected and does not abide by the typical rhythmic distribution. This type of analysis holds true for all rule-based rhythmic assignments that don't

1	1-and	2	2-and	3	3-and	4	4-and
12.5%	19.5%	16.6%	7.7%	8.9%	16.9%	9.6%	8.3%

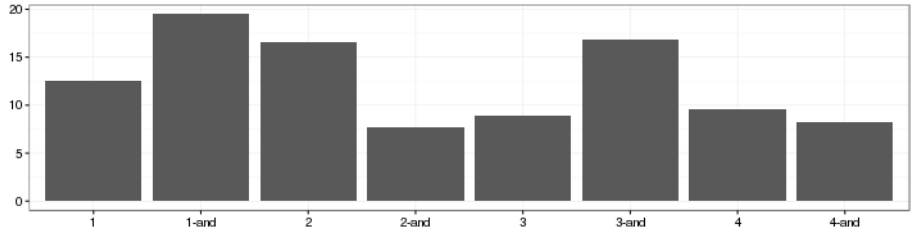


Figure 1.2: Beat Distribution in Takahashi and Miller Mapping of Human Thymidylate Synthase A

incorporate musical measures. Notes will start half a beat after the beat as often as they start on the beat. No special emphasis will be assigned to the down beats, or the beginning of measures.

1.3.2 Notes

Notes in western music are based on standing waves. If we think of a guitar, plucking the E string will give the first standing wave with 0 nodes and corresponds to a low E. The second standing wave on the E string has a node on the 12th fret, and corresponds to an E an octave higher. The third standing wave, with nodes on the 7th and 19th frets, corresponds to a B. The fourth standing wave, with nodes on the 5th fret, 12th fret, and beyond the fingerboard, corresponds to an E two octaves above the original. The fifth standing wave, with nodes on the 4th fret, 9th fret, 16th fret, and beyond the fretboard, corresponds to the G#. These standing waves are fundamental to western notes. A twelve-tone (chromatic) scale is the minimum number of pitches required to play these notes in the same octave with even tonal spacing.

A scale can be thought of as representing a contract between the composer, the player, and the listener. The scale denotes information on which notes will be played, but also connotes information about how different notes are important. Typically, the root, the fifth, and the third notes in the scale are the most im-

c	d	e	f	g	a	b
14.1%	13.8%	16.1%	10.3%	13.4%	16.6%	15.7%

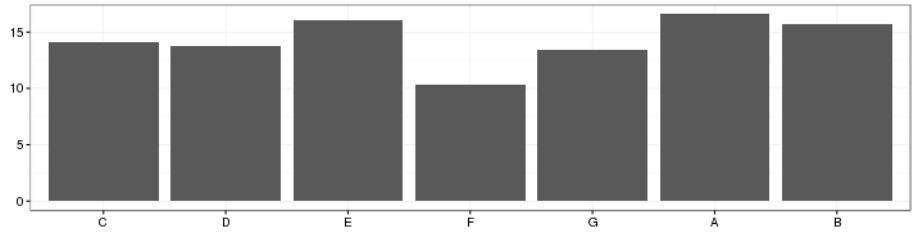


Figure 1.3: Note Distribution in Takahashi and Miller Mapping of Human Thymidylate Synthase A

portant, in that order of importance. This differing emphasis on the notes of the scale is what makes a piece in A minor sound distinctive from a piece written in C major. Although A minor and C Major share the same notes, the emphasis placed on those notes is different.

In their mapping, Takahashi and Miller use thirteen different chord root tones, along with seven inverted chords, as can be seen in Table 1.1. This large number of chords is atypical of most music. Due to the large number of chords, there is little emphasis on specific notes of the scale. The note distribution can be seen in Figure 1.3.

In the next chapter, we describe the four-dimensional musical space in which we map the 20 amino acids.

2. Musical Space

A new musical space is required to expand the number of amino acid properties that are captured in a mapping. We need at least four dimensions to represent the chosen properties of the amino acids. We could use the dimensions of pitch, duration, timbre, and dynamic, but these will lead to the previously discussed musical problems. A chordal mapping is more promising, but ultimately, insufficient to achieve musicality. We chose to map amino acids into musical measures, where different amino acid properties are represented by different musical parts.

2.1 First Mapping

Our first mapping was ultimately unsuccessful, but is important in understanding how we were able to improve to a better musical mapping. We followed the example of previous research in this area and used chords to represent the amino acid. We used G Major, C Major, and D Major chords, their relative minors, E minor, A minor, and B minor, respectively, and a Dsus4. Lastly, we used augmented notes at the top of the chords, dominant 7th, major 7th, dominant 7 sharp 9, and no augmentation. This provided 7 root chords and 4 augmentations, providing a space that could contain 28 items. To group the amino acids, we grouped hydrophobicity into either major or minor chords, we grouped charge by relative major or minor, and used the augmentation notes to differentiate between amino acids within these groupings.

While this mapping was interesting, it was too phonically dense. It became difficult to differentiate what chord was being played, along with its augmented note. For example, a G Major 7th chord (notes GBDF#) is very similar to a B minor chord (notes BDF#). To address this problem, we spread the notes out into three beat measures with the first beat being the major triad of the chord, the second beat being the augmentation of the chord, and the third beat being the full chord. A G Major 7th chord would be GBD on the first beat, F# on the second, and GBDF# on the third, while a B minor chord would be BDF# on the first beat, b on the second, and BDF# b on the third. This helped create a rhythmic structure, similar to a waltz, and allowed the user to listen over several beats for the important information in the measure. While this was an improvement, we became aware that we were now mapping into measures, which change the musical properties available to us.

Next, we discuss each of the four dimensions in detail.

2.2 Current Mapping

Using measures as the musical space provides better outcomes than using pitch and duration. This allows much more control of musical properties. Measures allow better rhythm assignment and note selection; they expand the number of properties we have available; and they line up such that multiple sequences can be played in tandem. For our mapping, we chose to follow pop-music conventions. We split up the music into a four part composition containing drum, chordal, melody, and bass parts. These four parts comprise the four dimensions required for our mapping. Additional dimensions may be added, but were unneeded for our mapping. Possible additional dimensions include chordal rhythm, harmony, and multiple chords per measure.

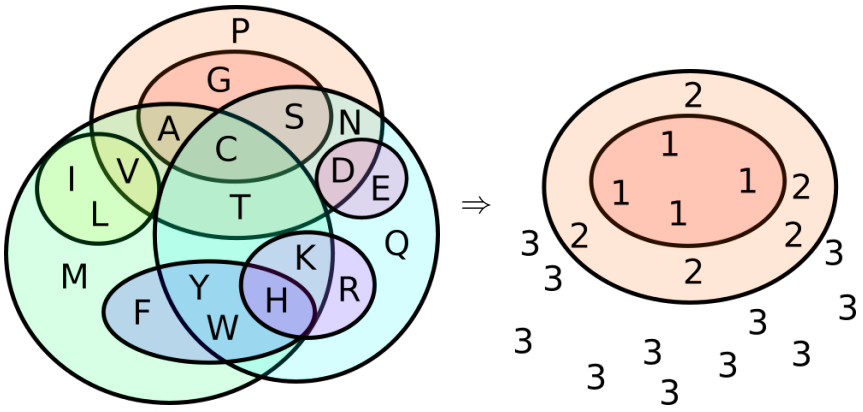


Figure 2.1: Drum Mapping

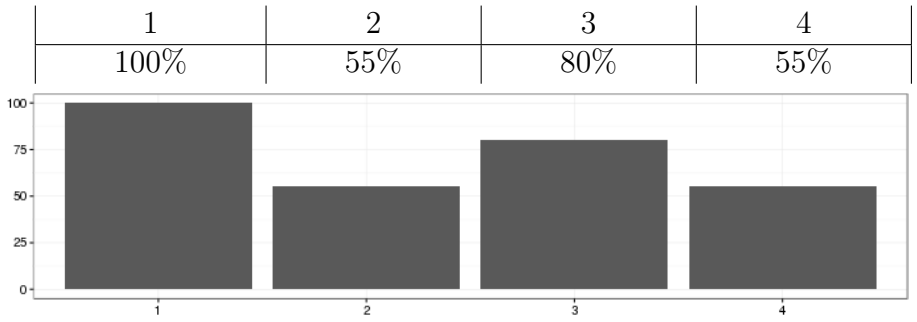


Figure 2.2: Analysis of Beat Distribution in New Mapping

2.2.1 Drum Mapping

The drum rhythm was the first part of the mapping we incorporated. We created three drum tracks for the amino acids all using bass drum strikes as seen in Figure 2.1. It was intuitive to us to use the drum track to represent the size of the amino acid. Rhythm is very important in imparting a sense of weight to music. Fewer drum strikes are associated with a lighter feel, more drum strikes are associated with a heavier feel. Tiny amino acids are assigned a drum strike on the first beat of the measure; small amino acids are assigned drum strikes on the first and third beats of the measure; regular amino acids are assigned drum strikes on all four beats of the measure.

When we do an analysis of the drum strikes, Figure 2.2, we can see we get a much more natural distribution of struck beats. There are other factors to consider to decide if we have a more natural rhythmic structure. All of our chord changes occur on the first beat of the measure, our chordal part uses standard rhythmic patterns, our melody uses straight quarter notes, and the bass plays 1-and-2, 3-and-4. All of these features match a standard 4/4 time signature.

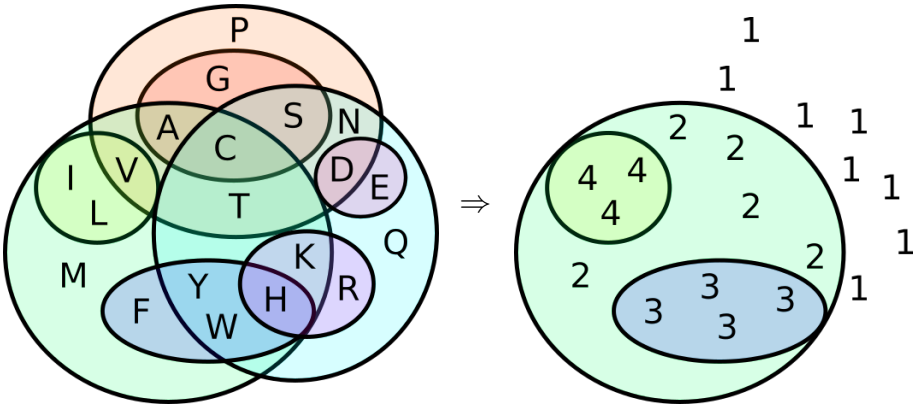


Figure 2.3: Chordal Mapping

2.2.2 Chordal Mapping

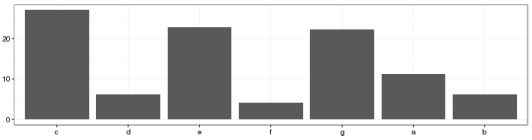
The chordal mapping is important for setting the style and feel of the music. Recall that by using Roman numerals, triad chords in any major key are given by: I-ii-iii-IV-V-vi-vii°-I, where uppercase numerals (first, fourth, and fifth degrees) represent major chords, lowercase numerals represent minor chords (the second, third, and sixth degrees), and ∘ represents a diminished chord (the seventh degree). We chose the pop-music progression convention due to its relatively simple structure and standard chord patterns. The I-V-vi-IV chord progressions, and its variants, are very common in pop-music. In the key of C, these chords are C

(with notes: c,e,g), G (with notes: g,b,d), Amin (with notes: a,c,e), and F (with notes: f,a,c), respectively. As an illustration of the popularity of this pattern in western music, the chordal parts of several songs and one work in classical music, transposed to C, can be seen in Table 2.1.

Although we could have used 20 different chords, one for each amino acids, we only used 4. Besides reducing the chordal space, we also avoid some troublesome chords, such as the B minor with a flat 5 (Bmb5).

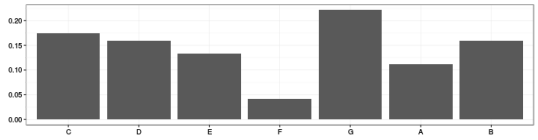
Next, we have to create the mapping of amino acids into chordal music, i.e., which amino acid receives which chord. Figure 2.4a shows the distribution of notes by the chords for mapping the C chord to the hydrophilic amino acids, and assigning one of the remaining chords (G, Amin, or F) to the hydrophobic amino acids. Figure 2.4b shows the distribution of notes by the chords for mapping the G chord to the hydrophylic amino acids, and assigning one of the remaining chords (C, Amin, or F) to the hydrophobic amino acids. The C chord assigned to hydrophilic amino acids gives a better distribution of notes and higher musicality, but the G chord works better for the similarity matrix (as seen in the next Chapter). For this project, we will assign the G chord to the hydrophilic amino acids, and the C chord to the hydrophobic amino acids, as the perceived change in musicality when playing the sequences is minor.

c	d	e	f	g	a	b
27.1%	6.3%	22.9%	4.2%	22.2%	11.2%	6.3%



(a) Chordal Note Distribution with C Assigned to Hydrophilic Amino Acids

c	d	e	f	g	a	b
17.5%	15.9%	13.3%	4.2%	22.2%	11.2%	15.9%



(b) Chordal Note Distribution with G Assigned to Hydrophilic Amino Acids

Figure 2.4: Analysis of Note Distribution in New Mapping

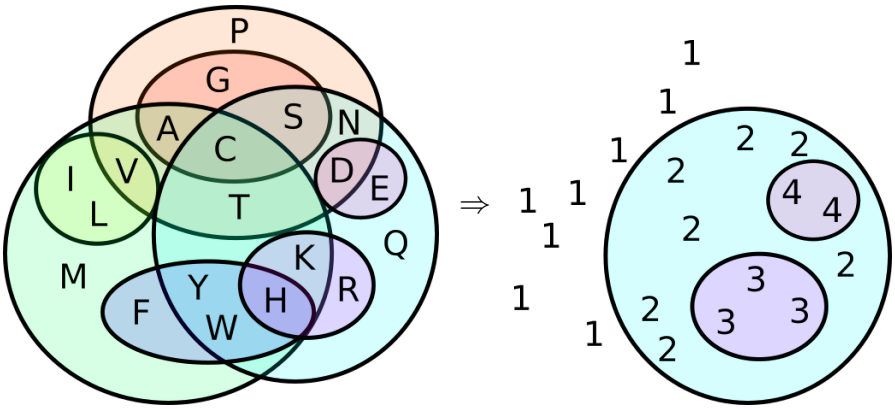
Table 2.1: Popular Songs Composed of Four Chords
Transposed to the Key of C

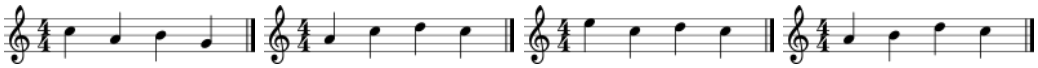
Year	Artist	Song	Part	Chords
c. 1706	Johann Pachelbel	Canon in D[4]	All	C G Am Em F C F G
1970	Beatles	Let It Be[5]	Verse	C G Am F C G F C
			Chorus	Am Am F C C G F C
1985	Journey	Don't Stop Believin'[5]	Verse	C G Am F C G Em F
			Chorus	F C
1987	U2	With or Without You[5]	All	C G Am F
1995	Nine Inch Nails	Hurt[5]	Chorus	Am F C G
2004	Old Crow Medicine Show	Wagon Wheel[5]	All	C G Am F C G F
2008	Lady Gaga	Poker Face[5]	All	C G Am F

2.2.3 Melody and Bass Mapping

The melody represents the charge properties of the amino acids. As can be seen in Figure 2.5, there are four melodies, all using quarter notes. By design, the melodies are kept very simple. They must be interchangeable; each melody is able to follow after any other melody, and also can be played over any chord.

