UNIVERSITY OF CALIFORNIA,
IRVINE

Advanced Dynamic Music: Composing Algorithmic Music in Video Games as an Improvisatory Device for Players

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Integrated Composition, Improvisation, and Technology

by

Yihui Liu

Dissertation Committee:
Professor Mari Kimura, Chair
Professor Christopher Dobrian
Professor Vincent Olivieri
Assistant Professor Theresa Jean Tanenbaum

2021

DEDICATION

To

my family and friends

in recognition of their worth

"But every time you hurt me, the less that I cry

And every time you leave me, the quicker these tears dry

And every time you walk out, the less I love you

Baby, we don't stand a chance, it's sad but it's true

I'm way too good at goodbyes"

Lines from a song by Sam Smith called *Too Good at Goodbyes* (the remixed version by Snakehips). I have been listening to this song hundreds of times while I was writing this paper and it has become a musical inspiration to me in many ways.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

ADM	Advanced Dynamic Music
ADSR	Attack, Decay, Sustain, Release
API	Application Programming Interface
BPM	Beat Per Minute
CG	Computer Graphics
DAW	Digital Audio Workstation
DOD	*Do or Die: A Roll Playing Adventure*
DSP	Digital Signal Processing
DVL	Distance of Voice Leading
FPS	Frames Per Second
GID	Generative-improvisatory Drum Group in the *Gameplay* movement
HPF	High Pass Filter
HSF	High Shelf Filter
IP	Immediate Pitch
m./Mm.	Musical Measure/Measures Number
MIDI	Musical Instrument Digital Interface
Mvt.	Movement
PC	Pitch Collection
QA	Quality Assurance
RPG	Role Playing Games
RTPC	Real Time Parameter Control

ACKNOWLEDGEMENTS

I would like to express the deepest appreciation to my committee chair, Professor Mari Kimura, who is a significant female composer and artist working in the field of interactive computer music. She has been a hero and role model to me, and I still recall the first class of hers that I attended at Juilliard which had started an adventurous and rewarding journey for me in music composition and music technology. She has also been an important life mentor to me who understands the struggles I have as a classically trained musician and have been so patient and caring to help me to become who I want to be and who I am today. Without her support and trust, I would not have been able to grow so much.

I would like to thank my committee members, Professor Christopher Dobrian and Professor Vincent Olivieri. Prof. Dobrian's musicianship and mastery in composition, music technology, and computer music have nourished my musical creativity and research in many ways. I have learned so much about sound design and digital audio in Prof. Olivieri's classes as well as the experience working with him on collaborative projects. Prof. Dobrian and Prof. Olivieri have always been an informative, supportive, and inspiring advisor for me. They have also been trusting my capabilities and work in written forms as well as actions. I have been offered many opportunities to present and share my work in forms of presentation, teaching, and collaborative activities. These opportunities not only enriched my scholarly capabilities but also allowed me to share my thoughts and ideas with other scholars.

In addition, a thank you to Professor Theresa Jean Tanenbaum, whose research and teaching in game studies as well as her feedback on my dissertation paper has provided me opportunities to enter the world of video games as a composer. I have participated three times in the local site of the annual Global Game Jam, organized by Prof. Tanenbaum at UCI, which is one of the most fun, memorable, and fruitful experiences that I had at University of California, Irvine. I have also collaborated and participated in multidisciplinary projects led by Prof. Tanenbaum and work with her students who come from a diverse background and disciplines. As a classically trained musician who used to play mostly solo music, these teamwork experiences have been life-changing for me.

I thank the University of California, Irvine for the financial, academic, career, cultural, and personal development support, as well as many other important mentors, advisors, and colleagues of mine who have provided guidance and feedback to my work.

Lastly, I thank the lovely beaches and weather of southern California and the welcoming and enthusiastic Californian people for participating in my unforgettable four years of life.

VITA

Yihui Liu is a composer, singer-songwriter, music producer, and keyboardist. Her music creativity embraces long-standing traditions and contemporary practices, merges technical skills and aesthetic purposes, celebrates commercial and artistic meanings in music, values conventional techniques and new technologies, and encompasses digital production and live performance. She evaluates cross-style awareness within the field of music as well as cross-disciplinary collaborations that bridge music with other fields. Yihui believes in the beauty of multitudes in music in the way she creates, researches, and teaches music.

2015	B.M. in Piano Performance, Oberlin College Conservatory
2017	M.M in Piano Performance, The Juilliard School
2021	Ph.D. in Integrated Composition, Improvisation and Technology, University of California, Irvine

ABSTRACT

Advanced Dynamic Music: Composing Algorithmic Music in Video Games as an Improvisatory Device for Players

by

Yihui Liu

Doctor of Philosophy in Music

University of California, Irvine, 2021

Professor Mari Kimura, Chair

Video games as interactive media provide a natural venue for algorithmic music as the soundtracks which are dynamic, adaptive, procedural, interactive, modular, and generative. This dissertation explores the composition of algorithmic game music as a device for transforming gameplay into musical improvisation, which I call Advanced Dynamic Music (ADM). The ADM system combines various compositional techniques and procedures with a real time algorithmic strategy that links types of music and game parameters to generate or "improvise" game music interacting with the actions of players. This paper first gives a review of existing theoretical and musical literature on game music. Based on this literature review, I explore the compositional approaches in two creative game music projects: *Do or Die: A Roll Playing Adventure* and *Mastery*.

Through my experience working on the two projects, I propose eight defining properties of ADM: variability, user-interaction, modularity, game-music association, multidimensionality, multiplicity, staticity, and coherency. ADM is the very first systematic and specific definition of a type of highly dynamic game music, which derived and developed from my experience in the two creative projects as the composer, game-music interaction designer, and music-game programmer and integrator. ADM can be used as an analytical methodology or taxonomy for game music scholars, as well as a compositional framework and solution for game composers in the future.

Chapter 1. Introduction to Game Music

1.1. Purpose of This Dissertation

In video games, music is dynamically triggered by algorithms due to the fact that video games are algorithm-driven interactive media programmed like any piece of computer software. In game code, musical cues often coincide with particular game events, so that the music can sync with the game progress like soundtracks accompanying films. The unique interactivity of video games provides opportunities for its soundtracks to have complex dynamism. This paper introduces Advanced Dynamic Music (ADM), which defines a type of game music that involves highly dynamic and interactive musical content. In a game with ADM scores, the gaming experience may simulate music performance. Firstly, the composer translates various types of compositional procedures into algorithmic processes that the computer can understand and perform. Secondly, those musical algorithms are connected with player input and types of game states or parameters. Lastly, when the game runs in real time, the game engine procedurally executes the music score live. A game controller therefore works like an instrument or a conducting baton that controls the performance of a music piece (Figure 1).

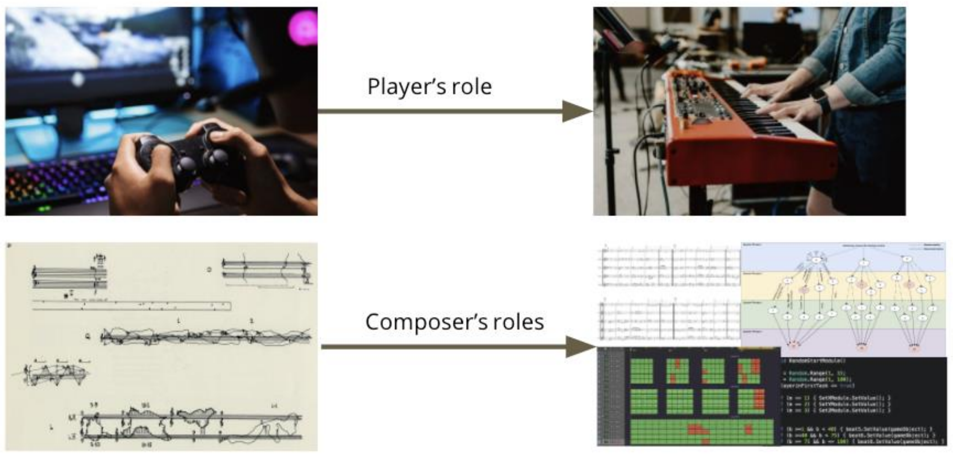


Figure 1 New roles to players and game composers in ADM.

Video games import and play a soundtrack, often as either one complete audio file or as chopped fragments, for games to compile in real time gameplay. When using chopped fragments, the musical structure of the composition embodies modularity (Medina-Gray, 2019). The music modules function as the building blocks that will only turn into a complete architecture when a player interactively participates and manipulates the modules in real time gameplay. Composers can utilize various types of musical parameters, such as phrasing, voicing-leading, and pitch generation, to create mobile musical modules and design the algorithmic music system. This system manages how modules are put together in gameplay. Players can then *perform* the music generated by the algorithmic music system while creating dynamic variations, like jazz musicians improvising on a pre-composed tune, all by simply playing the game.

The ADM system not only offers new roles and experiences to the players by superimposing music performance and improvisation to gaming, but I hope it can also introduce and inspire game composers to take advantage of creative techniques, procedures and approaches involved in a highly dynamic music system, which existing game music literature has not yet fully explored. The interactive and flexible nature of video games places challenging new demands on composers (Scott, 2014). A traditional composer would normally work with a written score or notation. A screen composer or electronic music producer also works with music production in digital audio workstations (DAWs) or other types of music sequencers. A computer musician might work with music-programming tools (e.g., MaxMSP, Pure Data) for composing algorithmic music. An audio software engineer develops algorithms to synthesize, process, sequence, and manipulate sounds and music in types of audio plug-ins or packages in game engines. However, an ADM game composer should be good at or at least familiar with all the tasks mentioned above. Through familiarity in these areas, composers will know the constraints and possibilities of the medium, enabling them to achieve, integrate, and execute a range of musical procedures.

The following section of this chapter will briefly introduce the historical background and the stylistic diversity of game music. The next chapter discusses previous scholarly studies and literature in game music, including musical and theoretical studies and compositional techniques and analyses. In chapter three and four, I discuss and document two of my game music composition projects, *Do or Die: A Roll Playing Adventure* and *Mastery*, grounded on the review of previous literature on specific compositional, designing, producing, and programming techniques and approaches.

Through my experiences doing two creative projects, I then propose and discuss eight defining properties of ADM that were previously missing from existing literature in chapter five. The eight defining criteria are: variability, user-interaction, modularity, game-music association, multidimensionality, multiplicity, staticity, and coherency. This new approach and theory support game composers in creating highly dynamic music systems that contain: more musical variations, more interaction with the player, better musical continuity, more meaningful musical expression connected to in-game events, more live-sounding realistic music, more multiplicity in the music variations, better stylistic identification and consistency, and better musical coherency.

1.2. Style-diverse Game Music

The earliest electronic video games emerged during the late 50s and early 60s in arcades, and only began to have sound in the early 70s (K. Collins, 2008). The earliest game music appeared with the first widely popular home game console - the Nintendo Entertainment System (Belinkie, 1999). Along with the technological evolution of video games, game audio has developed through "three main stages": the first stage of 8-Bit 'Bitcore' or 'chip-tune' (Wikipedia, 2021c); the second stage of the so called "in-between" years featuring the inclusion and utilization of MIDI, audio synthesis, and more sophisticated music; and finally the current stage, with the involvement of CD quality audio (and better) - a stage with the least amount of limitations for game composers and sound designers (K. Collins, 2005).

Soundtracks of video games in general embrace a broader range of musical styles and traditions than any other forms of visual media such as films or *anime* due to games' broad range of possibilities in genre, art style, narrative, and themes (Belinkie, 1999). Existing examples of music styles in video games include electronic dance music (*GNOG* and *Punishing: Gray Raven*), rock (*Killer Instinct: Season 3*), fusion (*Persona 5*), jazz (*L.A. Noire*), Medieval, epic orchestral, world music (*The Witcher 3: Wild Hunt*, *Civilization VI*, and the *Richman series*), punk, hip-hop, and rap such as in *Cyberpunk 2077*. Additionally, these examples have not even included diegetic soundtracks, such as music that a game character hears from a radio inside the game world, which would further the stylistic possibility of music appearing in games. Some types of game music might not fall into any specific style or genre other than "cinematic" music. They appear frequently in games' cut-scenes, often called CG (computer-graphics) videos[1], the part of a game where the player does not control the gameplay but watch a computer-generated short movie that introduces the next plot or back story, such as game trailers and introductory videos (*Diablo IV* and *Honor of Kings*), and CGs

[1] A CG stands for computer graphics which are animated short movies played in video games usually used to show the narratives that are important or informative to the players to proceed further in the gameplay. (Wikipedia, 2020a)

between battles (*God of War*). This cinematic music functions similarly to film music, which synchronously enhances the visuals, the immediate emotion of the images, and the unfolding of plot.

The technological evolution in both video game development and game audio has established certain stylistic expectations from the players that particular types of music match particular types of game. For instance, although technologies of today's video games have advanced tremendously from the 8-Bit chip-tune period, many pixel games[2] developed in the last decade are still accompanied by chip-tune music, such as *Super Time Force*, *Katana ZERO*, and *Monster Crown*, or at least preserve the 8-Bit quality of synthesized timbre as an element in the game soundtracks, such as *Hotline Miami*. This shows that a stylistic association has formed between game types and music genres. While some pixel games intentionally use other styles of music as soundtracks - such as *Stardew Valley* and *To the Moon*, which both have acoustic instrumental music - the stylistic association between a music genre and a game type can also express a form of commemoration and salute to the tradition of game development.

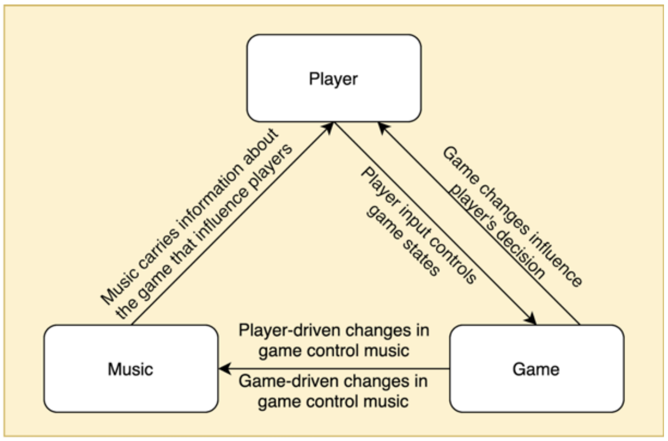
In conclusion, from no sound to immersive, CD quality music (and better), soundtracks in video games have developed their own musical tradition and player expectations. This tradition is unique and specific to video games and is quite different from screen music or other forms of visual media. The immersive and interactive quality of game experience carries massive thematic and artistic potentials that also provide venues for a variety of musical styles and genres in its soundtracks. It is advisable to say that as long as the game production team and the composer agree on the style of music that fits a game, there are no limitations to music styles in video games.

[2] A genre of video game that its graphics are built with pixels (Wikipedia, 2019).

Chapter 2. Research and Literature on Game Music

2.1. Characteristics of Game Music

Many scholars have described video game music as procedural, dynamic, adaptive, interactive, modular, algorithmic, and generative (Clark, 2007; K. Collins, 2009; Gungormusler et al., 2015; Hoffert, 2007; Kaae, 2017; Somberg, 2019; Szinger, 2017; Whitmore, 2018; Young, 2012). While these descriptions express similar meanings, they demonstrate the media-specific potential of game music, which has provided theoretical foundations to my ADM system in this paper. These characteristics of game music depend on a triangle shape collaboration formed between the music engine, game engine, and the player, as shown in Figure 2. Between the player and the game, there is a bi-directional chain of interaction, where the player's decision, control, and other input data directly influence the game engine and control the progression of the gameplay. The different states expressed by the visuals or narratives also provide immediate response to the player's previous control and influence the player's decisions. The interaction between player and game creates an ongoing exchange of information. The music engine is inevitably controlled by algorithmic commands sent from the game engine regardless of whether they are game-driven or player-driven. The music then sonically delivers information to the player which influences the player's decisions in the game afterwards.


Player
Music carries information about the game that influence players
Player input controls game states
Game changes influence player's decision
Player-driven changes in game control music
Music
Game-driven changes in game control music
Game
Figure 2 Music, game, player collaboration triangle.

It is arguable that the music engine should be one of the many components of the game engine rather than an independent subject in this triangle, given that video games exist as a multi-disciplinary medium. However, music has one unique characteristic that significantly differs it from other components of games. This unique characteristic is that music has its own timing, which is independent from the game's timing interactively influenced by the player. Thus, this characteristic creates another dimension of interactivity between the player and the music engine, and between the music engine and the game engine. Music could potentially be one of the few art forms that are heavily "trapped" in the realm of time. Music happens during time, moves in time, and unfolds in constant pace at musically defined time units (tempo, meter, beat). Additionally, unlike motion pictures, which put static images in time to realize concrete movements, music is abstract. Music's movement in time does not aim for realization. Music creators use their work to provide their audiences with a sonic journey that occurs through time. Every musical piece inherently embodies this time-travelling journey, and this journey requires careful and logical design by the composers.

According to Collins, "all game music is procedural," because the audio elements of a game "evolve in real time according to a series of rules set out in the game's software engine (K. Collins, 2009)." The word "procedural" also describes the technical implementation and programming of game music - the process of connecting music to the game code so that a particular music segment can be triggered at the correct moment.

Game music is dynamic also because of the way it is implemented and triggered in games. The dynamism makes the unfolding order, pattern, and other content of the music various and unpredictable:

> And of course, in games, even fixed linear recorded music is never played back twice in the same way, in the same order, with the same combination of other sound events. (p. 5)

Many other scholars have aligned themselves with Collins' definition (Kaae, 2017; Whitmore, 2018; Young, 2012).

Collins further divides dynamic audio into adaptive and interactive audio (K. Collins, 2009) depending on whether the triggering source is game-driven or player-driven, respectively. Aside from soundtracks of music game[3], which *are* the gameplay, music in games always adapts to game events (Clark, 2007; Gungormusler et al., 2015; Hoffert, 2007; Naushad, 2013; Somberg, 2019; Szinger, 2017). Adaptive audio represents game-driven musical events that "are cued by the game's engine based on in-game parameters," which is also "unaffected by the player's *direct* actions (K. Collins, 2009, p. 6)." For instance, the playing background music switches between the *village* and *forest* tracks when the player navigates between the two associated locations in the game. Although it's inevitable that player's actions *indirectly* affect game states, such as controlling a game character's location, Collins still considered them adaptive audio because they are "rarely immediately repeatable (p. 6)."

The action of gaming has been viewed as one of the closest types of performance to music performance. As discussed by Kanaga, "the relationship between video games and music 'is not just an analogy' (Kanaga, 2012, p. 2)." He believes that 'games are music, they have a real musical aspect to them. Leaving harmony aside, even the simplest games have a rhythm...'. This close relationship between the act of gaming and playing music allows video games, as an interactive device containing musical materials, to become an instrument. The player becomes a musician who plays music simply by gaming and uses no visible notations, because all musical assets are implemented inside the game engine to be triggered by game algorithms.

While adaptivity has specific meanings in the field of ludomusicology[4], the discussion of interactivity has a long history in music performance, especially computer music and algorithmic composition. These studies and theories help to clarify the interactivity of gaming and shed light on the design of ADM. A taxonomy for interactive music systems by (Rowe, 1993) was built on "a

[3] Because music games as a game genre, it has an explicit purpose and game mechanic that is music-making (Zehnder & Lipscomb, 2006). This paper will focus on applying the ADM system in any game genres except music games.
[4] Ludomusicology refers to the study of game music (Kamp et al., 2016).

combination of three dimensions, whose attributes help identify the musical motivations behind types of input interpretation, and methods of response (p. 6)." In Rowe's first dimension, game music functions as score-driven programs, as the music system of a game has predetermined events and pre-stored music fragments triggered by specific patterns of input from the player. For the second dimension, music in video games is capable of being transformative, generative and sequential. This is due to the fact that real time game data created by the player can be applied to transform the game music by applying signal processing effects to create variants of the music that are not necessarily recognized as related to the player's gaming input. The source materials stored in the music engine imported in the game program can be elementary or fragmentary to generate complete musical patterns in real time based on sets of rules. The source material can also be sequential, where real time gamer input can influence the sequencing of stored music fragments such as ordering, tempo, or meters. In the third dimension, game music sits between *instrument paradigm* and *systems following a player paradigm*. In the former, game performance gestures from the player are analyzed by the computer to generate a solo stream of musical output, while in the latter, the music system also has a "presence with a personality and behavior of its own (p. 8)." For example, the music system follows the player's algorithmic demands but only executes music generations on its own timing that is musical, or it generates types of indeterminate and aleatoric content independently from player's input control.

Dobrian (2004) discussed three elements that make a true interactive work of music by human and computer. Firstly, the computer "can only be purported to be acting autonomously if it is programmed to make some decisions of its own that are not fully predicted by the algorithm. This implies inclusion of some elements of unpredictability: the use of pseudorandomness at some structural level (p. 1)." Secondly, the human player of a truly interactive work should not play from a fixed score, "because the unpredictable behavior of the computer will have no influence on the live performer." Dobrian lastly states that:

> The prefix inter- in the word “interactivity” implies mutual influence between agents that are also in some way autonomous decision makers. Neither agent may be fully predetermined in its behavior; each must be able to modify its behavior—to improvise—based on unpredictable behavior by the other. Therefore, improvisation by both human and computer is an essential component of any truly interactive work. (p. 1)

Playing video games as an action provides venues for embodying all these characteristics of a true interactive work. The fact that video games are computer programs run by algorithms allows any forms of aleatoric procedures that can be programmed to occur. In fact, much game music already involves randomness in its music system, though in relatively primitive forms (e.g., choosing a random soundtrack from a list of songs). Aleatoric capabilities can be expanded to a variety of scenarios for diverse and unique musical effects and purposes, as later discussed in the ADM definition chapter in this paper. Additionally, the players have the agency and freedom of playing the game in the way they prefer while the actions control the game music at the same time. This freedom and agency allow the player to stay flexible to also receive influences from the game and the music system. Players create and send various gaming data to influence the music system, while the game audio, following game changes and states, constantly influences the player, as demonstrated in Figure 2. Both the music system and the player have independence while relying on and influencing each other at the same time.

The way music, game and player work in collaboration during gameplay can be complex and intertwined. It can be challenging to clearly distinguish whether a musical event is, as Collins suggested, “immediately repeatable (K. Collins, 2009).” Thus, based on both ludomusicologists and scholars of computer music and algorithmic compositions, this paper is less concerned about differentiating adaptivity from interactivity, because they are interchangeable in video games when looking from different perspectives. If looking at the relationship between the game engine and the music engine, game music is adaptive as it is implemented as part of the game to be manipulated by the game code. If we consider the game engine as a connection between the player and the music system, game music appears interactive because without a player's control, the game will not run and trigger any music.

Agreeing to the significance of dynamism of game music, music theorist Medina-Gray (2014) studied modularity in game music. She argues that game music "consists of many distinct modules together with rules that govern the modules' possible behavior and combination (p. 1)." An important element of modularity in game music is the "seams" where two music modules are connected. She also discussed two major aesthetic goals of game music that can be achieved at the seams - smoothness and disjunction. Her theoretical analysis and examination of game music inspired my ADM framework in the way that I compose and design connections and transitions between musical modules and the way that I think about where to place the seams in the music, whether between two phrases, two harmonies, or even two notes. This compositional thinking is a change of perspective from the traditional way of composing a piece. In traditional way of composing, notes are placed on a linear musical timeline or trajectory of a musical journey. In contrast, composing nonlinear or mobile structured music connects notes or other musical modules as a collection of nodes that spread in a stereoscopic space of time. In such a space, musical modules can be located and moved between different time points and be connected in various ways. As shown in Figure 3, in the latter way of composing, the composer is then able to connect nodes(notes and musical elements) in all possible directions in time.

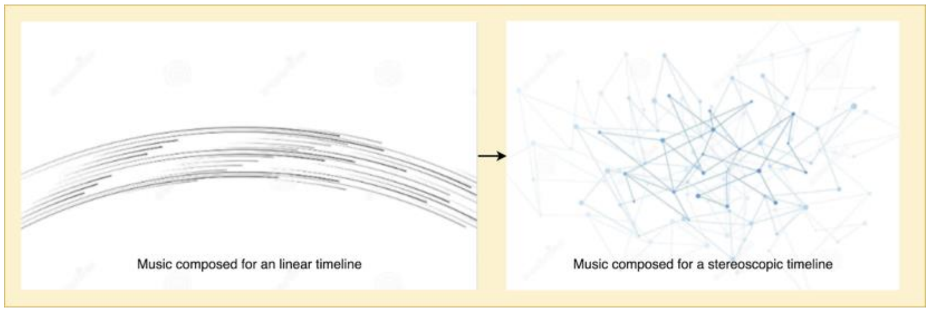

Music composed for an linear timeline
Music composed for a stereoscopic timeline

Figure 3 Two different ways of thinking in composition.

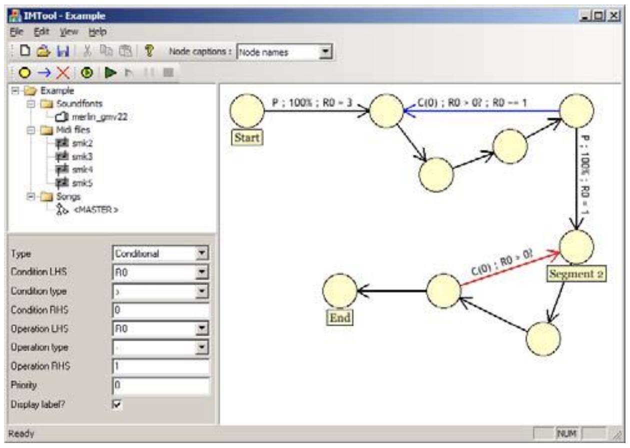
It seems needless to explain why game music is algorithmic because music is triggered by code in real time gameplay. However, algorithmic composition has a more specific and defined definition in contemporary computer music. Algorithmic composition "is the technique of using algorithms, whether computerized or otherwise, to automate the task of musical composition (Chiricota & Gilbert, 2007, p. 182)." Types of algorithmic composition systems can be grouped into three general aims or purposes: 1) creating totally new genres of music; 2) improvising within strictly defined limits, 3) "stitching melody segments together according to a predefined set of rules (p. 182)." Winkler describes algorithms in interactive music systems as "plans (methods) for performing actions, operations, or procedures on musical material or other data (Winkler, 1998, p. 173)." Other scholars such as Wooller et al. (2005) have presented frameworks for classifying different processes in algorithmic music systems. As discussed above, although video games as a medium embrace great stylistic diversity, it is generally preferable to have all the soundtracks of remain in a particular genre within one game so that they express an overall coherence. Thus, in order to have algorithmic composition suitable for a game music scenario, the second and third types of purposes categorized by Chiricota are more applicable. Techniques of algorithmic compositions aiming for the latter two purposes may include but are not limited to mobile form arrangement of pre-composed elements that manipulate musical structures, forms, and harmonic progressions. These arrangements also can create generative melodies, types of accompaniments for a melody, and any other manipulation of applicable musical parameters.

Generativity is closely associated with algorithmic composition, especially when an algorithmic program can "produce output in real time (N. Collins, 2008, p. 238)." In his article, he relates generative music to "G-art (generative art)" in the field of electronic and interactive art. He describes a generative music system as:

> a program allows the creation of a musical variation, typically unravelling in real time, on demand. These systems have the capacity to vary their output with each run, often from no more input

information than the seeding of a random number generator with
the start time. (p. 237)

This definition of generative music does not exclude the involvement of human work. Generative programs are considered "derivative intentionality" of writing (Searle, 2004, p. 20) because a human author has predetermined the code and the way how the programs run in real time. In video games, even though the player's input is constantly involved with the gameplay which directly or indirectly controls the music as well, the music system inside the game engine still has considerable independence in outputting generative materials. Four types of applications of generative music are summarized by Wooller et al. (2005) who has also impacted the way I think about designing and programming generative materials in my ADM projects.



Type	Conditional
Condition LHS	R0
Condition type	>
Condition RHS	0
Operation LHS	R0
Operation type	-
Operation RHS	1
Priority	0
Display label?	✓

Figure 4 The debugger user interface of IMTool. [5]

There are many existing algorithmic music systems that offer applicable algorithmic strategies or design ideas for game music. To name a few examples that have inspired this dissertation: IMTool (Chiricota & Gilbert, 2007) is a framework that hybridizes probabilistic and extended finite state machines to allow the music to be adaptive to the changes in gameplay by providing interactive mobility and variability in the musical structure. In the music engine, composers can define and import MIDI sequences, which are represented as nodes in the system as

[5] Reprinted from Chiricota, Y., & Gilbert, J.-M. (2007). IMTool: An Open Framework for Interactive Music Composition. Proceedings of the 2007 Conference on Future Play, 181–188. https://doi.org/10.1145/1328202.1328235

shown in Figure 4 and design the generation paths between automata so that sections of musical sequences will be played back in a certain order and form musical pieces. The probabilistic finite state machine (PFSM) used in IMTool is equivalent to a first-order Markov process.

Another example is Bach in a Box (McIntyre, 1994), which is an algorithmic compositional tool specifically for Baroque style four-part harmony. It is built with a pitch-labeling system that is defined and established by the creator so that the generated polyphony follows the rules of Baroque style counterpoint. This tool offers valuable insight into the generation of melodic and harmonic material in an algorithmic system which can be implemented into a game music context.

Lastly, Mind Music (Eladhari et al., 2006) is an interactive music system for generating different musical content in response to a game character's mood. Although the manipulation of the ambient sound mats and playing of situational music leitmotifs can be viewed today as a cliche, the dynamic meter aspect of this system provides an interesting and innovative design of algorithmic music that has not been widely adopted in the context of game music. More examples of algorithmic composition and music systems are included in the bibliography.

