SEAMLESS SONIC SCENERY WITH AN OPEN SOURCE AUDIO ENGINE

JACOB HILLS
Bachelor of Music, University of Lethbridge, 2015

A Support Paper
Submitted to The School of Graduate Studies
of the University of Lethbridge
in Partial Fulfilment of the
Requirements for the Degree

MASTER OF FINE ARTS

Department of New Media
University of Lethbridge
LETHBRIDGE, ALBERTA, CANADA

SEAMLESS SONIC SCENERY WITH AN OPEN SOURCE AUDIO ENGINE

JACOB HILLS

Date of Defence August 11, 2017

James Graham Supervisor	Associate Professor	M.F.A.
Dr. Rolf Boon Thesis Examination Committee Member	Associate Professor	Ph.D.
Dr. Georg Boenn, Thesis Examination Committee Member	Assistant Professor	Ph.D.
Dr. D. Andrew Stewart Chair, Thesis Examination Committee	Assistant Professor	D.Mus.

Acknowledgements

A big thank you to my wife, who became a new mom in the middle of this whole process. She has been patient with my physical and mental absence even when she could have used help with our new son.

Also, a big thank you to my advisory committee, who inspired and assisted me through this project.

Abstract

This support document contains my research into adaptive music using the horizontal resequencing technique. It outlines the music composing process I used to create three main musical themes, and the transitions that facilitate the smooth transition between them. It also discusses the creation and use of a software music engine created in the Pure Data programming environment, which utilizes the music that I composed, for a truly adaptive musical experience. In addition, this paper discusses the customization and expansion of the software. I chose to undertake this project because I felt that the current software that can organize music in this way is often complex to use and come with user agreements that charge the user based on the scope, budget or success of any given project. I felt that creating and distributing a simple open source audio resequencing program would provide a valuable alternative to the tools currently available. My engine is free of charge and anyone can use and modify it as they see fit. This software is a tool for educators to help teach about interactive music. Composers who want to experiment with horizontal resequencing with musical transitions will find that it is a useful tool. It is also a resource for makers, curators, and artists who want to add adaptive music to video games, public displays or other modern media. My project is under the creative common license CC BY-SA4.0 which means that anyone can download it, modify or remix it and use it however they choose as long as my work is recognized and any remix is shared under the same license. It is my hope that this open approach will foster creativity and help more people create immersive interactive experiences. The software can be downloaded at http://hillsmfa.weebly.com .

Contents

Chapter 1. Introduction

Interactive Audio

Throughout my research I have had many discussions with gamers, developers, and other graduate students about interactive media. In conversations with them one theme emerges time and time again: “The sound sold me on the experience”. That is not to say that alone the audio alone is the only thing that matters when dealing with immersive experiences but, it is an important aspect. A graduate student studying virtual reality confessed that he ducked as he heard the sound of wings flapping behind him. A gamer when talking about the game *Amnesia: The Dark Descent* (Frictional Games, 2010) confided in me that were it not for the audio design in that game he would not have been nearly so terrified. The right audio in a situation can have a tremendous impact on the user. In an article by Raymond Usher, an experiment is documented wherein game players are monitored while playing a game first without audio, and then with audio. (Usher, 2015). In this study, he found that breathing and heart rate increased when players experienced the audio in conjunction with the visual stimuli. This result really does not come as a surprise. In an article on the history of video game music author Glenn McDonald, suggests a simple experiment of playing through Super Mario Brothers for the Nintendo Entertainment System without audio and see if the experience is different than with the audio. (McDonald, 2005). There is something to be said when an action on our part leads to not only a visual response but, an auditory one as well. It was my own experiences with interactive media that piqued my interest in the area and as I spoke about interactivity and audio with others, I became convinced that this would be an excellent area in which to do some further exploration.

This thesis and graduate research project reflects my considerable interest and experience in music, and specifically the field of Interactive Audio. I wanted to learn about what was currently being done with interactivity in music and somehow contribute to that field. I found that there are many different techniques used in interactive music, but the one that became of most interest to me was the horizontal resequencing technique. Horizontal Resequencing has the potential to provide interactivity while using music that consists of a traditional harmony and melody. There are many different software solutions that can facilitate horizontal resequencing, but they come with obstacles. Many of the currently available solutions will charge a fee based on the size, scope and budget of a project. They can also be very complex to learn to use because of their many capabilities. Most of the companies that create these types of software are closed to community development, which means that they cannot be customized to specific projects. In response to these obstacles I developed a free and open source audio engine for interactivity in music using the horizontal resequencing technique, the development of which is documented in this paper. In contrast to current solutions, my engine is simple to use because it has one purpose. Additionally, there are no fees attached to its use. The source code is available so that it can be easily adapted to suit any interactive musical setting, both in the digital as well as the physical world. It is lightweight and cross-platform, meaning that it will run easily on older or less powerful systems, regardless of the operating system. I also document the composing of the adaptive music that I put into the engine. It is my hope that this research and project will be used in future instances of interactive media, ranging from public displays to video games. I also hope that the engine will be used as an educational tool to help composers create adaptive scores, and programmers in their study of interactive media software.

Historical Developments in Interactive Digital Audio

The development of interactivity in audio has been a major part of digital media for a few decades. It has shaped modern media into what it is today and is perhaps most noticeably present in the video game industry. It is true that in some forms of media interactive audio has been around for hundreds of years, the interactivity being provided by humans. Since my research is mainly focused in digital audio, my study will be confined to audio created or manipulated by electronics. In many instances innovations were driven by video games. However, before video games, pinball was a popular pastime. In the 1930s pinball machine manufacturers, wanting to draw more customers to their machines added bells and buzzers, a feature that had already been added to gambling slot machines with great success. The sounds were so effective at calling out to potential customers and generating excitement in current customers that pinball and slot machine makers continue to use audio to this day. The sounds are a little different in present day, in that now the sounds are digitally produced rather than mechanically made, but they fulfill much of the same function; they call out to customers, and generate excitement during the experience.

The first real step into digital interactive audio began in 1972 with an arcade tennis simulation called *Pong* made by a company named Atari. In the game there are three basic sounds which indicate a hit, a miss, and a ricochet. These three basic tones generate a unique minimalist musical accompaniment that changes with each time the game is played, and although the sounds may not have been intended as musical, they can be interpreted as such. (Donnelly, Gibbons, & Learner, 2014). In 1975, Pong became one of the hottest selling items of the holiday season, with the release of a home version of the

game which was to be hooked up to a television. The success of Pong caused other companies to create clones of the original game and eventually develop games of their own.

In 1978, Midway created an arcade game called *Space Invaders*. The game featured one of the first continuous non-diegetic scores; a descending four-note pattern that was looped. Not only that, but the music increased in speed as the game progressed. The increase in tempo of the music added an extra element of tension to the game play. This feature became a major selling point when it was later released for the Atari console, the Atari VCS 2600.

In 1977, one year prior to the arcade release of *Space Invaders*, Atari released a home console called the Atari VCS (video computer system) which would later be known as the Atari 2600. It was on this system that hit games like *Pac-Man* and *Space Invaders* would be brought into the home. The console was able to play many different games by using game cartridges, a piece of hardware that interfaced with the console, that had a game programmed into it. This new system allowed customers to change games just by swapping out a cartridge, rather than a console. The VCS processed audio through a chip the 'Stella' chip – which also processed the graphics. The chip controlled audio through three registers:

1. A four bit (sixteen option) register which contained sounds ranging from musical tones to explosions.
2. A five-bit register that divided a base frequency of 30kHz between one and 32, to derive a pitch.

3. And another four-bit register that controlled the volume of each sound.

Dividing a base frequency to achieve pitch was at times, problematic. Since perception of pitch is logarithmic, when using division, two adjacent notes in one region might not be heard as a semitone, while adjacent pitches in different register may be perceived as such. Preferences were given to pitches closest to the twelve-tone equal temperament tuning. These programming constraints made programming a western style, twelve-tone equal temperament melody extremely difficult, and made programming twelve-tone equal temperament polyphony nearly impossible.

Over the next few years, technology rose to accommodate the demands of more complex audio programming. By 1980, many game manufacturers began to use programmable sound generators or PSGs. These dedicated audio processing chips were far more powerful than the previous generation of audio production. The early PSGs typically featured three channels, which derived audio from dividing a base frequency, much like the earlier technology, but with 4096 divisors rather than 32. This made many more pitches available and allowed for better, but not perfect, compatibility with the twelve-tone equal temperament system. These PSGs also typically featured a noise channel which could be used to generate audio effects like explosions. In 1980, the first real competition to the Atari 2600, was released. It featured one of the PSG chips, and was known as the Mattel *Intellivision*.

Video game companies were not the only ones making innovations in digital audio technology. In 1982, a personal computer called the Commodore 64 was released. It had astounding audio capabilities for the era, and was more accessible than the other personal

computers of the time due to the affordability of the unit. The Commodore 64 featured a highly-sophisticated sound chip called the 'SID' 6581, designed by Bob Yannes, who later became a co-founder of the synthesizer company, Ensoniq. Bob noted in an interview, that at the time the other sound chips that were available, were designed by engineers rather than musicians, which led to difficulties when trying to program music. Bob's SID was designed to be a synthesizer chip which is one of the reasons why it was one of the best chips of its time. (Weske, 2015). The SID 6581 had a number of features that set it apart from the competition, including:

1. Three separate oscillators that could create saw, triangle pulse wave with variable duty cycle, and noise, independently of one another, allowing for harmonies or even complex tones using additive synthesis.
2. A multi-mode filter system to produce even more timbres, through which, subtractive synthesis was possible.
3. An ADSR envelope for each oscillator.
4. A Ring Modulator for each oscillator for added synthesis options
5. An external audio output, allowing the user to chain many SID 6581 chips together for even more polyphony or added complexity in synthesis.

The amazing capabilities of the Commodore 64 and the ubiquity of the unit, made it a perfect platform for gaming, and many game manufacturers took advantage of the Commodore 64. (Weske, 2015).

In 1984 the video game industry was in trouble, sales were down and people were losing interest in video games both at home and at the arcade. In 1985, Nintendo released the

North American version of the Famicom which had already been released in Japan. In North America, this system is known as the Nintendo Entertainment System. In addition to the graphics chip found in the Atari 400 and 800, the NES featured analog circuitry for two square wave generators, a triangle wave generator, a white noise generator for creating effects, and a special digital channel which allowed for the playback of digitized sound. (Weske, 2015). Although the digital channel was an important innovation what really made the NES stand out was from its competitors in the world of audio was the music that was composed for the games. To this day, Nintendo themes like the theme *for Super Mario Brothers* are widely known. (McDonald, 2005). Another step in the direction of interactive audio can be noted in games like *Super Mario Brothers* or *Kirby*, in which when the characters have a certain state like invincibility, the song changes to suit the situation. Likewise, when the state of the character changes, like in the event of a loss of invincibility, or death, the song changes to reflect the situation. However, as Karen Collins notes in the following statement, these changes in audio were not the type of interactivity that we expect today.

> At this stage in the development of game audio, the immersive power of audio was not yet understood, and music and sound effects were less responsive to action than they are today. Transitions between segments of music, for instance, would often cut abruptly without any regard for smoothness. (Collins, 2008).

MIDI (Musical Instrument Digital Interface) is one of the most important digital audio innovations to date. It is not an audible instrument, but a protocol to facilitate communication between different devices in the form of messages. (30 Years of MIDI:

A brief history | Music Radar, 2012). MIDI was developed in 1981 with input from some of the major synthesizer companies including Roland, Yamaha Korg and Kawai. In 1983 at the winter NAMM (National Association of Music Merchants) show a MIDI connection was demonstrated between a Prophet 600 synthesizer, and a Roland JP6 synthesizer. Up to this point in time, it was nearly unheard of for different audio companies to share a common format for anything. Following this demonstration, MIDI quickly became a standard for communication between many digital audio devices, and even found its way into the world of computers and video games. MIDI files were easy to create and store, much easier than the hard coding that had to take place when audio was needed in a video game or an application. Once the files were created, it was up to the hardware to decode the file and generate the sound. There were some difficulties with standardizing hardware, and not all computer MIDI cards sounded the same, but it was an important step in the right direction. With this easier format for storing audio, and using separate files for different voices and song sections, new possibilities began to open up as far as interactivity was concerned.

Before the creation of MIDI, there was not much opportunity to create interactive music. Up to that time any sort of interactive music was very costly as far as system processing power and storage were concerned. But with the invention of the MIDI file, much of the expensive storage for audio was replaced by a set of inexpensive instructions for the generating of music. The next innovation to usher a new age of interactive music was a sound engine called IMUSE (Interactive Music Streaming Engine) created by Michael Land and Peter McConnell of Lucas Arts and was created to allow the composer to write more dynamic music. (Collins, 2008). This engine, like other early interactive audio

systems of the time, was capable of modifying the texture of a piece of music by fading audio tracks in and out, but was also able to organize the sequence of MIDI data to create smooth transitions between pieces of music. (Land & McConnell, 1994; Mackey, 2012). The need for this system is described in the patent which was won in 1994 by Land and McConnell when they wrote the following:

> For example, suppose that there is a high-energy fight scene occurring in the game which, at any time, may end in either victory or defeat. In existing systems, there would likely be three music sequences: fight music (looped), victory music, and defeat music. When, the fight ends, the fight music would be stopped, and either victory or defeat music would be started. The switch from the fight music to the victory or defeat music occurs without taking into account what is happening in the fight music leading up to the moment of transition. Any musical momentum and flow which had been established is lost, and the switch sounds abrupt and unnatural. (Land & McConnell, 1994).

The IMUSE engine's, ability to adapt the sequence of MIDI data to suit an interactive situation, was able to ease the transition between two pieces of music and help music support the interactive experience. This capability to rearrange audio to enhance interactivity in music is not only technologically tricky, but also an exercise in composition. The composer must consider how each piece of music relates to the music that precedes and follows it and cannot treat it like linear music, meaning that in an interactive score, it is nearly impossible to synch up powerful musical moments with important momentary events like it is done in cinema. (Hoffert, 2007). The IMUSE audio engine was developed in 1991 but was discontinued in 1994 due to the rising demand of realistic sounds, which MIDI was not capable of at the time. (Mackey, 2012). Since the end of IMUSE, better MIDI samples and patches have been made available and therefore, the demand for interactivity can be filled by realistic musical sounds.

Interactivity through horizontal resequencing can now be achieved through complex and feature rich game engines like the Unreal Engine, and middleware such as Wwise, Fmod. Truly, these modern audio processors owe a great debt to IMUSE for the interactive ideas and techniques that they implement into their software. As of the creation of IMUSE, sequencing music in real time was possible, which allowed for people to make truly interactive digital music.

Adaptive Music

Most modern forms of interactive media, like video games, feature a dynamic sound track that seems to change in order to suit any given situation. Interactive music is much more than the sound of a gun when your character on screen pulls a trigger, or the sound of boots clicking on a hard surface as your character walks through a building. It is the soundtrack of your actions. Imagine high dissonant pulsing strings while alone in a dilapidated mansion, or low synthesizers when exploring a derelict spacecraft. The important thing to remember when dealing with interactive audio is that it has its roots in cinema music. (Donnelly et al., 2014). There is no question that watching a movie without music is quite a different experience than watching a movie with music, so it would not be unreasonable to conclude that the experience is similar with a piece of interactive media like a video game. Composing music that helps preserve the emotional relevance to a situation, which may change at any given time, is difficult. Fortunately, there are techniques that composers can use which help in writing a truly adaptive score.

Horizontal Resequencing

For my project, I made some adaptive music, and wrote a computer program for its implementation. The technique that I have chosen to use in my project is called horizontal resequencing. (Sweet, 2015). In his book, *Writing Interactive Music for Video Games: A Composer's Guide,* Michael Sweet describes the technique in this way.

> Horizontal resequencing is a method of interactive composition where the music branches from one section to another once it reaches the end of a phrase. For example, when the music is playing underneath the gameplay, it may reach a decision point where it could go to a new section of music or it could repeat the previous section; the decision depends on the player's actions. The word "horizontal" is used to describe this technique because the game is frequently using time to determine the change in the music and time is usually mapped to a horizontal axis. (Sweet, 2015, p. 62).

Just like any method for interactivity in music, horizontal resequencing has advantages and disadvantages. As an advantage, this technique can offer clean musical transitions between larger musical ideas, which follow the situation. Unfortunately, the weakness of horizontal resequencing is that changes in music are not instantaneous; the transition needs to wait until the current musical phrase is finished.

There is an important distinction to be made here. There is another technique called vertical remixing or layering which is very different from horizontal resequencing. In vertical remixing, the music is made up of many layers. To change the musical setting, different layers are added to, or taken away from the mix. This changes the texture of the piece, which can successfully communicate intensity or mood, however, it is difficult to change the style or key of the music. Vertical remixing is used extensively in the

BioWare 2014 release *Dragon Age: Inquisition*. (BioWare, 2014). Another excellent example of successful vertical remixing can be heard in the Bazaar of *The Legend of Zelda: The Skyward Sword*. (Nintendo, 2011). In contrast, the horizontal resequencing technique is more easily imagined as a linear system, and has been around in some form since the 1980s.

An easily recognizable and possibly the most famous predecessors of this technique is heard upon dying in *Super Mario Bros.* (Nintendo, 1985). As was experienced by many of us in the latter half of the 80s and early 90s, the level music would be happily playing as we were advancing through the level, and suddenly, tragedy! We would hear the final cadence as Mario fell off the screen. Horizontal Resequencing begins with a piece of music with points across its timeline. Each of these points indicates where a change is allowed. If the user of this interactive system makes a change, and the music must adapt to a new setting, a new music file will be played to reflect the new setting at one of the points of change. An excellent example of this can be found in Sega's 2011 release, *Sonic Generations.* In the level selection area of the game, the music changes as your character stands in front of different areas. Usually, the change is made upon a percussive feature or another sound that is common between the pieces being switched. There are times when the switch between musical sceneries is very smooth and makes musical sense, however, not every change facilitates melodic or harmonic continuity. In games like *The Witcher 3: The Wild Hunt*, the music, while still important, is less of a focal point in the game, and the disjunctive elements created by the horizontal resequencing process goes less noticed. (CD Project Red, 2015). To soften the discontinuity of the musical changes in the *Witcher 3: The Wild Hunt*, the horizontal

resequencing is supplemented by fading one audio track out, while at the same time, fading the new audio track in, however the abruptness of the change is still evident.