Similarly, the bass parts are very simple. Musically, we use the bass to accent the chords, and provide additional rhythmic structure. In the mapping, the bass line is used to differentiate between amino acids that are not differentiated by any other properties of their amino acids, specifically, Leucine and Isoleucine, and Tyrosine and Tryptophan. For all amino acids other than Leucine and Tryptophan, the bass line plays only the root note of the chord. For Leucine and Tryptophan, the bass line alternates between the first and fifth of the chord. Appendix B, Sheet Music of Each Amino Acid, gives a detailed description of the mapping of each of the 20 amino acids into music.





(a) Melody 1 (b) Melody 2 (c) Melody 3 (d) Melody 4

Figure 2.5: Melody Mappings

2.3 Mapping Conclusion

By redefining the musical space that we are mapping into, we are able to improve on current musical mappings. Our mapping follows common music patterns, increasing its musicality. By using familiar pop-music structures, we are better able to engage the listener, and easily expand the number of dimensions available in the musical space. By using measures, which have a constant length, we are able to compare multiple amino acids to each other. The full mapping can be seen in Figure 2.6.

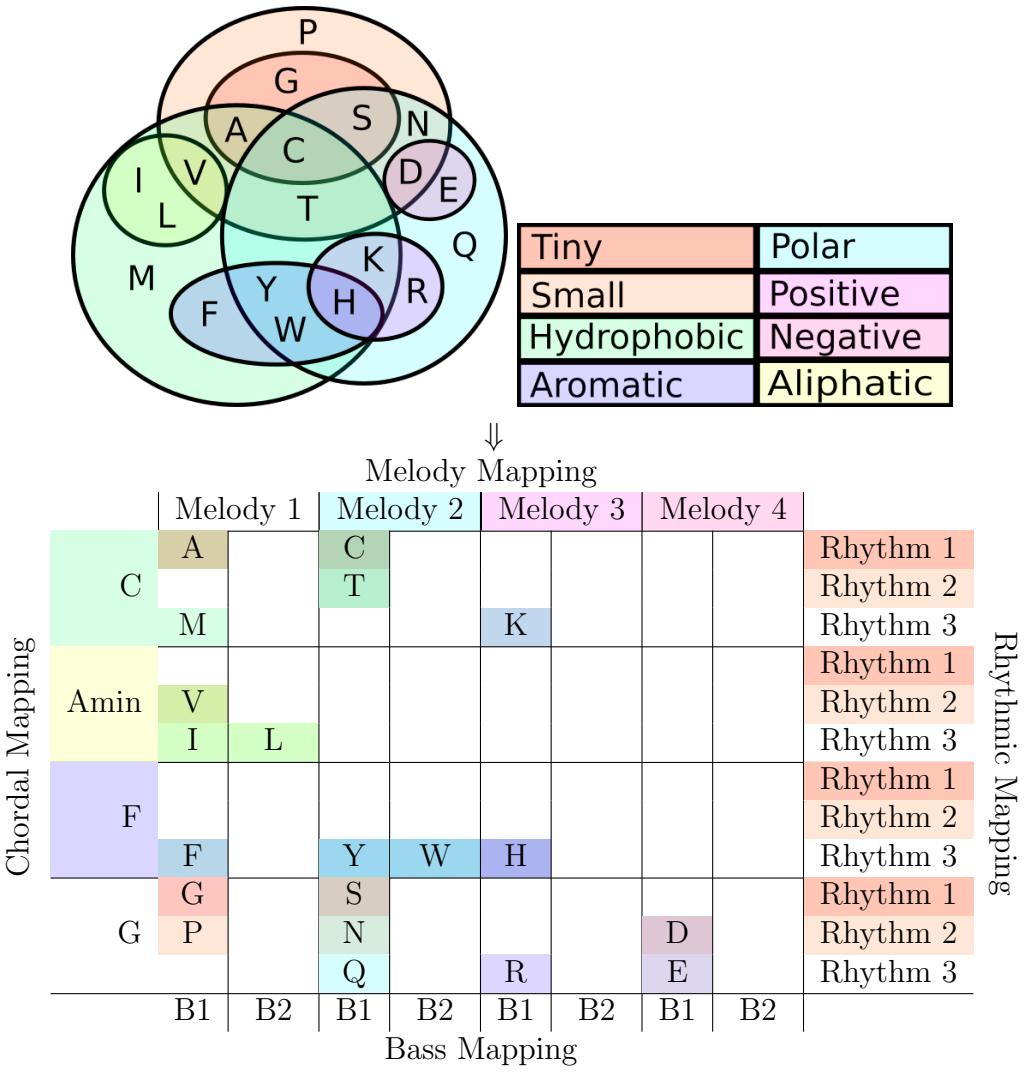


Figure 2.6: Full Mapping

3. Similarity Measurement

In this section, we will look at how we generate a similarity measurement from the music generated by two amino acids. Each musical property that is generated by the properties of the amino acids contains a similarity, and we average these similarities to create a final measurement of similarity between the amino acids.

3.1 Rhythmic Similarity

The rhythm (drum) part is chosen by the mass of the amino acids. Amino acids that fall into the same size category should have high rhythmic similarity; amino acids that are separated by one mass category should have a moderate rhythmic similarity; and amino acids that are separated by two mass categories should have a low rhythmic similarity.

The written rhythm part contains either a drum strike or a rest on each beat. We compare the rhythm parts by counting the number of times they both contain a strike or both contain a rest on the beats and dividing by four. As an example, tiny amino acids have a strike on the first beat and a rest on the remaining three, and small amino acids have strikes on the first and third beats and a rest on the second and fourth beats. Because both tiny and small amino acids have a strike on the first beat and a rest on the second and fourth, they agree three times in the four beats giving a similarity of 0.75.

Table 3.1 contains the full similarity matrix for the rhythm parts. As desired,

each rhythmic part has a similarity of 1 with itself, rhythmic parts that are separated by one mass class have moderate similarities of 0.75 or 0.5, and rhythmic parts separated by two mass classes have a low similarity of 0.25.

Table 3.1: Rhythmic Similarity Matrix

	tiny	small	regular
tiny	1	0.75	0.25
small	0.75	1	0.5
regular	0.25	0.5	1

3.2 Melodic Similarity

The melodic part is chosen by the charge of the amino acid. Non-charged and polar amino acids, as well as positive and negative amino acids should have low similarity. Polar and positive amino acids, as well as polar and negative amino acids should have moderate similarity. Amino acids that fall into the same charge category should have high similarity.

The written melodic part contains pitched quarter notes on each beat. We compare the melodic part by counting the number of times the pitches match and dividing by four. As an example, positive amino acids have the melody ECDC, and polar, but not positive or negative, amino acids have the melody ACDC. These are different on the first beat, but agree on the remaining three giving a similarity of 0.75.

Table 3.2 contains the full similarity matrix for the melodic parts. As desired, non-charged amino acids have 0 similarity to the other amino acids, and polar, with positive or negative, have a similarity of 0.75. However, positive and negative amino acids have a similarity of 0.5. Re-writing the melodic parts to lower this similarity would also lower the similarity of polar, with positive or negative, amino acids.

Table 3.2: Melodic Similarity Matrix

	non-polar	polar	positive	negative
non-polar	1	0	0	0
polar	0	1	0.75	0.75
positive	0	0.75	1	0.5
negative	0	0.75	0.5	1

3.3 Chordal Similarity

The chordal part is chosen by the hydrophobicity of the amino acids, as well as whether they are aliphatic or aromatic. As aliphatic and aromatic amino acids are non-overlapping subsets of hydrophobic amino acids, the three groups should have higher similarity. The fourth group, hydrophilic amino acids, should have a low similarity to the other three groups.

The note pitches in the chordal part may change on external factors, such as the instrument that is playing them. For comparison between amino acids, we instead compare the chordal triad. An aliphatic amino acid is given an A minor chord, ACE, and an aromatic amino acid is given an F chord, FAC. The order of the notes is ignored in the comparison. We can see that Amin and F agree on two pitches out of three, giving a similarity of 0.67.

Table 3.3 contains the full similarity matrix for the chordal parts. The chart is not as ideal as the melodic and rhythmic charts, but to make it more ideal would require choosing new chords, and reducing musicality. The chordal parts still record similarity. Hydrophilic amino acids, the most different, have a sum of similarities of 1.33, while the remaining groups, the most similar, have sums of similarity of 2.33, 2.33, and 2.

A Bmin chord might be used in place of the G chord. This would improve the similarity matrix, because the Bmin has no notes in common with the remaining assigned chords. However, the Bmin requires an f-sharp, which is not in the scale, and the musicality is reduced.

Table 3.3: Chordal Similarity Matrix

	hydrophobic	aliphatic	aromatic	hydrophilic
hydrophobic	1	0.67	0.33	0.33
aliphatic	0.67	1	0.67	0
aromatic	0.33	0.67	1	0
hydrophilic	0.33	0	0	1

3.4 Bass Similarity

The bass part is used to create some distance between amino acids that are otherwise not distinguishable by their characteristics, specifically, Isoleucine and Leucine, and Tryptophan and Tyrosine. All amino acids should have a high similarity, while those two pairs are slightly less similar.

The bass lines are either all root notes, or they alternate between the root and the dominant of the chordal property. The bass line is comprised of eighth notes and quarter notes, so we look at each eighth note for comparison. The alternate bass line plays the dominant note on the "and" of 1 and the "and" of 3, where the primary bass line plays all root notes. This gives a similarity of 0.75.

Measuring the similarity this way causes Leucine and Tryptophan to be slightly less similar compared to the remaining amino acids.

Table 3.4: Bass Similarity Matrix

	primary	alternate
primary	1	0.75
alternate	0.75	1

3.5 Similarity Conclusion

Through the musical properties that are assigned to each amino acid, we are able to generate a musical similarity. Two similar amino acids, such as Asparagine (N) and Aspartate (D) should have a high similarity. Asparagine is a small, polar, and hydrophilic amino acid, while Aspartate is a small, negative, and hydrophilic

amino acid. Their rhythmic, melodic, chordal, and bass similarities are 1, 0.75, 1, and 1 respectively. When we average these similarities, Asparagine and Aspartate have a total musical similarity of 0.9375.

Two dissimilar amino acids, such as Glycine (G) and Histidine (H) should have low similarity. Glycine is tiny, non-polar, and hydrophilic amino acid, while Histidine is a regular, positive, and aromatic amino acid. Their rhythmic, melodic, chordal, and bass similarities are 0.25, 0, 0, and 1, respectively. When we average these similarities, Glycine and Histidine have a total musical similarity of 0.3125.

The musical similarity does a good job of capturing the similarities of the properties of the amino acids. A full similarity matrix can be seen in Table 3.5.

Using the similarity of the individual amino acids, similarities between sequences can be found. If we calculate the similarity between two similar sequences, such as the first eight amino acids from the Human Beta Globin (HBB) and the first eight amino acids in HBB with the sickle cell anemia mutation (HBB-S), we see that they have a high similarity. The first eight amino acids of HBB, MVHLTPEE, and the first eight of HBB-S, MVHLTPVE, only disagree on the seventh amino acid. The seventh amino acid has a similarity of 0.38, while the remaining have a similarity of 1, giving a mean musical similarity of 0.92.

If we calculate the similarity between two dissimilar sequences, such as the first eight amino acids from HBB, and the first eight from Thymidylate Synthase (Thy-A), we see a low similarity. HBB, MVHLTPEE, and Thy-A, MPVAGSEL, have amino acid similarities of 1, 0.75, 0.38, 0.67, 0.52, 0.69, 1, and 0.44, giving a mean musical similarity of 0.68.