2.2. Game Music Composition

2.2.1. Techniques and Approaches

Besides the studies of game music's characteristics in gameplay, game composers have also discussed the specific techniques and approaches in composing dynamic music. Among these discussions, most of the techniques that can be categorized as widely known are horizontal and vertical arrangements (Hoffert, 2007; Kaae, 2017; Phillips, 2014; Thomas, 2017). All musical work can be disassembled in horizontal and vertical structures as well. As Kaae defined specifically that "vertical changes in music are … changes in the instrumentation or the amount of simultaneously sounding notes" while horizontal changes in music are the changes that "happen over time (p. 87)." Thinking in these two structures relate to the thinking habits of musicians and the tradition of music notation. For example, when looking at a piece of music on a written score, adding additional voices into the polyphony can be a vertical change, such as when tenor and bass singers start to sing in measure five where only the soprano and alto parts were singing from measure one to four. Changing instruments can be another example, such as the flute takes over the melody which was previously played on the Violin I part. A horizontal change can be one music section switching to another music section, or a dominant chord moving to the tonic chord.

Other techniques and approaches include how to create good loops, as video game music relies heavily on loop-based materials (Thomas, 2017). A good loop should be able to blur the gap between its beginning and end so that the looping sounds seamless. Some specific musical methods to achieve a smooth seam includes inserting a riser-type stinger[6] or short instrumental riff at the end of a loop connecting to a big percussive, impactful down beat to create an illusion of a "rising to an arrival" effect. Composers can also use harmonic progression to blend the loop seams better

[6] Stingers are brief musical phrases that are superimposed and mixed over the currently playing music (Audiokinetic, 2010).

such as making the loop end on a dominant chord in order to form a harmonic resolution when the music loops back to the beginning of the piece which is on the tonic chord.

2.2.2. Methods of Theoretical Analysis

An insightful and inspiring article has summarized dynamic game music with eight types of procedural approaches (Hou, 2020). As the technical specialist, Hou published this article in Audiokinetic Inc's journal[7]. Audiokinetic have published hundreds of articles and videos on game audio design and new technologies in game audio, as well as invited many game composers and sound designers to share their experiences in game audio design for game titles that they have worked on.

This particular article about the taxonomy of dynamic music procedures by Hou discusses dynamism of game music in three dimensions that each consists of a pair of values: whether the driving force of the dynamic material is *user-driven* or *game-driven*; whether the unfolding of music materials is *static* or *dynamic* (which can also be considered as *linearity* and *nonlinearity*); and whether the music dynamism appears in the *remixing* or *transition* (similar to *vertical* or *horizontal* structures). To determine if a music event is user-driven or game-driven, it depends on whether the dynamic music content is impacted by changes in game states that are independent from user input. Linearity and nonlinearity are primarily of concern for the composer or sound designer in terms of whether the generated musical result is predictable or not predictable - such as involving any degrees of indeterminacy - in real time gameplay. The eight categories of dynamic game music procedures are as follow:

1. User-Driven Static Transition
2. User-Driven Static Remixing
3. User-Driven Dynamic Transition

[7] Audiokinetic Inc. is the developer of the game audio middleware Wwise.

4. User-Driven Dynamic Remixing
5. Game-Driven Static Transition
6. Game-Driven Static Remixing
7. Game-Driven Dynamic Transition
8. Game-Driven Dynamic Remixing

This taxonomy can be viewed as an interesting methodology for composers to create dynamic music for video games. They include all patterns and logics of dynamic structure that can possibly appear in video game music. The three musical dimensions are also inspiring for composers at an early stage of their creative process to determine in which dimensions they can mobilize their music to fit specific game scenarios best.

2.3. Game Music VS. Film Music

Many scholars also addressed the discussion of game music in comparison with non-interactive screen music, which is broadly concerned as film music, but are not limited to television and cartoon music as well (Boyd, 2003; Z. N. Whalen, 2004; Z. Whalen, 2007; Clark, 2007; Mundhenke, 2013; Phillips, 2014). There are many similarities as well as dramatic differences in music's functionalities, media-specific characteristics, and the compositional techniques and procedures between game music and film music. We see traces where music of the two media expands on their similarities and learns from their differences. These comparative analyses of music's role in the two media can help and assist game composers to understand and make decisions when composing and creating dynamic music.

These scholars pointed out that certain basics apply to music in both media, such as the concept of diegetic and non-diegetic sound, though this concept usually treats sound effects as part of the game *music*. Other general similarities in music of both media include the functionality of music's expressiveness, such as "add or resolve tension, or create anticipation...evoking the spirit of times long past and places long vanished", for complementing and reinforcing the visuals emotionally (Clark, 2007, p. 1). Whalen argued that video game music has two functions: "to expand the concept of a game's fictional world or to draw the player forward through the sequence of gameplay (Z. N. Whalen, 2004)." The first function also relies on the game genre as some type of game may "have very little in the way of fictional space" such as *Tetris* while film music always accompanies the fictional narratives."

Munday (2007) summarizes two functionalities of music that appear in both films and video games. The first function is to imply a space. Munday presented that film music was considered to have a paradoxical purpose of not catching the audience's attention to the extent that audiences can feel the fictional reality enriched by the music while still focusing on the story or images. He named it "a process of deterritorialization" and he believes similar function exists in video game music in that game music can “block the sound from outside.” When film music fails in this respect, the music makes a *refrain* or signpost effect that occupies and “territorializes” the audience. This functionality relates closely to the second function of music that Munday argued, which is creating immersion. Cognitively, he argues that hearing music prevents the human brain from hunting for other stimuli outside of the game.

If game audio started imitating film audio, it did so with the principal goal of trying to bring film-quality acoustic experiences to the players (Boyd, 2003). Munday (2007) argued that the different tensions between myth and reality in films and games causes music to create mythic immersion differently. He suggested an “aesthetic of cinematic realism” found in film music has influenced video game music significantly. The aesthetic of cinematic realism refers to a powerful effect when music naturally mythologizes the moving pictures, while the pictures realize the mythical part of the music in turn. However, Munday continued to discuss how video game music faces different challenges than film music because the computer-generated images are already mythic and therefore the accompanying music loses its original function of “enriching the images” in films. Boyd also pointed out that game audio has a unique tension between “the desire to minimize repetitive sounds versus the limitations of the delivery mechanism” that film audio does not have. Along with the fast-growing market of video games, however, Boyd also noted a new trend that films have begun adopting the “aesthetic sense of games” and explained with specific examples.

Other scholars also have characterized music’s functionalities in the two media. For instance, music helps to motivate the players, such as the safety/danger binary in game music (Z. N. Whalen, 2004b), as well as to reflect the player’s psychological state (Munday, 2007). Solbach (2004) categorizes film music as having two major functions: affective and cognitive. Lissa (1965) and Koebner (2004) defined five functions of film music: descriptive, affective, structural, expository, and memory-guiding. Grimshaw and Schott (2007) have classified the functions of game soundscapes (which includes sound effects, dialogues, and music) in significant detail such as casual, semantic, and navigational. They also differentiate sounds made by one player and sounds from other players in multi-player games.

Scholars of interactive digital media Bizzocchi and Tanenbaum (2011) have defined three challenges in analyzing video games which also express the medium-specific differences between games and films: indeterminacy, scope, and difficulty. Films as non-interactive media do not contain indeterminacy in the unfolding of their narratives. The narrative content also does not rely on the audience's participation or game-playing skills as in video games. These two features determine that a linear unfolding film has no ability to carry dynamic musical content in its soundtracks because the music is played in sync with the visuals in exactly the same way every time. Without interactive participation from the audiences, films (unlike video games) have no way to show dynamic content in the soundtracks. Additionally, a feature movie can typically last one to three hours in duration, whereas some games may involve hundreds of hours of gameplay. This is why game music uses exclusively loopable structures, because looping music significantly saves storage space. In order to reduce monotony, composers need to seek methods to compose dynamic content to create variations to the looping music that fulfill the long duration of gameplay.

Some scholars and composers also discussed compositional techniques for game music that is able to learn from film music (Phillips, 2014; T. Summers, 2011; Z. Whalen, 2007). Whalen introduced a significant screen scoring technique that can be adopted from cartoon music to games in order to reinforce "the impact of the visuals," termed "mickey mousing." With this technique, the music synchronizes to the narrative through musical gestures, such as an ascending chromatic scale played on the flute accompanying the chasing and escaping footages in the legendary *Tom and Jerry*. Similar techniques discussed by Phillips introduced the use of leitmotifs and idee fixe adapted from symphonic and operatic works by Berlioz and Wagner for use in the *Star Wars* films and game music. Kaae also discussed how game music composition can learn the techniques of creating music passages with chromaticism from film so that it can be quickly lengthened or shortened “without leading to incoherent” content.

2.4. Towards ADM

In game music, it is challenging to clearly define the intensities of musical dynamism. In existing theoretical analyses of game music, studies focus on discussing the dynamism of game music either as a whole or providing a relatively general analysis of this characteristic. Research that concentrates on extracting various patterns, taxonomies, and different levels of intensities of the dynamism are missing from the current scholarly literature. These details about dynamic music in games, in my opinion, are quite important and deserve in-depth exploration and classification, especially for game composers. The following are two simple examples showing how music dynamism may differ in intensities: a game contains simply one looping melody that is triggered when the player clicks "start game" and stops when the player fails. Another game, which is in a pinball style, has a backing music track that triggers a tonal modulation when the player throws a new ball, or a new instrumental riff on top of the ongoing track when the player hits a special object. Here, the musical dynamism in the soundtracks of the two games have a huge difference. The second game is clearly more complex in its dynamism design while the first one has a primitive dynamic process. Although both games can be considered as interactive, dynamic, and algorithmic, these terminologies are too vague and general to describe the differences in intensities of dynamism.

Game composers in industry have experimented with incorporating highly dynamic and interactive music design, such as in *Killer Instinct: Season 3*, *Peggle Blast*, and *No Straight Roads*. In *Killer Instinct* the music switches between the intro, verse, and chorus sections of the song in a horizontal structure, while the instrumentation and texture of the music change in a vertical structure. These changes in the music dynamically respond to the player's control and game states. In *Peggle Blast,* the backing part of the music modulates by a perfect 5th on every ball thrown by the players while each collision of the ball with the blocks in the level also triggers a note and eventually forms a rising scale played on a harp. In *No Straight Roads*, dynamic musical arrangement is triggered by various states of the boss. Although techniques of dynamic music in video games have been explored in some projects, the potential of algorithmic compositional procedures in game music has not yet been widely realized. This potential deserves continuous exploration, explanation, and promotion by composers, game music scholars, and game companies.

Grounded with the research on existing literature of game music, I explore, in my two creative projects - *Do or Die: A Roll Playing Adventure* and *Mastery*, with composing, designing, and integrating a variety of dynamic music procedures that are responsive, procedural, algorithmic, and interactively influenced by the player input and other game data. Through this creative experience, I summarize and theorize eight significant defining properties of a type of highly dynamic game music called ADM.

2.5. Motivation for Developing ADM

My motivation to develop ADM was that I believe ADM has potential to enhance the current practices and experiences of game music. ADM can reduce monotony with its capability of adding large numbers of musical variations to looped music. Music for gameplay (except those for game CGs) is often composed in loop-based structures. When looping music accompanies a long time of gameplay, static soundtracks may quickly become monotonous and bore the players. In a research done by Velardo (2018), about 50% of players who participated in the survey expressed that "looped music is annoying". The modularity, indeterminacy, and multidimensional game-music associations of ADM can allow large numbers of variations to be generated with limited assets.

ADM also has the potential to further optimize the connection between the image and the music with more granular-sized music modules. In films, musical activities and expressions follow the images and sync to the pictures. Lalo Schifrin (2011) described this phenomenon as an image-music counterpoint. Video games can only bring this image-music synchronism through highly flexible and mobilizable music modules due to games' interactive nature. Many existing game soundtracks already use dynamic structure to adapt and respond to a variety of in-game changes, though the modules used are often larger chunks. However, ADM has the potential to accompany the interactive gameplay with much smaller sizes of musical modules, which can be as small as individual notes, measures, or phrase segments.

ADM could allow players to participate in the music. This player involvement can serve as a means of conversation or collaboration with the composer (where players become the music improvisers) as well as a way of expressing the player's individuality. This is because the player actively influences and "co-creates" the music and can hear content specifically triggered by their own gaming habits and patterns.

ADM can give signals or hints about types of in-game information to the player. Because of the various sizes of music modules and their mobility and flexibility in time in ADM, these modules can be associated with in-game events of different types and timings. By connecting a module length to a game parameter, which has similar timing with the music module, ADM can musically inform players of a variety of game events or states.

Chapter 3. Composing ADM - *Do or Die: A Roll Playing Adventure*

3.1. Structural Design

3.1.1. Introduction

Do or Die: A Roll Playing Adventure (*DOD*) is a tabletop, RPG-inspired twin stick shooter game. It consists of three game scenes which commonly appear in many video games on the market: the title scene, the gameplay scene, and a reception scene which displays a gameplay result of a win or a loss. Creating a convincing dynamic game score involves not only technical design of the interactive system but also design of the compositional approaches and principles, as dynamic musical content should meaningfully relate to game content. In *DOD*, I have associated a number of in-game parameters with musical parameters so that the two work in coordination, resulting in musical and responsive dynamism. I have also applied compositional techniques and procedures commonly used in traditional classical music to create dynamic game music content in *DOD*. Table 1 shows the quantity of the types of musical components involved in the game-music interaction of this game project.

Musical Parameter	**Quantity**
Movement	3
Tempo	5
Meter	3
Timbre	34
Rhythmic phrase	59
Real Time Sound Processing Effect	4
TOTAL	*117*

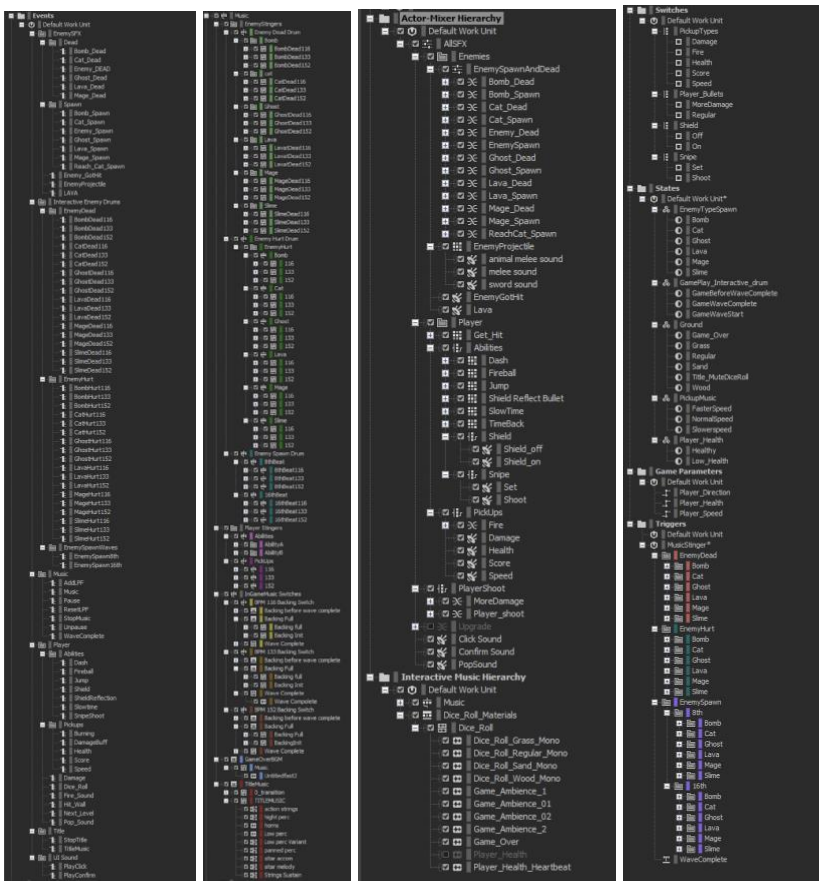
Table 1 Summary of dynamic music elements.

3.1.2. Tools and Software

The tools and software that I used in this project include: *Unity* - a game developing engine, *Microsoft Visual Studio* - a scripting environment, *Wwise* - a game audio middleware, and *Logic Pro X* and *Ableton Live 10* - the DAWs. The tools were chosen based on our team's own preferences; however, similar tools, software, and methods can produce the same outcome.

Wwise is a highly scriptable and flexible audio middleware. It makes audio integration and design systematic and logical, and users can further customize the system with scripts. Wwise comes with built-in functionalities for designing the digital signal processing (DSP), algorithmic procedures, and many other audio editing and programming functionalities that are easy to access for users. Wwise also has a solid API built in its end code that connects to Unity and which users do not need to concern themselves with. Many advanced uses of Wwise can be especially suitable for algorithmic compositions and dynamic and even generative musical content.

For musical materials produced as audio files, I used Logic for interactive segments. This is because the application can export large amounts of audio samples with one click. I used Ableton Live for pitched materials for its time stretching function in audio editing and simplicity of mixing and routing chains of audio effects. Figure 5 is a screenshot showing all the music segments implemented in the ADM system in the Wwise project. Figure 6 shows the music modules created in the Logic Pro project.



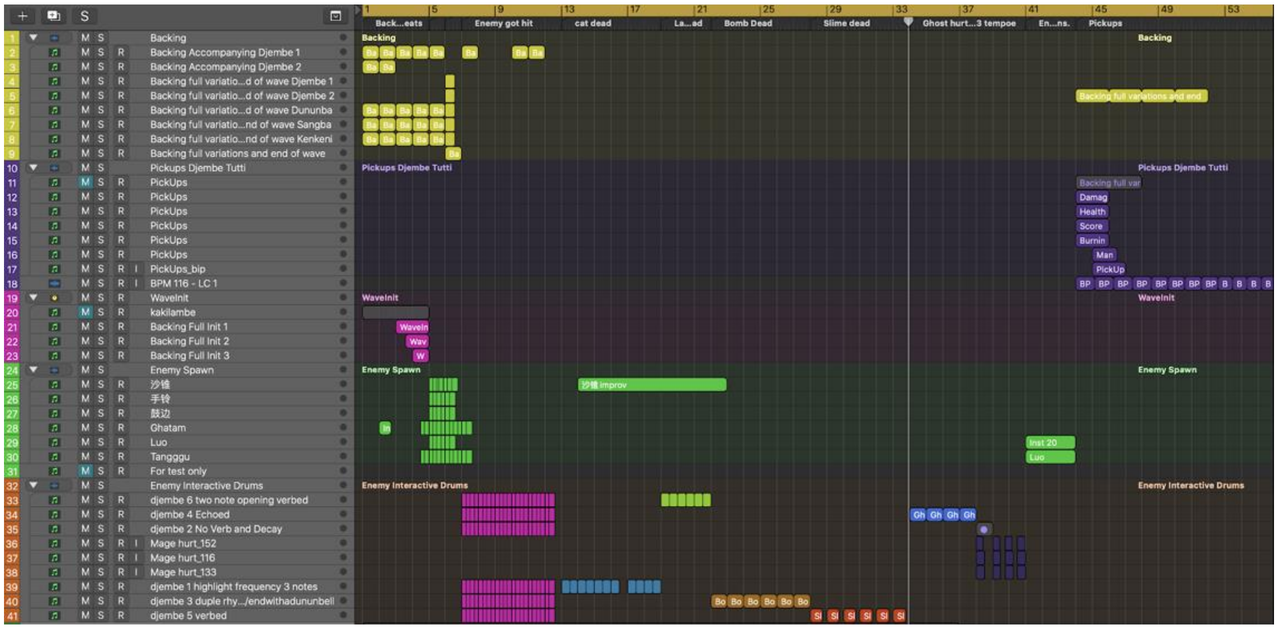
```
Events
  Default Work Unit
    EnemySFX
      Dead
        Bomb_Dead
        Cat_Dead
        Enemy_DEAD
        Ghost_Dead
        Lava_Dead
        Mage_Dead
      Spawn
        Bomb_Spawn
        Cat_Spawn
        Enemy_Spawn
        Ghost_Spawn
        Lava_Spawn
        Mage_Spawn
        Reach_Cat_Spawn
      Enemy_GotHit
      EnemyProjectile
      LAVA
    Interactive Enemy Drums
      EnemyDead
        BombDead116
        BombDead133
        BombDead152
        CatDead116
        CatDead133
        CatDead152
        GhostDead116
        GhostDead133
        GhostDead152
        LavaDead116
        LavaDead133
        LavaDead152
        MageDead116
        MageDead133
        MageDead152
        SlimeDead116
        SlimeDead133
        SlimeDead152
      EnemyHurt
        BombHurt116
        BombHurt133
        BombHurt152
        CatHurt116
        CatHurt133
        CatHurt152
        GhostHurt116
        GhostHurt133
        GhostHurt152
        LavaHurt116
        LavaHurt133
        LavaHurt152
        MageHurt116
        MageHurt133
        MageHurt152
        SlimeHurt116
        SlimeHurt133
        SlimeHurt152
      EnemySpawnWaves
        EnemySpawn8th
        EnemySpawn16th
    Music
      AddLPF
      Music
      Pause
      ResetLPF
      StopMusic
      Unpause
      WaveComplete
    Player
      Abilities
        Dash
        Fireball
        Jump
        Shield
        ShieldReflection
        Slowtime
        SnipeShoot
      Pickups
        Burning
        DamageBuff
        Health
        Score
        Speed
      Damage
      Dice_Roll
      Fire_Sound
      Hit_Wall
      Next_Level
      Pop_Sound
    Title
      StopTitle
      TitleMusic
    UI Sound
      PlayClick
      PlayConfirm
```

```
Music
  EnemyStingers
    Enemy Dead Drum
      Bomb
        BombDead116
        BombDead133
        BombDead152
      cat
        CatDead116
        CatDead133
        CatDead152
      Ghost
        GhostDead116
        GhostDead133
        GhostDead152
      Lava
        LavaDead116
        LavaDead133
        LavaDead152
      Mage
        MageDead116
        MageDead133
        MageDead152
      Slime
        SlimeDead116
        SlimeDead133
        SlimeDead152
    Enemy Hurt Drum
      EnemyHurt
        Bomb
          116
          133
          152
        Cat
          116
          133
          152
        Ghost
          116
          133
          152
        Lava
          116
          133
          152
        Mage
          116
          133
          152
        Slime
          116
          133
          152
    Enemy Spawn Drum
      8thBeat
        8thBeat116
        8thBeat133
        8thBeat152
      16thBeat
        16thBeat116
        16thBeat133
        16thBeat152
  Player Stingers
    Abilities
      AbilityA
      AbilityB
    PickUps
      116
      133
      152
  InGameMusic Switches
    BPM 116 Backing Switch
      Backing before wave complete
      Backing Full
        Backing full
        Backing Init
      Wave Complete
    BPM 133 Backing Switch
      Backing before wave complete
      Backing Full
        Backing full
        Backing Init
      Wave Complete
        Wave Complete
    BPM 152 Backing Switch
      Backing before wave complete
      Backing Full
        Backing Full
        BackingInit
      Wave Complete
  GameOverBGM
    Music
      Untitled[illegible]
  TitleMusic
    0_transition
    TITLEMUSIC
      action strings
      hight perc
      horns
      Low perc
      Low perc Variant
      panned perc
      sitar accom
      sitar melody
      Strings Sustain
```

```
Actor-Mixer Hierarchy
  Default Work Unit
    AllSFX
      Enemies
        EnemySpawnAndDead
          Bomb_Dead
          Bomb_Spawn
          Cat_Dead
          Cat_Spawn
          Enemy_Dead
          EnemySpawn
          Ghost_Dead
          Ghost_Spawn
          Lava_Dead
          Lava_Spawn
          Mage_Dead
          Mage_Spawn
          ReachCat_Spawn
        EnemyProjectile
          animal melee sound
          melee sound
          sword sound
        EnemyGotHit
        Lava
      Player
        Get_Hit
        Abilities
          Dash
          Fireball
          Jump
          Shield Reflect Bullet
          SlowTime
          TimeBack
          Shield
            Shield_off
            Shield_on
          Snipe
            Set
            Shoot
        PickUps
          Fire
          Damage
          Health
          Score
          Speed
      PlayerShoot
        MoreDamage
        Player_shoot
      Upgrade
      Click Sound
      Confirm Sound
      PopSound
Interactive Music Hierarchy
  Default Work Unit
    Music
    Dice_Roll_Materials
      Dice_Roll
        Dice_Roll_Grass_Mono
        Dice_Roll_Regular_Mono
        Dice_Roll_Sand_Mono
        Dice_Roll_Wood_Mono
        Game_Ambience_1
        Game_Ambience_01
        Game_Ambience_02
        Game_Ambience_2
        Game_Over
        Player_Health
        Player_Health_Heartbeat
```

```
Switches
  Default Work Unit
    PickupTypes
      Damage
      Fire
      Health
      Score
      Speed
    Player_Bullets
      MoreDamage
      Regular
    Shield
      Off
      On
    Snipe
      Set
      Shoot
States
  Default Work Unit*
    EnemyTypeSpawn
      Bomb
      Cat
      Ghost
      Lava
      Mage
      Slime
    GamePlay_Interactive_drum
      GameBeforeWaveComplete
      GameWaveComplete
      GameWaveStart
    Ground
      Game_Over
      Grass
      Regular
      Sand
      Title_MuteDiceRoll
      Wood
    PickupMusic
      FasterSpeed
      NormalSpeed
      Slowerspeed
    Player_Health
      Healthy
      Low_Health
Game Parameters
  Default Work Unit
    Player_Direction
    Player_Health
    Player_Speed
Triggers
  Default Work Unit
  MusicStinger*
    EnemyDead
      Bomb
      Cat
      Ghost
      Lava
      Mage
      Slime
    EnemyHurt
      Bomb
      Cat
      Ghost
      Lava
      Mage
      Slime
    EnemySpawn
      8th
        Bomb
        Cat
        Ghost
        Lava
        Mage
        Slime
      16th
        Bomb
        Cat
        Ghost
        Lava
        Mage
        Slime
    WaveComplete
```

Figure 5 Screenshot of all programmed objects in Wwise.



```
1   M S     Backing
2   M S R   Backing Accompanying Djembe 1
3   M S R   Backing Accompanying Djembe 2
4   M S R   Backing full variatio...d of wave Djembe 1
5   M S R   Backing full variatio...d of wave Djembe 2
6   M S R   Backing full variatio...d of wave Dununba
7   M S R   Backing full variatio...nd of wave Sangba
8   M S R   Backing full variatio...nd of wave Kenkeni
9   M S R   Backing full variations and end of wave
10  M S     Pickups Djembe Tutti
11  M S R   PickUps
12  M S R   PickUps
13  M S R   PickUps
14  M S R   PickUps
15  M S R   PickUps
16  M S R   PickUps
17  M S R I PickUps_bip
18  M S R I BPM 116 - LC 1
19  M S R   WaveInit
20  M S R   kakilambe
21  M S R   Backing Full Init 1
22  M S R   Backing Full Init 2
23  M S R   Backing Full Init 3
24  M S     Enemy Spawn
25  M S R   沙锤
26  M S R   手铃
27  M S R   鼓边
28  M S R   Ghatam
29  M S R   Luo
30  M S R   Tangggu
31  M S R   For test only
32  M S     Enemy Interactive Drums
33  M S R   djembe 6 two note opening verbed
34  M S R   djembe 4 Echoed
35  M S R   djembe 2 No Verb and Decay
36  M S R I Mage hurt_152
37  M S R I Mage hurt_116
38  M S R I Mage hurt_133
39  M S R   djembe 1 highlight frequency 3 notes
40  M S R   djembe 3 duple rhy.../endwithadununbell
41  M S R   djembe 5 verbed
```

Figure 6 Screenshot of all interactive music segments in Logic Pro.

3.1.3. Game Triggers and Parameters

The musical parameters work in real time coordination with events happening in the game engine. Game algorithms stay “ready” on a continuing basis until the player clicks “quit game”, which stops the application. The algorithmic functions in the game application can be divided into two general categories: game triggers and game parameters. Game triggers are algorithmic functions that execute only once in the game engine when called at a specific, nonrecurring time. Once called, they are sent to the music system where they trigger their associated musical events in the Wwise engine, such as a different soundtrack. Game parameters are algorithmic functions that update their value at every frame. Thus, they are continuously affecting the music system, for instance changing volume over time. Table 2 and Table 3 present the game triggers and parameters used in the ADM system of this project.

Game Triggers	**Types**	**Quantity**
Level states	4: WaveInit, WaveStart, WaveBeforeEnd, WaveEnds	20 levels
Ground surface	4: Regular, Sand, Grass, Wood	
Enemy events	6: Slime, Cat, Lava, Bomb, Ghost, Mage	3: EnemySpawn, EnemyHurt, EnemyDie
Pick-Ups	6: Damage, Score, SpeedUp, Health, Mana, Burning power	
Player abilities	2: PlayerAbility A, PlayerAbility B	3: Warrior, Ranger, Mage

Table 2 Game triggers.

In *DOD*, I scored three soundtracks that form a multi-movement piece to accompany the three game scenes. Most algorithms and settings developed in a game scene remain in that scene only. This also has an impact on what game parameters can be assigned to interact with the dynamic music system. There are specific algorithms that are shared between scenes, such as the score recorder, the game HUD, and game objects that are purposely held to not to be destroyed when loading the next scene (user selections and settings, audio listener). These connecting algorithms are useful for mapping to musical transitions.

Game Parameters (Updates on every frame)	Player health	Player position
	Player speed	Controller movement
	Enemy count	Collision
	Wave timer	Location of Enemies
	Enemy health	Location of Pick-ups
	Enemy position	…

Table 3 Game parameters.