For my project, *Seamless Sonic Scenery with an Open Source Audio Engine* I wanted to have music with thematic memorable melody, without the sudden and sometimes awkward changes that are found in *Sonic Generation.* I did not want to use long fades in conjunction with horizontal resequencing like *The Witcher,* therefore, I built an audio engine that is capable of horizontal resequencing, and composed my own music that would remain melodic and thematic, and retain the ability to change musical settings by way of a short extra piece of music that serves as a transition. It is true that modern audio middleware like Wwise, Fmod, and powerful game engines like Unity3d and the Unreal Engine can sequence music in this way and they have been used in many successful projects. I have been through the Wwise documentation, and have become proficient in that program. I have also begun a few very small projects using Unity. Through these experiences I have found that, because they can do so much they can be difficult to learn how to use, despite their excellent online documentation and tutorial videos. Much of this software also has complex terms and conditions of use or hefty licensing fees. The audio engine that I have designed has only one purpose, so it is easy to use. I built it using an open source programming language, and offer it to the interactive audio community free of any fees under the creative commons license, (CC BY-SA4.0). Under this license, all are free to use, share and adapt the audio engine, upon conditions of attribution, and that any remixes or redistribution of the software are under the same license.

Chapter 2. Composition

Music composition played an indispensable part of my project. This was my first time composing music for an interactive system, while it was challenging, I am very pleased with the result. I began by composing three pieces of music which were to represent ideas or places. I then decided where my transition points in each of those pieces were, and cut the pieces of music accordingly. I then composed banks of transitions that would make a smooth musical transition from the end of each of those segments that created the musical setting, into the beginning of different musical settings. See figure 1.

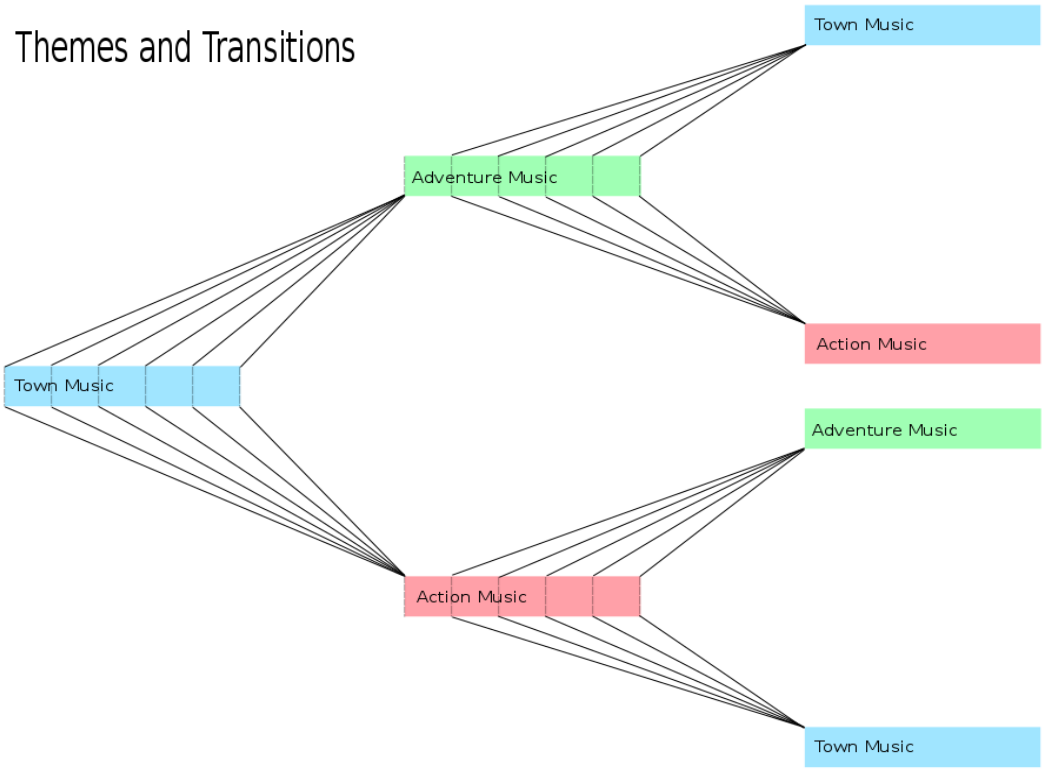


Figure 1:

The dashed lines inside the colored boxes represent the small audio files that make up the loops, and the solid lines represent musical transitions that tie the ends of the small audio files to the beginning of another theme. This figure should be read from left to right.

Three Themes

I decided that I was going to compose three musical settings, which needed to fit certain criteria. First, the pieces of music needed to be able to play in a loop. Second, they needed to have a clear motivic idea, to represent a location or emotion. Lastly, the musical statements made in the pieces needed to be concise so that the pieces could be cut into smaller pieces without disrupting the musical flow in the event of a transition. In each of my three themes, I chose an orchestral approach which featured a simple melody.

I did this in the hopes that some sort of continuity could be preserved between the pieces, so that it felt like all of the works could be part of the same thing. The scores for the themes are found in the appendix. The audio files are included with the audio engine. All the music I composed for this project is under copyright. They are intended to be used to demonstrate the capabilities of the engine, and to give an example of how to compose interactive music using the horizontal resequencing technique. Any use of the music beyond personal study, or the demonstration of the engine is prohibited, unless my written consent is obtained.

Town

The music I composed representing the Town features relatively small ensemble of flute, oboe, horn, violin 1 and 2, viola, cello, and double bass. It was written in the very common key of C major, to represent the familiarity of a hometown. I chose the odd time signature of 7/4 to push against that familiarity and suggest some small struggle, which then resolves to a more familiar 6/4 for a brief time, as if to say that in town, the small struggles are easily resolved. In this piece, the melody is usually carried by the oboe, and at times, supported with a harmony from the flute. There are times however, that the melody is carried by the horn. The horn is a very heroic sounding instrument, and it is meant to represent the protagonist or hero, while the oboe and flute are to represent the town itself. This piece is highly repetitive, which lends itself to playing in a loop quite nicely. The cadences are clean, which is to say that at the end of the cadences, there are no sounds that hold over into the next section. This makes breaking this piece into smaller sections an easier task.

Adventure

The piece titled *Adventure*, is the piece that I wrote to represent what happened after our protagonist has left the safety of the town walls. While listening to this piece, it easy to imagine a young hero, wooden sword in hand with the expanse of the whole world ahead. The ensemble for this piece is slightly larger than the one playing the town music and consists of a flute, an oboe, a clarinet, a bass clarinet, a horn, a trumpet, a trombone, timpani, a glockenspiel, cymbals, a harp, two violins, and a cello. This piece was written in binary form. Part A begins with an upbeat rhythm featuring the strings and timpani. Then the horn comes in with the main theme, which is punctuated by harmony in the trumpet and the flute. Harmonically, this first section is very stable, utilizing many fourths and fifths which firmly establishes the key of C Major. Between the rhythm and the intervals used, this first part of the piece easily communicates bravery and adventure. Part B is reflective in nature, and features a harp, and a flute, which takes the melody. It could represent a number of sentimental ideas, like perhaps, a longing for home, the beauty of the adventure, or the motivation behind the adventure. At the end of the second part, the rhythm and harmony is announced by the bass clarinet, the rest of the ensemble washes in with a cymbal roll, and the adventure music is back to the upbeat anthem that it was before. Like the town piece, this adventure music features clearly defined sections, which is important when dividing up the whole piece into smaller files.

Action

I also created a theme to represent a fight or intense conflict. This piece is very rhythmic and relies heavily on large taiko drums. It also features timpani; an aggressive strings

section including two violins, a viola, a cello, and a bass; a bright brass section consisting of a trumpet, and a trombone; and a soaring wood winds section featuring a flute, an oboe, and a clarinet. It is very firmly written in the key of d minor which contrasts the key of C major found in the other two pieces. It is mostly in a 5/4 time signature which is a great time signature for the instability of a dire conflict. The piece begins with a strong rhythm on the taiko. Soon after, the texture is thickened by strings. Once the strings have been added to the piece, melody is added by short blasts in the brass and winds sections. The texture then thins out for the middle section where the taiko quietly keeps the rhythm, but drifts between time signatures. Soon we hear the familiar 5/4 strongly return and we begin again, with added textures from the strings section. We end with the taiko quietly keeping time, with a lone trombone carrying a piece of the melody. Due to the highly rhythmic nature of this piece, it was not hard to separate into sections, which made it easy to integrate into my audio engine.

Transitions

It was necessary to compose transitions in order to have each piece of music flow easily between themes. Each piece of music was written so that it could be broken up into many musical files that, when played in the proper order, would provide the musical setting for either the town, the adventure, or the action. The ends of each of these smaller musical files required a transition that would be able to go from the current musical idea, and lead into the beginning of one of the other two themes. See Figure 2.

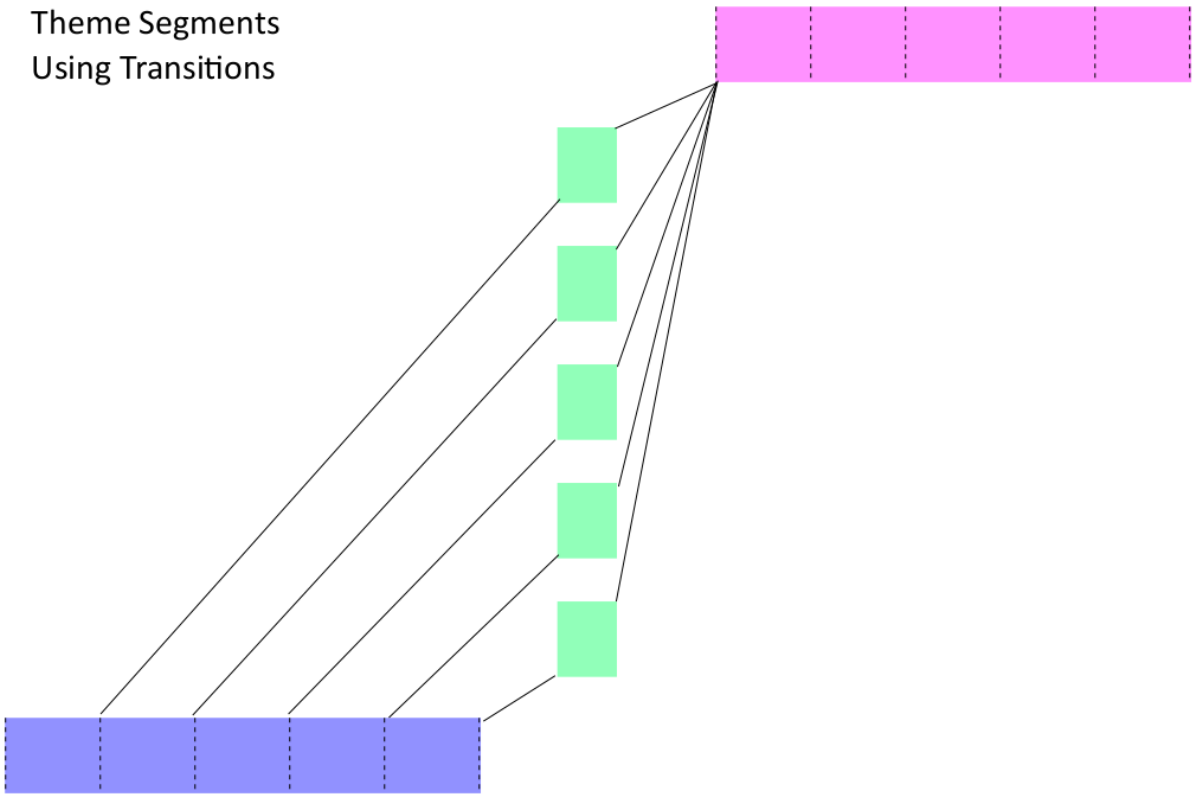


Figure 2

In this figure, we see a musical setting in blue. It is segmented by a dashed line which is to represent the smaller files that make up the theme. At any of those points within the musical setting, the system may use a short green audio transition, in order to move to the beginning of the new sonic setting in pink.

As is made evident in Figure 2, when moving from one theme to another, there needs to be a transition composed for the end of each audio file that makes up a musical setting. These transitions need to bring the music from those specific points in the theme, to the

beginning of another theme. The number of individual transitions necessary is equal to the number of files that your theme has. The group of transitions that move one theme to another is called a bank. There needs to be one bank of transitions for each change between themes. If there are only two themes to switch between, two banks are needed. If there are three themes to change between, six banks are necessary. The corresponding equation is $b = t \times (t-1)$ where t is the number of themes, and b are the number of transition banks required.

There are important things to take into consideration when composing transitions. In his book *Music for New Media Composing for Videogames, Web Sites, Presentations, and other Interactive Media,* Paul Hoffert discusses seven key things to keep in mind in order to maintain control of your musical transitions. They are as follows:

1. Volume
2. Tempo
3. Rhythm
4. Key
5. Harmony
6. Texture
7. Style

(Hoffert, 2007, p.33).

In writing my transitions between pieces, I tried to address each of these things as much as it were possible, in the short time of a transition.

Volume

Dealing with volume is a very important aspect of transition writing. If this aspect of transitions is neglected, the result can be jarring. I took special care to determine the volume of each of the audio files of my musical settings, and I composed transitions that

began at that level, and changed to the level of the new musical setting. For the most part, the town music felt the quietest, and the action music was the loudest, so when composing transitions between these two settings, I had to pay particular attention to the volume. Control of the volume is easily taken care of by automation in a digital audio workstation.

Tempo

There were differences in tempo between all my sonic settings, and I dealt with those changes in a couple of different ways. In some instances, I steadily increased the tempo over the course of the transition. The transition bank between the town music, and the adventure music is a great example of this, and that is one of the reasons why the change in theme seems to flow as easily as opening a door. In other instances, I quickly decreased the tempo, in order to create some contrast between the two sonic settings. A good example of a rapid decrease in tempo is the bank between the action music, and the town music. In this bank of transitions, I finish with a quick aggressive flourish of the strings, and strong percussive hits, and as the sounds of that finish are still resonating, the chords of the town theme swell like a slow breath and prepare the listener for the town music.

Rhythm

Each of my musical settings has a different predominant time signature. The town theme is in 7/4, the adventure theme is in 4/4, and the action is in 5/4. As such they could not have the same rhythms. All of my banks of transitions had to account for the changes in time signature and therefore rhythms. I found that many of these were easy to mask

while manipulating tempo changes. An example of this can be found in the changes that go from town music, when the horn has the melody, marked by E and F in the score (see appendix), to the adventure music. Other times, rhythms could be easily changed over a common repeating percussive element, like the change from the harp section of the adventure music, marked D, and E in the score (see appendix) to the action music, using the repeating taiko and timpani drums to help change the rhythm and meter.

Key & Harmony

Key and harmony were easy to address between the town and adventure themes. They were both in the same key and I did not need to modulate in order to switch between the two musical settings. The more difficult changes were made around the action theme. The action theme was written in the key of d minor, a key which is not very strongly related to the C major of the other two themes. The best way that I found to change into d minor was to go stepwise from the C to d in one or more voices and repeat that figure. In this way, d minor was established through stepwise motion and repetition. To return from d minor to C, I used a different approach. In most of my transitions into my adventure music, I aggressively finished off with a powerful cadence in d, and moved from that key region to the key of C through a quick scale ending on the tonic, C, thus establishing the new key and allowing a successful transition from action to adventure music. In most of the transitions from the adventure music to the town music, the aggressive cadence stays the same, but is followed by an orchestral crescendo of the b7 chord with an added G. This is an effective chord because the added G allows it to serve a dominant function, which resolves easily to C to begin the town theme.

Texture

While composing transitions, it is important to remember to take texture into account. I have three different musical settings with different textures. In all of these transitions I was able to successfully phase in new instruments over old ones to make sure that the changes in timbre were not too abrupt. While I was adding new instruments, I had to be careful that the textures did not become too thick. Critical listening helped me to hear what needed to be turned down so that the transitions did not become overwhelming when it came to texture.

Style

The musical style of all my themes were written in a simple orchestral style. I decided to compose my themes in this way to create continuity across my music. Because I did this, I did not need to use my transitions to change styles.

In closing, trying to address each of these seven considerations was challenging, however, my transitions are stronger because I did. The music of my large loops is repetitive in nature, and sometimes did not require that an entirely new musical transition for each change of sonic setting. Of course, the option is there to write unique transitions for every musical file in a musical setting if it is desired.

Chapter 3. The Engine

At least as important as the music I wrote, was the development of the interactive audio engine. I created the engine using the Pure Data programming language. It has the ability to do horizontal resequencing with a transition and is quite easy to use. It was created not only to help me test my music in this interactive setting, but also as a tool for any composer who wishes to have an easy interface for transition based horizontal resequencing. It is easy to use and applicable to many specific needs. The program itself had very small beginnings with very limited functionality. The first iteration of the program played two files back to back, with no transitions, and was computationally inefficient. Nearly twenty versions later, the program runs smoothly and is much more efficient. It has turned out to be the horizontal resequencing engine that I had envisioned.

Open Source Software

I used the Pure Data programming language to write the software. Pure Data was first developed by Miller Puckette, and is considered to be open source software. ("Pure Data - pd Community Site," 2017). This means that the source code is available and free for anyone to use or modify. ("What Is Open Source Software? | opensource.com," 2017). There are many reasons to use open source software. Some people like to use it because they have greater control over a program or design, and can alter, or customize it without legal repercussions. Others use it as an area of exploration to hone programming skills. Open source software is also used for security and stability reasons; programmers all around the world can, at any time look at the source code and patch weaknesses. (Ibid, 2017).