Table 3.5: Amino Acid Similarity Matrix

	A	C	D	E	F	G	H	I	K	L	M	N	P	Q	R	S	T	V	W	Y
A	1.00	0.75	0.52	0.40	0.65	0.83	0.40	0.73	0.56	0.67	0.81	0.52	0.77	0.40	0.40	0.58	0.69	0.85	0.33	0.40
C	0.75	1.00	0.71	0.58	0.40	0.58	0.58	0.48	0.75	0.42	0.56	0.77	0.52	0.65	0.58	0.83	0.94	0.60	0.58	0.65
D	0.52	0.71	1.00	0.88	0.38	0.69	0.50	0.38	0.58	0.31	0.46	0.94	0.75	0.81	0.75	0.88	0.77	0.50	0.50	0.56
E	0.40	0.58	0.88	1.00	0.50	0.56	0.63	0.50	0.71	0.44	0.58	0.81	0.63	0.94	0.88	0.75	0.65	0.38	0.63	0.69
F	0.65	0.40	0.38	0.50	1.00	0.56	0.75	0.92	0.58	0.85	0.83	0.38	0.63	0.50	0.50	0.31	0.46	0.79	0.69	0.75
G	0.83	0.58	0.69	0.56	0.56	1.00	0.31	0.56	0.40	0.50	0.65	0.69	0.94	0.56	0.56	0.75	0.52	0.69	0.25	0.31
H	0.40	0.58	0.50	0.63	0.75	0.31	1.00	0.67	0.83	0.60	0.58	0.56	0.38	0.69	0.75	0.50	0.65	0.54	0.88	0.94
I	0.73	0.48	0.38	0.50	0.75	0.56	0.50	1.00	0.67	0.94	0.92	0.38	0.63	0.50	0.50	0.31	0.54	0.88	0.44	0.50
K	0.56	0.75	0.58	0.71	0.58	0.40	0.83	0.67	1.00	0.60	0.75	0.65	0.46	0.77	0.83	0.58	0.81	0.54	0.71	0.77
L	0.67	0.42	0.31	0.44	0.69	0.50	0.44	0.94	0.60	1.00	0.85	0.31	0.56	0.44	0.44	0.25	0.48	0.81	0.50	0.44
M	0.81	0.56	0.46	0.58	0.83	0.65	0.58	0.92	0.75	0.85	1.00	0.46	0.71	0.58	0.58	0.40	0.63	0.79	0.52	0.58
N	0.52	0.77	0.94	0.81	0.38	0.69	0.56	0.38	0.65	0.31	0.46	1.00	0.75	0.88	0.81	0.94	0.83	0.50	0.56	0.63
P	0.77	0.52	0.75	0.63	0.63	0.94	0.38	0.63	0.46	0.56	0.71	0.75	1.00	0.63	0.63	0.69	0.58	0.75	0.31	0.38
Q	0.40	0.65	0.81	0.94	0.50	0.56	0.69	0.50	0.77	0.44	0.58	0.88	0.63	1.00	0.94	0.81	0.71	0.38	0.69	0.75
R	0.40	0.58	0.75	0.88	0.50	0.56	0.75	0.50	0.83	0.44	0.58	0.81	0.63	0.94	1.00	0.75	0.65	0.38	0.63	0.69
S	0.58	0.83	0.88	0.75	0.31	0.75	0.50	0.31	0.58	0.25	0.40	0.94	0.69	0.81	0.75	1.00	0.77	0.44	0.50	0.56
T	0.69	0.94	0.77	0.65	0.46	0.52	0.65	0.54	0.81	0.48	0.63	0.83	0.58	0.71	0.65	0.77	1.00	0.67	0.65	0.71
V	0.85	0.60	0.50	0.38	0.63	0.69	0.38	0.88	0.54	0.81	0.79	0.50	0.75	0.38	0.38	0.44	0.67	1.00	0.31	0.38
W	0.33	0.58	0.50	0.63	0.69	0.25	0.88	0.60	0.71	0.67	0.52	0.56	0.31	0.69	0.63	0.50	0.65	0.48	1.00	0.94
Y	0.40	0.65	0.56	0.69	0.75	0.31	0.94	0.67	0.77	0.60	0.58	0.63	0.38	0.75	0.69	0.56	0.71	0.54	0.94	1.00

4. Conclusion

In this project, we converted amino acid sequences to music. We believe we overcame the challenge discovered by researchers of generating pleasant music while preserving relationships between amino acids properties. A good musical mapping is easier to listen to what is generated and easier to remember repeated sections. Unlike previous mappings that missed important musical structures and used only one instrument, we introduced instrumentation commonly found in pop music. Our mapping of amino acids to musical notes is embellished by using drums to dictate the size of the amino acid, by guitar/keyboard to differentiate between charged and uncharged, aliphatic, and aromatic amino acids and lastly by four different melodies to differentiate between non-polar, polar, positively and negatively charged amino acids. Our mapping follows common musical structures, increasing musicality, and thus, making biology, music, and computer science more appealing to a wider range of audiences.

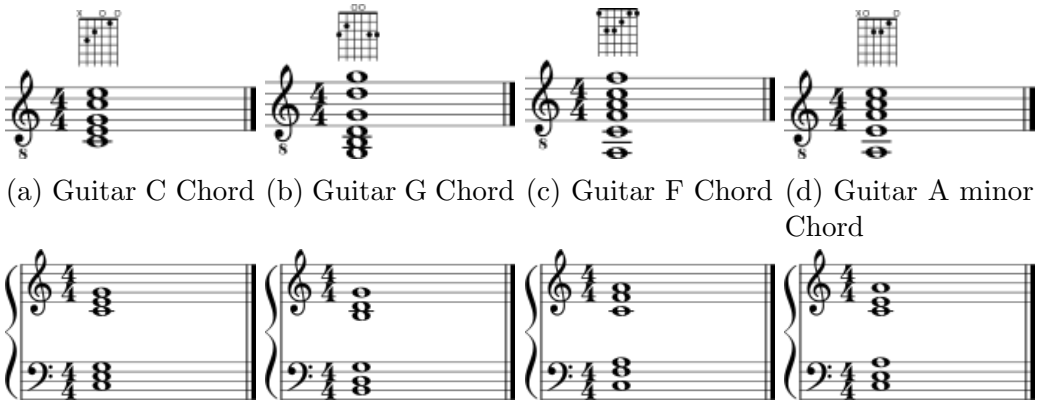
A possible extension to this work is to build databases to analyze protein sequences that have been converted to music according to the mapping described in our work. For example, one could develop a database or recordings of orthologs: similar proteins found in different organisms. By playing a protein by itself, one can hear some of the proteins' characteristics, such as variability and repeated regions. The database will also contain recordings of pairwise alignments of orthologous proteins. By juxtaposing and recording two orthologous proteins simultaneously, but with different instrumentations, one can hear the conserved

regions, also known as protein domains, between the two proteins. The conserved region is a theme (also known as pattern or motif) and the various orthologous proteins that contain this theme, play it as variations on that theme. Even though orthologous proteins are not identical, they are similar enough, and so when converted to music, will contain recognizable variations on common themes.

Appendices

A. Fine Tuning

During the course of this project, we made several minor changes to the generated music in order to improve the quality of the midi music that is generated. These changes do not have an effect on the measured similarity of the music.



(a) Guitar C Chord (b) Guitar G Chord (c) Guitar F Chord (d) Guitar A minor Chord

(e) Piano C Chord (f) Piano G Chord (g) Piano F Chord (h) Piano A minor Chord

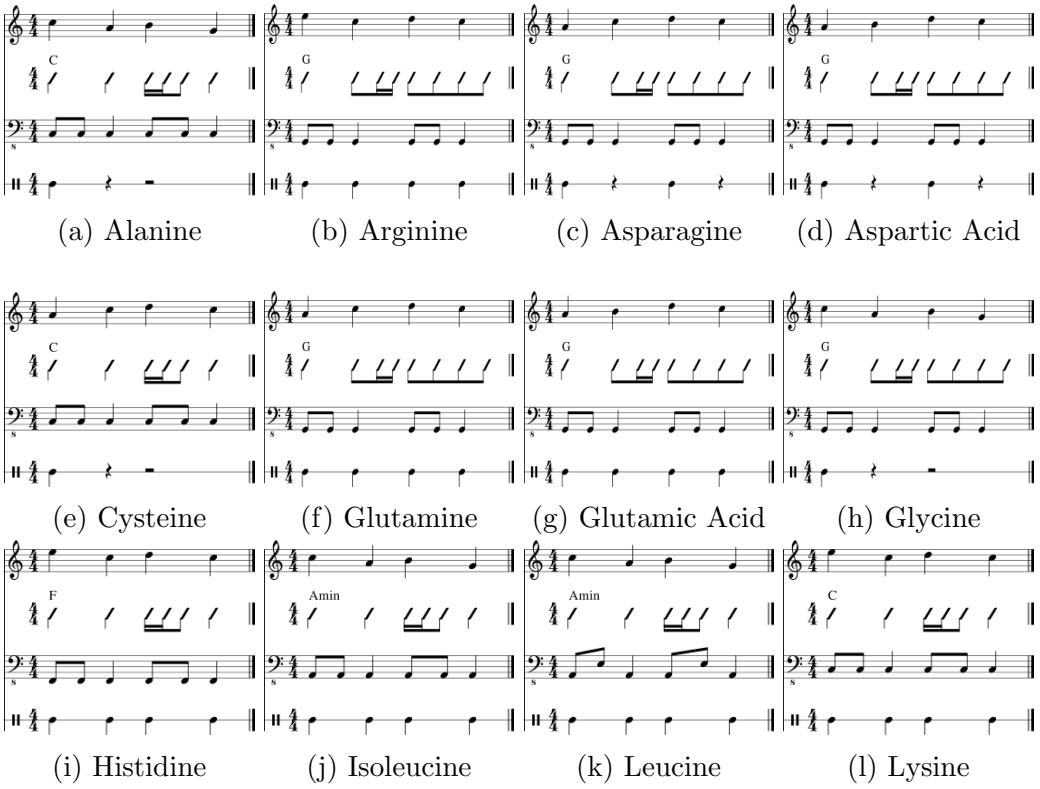
Figure A.1: Full Chords as Played by Rhythm Guitar and Piano Parts

Several adjustments were made to the way chords are played. The guitar chordal part uses the notes from the open position chords as they would be played on a guitar fretboard. Because of the design of the fretboard, occasional notes are skipped in the chords, for example, the higher b in the G chord. Additionally, the notes are played in rapid succession, instead of at the same time. These notes are played either from lowest to highest, creating a downstroke, or from highest to lowest, creating an upstroke. These changes make the generated guitar sound more similar to a real guitar.

The piano chords were modified to follow how a pianist would likely play them. The chords are all two triads, or 6 notes. The C is played in the root position, the Am and G are played as the 1st inversion, and the F is played as the 2nd inversion. This is a common method for a pianist to play these chords, as it requires little lateral movement of the wrists on the keyboard.

Finally, two rhythm patterns are used in the chordal part. These patterns have no importance in regards to the properties of the amino acids. They do provide some variation in the rhythm of the chordal part, which significantly improves the musicality of the generated music.

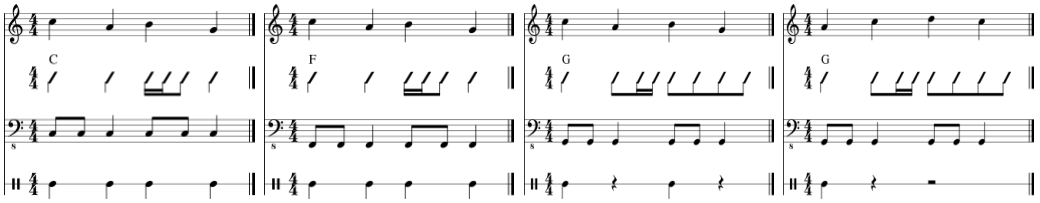
B. Sheet Music for Each Amino Acid



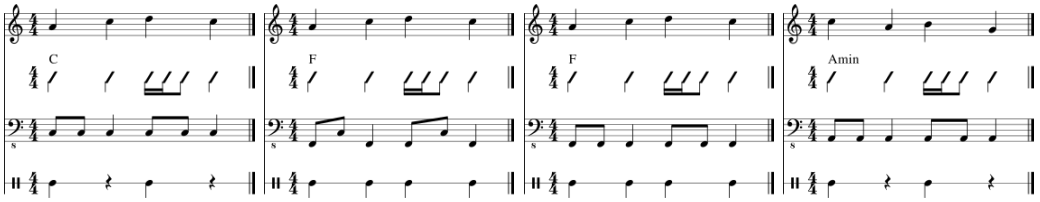
(a) Alanine (b) Arginine (c) Asparagine (d) Aspartic Acid

(e) Cysteine (f) Glutamine (g) Glutamic Acid (h) Glycine

(i) Histidine (j) Isoleucine (k) Leucine (l) Lysine



(m) Methionine (n) Phenylalanine (o) Proline (p) Serine



(q) Threonine (r) Tryptophan (s) Tyrosine (t) Valine

Figure B.1: Composition for Individual Amino Acids

C. Human Beta Globin Composition

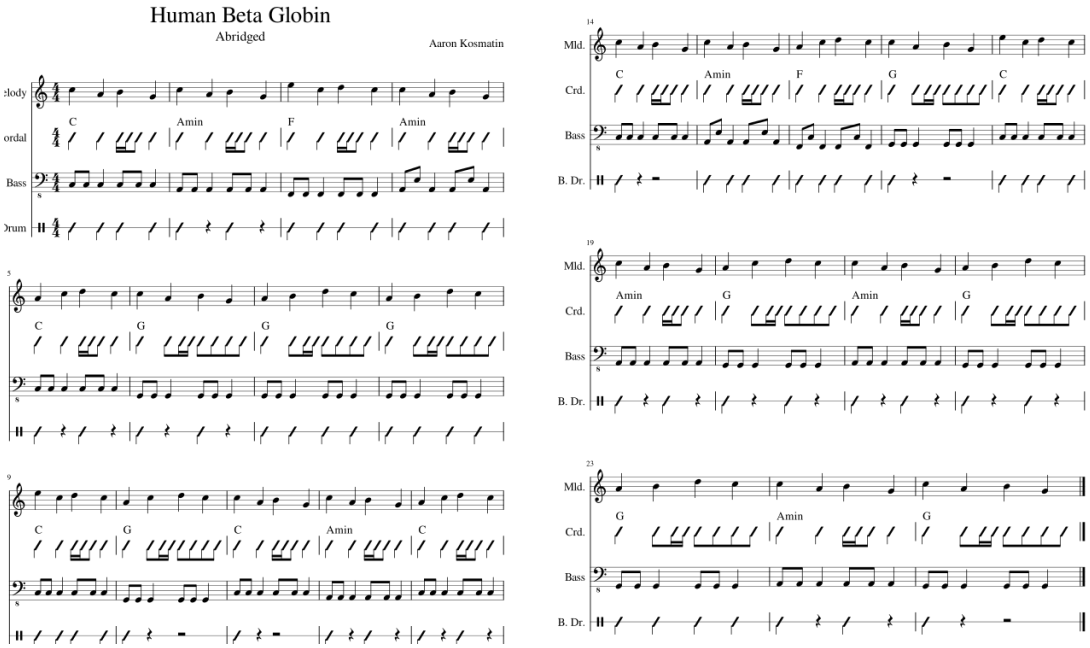


Figure C.1: First 25 Measures of Human Beta Globin

Bibliography

[1] Takahashi, Rie, and Jeffrey H. Miller. "Conversion of amino-acid sequence in proteins to classical music: search for auditory patterns." *Genome Biology 8.5 (2007): 405.*

[2] Ohno, Susumu, and Midori Ohno. "The all pervasive principle of repetitious recurrence governs not only coding sequence construction but also human endeavor in musical composition." *Immunogenetics 24.2 (1986): 71-78.*

[3] Gena P, Strom C: A physiological approach to DNA music. In: CADE 2001. Glasgow, UK: Glasgow School of Art Press; 2001:129-134

[4] "Pachelbel's Canon." *Wikipedia*. Wikimedia Foundation, 03 July 2017. Web. 08 July 2017.