3.1.4. The Musical Movements

A multi-movement piece, like a classical sonata, a string quartet, or a symphonic piece, has signature characteristics that can provide design patterns for a multi-movement game score. In *DOD*, the title scene gives an introduction to the game and its art style, narrative, and tone. The basic functions of the Title scene include: loading the gameplay that follows, giving the player options to change controls, and a welcoming of the player prior to the gameplay. In this scene, players can remain relaxed since there is no game challenge. A player can wander and explore the title scene as long as they want to, and they retain the agency to leave and execute the next scene when they decide to. The tempo of the *Title Theme* music is relatively slow, stable, and with a calm emotional tone.

Figure 7 Screenshot of the Title scene of *DOD*.

The gameplay mechanics of *DOD* consists of a top-down view shooter in classic high-score seeking arcade style. The game art and design are inspired by traditional tabletop role playing games (RPGs). This is why the playground is in the style of a board, the main character controlled by the player is a polyhedral dice fighting with waves of enemies in different levels. The enemies are modeled like card characters with bases that can stand on a paper map in board games. These enemy game objects are constantly being dropped onto the game board during the gameplay in *DOD*. The player can choose from one of three classes[8]: Mage, Ranger, and Warrior. Each class comes with two abilities controlled by the A and B buttons on the joystick.

Because the game mechanics involves a survival shooter game, it is fast paced, challenging, and full of real time player actions. Thus, a relatively fast tempo and energetic score fits it well. During gameplay, there are many game events that significantly influence game progress. Two game events are complemented especially well by changes in the musical tempo: the SpeedUp buff and the SlowTime ability of the Mage class. The buff and player ability increase and decrease the

[8] A "class" in video games refers to a player character with specific roles and abilities.

player's speed of movement and would thus be well complemented by adaptive musical tempos in the ADM tracks reflective of these changes.

This game currently does not have a victory scene. As an infinite survival game, gameplay is designed to be challenging. However, a *Game Over* state will be triggered when the player loses all health or falls off the board. A Game Over score that sounds spacious or empty can express the feeling of loss. Thus, the movement can be set to a relatively slow tempo.

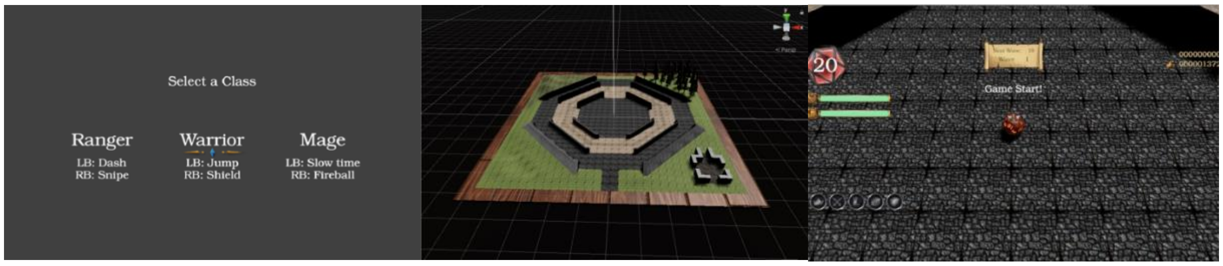


Figure 8 Screenshots of the scene for selecting class, the tabletop RPG style game board, and gameplay view (left to right).

3.1.5. Tempo Design

With some general ideas about the game's music in mind, I deployed a detailed dynamic tempo scheme for the three movements, adopting tempo design practices from classical multi-movement music, specifically the sonata form. In classical sonatas and symphonies, it is very common for the tempo of each movement to be musically related.

In classical music, it was gradually accepted as a norm to write out specific tempo marks after Johann Nepomuk Maelzel's invention of the metronome (Kolisch & Mendel, 1943). Because of this innovation, contemporary composers and performers are able to study the old masters' minds through these numbers. It is widely known among classical music performers that the tempos of movements are related proportionally and musically. It is a crucial ability for a professional performer to find the right tempo quickly when transitioning between movements. I can recall many times that my piano professors have suggested something like this when I have struggled to play at the right tempo: "before you start playing the third movement, think about the tempo you played in the first movement. Then treat the total length of three quarter notes from the first movement as the tempo for one bar in the third movement and play slightly slower than that tempo."

In analyzing the tempo marks of classical repertoires, these tempos relationships become more logical. This is especially apparent to Beethoven's music. In Beethoven's Piano Sonata Op.106 (Figure 9), a half note is marked for BPM 138 in a simple quadruple meter; a dotted half note in the second movement is marked for BPM 80 in 3/4 meter; an eighth note in the third movement is marked for BPM 92 in 6/8 meter; a 16th note in the fourth movement is marked for BPM 76 in 2/2 meter; and in the final movement, a quarter note is marked for BPM 144 in 3/4 meter.

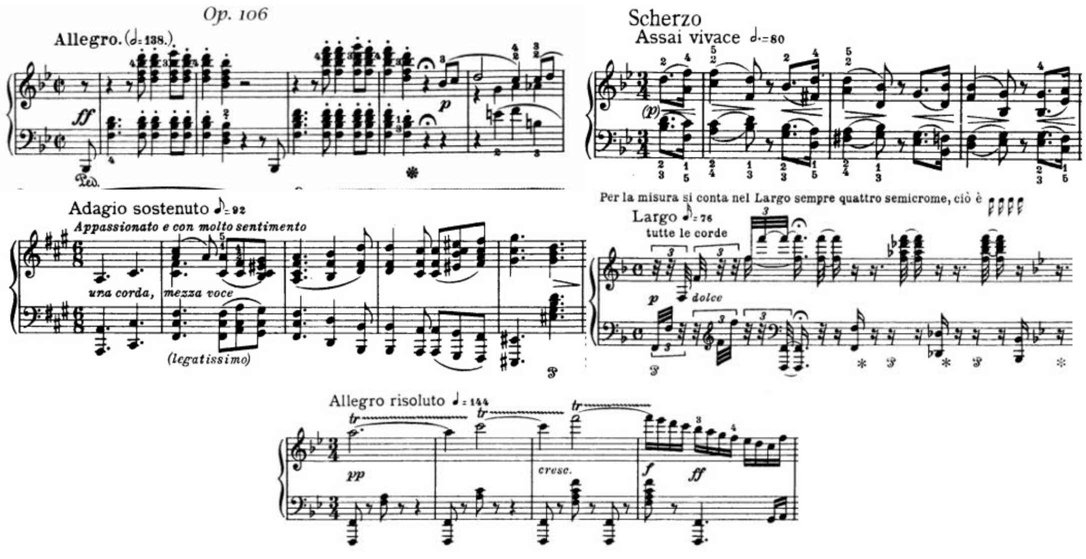


Figure 9 Movements of Beethoven's piano sonata Op.106.

Tempo is of crucial importance in game music composition because the unfolding of music happens simultaneously with the real time game procedure which has its own internal rhythm and tempo. Polyrhythm is a musical device commonly used to express a proportional relationship between two different musical timings. In a piece of music composed in simple meters, a beat can be divided into either two or three parts. When the two types of beat division are included in the same tempo, the total durations of the triplet and the duplet are the same and the speed of each note will form a two-against-three polyrhythm resulting in a tempo ratio of 2:3. For example, in a

4/4 meter, if a duplet eighth note equals BPM 80, a triplet eighth note equals BPM 120, while one quarter note equals BPM 40.

This proportional relationship embodied in polyrhythms can provide interesting patterns for tempo design similar to what exists in many classical music literatures. I developed a simple formula collection which can explain the internal relationship of Beethoven's Op.106: {$t/3$, $t*3$, $t/2$, $t*2$, $t * 3/2$, $t * 2/3$} (t being the original tempo of the piece). I collected the resulting tempos to compare with tempos of the other movements in order to find out if any tempos match or are very close to the tempos calculated from these formulas. I found that duration of three half notes in mvt. I is equivalent to two bars in mvt. II; an eighth note in mvt. III is equivalent to three quarter notes in mvt. I; and the tempo of mvt. V is very close to mvt. I.

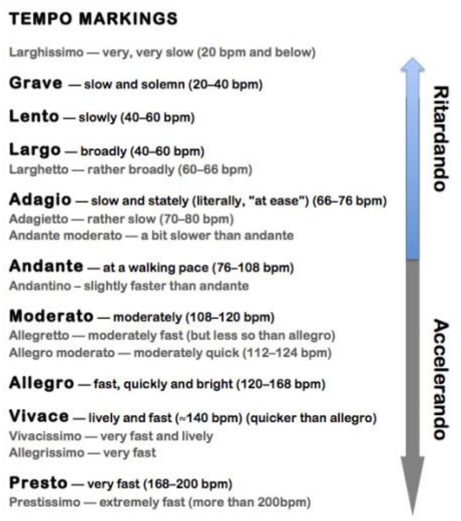
Another formula helps the design or analysis of the tempo relationship in reverse by simply comparing two tempos in a ratio (t_1 / t_2). For instance, in Beethoven's ninth symphony, a quarter note is equal to BPM 88 in the first movement, while a dotted half note is equal to BPM 116 in the second movement. This ratio is exactly 3:4 (88/116 = 0.75) which means that the total duration of three quarter notes in the first movement is equivalent to four dotted half notes in the second movement. Or in other words, four quarter notes in the first movement are equal to three measures in the second movement. Here, the tempo markings by Beethoven seem to indicate that he had an intentional plan for two proportional tempos.

As mentioned previously, a total of five tempos are needed to accommodate the *DOD*'s rhythm and timing: a slow tempo for the *Title Theme*, three tempos for gameplay including NormalSpeed, SpeedUp (caused by the SpeedUp pick-up buff), and SlowTime (PlayerAbilityB), and another slow tempo for Game Over. The initial tempo to be established among the three gameplay tempos is the NormalSpeed tempo and the other two can be established in relation to NormalSpeed. I used BPM 133 for NormalSpeed. This is not only because that this tempo expresses a sense of movement and grooving feeling, but because it also eases the implementation process. In Wwise, the fastest tempo possible is BPM 400. Since the Gameplay score is in 6/4 meter, setting the audio files at BPM 399 allows the grid to be placed on eighth notes.[9]

I made several rules when determining a tempo. First, by testing with a metronome and listening to these soundtracks exported in different tempos, I found that a tempo difference of 15 in BPM is about the minimum tempo difference that a player could potentially notice. A speed difference smaller than that starts to get unnoticeable. There are existing examples of these noticeable tempo differences in traditional music procedure. In classical repertoires, composers use *names* such as *Andante* or *Presto* to refer to tempos. Figure 10 shows that pieces with tempo differences of larger than 20 BPM are named differently because their musical character is substantially distinct. Second, it is better that tempo differences remain stable as much as possible (for instance the difference between NormalSpeed and SlowTime tempo should sound similar to the tempo difference between NormalSpeed and SpeedUp). I used a ±3 in BPM as the measurement for defining an appropriate distance between tempo differences of all movements except the Game Over. Lastly, as summarized above, all ratios between tempos should be taken from musically meaningful patterns such as falling into polyrhythmic patterns of 2:3 (two notes having the same length as three notes), 3:4, 4:5, 5:6, 6:7, 7:8, 8:9 and so forth.[10] This allows the tempo differences between sections or movements to sound natural and musical.

[9] Wwise does have the function to slice music into smaller divisions of beats, such as 8th notes, 16th notes, or even triplet notes, however, this function does not always work in this project. This could be because a BPM of 133 is too fast for the software to locate 16th notes. Setting the tempo at as BPM 399 and snapping the grid to every beat was a successful solution to this problem. Since the SpeedUp tempo of gameplay is an even faster tempo, I set custom cues on every beat on the tracks manually for better beat-tracking and performance. I will explain the implementation process in detail in 4.3.3. Triggering conditions and transitions.

[10] In theory, it does not matter what numbers are on the two sides of the ratio as long as they are integers, but in music tradition, we see six against seven polyrhythms much more often than a five against seven or five against eight and so forth. It wasn't until the modern era of contemporary music that more types of polyrhythms that were previously rarely used appeared in wider application.



TEMPO MARKINGS

Larghissimo — very, very slow (20 bpm and below)

Grave — slow and solemn (20–40 bpm)

Lento — slowly (40–60 bpm)

Largo — broadly (40–60 bpm)
Larghetto — rather broadly (60–66 bpm)

Adagio — slow and stately (literally, "at ease") (66–76 bpm)
Adagietto — rather slow (70–80 bpm)
Andante moderato — a bit slower than andante

Andante — at a walking pace (76–108 bpm)
Andantino – slightly faster than andante

Moderato — moderately (108–120 bpm)
Allegretto — moderately fast (but less so than allegro)
Allegro moderato — moderately quick (112–124 bpm)

Allegro — fast, quickly and bright (120–168 bpm)

Vivace — lively and fast (≈140 bpm) (quicker than allegro)
Vivacissimo — very fast and lively
Allegrissimo — very fast

Presto — very fast (168–200 bpm)
Prestissimo — extremely fast (more than 200bpm)

Figure 10 Tempo names and BPM[11]

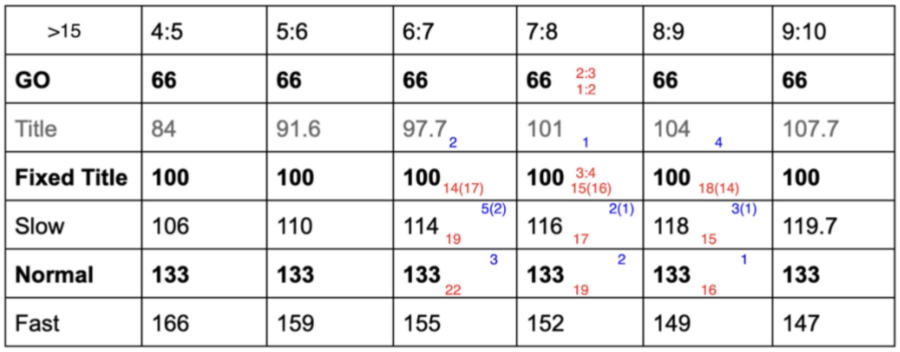
I started by defining the tempo relationship between the three movements (NormalSpeed, Title, and Game Over). I made the Game Over movement BPM 66, which is about one half the Gameplay tempo. Next, similar to Beethoven's Ninth Symphony in which a 3:4 ratio provides a tempo contrast without sounding exceedingly dramatic, I chose BPM 100 (BPM 133 / 4 * 3) for the Title movement (see the "Fixed Title" row in Table 4). While the 3:4 ratio is good for the mood change between Gameplay and the Title scene (which has a tempo difference of 33 in BPM), the three tempo variants of Gameplay provide a noticeable but more subtle difference in feeling. Thus, I designed the tempo difference to about 20 in BPM.

Table 4 shows the six ratios of all tempo differences within the three movements. Black numbers are BPMs. Ratios marked in red numbers show the tempo relationship between Game Over, Title, and Normal[12]. Red numbers inside or outside parentheses are tempo differences between every two adjacent rows. Blue numbers outside parentheses indicate whether the distances between tempo differences follow the ±3 BPM rule. Blue numbers in parentheses show

[11] Reprinted from Tempo Markings—ThingLink | Music theory lessons, Piano teaching, Homeschool music. (n.d.). Retrieved April 6, 2021, from https://www.pinterest.com/pin/424042121160098923/
[12] NormalSpeed.

the differences between the Title row and Slow[13] row. The results of ratios from 4:5 to 9:10 are presented here. It is evident that below the ratio of 6:7 or beyond the ratio of 8:9, the resulting tempo differences become either in excess of 20 in BPM or smaller than 15 in BPM, which neither is desirable. Among the remaining three ratios (6:7, 7:8, 8:9), 7:8 is the best suited ratio. For instance, under the 6:7 column, Fixed Title and Slow has a difference of BPM 14, Slow and Normal has a difference of BPM 19, and Normal and Fast[14] has a difference of BPM 22. Between 19 and 22, there is a difference of 3, which is within the ±3 range, thus the two tempo changes have a good balance between each other. However, because between 14 and 19 the difference exceeds 3, the Fixed Title tempo isn't appropriate to maintain evenly spread distances between all tempo changes. Therefore, the value of Title is better than Fixed Title because there between BPM 97.7 to BPM114 has a difference of 17, resulting in a distance to the adjacent tempo difference of 2, within the set range of ±3. The 7:8 ratio is even more appropriate than the 6:7 ratio because distance between all the tempo differences (15 BPM, 16 BPM, 17 BPM and 19 BPM) is less than 2 BPM, which is within the desired ±3 BPM range tempo. Using this 7:8 ratio, I got BPM 101 (BPM 116 /8 * 7) for the Title (see the Title row in gray in Table 4), which is also the closest value to the Fixed Title tempo which was the initial compositional tempo I used for the movement. Thus, the ratio of 7:8 for the Title and three gameplay tempos is a good complement to the ratio of 3:4 for between the GO[15] and Fixed Title, 2:3 between GO and Title, and 1:2 between GO and Normal. All tempos are therefore related in musical, polyrhythmic ways, as demonstrated in Table 5.

[13] SlowTime.
[14] SpeedUp.
[15] Game Over.



>15	4:5	5:6	6:7	7:8	8:9	9:10
GO	**66**	**66**	**66**	**66** 2:3 1:2	**66**	**66**
Title	84	91.6	97.7 2	101 1	104 4	107.7
Fixed Title	**100**	**100**	**100** 14(17)	**100** 3:4 15(16)	**100** 18(14)	**100**
Slow	106	110	114 19 5(2)	116 17 2(1)	118 15 3(1)	119.7
Normal	**133**	**133**	**133** 22 3	**133** 19 2	**133** 16 1	**133**
Fast	166	159	155	152	149	147

Table 4 Tempo design chart.

Movement/Section	**BPM**	**Relationship**	**Ratio to NormalSpeed**
Title	100	one beat equals to one dotted half note of Normal speed Gameplay	3:4
Gameplay (NormalSpeed)	133		1:1
Gameplay (Speedup)	152	octuplet equals seven eighth notes in NormalSpeed	8:7
Gameplay (Slowtime)	116	8 eighth notes equal to 7 eighth	7:8
Gameover	66.5	twice slower than normal tempo gameplay	1:2 (2:3 to Title)

Table 5 All tempo ratio relationship.

3.1.6. Meter Design

In Classical forms of multi-movement pieces, it is also common to switch meters between movements (Cole, 1974). I adopted this technique to design *DOD*'s dynamic metric plan. The Title movement is in compound quadruple meter (12/8), which divides twelve eighth notes into four groups of three. The music strides at a moderate tempo with a festive meter. The Gameplay movement is in a compound duple meter (6/4) which is a livelier, faster tempo. For the Game Over movement, I used a simple quadruple meter under which each beat is divided into two. Beats in this meter are spread out in time and the emotion of the music recalls a march-like feeling which can be suitable for a "game over" feeling.

Besides changing the music from one tempo to another in a vertical structure, the Gameplay movement also involves different musical sections in a horizontal structure. In each level, this movement always begins in 6/4. However, the meter switches to compound quadruple (4/4) in WaveBeforeEnd, one of the horizontal sections in the Gameplay movement. The design of the horizontal sections will be discussed in detail in 3.3. Musical Sequences and Sections. This additional meter provides a unique rhythm which signals to the player that they are approaching the conclusion of a level. The meter change does not affect the tempo, but simply divides the twelve beat grids into different groupings. The 6/4 meter can be considered a 12/8 meter in which each quarter note is divided into two eighth notes. Alternatively, the 4/4 meter can also be considered 12/8 meter if each quarter note is further divided into three eighth notes. The two patterns accent different beats of the twelve: beat one, four, seven, and ten for 4/4, and beat one, three, five, seven, nine, and eleven for 6/4. This change of accented beats brings a noticeable rhythmic contrast to the Gameplay movement. The overall dynamic metric design of *DOD* is shown below:[16]

- Title: compound quadruple, 12/8(4/4) - 3, 3, 3, 3
- Gameplay initiation: compound duple, 6/4(12/8) - 2, 2, 2, 2, 2, 2
- Gameplay's *WaveBeforeEnd* section: compound quadruple, 12/8(4/4) - 3, 3, 3, 3
- Game Over: simple quadruple, 4/4 - 2, 2, 2, 2

[16] Following each dash are the quantity of beats grouped together as a rhythmic unit.

3.2. The Musical Assets: Instrumentation, Timbre, and Audio Samples

3.2.1. The Title Scene

The Title movement is a pitch-based piece played by multiple virtual instruments as displayed in Figure 11. Instead of being one individual audio file, instrumental layers are exported as stems, dynamically mixed by Wwise and played in real time gameplay. The instrumental layers are:

1. Reverbed cymbal samples from Drum Rack
2. Echoing samples from Drum Rack
3. *West Africa Kakilambe* ensemble *(Dununba, Sangba, Kenkeni,* two *Djembes)*
4. *West Africa* solo *Djembe*
5. *Bansuri*
6. *Sitar* melody
7. *Sitar* accompaniment
8. Low strings sustaining notes
9. High strings rhythmic phrases
10. Horns

The first two layers are percussive phrases composed for a drum rack instrument in Ableton Live. The drum rack contains six processed percussion samples. Following the drum rack are two layers of sampler instruments called *Kontakt,* produced by Native Instruments. This sampler instrument is capable of loading and processing large audio samples with impressive quality and latency control. The virtual instrument I used here is the *Kakilambe* preset from the library *West Africa*. *Kakilambe* is a traditional African ritual dance and rhythm. The ensemble consists of three cylindrical drums that each include a bell: the *Dununba*, the *Sangba*, and the *Kenkeni*. There are also three goblet drums: two ensemble *Djembes* and one solo *Djembe*. I placed the two ensemble *Djembes* on the third layer and the solo *Djembe* on the fourth layer. The next three layers are *Bansuri* and two *Sitar*s from the *India* library. Lower section strings are from the library *Session String Pro 2*, high strings are *Action Strings*. Lastly, the horns are from the *Cinematic Brass* library. A video demonstrating the Title movement as well as the sound of each layer in the Ableton Live project can be found in a video playlist via the link provided in Appendix D.

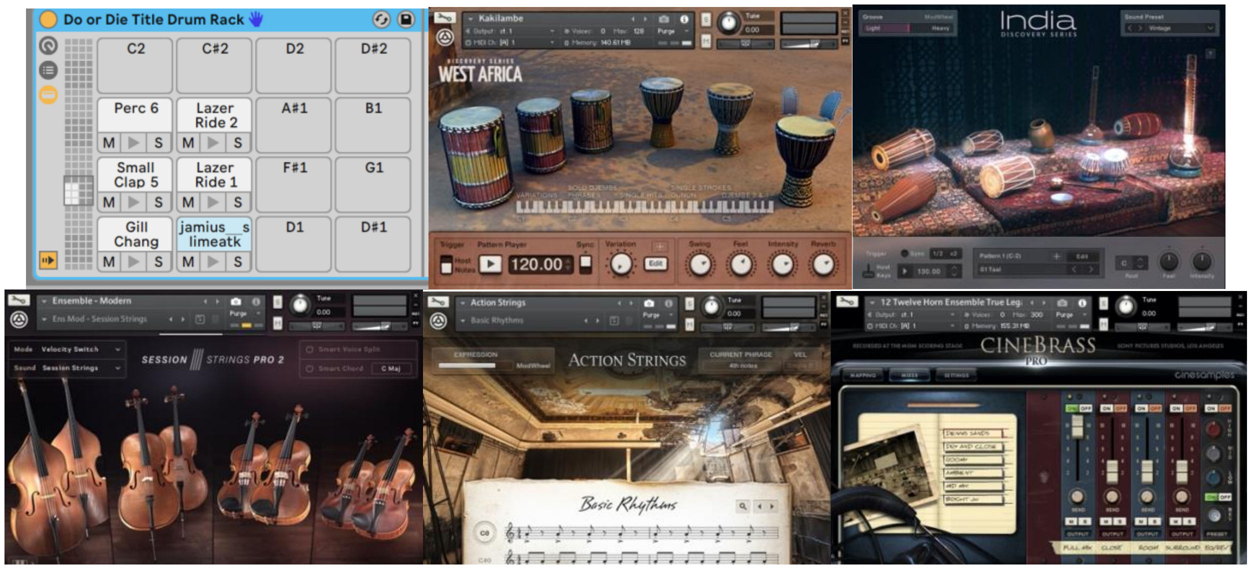


Figure 11 Screenshot of all instrumental layers in the Title scene.

3.2.2. The Gameplay Scene

3.2.2.1. A Percussion Piece

In the Gameplay movement, I used solely non-pitched timbres to create a percussion piece that is interactively dynamic to gameplay. This is partially because pitched elements like harmonies and melodies have many restrictions when applied to a mobile form. Harmonies and melodic motifs are very dependent on the larger tonal context--where a pitch comes from, where it moves to, and how pitches are organized. From a more conventional perspective to the average audience of video games, the arrangement of pitches is the key to differentiating music from sound. In this project I hope to design ADM with a concentration on the arrangement of musical timings instead of the manipulation of pitched material.

I hope to use this project to demonstrate a foundational infrastructure of ADM audible to composers, sound designers, and game developers, so that each may apply this infrastructure to their own projects with accurate expectations of the resulting sound effects. The next game project discussed in Chapter 5 focuses on how ADM structure is applied to pitch-related musical content, which would be a useful comparison with this chapter that mostly focuses on rhythmic content.

A fast percussion piece could suit *DOD* well because non-melodic material is potentially less attention grabbing than melodic material, and a percussive score may complement the rhythm of gameplay well. With highly interactive and generative content occurring in real time during the Gameplay movement, there is a fair amount of auditory material for the player to follow.

3.2.2.2. The Two Groups in a Vertical Structure

This movement has two groups of music materials: the *backing group*, and the *generative-improvisatory drum group (GID group)*. The *backing group* is the foundational layer of the piece that consists of several short musical segments combined in dynamic sequence. It is played by the same Kontakt virtual instrument library *West Africa* used in the Title movement but plays different rhythmic patterns. In *West Africa*, each type of cylindrical drum (the *Dununba*, *Sangba*, and *Kenkeni*) has four different drum sound samples and three bell sound samples. The drum and bell sound can be customized with tuning and dynamics, and users can also add swing to the pattern. Three *Djembes* (two playing in the ensemble part and one playing as a solo instrument) come seven selectable samples with customizable tuning and rhythmic swing. They each also come with a supplemental tapping sound whose volume and feeling users can manipulate. I used all ensemble drums for the *backing group* except the solo *Djembe* which I left for the *GID group*. Table 6 shows all timbre customization in *West Africa*.

DRUM TYPE	**SAMPLE SELECTION**	**OTHER CUSTOMIZATION**
Dununba with bell	Sample 1, bell sample 3	Tuned to C, volume at 50%
Sangba with bell	Sample 3, bell sample 2	Tuned up, volume at 40%
Kenkeni with bell	Sample 4, bell sample 3	Tuned up, volume at 60%, adding swing
Ensemble *Djembe* 1	Sample 2	
Ensemble *Djembe* 2	Sample 7	Tuned down

Table 6 The types of drum, sample selection, and other customization in the *backing group*.

The *GID group* is a combination of percussive sounds and rhythmic phrases that either interactively respond to real time, game-driven or user-driven data to form larger generative musical content. This group is played in sync with the *backing group* and has two sub-groups: *enemy events* (game-driven), and *player activities* (user-driven). *Enemy events* include three events: the spawning of an enemy (EnemySpawn), an enemy being hit (EnemyHurt), and the death of an enemy (EnemyDie). These events may occur to a total of six enemy types: Slime, Cat, Lava, Bomb, Ghost, and Mage. Therefore, there are a total of eighteen events where each enemy type has a unique set of percussion sounds and phrases assigned to each of the three events. The rhythmic phrases and samples also have variations such as slightly different rhythms, tone, or swing.

Player activities include acquiring six types of buff and using the two class-specific abilities. The six types of buff are: DamageUp, HealthUp, ScoreUp, ManaUp, BurningPower, and SpeedUp (which causes the *backing group* change to BPM 152 as discussed previously). Acquiring each type of PickUp buff triggers a unique drum phrase. Each of the two class-specific abilities are triggered by the left and right buttons on the game controller. I composed two drum phrases for the two buttons and all three player classes share the same phrases assigned to the two buttons. I labeled the two phrases *PlayerAbility A* and *PlayerAbility B*. Thus, there are a total of eight different percussion phrases in this group. In conclusion, the *GID group* needs a total of 26 unique percussion sounds and phrases.

Enemy Types/Events	**EnemySpawn**		**EnemyHurt**	**EnemyDie**
Slime	*Ghatam*	Ensemble claps	Solo *Djembe* sample 5, short-decay reverb, HSF	
Cat	Hand bell		Solo *Djembe* sample 1, EQ-ed 1.6k Hz	
Lava	Drum rim shot		Solo *Djembe* sample 6, short-decay reverb	
Bomb	Sand hammer		Solo *Djembe* sample 3, echo, HPF	
Ghost	*Luo*		Solo *Djembe* sample 4, echo	
Mage	*Tanggu*		Solo *Djembe* sample 2, dry source (no reverb)	

Table 7 Enemy Events Design Chart.

Samples chosen for the EnemySpawn events are significantly different from samples for the *backing group* (shown in Table 7). I used the Ghatam, an Indian instrument for Slime, hand bells for Cat, drum rim shot for Lava, the sand hammer for Bomb, the Chinese instrument *Luo* for Ghost, and the Chinese instrument *Tanggu* for Mage. These samples come from several virtual instrument libraries: *India*, *ShimmerShakeStrike*, *Studio Drummer*, and *Traditional Chinese Instrument* libraries (see Figure 12). These EnemySpawn samples are triggered either on the downbeat or the eighth note upbeat.[17] This will be discussed in 3.4.3. Generative Patterns, Timer, and Probabilities.