Pure Data Programming

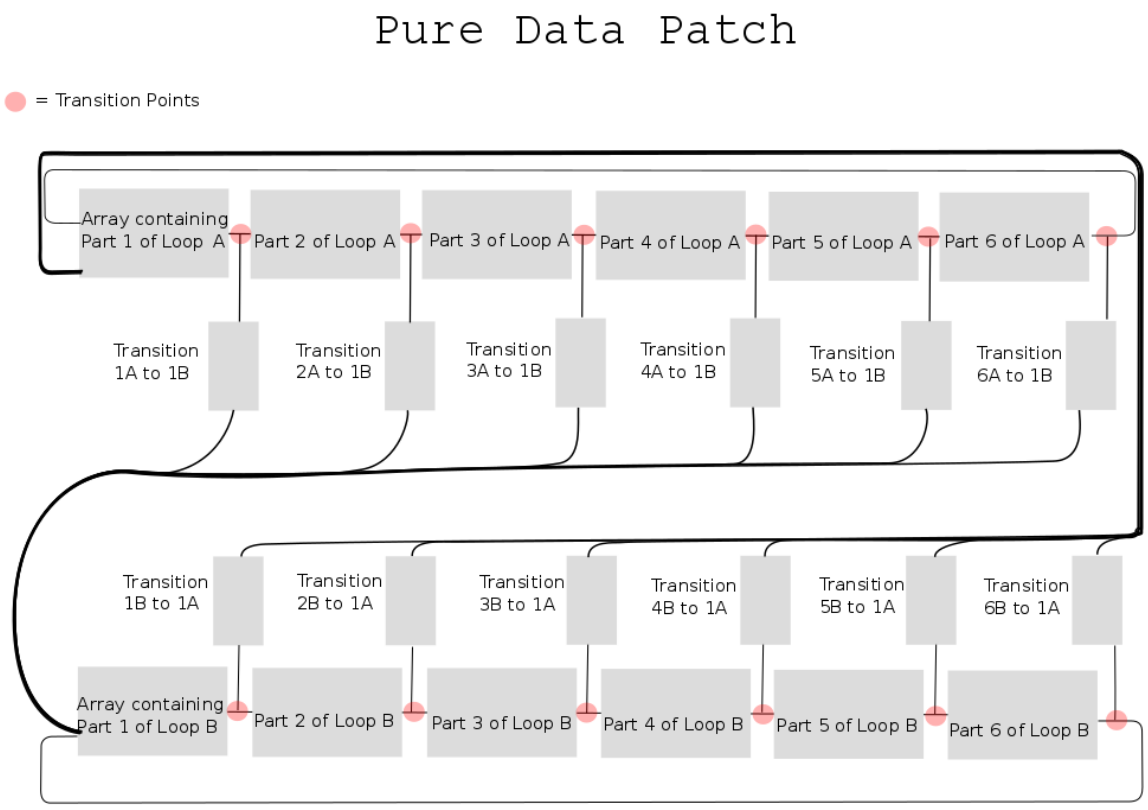
Pure Data (often abbreviated, Pd) is a powerful visual, real-time programming environment. It is used for audio, video, and graphical processing. Because it is so powerful, it has been used in many different roles, including live performance, composition, audio design, analysis, and audio and visual processing. It is able to send and receive many different types of messages, and can therefore also be used to manipulate things in the physical world.

Programming in Pure Data is intuitive. The basic unit of functionality is a box wherein the user types the name of a Pd object. Objects can interact with other objects, simply by tying them together in the environment, much like one would plug a guitar into an amplifier using a patch cord. Complex programs can be made by connecting numerous objects together in this way. This is in contrast to traditional or code based programing languages, where many lines of compiled code create your program, and altering the program may mean changing huge sections of code, as opposed to simply moving a patch cord from one object into another. (Princic, Hide, & Holzer, 2017).

The Engine Design

The audio engine needed to be able to play audio files in a sequence that could be disrupted if a change was required by the user. Upon that change, the appropriate transition needed to be selected and played which would be followed by a new sequence of files, representing a new musical scene. See Figure 3.

Figure 3.



The logic represented by the pseudo code figure 3 is as follows. There are two banks or arrays of files which make up loops A and B respectively. We will begin with part 1 of loop A. At the end of this segment, there is a transition point. If no condition is met, the next musical file that will be played is part 2 of loop A. If by the end of part 2 in loop A, the user has chosen to move to loop B, the next file played will not be part 3 of loop A, but rather the transition that goes from 2A to 1B. After that transition is played, the engine will begin playing through the bank that makes up loop B, until such time that the user chooses to transition to another loop. If there is no further user interaction, the engine will continue playing through the bank of files as a loop. In figure 3, there are only two possible loops to choose from. In my engine, I chose to program three loops, with all the possible transitions between them.

Pieces of the Engine

The Pure Data programming environment begins as a blank canvas. If we want our program to do something, we must put something on this canvas. So, we put a box on the screen, give it a name, and if the name is recognized as a Pd object, the box will gain inlets, outlets or both, depending on which object it is. We build our program by patching outlets of objects to inlets of other objects. When we are satisfied with what we have done, creating objects, and connecting them to one another to serve our desired function, we save our work and the resulting file becomes what is known as a patch. (Farnell, 2010, p.152).

In Pure Data, it is possible to put a patch within another patch. These sub patches are very useful in keeping complex programs more organized. My engine has many different parts in it, most of which are separated into many sub patches. This modular design is useful for debugging, organization, extending or modifying the program, and reusability in other similar projects. The rest of this chapter will discuss the major parts of my audio engine. Pictures of the different parts of the engine are included in the Appendix, rather than in the body of the text. The files are available under the creative commons license, (CC BY-SA4.0). In the following explanation of my engine, I will not describe the function of every single object used, but instead will address some of the more key pieces of the engine. A basic understanding of Pure Data will be useful but not mandatory at this point. I will use [ ] when discussing objects, and { for messages.

Main Patch

This is the patch that we see upon opening the file. There are no vital functions performed on this page. Nearly all the objects in this patch are sub patches which perform different functions. On the top of the page, there is an area selection which allows the user to choose between a town, adventure or action setting. The buttons are activated by clicking them. This patch could very easily be modified to be used as the music engine for a video game. All that would need to be done, is an object would need to be placed to monitor a port for a non-zero message from the game engine using OSC, MIDI or another protocol. Once received, the message could be used to trigger any one of the three options which changes the musical setting. Below the selection section, we see two boxes. These will display the left and right channels of which ever sound file is currently loaded in the engine. The fader to the right is an extra volume control. At the bottom, there is a toggle box which turns the engine on or off, and a manual start which begins the music.

File Loader

The portion of the patch that loads the file and stores it in the system memory is called [pd ReadStore0] and can be found just to the left of the wave display. One can access this sub patch by simply clicking the [pd ReadStore0] box. Once open, we see several boxes and wires. The object [r file0] sends information from another sub patch which contains a file path to an audio file. The [read] and [soundfiler] objects read the file and then loads that information into the left and right arrays which makes them ready to play. The rest

of the patch is devoted to giving other patches necessary information, including; when to fade the audio in, what the sample length of the current audio file is, and how fast to play it.

File Player

The part of the patch that plays the files can be found on the main page of the patch and is called [pd Play0]. This sub patch is responsible for playing the files loaded into the left and right arrays. Click [pd Play0] to see what is inside. At the top of the patch, I have an object called [r start0], which is connected to [phasor~]. In Pd, a phasor is a sawtooth wave generator which begins at 0 and goes to 1. The [r start0] tells the [phasor] to begin at 0. The [r speed], which was sent from the [pdReadStore0] sub patch, tells the phasor how long it takes to go from 0 to 1. That number is multiplied by [r SampleLength0] which contains the number of samples in each file. The resulting saw tooth wave is then sent to [tabread4~ Sample0L] and [tabread4~ Sample0R]. As the saw tooth sweeps through the [tabread4~] objects, each sample in those arrays is read. If the speed of the [phasor~] object is correct, the result will be that the audio files loaded in the arrays play with no speed or pitch distortion. To the left is the part of the patch which is responsible for triggering the loading of new samples into the arrays, as well as the message to update the speed of the phasor, and when to fade out. These messages are sent as soon as a threshold on the phasor is met. The section on the right of the patch is responsible for providing the moving line across the graphical representations of the sample on the main patch as it is being played.

Sample Index

The sub patch called [pd sample index] is located right under [pd ReadStore0] just to the left of the graphical representations of the samples. When this sub patch is opened, we can see two columns of more sub patches. On the left we are the sub patches containing each of the musical settings, Town, Adventure and Action. On the right, are the transition banks. If the one of the musical settings sub patches were to be opened, we would see a large patch which can be easily divided into pieces of logic. The top section which ends in a [spigot] object directly above a [route] object, is a counter and a switch. The counter is activated at the end of whichever sample is currently playing. The switch is a piece of a larger switch found in the [pd transition] and it tells the information produced by the counter to go into either the bank of audio file making up the loop, or the banks of files which contain transitions. After the [spigot] object, the number generated by the counter is passed into a [route] object. This object will send the signal out through one of the many different outlets, depending on what number it receives. This signal then passes through a {symbol} message, which contains the file path of the next audio file to be loaded into the engine. This message is then sent through [s file0] which goes back to the [pd ReadStore0] patch which was previously discussed.

Selector

The next major piece of the engine is the switch. On the main page of the patch, this can be accessed by clicking on the [pd transition] sub patch. This was perhaps the most difficult piece of the program for me to write. It is responsible for switching between

themes and choosing the appropriate transition bank based on the selection made by the user. Along the bottom of [pd transition], there is a piece of code that is detached. This is what [pd transition] grew out of, and I kept it there as a debugging tool. This same code can be found in [pd switch], which is just above the detached section at the bottom. [pd transition] is broken into three parts, one for each musical setting. When a setting is selected, it activates a toggle which sends messages to the transition banks which transition to that setting. For example, if the user selects the adventure music on the main page, that information triggers messages which are sent to the banks that go from town to adventure, and action to adventure. At this point, the pd patch also receives information from the currently playing patch (see section heading "Sample Index"), and with that information, it makes the appropriate selection for which bank from which to choose the transitions. At the end of playing the appropriate transition piece, a message is sent to begin playing the musical setting that was selected by the user.

Chapter 4. Using the Engine

This engine is a useful tool for anyone who would like to explore interactive music through horizontal resequencing. It is not difficult to use, and the logic of the engine lends itself to expansion and modification. As stated before, anyone is able to use or modify this engine as they choose as long as my work is cited.

Drivers and Options

No specialized audio drivers are required to run this patch. Of course, better hardware and drivers have a tendency to create a better audio experience, however my engine works just fine on default audio drivers and built in hardware. If clicks are heard while the engine is switching samples, the setting to alter is in the media pull down menu under audio settings. At the top left of the audio settings window is a section labeled delay (msec). That click will go away by increasing the number in that field to 120, or 250 for an older system.

File Paths

The file paths in this patch are important because they tell the Pure Data patch where to look for the audio files. The default paths in the engine are relative to the folder that the patch is in, so if the patch and sample music is downloaded and everything is kept in the original folders, the patch should be able to locate the audio files it needs. Altering the file paths to use other audio files is fairly straightforward. In the sample index, the file paths can be changed one at a time by unlocking the patch, clicking the file paths, and changing them. Faster changes can be made by opening up the Pure Data patch, in

notepad or similar ascii text editor, and finding and replacing the file paths right in the code. Make sure that the file paths are correct, or the engine will not work, and an error message will appear in the console window of Pure Data.

Customization and Expansion

The modular design of this audio engine lends itself to easy alteration. It should be said that care must be taken while programming audio so that damage to your equipment, or more importantly, damage to your ears can be avoided.

Adding to the Sample Index

The first and easiest way to expand this engine is to change the file paths in the sample index. Doing this will point the engine to your own personal music that you wish to use in the engine. Instructions for this are under the heading *File Paths* above. The number of files per loop can also be changed by altering the [route] object in the appropriate sample index. As an example, suppose that a new piece of music had been written with only 6 files being played before the end of a loop, and the file paths had been changed accordingly. The next step would be to change the [route] object so that it would reflect the number of files needed in the loop. When changing the [route] object, it is important to consider which outlets are connected. If the first outlet is unconnected in the original program, then ensure that it is unconnected when altered. The other thing to make note of is that on the last outlet there is a [trigger] object which sends an object to a [s $clear]. This must be connected to the last outlet of [route] or the patch will not loop.

This program is expandable beyond three loops of music and six banks of transitions. Although the logic of the program is fairly easy to trace, I would not recommend expanding the engine past its current three loops and corresponding transitions without familiarity with programming in the Pure Data environment. The areas that will need to be altered are the indexes for the loop and transitions, and the switch, and this can be done by duplicating and renaming the pieces of the engine that you want to expand. The sub patch [pd transition] will need to be expanded as will the switch logic in [pd switch]. Care will need to be given to making sure that the send objects ([s $]) and return objects ([r $]) correspond between the [pd transition] sub patch and the sample indices containing both the loops and the transitions. There will also need to be added sends from [pd transition] to the pre-existing loops and transitions to make sure that they go from loops to transitions at the right time. See existing logic as an example.

Chapter 5. Conclusion

The research that I have completed and the project that I have developed have taught me many lessons about the importance of audio in selling an experience. I have also learned more about programming and perseverance. Over the course of my graduate program, I have composed three main themes, and written six banks of transitions for seamless music transitions between the themes. I have implemented that music on an interactive music engine that I created using the Pure Data programming environment. Under the creative commons license, I have made that engine available to everyone to use, remix, and redistribute under the same license, with appropriate attribution. Composers may wish to use the engine as a tool to help them develop interactive music. Artists might find this software useful in adding adaptive music to their exhibits. It can also be used as a main music engine to implement adaptive music to video games or other pieces of interactive media. It can also be useful as an educational tool when discussing audio programming. I am pleased with these outcomes and feel that I have made a meaningful contribution to the creative community. I am pleased with what I have been able to accomplish; I also know that there are possibilities for future research which could include this project as a foundation.

In the future, I would like to experiment with an optional layering of currently playing samples so that I could use horizontal resequencing with the added dimension of vertical remixing. I would also like to find ways to further simplify and expand the engine, and make it even more user friendly. I would also, of course, love to implement this system in a game or another interactive environment such as a sonic art instillation. Future versions of the engine could feature an enhancement like 5.1 audio capability, or objects

ready to receive OSC, MIDI or other messages so a variety of controllers and programs may speak to the audio engine.

I hope to one day hear that someone has used this tool in education or another creative endeavor. I would also like to see someone creatively use the engine and improve upon its design. This has been informative and exciting research for me, and my hope is that all who experiment with this engine enjoy it as much as I have.

Bibliography

30 Years of MIDI: A brief history | Music Radar. (2012). Retrieved April 18, 2016, from http://www.musicradar.com/news/tech/30-years-of-midi-a-brief-history-568009

BioWare. (2014). *Dragon Age: Inquisition*. EA Games. Retrieved from https://www.dragonage.com/

Brame, J. (2011). *Thematic Unity Across a Video Game Series . Act – Zeitschrift Für Musik Performance*, *2*. Retrieved from http://opus.ub.uni-bayreuth.de/volltexte/2011/900

Brown, M. (2014). *Adaptive Soundtracks: A Game Maker's Toolkit*. Retrieved January 5, 2017, from https://www.youtube.com/watch?v=b0gvM4q2hdI

CD Project Red. (2015). *The Witcher 3*. CD Project Red. Retrieved from http://thewitcher.com/en/witcher3

Collins, K. (2008). *Game sound: An introduction to the history, theory, and practice of video game music and sound design*. Cambridge, Mass.: MIT Press.

Donnelly, K. J., Gibbons, W., & Learner, N. (2014). *Music in video games: Studying play*. (K. J. Donnely, Ed.). New York, NY.: Routledge.

Farnell, A. (2010). *Designing Sound*. Cambridge, Mass.: MIT Press.

Frictional Games. (2010). *Amnesia: The Dark Descent*. Frictional Games. Retrieved from https://www.amnesiagame.com

Hoffert, P. (2007). *Music for new media: Composing for video games, websits, presntations, and other interactive media*. Boston, MA: Berklee Press.

Land, M. Z., & McConnell, P. N. (1994). *Method and apparatus for dynamically composing music and sound effects using a computer entertainment system*. *US Patent 5,315,057*, (November 25, 1991), 1–30. http://doi.org/US5315057

Mackey, B. (2012). *IMuse and the Secret of Organic Music from 1up.com*. Retrieved March 16, 2016, from http://www.1up.com/features/imuse-secret-organic-music

McDonald, G. (2005). *A History of Video Game Music*. Retrieved December 10, 2016, from http://www.gamespot.com/articles/a-history-of-video-game-music/1100-6092391/

Nintendo. (1985). *Super Mario Bros.* Nintendo. Retrieved from http://mario.nintendo.com/

Nintendo. (2011). *The Legend of Zelda: Skyward Sword.* Nintendo. Retrieved from www.zelda.com/skywardsword/

Princic, L., Hide, A., & Holzer, D. (2017). */Chapter: Introduction2/ PURE DATA.* Retrieved January 1, 2017, from http://write.flossmanuals.net/pure-data

Pure Data - pd Community Site. (2017). Retrieved from http://puredata.info/

Sweet, M. (2015). *Writing interactive music for video games: A composer's guide*. Addison Wesley.