[5] "List of songs containing the I–V–vi–IV progression." *Wikipedia*. Wikimedia Foundation, 07 July 2017. Web. 08 July 2017.

[6] *A protein primer: a musical introduction to protein structure. N.p., n.d. Web. 22 July 2017.*

```
package AminoAcidGUI.Data;

import java.util.LinkedHashMap;

public class Acids {
    public LinkedHashMap<String, String> getAcids() {
        return acids;
    }

    private LinkedHashMap<String, String> acids = new LinkedHashMap<>();

    private static Acids ourInstance = new Acids();

    public static Acids getInstance() {
        return ourInstance;
    }

    private Acids() {
        acids.put("None","");
        acids.put("HBB Human",          "
            MVHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVV
YPWTQRFFESFGDLSTPDAVMGNPKVKAHGKKVLGAFSDGLAHLDNLKGTFATLSELHCDKLHVDPENFRLLGNVLVCVL

AHHFGKEFTPPVQAAYQKVVAGVANALAHKYH");
        acids.put("HBB Similar",        "
            MVHLTDAEKSAVSCLWAKVNPDEVGGEALGRLLVV
YPWTQRYFDSFGDLSSASAIMGNPKVKAHGKKVITAFNEGLKNLDNLKGTFASLSELHCDKLHVDPENFRLLGNAIVIVL

GHHLGKDFTPAAQAAFQKVVAGVATALAHKYH");
```

```
        acids.put("HBB S",              "
            MVHLTPVEKSAVTALWGKVNVDEVGGEALGRLLVV
YPWTQRFFESFGDLSTPDAVMGNPKVKAHGKKVLGAFSDGLAHLDNLKGTFATLSELHCDKLHVDPENFRLLGNVLVCVL

AHHFGKEFTPPVQAAYQKVVAGVANALAHKYH");
        acids.put("HBB C",              "
            MVHLTPKEKSAVTALWGKVNVDEVGGEALGRLLVV
YPWTQRFFESFGDLSTPDAVMGNPKVKAHGKKVLGAFSDGLAHLDNLKGTFATLSELHCDKLHVDPENFRLLGNVLVCVL

AHHFGKEFTPPVQAAYQKVVAGVANALAHKYH");
        acids.put("HBB E",              "
            MVHLTPEEKSAVTALWGKVNVDEVGGKALGRLLVV
YPWTQRFFESFGDLSTPDAVMGNPKVKAHGKKVLGAFSDGLAHLDNLKGTFATLSELHCDKLHVDPENFRLLGNVLVCVL

AHHFGKEFTPPVQAAYQKVVAGVANALAHKYH");
        acids.put("HBA1 Human",         "
            MVLSPADKTNVKAAWGKVGAHAGEYGAEALERMFL
SFPTTKTYFPHFDLSHGSAQVKGHGKKVADALTNAVAHVDDMPNALSALSDLHAHKLRVDPVNFKLLSHCLLVTLAAHLP

AEFTPAVHASLDKFLASVSTVLTSKYR");
        acids.put("DPH6 Human",         "
            MRVAALISGGKDSCYNMMQCIAAGHQIVALANLRP
AENQVGSDELDSYMYQTVGHHAIDLYAEAMALPLYRRTIRGRSLDTRQVYTKCEGDEVEDLYELLKLVKEKEEVEGISVG

AILSDYQRIRVENVCKRLNLQPLAYLWQRNQEDLLREMISSNIQAMIIKVAALGLDPDKHLGKTLDQMEPYLIELSKKYG

VHVCGEGGEYETFTLDCPLFKKKIIVDSSEVVIHSADAFAPVAYLRFLELHLEDKVSSVPDNYRTSNYIYNF
    ");
        acids.put("DPH6 Mouse",         "
```

```
            MRVAALISGGKDSCYNMMQCIAEGHQIVALANLRP
DENQVESDELDSYMYQTVGHHAIDLYAEAMALPLYRRAIRGRSLETGRVYTQCEGDEVEDLYELLKLVKEKEEIEGVSVG

AILSDYQRGRVENVCKRLNLQPLAYLWQRNQEDLLREMIASNIKAIIIKVAALGLDPDKHLGKTLVEMEPYLLELSKKYG

VHVCGEGGEYETFTLDCPLFKKKIVVDSSEAVMHSADAFAPVAYLRLSRLHLEEKVSSVPADDETANSIHSS
    ");
        acids.put("DPH6 Rat",            "
            MRVAALISGGKDSCYNMMRCIAEGHQIVALANLRP
DDNQVESDELDSYMYQTVGHHAIDLYAEAMALPLYRRTIRGRSLETGRVYTRCEGDEVEDLYELLKLVKEKEEIEGVSVG

AILSDYQRVRVENVCKRLNLQPLAYLWQRNQEDLLREMIASNIEAIIIKVAALGLDPDKHLGKTLGEMEPYLLELSKKYG

VHVCGEGGEYETFTLDCPLFKKKIVVDTSEAVIHSADAFAPVAYLRLSGLHLEEKVSSVPGDDETTSYIHNS
    ");
        acids.put("DPH6 ZebraFish",      "
            MRVVGLISGGKDSCFNMLQCVSAGHSIVALANLRP
ADHAASDELDSYMYQTVGHQAVDLIAEAMGLPLYRRTIEGSSVHIDREYSPTDGDEVEDLYQLLKHVKEEMHVDGVSVGA

ILSDYQRVRVENVCARLQLQPLAYLWRRDQAALLSEMISSGLHAILIKVAAFGLHPDKHLGKSLAEMELYLHELSEKYGV

HICGEGGEYETFTLDCPLFKKKIIIDATETVIHSDDAFAPVGFLRFTKMHTEDKTEGSGGPPPPSLSACPCQSAIDRMTE

ELEYADKTADVQRECPSHTQSTWQLDEGCEVSHSSSSSGFQWISGLSALPSEHPDIQSQAQHVFTLLQSRLQEMGSALRH

VLLVHLYVSSMQDFGLINSIYSRLFTHNPPARVCVQASLPVGQQLQMDVLLQDQTKASPSSSSSVCEEECFPQRETLHVQ

SVSHWAPANIGPYSQATQVQLCFLLTAAASAVFSTVFYISTSAAQWLSGQHCGFTARRSLV");
        acids.put("DPH6 D. melanogaster", "
```

```
            MRVVAMVSGGKDSCYNMMQCVAEGHEIVALANLHP
KDRDELDSFMYQTVGHMGIEILASAMGLPLYRRETKGKSTQTGKQYVPTDDDEVEDLYSLLETCKHELQVDAVAVGAILS

DYQRVRVENVCSRLNLISLAYLWRRDQTELLQEMIDCQVHAIIIKVAALGLVPDRHLGKSLREMQPHLLKMRDKYGLNVC

GEGGEYETFTLDCPLFRQRIVVEDIQTIISSADPICPVGYINFTKLTLQPKEAAGAASSGGNEVVFVKRSLDYISDLNES

TYSDLSDPDFSETELELIEKETRLRESLSQSELISRSNSFGRHLAATASSPIPIVTKSASVDEPTAAAAPILGGVGGPPI

CSTSACASMLLTTTADGLSSLASSQSQGGGHGLGSSTAAVCGSLSLAISSLGLSANTCCHPGGAGGGGGVGIGVGAGAGA

GAPSATTQPPSPLKYEREFRPLANEARAAINAKGWMWLAGIQGSGTEGIEQGMQQALDTLRDLCQAKGYDLQDLCYVTLY

VRSIGEYPLLNRVYHRAFDFHNPPTRVCVECPLPDGCHVVMEAIAYRQPVAGTISSAEERDREGEETAAALLNGRRNTMH

VQGISHWAPANIGPYSQSTRIGDITYISGQIALVPGSMTIIEGGIRPQCKLTLRHISRIAKAMNAHGQLRDVVHGICFVT

HPAFIGEARRQWERRTTNAIMDYIVLPALPREALVEWQVWAHTHNDRFDYEETGCSVGDYTISIRRRWNYENNCAAIVCY

VSTGLASSTTQLTQLSDDILGNHCRLAQAVNAEHLDEIFTYVVNRLLKDYPLAKKQASQPTNSATPPATPTQPGGAGGDQ

QQPVPAIHLKLFYQVNAAPATDLLLQALHDFRLKCQDTAAIVYTVLPACSLHNFSTFLSICGVRHE");
       acids.put("DPH6 C. elegans",     "
           MQVVGLISGGKDSCYNLMCAVREGHQIVALANLHP
PKDAKSDELDSYMYQSVGADGVELYGEAMQLPLYRREITGEPKNQKSDYEKTDGDEVEDLFELLCEVKKHHPEVKGVSAG

AILSSYQKVRVEDICRRLDLVPLCFLWEREQNGLLAEMVENGLDAILIKVAAIGLGEQHLGKTLSEMAPIMKVLQDKYGV

HPCGEGGEFESFVRDCPLFKKRIVIDETETVTHQDDPIAPVFYLRLKKMHLEDK");
```

```
        acids.put("DPH6 S. cerevisiae", "
           MKFIALISGGKDSFYNIFHCLKNNHELIALGNIYP
KESEEQELDSFMFQTVGHDLIDYYSKCIGVPLFRRSILRNTSNNVELNYTATQDDEIEELFELLRTVKDKIPDLEAVSVG

AILSSYQRTRVENVCSRLGLVVLSYLWQRDQAELMGEMCLMSKDVNNVENDTNSGNKFDARIIKVAAIGLNEKHLGMSLP

MMQPVLQKLNQLYQVHICGEGGEFETMVLDAPFFQHGYLELIDIVKCSDGEVHNARLKVKFQPRNLSKSFLLNQLDQLPV

PSIFGNNWQDLTQNLPKQQAKTGEQRFENHMSNALPQTTINKTNDKLYISNLQSRKSETVEKQSEDIFTELADILHSNQI

PRNHILSASLLIRDMSNFGKINKIYNEFLDLSKYGPLPPSRACVGSKCLPEDCHVQLSVVVDVKNTGKEKINKNKGGLHV

QGRSYWAPCNIGPYSQSTWLNDDANQVSFISGQIGLVPQSMEILGTPLTDQIVLALQHFDTLCETIGAQEKLLMTCYISD

ESVLDSVIKTWAFYCSNMNHRSDLWMDKSDDVEKCLVLVKISELPRGAVAEFGGVTCKRLIVDDNDSDKKEREENDDVST

VFQKLNLNIEGFHNTTVSAFGYNRNFITGFVDSREELELILEKTPKSAQITLYYNPKEIITFHHHIGYYPVEKLFDYRGK

EHRFGLHIRS");
        acids.put("Thymidylate synthase", "
           MPVAGSELPRRPLPPAAQERDAEPRPPHGELQYLG
QIQHILRCGVRKDDRTGTGTLSVFGMQARYSLRDEFPLLTTKRVFWKGVLEELLWFIKGSTNAKELSSKGVKIWDANGSR

DFLDSLGFSTREEGDLGPVYGFQWRHFGAEYRDMESDYSGQGVDQLQRVIDTIKTNPDDRRIIMCAWNPRDLPLMALPPC

HALCQFYVVNSELSCQLYQRSGDMGLGVPFNIASYALLTYMIAHITGLKPGDFIHTLGDAHIYLNHIEPLKIQLQREPRP

FPKLRILRKVEKIDDFKAEDFQIEGYNPHPTIKMEMAV");
    }
```

```
}
```

```
package AminoAcidGUI.GUIElements;

import javax.sound.midi.Track;
import javax.swing.*;

public class CustomJSlider extends JSlider {
    private Track track;
    private int trackIndex;

    public CustomJSlider(Track track, int trackIndex){
        super();
        this.track = track;
        this.trackIndex = trackIndex;
    }

    public Track getTrack() {
        return track;
    }

    public int getTrackIndex() {
        return trackIndex;
    }
}
```

```
package AminoAcidGUI.GUIElements;
```

```
import javax.swing.*;

public class CustomJToggleButton extends JToggleButton {
    private String type;
    private int trackIndex;

    public CustomJToggleButton(String type, int trackIndex){
        super(type);
        this.type = type;
        this.trackIndex = trackIndex;
    }

    public int getTrackIndex() {
        return trackIndex;
    }

    public String getType() {
        return type;
    }
}



package AminoAcidGUI.GUIElements;

import javax.sound.midi.Track;
import java.util.LinkedList;

public class GroupedJSlider extends CustomJSlider{
    private static LinkedList<CustomJSlider> allGroupedSliders = new
```

```
        LinkedList<
>();

    public GroupedJSlider(Track track, int trackIndex){
        super(track, trackIndex);
        allGroupedSliders.add(this);
    }

    public static LinkedList<CustomJSlider> getAllGroupedSliders() {
        return allGroupedSliders;
    }
}
```

```
package AminoAcidGUI.GUIElements;

import java.util.LinkedList;

public class GroupedToggleButton extends CustomJToggleButton {
    private static LinkedList<CustomJToggleButton> allGroupedButtons =
        new Linke
dList<>();

    public GroupedToggleButton(String type, int trackIndex) {
        super(type, trackIndex);
        allGroupedButtons.add(this);
    }
```

```
    public static LinkedList<CustomJToggleButton> getAllGroupedButtons()
         {
        return allGroupedButtons;
    }

    public static LinkedList<CustomJToggleButton> getGroupedByType(
        String type){
        LinkedList<CustomJToggleButton> returnValue = new LinkedList<>();
        for (CustomJToggleButton tb: allGroupedButtons){
            if(tb.getType().equals(type)){
                returnValue.add(tb);
            }
        }
        return returnValue;
    }
}



package AminoAcidGUI;

import javax.sound.midi.Track;
import javax.swing.*;

class DropDown{
    private Track track;
    private int trackIndex;
    private JComboBox<String> comboBox;
    private int defaultInstrument;
```

```
    public DropDown(Track track, int trackIndex, JComboBox<String>
        comboBox, int
 defaultInstrument){
        this.track=track;
        this.trackIndex=trackIndex;
        this.comboBox=comboBox;
        this.defaultInstrument = defaultInstrument;
    }

    public Track getTrack() {
        return track;
    }

    public int getTrackIndex() {
        return trackIndex;
    }

    public JComboBox<String> getComboBox() {
        return comboBox;
    }

    public int getDefaultInstrument() {
        return defaultInstrument;
    }
}

package AminoAcidGUI;

import javax.swing.*;
```

```
public class Main {
    public static void main(String[] args) {
        JFrame frame = new JFrame("AminoAcidPlayer");
        frame.setContentPane(new AminoAcidPlayer().aminoAcidPlayerPanel);
        frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        frame.pack();
        frame.setVisible(true);
    }
}
```

```
package AminoAcidGUI;

import AminoAcidComposition.MidiInterface$;
import AminoAcidGUI.Data.Acids;
import AminoAcidGUI.GUIElements.CustomJSlider;
import AminoAcidGUI.GUIElements.CustomJToggleButton;
import AminoAcidGUI.GUIElements.GroupedJSlider;
import AminoAcidGUI.GUIElements.GroupedToggleButton;
import cc.mallet.util.ArrayUtils;

import javax.swing.*;
import java.io.IOException;
import java.util.*;
import java.util.Timer;

public class AminoAcidPlayer extends JFrame {
```

```
protected JPanel aminoAcidPlayerPanel;
private MidiInterface$ mi = MidiInterface$.MODULE$;
private Boolean trackPresent = false;
private Boolean paused = false;
private long pauseLocation = 0;
private Timer updateSelection;
private int currentAcid = 0;
private JTextField firstAminoAcid;
private JTextField secondAminoAcid;
private JButton playButton;
private JButton stopButton;
private GroupedToggleButton FirstMelodyMuteButton;
private GroupedToggleButton FirstMelodySoloButton;
private GroupedToggleButton FirstRhythmMuteButton;
private GroupedToggleButton FirstRhythmSoloButton;
private GroupedToggleButton FirstBassMuteButton;
private GroupedToggleButton FirstBassSoloButton;
private GroupedToggleButton FirstDrumMuteButton;
private GroupedToggleButton FirstDrumSoloButton;
private GroupedToggleButton SecondMelodyMuteButton;
private GroupedToggleButton SecondMelodySoloButton;
private GroupedToggleButton SecondRhythmMuteButton;
private GroupedToggleButton SecondRhythmSoloButton;
private GroupedToggleButton SecondBassMuteButton;
private GroupedToggleButton SecondBassSoloButton;
private GroupedToggleButton SecondDrumMuteButton;
private GroupedToggleButton SecondDrumSoloButton;
private JComboBox<String> FirstMelodyDropDown;
private JComboBox<String> FirstRhythmDropDown;
```

```
    private JComboBox<String> FirstBassDropDown;
    private JComboBox<String> SecondMelodyDropDown;
    private JComboBox<String> SecondRhythmDropDown;
    private JComboBox<String> SecondBassDropDown;
    private JTextField saveField;
    private JButton saveButton;
    private JButton saveAsButton;
    private JButton convertButton;
    private JComboBox<String> FirstAminoAcidDropDown;
    private JComboBox<String> SecondAminoAcidDropDown;
    private JPanel SequencePanel;
    private JPanel PlayControlPanel;
    private JPanel SaveControlPanel;
    private JPanel InstrumentControlPanel;