There are sample variations for EnemySpawn events. They differ subtly in their tone and rhythmic swing. These variations provide the virtual instrument played by the computer a more humanized and lively sound. The Slime, Bomb, and Mage enemy types each have eight sample variations, while the Cat, Ghost and Lava types each have four.

[17] In 6/8 meter.

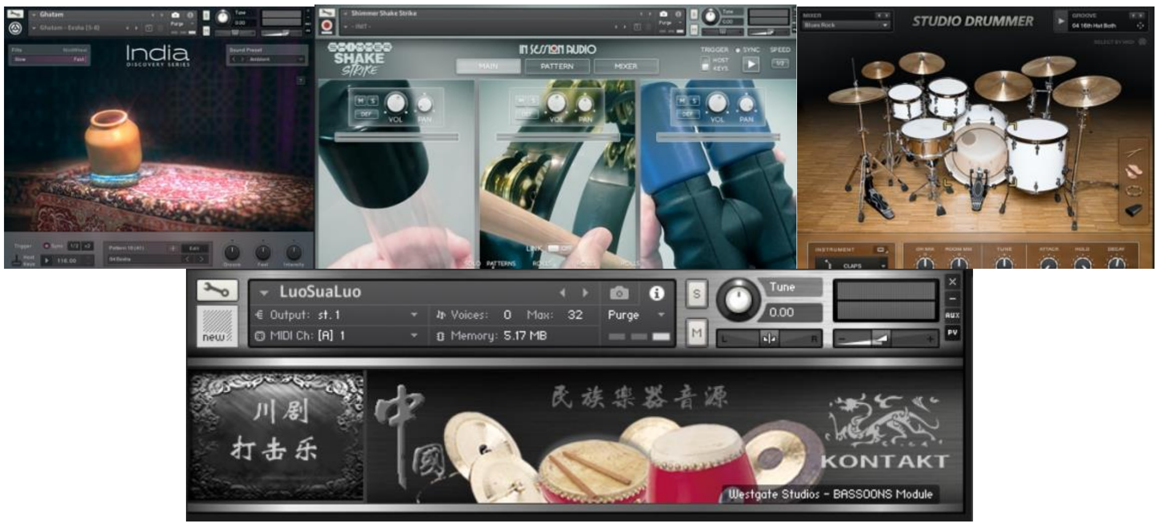


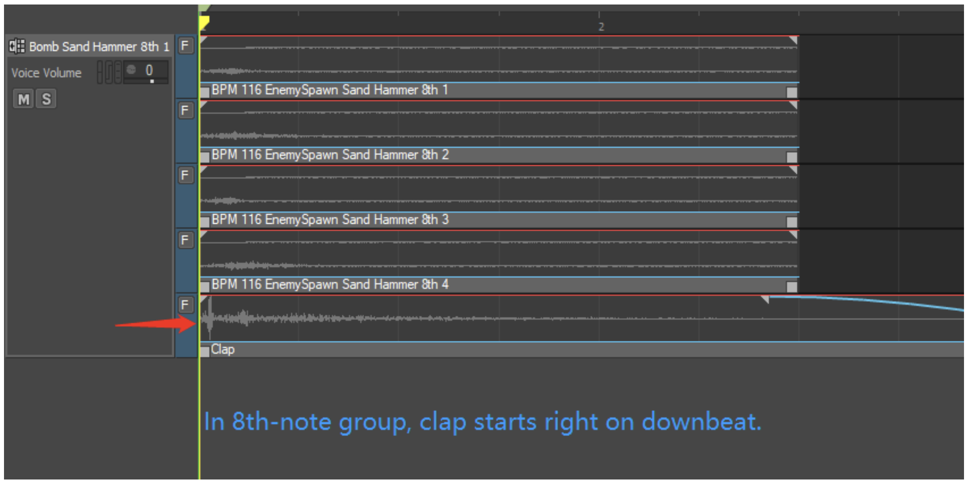
Figure 12 Kontakt virtual instrument libraries used for the *GID group*.

All enemy types share one additional "ensemble clapping" sample which may be triggered as one of the sample variations and which has a fixed probability of playing (Figure 13) during an EnemySpawn event. The purpose of adding this additional shared sample was to avoid undesirable repetition.[18] The universal clapping sample brings contrast to the enemy type-specific sample. It also functions as a musical rest during which the ensemble drums do not play.

This technique of combining rare and common elements functions to enliven the composition. This compositional principle is observable in many existing repertoires. For example, individual instruments in an instrumental choir may use a similar tone and timbre while playing different parts, or different instruments having dramatically different timbres may play in unison.[19] The same clapping sample at times unites all enemy types sonically while at other times, while employing their respective samples, each type sounds distinct. Because enemy types are unlocked gradually through levels, the player also gradually unlocks new sounds associated with each enemy

[18] In many levels, the enemies spawned are of the same type. If there were only one sound assigned to the enemy type, the repeated sound could potentially become monotonous.
[19] This is an application of the ancient artistic principle "unity in variety" to music composition. Unity is wholeness and harmony which unites elements while variety contrasts and provides diversity within a work (Lamp, 2020). The relationship of unity and variety is bi-directional and parallel in that they can embody each other.

type. The collections of percussive samples played at each level thus create an adaptive percussive pattern which signals to the player which enemies appear in the level.


Bomb Sand Hammer 8th 1
Voice Volume
0
M
S
F
BPM 116 EnemySpawn Sand Hammer 8th 1
BPM 116 EnemySpawn Sand Hammer 8th 2
BPM 116 EnemySpawn Sand Hammer 8th 3
BPM 116 EnemySpawn Sand Hammer 8th 4
Clap
In 8th-note group, clap starts right on downbeat.
Figure 13 Ensemble clap sample.

For the EnemyHurt and EnemyDie events, I used the solo *Djembe* from *West Africa* with a different signal processing path. In this fashion, another application of “unity and variety” is here employed. The EnemySpawn sounds are here the element that brings “variety” while the solo *Djembe* used for EnemyHurt and EnemyDie events is the element that contributes to “unity”, because its tone blends well with the *backing group*.

In EnemyHurt and EnemyDie events I used sample six of the solo *Djembe* with a custom reverb and a short decay of 538 ms for the Lava enemy type, sample four with echoes for the Ghost, sample two without reverb for the Mage, sample one with an EQ highlighting 1.6kHz for the Cat, echoed and HPFed (high pass filtered) sample three for the Bomb, and sample five with 693 ms decay reverb with a subtle boost at 3.1kHz with an HSF (high shelf filter). EnemyHurt events trigger the playing of one-beat phrases while EnemyDie events trigger the playing of short rhythmic phrases. I will discuss EnemyDie phrase patterns in 3.4.2. Phrases and Patterns in the Gameplay.

Table 8 discusses specifically the *player activities* category. The six types of pick-up buff are each assigned a drum phrase played by the *Djembe* choir (two ensemble and one solo) and bells on the cylindrical drums. The class-specific abilities controlled by the two buttons on the joystick are assigned phrases played by the drum rack instrument that I created for the Title movement (refer to Figure 11).

Player Activities	Pick-up buffs	Abilities
Timbre assigned	3 *Djembes* + bells	Drum Rack in Title movement

Table 8 Player activities sounds.

3.2.3. The Gameover Scene

As discussed in 3.1.4. The Music Movements, the emotional feeling of the Game Over movement is less energetic and eventful than other movements. I reduced the instrumental layers in this short movement to create more empty space between musical timbres as compared to the Title and Gameplay movements. It was produced with the same Ableton Live project preset, and I used only the sustaining low strings, the *West Africa* drum ensemble, and the drum rack. This movement is exported as a straight looping track with no dynamic content.

3.3. Music Sequences and Sections

When considering musical *sequences*, the horizontal arrangement generated by combining musical sections together within movements and the triggering conditions and transitions between movements, sections, and generative materials during the Gameplay movement must be discussed. Topics in these discussions include ordering and playback behaviors.

In order to save disk space as well as to provide an optimized game audio experience, it is necessary and desirable to recycle musical materials. This recycling can be achieved by looping, shuffling, changing orders, delaying, selecting materials from a list of options with different probabilities of playing, and by creating combinations to expand smaller amounts of material to fill indefinite gameplay with variety. Each of the three movements of *DOD*'s game score uses a variation of looping structure and provides an audible demonstration of how these structures differ.

3.3.1. Title and Game Over Movements: Dynamic Loops and Static Loops

The Game Over movement has the simplest looping structure, a **static loop**, a structure common in many video games. In a static loop of the Game Over scene, stems or layers of the music are exported into one track which loops indefinitely in the game as long as the player remains in the scene (as shown in Figure 14). The mixing of the Game Over track remains identical with each repetition, thus it is a static loop.

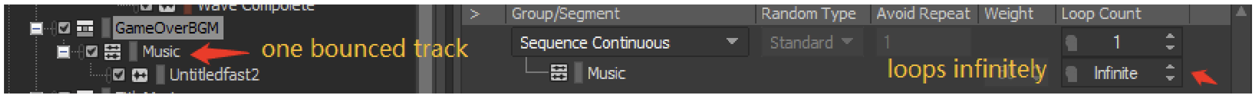


Figure 14 Game Over loop.

The Title movement, by comparison, is a **dynamic loop**. The ten stems or layers of instruments are randomly selected and dynamically mixed in Wwise in real time during gameplay. Figure 15 and Figure 16 show the Title movement broken down into individual stems in Wwise. There are eleven stems (the *Sitar* melody and action strings lines also have an additional layer) in nine layers of instruments. Instrumental layers were described in 3.2.1. The Title Scene. Each stem has a volume value, followed by blue headers in front of its sub-tracks (Figure 16). The blue headers indicate that sub-tracks of this stem will be randomly selected to play. For instance, the *Strings Sustain* stem has two sub-tracks with a blue header, thus the system will randomly choose one of the two sub-tracks to play when the game engine calls Wwise to play this mix. This also means that sub-tracks on the same stem will never be played simultaneously. Because of this, I put the *Sitar melody* and the *Bansuri* into one stem so that they are separated in the mix. If they are played together, the most important melody becomes confusing. Some stems have empty sub-tracks such as the *action strings* layer or the *Sitar accom* layer*.* This means that there is a 50% chance this stem will play silence and is a way to omit a track in the mix based on a determined probability.

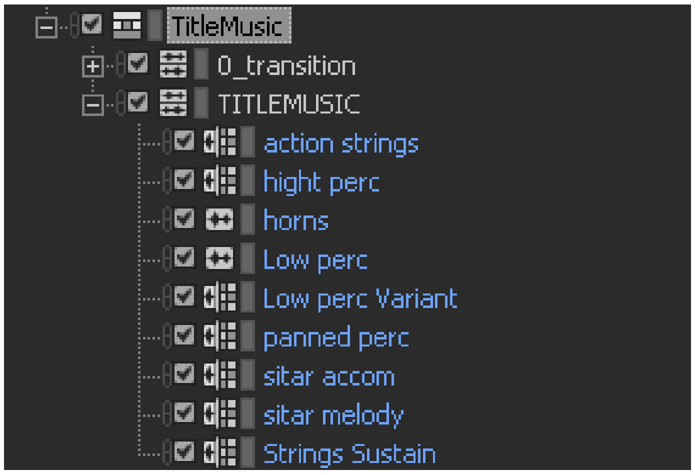


Figure 15 Title movement stems overview.

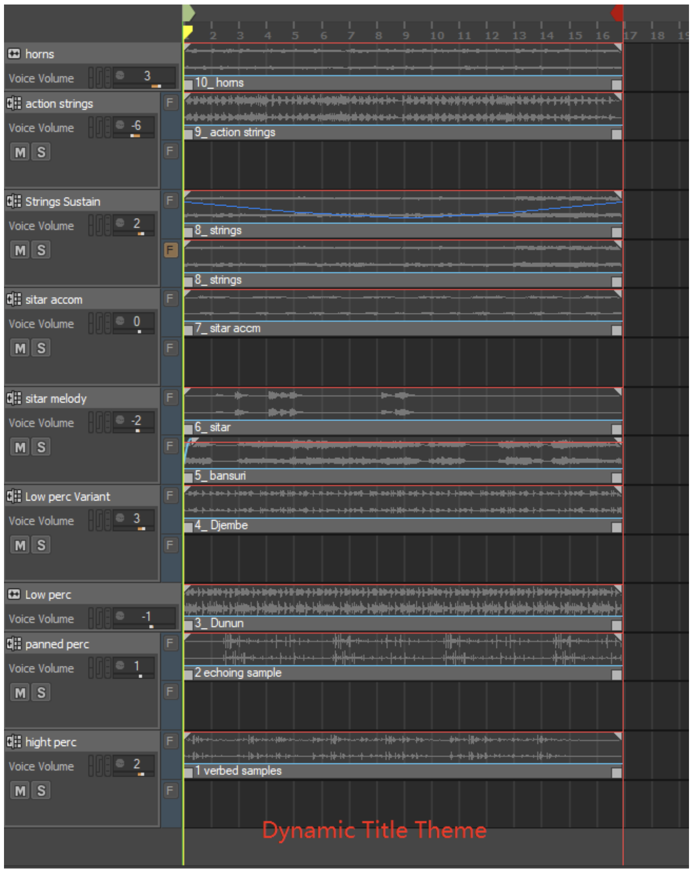
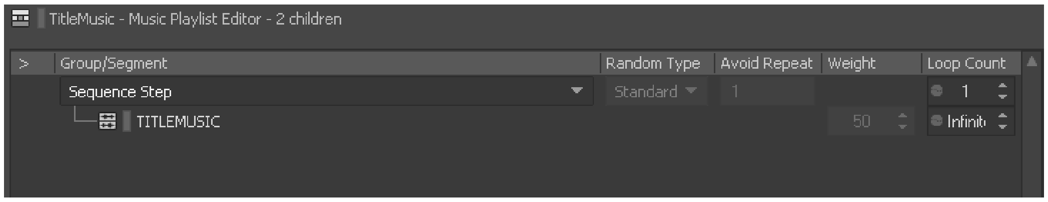


Figure 16 Dynamic stems of Title movement.

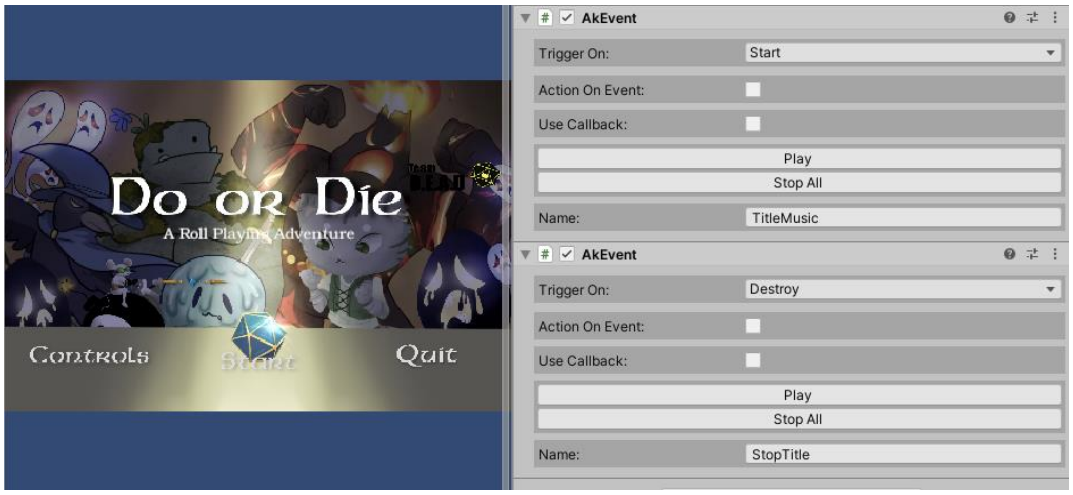
The entire Title mix is placed in a Music Playlist object in Wwise as shown in Figure 17. It loops indefinitely as long as the soundtrack is being triggered. Each time the Title mix loops, a new mixing will be generated by the Wwise engine, with the result that the combination of stems varies each time. The *horns* and *Low perc* tracks in Figure 16 do not contain random sub-tracks, thus they are heard in every loop. As a result, we have a dynamic loop that may play the *Sitar melody*, *panned perc*, *horns* and the *Low perc* on the first play of the loop, but only the *horns* and *Low perc* at the second play of the loop.


TitleMusic - Music Playlist Editor - 2 children

Group/Segment	Random Type	Avoid Repeat	Weight	Loop Count
Sequence Step	Standard	1		1
TITLEMUSIC			50	Infinit

Figure 17 Title movement loops indefinitely.

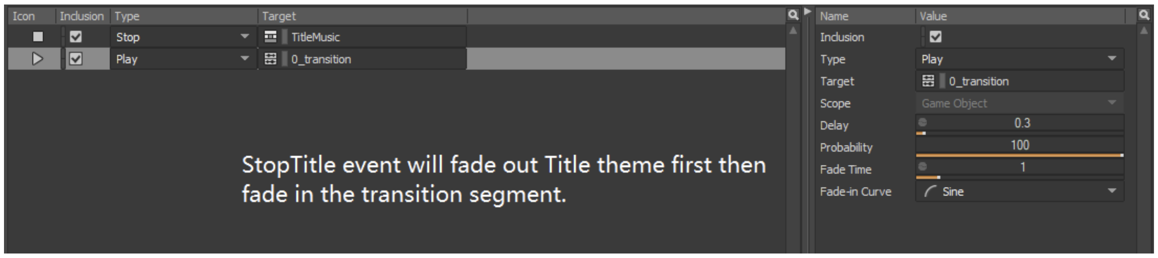
Figure 18 shows that the Wwise event object TitleMusic will be triggered when executing the "start" function (start()) in the game code which is the first method initiated after loading a game scene.[20] Another event called StopTitle will be initiated when the game scene is dissolved during transition to the gameplay scene. As long as the player stays in the Title scene, the Title mix will continue looping and varying its mix. As shown in Figure 19, upon initiating the StopTitle event, Wwise will first fade out the Title movement mix, then trigger a brief one-beat transitional chord played on the horns. In order to avoid the transitional passage overlapping with the beginning of the Gameplay movement, I programmed a one second delay to the initiation of the Gameplay movement (see Figure 20). This delay allows the transition to finish before the next movement starts to play.



AkEvent
Trigger On: Start
Action On Event:
Use Callback:
Name: TitleMusic

AkEvent
Trigger On: Destroy
Action On Event:
Use Callback:
Name: StopTitle

Figure 18 Game engine triggers.



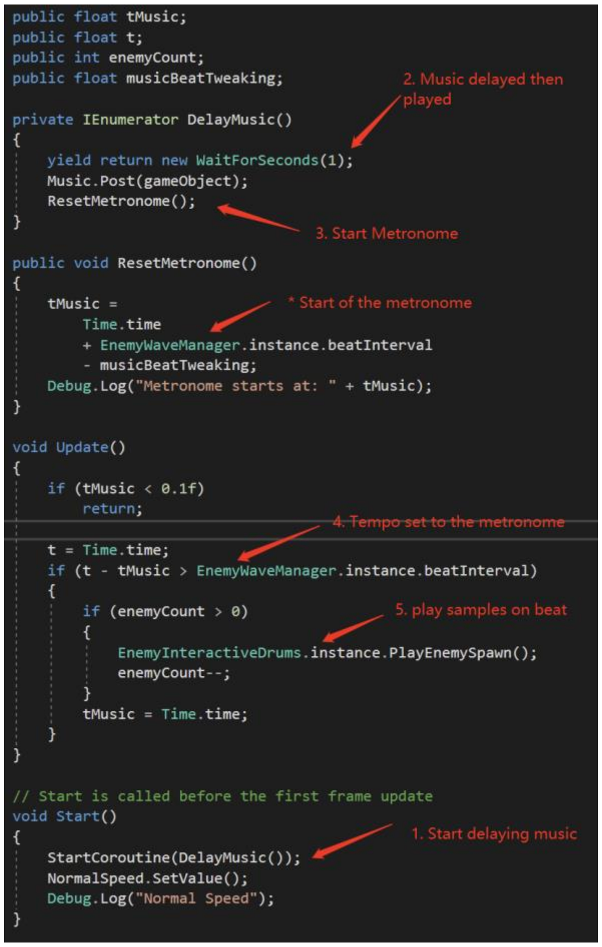
Icon	Inclusion	Type	Target
■	✓	Stop	TitleMusic
▷	✓	Play	0_transition

Name	Value
Inclusion	✓
Type	Play
Target	0_transition
Scope	Game Object
Delay	0.3
Probability	100
Fade Time	1
Fade-in Curve	Sine

StopTitle event will fade out Title theme first then fade in the transition segment.

Figure 19 The StopTitle event.

[20] In Unity, before the execution of the Start() method, there is the Awake() method if the designer wants to execute something specific while the game scene is loading.

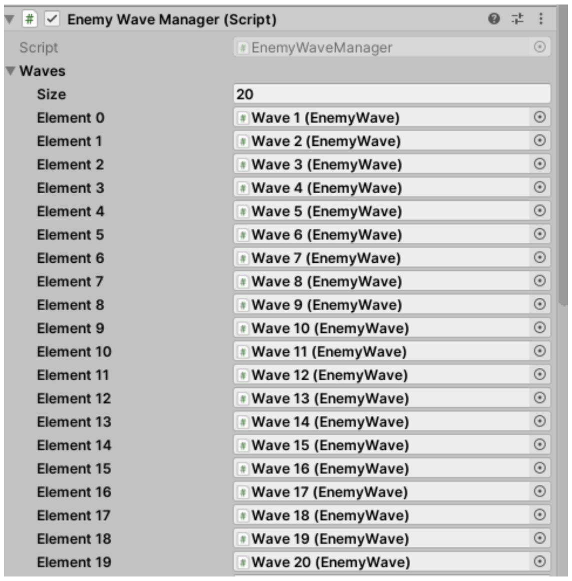


```
public float tMusic;
public float t;
public int enemyCount;
public float musicBeatTweaking;

private IEnumerator DelayMusic()
{
    yield return new WaitForSeconds(1);
    Music.Post(gameObject);
    ResetMetronome();
}

public void ResetMetronome()
{
    tMusic =
        Time.time
        + EnemyWaveManager.instance.beatInterval
        - musicBeatTweaking;
    Debug.Log("Metronome starts at: " + tMusic);
}

void Update()
{
    if (tMusic < 0.1f)
        return;

    t = Time.time;
    if (t - tMusic > EnemyWaveManager.instance.beatInterval)
    {
        if (enemyCount > 0)
        {
            EnemyInteractiveDrums.instance.PlayEnemySpawn();
            enemyCount--;
        }
        tMusic = Time.time;
    }
}

// Start is called before the first frame update
void Start()
{
    StartCoroutine(DelayMusic());
    NormalSpeed.SetValue();
    Debug.Log("Normal Speed");
}
```

Figure 20 Delaying the Gameplay movement with code (see steps one and two).

3.3.2. *The Backing Group* in Gameplay

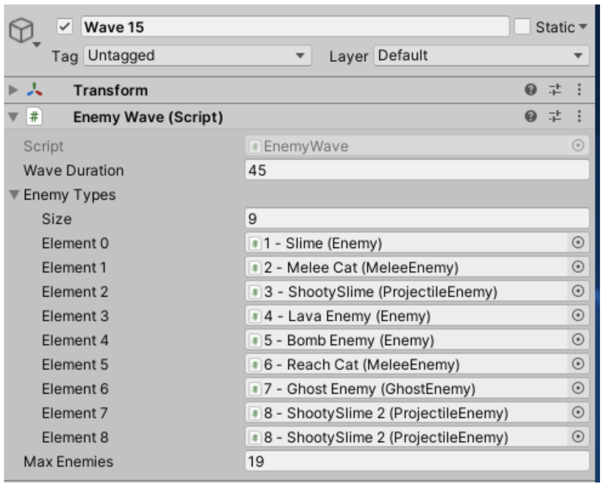
The *backing group* sequences four horizontal musical sections: WaveInit, WaveStart, WaveBeforeEnd, and WaveEnds, each of which are directly associated with a stage in the progress of a level.



Enemy Wave Manager (Script)

Script	EnemyWaveManager
Waves	
Size	20
Element 0	Wave 1 (EnemyWave)
Element 1	Wave 2 (EnemyWave)
Element 2	Wave 3 (EnemyWave)
Element 3	Wave 4 (EnemyWave)
Element 4	Wave 5 (EnemyWave)
Element 5	Wave 6 (EnemyWave)
Element 6	Wave 7 (EnemyWave)
Element 7	Wave 8 (EnemyWave)
Element 8	Wave 9 (EnemyWave)
Element 9	Wave 10 (EnemyWave)
Element 10	Wave 11 (EnemyWave)
Element 11	Wave 12 (EnemyWave)
Element 12	Wave 13 (EnemyWave)
Element 13	Wave 14 (EnemyWave)
Element 14	Wave 15 (EnemyWave)
Element 15	Wave 16 (EnemyWave)
Element 16	Wave 17 (EnemyWave)
Element 17	Wave 18 (EnemyWave)
Element 18	Wave 19 (EnemyWave)
Element 19	Wave 20 (EnemyWave)

Figure 21 Ordering of the 20 levels.

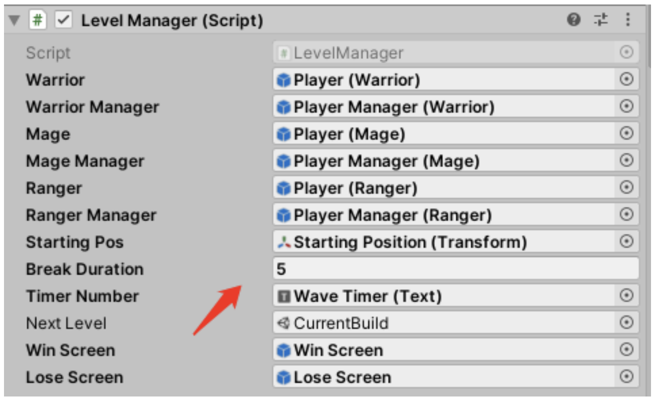


Wave 15 — Static
Tag Untagged — Layer Default
Transform
Enemy Wave (Script)

Script	EnemyWave
Wave Duration	45
Enemy Types	
Size	9
Element 0	1 - Slime (Enemy)
Element 1	2 - Melee Cat (MeleeEnemy)
Element 2	3 - ShootySlime (ProjectileEnemy)
Element 3	4 - Lava Enemy (Enemy)
Element 4	5 - Bomb Enemy (Enemy)
Element 5	6 - Reach Cat (MeleeEnemy)
Element 6	7 - Ghost Enemy (GhostEnemy)
Element 7	8 - ShootySlime 2 (ProjectileEnemy)
Element 8	8 - ShootySlime 2 (ProjectileEnemy)
Max Enemies	19

Figure 22 Level design of level No.15.

There are a total of 20 levels in *DOD*. Each level contains specific enemy types of a predetermined number(Figure 21). Each level also has a fixed duration. Figure 22 shows level 15's design as an example of the combination of enemies that may occur and an illustration of how long a level last. The game parameter that is useful for determining horizontal musical sections is the level's duration. Regardless of how many enemies appear in a level, they are cleared automatically by the conclusion of the level. A message then tells the player that they have survived the level and after a short break (which is currently set to five seconds but is alterable, as shown in Figure 23), the next level loads. The game mechanics also involve a looping process that levels come and go repeatedly (upper part in Figure 24). Thus, a sequential loop in the music can accompany this game mechanics' loop. The timing of the musical loop and the game loop are perfectly aligned with each other, as shown in Figure 24. In order to musically reflect the break between levels, I did not use any music during the break. This point could be considered the end of a complete loop in both the music and the game mechanics. The music begins to loop again when the next level begins.



Level Manager (Script)

Script	LevelManager
Warrior	Player (Warrior)
Warrior Manager	Player Manager (Warrior)
Mage	Player (Mage)
Mage Manager	Player Manager (Mage)
Ranger	Player (Ranger)
Ranger Manager	Player Manager (Ranger)
Starting Pos	Starting Position (Transform)
Break Duration	5
Timer Number	Wave Timer (Text)
Next Level	CurrentBuild
Win Screen	Win Screen
Lose Screen	Lose Screen

Figure 23 Setting break time between levels in seconds.