Usher, R. (2015). *How Does In-Game Audio Affect Players?* Retrieved March 16, 2016, from http://www.gamasutra.com/view/feature/168731/how_does_ingame_audio_affect_.php?page=1

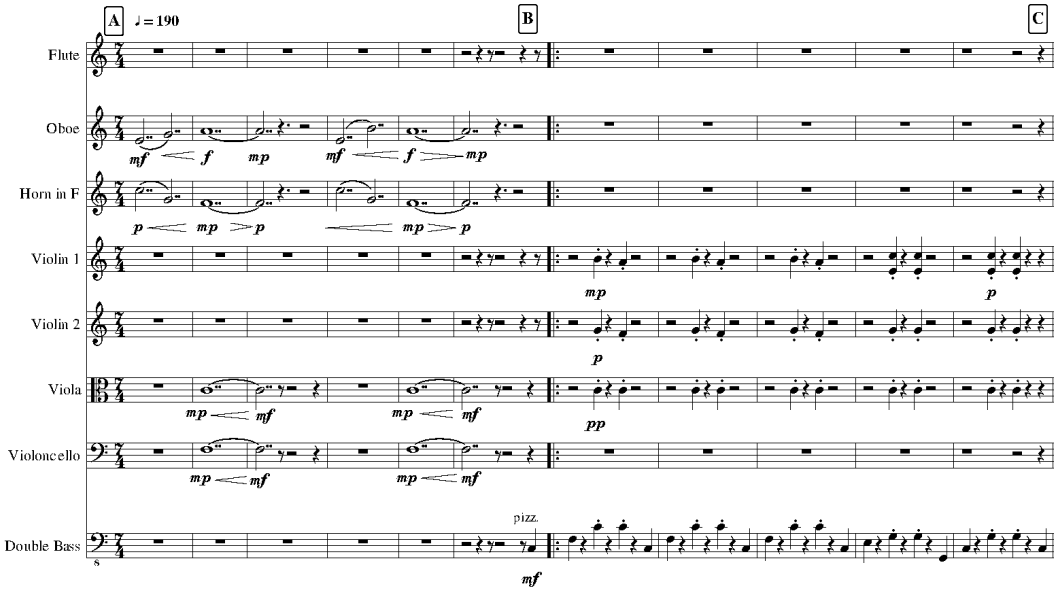
Weske, J. (2015). *History - Digital sound in music and computer games*. Retrieved December 5, 2015, from http://3daudio.info/gamesound/history.html

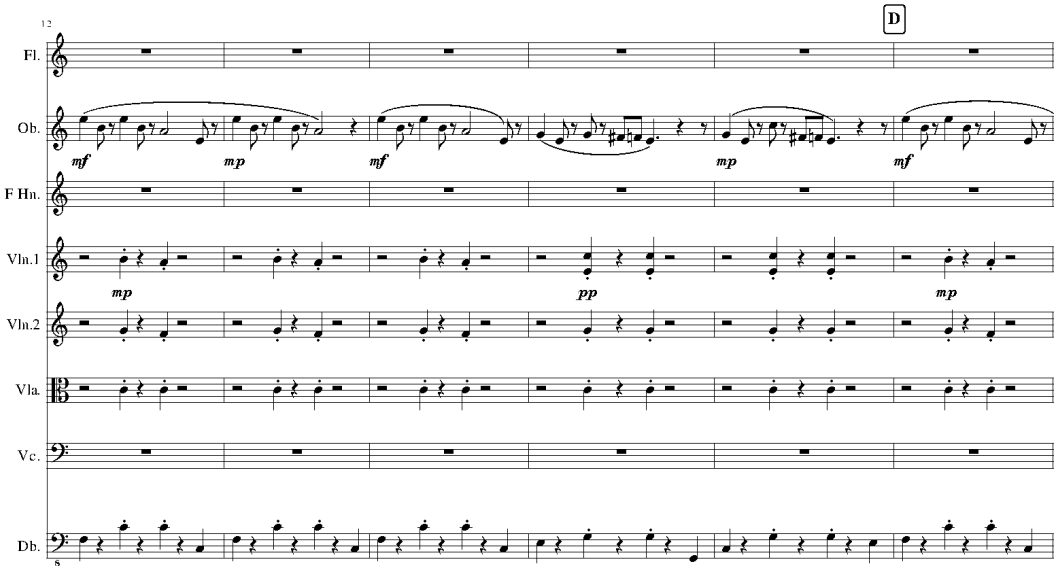
What Is Open Source Software? | opensource.com. (2017). Retrieved January 5, 2017, from https://opensource.com/resources/what-open-source