    private JPanel TrackControlPanel;
    private GroupedJSlider firstMelodySlider;
    private GroupedJSlider secondMelodySlider;
    private GroupedJSlider firstRhythmSlider;
    private GroupedJSlider secondRhythmSlider;
    private GroupedJSlider firstBassSlider;
    private GroupedJSlider secondBassSlider;
    private GroupedJSlider firstDrumSlider;
    private GroupedJSlider secondDrumSlider;
    private JButton experimentButton;


    private LinkedHashMap<String, String> acids = Acids.getInstance().
        getAcids()
;
```

```
    private DropDown[] dropDowns = new DropDown[6];
    private LinkedList<CustomJSlider> volumeSliders = GroupedJSlider.
        getAllGroup
edSliders();

    public AminoAcidPlayer() {
        dropDowns[0] = new DropDown(mi.rhythmGuitarTrack(), mi.
            rhythmGuitarTrack
Index(), FirstRhythmDropDown, 24);
        dropDowns[1] = new DropDown(mi.melodyTrack(), mi.melodyTrackIndex
            (), Fir
stMelodyDropDown, 48);
        dropDowns[2] = new DropDown(mi.bassTrack(), mi.bassTrackIndex(),
            FirstBa
ssDropDown, 35);
        dropDowns[3] = new DropDown(mi.altRhythmGuitarTrack(), mi.
            altRhythmGuita
rTrackIndex(), SecondRhythmDropDown, 0);
        dropDowns[4] = new DropDown(mi.altMelodyTrack(), mi.
            altMelodyTrackIndex(
), SecondMelodyDropDown, 4);
        dropDowns[5] = new DropDown(mi.altBassTrack(), mi.
            altBassTrackIndex(), S
econdBassDropDown, 35);

        for(DropDown dd: dropDowns){
            for (String instrument : mi.listInstruments()) {
                dd.getComboBox().addItem(instrument);
            }
```

```
            dd.getComboBox().setSelectedIndex(dd.getDefaultInstrument());
        }

        for (String acid : acids.keySet()) {
            FirstAminoAcidDropDown.addItem(acid);
            SecondAminoAcidDropDown.addItem(acid);
        }

        FirstAminoAcidDropDown.addActionListener(e -> {
            firstAminoAcid.setText(acids.get(FirstAminoAcidDropDown.
                getSelectedI
tem()));
            firstAminoAcid.setCaretPosition(0);
        });

        SecondAminoAcidDropDown.addActionListener(e -> {
            secondAminoAcid.setText(acids.get(SecondAminoAcidDropDown.
                getSelecte
dItem()));
            secondAminoAcid.setCaretPosition(0);
        });

        playButton.addActionListener(e -> {
            if (!trackPresent) {
                createTrack();
                playMidi(120);
                paused = false;
                trackPresent = true;
                playButton.setText("pause");
```

```
                pauseLocation = 0;
                firstAminoAcid.requestFocus();
                updateSelection = new Timer();
                updateSelection.schedule(new TimerTask() {
                    @Override
                    public void run() {
                        if (currentAcid > firstAminoAcid.getText().length
                            () &&
                                currentAcid > secondAminoAcid.getText().
                                    length()
) {
                            stopPlayback();
                            currentAcid = -1;
                        }
                        firstAminoAcid.setScrollOffset((currentAcid - 32)
                            * 8);
                        secondAminoAcid.setScrollOffset((currentAcid - 32)
                             * 8);
                        if (firstAminoAcid.hasFocus()) {
                            highlightCurrentAcid(secondAminoAcid,
                                currentAcid);
                            highlightCurrentAcid(firstAminoAcid,
                                currentAcid);
                        } else if (secondAminoAcid.hasFocus()) {
                            highlightCurrentAcid(firstAminoAcid,
                                currentAcid);
                            highlightCurrentAcid(secondAminoAcid,
                                currentAcid);
                        }
```

```
                    currentAcid++;
                }
            }, 0, 2000);
        } else {
            pauseLocation = stopPlayback();
            paused = true;
            playButton.setText("play");
        }
    });


    stopButton.addActionListener(e -> {
        stopPlayback();
        currentAcid = 0;
    });

    for (DropDown dd : dropDowns) {
        dd.getComboBox().addActionListener(e -> mi.setInstrument(dd.
            getTrack
(), dd.getTrackIndex(), dd.getComboBox().getSelectedIndex()));
    }

    for (CustomJToggleButton sb : GroupedToggleButton.
        getGroupedByType("Solo
")) {
        sb.addActionListener(e -> mi.soloTrack(sb.getTrackIndex()));
    }

    for (CustomJToggleButton mb : GroupedToggleButton.
```

```
            getGroupedByType("Mute
")) {
            mb.addActionListener(e -> mi.muteTrack(mb.getTrackIndex()));
        }

        for (CustomJSlider vs : volumeSliders) {
            vs.addChangeListener(e -> mi.changeVolume(vs.getTrack(), vs.
                getTrack
Index(), vs.getValue()));
        }

        saveAsButton.addActionListener(e -> {
            final JFileChooser fc = new JFileChooser();
            int retVal = fc.showSaveDialog(AminoAcidPlayer.this);
            if (retVal == JFileChooser.APPROVE_OPTION) {
                if (!trackPresent) {
                    createTrack();
                }
                saveField.setText(fc.getSelectedFile().toString());
                mi.writeMidi(fc.getSelectedFile().toString());
            }
        });
        saveButton.addActionListener(e -> {
            if (!trackPresent) {
                createTrack();
            }
            mi.writeMidi(saveField.getText());
        });
        convertButton.addActionListener(e -> {
```

```
            Runtime rt = Runtime.getRuntime();
            try {
                rt.exec("timidity " + saveField.getText() + " -Ow").
                    waitFor();
                JOptionPane.showMessageDialog(AminoAcidPlayer.this, "
                    Converted t
o wav");
            } catch (IOException e1) {
                JOptionPane.showMessageDialog(AminoAcidPlayer.this, "
                    Failed to c
onvert. Requires timidity.", "error", JOptionPane.ERROR_MESSAGE);
                e1.printStackTrace();
            } catch (InterruptedException e1) {
                e1.printStackTrace();
            }

        });
        experimentButton.addActionListener(e -> {
            double[][] scores = {
                    {1.00,0.75,0.52,0.40,0.65,0.83,0.40,0.73,0.56,0.67,0.81,0.52

,0.77,0.40,0.40,0.58,0.69,0.85,0.33,0.40},
                    {0.75,1.00,0.71,0.58,0.40,0.58,0.58,0.48,0.75,0.42,0.56,0.77

,0.52,0.65,0.58,0.83,0.94,0.60,0.58,0.65},
                    {0.52,0.71,1.00,0.88,0.38,0.69,0.50,0.38,0.58,0.31,0.46,0.94

,0.75,0.81,0.75,0.88,0.77,0.50,0.50,0.56},
                    {0.40,0.58,0.88,1.00,0.50,0.56,0.63,0.50,0.71,0.44,0.58,0.81
```

```
,0.63,0.94,0.88,0.75,0.65,0.38,0.63,0.69},
                {0.65,0.40,0.38,0.50,1.00,0.56,0.75,0.75,0.58,0.69,0.83,0.38

,0.63,0.50,0.50,0.31,0.46,0.63,0.69,0.75},
                {0.83,0.58,0.69,0.56,0.56,1.00,0.31,0.56,0.40,0.50,0.65,0.69

,0.94,0.56,0.56,0.75,0.52,0.69,0.25,0.31},
                {0.40,0.58,0.50,0.63,0.75,0.31,1.00,0.50,0.83,0.44,0.58,0.56

,0.38,0.69,0.75,0.50,0.65,0.38,0.88,0.94},
                {0.73,0.48,0.38,0.50,0.92,0.56,0.67,1.00,0.67,0.94,0.92,0.38

,0.63,0.50,0.50,0.31,0.54,0.88,0.60,0.67},
                {0.56,0.75,0.58,0.71,0.58,0.40,0.83,0.67,1.00,0.60,0.75,0.65

,0.46,0.77,0.83,0.58,0.81,0.54,0.71,0.77},
                {0.67,0.42,0.31,0.44,0.85,0.50,0.60,0.94,0.60,1.00,0.85,0.31

,0.56,0.44,0.44,0.25,0.48,0.81,0.67,0.60},
                {0.81,0.56,0.46,0.58,0.83,0.65,0.58,0.92,0.75,0.85,1.00,0.46

,0.71,0.58,0.58,0.40,0.63,0.79,0.52,0.58},
                {0.52,0.77,0.94,0.81,0.38,0.69,0.56,0.38,0.65,0.31,0.46,1.00

,0.75,0.88,0.81,0.94,0.83,0.50,0.56,0.63},
                {0.77,0.52,0.75,0.63,0.63,0.94,0.38,0.63,0.46,0.56,0.71,0.75

,1.00,0.63,0.63,0.69,0.58,0.75,0.31,0.38},
```

```
                {0.40,0.65,0.81,0.94,0.50,0.56,0.69,0.50,0.77,0.44,0.58,0.88

,0.63,1.00,0.94,0.81,0.71,0.38,0.69,0.75},
                {0.40,0.58,0.75,0.88,0.50,0.56,0.75,0.50,0.83,0.44,0.58,0.81

,0.63,0.94,1.00,0.75,0.65,0.38,0.63,0.69},
                {0.58,0.83,0.88,0.75,0.31,0.75,0.50,0.31,0.58,0.25,0.40,0.94

,0.69,0.81,0.75,1.00,0.77,0.44,0.50,0.56},
                {0.69,0.94,0.77,0.65,0.46,0.52,0.65,0.54,0.81,0.48,0.63,0.83

,0.58,0.71,0.65,0.77,1.00,0.67,0.65,0.71},
                {0.85,0.60,0.50,0.38,0.79,0.69,0.54,0.88,0.54,0.81,0.79,0.50

,0.75,0.38,0.38,0.44,0.67,1.00,0.48,0.54},
                {0.33,0.58,0.50,0.63,0.69,0.25,0.88,0.44,0.71,0.50,0.52,0.56

,0.31,0.69,0.63,0.50,0.65,0.31,1.00,0.94},
                {0.40,0.65,0.56,0.69,0.75,0.31,0.94,0.50,0.77,0.44,0.58,0.63

,0.38,0.75,0.69,0.56,0.71,0.38,0.94,1.00}};
          String acids = "ACDEFGHIKLMNPQRSTVWY";
                //"'A', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'K', 'L', '
                   M', 'N
', 'P', 'Q', 'R', 'S', 'T', 'V', 'W', 'Y'});
          String sequence1 = firstAminoAcid.getText();
          String sequence2 = secondAminoAcid.getText();

          double sum = 0.0;
```

```
            int count = 0;
            for(int i = 0; i<sequence1.length() && i<sequence2.length();i
                ++){
                count++;
                sum+=scores[acids.indexOf(sequence1.charAt(i))][acids.
                    indexOf(se
quence2.charAt(i))];
            }
            if(sum>0){
                JOptionPane.showMessageDialog(this.getFrames()[0], sum /
                    count,
"Score", JOptionPane.PLAIN_MESSAGE);
            } else {
                JOptionPane.showMessageDialog(this.getFrames()[0], "
                    Requires Seq
uences");
            }
        });
    }

    private static void highlightCurrentAcid(JTextField textField, int
        currentAc
id) {
        try {
            textField.requestFocus();
            textField.setSelectionStart(currentAcid - 1);
            textField.setSelectionEnd(currentAcid);
        } catch (IndexOutOfBoundsException ignored) {
        }
```

```
    }

    private void createTrack() {
        createTrack("-" + firstAminoAcid.getText(), "-" + secondAminoAcid
            .getTex
t());
    }

    private void createTrack(String firstTrack, String secondTrack) {
        mi.stopMidi();
        mi.createMidi(firstTrack, secondTrack);
        for(DropDown dd: dropDowns){
            mi.setInstrument(dd.getTrack(), dd.getTrackIndex(), dd.
                getComboBox()
.getSelectedIndex());
        }
    }

    private void playMidi(int bpm) {
        mi.playMidi(bpm);
        mi.setLocation(pauseLocation);
        for(CustomJSlider vs: volumeSliders){
            vs.setEnabled(true);
        }
        for(CustomJToggleButton tb: GroupedToggleButton.
            getAllGroupedButtons()){
            tb.setEnabled(true);
        }
    }
```

```
private long stopPlayback(){
    long currentTick;
    try{
        currentTick = mi.stopMidi();
    }catch (NullPointerException e){
        currentTick = 0;
    }
    paused = false;
    playButton.setText("play");
    trackPresent = false;
    try{
        updateSelection.cancel();
    }catch (NullPointerException ignored){}
    for(CustomJSlider vs: volumeSliders){
        vs.setValue(100);
        vs.setEnabled(false);
    }
    for(CustomJToggleButton tb: GroupedToggleButton.
        getAllGroupedButtons()){
        tb.setSelected(false);
        tb.setEnabled(false);
    }
    return currentTick;
}

private void createUIComponents() {
    FirstMelodyMuteButton = new GroupedToggleButton("Mute", mi.