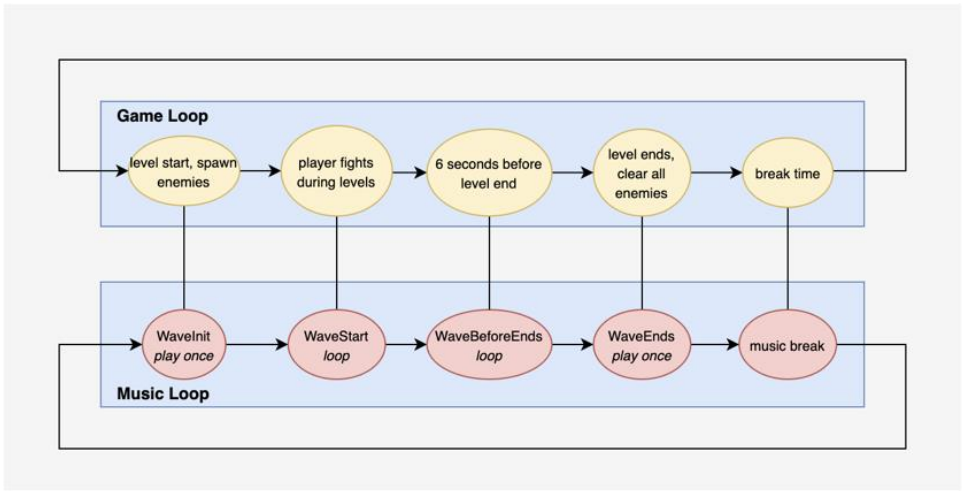
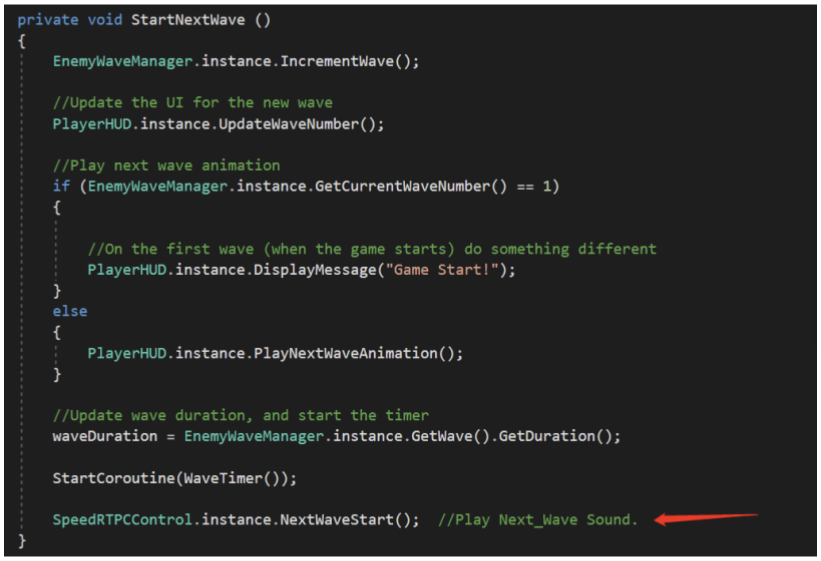


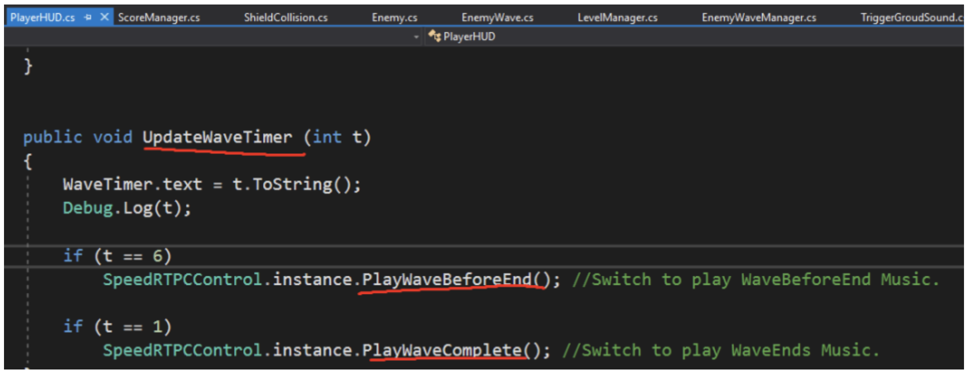
Figure 24 Level loop with music loop.

WaveInit, a musical section which provides a sense of settling into a level, plays at the beginning of each level. Every time a new level is executed (including level 1), the game engine will process the StartNextWave() function shown in Figure 25. This function is also used to trigger the WaveInit section. The next algorithmic time-point is six seconds before the end of each level. This is represented by the integer t in Figure 26. A level's duration is tracked using the UpdateWabeTimer(int t) function which counts down the time remaining in each level. When t=6, the music system switches to the WaveBeforeEnd section.



```
private void StartNextWave ()
{
    EnemyWaveManager.instance.IncrementWave();

    //Update the UI for the new wave
    PlayerHUD.instance.UpdateWaveNumber();

    //Play next wave animation
    if (EnemyWaveManager.instance.GetCurrentWaveNumber() == 1)
    {
        //On the first wave (when the game starts) do something different
        PlayerHUD.instance.DisplayMessage("Game Start!");
    }
    else
    {
        PlayerHUD.instance.PlayNextWaveAnimation();
    }

    //Update wave duration, and start the timer
    waveDuration = EnemyWaveManager.instance.GetWave().GetDuration();

    StartCoroutine(WaveTimer());

    SpeedRTPCControl.instance.NextWaveStart();  //Play Next_Wave Sound.
}
```

Figure 25 StartNextWave() triggers WaveInit.



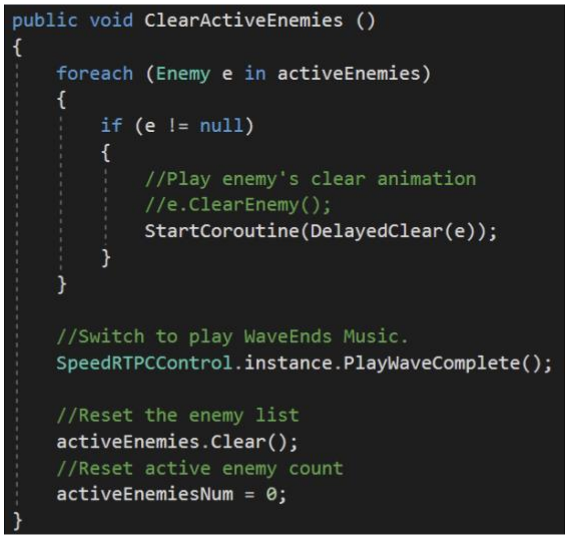
```
PlayerHUD.cs  ScoreManager.cs  ShieldCollision.cs  Enemy.cs  EnemyWave.cs  LevelManager.cs  EnemyWaveManager.cs  TriggerGroudSound.c
                                             PlayerHUD
}


public void UpdateWaveTimer (int t)
{
    WaveTimer.text = t.ToString();
    Debug.Log(t);

    if (t == 6)
        SpeedRTPCControl.instance.PlayWaveBeforeEnd(); //Switch to play WaveBeforeEnd Music.

    if (t == 1)
        SpeedRTPCControl.instance.PlayWaveComplete(); //Switch to play WaveEnds Music.
```

Figure 26 Locating the six seconds countdown from level end.

Time between the WaveInit and WaveBeforeEnds phrases is reserved for the WaveStart section to play. The WaveStart section is automatically triggered when WaveInit finishes playing. Because each level has a unique time duration, the total time between WaveInit and WaveBeforeEnd also varies. In order to fill the time in between, WaveStart is designed as a looping section. In contrast to the WaveStart section, which has a different looping length in each level, the WaveBeforeEnd section is set to loop for six seconds in all levels. The WaveBeforeEnd section and

WaveStart section differ in their rhythmic patterns and emotional connotation. The WaveStart section sounds like a statement while the WaveBeforeEnd section is energetic and anxious. As described in 3.1.6. Meter Design, the meters of the two sections also differ.

I used the ClearActiveEnemies() function, called at the end of a level, to trigger the PlayWaveComplete() function which switches the music to the WaveEnds section in Wwise (Figure 27). I did not implement this trigger in the UpdateWaveTimer() function in order to avoid covering the music change with the sound effect that occurs when enemies die. This avoids acoustic ambiguity and chaos. Putting the WaveEnds section's trigger inside the ClearActiveEnemies() function, especially after the foreach function, ensures the music change will be heard clearly after hearing the enemy dead sound effects. The WaveEnds section is played only once on each level.



```
public void ClearActiveEnemies ()
{
    foreach (Enemy e in activeEnemies)
    {
        if (e != null)
        {
            //Play enemy's clear animation
            //e.ClearEnemy();
            StartCoroutine(DelayedClear(e));
        }
    }

    //Switch to play WaveEnds Music.
    SpeedRTPCControl.instance.PlayWaveComplete();

    //Reset the enemy list
    activeEnemies.Clear();
    //Reset active enemy count
    activeEnemiesNum = 0;
}
```

Figure 27 Clearing active enemies on the game board when level ends.

3.3.3. Triggering Conditions and Transitions

3.3.3.1. Between Ground Types

During the WaveStart sectional loop, there is another set of algorithms that will trigger different phrases to be played. Specific phrases will be triggered in association with the type of ground that the player-controlled dice character is on: regular, sand, grass, or wood ground type. The dynamic phrase interaction with ground types only appears in the WaveStart section. The game board with labeled ground types is shown in Figure 28.

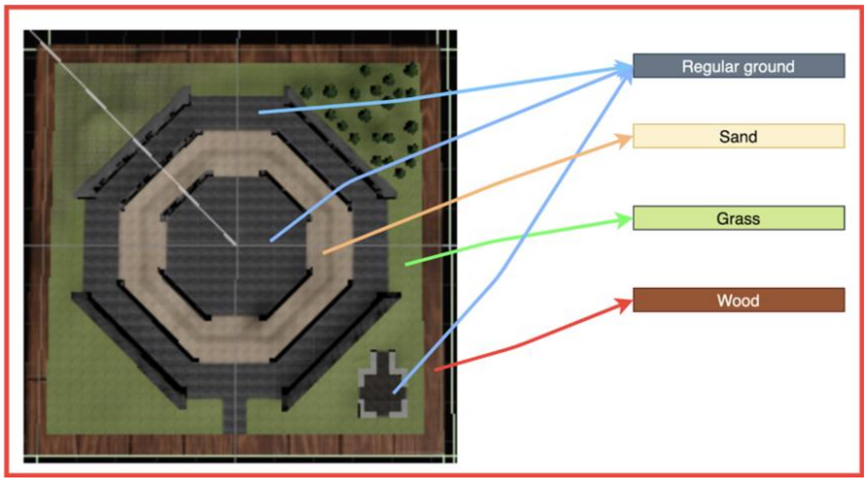


Figure 28 The different ground types on the game board.

The *backing group* is played by the *West Africa* ensemble which contains the *Dununba*, *Sangba*, *Kenkeni*, and two *Djembes*. Among these instrumental layers, the phrase patterns played on the two *Djembes* are not affected by changes of ground type. In order to properly trigger corresponding phrases, I created specific colliders for each ground type and set the colliders to trigger the associated drum phrase change when the player collides with them, as shown in Figure 29. Figure 30 shows that a script controls the interaction between ground type and music. Because this interactivity should respond and reflect the game states, the phrase change happens immediately upon being triggered. I imported all phrases for the four ground types to Wwise. They are set to play together while an individual phrase can be dynamically muted or unmuted according to the associated game events. As shown in Figure 31, the Sand phrase is set to play at normal volume when the Sand state is triggered, while if another state were triggered, the Sand phrase would be reduced to -108 dB. A video demonstrating the ground type colliders can be found in the demo video playlist via the link provided in Appendix D.

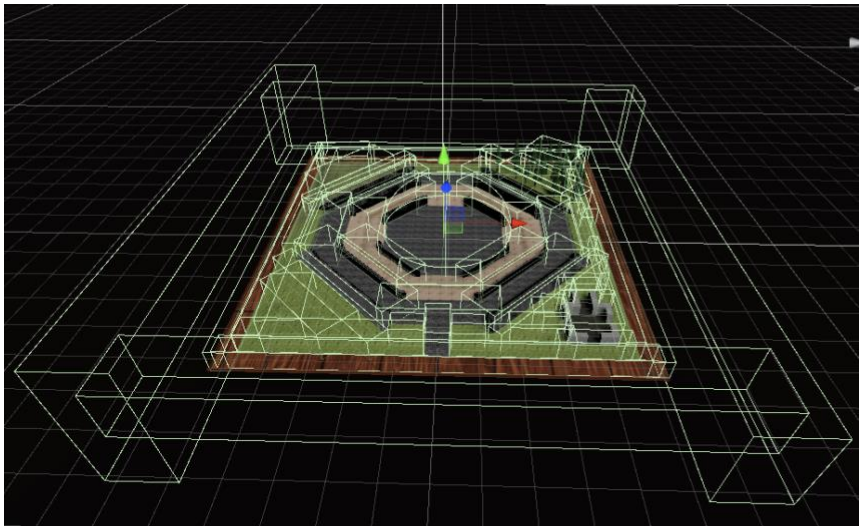
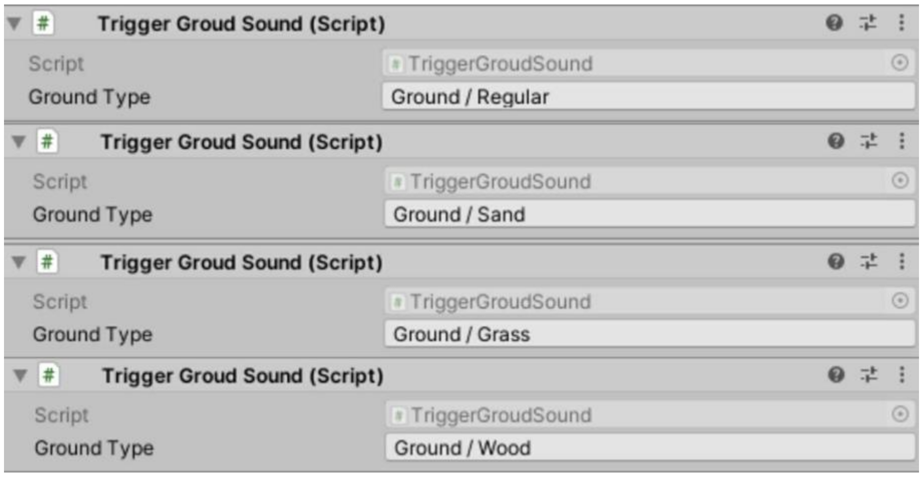


Figure 29 Game area of the ground types marginalized with colliders.



Trigger Groud Sound (Script)

Script	TriggerGroudSound
Ground Type	Ground / Regular

Trigger Groud Sound (Script)

Script	TriggerGroudSound
Ground Type	Ground / Sand

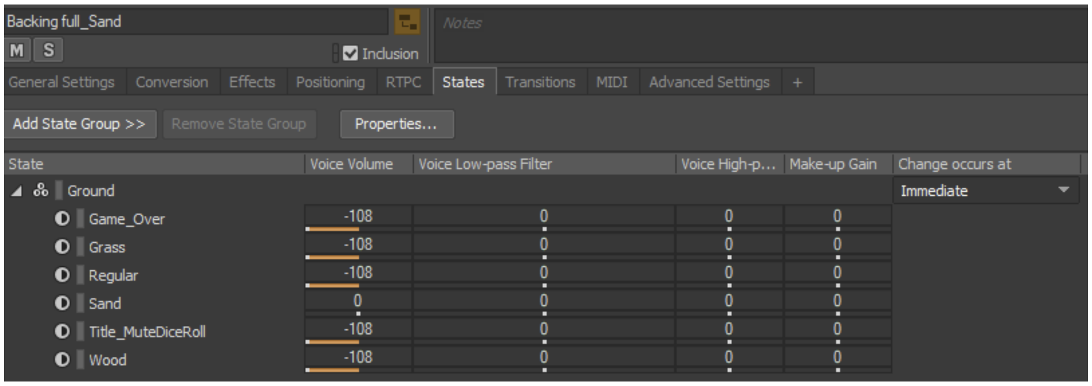
Trigger Groud Sound (Script)

Script	TriggerGroudSound
Ground Type	Ground / Grass

Trigger Groud Sound (Script)

Script	TriggerGroudSound
Ground Type	Ground / Wood

Figure 30 Each collider is set with a Wwise state type.


Backing full_Sand

Inclusion ☑

State	Voice Volume	Voice Low-pass Filter	Voice High-p...	Make-up Gain	Change occurs at
Ground					Immediate
Game_Over	-108	0	0	0	
Grass	-108	0	0	0	
Regular	-108	0	0	0	
Sand	0	0	0	0	
Title_MuteDiceRoll	-108	0	0	0	
Wood	-108	0	0	0	

Figure 31 Wwise setting for the Sand phrase.

3.3.3.2. Between Generative Materials, BPMs, and Sections - Custom Cue Points

In each of the tracks in the *backing group*, I have set up custom cue points at different rhythmic points that are musical, such as on every beat or every two bars. Wwise allows triggers to be set at every beat or measure without using a custom cue point, but for the goal here, a trigger on every two bars was required.[21] It was also necessary to label every beat with ascending numbers, thus custom cue points were necessary. Notice in Figure 32, the waveform view in Wwise cannot display beats divided in triplets. This was another reason to create custom cues. The numbered custom cues represent eighth notes, and there are the “2bar” custom cues on every six beats. I use these custom cues to trigger 1) the materials belonging to the *generative-improvisatory drum group*, 2) different BPMs, and 3) transitions between horizontal sections of the gameplay.

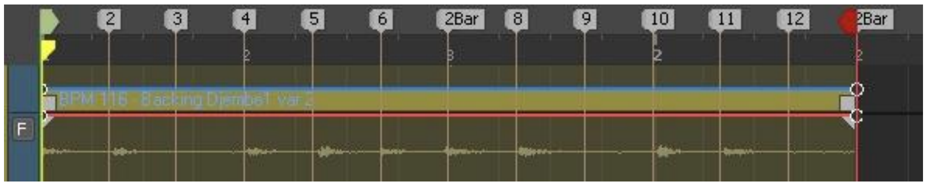


Figure 32 Creating and naming the custom cues.

The triggering condition for generative material is set to execute at “next beat.” Since the *backing group* is constantly playing during the Gameplay movement, generative material such as the EnemySpawn, EnemyHurt, EnemyDie, Pick-up types, and Player abilities will be triggered only until the next musical beat of the *backing group* tracks. This process syncs the ongoing *backing group* with the generative content and is explained in Figure 33. The Wwise setting is shown in Figure 34. The “Play At Next Custom Cue” option takes any type of custom cues including the

[21] The “two bars” here actually mean the positions on the track’s waveform in the Wwise project, which does not necessarily have to equal to two measures of the music in 12/8 meter. Since I had set in Wwise that all tracks are in 3/4 meter even though it has nothing to do with the actual music meters, the duration of the two bars actually is equal to the length of one measure in the music.

numbered beat cues or the "2Bar" cues; which one is triggered depends on whichever is first encountered.

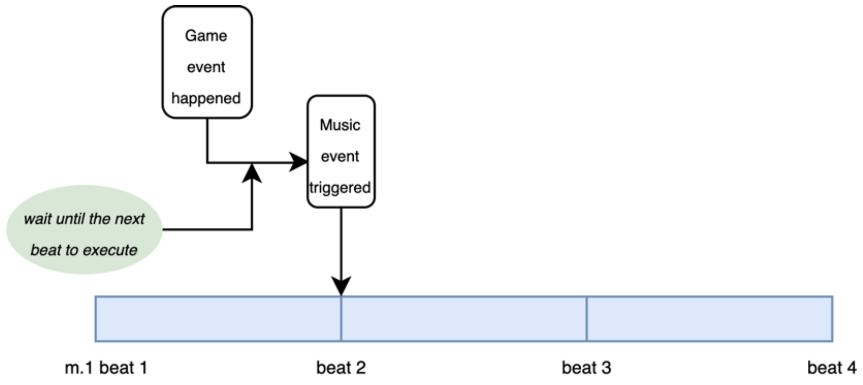
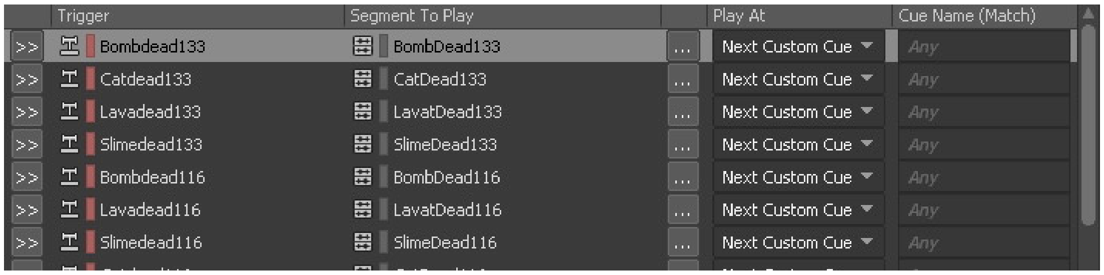


Figure 33 Trigger and Execute on the next beat.



Trigger	Segment To Play		Play At	Cue Name (Match)
Bombdead133	BombDead133	...	Next Custom Cue	Any
Catdead133	CatDead133	...	Next Custom Cue	Any
Lavadead133	LavatDead133	...	Next Custom Cue	Any
Slimedead133	SlimeDead133	...	Next Custom Cue	Any
Bombdead116	BombDead116	...	Next Custom Cue	Any
Lavadead116	LavatDead116	...	Next Custom Cue	Any
Slimedead116	SlimeDead116	...	Next Custom Cue	Any

Figure 34 Play stingers at the next custom cue.

Transitions between all BPMs in the Gameplay are made smooth and possible by assigning numbered custom cues to every beat. Ideally, when the game state requests that the music change tempo (as occurs at SpeedUp or SlowTime events), the tempo change should happen as soon as possible. The new tempo must also land perfectly at the same position in the new tempo's track during the tempo switch. For instance, when switching from BPM116 to BPM133 at measure one beat two, the slower tempo track should stop playing at the next custom cue (here that occurs at measure one beat three), while the new tempo track should start to play at measure one beat three despite the fact that these tracks have different durations for a standard beat and measure because of their different BPMs. This specific hurdle has been very challenging to overcome during the implementation process.

Two potential solutions exist, which I evaluated. I started by using one of Wwise's built in transition destinations, the "Same Time As Playing Segment" option, to see if the track being cued in enters at the appropriate moment. However, the result shows that this option's designated use is for transition between tracks that have the same tempo. In other words, the "Same Time As Playing Segment" indicates not the musical time, but the actual minute and second in both tracks, and which is problematic for transition between tracks of different tempos. A two-measure phrase in the 12/8 meter at BPM 133 is about 1804 milliseconds, while the same length phrase in BPM 152 is about 1579 milliseconds. This "Same Time As Playing Segment" method will not result in a smooth and precise transition.

The next method I explored was setting the transition destination to the "Random Custom Cue" option. It was not my intention to let the music transit totally randomly, and I further customized the transition by choosing the "Match source cue name" option. By doing this I was able to locate a specific beat using the numbered labels I created for each custom cue on every beat. When a transition is called, the music engine first takes the numbered label of the next custom cue from the currently playing track, then looks for a custom cue that has the same numbered label in the new track. For example, if the next custom cue on the current track is at cue "3," the new track will start playing on its own at cue "3." Figure 35 is a diagram that explains this process.

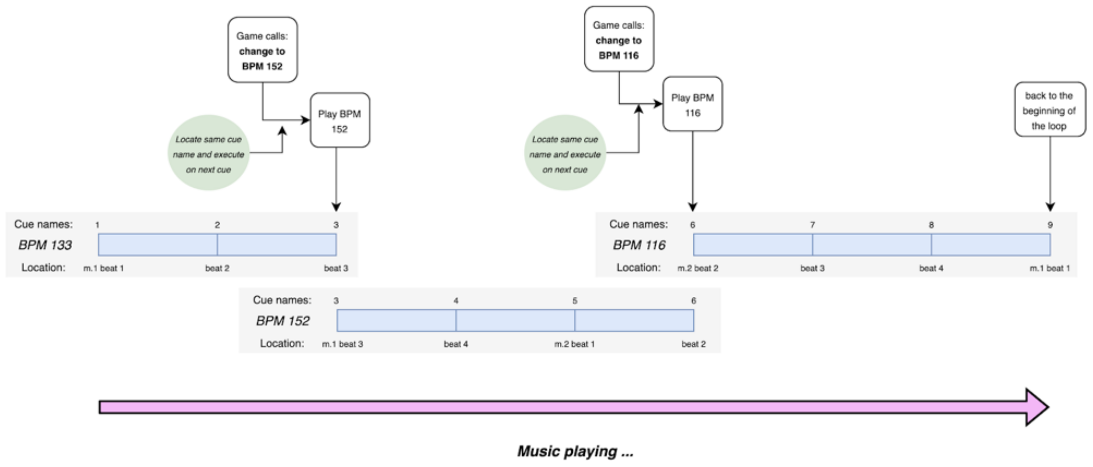
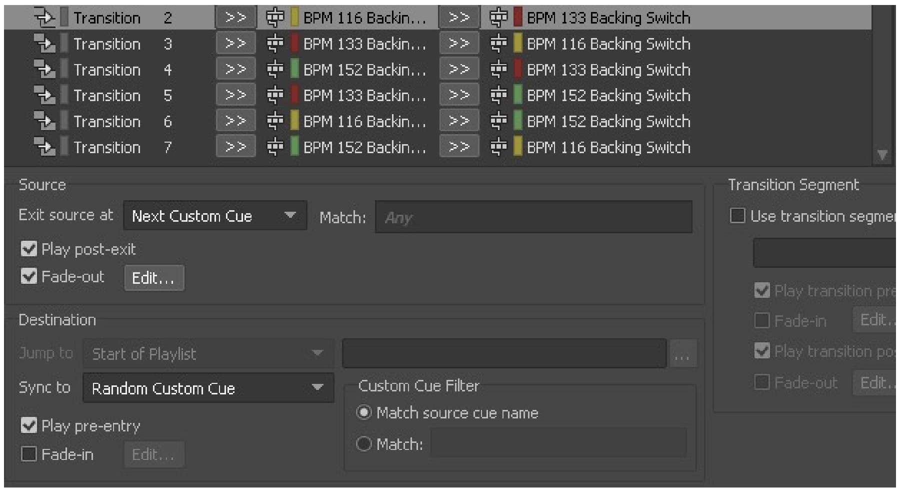

Game calls: change to BPM 152
Play BPM 152
Locate same cue name and execute on next cue
Game calls: change to BPM 116
Play BPM 116
Locate same cue name and execute on next cue
back to the beginning of the loop
Cue names: 1 2 3
BPM 133
Location: m.1 beat 1 beat 2 beat 3
Cue names: 6 7 8 9
BPM 116
Location: m.2 beat 2 beat 3 beat 4 m.1 beat 1
Cue names: 3 4 5 6
BPM 152
Location: m.1 beat 3 beat 4 m.2 beat 1 beat 2
Music playing ...

Figure 35 Transiting between different BPMs.

I did a quick listening test to compare the two methods by simply playing an audio track of a person counting from one to nine rhythmically, because words are clearer to hear than music especially when testing this tempo change function. The comparison clearly shows that the second method works best for the intended dynamic tempo effect. Videos demonstrating this comparison are available in the demo video playlist via the link provided in Appendix D: the videos show that using Sync to Same Time is not smooth at all in sound and creates significant glitches that some beats are repeated unexpectedly. However, using Sync to Custom Cue sounds smooth and effective. After this comparison, I set all tempo transitions to sync to "Random Custom Cue," as shown in Figure 36.



Transition	2	>>	BPM 116 Backin…	>> BPM 133 Backing Switch
Transition	3	>>	BPM 133 Backin…	>> BPM 116 Backing Switch
Transition	4	>>	BPM 152 Backin…	>> BPM 133 Backing Switch
Transition	5	>>	BPM 133 Backin…	>> BPM 152 Backing Switch
Transition	6	>>	BPM 116 Backin…	>> BPM 152 Backing Switch
Transition	7	>>	BPM 152 Backin…	>> BPM 116 Backing Switch

Source
Exit source at: Next Custom Cue   Match: Any
☑ Play post-exit
☑ Fade-out Edit…

Destination
Jump to: Start of Playlist
Sync to: Random Custom Cue
☑ Play pre-entry
☐ Fade-in Edit…

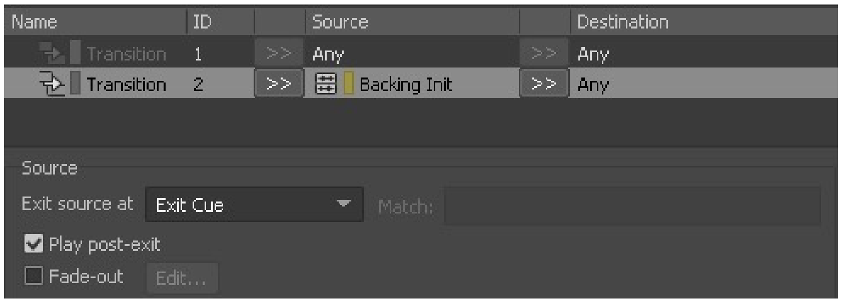
Custom Cue Filter
● Match source cue name
○ Match:

Transition Segment
☐ Use transition segme
☑ Play transition pre
☐ Fade-in Edit
☑ Play transition po
☐ Fade-out Edit

Figure 36 Sync to "Random Custom Cue".

Transitions between horizontal sections fall into two categories. From the WaveInit to the WaveStart section, the transition is triggered by locating the end of the WaveInit, as shown in Figure 37.[22] Other transitions between horizontal sections are triggered at the "2Bar" custom cues, as shown in Figure 38. Therefore, whenever the gameplay cues the WaveBeforeEnd or WaveEnds, the actual musical transition only executes when the current play cursor reaches the next custom cue in "2Bar."

[22] In the figure, the "Backing Init" actually refers to the WaveInit section.



Name	ID		Source		Destination
Transition	1	>>	Any	>>	Any
Transition	2	>>	Backing Init	>>	Any

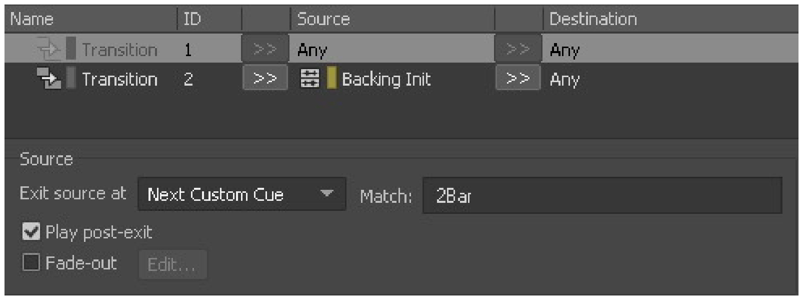
Source

Exit source at: Exit Cue — Match:

☑ Play post-exit

☐ Fade-out — Edit...