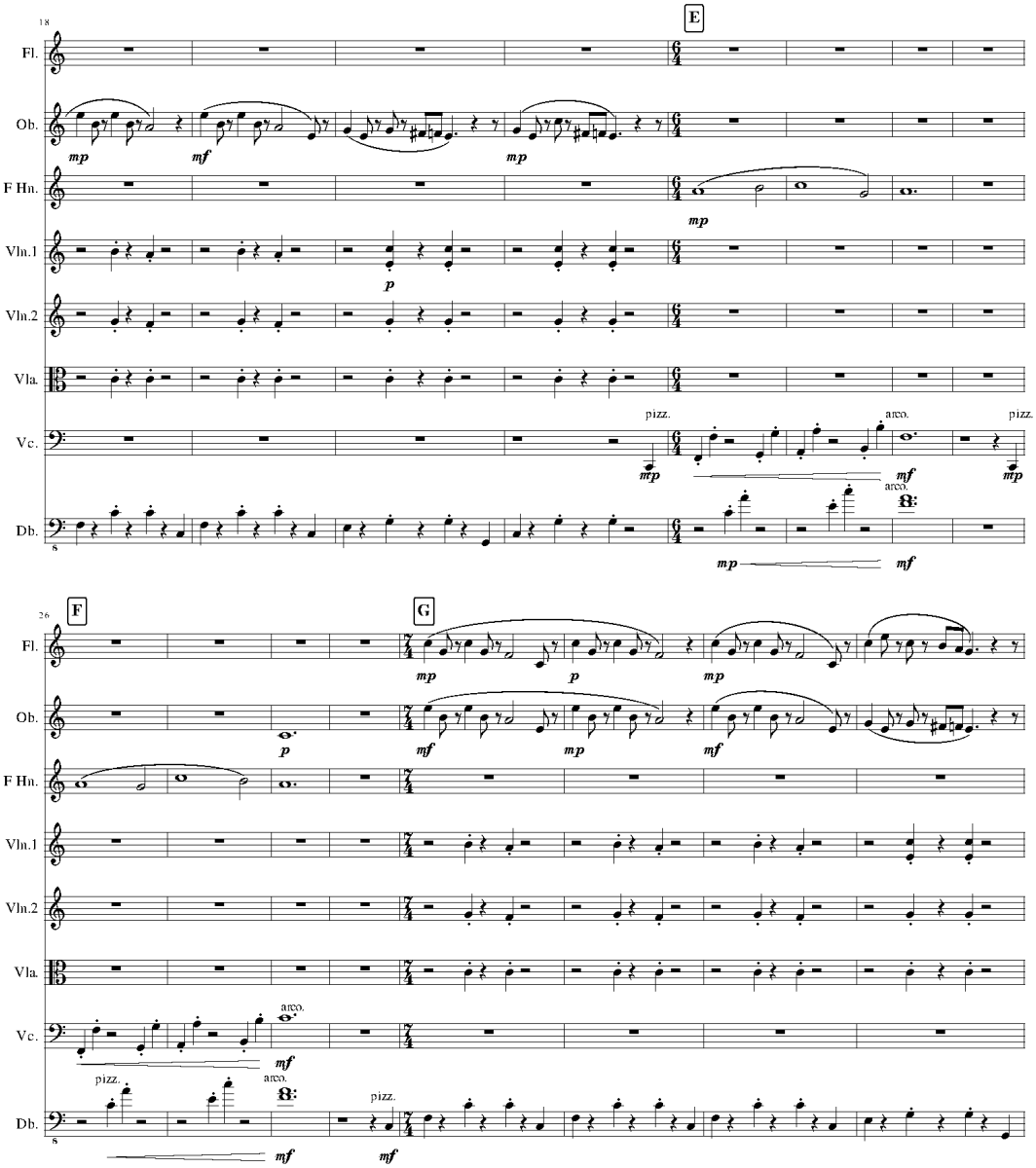
Appendix

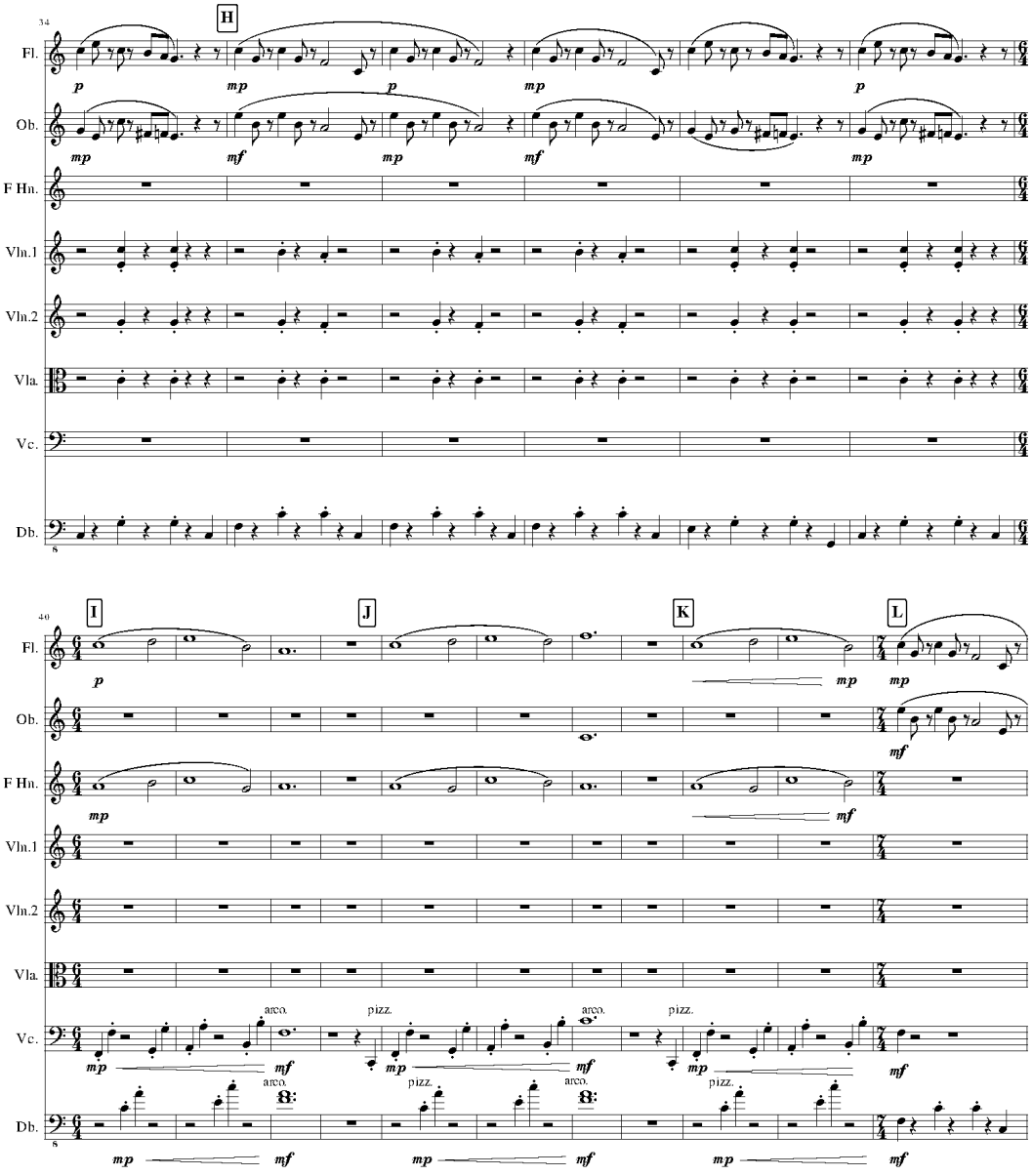
Town

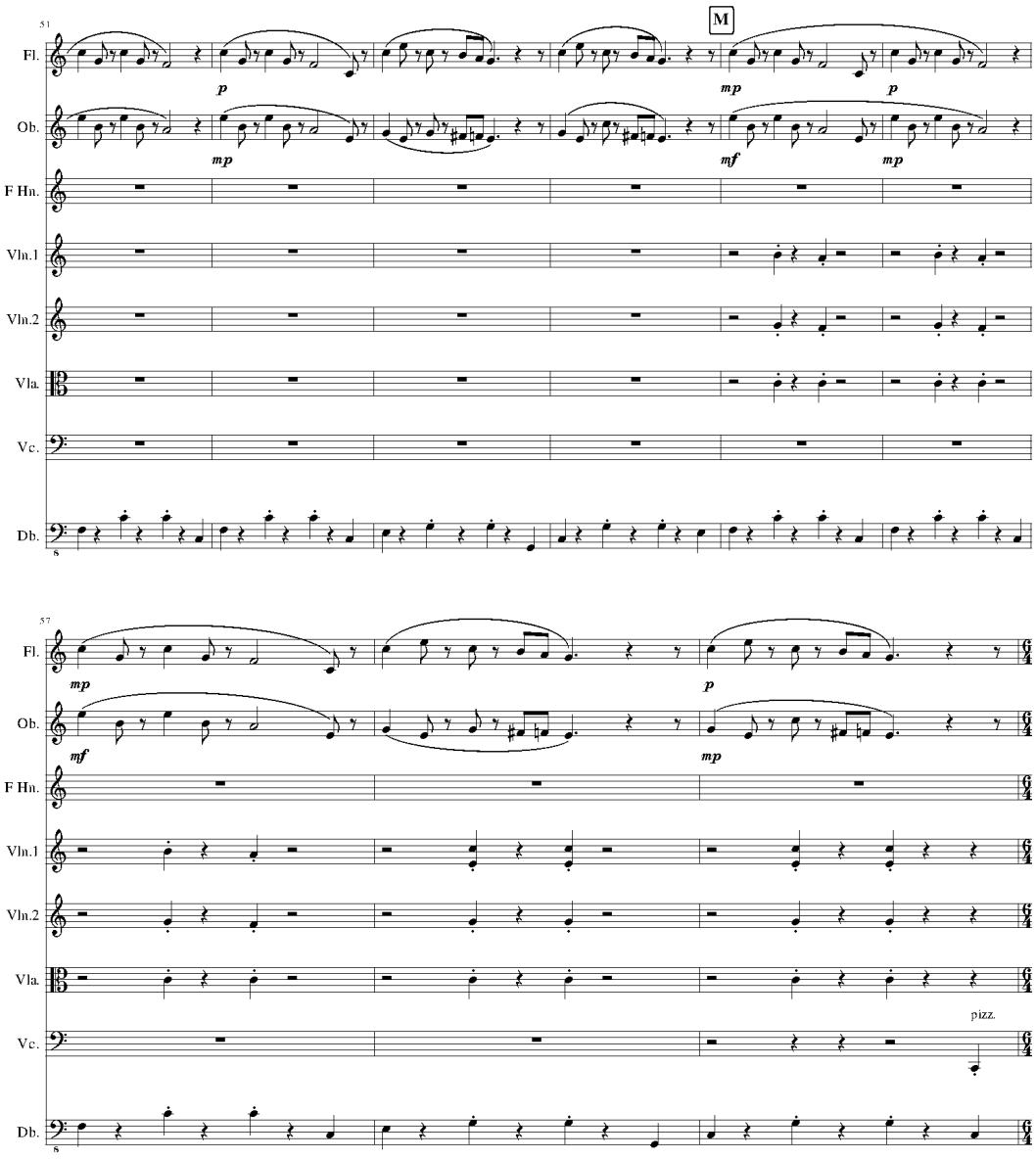
Jacob Hills

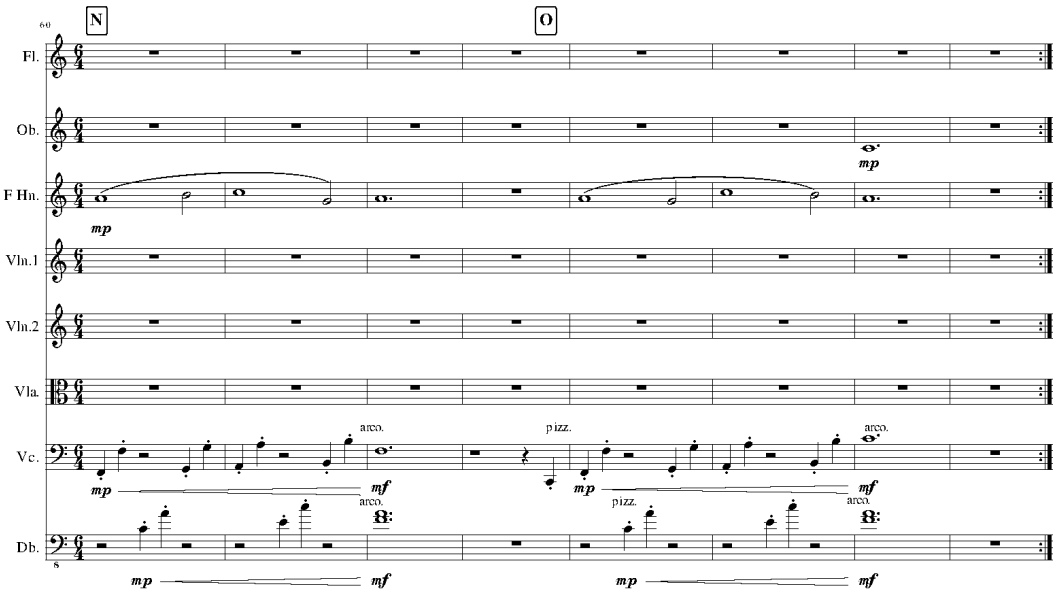






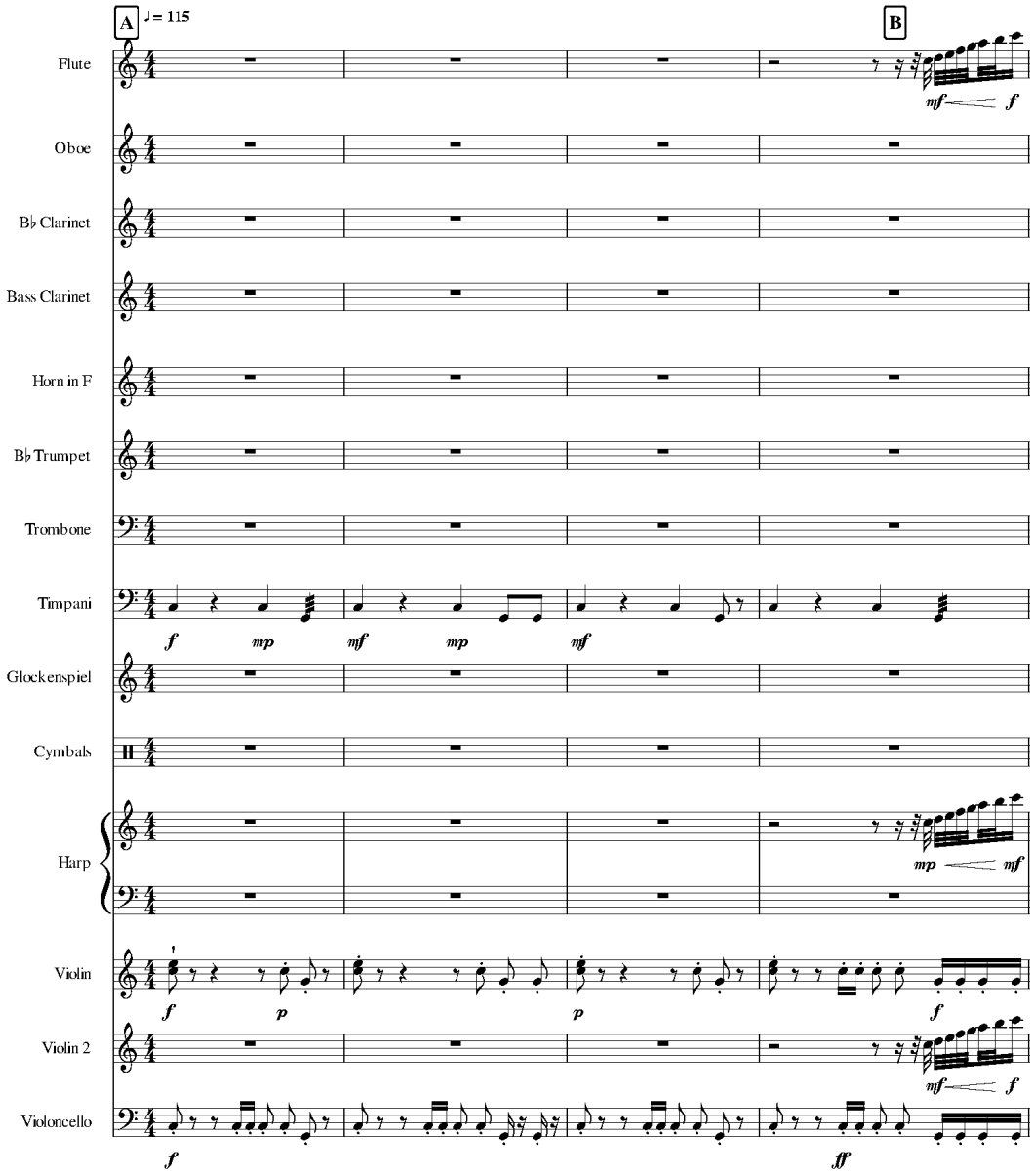


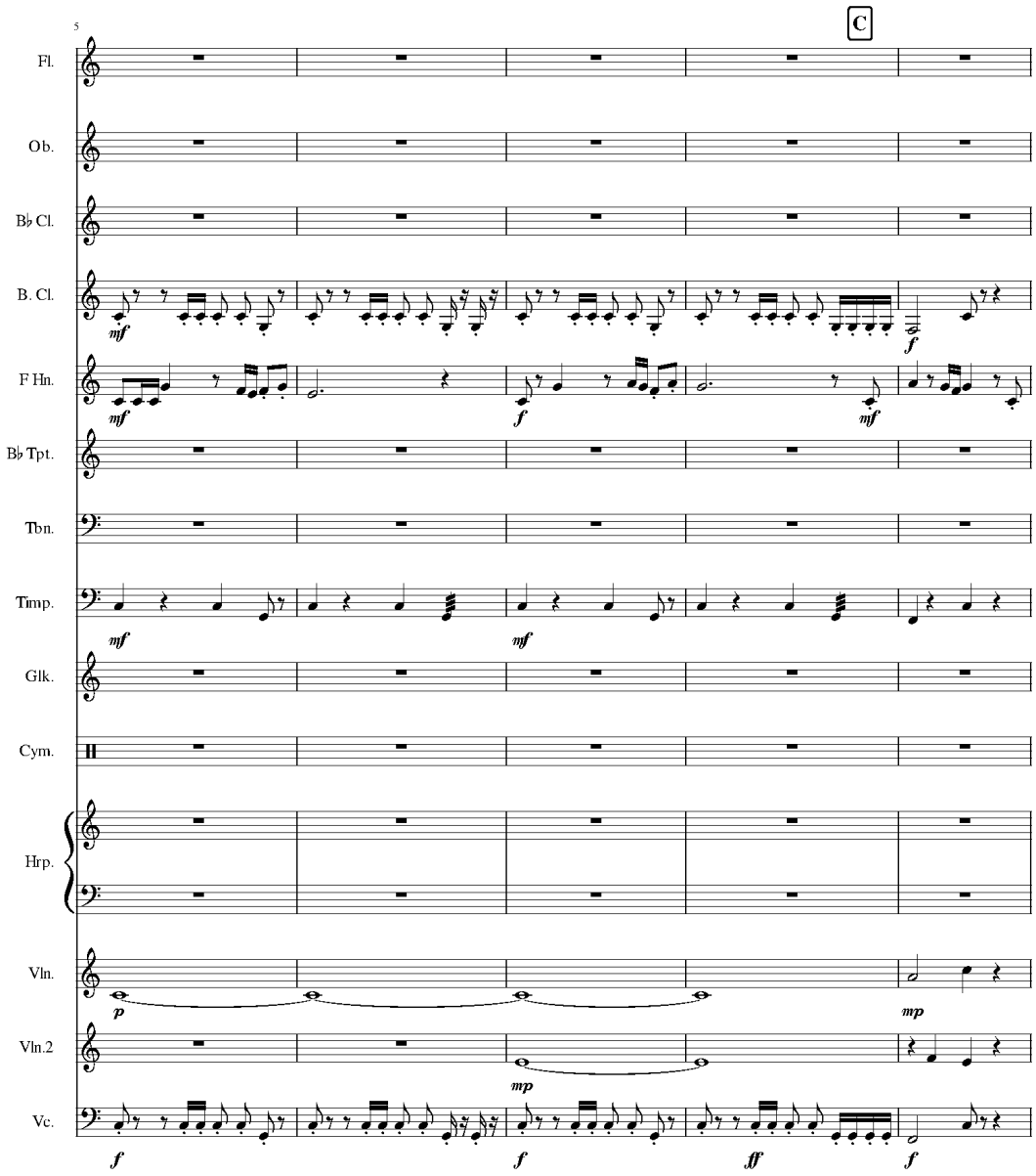

51
M
Fl.
Ob.
F Hn.
Vln.1
Vln.2
Vla.
Vc.
Db.
p
mp
mf
mp
p
57
Fl.
Ob.
F Hn.
Vln.1
Vln.2
Vla.
Vc.
Db.
mp
mf
p
mp
pizz.

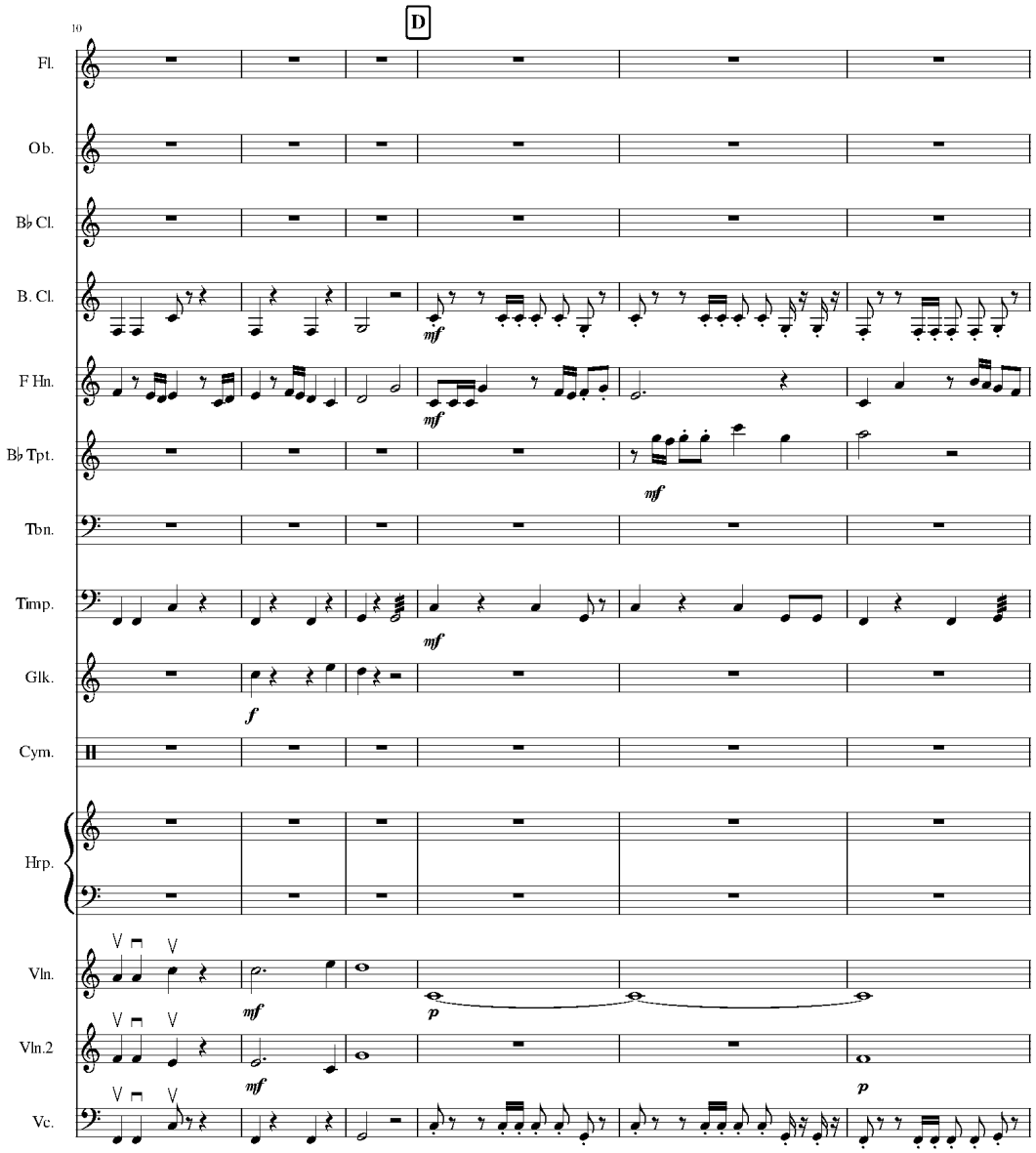


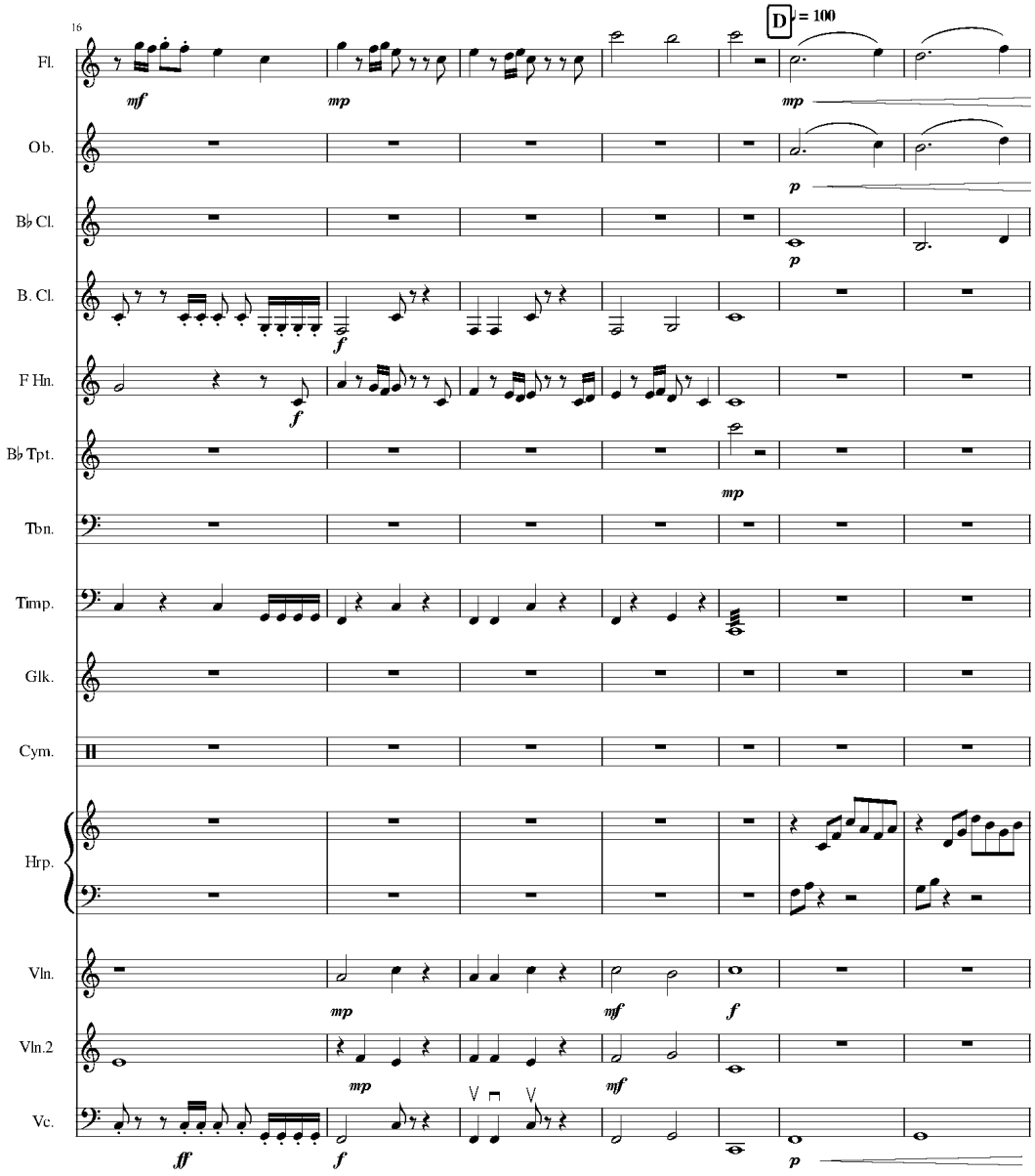
Adventure

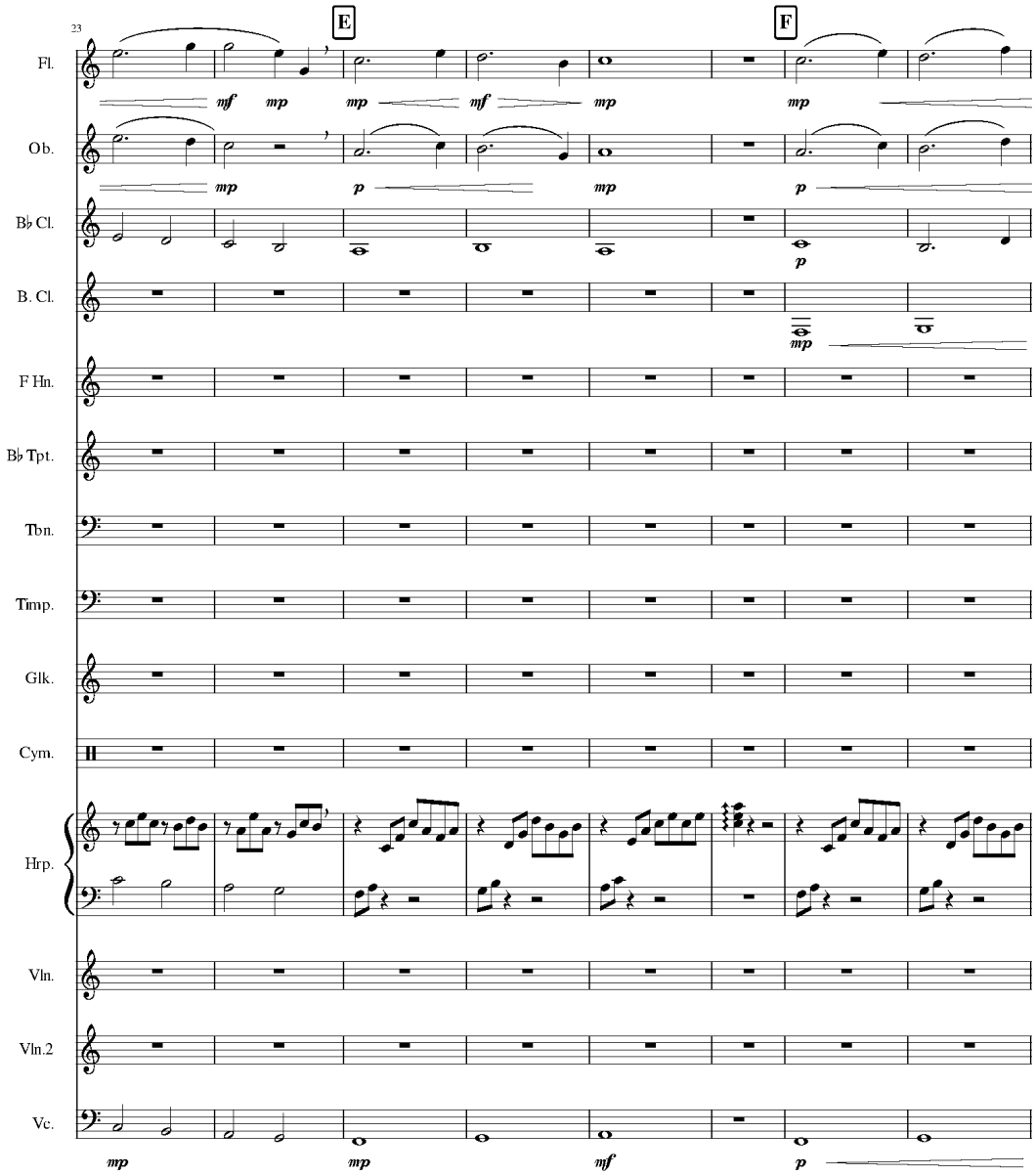
Jacob Hills

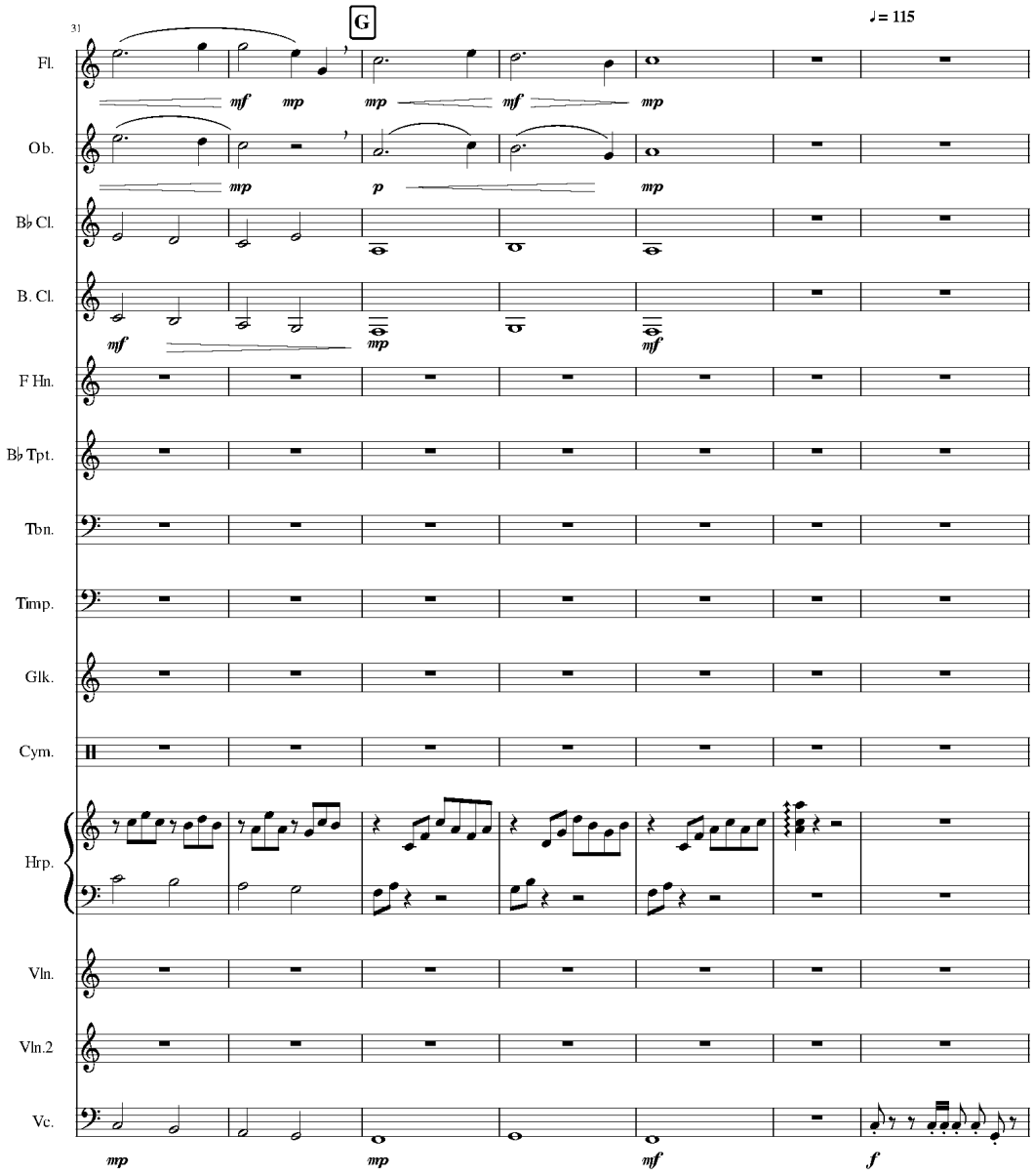