        melodyTrackIn
```

```
dex());
        FirstRhythmMuteButton = new GroupedToggleButton("Mute", mi.
            rhythmGuitarT
rackIndex());
        FirstBassMuteButton = new GroupedToggleButton("Mute", mi.
            bassTrackIndex(
));
        FirstDrumMuteButton = new GroupedToggleButton("Mute", mi.
            drumTrackIndex(
));
        SecondMelodyMuteButton = new GroupedToggleButton("Mute", mi.
            altMelodyTra
ckIndex());
        SecondRhythmMuteButton = new GroupedToggleButton("Mute", mi.
            altRhythmGui
tarTrackIndex());
        SecondBassMuteButton = new GroupedToggleButton("Mute", mi.
            altBassTrackIn
dex());
        SecondDrumMuteButton = new GroupedToggleButton("Mute", mi.
            altDrumTrackIn
dex());


        FirstMelodySoloButton = new GroupedToggleButton("Solo", mi.
            melodyTrackIn
dex());
        FirstRhythmSoloButton = new GroupedToggleButton("Solo", mi.
            rhythmGuitarT
rackIndex());
```

```
        FirstBassSoloButton = new GroupedToggleButton("Solo", mi.
            bassTrackIndex(
));
        FirstDrumSoloButton = new GroupedToggleButton("Solo", mi.
            drumTrackIndex(
));
        SecondMelodySoloButton = new GroupedToggleButton("Solo", mi.
            altMelodyTra
ckIndex());
        SecondRhythmSoloButton = new GroupedToggleButton("Solo", mi.
            altRhythmGui
tarTrackIndex());
        SecondBassSoloButton = new GroupedToggleButton("Solo", mi.
            altBassTrackIn
dex());
        SecondDrumSoloButton = new GroupedToggleButton("Solo", mi.
            altDrumTrackIn
dex());


        firstMelodySlider = new GroupedJSlider(mi.melodyTrack(), mi.
            melodyTrackI
ndex());
        secondMelodySlider = new GroupedJSlider(mi.altMelodyTrack(),mi.
            altBassTr
ackIndex());
        firstRhythmSlider = new GroupedJSlider(mi.rhythmGuitarTrack(),mi.
            rhythmG
uitarTrackIndex());
        secondRhythmSlider = new GroupedJSlider(mi.altRhythmGuitarTrack()
```

```
            ,mi.alt
RhythmGuitarTrackIndex());
        firstBassSlider = new GroupedJSlider(mi.bassTrack(),mi.
            bassTrackIndex())
;
        secondBassSlider = new GroupedJSlider(mi.altBassTrack(),mi.
            altBassTrackI
ndex());
        firstDrumSlider = new GroupedJSlider(mi.drumTrack(),mi.
            drumTrackIndex())
;
        secondDrumSlider = new GroupedJSlider(mi.altDrumTrack(),mi.
            altDrumTrackI
ndex());
    }
}
```

```
package AminoAcidComposition.AminoAcid

import AminoAcidComposition.SongStructure.Bass.Bass1
import AminoAcidComposition.SongStructure.ChordRoot.CMajor
import AminoAcidComposition.SongStructure.Drum.Drum1
import AminoAcidComposition.SongStructure.MeasureStructure
import AminoAcidComposition.SongStructure.Melody.Melody1
import AminoAcidComposition.SongStructure.Chordal.Chordal1

object A extends AminoAcidAbstract{
  val tracks = new MeasureStructure(CMajor, Bass1, Drum1, Melody1,
```

```
      Chordal1)
}
```

```
package AminoAcidComposition.AminoAcid

import AminoAcidComposition.SongStructure.MeasureStructure

abstract class AminoAcidAbstract {
  val tracks: MeasureStructure
  def addTracks(measureOffset:Int, alternativeTrack:Boolean = false) = {
    tracks.addTrack(measureOffset, alternativeTrack)
  }
}
```

```
package AminoAcidComposition.AminoAcid

import AminoAcidComposition.SongStructure.Bass.Bass1
import AminoAcidComposition.SongStructure.ChordRoot.CMajor
import AminoAcidComposition.SongStructure.Drum.Drum1
import AminoAcidComposition.SongStructure.MeasureStructure
import AminoAcidComposition.SongStructure.Melody.Melody2
import AminoAcidComposition.SongStructure.Chordal.Chordal1

object C extends AminoAcidAbstract {
  val tracks = new MeasureStructure(CMajor, Bass1, Drum1, Melody2,
```

```
      Chordal1)
}
```

```
package AminoAcidComposition.AminoAcid

import AminoAcidComposition.SongStructure.Bass.Bass1
import AminoAcidComposition.SongStructure.ChordRoot.GMajor
import AminoAcidComposition.SongStructure.Drum.Drum2
import AminoAcidComposition.SongStructure.MeasureStructure
import AminoAcidComposition.SongStructure.Melody.Melody4
import AminoAcidComposition.SongStructure.Chordal.Chordal2

object D extends AminoAcidAbstract {
  val tracks = new MeasureStructure(GMajor, Bass1, Drum2, Melody4,
      Chordal2)
}
```

```
package AminoAcidComposition.AminoAcid

import AminoAcidComposition.SongStructure.Bass.Bass1
import AminoAcidComposition.SongStructure.ChordRoot.GMajor
import AminoAcidComposition.SongStructure.Drum.Drum3
import AminoAcidComposition.SongStructure.MeasureStructure
import AminoAcidComposition.SongStructure.Melody.Melody4
import AminoAcidComposition.SongStructure.Chordal.Chordal2
```

```
object E extends AminoAcidAbstract {
  val tracks = new MeasureStructure(GMajor, Bass1, Drum3, Melody4,
      Chordal2)
}
```

```
package AminoAcidComposition.AminoAcid

import AminoAcidComposition.SongStructure.Bass.Bass1
import AminoAcidComposition.SongStructure.ChordRoot.FMajor
import AminoAcidComposition.SongStructure.Drum.Drum3
import AminoAcidComposition.SongStructure.MeasureStructure
import AminoAcidComposition.SongStructure.Melody.Melody1
import AminoAcidComposition.SongStructure.Chordal.Chordal1

object F extends AminoAcidAbstract {
  val tracks = new MeasureStructure(FMajor, Bass1, Drum3, Melody1,
      Chordal1)
}
```

```
package AminoAcidComposition.AminoAcid

import AminoAcidComposition.SongStructure.Bass.Bass1
import AminoAcidComposition.SongStructure.ChordRoot.GMajor
import AminoAcidComposition.SongStructure.Drum.Drum1
```

```
import AminoAcidComposition.SongStructure.MeasureStructure
import AminoAcidComposition.SongStructure.Melody.Melody1
import AminoAcidComposition.SongStructure.Chordal.Chordal2

object G extends AminoAcidAbstract {
  val tracks = new MeasureStructure(GMajor, Bass1, Drum1, Melody1,
      Chordal2)
}
```

```
package AminoAcidComposition.AminoAcid

import AminoAcidComposition.SongStructure.Bass.Bass1
import AminoAcidComposition.SongStructure.ChordRoot.FMajor
import AminoAcidComposition.SongStructure.Drum.Drum3
import AminoAcidComposition.SongStructure.MeasureStructure
import AminoAcidComposition.SongStructure.Melody.Melody3
import AminoAcidComposition.SongStructure.Chordal.Chordal1

object H extends AminoAcidAbstract {
  val tracks = new MeasureStructure(FMajor, Bass1, Drum3, Melody3,
      Chordal1)
}
```

```
package AminoAcidComposition.AminoAcid
```

```
import AminoAcidComposition.SongStructure.Bass.Bass1
import AminoAcidComposition.SongStructure.ChordRoot.AMinor
import AminoAcidComposition.SongStructure.Drum.Drum3
import AminoAcidComposition.SongStructure.MeasureStructure
import AminoAcidComposition.SongStructure.Melody.Melody1
import AminoAcidComposition.SongStructure.Chordal.Chordal1

object I extends AminoAcidAbstract {
  val tracks = new MeasureStructure(AMinor, Bass1, Drum3, Melody1,
      Chordal1)
}



package AminoAcidComposition.AminoAcid

import AminoAcidComposition.SongStructure.Bass.Bass1
import AminoAcidComposition.SongStructure.ChordRoot.CMajor
import AminoAcidComposition.SongStructure.Drum.Drum3
import AminoAcidComposition.SongStructure.MeasureStructure
import AminoAcidComposition.SongStructure.Melody.Melody3
import AminoAcidComposition.SongStructure.Chordal.Chordal1

object K extends AminoAcidAbstract {
  val tracks = new MeasureStructure(CMajor, Bass1, Drum3, Melody3,
      Chordal1)
}
```

```
package AminoAcidComposition.AminoAcid

import AminoAcidComposition.SongStructure.Bass.Bass2
import AminoAcidComposition.SongStructure.ChordRoot.AMinor
import AminoAcidComposition.SongStructure.Drum.Drum3
import AminoAcidComposition.SongStructure.MeasureStructure
import AminoAcidComposition.SongStructure.Melody.Melody1
import AminoAcidComposition.SongStructure.Chordal.Chordal1

object L extends AminoAcidAbstract {
  val tracks = new MeasureStructure(AMinor, Bass2, Drum3, Melody1,
      Chordal1)
}
```

```
package AminoAcidComposition.AminoAcid

import AminoAcidComposition.SongStructure.Bass.Bass1
import AminoAcidComposition.SongStructure.ChordRoot.CMajor
import AminoAcidComposition.SongStructure.Drum.Drum3
import AminoAcidComposition.SongStructure.MeasureStructure
import AminoAcidComposition.SongStructure.Melody.Melody1
import AminoAcidComposition.SongStructure.Chordal.Chordal1

object M extends AminoAcidAbstract {
  val tracks = new MeasureStructure(CMajor, Bass1, Drum3, Melody1,
      Chordal1)
```

```
}
```

```
package AminoAcidComposition.AminoAcid

import AminoAcidComposition.SongStructure.Bass.Bass1
import AminoAcidComposition.SongStructure.ChordRoot.GMajor
import AminoAcidComposition.SongStructure.Drum.Drum2
import AminoAcidComposition.SongStructure.MeasureStructure
import AminoAcidComposition.SongStructure.Melody.Melody2
import AminoAcidComposition.SongStructure.Chordal.Chordal2

object N extends AminoAcidAbstract{
  val tracks = new MeasureStructure(GMajor, Bass1, Drum2, Melody2,
      Chordal2)
}
```

```
package AminoAcidComposition.AminoAcid

import AminoAcidComposition.SongStructure.Bass.Bass1
import AminoAcidComposition.SongStructure.ChordRoot.GMajor
import AminoAcidComposition.SongStructure.Drum.Drum2
import AminoAcidComposition.SongStructure.MeasureStructure
import AminoAcidComposition.SongStructure.Melody.Melody1
import AminoAcidComposition.SongStructure.Chordal.Chordal2
```

```
object P extends AminoAcidAbstract {
  val tracks = new MeasureStructure(GMajor, Bass1, Drum2, Melody1,
      Chordal2)
}
```

```
package AminoAcidComposition.AminoAcid

import AminoAcidComposition.SongStructure.Bass.Bass1
import AminoAcidComposition.SongStructure.ChordRoot.GMajor
import AminoAcidComposition.SongStructure.Drum.Drum3
import AminoAcidComposition.SongStructure.MeasureStructure
import AminoAcidComposition.SongStructure.Melody.Melody2
import AminoAcidComposition.SongStructure.Chordal.Chordal2

object Q extends AminoAcidAbstract {
  val tracks = new MeasureStructure(GMajor, Bass1, Drum3, Melody2,
      Chordal2)
}
```

```
package AminoAcidComposition.AminoAcid

import AminoAcidComposition.SongStructure.Bass.Bass1
import AminoAcidComposition.SongStructure.ChordRoot.GMajor
import AminoAcidComposition.SongStructure.Drum.Drum3
import AminoAcidComposition.SongStructure.MeasureStructure
```

```
import AminoAcidComposition.SongStructure.Melody.Melody3
import AminoAcidComposition.SongStructure.Chordal.Chordal2

object R extends AminoAcidAbstract {
  val tracks = new MeasureStructure(GMajor, Bass1, Drum3, Melody3,
      Chordal2)
}
```

```
package AminoAcidComposition.AminoAcid

import AminoAcidComposition.SongStructure.Bass.Bass1
import AminoAcidComposition.SongStructure.ChordRoot.GMajor
import AminoAcidComposition.SongStructure.Drum.Drum1
import AminoAcidComposition.SongStructure.MeasureStructure
import AminoAcidComposition.SongStructure.Melody.Melody2
import AminoAcidComposition.SongStructure.Chordal.Chordal2

object S extends AminoAcidAbstract {
  val tracks = new MeasureStructure(GMajor, Bass1, Drum1, Melody2,
      Chordal2)
}
```

```
package AminoAcidComposition.AminoAcid

import AminoAcidComposition.SongStructure.Bass.Bass1
```

```
import AminoAcidComposition.SongStructure.ChordRoot.CMajor
import AminoAcidComposition.SongStructure.Drum.Drum2
import AminoAcidComposition.SongStructure.MeasureStructure
import AminoAcidComposition.SongStructure.Melody.Melody2
import AminoAcidComposition.SongStructure.Chordal.Chordal1

object T extends AminoAcidAbstract {
  val tracks = new MeasureStructure(CMajor, Bass1, Drum2, Melody2,
      Chordal1)
}
```

```
package AminoAcidComposition.AminoAcid

import AminoAcidComposition.SongStructure.Bass.Bass1
import AminoAcidComposition.SongStructure.ChordRoot.AMinor
import AminoAcidComposition.SongStructure.Drum.Drum2