Figure 37 WaveInit to WaveStart transition.



Name	ID		Source		Destination
Transition	1	>>	Any	>>	Any
Transition	2	>>	Backing Init	>>	Any

Source

Exit source at: Next Custom Cue — Match: 2Bar

☑ Play post-exit

☐ Fade-out — Edit...

Figure 38 Transition between other sections in gameplay.

3.3.3.3. Between movements

When moving from the Gameplay to Game Over movement, or from the Game Over movement to the Title movement, no transitional passages are played. The switch between movements is triggered by game events (e.g., the player died, or the player chose to return to the Title scene), and like the generated materials above, these musical changes execute on the appropriate musical beat, or custom cue of the currently playing tracks.

3.4. Progressions, Phrases, and Patterns

In contrast to the above chapter which primarily discusses the organization of music sections at a relatively large scale, or how they are put together to form a larger form or sequence, this chapter focuses on the arrangement of smaller scale musical modules, specifically, the musical progressions, phrases, and rhythmic patterns within those modules. Figure 39 illustrates the structure of *DOD*'s entire game score. Within Game Music, there are three movements which have the five tempos and three meters in the parallel hierarchy. Sections and sequences are the next enclosed hierarchy, while progressions, phrases, and rhythmic patterns are located further down in the succeeding lower hierarchy.

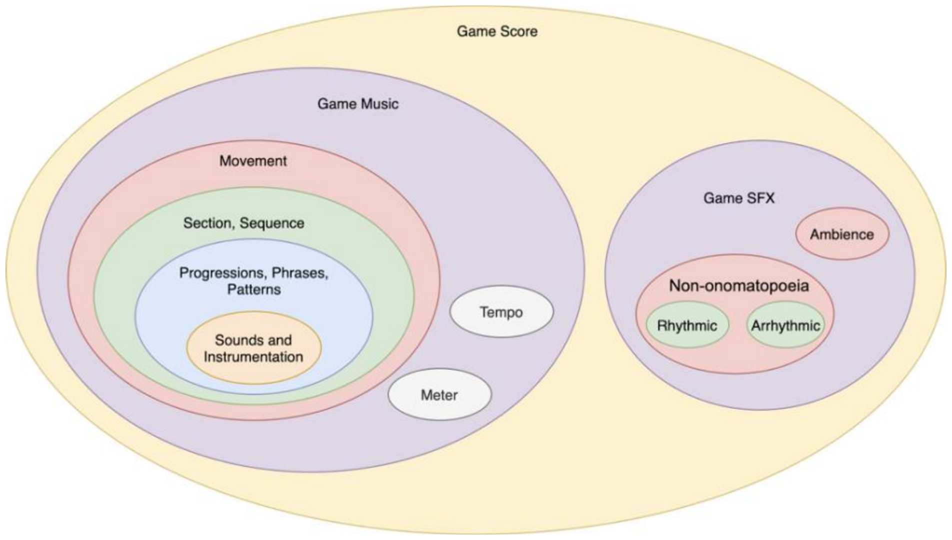


Figure 39 The hierarchical plan of the audio system of *DOD*.

I have specific definitions of *progressions*, *phrases* and *patterns* in this project. *Progression* exclusively describes the use of harmony in succession and pitches in collection and is only applicable to pitched movements (these being the Title and Game Over movements). *Phrases* are primarily short rhythmic passages that are pre-produced and rendered as audio files for the percussive movement (in this case, the Gameplay movement).[23] *Patterns* are musical modules that form generative rhythmic passages by algorithmically triggering one-shot samples[24] in musical time. For instance, samples corresponding with EnemySpawn and EnemyHurt events are one-shot samples. Generative *Patterns* may be dramatically different each time they occur and are either adaptive to the game states or interactive with player input.

3.4.1. Harmonic Progression in the Title and Game Over Movements

The Title and Game Over movements contain pitched instruments in which harmonic progressions are embodied. The fundamental harmonic progression here forms a plagal progression, represented by Roman numerals I-IV-I. Between the first twelve bars, the track sits on a pedaling note, the tonic, while harmonically switching between the tonic chord (I) and the Neapolitan chord (bII) on top of the pedal note. Figure 40 is a piano reduction of the Title movement. The first two chords are close to each other and move back and forth between one another several times. Between I and bII, the voice leading moves in stepwise motion; and application of the I pedal helps create a smooth tone blending and transition between the two chords. I6 is simply the first inversion of the I chord. I6 and IV9 share three common tones. Transitioning from IV to I forms the plagal cadence.

The repeat mark in the score indicates the end of the loop, while the last bar functions as a transition passage to the Gameplay movement and is played by a horn choir. Since the transition may be cued at any time (such as when the player decides to leave the Title scene and presses the "Start Game" button), the close musical relationship between all chords enables a smooth and natural arrival at the transitional bar from any point (I to I, bII to I, I6 to I, and IV^9 to I).

[23] A pre-produced and rendered track is an audio file whose content is not alterable regardless of how it interfaces with gameplay in real time.
[24] One-shot samples refer to samples that either have one occurrence of a sound's envelope ADSR (attack, decay, sustain, release), or samples whose duration is shorter than one eighth note in 12/8 meter at any tempos used in the Gameplay movement.

The melodic motifs played by the *India* virtual instrument library use the C Phrygian scale, also called the *Bhairavi* scale in North Indian music or *Hanumatodi* in South Indian music. This is also why the C minor triad is indicated as the tonic chord, but the key signature reflects a key with four flats.

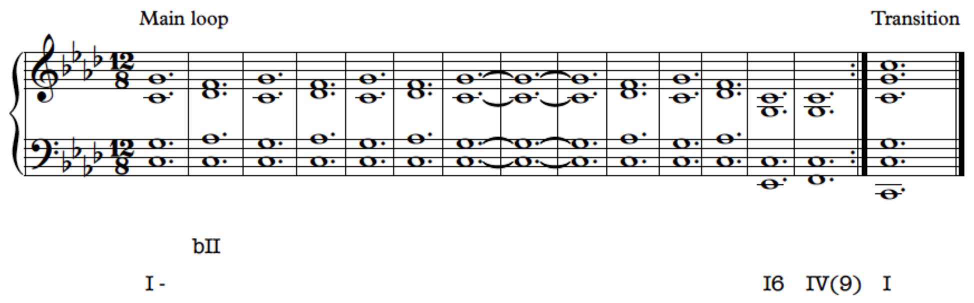


Figure 40 Piano reduction of Title movement.

The Game Over movement has a looping bassline as shown in Figure 41. The primary purpose of the Game Over movement in this project is to participate as a part of the larger scale effect of dynamic tempos and meters. When the Gameplay movement switches to the Game Over movement, a switch from non-pitched to pitched material occurs. When the Game Over movement transitions back to the Title movement, any note from this looping bassline can land on the tonic chord of the Title movement within the reach of an intervallic distance equals to or less than a third (E to C - an interval of third, D to C - second, or C# to C - semitone).

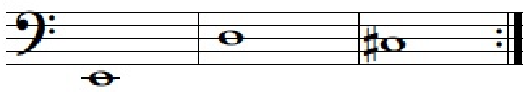


Figure 41 Bassline of the Game Over movement.

3.4.2. Phrases and Patterns in the Gameplay

As described in previous chapters, the gameplay movement consists of two parts, the *backing group* and the *GID group*. In this section, I will discuss the design of phrases and patterns within each of these groups.

The Backing Group

The *backing group* loop starts with a "Wave Initializer" which I named WaveInit in Wwise. This is also occasionally named "Backing Init" in Wwise. The musical phrase begins by introducing the sound of each cylindrical drum in succession, eventually beginning to play in tutti after two measures (Figure 42). The phrase begins with the *Dununba* with three bells, then adds the *Sangba* (using both open and muted samples) with the mid-register bell at bar one beat five, and lastly it adds the *Kenkeni* (using both open and muted samples) with the high-register bell at bar two. Each drum plays two parts on a three-line staff. The notes with stem down represent drumbeats. Drum notes on the bottom line indicate that the drum is hit openly. Notes on the middle line indicate that the drum is hit muted. Notes with stems up indicate bells.

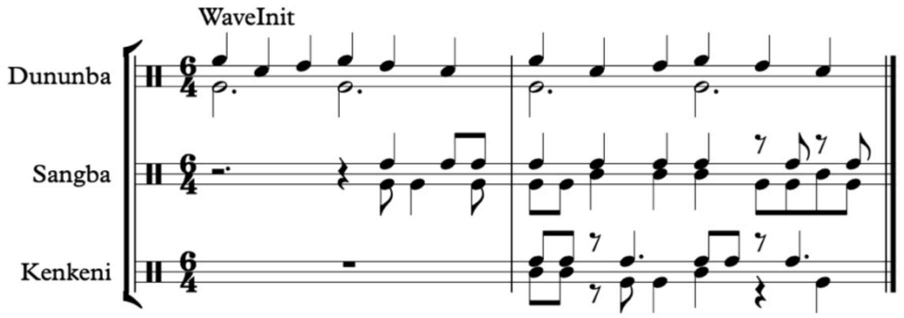


Figure 42 WaveInit score.

As discussed previously, the three cylindrical drums play four distinct rhythmic phrases which correspond to the four ground types during the WaveStart section. The *Dununba* is of the lowest register and provides foundation to the rhythmic phrases. Its phrases are less dynamic, as only the drum's timbral color and the tone of the bells vary subtly in each ground type. The *Dununba* drum plays an open hit on stronger beats (beat one and four in 6/4) which sets up a stable rhythmic foundation. The *Sangba* and the *Kenkeni* phrases vary more dramatically in rhythm in relation to each ground type. Figure 43 is a score of the three cylindrical drums' phrases played over each ground type. The notation here indicates drums and bells in a similar way to the notation of the WaveInit score.

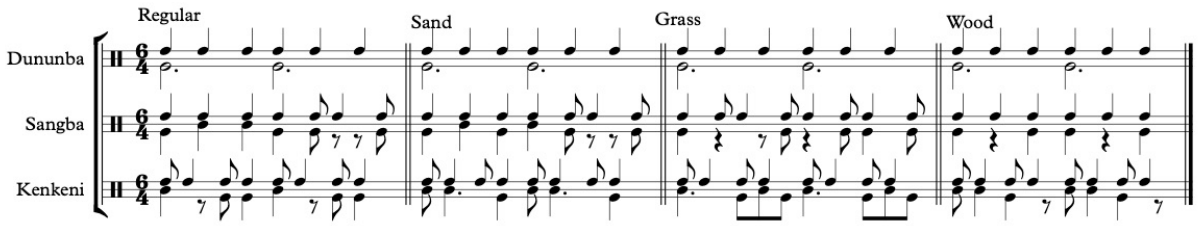


Figure 43 The cylindrical drum choir on different ground types.

When the cylindrical drums change phrases through different ground types during the WaveStart section, the two ensemble *Djembes* are added and played along. However, each *Djembe* phrase is randomly chosen from several phrase variations by the Wwise engine whenever the two bar phrase repeats in loop. I created five phrase variations for *Djembe* One and two variations for *Djembe* Two. In *West Africa*, each *Djembe* phrase includes two bass hit samples, two open hit samples, and two slap samples. Because the bass hits have similar sound with the *Dununba*, I omitted them from *Djembe* phrases in order to avoid doubling the *Dununba* on strong beats. In the score shown in Figure 44, notes on the middle line of the staff indicate open hits, while notes in the space directly above this line represent varied samples of the hits. The top line is indicative of a slap hit similar to an open hit. What appears to be pitches notated on the score represent variations of timbral color instead of actual pitch variations.

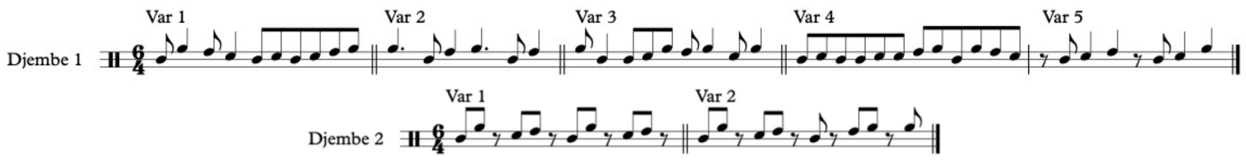


Figure 44 Score of the two ensemble *Djembes* in WaveStart section.

The WaveBeforeEnd phrase eliminates the two *Djembes* and only the cylindrical drums remain playing to the end of the phrase. The ensemble plays a more aggressive, looping energetic pre-coda phrase in this section. There is no dynamic content in this section because the WaveBeforeEnd only loops for six seconds at each level. As discussed in section 3.1.6. Meter Design, the WaveBeforeEnd section shifts from a compound duple to a compound quadruple meter (from 2, 2, 2, 2, 2, 2, to 3, 3, 3, 3). Having the drums playing on the four accented beats instantly communicates the meter change and a different rhythmic feeling. The score of this section is shown in Figure 45.

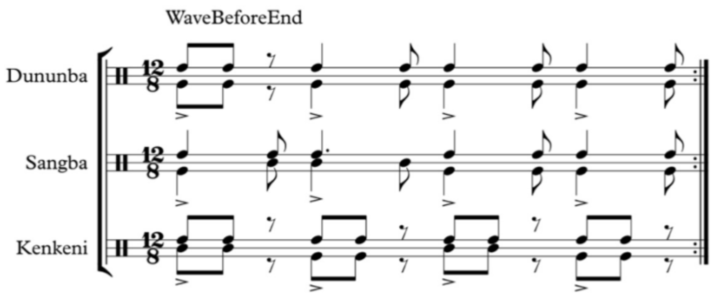


Figure 45 WaveBeforeEnd.

The WaveEnds phrase functions as a musical conclusion and is played at the end of each level. The phrase pattern at this point returns to 6/4 meter as shown in the score in Figure 46. This phrase signals to the player that the current level has ended, and the next level is about to begin.

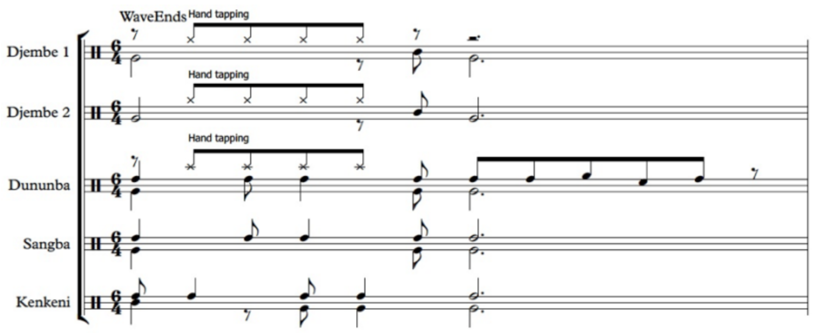


Figure 46 WaveEnds.

The GID Group

The EnemyDie, PlayerAbilities, and Pick-up Buffs phrases triggered in real time to form larger generative sequences do not use one-shot samples. These phrases also are exported as pre-rendered audio files. The six enemy types each have a unique EnemyDie phrase, and each phrase has several phrase variations. The phrases are played by different samples of the solo *Djembe* in the *West Africa* ensemble. Some also have additional sound effects and signal processing.

Phrases connected with the CatDie event are four beats long and contain different combinations of the three previously discussed timbres (bass, open hit, and slap). There are eleven variations of this phrase. The common element of the eleven variations is that they always end with an eighth note on the fourth beat. The patterns played in the first three beats can be divided into two groups: one which has hits in 16th notes or triplets, the other consists of four even eighth notes. Phrases associated with the LavaDie event are of six variations of a phrase, each of which begins with two hits in the bass timbre followed by a hit at a higher register, then another hit at the lower register (see Figure 47, row three). Additionally, the phrase always contains a flam hit, sometimes two, on either the third or fourth beat. Phrases connected with the BombDie event are five beats long and have four variations. On the last hit of each variation, a solo *Djembe* joins. Phrases associated with the SlimeDie event occur in six variations, all of which are five beats long. Every variation rests on beat three and ends with two hits on or between beats four and five. Phrases associated with the GhostDie event are six beats in length and occur in four variations. Each of these variations ends with three beats of rolling 16th notes. Last but not least, the MageDie event is associated with phrases that are six beats in length and which occur in four variations. Each phrase consists of two sets of dotted rhythm. All scores of the associated EnemyDie phrases are shown in Figure 47. The notation system layout is similar to Figure 44, where the three staff lines each represent the three timbres. Since the phrases are shorter than six beats, they are notated in 6/8 meter instead of 12/8 meter.

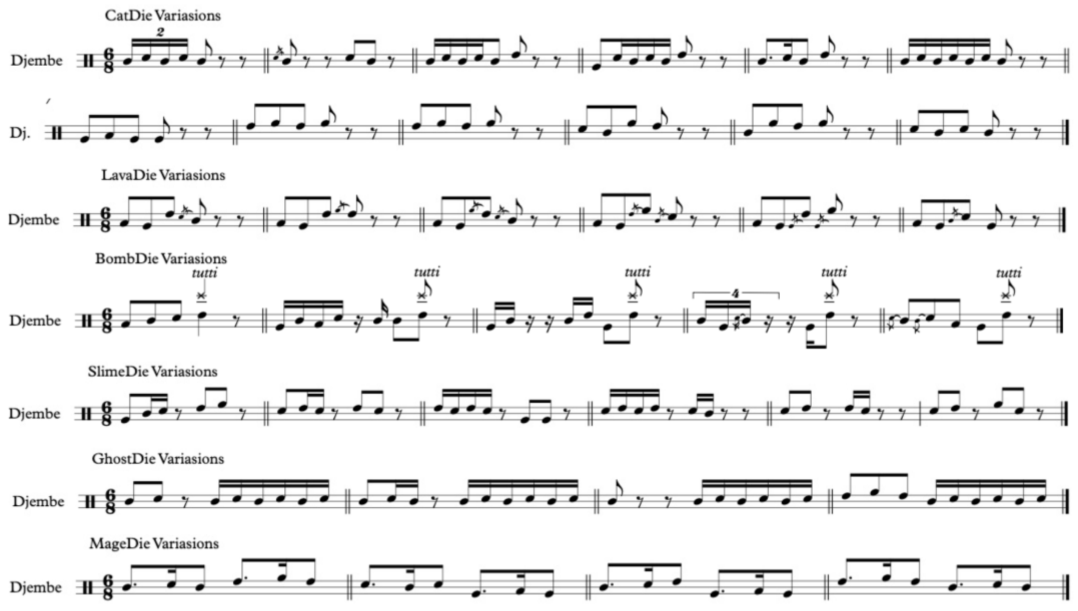


Figure 47 EnemyDie phrases.

PlayerAbilities and PickUp Buff Types are associated with longer phrases that are triggered in real time gameplay, shown in Figure 48.[25] In the phrases associated with PickUp events, I have experimented with different beat groupings for a total of twelve beats. For instance, the phrase associated with the ManaPickUp event occurs in a 4, 4, 4 pattern (see the topmost staff in Figure 48), the ScorePickUp event's associated phrase occurs in a 3, 3, 3, 3 pattern (see the fourth staff in Figure 48). Other pick-up associated phrases occur in a 6, 6 pattern. Two phrases associated with PlayerAbilities events are performed by the same drum rack device used in the Title movement. AbilityA is associated with a phrase which uses a 6, 6 pattern with the Slime's attack sound effect appearing at the last beats, and the AbilityB is associated with a phrase which uses a 3, 3, 3, 3 pattern shown in Figure 49.

[25] Because PickUp associated phrases are played by the three *Djembes* and bells in the ensemble, I used one music staff to notate *Djembes* of each type.

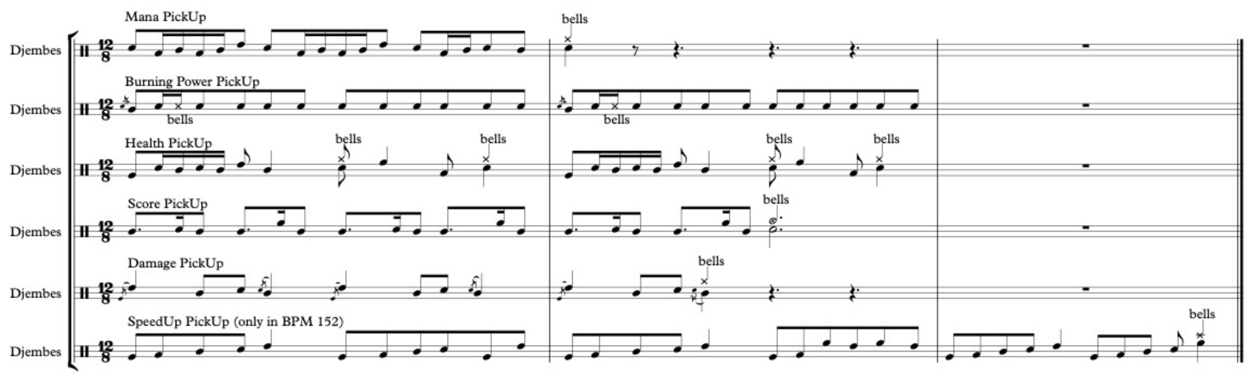


Figure 48 PickUp Types.

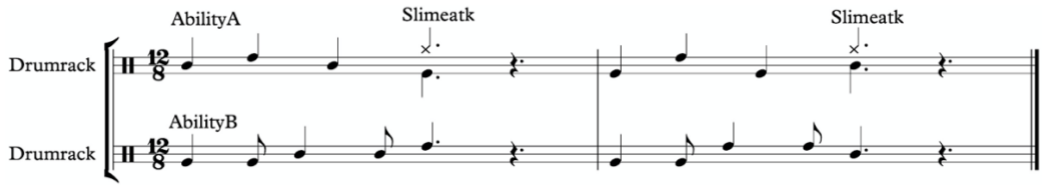


Figure 49 Player Abilities.

Game events which generate musical content may occur on any beat during real time gameplay.[26] This means that even though the phrase shown in the score starts on the downbeat of a measure, it can play on any beat in a measure in the real time score. This can result in a variety of polyrhythmic patterns in the *backing group*.

3.4.3. Generative Patterns, Timer, and Probabilities

Pattern Length

One-shot samples (used for the EnemySpawn and EnemyHurt events) in the *GID group* can form a series of percussive beats which form a generative rhythmic pattern in real time during gameplay. The EnemySpawn event generated patterns are normally most frequent at the beginning of each level, because enemies associated with each level are usually spawned during this time.[27] In

[26] Music events are triggered on the next beat as explained in 4.3.3. Triggering Conditions and Transitions.
[27] When an enemy die during a level, according to the total number of enemies designed for the level, a new enemy will be spawned right after.

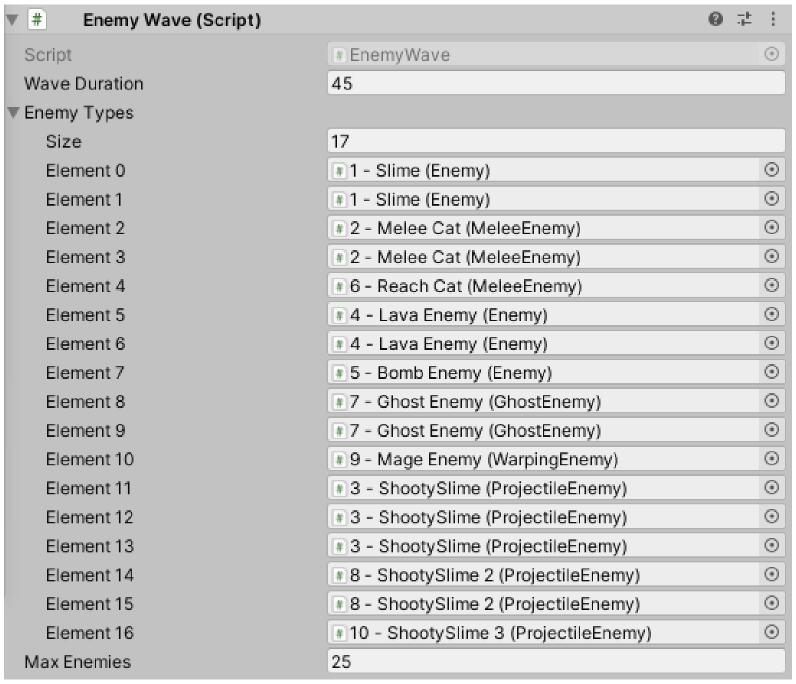
the dynamic music system, I created sonic hints to the player which indicate the quantity and type of enemy in the music. Enemies are spawned based on the game's timing and the enemies' ordering of appearance. Musical phrases associated with the EnemySpawn events also play in the same order at the nearest appropriate musical time point (on the next eighth or 16th note). These musical events also sync to the backing tracks. For instance, if a level spawns two Cat enemies and four Slime enemies at beginning, the player will hear the two associated sample types and a generated rhythmic phrase containing six hits which includes samples played in the same order in which the enemies were spawned. Additionally, although the enemies might all be spawned within one frame of gameplay, only one sample will be played at a musical time point. This avoids having all six hits occur on beat one, for example.

It was necessary to ensure that the EnemySpawn generative phrase is compact and finishes playing as quickly as possible so that it does not interfere with musical material generated by other game events, such as an EnemyHurt or EnemyDie event. A compact phrase also differs in character from an EnemyHurt and EnemyDie event; these are more loosely triggered and placed during gameplay. In order to create such a compact generative pattern, I limited the total length of the pattern triggered by an EnemySpawn event to finish within the length of two measures. Thus, the EnemySpawn event generated phrase should ideally finish with the WaveInit section (which is also two measures in length), without continuing into the WaveStart section.

I have previously introduced *DOD's* level design in Figure 21 and Figure 22. The level that contains the most enemies is level 20. As shown in Figure 50, it has a total of 25 enemies and includes all enemy types. During the designing process, I was unsure of how to fit 25 percussive hits into two measures. The solution was to trigger samples on a grid shorter than one beat. I eventually set the triggering musical grid to the length of a16th note.[28] If 16th notes are played in 6/8 meter

[28] The actual music here occurs in 8th notes, not 16th notes, but I mention it as "16th" because I treated the music as if it were in 6/8 meter, not 12/8 or 6/4, when designing the EnemySpawn event triggered musical patterns in Wwise.

for two measures, there are a total of 24 hits. If the EnemySpawn generated pattern starts on measure one beat two[29], 25 hits can be fit in with just two additional 16th notes continuing into the beginning of the WaveStart section. This is acceptable and the closest result to the initial design goal.[30]



Enemy Wave (Script)	
Script	EnemyWave
Wave Duration	45
Enemy Types	
Size	17
Element 0	1 - Slime (Enemy)
Element 1	1 - Slime (Enemy)
Element 2	2 - Melee Cat (MeleeEnemy)
Element 3	2 - Melee Cat (MeleeEnemy)
Element 4	6 - Reach Cat (MeleeEnemy)
Element 5	4 - Lava Enemy (Enemy)
Element 6	4 - Lava Enemy (Enemy)
Element 7	5 - Bomb Enemy (Enemy)
Element 8	7 - Ghost Enemy (GhostEnemy)
Element 9	7 - Ghost Enemy (GhostEnemy)
Element 10	9 - Mage Enemy (WarpingEnemy)
Element 11	3 - ShootySlime (ProjectileEnemy)
Element 12	3 - ShootySlime (ProjectileEnemy)
Element 13	3 - ShootySlime (ProjectileEnemy)
Element 14	8 - ShootySlime 2 (ProjectileEnemy)
Element 15	8 - ShootySlime 2 (ProjectileEnemy)
Element 16	10 - ShootySlime 3 (ProjectileEnemy)
Max Enemies	25

Figure 50 Level design of level 20.

Timer, Metronome, and Sequencer

Game programming with Unity scripts has a specific order of execution, as documented by Unity (Unity Technologies, 2020). The most popular functions are commonly executed in two fashions. One is similar to the Start() or StartNextWave() functions which execute their content once when they are called. The other requires functions to be placed inside the Update() method

Therefore, every note value discussed in this section actually sounds two times slower than its written note value in this paragraph (8th note = quarter note; 16th note = 8th note).

[29] Additionally, in order to avoid the EnemySpawn sound effect clashing with the generated one-shot samples in the music, the phrase is set to start on beat two.

[30] In any scenario, if the game expands to have more than 25 enemies in a level (which would be very challenging for the player), even smaller grids of the music can be used to generate phrases.

which is executed on a continuous basis because this function is called on at every frame.[31] The two types of functions are important because most game states and mechanics can be achieved with them and because by applying specific conditions and limitations as well as different combinations of these two types of functions, sophisticated timing algorithms can be created for a variety of uses in the music.

In the case of the ADM system, one of the most important and convenient features of Wwise is its ability to manipulate algorithms while using musical timing and musical concepts like tempo, beats, bars, and note values. Musical events would not be meaningful and harmonious if executed without musical timing.

Several things have made using the stinger function in Wwise to trigger samples associated with EnemySpawn events impossible. I had to make customized scripts to successfully trigger EnemySpawn event related phrases. In *DOD*, the duration of one eighth note in any of the three available tempos is extremely short (172.41ms at BPM 116, 150.38ms at BPM 133, and 131.58ms at BPM 152). I used float numbers to represent these time durations and named them beatInterval in the scripts. There is a 400ms cooldown time at the beginning of the level, before which enemies cannot spawn. This is much greater than the beatInterval's value, so it is not possible to use the Start() function to trigger the EnemySpawn event associated samples. Additionally, although the total number of enemies in each level is fixed and known, the enemy quantity in the scripts, first, cannot simply be used for triggering Wwise to play the EnemySpawn samples for a certain number of beats in succession at a constant pace.[32] Second, the game triggers have to be set at a musically meaningful timing which syncs to the backing track's beats. Having generative MIDI data play the

[31] More info about time and frame rate management in Unity can be viewed at: https://docs.unity3d.com/Manual/TimeFrameManagement.html
[32] In Wwise, new music events can only be triggered in two ways: either by a direct call from the game engine, or by a call by any currently playing tracks. The already playing tracks are also triggered by a call from the game engine.

samples would solve the problem, but Wwise currently does not have a function for generating real time MIDI data.