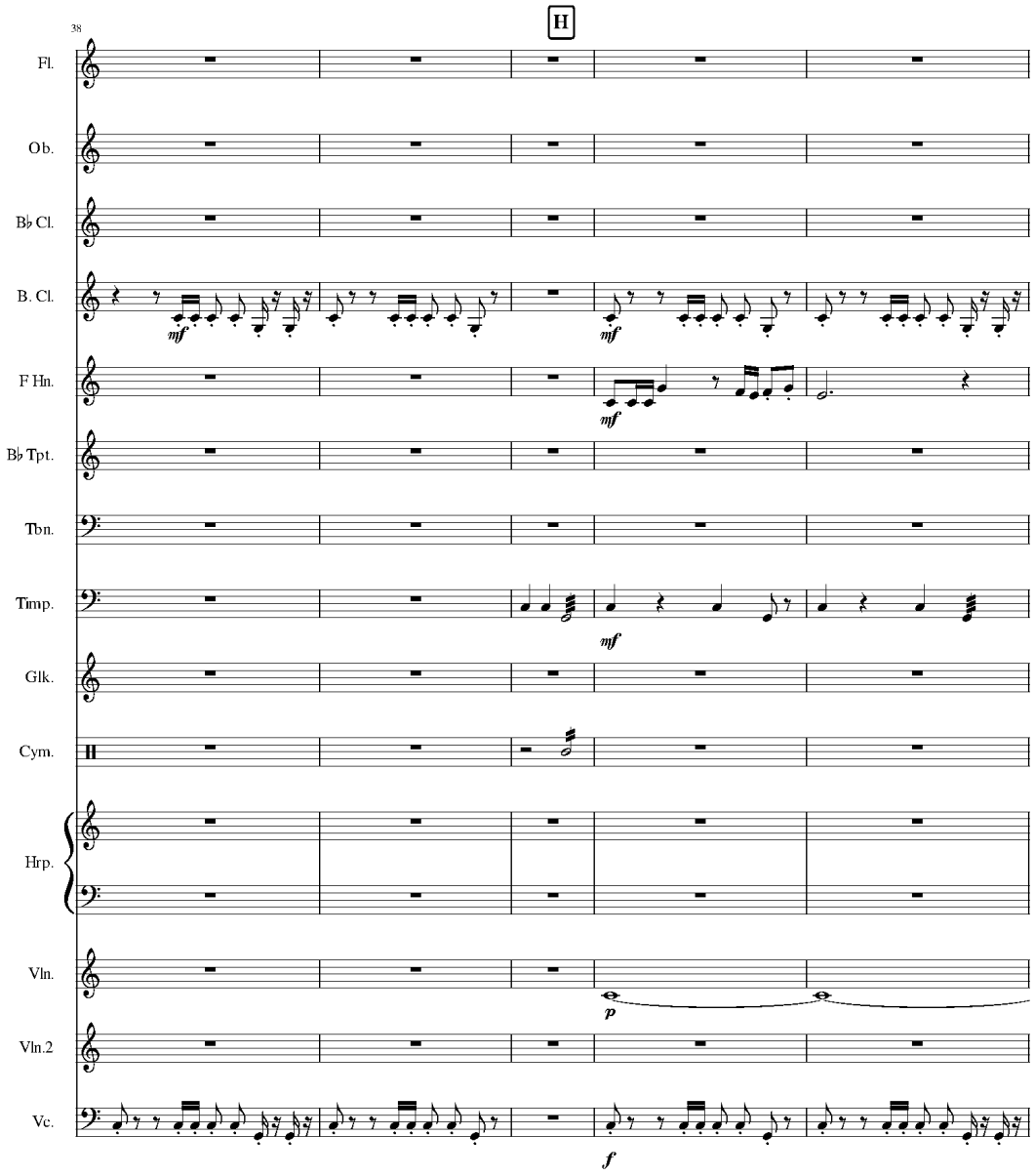

5
C
Fl.
Ob.
B♭ Cl.
B. Cl.
F Hn.
B♭ Tpt.
Tbn.
Timp.
Glk.
Cym.
Hrp.
Vln.
Vln.2
Vc.
mf
f
mf
f
mf
mf
mf
p
mp
mp
f
f
ff
f

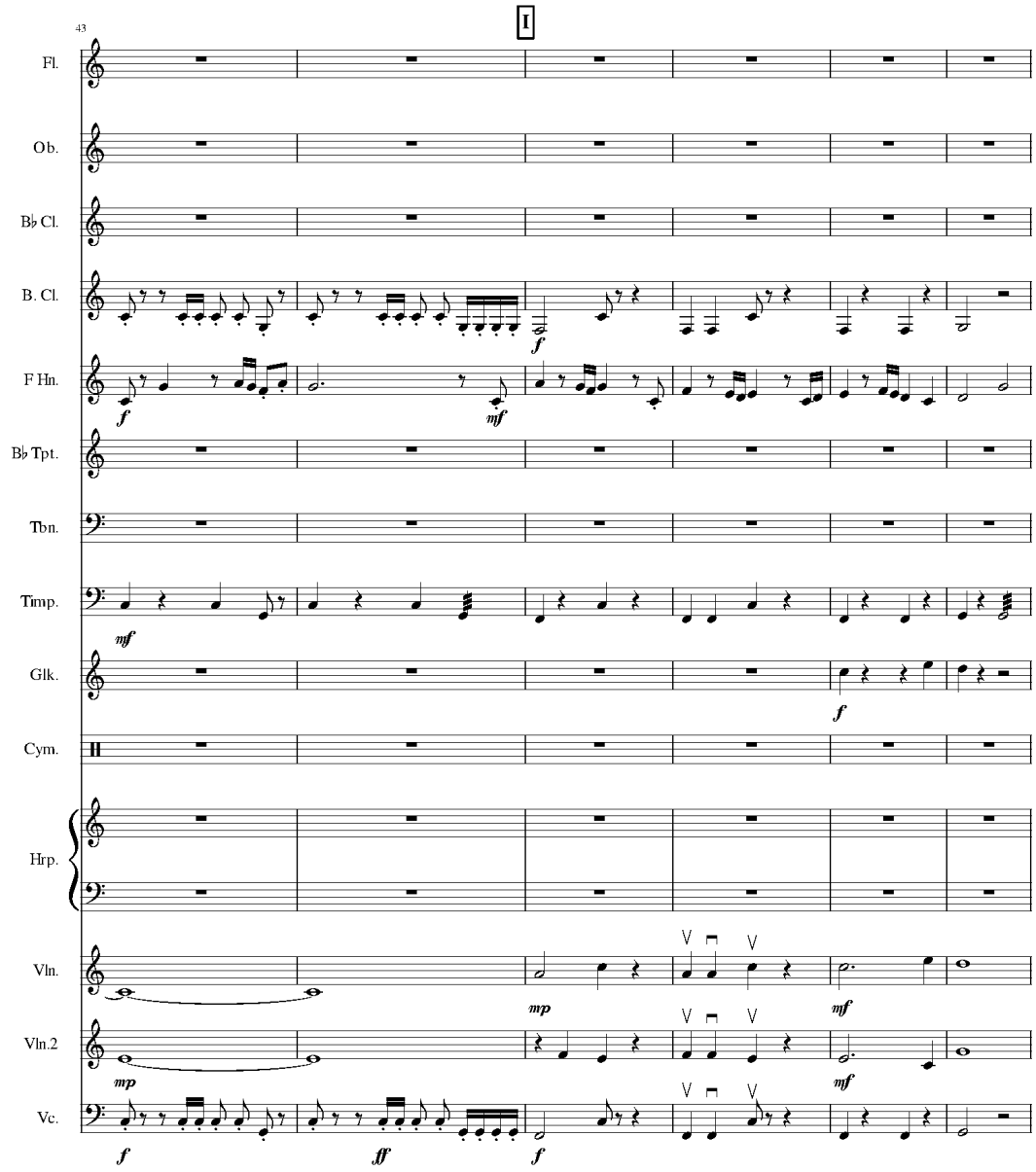

10
D
Fl.
Ob.
B♭ Cl.
B. Cl.
F Hn.
B♭ Tpt.
Tbn.
Timp.
Glk.
Cym.
Hrp.
Vln.
Vln.2
Vc.
mf
mf
mf
mf
f
mf
p
mf
p

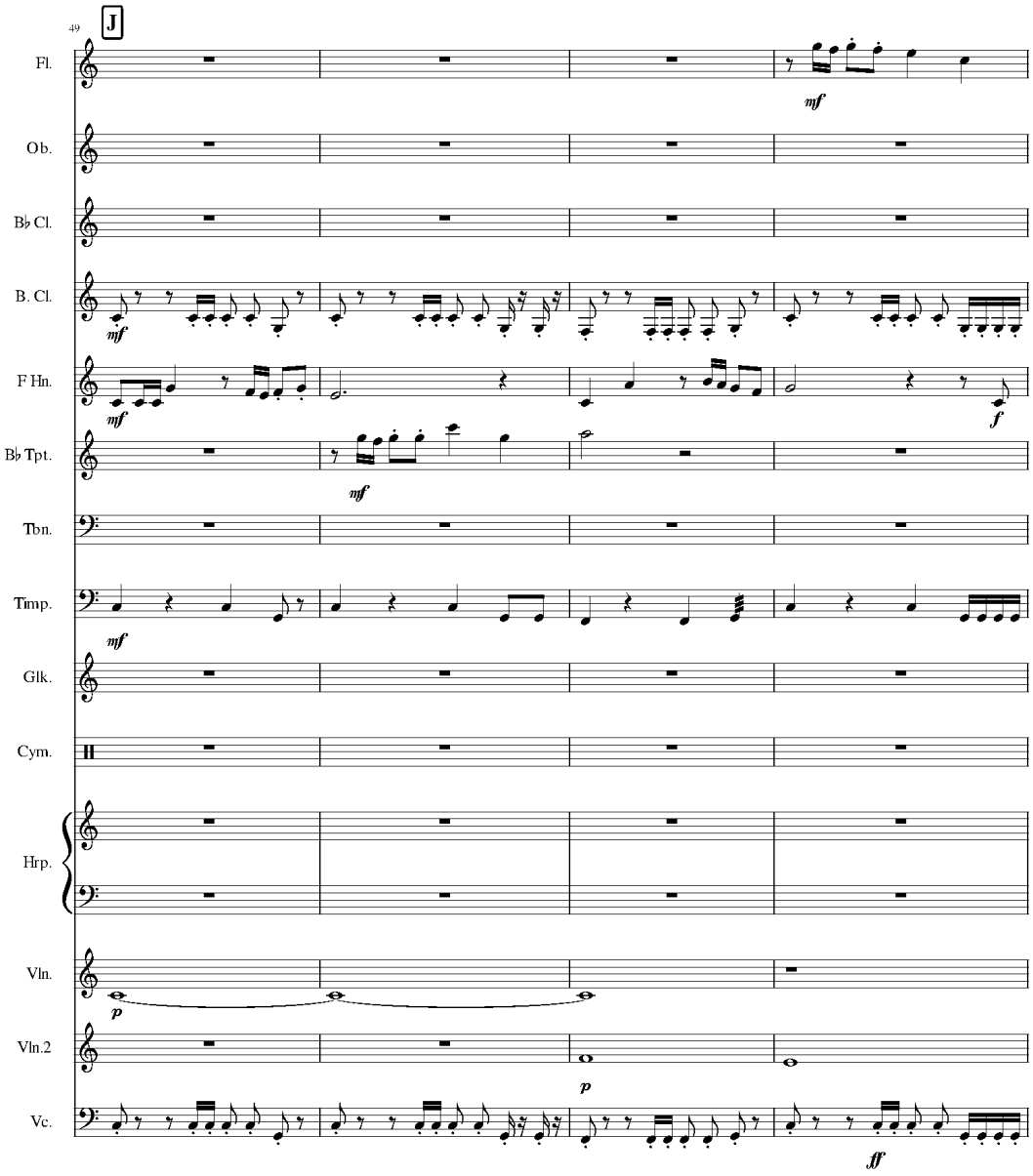


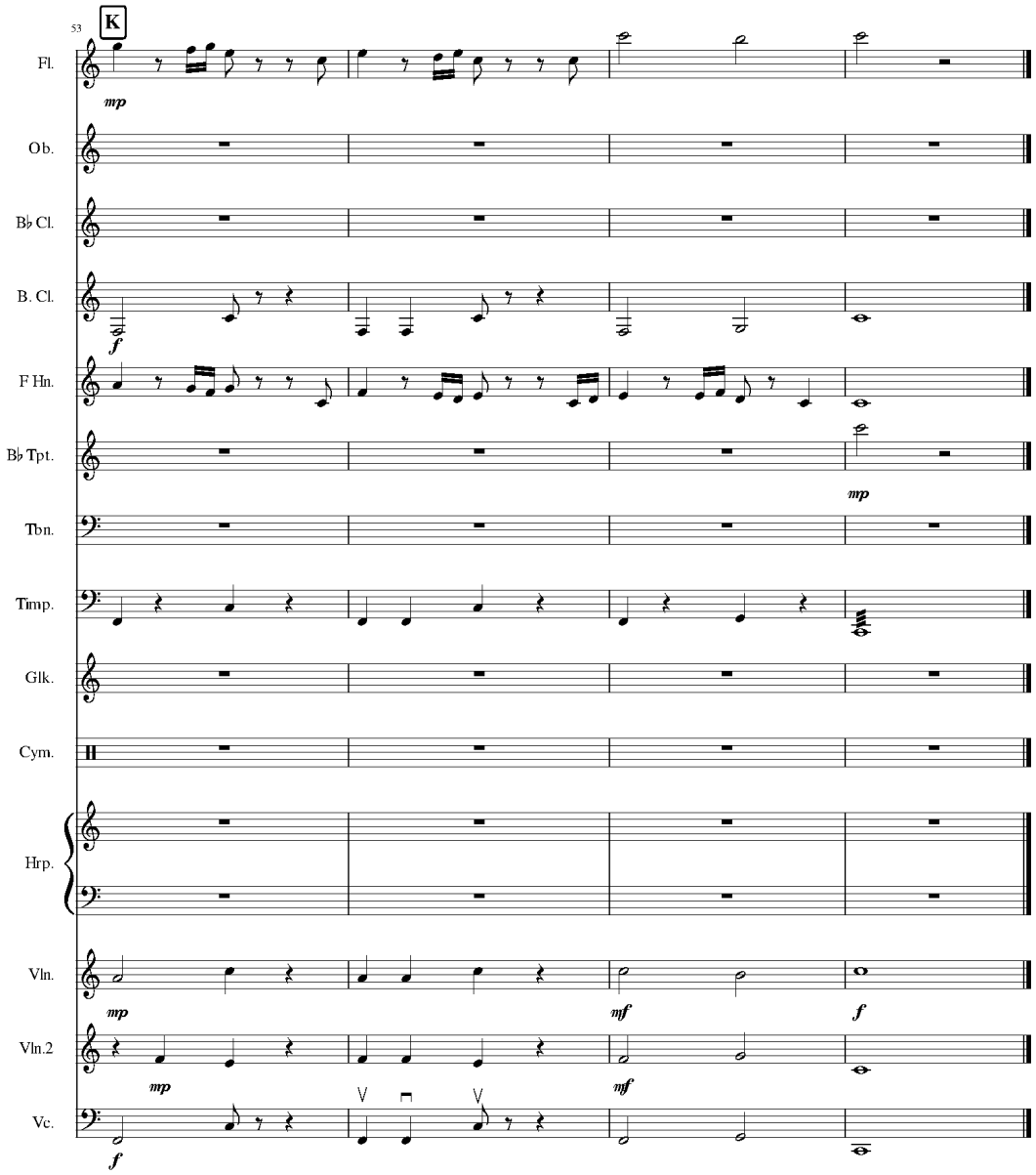



31
G
♩= 115
Fl.
Ob.
B♭ Cl.
B. Cl.
F Hn.
B♭ Tpt.
Tbn.
Timp.
Glk.
Cym.
Hrp.
Vln.
Vln.2
Vc.
mf
mp
p
f


H
38
Fl.
Ob.
B♭ Cl.
B. Cl.
F Hn.
B♭ Tpt.
Tbn.
Timp.
Glk.
Cym.
Hrp.
Vln.
Vln.2
Vc.
mf
mf
mf
mf
p
f