import AminoAcidComposition.SongStructure.MeasureStructure
import AminoAcidComposition.SongStructure.Melody.Melody1
import AminoAcidComposition.SongStructure.Chordal.Chordal1

object V extends AminoAcidAbstract {
  val tracks = new MeasureStructure(AMinor, Bass1, Drum2, Melody1,
      Chordal1)
}
```

```
package AminoAcidComposition.AminoAcid

import AminoAcidComposition.SongStructure.Bass.Bass2
import AminoAcidComposition.SongStructure.ChordRoot.FMajor
import AminoAcidComposition.SongStructure.Drum.Drum3
import AminoAcidComposition.SongStructure.MeasureStructure
import AminoAcidComposition.SongStructure.Melody.Melody2
import AminoAcidComposition.SongStructure.Chordal.Chordal1

object W extends AminoAcidAbstract {
  val tracks = new MeasureStructure(FMajor, Bass2, Drum3, Melody2,
      Chordal1)
}
```

```
package AminoAcidComposition.AminoAcid

import AminoAcidComposition.SongStructure.Bass.Bass1
import AminoAcidComposition.SongStructure.ChordRoot.FMajor
import AminoAcidComposition.SongStructure.Drum.Drum3
import AminoAcidComposition.SongStructure.MeasureStructure
import AminoAcidComposition.SongStructure.Melody.Melody2
import AminoAcidComposition.SongStructure.Chordal.Chordal1

object Y extends AminoAcidAbstract {
  val tracks = new MeasureStructure(FMajor, Bass1, Drum3, Melody2,
      Chordal1)
}
```

```
package AminoAcidComposition.SongStructure.Bass

import AminoAcidComposition.SongStructure.Note

object Bass1 extends BassTrait {
  def getTrack(rootNote:Int, altTrack:Boolean = false, firstChord:
      Boolean = true
) = {
    val notes = if(firstChord) {
      Seq(
        (rootNote-12,one,eighth),
        (rootNote-12,oneAnd,eighth),
        (rootNote-12,two,quarter)
      )
    } else {
      Seq(
        (rootNote-12,three,eighth),
        (rootNote-12,threeAnd,eighth),
        (rootNote-12,four,quarter)
      )
    }

    if(!altTrack){
      notes.map(n=>Note(mc.bassTrack,mc.bassTrackIndex,n._1,velocity,n._2
          ,n._3))
    } else {
```

```
      notes.map(n=>Note(mc.altBassTrack,mc.altBassTrackIndex,n._1,
         velocity,n._2,
n._3))
    }

  }
}
```

```
package AminoAcidComposition.SongStructure.Bass

import AminoAcidComposition.SongStructure.Note

object Bass2 extends BassTrait {
  def getTrack(rootNote:Int, altTrack:Boolean = false, firstChord:
     Boolean = true
) = {
    val notes = if(firstChord) {
      Seq(
        (rootNote-12,one,eighth),
        (rootNote-5,oneAnd,eighth),
        (rootNote-12,two,quarter)
      )
    } else {
      Seq(
        (rootNote-12,three,eighth),
        (rootNote-5,threeAnd,eighth),
        (rootNote-12,four,quarter)
```

```
      )
    }
    if(!altTrack){
      notes.map(n=>Note(mc.bassTrack,mc.bassTrackIndex,n._1,velocity,n._2
          ,n._3))
    } else {
      notes.map(n=>Note(mc.altBassTrack,mc.altBassTrackIndex,n._1,
          velocity,n._2,
n._3))
    } }
}
```

```
package AminoAcidComposition.SongStructure.Bass

import AminoAcidComposition.SongStructure.{Instrument, MusicCommon}

trait BassTrait extends Instrument with MusicCommon {
  val velocity = 100
}
```

```
package AminoAcidComposition.SongStructure.Chordal

import AminoAcidComposition.SongStructure.Note

object Chordal1 extends ChordalTrait {
```

```
  def getTrack(rootNote:Int, altTrack:Boolean = false, firstChord:
      Boolean = true
) = {
    if (!altTrack) {
      val chord = rootNote match {
        case 45 => aMinorGuitar
        case 48 => cMajorGuitar
        case 41 => fMajorGuitar
        case 43 => gMajorGuitar
      }

      def downStroke = downStrokeCurry(chord) _
      def upStroke = upStrokeCurry(chord) _

      val rhythmTrack = if (firstChord) {
        downStroke(one, measure, velocity) ++ downStroke(two, dottedHalf,
            (veloc
ity * 1.2).toInt)
      } else {
        downStroke(three, half, velocity) ++ upStroke(three + sixteenth,
           dottedQ
uarter + sixteenth, velocity) ++ downStroke(threeAnd, dottedQuarter,
    velocity) +
+ downStroke(four, quarter, (velocity * 1.2).toInt)
      }

      for (n <- rhythmTrack) yield {
        Note(mc.rhythmGuitarTrack, trackNumber, n._3, n._4, n._1, n._2)
      }
```

```
    } else {
      val chord = rootNote match {
        case 45 => aMinorPiano
        case 48 => cMajorPiano
        case 41 => fMajorPiano
        case 43 => gMajorPiano
      }

      val notes = if (firstChord) {
        chord.flatMap(n =>
          Seq(
            (one, quarter, n),
            (two, quarter, n)
          )
        )
      } else {
        chord.flatMap(n =>
          Seq(
            (three, sixteenth, n),
            (three + sixteenth, sixteenth, n),
            (threeAnd, eighth, n),
            (four, quarter, n)
          )
        )
      }
      notes.map(n=>Note(mc.altRhythmGuitarTrack, altTrackNumber,n._3,
          altVelocity
,n._1,n._2))
    }
```

```
  }
}



package AminoAcidComposition.SongStructure.Chordal

import AminoAcidComposition.SongStructure.Note
object Chordal2 extends ChordalTrait {
  def getTrack(rootNote:Int, altTrack:Boolean = false, firstChord:
      Boolean = true
) = {
    if (!altTrack) {
      val chord = rootNote match {
        case 45 => aMinorGuitar
        case 48 => cMajorGuitar
        case 41 => fMajorGuitar
        case 43 => gMajorGuitar
      }

      def downStroke = downStrokeCurry(chord) _
      def upStroke = upStrokeCurry(chord) _

      val rhythmTrack = if (firstChord) {
        downStroke(one, half, velocity) ++ downStroke(two, quarter,
            velocity) ++
 downStroke(twoAnd, eighth, velocity) ++ upStroke(twoAnd + sixteenth,
    sixteenth,
 velocity)
```

```
    } else {
      downStroke(three, half, velocity) ++ downStroke(threeAnd,
          dottedQuarter,
velocity) ++ downStroke(four, quarter, velocity) ++ downStroke(fourAnd,
    eighth,
velocity)
    }

    for (n <- rhythmTrack) yield {
      Note(mc.rhythmGuitarTrack, trackNumber, n._3, n._4, n._1, n._2)
    }
  } else {
    val chord = rootNote match {
      case 45 => aMinorPiano
      case 48 => cMajorPiano
      case 41 => fMajorPiano
      case 43 => gMajorPiano
    }

    val notes = if (firstChord) {
      chord.flatMap(n =>
        Seq(
          (one, quarter, n),
          (two, eighth, n),
          (twoAnd,sixteenth, n),
          (twoAnd+sixteenth,sixteenth, n)
        )
      )
    } else {
```

```
        chord.flatMap(n =>
          Seq(
            (three, eighth, n),
            (threeAnd,eighth, n),
            (four, eighth, n),
            (fourAnd,eighth,n)
          )
        )
      }
      notes.map(n=>Note(mc.altRhythmGuitarTrack, altTrackNumber,n._3,
          altVelocity
,n._1,n._2))
    }
  }
}
```

```
package AminoAcidComposition.SongStructure.Chordal

import AminoAcidComposition.SongStructure.{Instrument, MusicCommon}

trait ChordalTrait extends Instrument with MusicCommon {
  protected val trackNumber = mc.rhythmGuitarTrackIndex
  protected val altTrackNumber = mc.altRhythmGuitarTrackIndex
  protected val velocity = 70
  protected val altVelocity = 70

  protected val aMinorGuitar = Seq(45, 52, 57, 60, 64)
```

```
  protected val cMajorGuitar = Seq(48, 52, 55, 60, 64)
  protected val fMajorGuitar = Seq(41, 48, 53, 57, 60, 65)
  protected val gMajorGuitar = Seq(43, 47, 50, 55, 62, 67)
  //  protected val gMajorGuitar = Seq(47, 54, 59, 62, 66)


  protected val aMinorPiano = Seq(48, 52, 57, 60, 64, 69)
  protected val cMajorPiano = Seq(48, 52, 55, 60, 64, 67)
  protected val fMajorPiano = Seq(48, 53, 57, 60, 65, 69)
  protected val gMajorPiano = Seq(50, 55, 59, 62, 67, 71)

  protected def downStrokeCurry(chord: Seq[Int])(start: Int, duration:
     Int, velo
city: Int) = {
    for (i <- chord.indices) yield {
      if (i == chord.length - 1) {
        (start + i, duration - i, chord(i), (velocity * 1.1).toInt)
      } else {
        (start + i, duration - i, chord(i), velocity)
      }
    }
  }

  protected def upStrokeCurry(chord: Seq[Int])(start: Int, duration: Int
     , veloci
ty: Int) = {
    for (i <- chord.indices) yield {
      if (i == 0) {
        (start + i, duration - i, chord(chord.length - 1 - i), (velocity
```

```
          * 1.1).
toInt)
      } else {
        (start + i, duration - i, chord(chord.length - 1 - i), velocity)
      }
    }
  }
}
```

```
package AminoAcidComposition.SongStructure.ChordRoot

object AMinGMaj extends Chord{
  val rootNotes = (45, 43)
}
```

```
package AminoAcidComposition.SongStructure.ChordRoot

object AMinor extends Chord{
  val rootNotes = (45, 45)
}
```

```
package AminoAcidComposition.SongStructure.ChordRoot
```

```
trait Chord {
  val rootNotes: (Int, Int)
}
```

```
package AminoAcidComposition.SongStructure.ChordRoot

object CMajFMaj extends Chord{
  val rootNotes = (48, 41)
}
```

```
package AminoAcidComposition.SongStructure.ChordRoot

object CMajGMaj extends Chord{
  val rootNotes = (48, 43)
}
```

```
package AminoAcidComposition.SongStructure.ChordRoot

object CMajor extends Chord{
  val rootNotes = (48, 48)
}
```

```
package AminoAcidComposition.SongStructure.ChordRoot

object FMajor extends Chord{
  val rootNotes = (41, 41)
}
```

```
package AminoAcidComposition.SongStructure.ChordRoot

object GMajor extends Chord{
  val rootNotes = (43, 43)
}
```

```
package AminoAcidComposition.SongStructure.Drum

import AminoAcidComposition.SongStructure.Note

object Drum1 extends DrumTrait {
  def getTrack(rootNote:Int, altTrack:Boolean = false, firstChord:
      Boolean = true
) = {
    val drumTrack = if(altTrack){
      mc.altDrumTrack
    }else{
      mc.drumTrack
```

```
    }

    val notes = if (!altTrack) {
      Seq(
        (35, 120, one, quarter)
      )
    } else {
      Seq(
        (35, 120, oneAnd, eighth)
      )
    }
    notes.map(n => Note(drumTrack, 9, n._1, n._2, n._3, n._4))
  }
}
```

```
package AminoAcidComposition.SongStructure.Drum

import AminoAcidComposition.SongStructure.Note

object Drum2 extends DrumTrait {
  def getTrack(rootNote:Int, altTrack:Boolean = false, firstChord:
      Boolean = true
) = {
    val track = if(altTrack){
      mc.altDrumTrack
    }else{
      mc.drumTrack
```

```
    }

    val notes = if (!altTrack) {
      Seq(
        (35, 120, one, quarter),
        (35, 120, three, quarter)
      )
    } else {
      Seq(
        (35, 120, oneAnd, eighth),
        (35, 120, threeAnd, eighth)
      )
    }
    notes.map(n => Note(track, 9, n._1, n._2, n._3, n._4))
  }
}
```

```
package AminoAcidComposition.SongStructure.Drum

import AminoAcidComposition.SongStructure.Note

object Drum3 extends DrumTrait {
  def getTrack(rootNote:Int, altTrack:Boolean = false, firstChord:
      Boolean = true
) = {
    val track = if(altTrack){
      mc.altDrumTrack
```

```
    }else{
      mc.drumTrack
    }
    val notes = if (!altTrack) {
      Seq(
        (35, 120, one, quarter),
        (35, 120, two, quarter),
        (35, 120, three, quarter),
        (35, 120, four, quarter)
      )
    } else {
      Seq(
        (35, 120, oneAnd, eighth),
        (35, 120, twoAnd, eighth),
        (35, 120, threeAnd, eighth),
        (35, 120, fourAnd, eighth)
      )
    }
    notes.map(n => Note(track, 9, n._1, n._2, n._3, n._4))
  }
}
```

```
package AminoAcidComposition.SongStructure.Drum

import AminoAcidComposition.SongStructure.{Instrument,MusicCommon}

trait DrumTrait extends Instrument with MusicCommon{}
```

```
package AminoAcidComposition.SongStructure.Melody

import AminoAcidComposition.SongStructure.Note

object Melody1 extends MelodyTrait{
  val notes = Seq(
    (c + octave, one),
    (a, two),
    (b, three),
    (g, four)
  )

  def getTrack(rootNote:Int, altTrack:Boolean = false, firstChord:
      Boolean = true
) = {
    for(n<-notes) yield {
      if(!altTrack) {
        Note(mc.melodyTrack, trackNumber, n._1, velocity, n._2, quarter)
      }else{