Not using Wwise's stinger function means that some of the game-music interactivity has to be programmed within the game algorithm. Some of this includes:

- A *time recorder* that records the start of the WaveInit section and initialize a *metronome.*[33]
- When the WaveInit section begins, the metronome outputs a series of calls at a constant pace which is exactly matched to the currently playing music's tempo.
- Then, an algorithmic function detects the number of enemies that are spawned in a level so that the number of EnemySpawn event associated samples played can match it.
- Last, executing EnemySpawn event triggered samples are executed on each of the calls put out by the metronome, with the total quantity matching the quantity of spawned enemies.

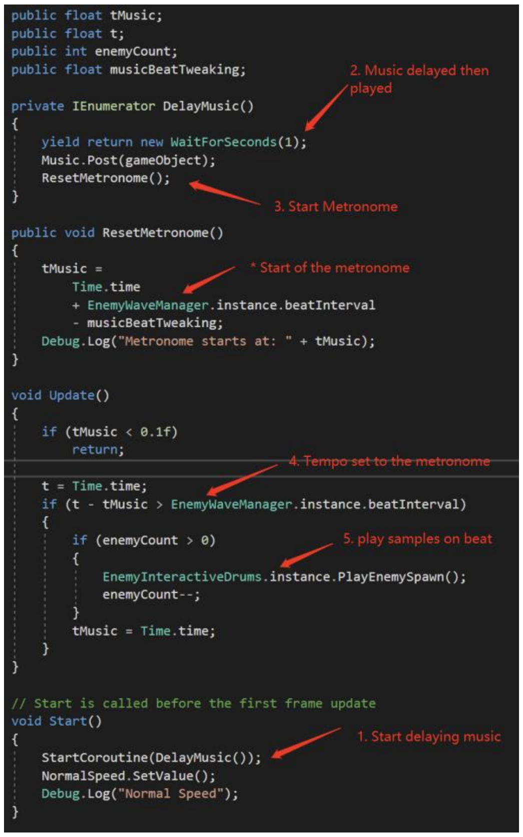
The above process functions as a coded music sequencer with regularly occurring beats equalized to dynamic tempos (with BPM values of 116, 133, and 152) for a determined duration (two bars plus two beats for the maximum quantity of 25 spawned enemies, as discussed previously).

Figure 51 is a series of screenshots from the game algorithms that show how I scripted this music sequencing process. The *backing* tracks are triggered under the Start() method in the StartCoroutine(DelayMusic()) function. The backing is then delayed for one second ("WaitForSeconds(1)" in code) to avoid it clashing with the Title Transition. After the delay, the WaveInit section ("Music.Post(gameObject);" in code) begins. Immediately after WaveInit begins, the metronome is set by the ResetMetronome() function. tMusic is the start time of the metronome,

[33] The metronome is a steady pulse set up in the game engine which is used for locating musical beats in various tempos.

which is determined by the sum of metronome's set time (Time.time) and beatInterval[34] minus musicBeatTweaking.[35] After the start time of the metronome is recorded, the Update() function will start the metronome which outputs a series of calls at a constant rate reflective of the current music's tempo. The metronome can function properly due to the fact that it constantly compares the last recorded tMusic value with the ongoing time as t. The continuous game calls will be sent out when the difference between tMusic and t exactly matches the beatInterval. Then, the current value of t will be used for updating the tMusic's value. This process of comparing time differences and updating tMusic's value continues to loop. The enemyCount variable in the function keeps count of how many new enemies have been spawned. This count is then used to execute the same number of EnemySpawn event associated samples (using "PlayEnemySpawn()" in the code). This process is demonstrated graphically in Figure 52.

[34] The beatInterval value is used to avoid music clashing with enemy associated sound effects.
[35] The musicBeatTweaking value is a customizable value for fine tuning the beat synchronization or adding swing to the EnemySpawn event associated samples.



```
public float tMusic;
public float t;
public int enemyCount;
public float musicBeatTweaking;

private IEnumerator DelayMusic()
{
    yield return new WaitForSeconds(1);
    Music.Post(gameObject);
    ResetMetronome();
}

public void ResetMetronome()
{
    tMusic =
        Time.time
        + EnemyWaveManager.instance.beatInterval
        - musicBeatTweaking;
    Debug.Log("Metronome starts at: " + tMusic);
}

void Update()
{
    if (tMusic < 0.1f)
        return;

    t = Time.time;
    if (t - tMusic > EnemyWaveManager.instance.beatInterval)
    {
        if (enemyCount > 0)
        {
            EnemyInteractiveDrums.instance.PlayEnemySpawn();
            enemyCount--;
        }
        tMusic = Time.time;
    }
}

// Start is called before the first frame update
void Start()
{
    StartCoroutine(DelayMusic());
    NormalSpeed.SetValue();
    Debug.Log("Normal Speed");
}
```

Figure 51 Music sequencer for EnemySpawn.

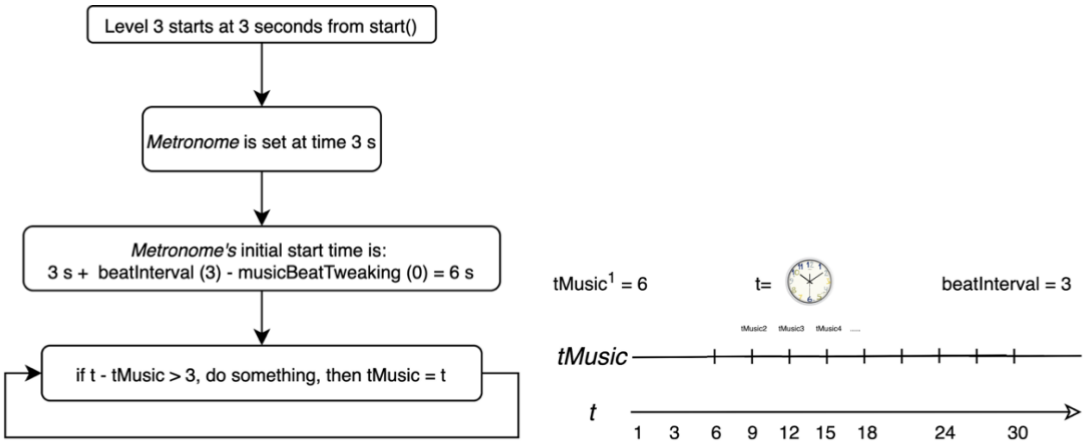
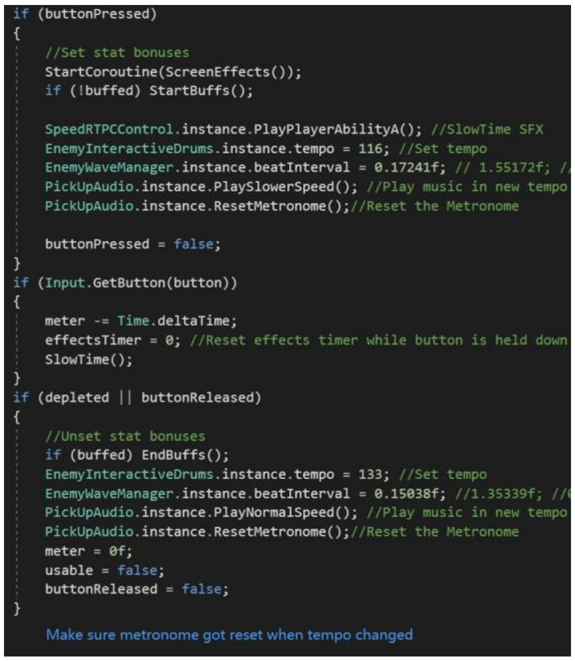

Level 3 starts at 3 seconds from start()
Metronome is set at time 3 s
Metronome's initial start time is:
3 s + beatInterval (3) - musicBeatTweaking (0) = 6 s
if t - tMusic > 3, do something, then tMusic = t
tMusic1 = 6
t=
beatInterval = 3
tMusic2
tMusic3
tMusic4
tMusic
t
1
3
6
9
12
15
18
24
30

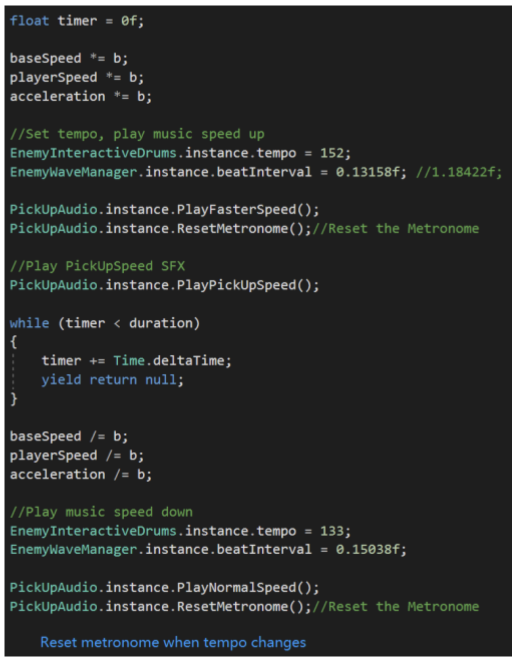
Figure 52 The programmed metronome.

The "beatInterval" variable is a variable that dynamically reflects the backing's tempo changes in real time. Figure 53 shows how beatInterval values follow the dynamic tempo. Whenever the tempo changes, the ResetMetronome() function in Figure 51 is immediately triggered, the metronome resets and the generative sample-sequencing process restarts, syncing the music to the new tempo.



```
if (buttonPressed)
{
    //Set stat bonuses
    StartCoroutine(ScreenEffects());
    if (!buffed) StartBuffs();

    SpeedRTPCControl.instance.PlayPlayerAbilityA(); //SlowTime SFX
    EnemyInteractiveDrums.instance.tempo = 116; //Set tempo
    EnemyWaveManager.instance.beatInterval = 0.17241f; // 1.55172f; /
    PickUpAudio.instance.PlaySlowerSpeed(); //Play music in new tempo
    PickUpAudio.instance.ResetMetronome();//Reset the Metronome

    buttonPressed = false;
}
if (Input.GetButton(button))
{
    meter -= Time.deltaTime;
    effectsTimer = 0; //Reset effects timer while button is held down
    SlowTime();
}
if (depleted || buttonReleased)
{
    //Unset stat bonuses
    if (buffed) EndBuffs();
    EnemyInteractiveDrums.instance.tempo = 133; //Set tempo
    EnemyWaveManager.instance.beatInterval = 0.15038f; //1.35339f; //
    PickUpAudio.instance.PlayNormalSpeed(); //Play music in new tempo
    PickUpAudio.instance.ResetMetronome();//Reset the Metronome
    meter = 0f;
    usable = false;
    buttonReleased = false;
}
```

Make sure metronome got reset when tempo changed



```
float timer = 0f;

baseSpeed *= b;
playerSpeed *= b;
acceleration *= b;

//Set tempo, play music speed up
EnemyInteractiveDrums.instance.tempo = 152;
EnemyWaveManager.instance.beatInterval = 0.13158f; //1.18422f;

PickUpAudio.instance.PlayFasterSpeed();
PickUpAudio.instance.ResetMetronome();//Reset the Metronome

//Play PickUpSpeed SFX
PickUpAudio.instance.PlayPickUpSpeed();

while (timer < duration)
{
    timer += Time.deltaTime;
    yield return null;
}

baseSpeed /= b;
playerSpeed /= b;
acceleration /= b;

//Play music speed down
EnemyInteractiveDrums.instance.tempo = 133;
EnemyWaveManager.instance.beatInterval = 0.15038f;

PickUpAudio.instance.PlayNormalSpeed();
PickUpAudio.instance.ResetMetronome();//Reset the Metronome
```

Reset metronome when tempo changes

Figure 53 Tempo change, beatInterval follows, then metronome resets.

As discussed previously, I eventually used both eighth note and 16th note samples to generate EnemySpawn event associated phrases because this allows 25 EnemySpawn hits to occur within roughly two measures. Instead of producing all the one-shot samples in both eighth note and 16th note versions, I produced only eighth note samples because they are sufficient to all the pattern types needed. One is an eighth note sample to be played on the downbeat, the other is an eighth note sample to be played on the upbeat with a 16th note rest at its beginning. As shown in Figure 54, a 16th note sample playing on the downbeat will sound similar to an eighth note sample, and in a similar way, placing two 16th notes together will sound similar to placing two eighth note samples together. This exploration shows that the more important factor in creating an algorithmic rhythm isn't the note's length but where on the rhythmic grid a note is placed.

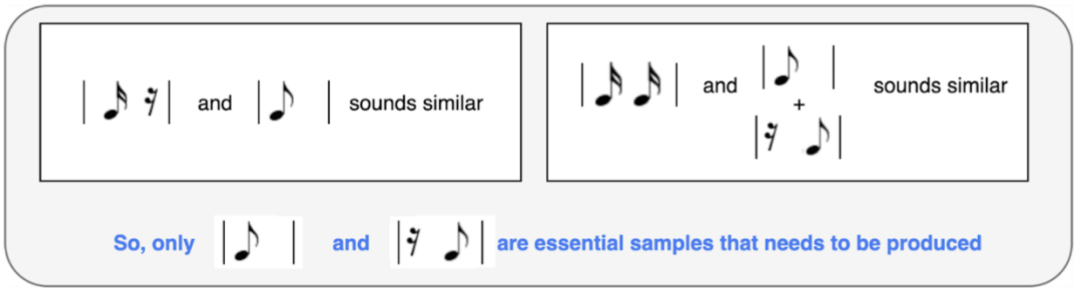


Figure 54 Essential samples needed for creating different rhythmic patterns that divide in both eighth note and 16th note.

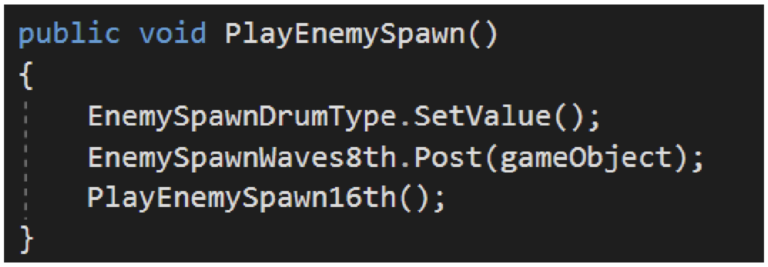
This strategy not only saves disc storage, but it is also easier to implement. Since the second type of essential samples start with a rest, its triggering condition and synchronization setting can share the same setting with the eighth note sample playing on the downbeat. The ensemble clap sample also has a16th note version where the clapping sound is played on an upbeat.

Probability

I put the eight note and 16th note samples in separate groups.[36] The 16th note samples are triggered using algorithmically increasing probability. Before level nine, the highest number of enemies spawned in a level is less than twelve, which means the generative phrase produced will take exactly two measures to play if using only eighth notes. Using 16th notes can sound too dense, especially on easier levels. Thus, I designed the 16th note samples not to occur until level nine. I achieved this effect by setting the triggering probability value to 0% for levels one through eight.

Probability is a very useful tool in algorithmic composition. It also works well here to trigger samples at random times but for a specific number of times. I had to code custom algorithms to trigger the 16th note samples, because probabilities cannot be dynamically changed within Wwise in response to real time game data.[37]

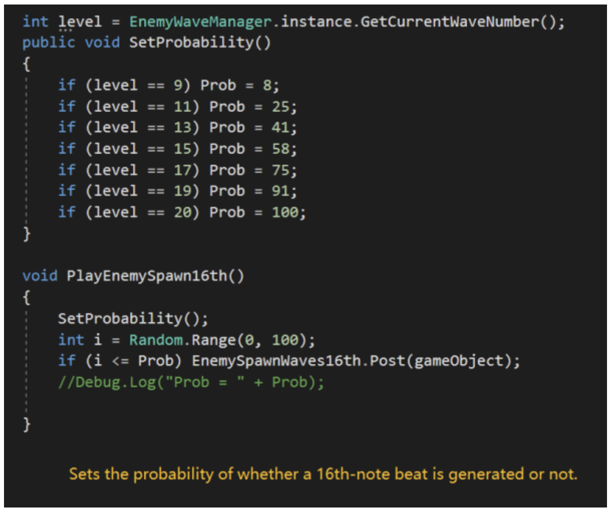
[36] From now on, eighth note samples are eighth note samples placed on the downbeat, while 16th note samples are eighth note samples placed on the upbeat.
[37] It is possible to create numbers of Wwise events each set to a specific probability value and then trigger specific events from the game engine. However, scripting a custom algorithm for probability in the game engine eliminates this unnecessary work.



```
public void PlayEnemySpawn()
{
    EnemySpawnDrumType.SetValue();
    EnemySpawnWaves8th.Post(gameObject);
    PlayEnemySpawn16th();
}
```

Figure 55 PlayEnemySpawn() function.

Each time the PlayEnemySpawn() function is called (see Figure 55 for a screenshot of this), the game engine first detects the enemy type in order that the correct samples can be chosen to play. Then the game engine will play the appropriate eighth note samples and execute the PlayEnemySpawn16th() function. Figure 56 shows the inside of PlayEnemySpawn16th() function. The integer i is assigned a random value from 0 to 100 every time the PlayEnemySpawn16th() function is called. The SetProbability() function is a list of probability values (called Prob in the code) assigned to levels that contain more than twelve enemies. If i is smaller or equal to Prob, the 16th note samples are triggered, otherwise they are not triggered.



```
int level = EnemyWaveManager.instance.GetCurrentWaveNumber();
public void SetProbability()
{
    if (level == 9) Prob = 8;
    if (level == 11) Prob = 25;
    if (level == 13) Prob = 41;
    if (level == 15) Prob = 58;
    if (level == 17) Prob = 75;
    if (level == 19) Prob = 91;
    if (level == 20) Prob = 100;
}

void PlayEnemySpawn16th()
{
    SetProbability();
    int i = Random.Range(0, 100);
    if (i <= Prob) EnemySpawnWaves16th.Post(gameObject);
    //Debug.Log("Prob = " + Prob);

}
```

Sets the probability of whether a 16th-note beat is generated or not.

Figure 56 The 16th-note samples' probabilities.

The probabilities assigned reflect the ratio of the number of enemies that exceeds twelve to the total number of twelve beat slots available for 16th notes, as indicated in Table 9. The resulting probabilities are listed in the last column while the number in parentheses is the number of 16th notes needed according to the probability result unique to that level. For instance, level nine requires thirteen EnemySpawn hits, because in addition to the twelve EnemySpawn hits played by the eighth note samples spread in two measures, one additional 16th note sample is needed. This additional sample is indicated in parentheses in the last column. Figure 57 further explains this algorithmically generated phrase by showing two example patterns that could be generated.

Level #	#'s of Enemy	Note Value Used	Types of Enemy	#'s of Timbre	16th-note Playback Probability by 12
1	5	8th	Slime	1	0
2	4	8th	Cat	1	0
3	7	8th	Slime + Cat	2	0
4	4	8th	Shooty Slime	2	0
5	9	8th	Slime + Cat + Shooty Slime	3	0
6	6	8th	Lava	1	0
7	11	8th	Slime + Cat + Shooty Slime + Lava	3	0
8	6	8th	Bomb	1	0
9*	13	8th+16th	Slime + Cat + Shooty Slime + Lava + Bomb	4	0.08 (1)
10	6	8th	Reach Cat	1	0
11*	15	8th+16th	Slime + Cat + Shooty Slime + Lava + Bomb + Reach Cat	4	0.25 (3)
12	6	8th	Ghost	1	0
13*	17	8th+16th	Slime + Cat + Shooty Slime + Lava + Bomb + Reach Cat + Ghost	5	0.41 (5)
14	6	8th	Shooty Slime 2	1	0
15*	19	8th+16th	Slime + Cat + Shooty Slime + Lava + Bomb + Reach Cat + Ghost + Shooty Slime 2	5	0.58 (7)
16	6	8th	Mage	1	0
17*	21	8th+16th	Slime + Cat + Shooty Slime + Lava + Bomb + Reach Cat + Ghost + Shooty Slime 2 + Mage	6	0.75 (9)
18	6	8th	Shooty Slime 3	1	0
19*	23	8th+16th	Slime + Cat + Shooty Slime + Lava + Bomb + Reach Cat + Ghost + Shooty Slime 2 + Mage + Shooty Slime 3	6	0.91 (11)
20*	25	8th+16th	All Enemy Types	6	1.08 (13)

Table 9 Level and music design associated with 16th note samples probability of playing.

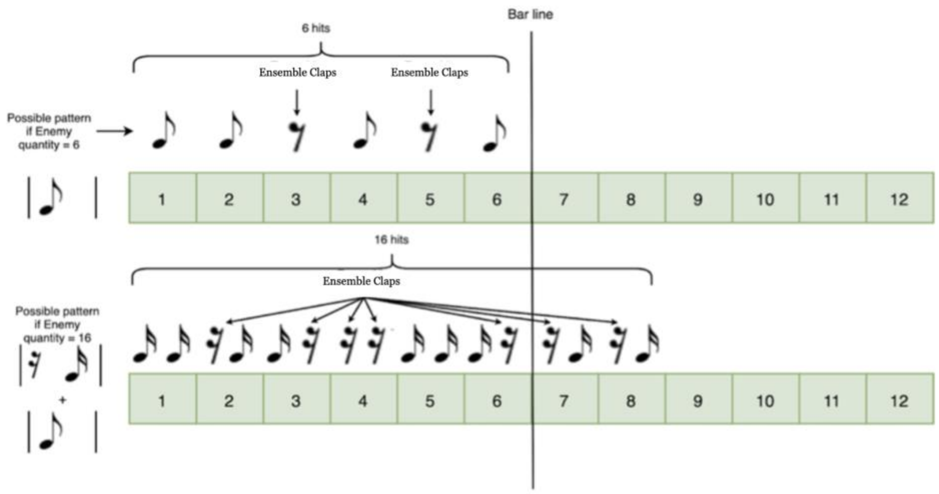
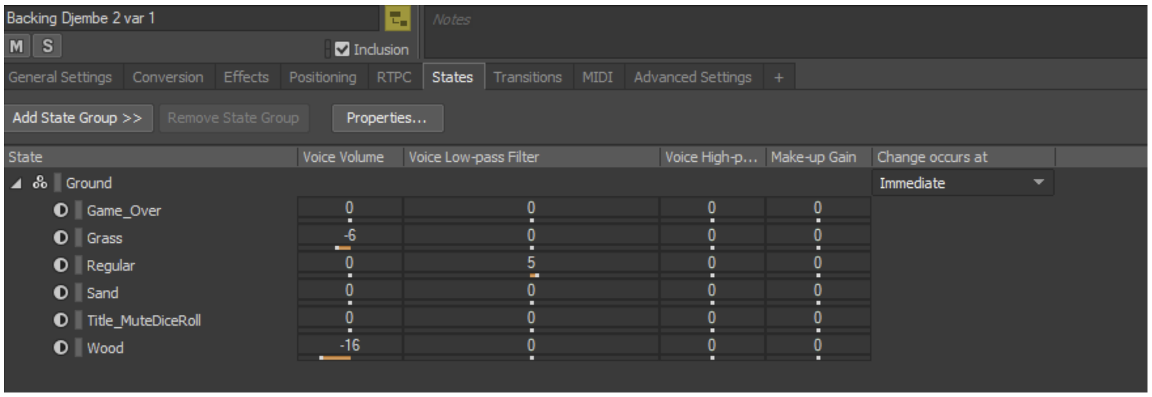


Figure 57 Two possible EnemySpawn phrases.

3.5. Real Time Signal Processing and Spatial Design

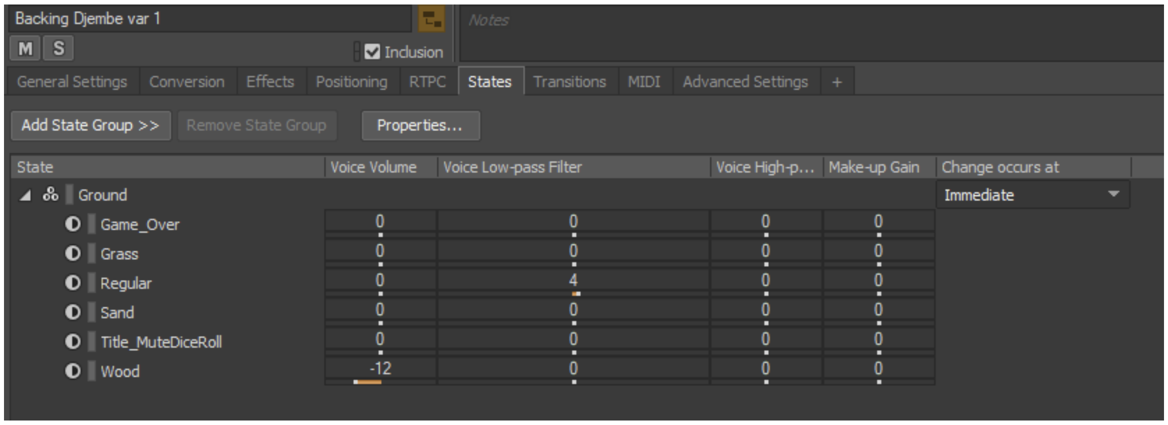
Types of signal processing used in *DOD* include volume control and filtering. Real time signal processing is interactively controlled by associated game parameters. Besides the game parameters discussed above, such as a level's duration, player's input on the joystick, enemy's events and states, and so on, parameters that primarily interact with real time signal processing and spatial design include: player's moving speed (the speed at which the dice roll), player's health, spatial location of the player, enemies, and PickUp Buffs.

Real time volume control is applied to the two ensemble *Djembes* which are interactively controlled by the player's spatial location in the game scene and the player's speed. Because the player will fail the game if they fall off the game board, when the player is on grass or wood, the volume of each *Djembe* phrase will dynamically decrease, as shown in Figure 58 and Figure 59. The closer the player stands to the edge, the more volume reduction on the *Djembe* phrases. The player hears less instrumental sound as they approach the edges of the game board. This is meant to create a sense of loss of advantage and security. Volume decrease is also controlled by the player-controlled dice's roll speed on the game board. Figure 60 is a screenshot of the project's RTPC (Real Time Parameter Control) setting in Wwise that shows the volume change to player speed relationship. The range of volume change controlled by the player's speed in real time is between 0 dB and -7 dB.



State	Voice Volume	Voice Low-pass Filter	Voice High-p...	Make-up Gain	Change occurs at
Ground					Immediate
Game_Over	0	0	0	0	
Grass	-6	0	0	0	
Regular	0	5	0	0	
Sand	0	0	0	0	
Title_MuteDiceRoll	0	0	0	0	
Wood	-16	0	0	0	

Figure 58 *Djembe* Two - volume change with ground type in real time.


Backing Djembe var 1

M S

☑ Inclusion

General Settings | Conversion | Effects | Positioning | RTPC | States | Transitions | MIDI | Advanced Settings | +

Add State Group >> | Remove State Group | Properties...

State	Voice Volume	Voice Low-pass Filter	Voice High-p...	Make-up Gain	Change occurs at
Ground					Immediate
Game_Over	0	0	0	0	
Grass	0	0	0	0	
Regular	0	4	0	0	
Sand	0	0	0	0	
Title_MuteDiceRoll	0	0	0	0	
Wood	-12	0	0	0	

Figure 59 *Djembe* One - volume change with ground type in real time.

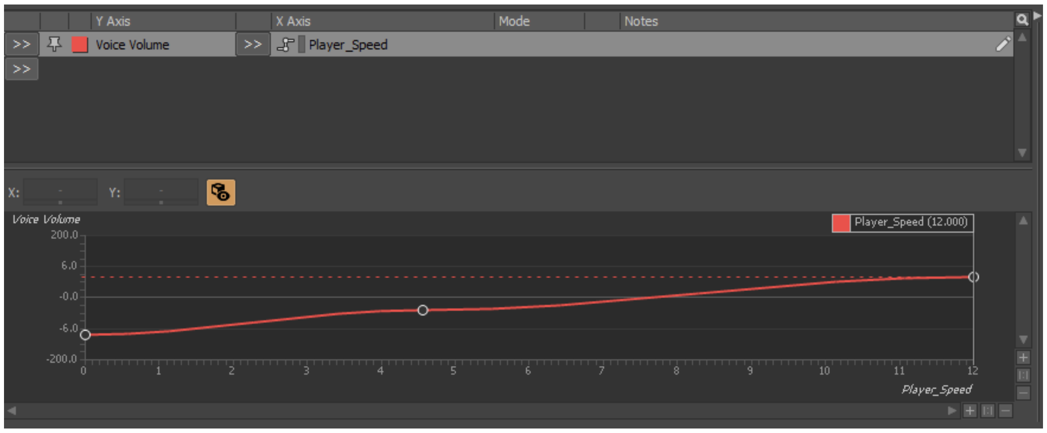

	Y Axis	X Axis	Mode	Notes
>>	Voice Volume	Player_Speed		

Figure 60 *Djembes* One and Two - volume change with player_speed in real time.

Another real time signal processing is set to the entire game music channel which can be considered as a global effect that influences all music segments. The effect is a high pass filter which will increase when the player stands on grass (+8 dB) or wood (+14 dB), as shown in Figure 61. This creates a gradual and effective acoustic shift to warn the player that they are getting close to the edge of the game board which is dangerous. A video demonstrating the ground type phrases and HPFs can be found in the demo video playlist via the link provided in Appendix D.