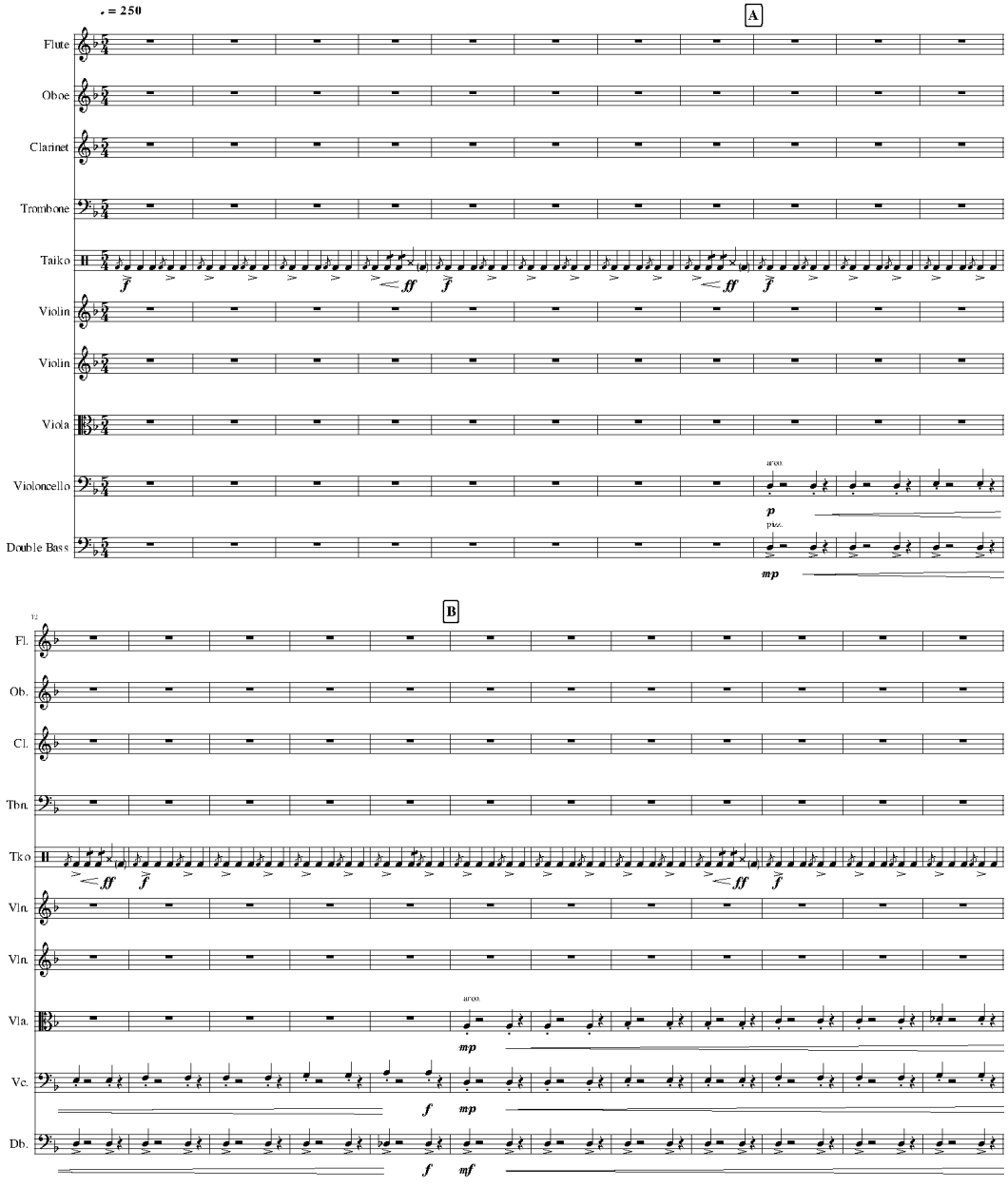

I
43
Fl.
Ob.
B♭ Cl.
B. Cl.
F Hn.
B♭ Tpt.
Tbn.
Timp.
Glk.
Cym.
Hrp.
Vln.
Vln.2
Vc.
f
mf
mp
ff