        Note(mc.altMelodyTrack, altTrackNumber, n._1, velocity + 5, n._2,
            quarte
r)
      }
    }
  }
}
```

```
//val notes = Seq(
//(c+octave, one),
//(a, two),
//(b, three),
//(g, four)
//)

//val notes = Seq(
//(a, one),
//(c+octave, two),
//(d+octave, three),
//(c+octave,four)
//)

//val notes = Seq(
//(c+octave, one),
//(a, two),
//(b, three),
//(a,four)
//)

//val notes = Seq(
//(a, one),
//(b, two),
//(d+octave, three),
//(c+octave,four)
//)
```

```
package AminoAcidComposition.SongStructure.Melody

import AminoAcidComposition.SongStructure.Note

object Melody2 extends MelodyTrait{
  val notes = Seq(
    (a, one),
    (c + octave, two),
    (d + octave, three),
    (c + octave, four)
  )

  def getTrack(rootNote:Int, altTrack:Boolean = false, firstChord:
      Boolean = true
) = {
    for(n<-notes) yield {
      if(!altTrack) {
        Note(mc.melodyTrack, trackNumber, n._1, velocity, n._2, quarter)
      }else{
        Note(mc.altMelodyTrack, altTrackNumber, n._1, velocity + 5, n._2,
            quarte
r)
      }
    }
  }
}
```

```
package AminoAcidComposition.SongStructure.Melody

import AminoAcidComposition.SongStructure.Note

object Melody3 extends MelodyTrait{
  val notes = Seq(
    (e + octave, one),
    (c + octave, two),
    (d + octave, three),
    (c + octave, four)
  )

  def getTrack(rootNote:Int, altTrack:Boolean = false, firstChord:
      Boolean = true
) = {
    for(n<-notes) yield {
      if(!altTrack) {
        Note(mc.melodyTrack, trackNumber, n._1, velocity, n._2, quarter)
      }else{
        Note(mc.altMelodyTrack, altTrackNumber, n._1, velocity + 5, n._2,
            quarte
r)
      }
    }
  }
}
```

```
package AminoAcidComposition.SongStructure.Melody

import AminoAcidComposition.SongStructure.Note

object Melody4 extends MelodyTrait{
  val notes = Seq(
    (a, one),
    (b, two),
    (d + octave, three),
    (c + octave, four)
  )

  def getTrack(rootNote:Int, altTrack:Boolean = false, firstChord:
      Boolean = true
) = {
    for(n<-notes) yield {
      if(!altTrack) {
        Note(mc.melodyTrack, trackNumber, n._1, velocity, n._2, quarter)
      }else{
        Note(mc.altMelodyTrack, altTrackNumber, n._1, velocity + 5, n._2,
            quarte
r)
      }
    }
  }
}
```

```
package AminoAcidComposition.SongStructure.Melody

import AminoAcidComposition.SongStructure.{Instrument, MusicCommon}

trait MelodyTrait extends Instrument with MusicCommon{
  val trackNumber = mc.melodyTrackIndex
  val altTrackNumber = mc.altMelodyTrackIndex
  val velocity = 50
}
```

```
package AminoAcidComposition.SongStructure

import javax.sound.midi.Track

case class Note(track:Track, trackNumber:Int, note:Int, velocity:Int,
   start:Int,
 duration:Int)

trait Instrument {
  def getTrack(rootNote:Int, altTrack:Boolean = false, firstChord:
     Boolean = true
):Seq[Note]
}
```

```
package AminoAcidComposition.SongStructure

import javax.sound.midi.Track

case class Note(track:Track, trackNumber:Int, note:Int, velocity:Int,
   start:Int,
 duration:Int)

trait Instrument {
  def getTrack(rootNote:Int, altTrack:Boolean = false, firstChord:
     Boolean = true
):Seq[Note]
}
```

```
package AminoAcidComposition.SongStructure

import AminoAcidComposition.SongStructure.Bass.BassTrait
import AminoAcidComposition.SongStructure.ChordRoot.Chord
import AminoAcidComposition.SongStructure.Drum.DrumTrait
import AminoAcidComposition.SongStructure.Melody.MelodyTrait
import AminoAcidComposition.SongStructure.Chordal.ChordalTrait

class MeasureStructure(ch: Chord, ba: BassTrait, dr: DrumTrait, m:
   MelodyTrait,
rg: ChordalTrait) extends MusicCommon {
  def addTrack(measureOffset:Int, alternateTrack:Boolean = false) = {
```

```
    for{n <-
        dr.getTrack(ch.rootNotes._1, alternateTrack) ++
          rg.getTrack(ch.rootNotes._1, alternateTrack, firstChord = true)
              ++
          rg.getTrack(ch.rootNotes._2, alternateTrack, firstChord = false
             ) ++
          ba.getTrack(ch.rootNotes._1, alternateTrack, firstChord = true)
              ++
          ba.getTrack(ch.rootNotes._2, alternateTrack, firstChord = false
             ) ++
          m.getTrack(ch.rootNotes._1, alternateTrack)
    } {
      mc.addNote(n.track,n.trackNumber, n.note, n.velocity, measureOffset
           + n.st
art, n.duration)
    }
  }
}
```

```
package AminoAcidComposition.SongStructure

import java.io.File
import javax.sound.midi._

trait MusicCommon {
  val mc = MusicCommon
  val noteDivision = 64
```

```
  val sixteenth = noteDivision/4
  val eighth = noteDivision/2
  val quarter = noteDivision
  val dottedQuarter = 3*noteDivision/2
  val half = 2*noteDivision
  val dottedHalf = 5*noteDivision/2

  val one = 0
  val oneAnd = eighth
  val two = quarter
  val twoAnd = quarter + eighth
  val three = 2*quarter
  val threeAnd = 2*quarter + eighth
  val four = 3*quarter
  val fourAnd = 3*quarter + eighth
  val measure = 4*quarter

  val c = 72
  val d = 74
  val e = 76
  val f = 77
  val g = 79
  val a = 81
  val b = 83
  val octave = 12
  val flat = -1
  val sharp = 1
}
```

```
object MusicCommon extends MusicCommon{
  private val sequence = new Sequence(Sequence.PPQ, noteDivision)
  private val sequencer = MidiSystem.getSequencer
  private val inst = MidiSystem.getSynthesizer.getDefaultSoundbank.
      getInstrument
s

  val rhythmGuitarTrackIndex = 0
  val altRhythmGuitarTrackIndex = 1
  val melodyTrackIndex = 2
  val altMelodyTrackIndex = 3
  val bassTrackIndex = 4
  val altBassTrackIndex = 5
  val drumTrackIndex = 6
  val altDrumTrackIndex = 7
  val rhythmGuitarTrack = sequence.createTrack()
  val altRhythmGuitarTrack = sequence.createTrack()
  val melodyTrack = sequence.createTrack()
  val altMelodyTrack = sequence.createTrack()
  val bassTrack = sequence.createTrack()
  val altBassTrack = sequence.createTrack()
  val drumTrack = sequence.createTrack()
  val altDrumTrack = sequence.createTrack()

  def changeVolume(track: Track, trackIndex: Int, volume: Int) = {
    addMidiEvent(track, ShortMessage.CONTROL_CHANGE, trackIndex, 7,
        volume, sequ
encer.getTickPosition)
```

```
  }

  //param1 and param2 are poorly named, they are respectively
  //note and velocity for adding notes
  //program and bank for changing instrument
  //something and something for changing tempo
  private def addMidiEvent(track: Track, midiCommand: Int, trackNumber:
      Int, par
am1: Int, param2: Int, location: Long) = {
    val message = new ShortMessage()
    try {
      message.setMessage(midiCommand,
        trackNumber,
        param1,
        param2)
    }
    track.add(new MidiEvent(message, location))
  }

  def changeInstrument(track:Track, trackNumber:Int, instrument:Int) = {
    addMidiEvent(track, ShortMessage.PROGRAM_CHANGE, trackNumber, inst(
        instrume
nt).getPatch.getProgram, inst(instrument).getPatch.getBank, sequencer.
    getTickPos
ition)
  }

  def muteTrack(trackIndex:Int) = {
    if(sequencer.getTrackMute(trackIndex)){
```

```
      sequencer.setTrackMute(trackIndex,false)
    } else {
      sequencer.setTrackMute(trackIndex,true)
    }
  }

  def soloTrack(trackIndex:Int) = {
    if(sequencer.getTrackSolo(trackIndex)){
      sequencer.setTrackSolo(trackIndex,false)
    } else {
      sequencer.setTrackSolo(trackIndex,true)
    }
  }

  def addNote(track:Track, trackNumber:Int, note:Int, velocity: Int,
      start: Int,
 duration:Int) = {
    addMidiEvent(track, ShortMessage.NOTE_ON, trackNumber, note,
       velocity, start
)
    addMidiEvent(track, ShortMessage.NOTE_OFF, trackNumber, note,
       velocity, star
t+duration)
  }

  def listInstruments() = {
    inst.map(_.getName)
  }
```

```
  def writeMidi(file:String) = {
    val outputFile = new File(file)
    if(outputFile.exists()){
      outputFile.delete()
      outputFile.createNewFile()
    }
    MidiSystem.write(sequence, 1, outputFile)
  }

  def playMidi(bpm:Float) = {
    sequencer.open()
    sequencer.setSequence(sequence)
    sequencer.setTempoInBPM(bpm)
    sequencer.start()
  }

  def stopMidi() = {
    val location = try {
      val current = sequencer.getTickPosition
      sequencer.stop()
      sequencer.close()
      current
    }catch{
      case _:Throwable => 0
    }
    for (track <- Seq(rhythmGuitarTrack, altRhythmGuitarTrack,
       melodyTrack, altM
elodyTrack, bassTrack, altBassTrack, drumTrack, altDrumTrack)) {
      while (track.size() != 0) {
```

```
        track.remove(track.get(0))
      }
    }
    location
  }

  def setLocation(location:Long) = {
    sequencer.setTickPosition(location)
  }
}
```

```
package AminoAcidComposition

/**
 * Created by Aaron Kosmatin on 7/12/15.
 * The main interface for midi generation
 */

import javax.sound.midi.Track

import AminoAcidComposition.AminoAcid.AminoAcidAbstract
import AminoAcidComposition.SongStructure.MusicCommon

object MidiInterface extends MusicCommon {
  val rhythmGuitarTrackIndex = mc.rhythmGuitarTrackIndex
  val altRhythmGuitarTrackIndex = mc.altRhythmGuitarTrackIndex
  val melodyTrackIndex = mc.melodyTrackIndex
```

```
val altMelodyTrackIndex = mc.altMelodyTrackIndex
val bassTrackIndex = mc.bassTrackIndex
val altBassTrackIndex = mc.altBassTrackIndex
val drumTrackIndex = mc.drumTrackIndex
val altDrumTrackIndex = mc.altDrumTrackIndex

val rhythmGuitarTrack = mc.rhythmGuitarTrack
val altRhythmGuitarTrack = mc.altRhythmGuitarTrack
val melodyTrack = mc.melodyTrack
val altMelodyTrack = mc.altMelodyTrack
val bassTrack = mc.bassTrack
val altBassTrack = mc.altBassTrack
val drumTrack = mc.drumTrack
val altDrumTrack = mc.altDrumTrack

/**
 * Generates the midi sequence for playing or saving.
 * @param sequence1 String of amino acids for the primary music part
 * @param sequence2 String of amino acids for the secondaty music part
 */
def createMidi(sequence1:String, sequence2:String) = {
  if (!sequence1.isEmpty) {
    _createMidi(sequence1)
  }
  if (!sequence2.isEmpty) {
    _createMidi(sequence2, true)
  }

  def _createMidi(seq:String, alt:Boolean = false) = {
```

```
    for (i <- seq.indices) {
      try {
        val aminoAcid = {
          val objectName = "AminoAcidComposition.AminoAcid." + seq.
              charAt(i) +
"$"
          val cons = Class.forName(objectName).getDeclaredConstructors
          cons(0).setAccessible(true)
          cons(0).newInstance().asInstanceOf[AminoAcidAbstract]
        }
        aminoAcid.addTracks(measure * i, alt)
      }
      catch {
        case _ => {}
      }
    }
  }
}

/**
 * Plays the midi tracks
 * @param bpm Beats per minute, default is 120
 */
def playMidi(bpm:Int = 120) = mc.playMidi(bpm)

/**
 * Skip to a location in the midi track
 * @param location the tick number
 */
```

```
  def setLocation(location:Long) = mc.setLocation(location)


  /**
   * stops and clears the midi tracks
   * @return the tick where playback was stopped
   */
  def stopMidi() = mc.stopMidi()


  /**
   * Returns the list of supported midi instruments
   * @return the list of supported midi instruments
   */
  def listInstruments() = mc.listInstruments()


  /**
   * Change the instrument in a track before playback
   * @param track
   * @param trackNumber
   * @param instrument
   * @return
   */
  def setInstrument(track:Track, trackNumber:Int, instrument:Int) = mc.
      changeIns
trument(track, trackNumber, instrument)
```

```
  /**
   * Mute track
   * @param trackIndex
   */
  def muteTrack(trackIndex:Int) = mc.muteTrack(trackIndex)

  /**
   *
   * @param trackIndex
   */
  def soloTrack(trackIndex:Int) = mc.soloTrack(trackIndex)


  /**
   *
   * @param filename
   * @return
   */
  def writeMidi(filename:String) = {
    mc.writeMidi(filename)
  }

  def changeVolume(track:Track, trackIndex:Int, volume:Int) = mc.
      changeVolume(tr
ack, trackIndex, volume)
}
```