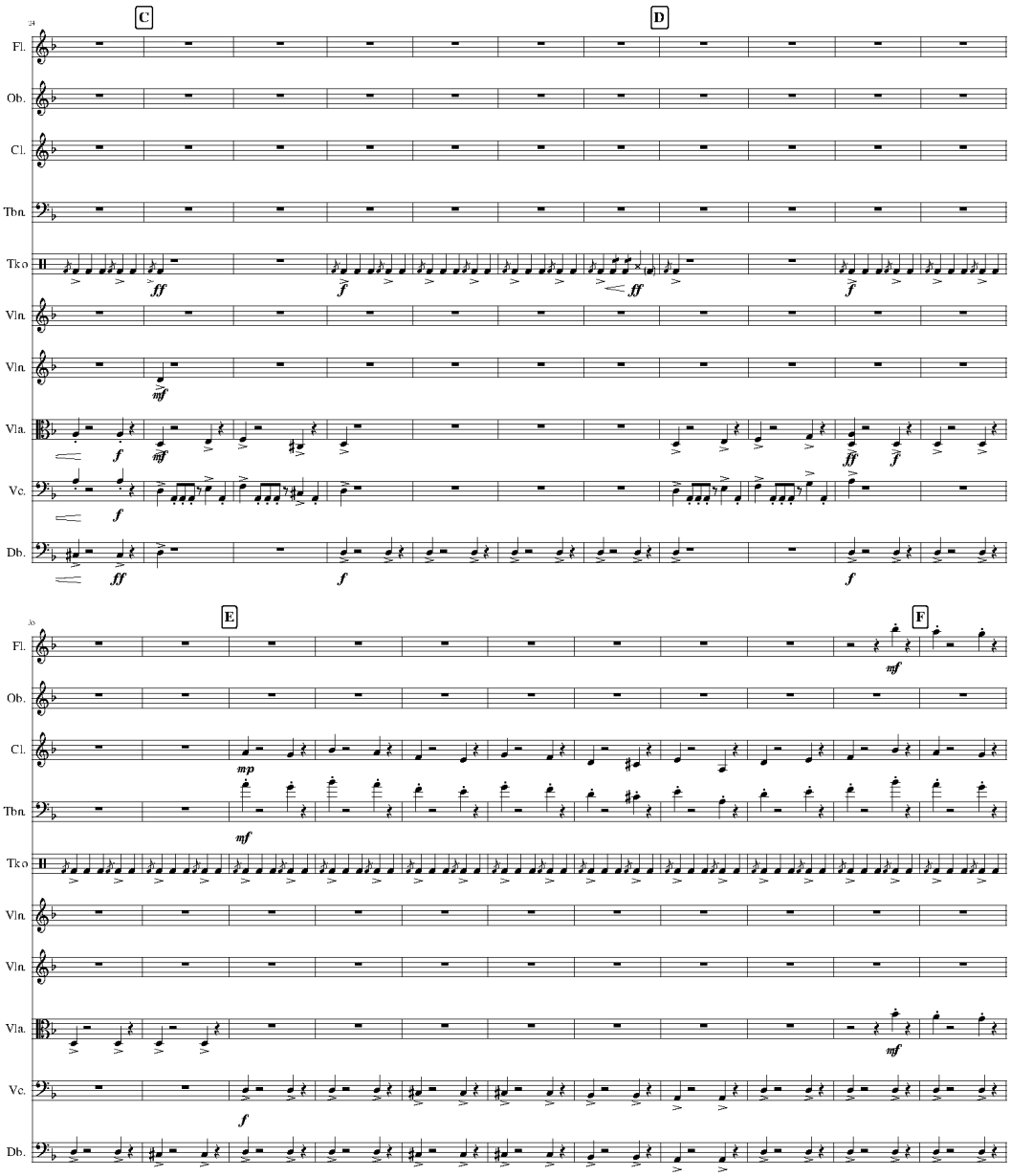

49
J
Fl.
Ob.
B♭ Cl.
B. Cl.
F Hn.
B♭ Tpt.
Tbn.
Timp.
Glk.
Cym.
Hrp.
Vln.
Vln.2
Vc.
mf
mf
mf
f
mf
p
p
ff

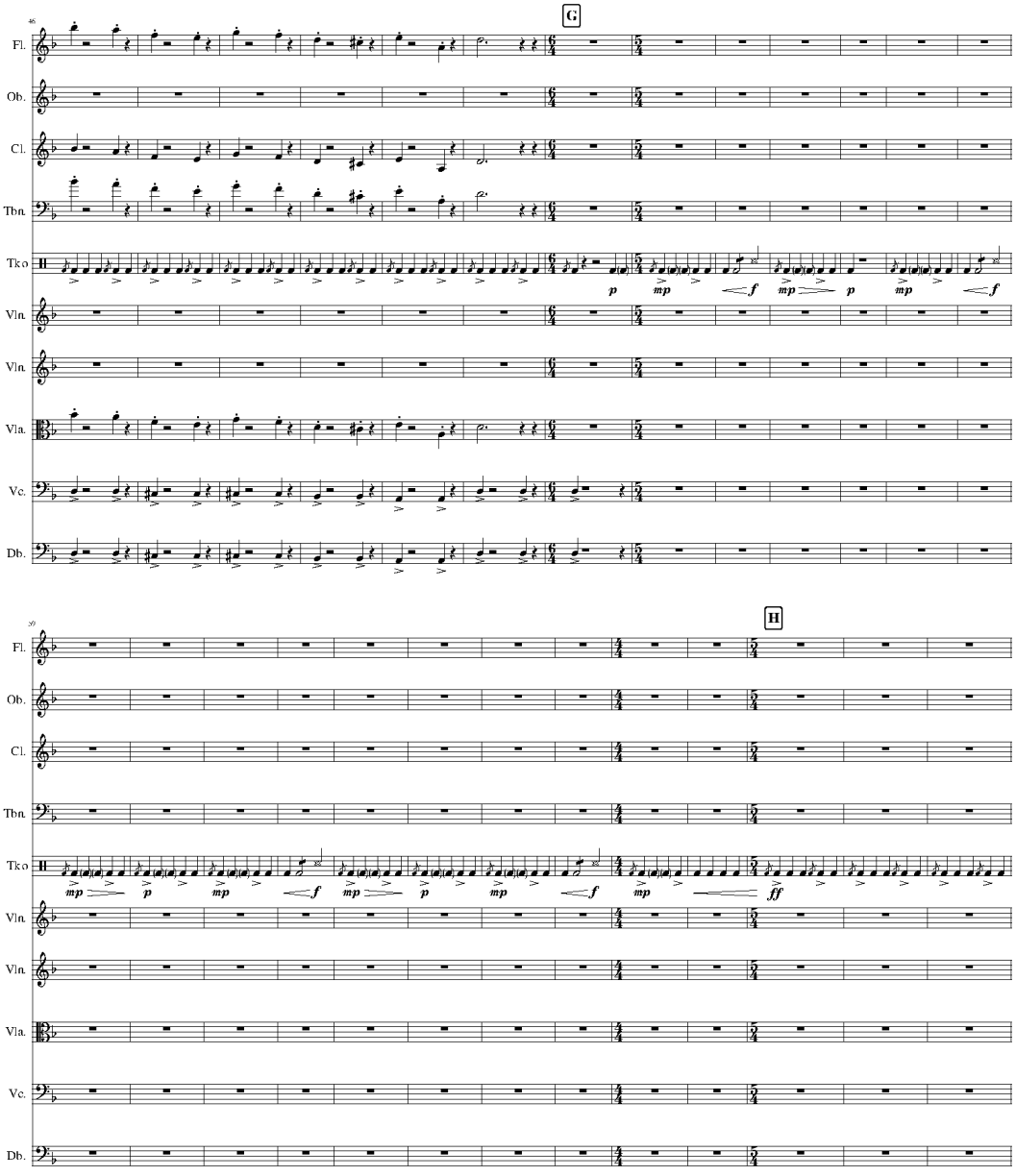

53
K
Fl.
mp
Ob.
B♭ Cl.
B. Cl.
f
F Hn.
B♭ Tpt.
mp
Tbn.
Timp.
Glk.
Cym.
Hrp.
Vln.
mp
mf
f
Vln.2
mp
mf
Vc.
f

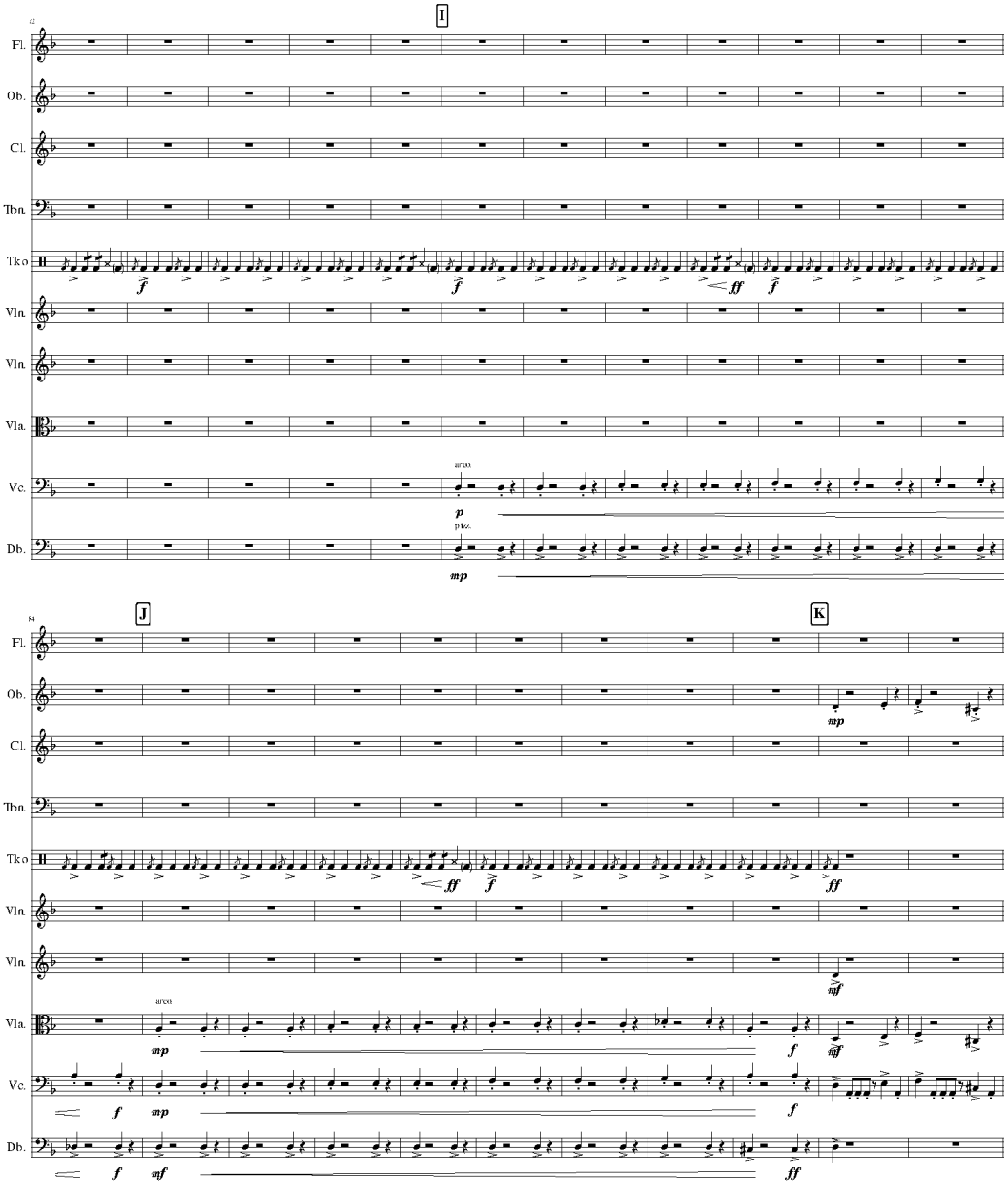
Action

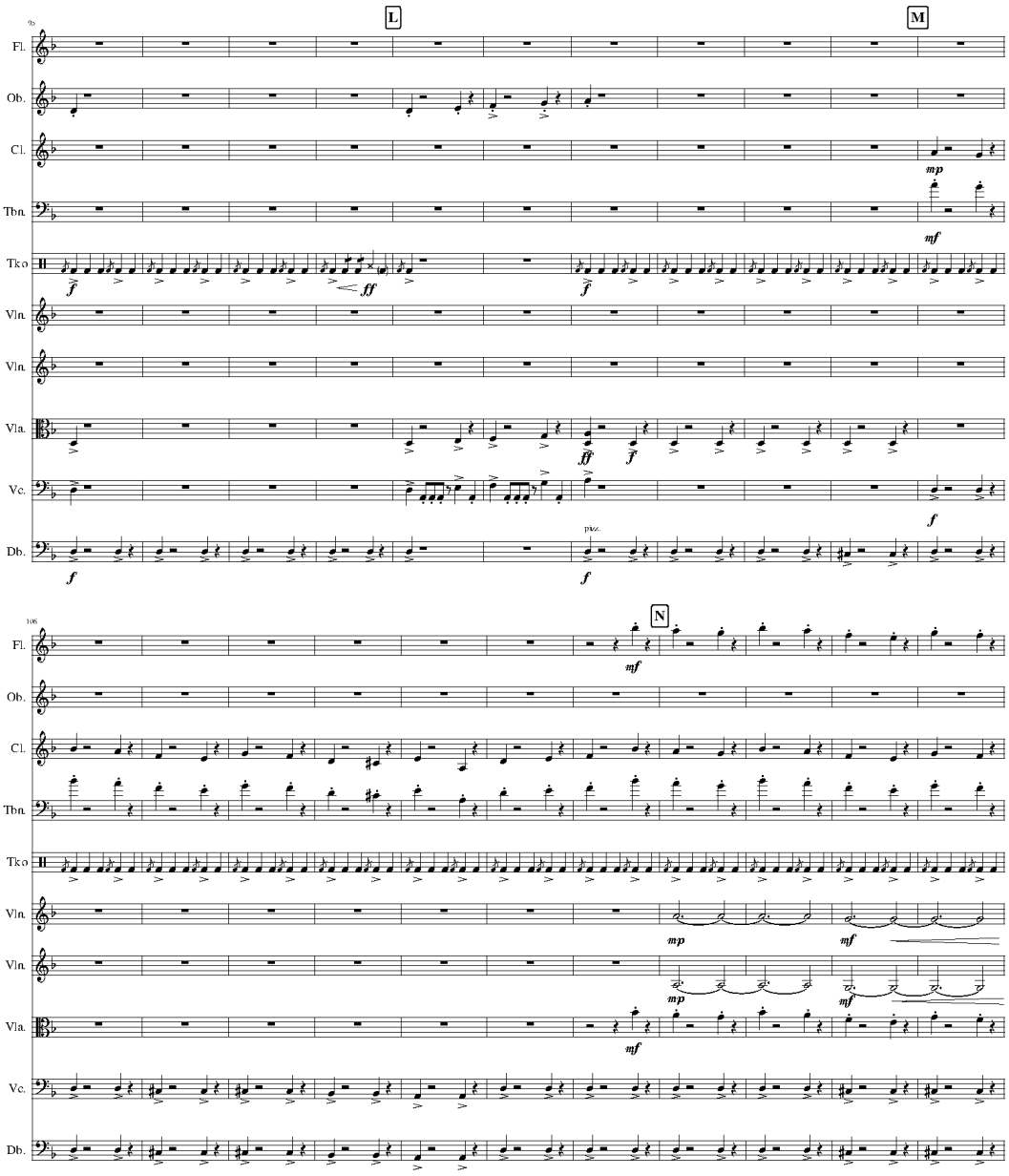
Jacob Hills

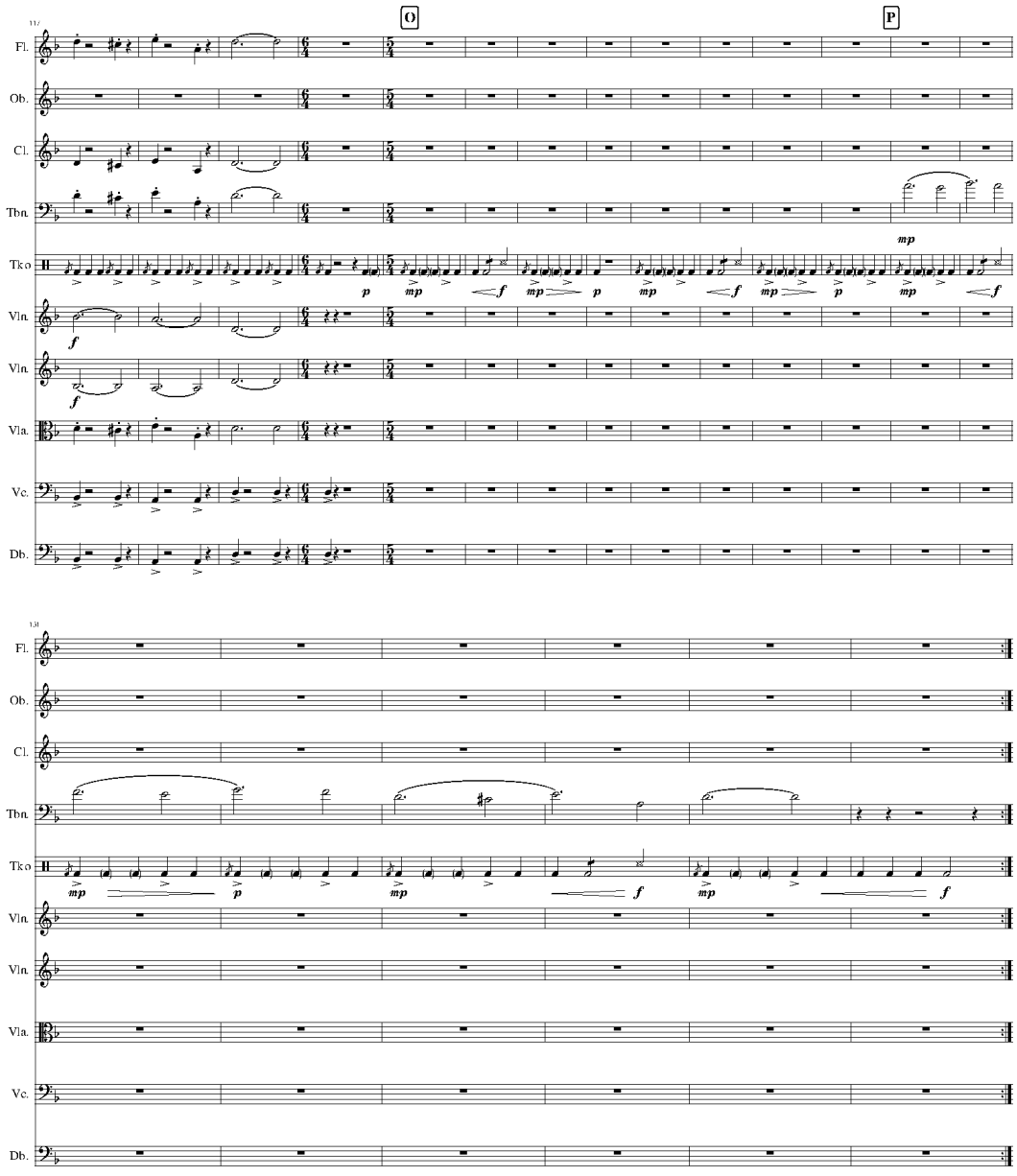




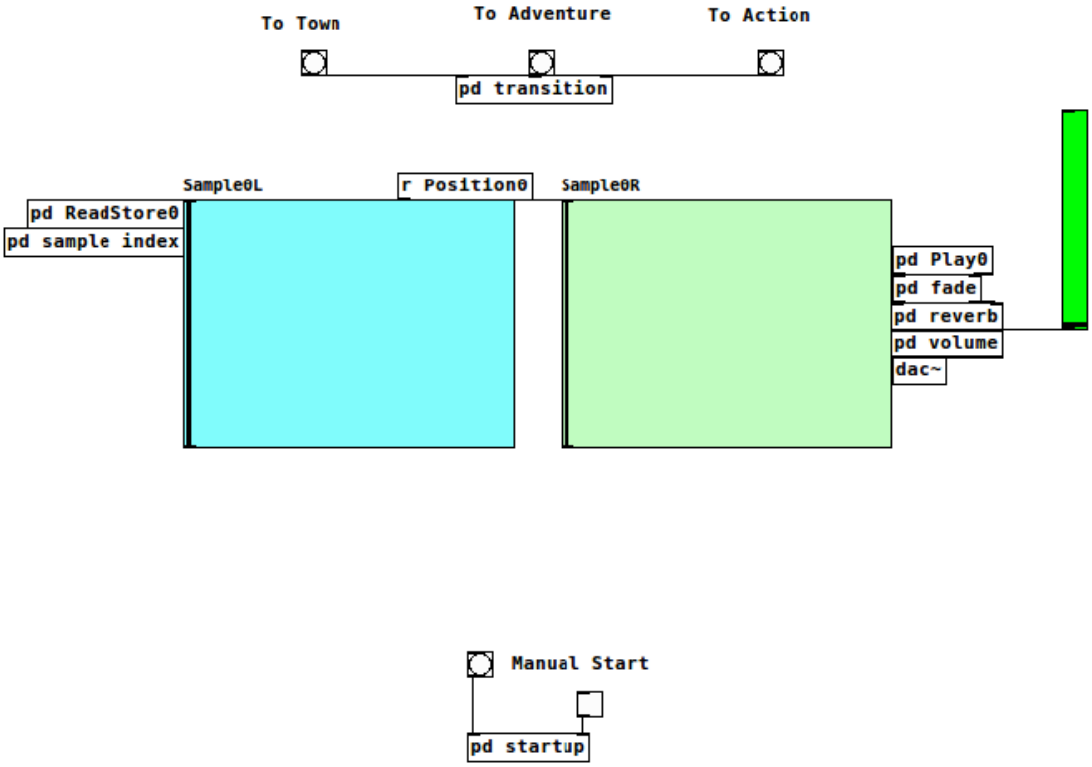



I
Fl.
Ob.
Cl.
Tbn.
Tkɔ
Vln.
Vln.
Vla.
Vc.
Db.
arco
p
pizz.
mp
J
K
mp
ff
f
mf
ff
arco
mp
f
f
f
mp
f
mf
ff

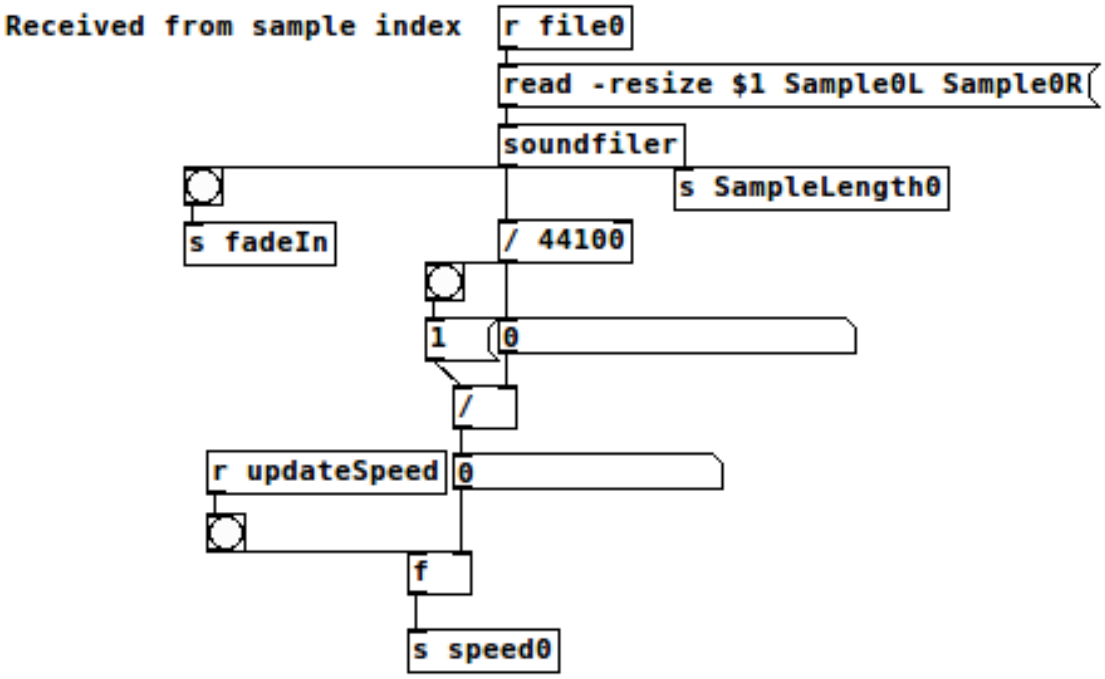




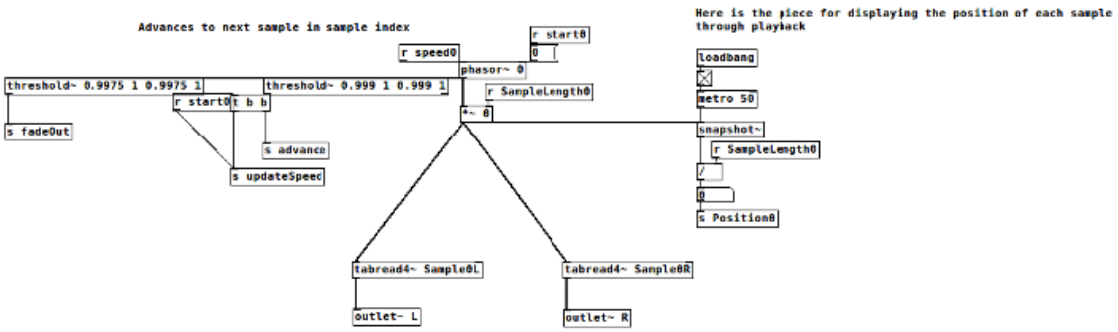
Main Patch


To Town
To Adventure
To Action
pd transition
Sample0L
r Position0
Sample0R
pd ReadStore0
pd sample index
pd Play0
pd fade
pd reverb
pd volume
dac~
Manual Start
pd startup

ReadStore

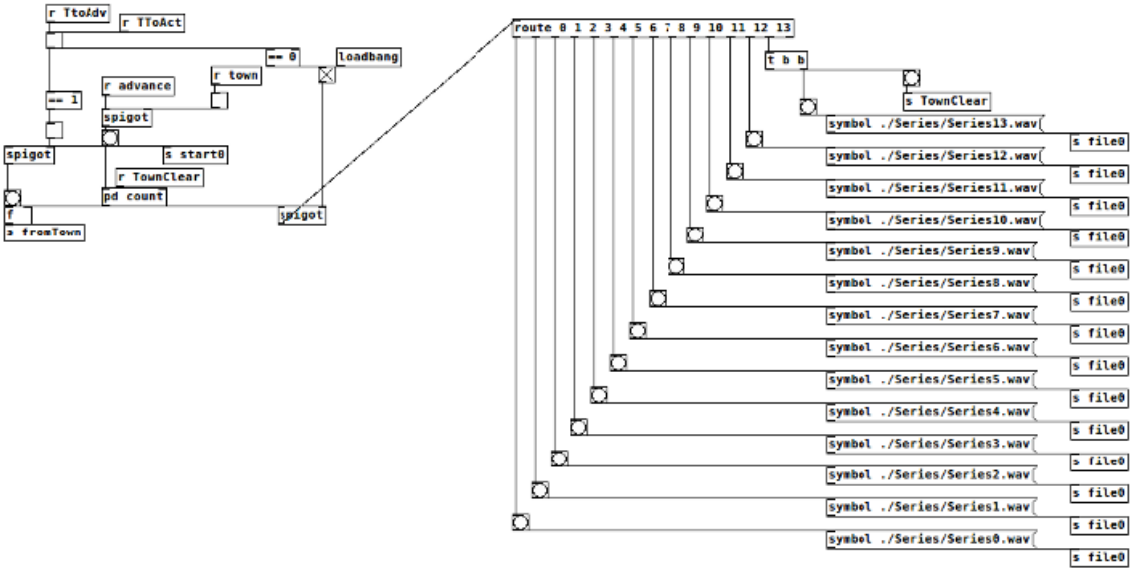

Received from sample index
r file0
read -resize $1 Sample0L Sample0R
soundfiler
s SampleLength0
s fadeIn
/ 44100
1
0
/
r updateSpeed
0
f
s speed0

Play

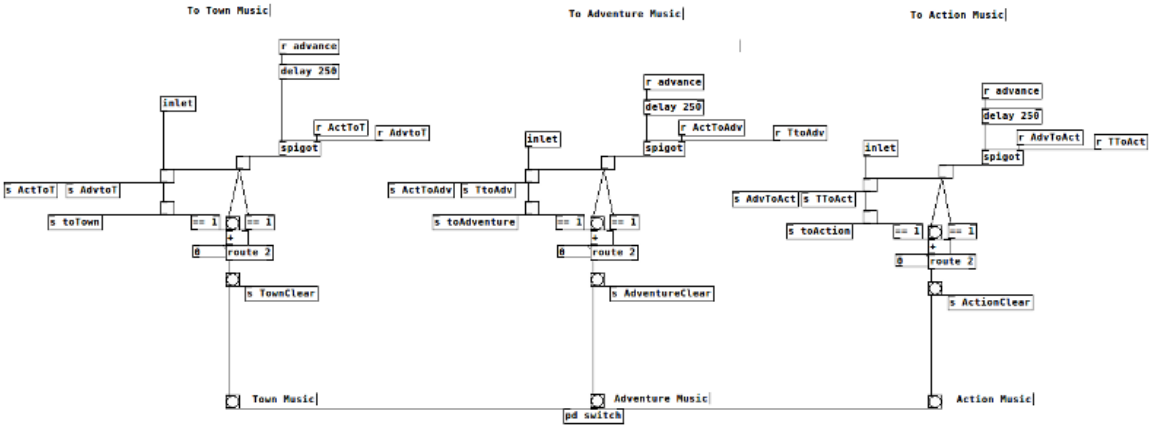

Advances to next sample in sample index
Here is the piece for displaying the position of each sample through playback
r start0
r speed0
0
phasor~ 0
threshold~ 0.9975 1 0.9975 1
threshold~ 0.999 1 0.999 1
r SampleLength0
*~ 0
loadbang
metro 50
snapshot~
r SampleLength0
/
0
s Position0
r start0
t b b
s fadeOut
s advance
s updateSpeed
tabread4~ Sample0L
tabread4~ Sample0R
outlet~ L
outlet~ R

Sample Index

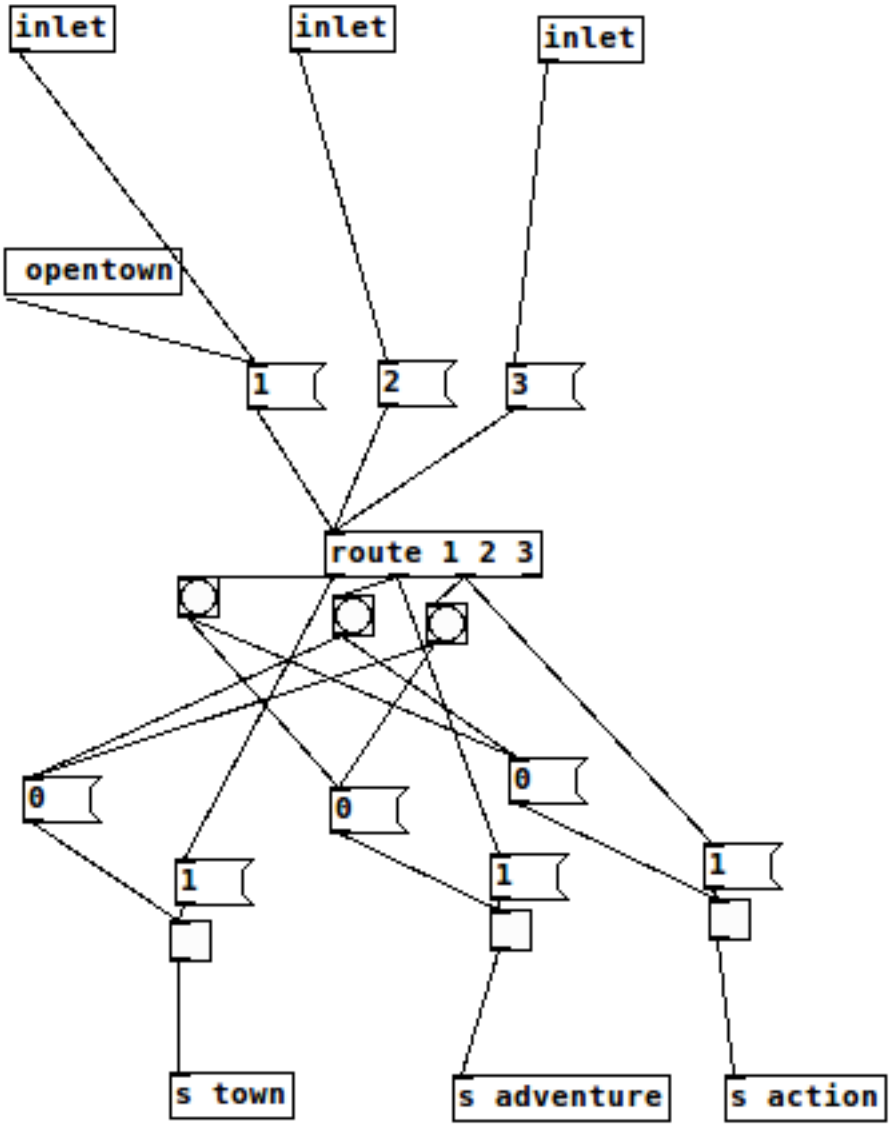
Sample Selector



Transition


To Town Music
r advance
delay 250
inlet
r ActToT
r AdvtoT
spigot
s ActToT
s AdvtoT
s toTown
== 1
== 1
0
route 2
s TownClear
Town Music
To Adventure Music
r advance
delay 250
r ActToAdv
r TtoAdv
inlet
spigot
s ActToAdv
s TtoAdv
s toAdventure
== 1
== 1
0
route 2
s AdventureClear
Adventure Music
To Action Music
r advance
delay 250
r AdvToAct
r TToAct
inlet
spigot
s AdvToAct
s TToAct
s toAction
== 1
== 1
0
route 2
s ActionClear
Action Music
pd switch

Switch


inlet
inlet
inlet
opentown
1
2
3
route 1 2 3
0
0
0
1
1
1
s town
s adventure
s action