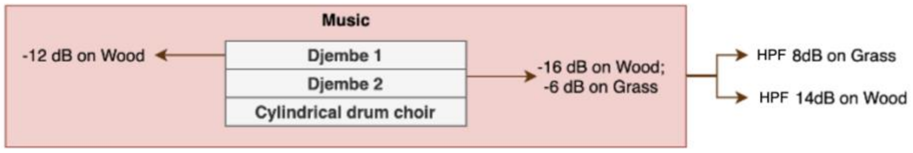


Figure 61 Real time signal processing on the backing group and the entire music bus.

In game audio design, sound designers need to set up a game object as the initial audio listener, which is where all sound should be heard from. In *DOD*, the listener is added to the player - the dice object. The *backing* group tracks of the music are largely set to be played without specific spatialization other than straight stereo sound. They maintain constant volume and pan in stereo regardless of how the listener moves. However, samples and phrases in the *GID group* are played with specific spatialization settings. The sound is emitted from where its associated game object is located, thus panning and volume are calculated based on the directional and distance relationship of the listener to the sound-emitting object. For instance, sound associated with enemies (like the sounds of an EnemySpawn, EnemyHurt, or EnemyDie event) or sound associated with PickUp items (the ScoreUp and HealthUp events) will be emitted from where these game objects are located in relation to the player's or listener's location. These settings create a dynamic sound space, and the result is that algorithmically generated phrases and beats may sound from anywhere in 360◦around the player while the *backing group* tracks always remain centered. Figure 62 shows a graphic representation of the spatial design of *DOD* in relation to the player's ears.

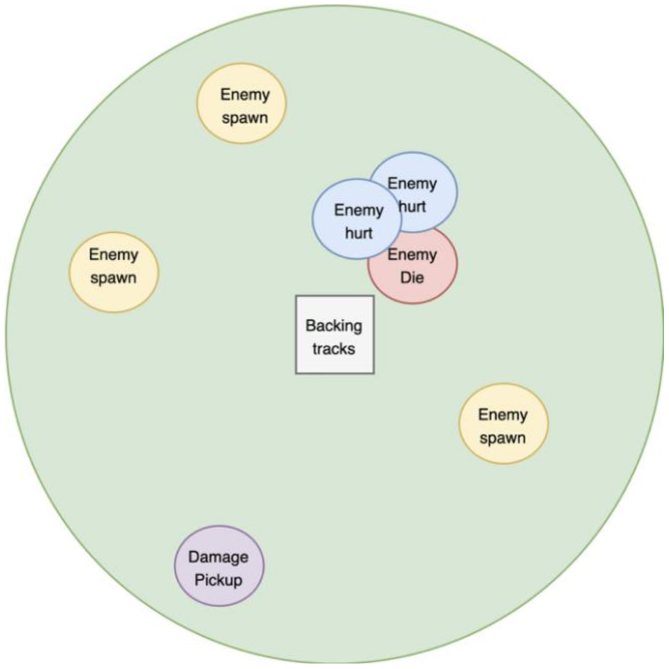


Figure 62 Sound spatialization.

3.6. Summary of Project

In this project, I have explored composing an ADM game score and denning an ADM system, and have explained ways to manipulate this system in order to create an adaptive, interactive and dynamic game score which not only closely relates to the game design, but also produces a massive amount of music variation with limited pre-produced assets. The dynamic music system of this project covered a number of algorithmic behaviors that can be implemented in a game context, some of which are mostly possible through use of the audio middleware Wwise. Still other algorithms had to be programmed with custom code in the game engine.

There are three movements. The Title and Game Over movements are pitch-based and the Gameplay movement is a non-pitch-based percussion piece. While the Title movement consists of a dynamic loop which varies with each iteration of the loop, the Game Over movement is a static loop which is played indefinitely as long as the player stays in the associated game over game state. For the three movements, the music involves a dynamic tempo and meter plan. The dynamic tempos and meters reflect both the emotional tone of the game scene as well as specific changes to the game rhythm and speed-related game states (e.g., the SpeedUp Pick-Up Buff makes the player move faster, and the SlowTime player ability makes the player move slower).

In a horizontal structure, the Gameplay movement involves four algorithmically triggered sections (the WaveInit, WaveStart, WaveBeforeEnd, and WaveEnds sections), also called the *backing group*, played by cylindrical drums. The two ensemble *Djembes* also play during Gameplay but have phrase variations which are randomly selected by the Wwise engine. In a vertical structure, there are types of one-shot samples or short phrases which are generated on top of the *backing group*, also called the *GID group*. Short rhythmic phrases are associated with different real time game states such as the EnemyDie event common to six types of enemy, the six types of Pick-up Buffs, and two player abilities. One-shot samples associated with EnemySpawn and EnemyHurt events also form generative rhythmic phrases at the beginning of each level as well as throughout each level. Real time interactive signal processing (HPF, volume, and spatialization) are also applied to the *GID group* instruments in Gameplay.

Figure 63 is a diagram that summarizes this entire project's hierarchical plan of all audio assets as well as its algorithmic procedure. All demo videos (including a screen capture video of gameplay with both music and sound effects, and a screen capture video of gameplay with only music), game soundtrack album, and the game's executables (for PC only, available in full version and music-only version) can be downloaded via the link provided in Appendix D. The full executable version of this project can also be downloaded at the official itch.io page at https://teamdead.itch.io/doordie (full version only). The game should be played with a standard game joystick.

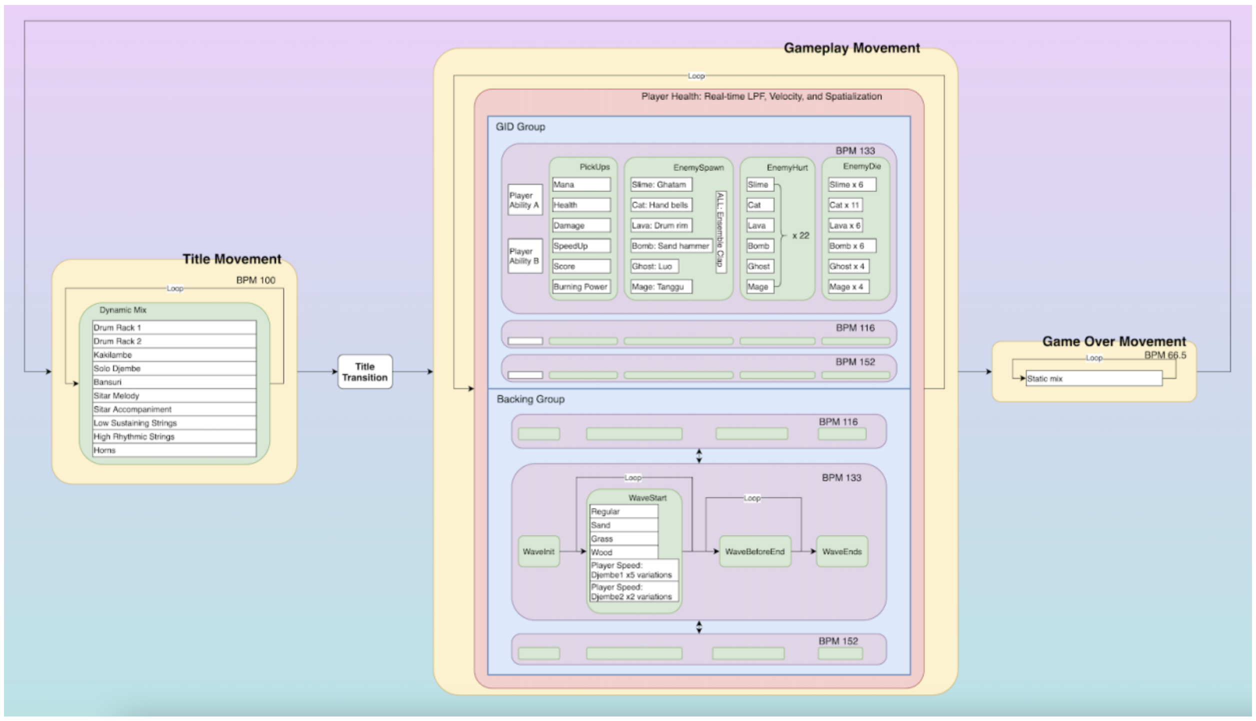


Figure 63 Summarizing diagram of the entire ADM System of *DOD*.

Chapter 4. Composing ADM - *Mastery*

4.1. Introduction

Mastery is a 2D platformer game developed using the *Unity* game engine. Soundtracks of *Mastery* shows an example of composing ADM with pitch-based musical materials. This project generates musical variations through various combinations of pitch selection, melodic phrasing, instrumentation, meter, and tonality. Music interacts with the real time gameplay by different musical parameters linked with game parameters. This game has more levels in production, but in this chapter, I will concentrate on the finished, first level, *Foothill*.

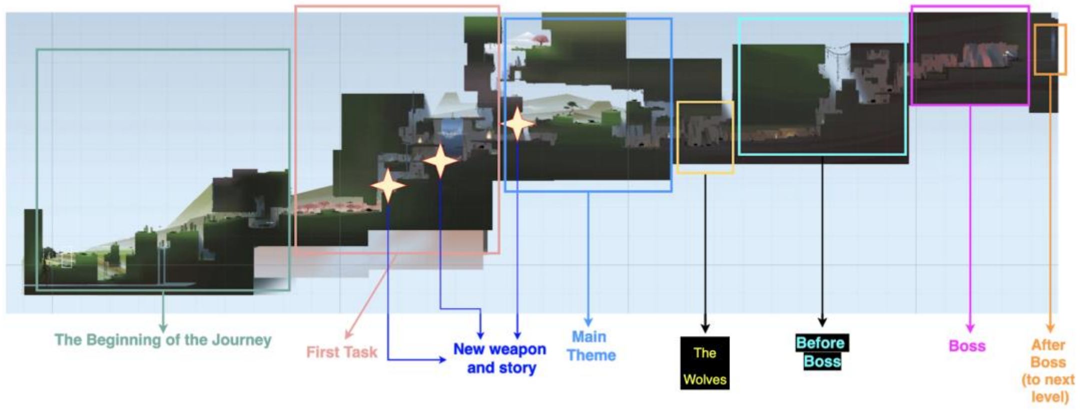


Figure 64 Overview of *Foothill* level of *Mastery* and the soundtracks' location.

4.1.1. A Multi-movement Piece

The music for *Foothill* consists of one soundtrack, *Title Theme,* for the title scene; one event soundtrack, *The Achievement*, to be triggered at the three starred places in the platform as shown in Figure 64; and six loop-based dynamic movements assigned to each area of the level:

1. *The Beginning of the Journey*
2. *First Task*
3. *Main Theme* (the *Title Theme* uses a different version of this movement)
4. *The Wolves*
5. *Before Boss*

6. *The Boss*

4.1.2. Collaboration

As the primary sound designer and composer of this project, I collaborated with a group of game developers and designers from *Tunacat Studio* from the Rensselaer Polytechnic Institute. The sound effects were recorded in collaboration with sound designer Ningru Guo at University of California-Irvine. I composed, orchestrated, and produced *The Achievement* track, as well as the first, second, and sixth movements of this project. The third, fourth, and fifth movements were originally composed by Tong Zhang, who initially invited me to join this project. Zhang provided the written score of the three movements composed as one connected piece, which I then orchestrated, made adjustments to form, reharmonized, added polyphony, and produced in DAWs. In addition to these duties, I transformed the tracks into ADM tracks, integrated all audio assets into the game engine *Unity* via the audio middleware *Wwise*, and programmed the music's interactive behavior using C# code.

Figure 65 A screenshot from the *Mastery*'s game trailer.

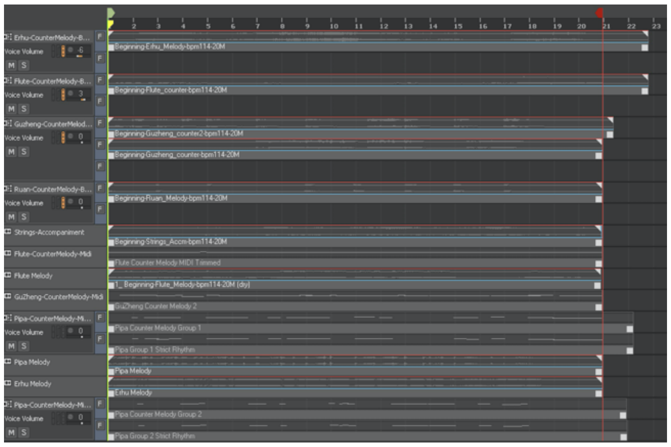
4.2. The Generative Counter Melodies - A Pitch Ranking System

4.2.1. The Beginning of the Journey

The Beginning of the Journey is the first movement of *Foothill*. In this track I explored the application of ADM in monophonic melody generation. The track starts playing when the player-controlled main character, Yan Ching, is initially spawned. It has 20 measures that loop indefinitely as long as the player remains in the initial game area. The orchestration involves three groups of instruments that are differentiated by their roles in the piece: primary melodies, counter melodies, and accompaniment played with sustaining notes on orchestral strings. The primary melodies and the string accompaniment are imported as audio files. The string accompaniment serves as a supportive foundation while the primary melodies serve for introducing a memorable, recurring theme of this movement.

The counter melodies divide further into two subgroups - one with looping audio files similar to the primary melodies and string accompaniment, the other with an algorithmically generative melody that uses three MIDI tracks to trigger instrumental samples in Wwise. The generation of each note relies on the player's input data, as well as the note generated prior. This algorithmic melody utilizes a first order Markov chain type of data generation, meaning that the value of a node is closely affected by the nearest node selected before.[38] Numerous existing studies and creative works provided inspiration to the application of Markov chain procedures in the design of generative melodies in this movement (Ames, 1989; Chiricota & Gilbert, 2007; Clinton, 2019; T. Collins et al., 2011).

[38] This measurement of the generation is also referred to as nodes in a Markov Chain composition, and the resulting matrix of all probabilities and possible movement between nodes can be demonstrated with a suffix trie. A Markov chain consists of a finite number of states, a probabilistic transition function that indicates the probability of making a transition from one state to another, and an initial state. Each state in the model may also have an output, or emission (Verbeurgt, Dinolfo, Fayer, et al., 2004).


Erhu-CounterMelody-B...
Voice Volume
Beginning-Erhu_Melody-bpm114-20M
Flute-CounterMelody-B...
Voice Volume
Beginning-Flute_counter-bpm114-20M
Guzheng-CounterMelod...
Voice Volume
Beginning-Guzheng_counter2-bpm114-20M
Beginning-Guzheng_counter-bpm114-20M
Ruan-CounterMelody-B...
Voice Volume
Beginning-Ruan_Melody-bpm114-20M
Strings-Accompaniment
Beginning-Strings_Accm-bpm114-20M
Flute-CounterMelody-Midi
Flute Counter Melody MIDI Trimmed
Flute Melody
1_ Beginning-Flute_Melody-bpm114-20M (dry)
GuZheng-CounterMelody-Midi
GuZheng Counter Melody 2
Pipa-CounterMelody-Mi...
Voice Volume
Pipa Counter Melody Group 1
Pipa Group 1 Strict Rhythm
Pipa Melody
Pipa Melody
Erhu Melody
Erhu Melody
Pipa-CounterMelody-Mi...
Voice Volume
Pipa Counter Melody Group 2
Pipa Group 2 Strict Rhythm
Figure 66 Full mix view of *The Beginning of the Journey* in Wwise

Besides the pitch generation in the generative counter melodies, the mixing of instrumental layers also interacts with the player's real time input. I will focus on explaining the generative counter melodies in this section and discuss the interactive instrumentation in 4.2.5. Dynamic Instrumentation and Sequencing. A screenshot of all stems in the mixer view of Wwise is shown in Figure 66.

4.2.2. Algorithmic Rules for Harmonious Generation

In order to make the generative melody work, the first step is to design a clear way to translate musical information into information that the computer can understand and manipulate, such as numbers. Thus, I created a pitch ranking system in which every note in the generative melody can be represented by a number. Then, by setting rules and transition conditions for the Markov chain generation, the computer continuously generates the "next number," which then triggers the "next note." After the number and the musical note is decided, Wwise will play the audio sample of the chosen note.

The rules of generation must follow the principle of functional harmony theory in order to have the generated melody stay in tune with the track's tonal and harmonic plan heard in the rest instrumental groups (primary melodies, string accompaniment). I created five rules to enable the melodic generation algorithm to achieve that goal:

I. Predetermined Pitch Collection.

The generation algorithm needs to choose a note from a pitch collection (PC) that is harmonious with the underlying harmony. This step defines the tonality of the generative melody and makes sure it follows the harmonic progression of the track. Using predetermined PCs filters out pitches that are outside of or do not exist in the harmonies at an early stage of the melody generation. Every harmony in this movement has an associated PC from which melody notes are chosen. The algorithm also needs to dynamically select the correct PC in real time while the static instrumental groups (primary melodies, strings accompaniment, etc.) continue looping. This pitch-filtering strategy is necessary for voice-leading so that the melodic intervals between notes remain diatonic, regardless of what pitch ranking number is generated. As a crucial step of this generative melody, predetermining these pitch collections puts limits to the Markov chain process in order to achieve the intended generation result.

II. Chordal Position.

Each PC contains all chordal tones or notes that sound consonant to the underlying chord. Although each PC might contain different notes, the position of the notes in relation to the chords are similar, such as the chordal root or seventh. Additionally, there are common intervals that exist in every harmony such as intervals of third, fourth, and fifth. Thus, the ranking numbers should also reflect the notes' universal positions in their associated chords. For instance, the chordal third in A minor triad and B minor triad, C, and D can be labeled with the same number. This rule also aids the manipulation of voice-leading among the different harmonies. Thus, pitches having closer ranking numbers will form smaller melodic intervals in voice-leading and vice versa.

III. Range of Ranking Numbers.

Some harmonies may contain more notes than the other. For instance, a seventh chord has more notes than a triad. Thus, the biggest number for labelling notes is determined by the harmony that has the most pitches in a piece.

IV. Register and Range.

The generated melody should have a predetermined range and register. Such specifications are typical for acoustic instrumental compositions because every instrument has its best performing range and register, especially monophonic instruments. I set the melodic range to a maximum of two octaves, while some harmonies stay within one octave. Because of this pitch range, some PCs have more notes than others, which effects and expands the third rule further. In some previous works of generative and algorithmic compositions when the generation process does not specifically limit the resulting musical notes to stay within a certain register and range, the resulting music may be less useful for the purpose of a game score, such as the one shown in Figure 67 (Verbeurgt, Dinolfo, Fayer, et al., 2004). For this movement, the music has multiple layers of instruments while only the counter melodies are algorithmically generative; a suitable register and range separates the generative melody from the other layers and brings clarity of the polyphony to the ears.

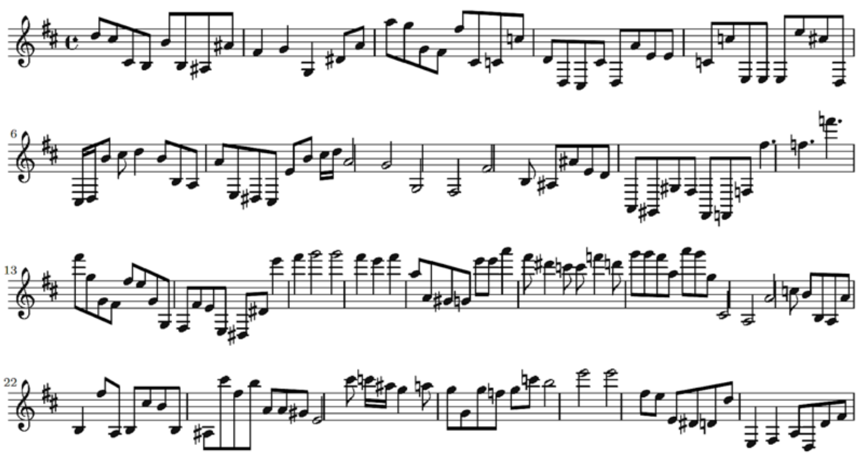


Figure 67 A passage of music generated by a Markov chain[39]

[39] Reprinted from Verbeurgt, K., Dinolfo, M., & Fayer, M. (2004). Extracting Patterns in Music for Composition via Markov Chains. In B. Orchard, C. Yang, & M. Ali (Eds.), Innovations in Applied Artificial Intelligence (Vol. 3029, pp. 1123–1132). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-24677-0_115

The register chosen for the counter melodies in this movement ranges from E4 to B5 for the Chinese flute *Xiao*, and from E3 to B4 for *Guzheng* and *Pipa*.[40] Thus, harmonies that stay within the range of one octave have fewer notes contained in its associated PC than the harmonies that stay within the range of two octaves.

V. Non-chordal Tone PCs.

Finally, if there are PCs of chordal notes, there should also be PCs of non-chordal notes. Looking into the classical repertoire of tonal music, melodies consist of not only notes that sound consonant to the underlying harmony, but also non-chordal tones which may serve as passing, neighboring, or more vivid intervallic leaps that fill between chordal tones. In many standard tonal compositions, we hear non-chordal tones placed on upbeats--though not always, such as the example of an appoggiatura.

During the progression of the harmonies, a note may have constantly changing roles. For instance, in the V_7-I progression in C minor, a Eb note may be considered non-chordal to the G dominant 7th chord but chordal to the tonic chord. Hearing the Eb note before the arrival of the tonic chord could potentially break the musical tension, since the expected harmonic resolution arrived too early. Therefore, I also intentionally avoided having an early resolution happen in the generative melody in this movement, which I achieved by a static-note strategy that will be discussed later in this chapter.

4.2.3. The Pitch Collections (PCs)

Before “upgrading” into an ADM game score, this movement was composed as a linear composition - like any traditional composition composed for playing from the beginning to end in live performances - with static materials written on a score. The linear static piece had no dynamic content. An excerpt of the original score is shown in Figure 68. It served as a good starting point for

[40] Since samples of the *Guzheng* and *Pipa* are exactly one octave lower than the *Xiao*, in later chapters we will use *Xiao* as a reference, but the discussion applies to both *Guzheng* and *Pipa* as well.

building the ADM version. The original melody played on the first *Xiao* became the melody for the *primary melodies* instrument group as introduced in 4.2.1. *The Beginning of the Journey*. I also extracted the structure of the static melody and used it to define the rhythm and PC pattern of the generative instrument group *counter melodies*.

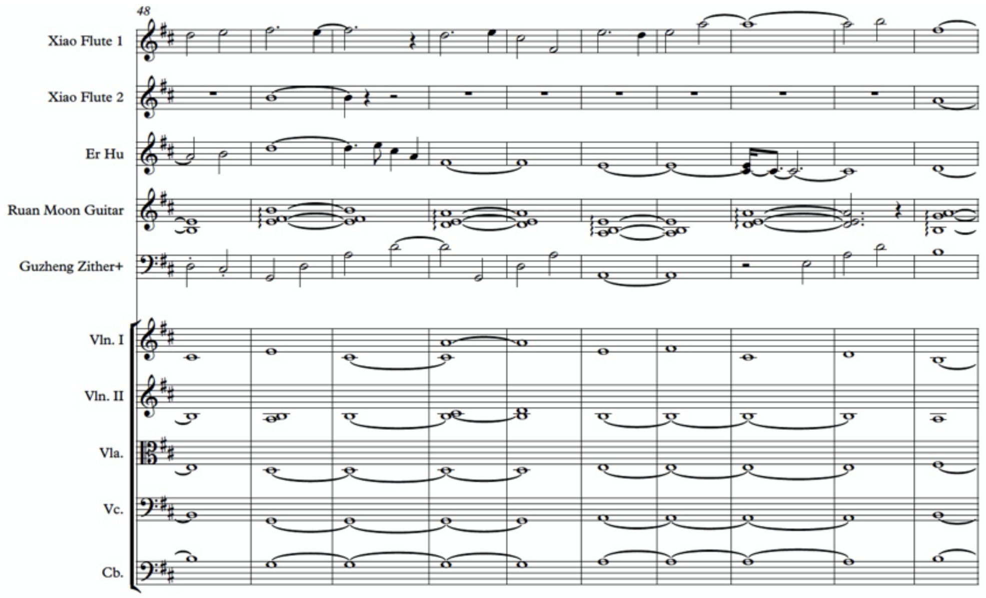


Figure 68 An excerpt from the static score of the movement *The Beginning of the Journey*.

A score showing the transformation of the original melody to the generative melody is shown in Figure 69. It lays out the step-by-step process of how the original melody became the final MIDI tracks representing the pitch ranking system. Above the staff is the harmonic progression in Roman numerals. There are five harmonies in this segment - i^{11}, $\text{ii}^{\emptyset 42}$, VI^{sus}, VI^{13}, and VII^{9}. Beneath the first staff, the *Original Melody*, are the *Pitches in Chordal* and *Pitches in Passing,* which represent the PCs of the chordal and non-chordal notes according to the five harmonies of the original melody. Each harmony has both a chordal PC and a non-chordal PC, resulting in a total of ten PCs as listed in Table 10. The next two staffs in Figure 69 represent all notes within the predetermined register and the two-octave range.

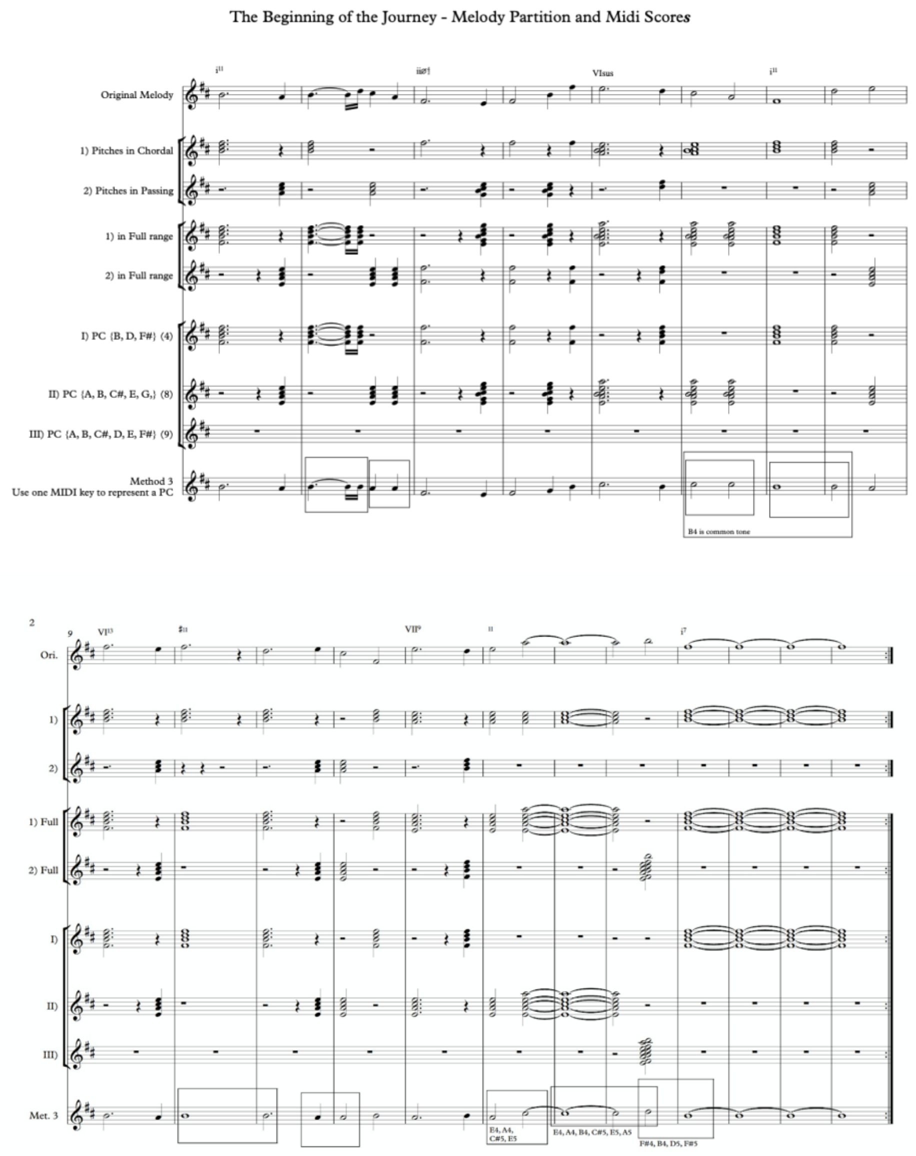


Figure 69 Transformations of the Original Melody into the generative melody.

It drew my attention that the chordal and non-chordal PCs in this track share many common tones. Some PCs are even exactly the same. As demonstrated by the highlighted letters in Table 10, for instance, the non-chordal tone PC of i_{11} chord has two notes (C#, E) in common with the chordal tone PC of the adjacent $\text{ii}_{ø42}$. The two notes are also held in common in the chordal tone PC of VII_{9} and the non-chordal tone PC of VI_{13}. This discovery led me to integrate and reorganize the ten chordal and non-chordal PCs into a reduced quantity of only five PCs. The “chordal” or “non-chordal" labels no longer matter because more important are the actual pitches included. Having fewer PCs can simplify the implementation procedures, save storage space, and make the computer algorithm more efficient.[41] Figure 70 shows the original implementation of the ten PCs scheme.

Chord#	**Chord type**	**Chordal**	**Non-Chordal**	**Absent Notes**
1	**i**^{11} (mm.1-2)	B, D, F#	A, C#, E	G;
2	**ii half dim 42** (mm.3-4)	C#, E, G, B	D, F#	A
3	**VI**^{sus} (mm.5-6)	B, E, A, C#	D, F#	G
1	**I**^{11} (mm.7-8)	B, D, F#	A, C#, E	G;
4	**VI**^{13} (mm.9-12)	G, D, B, F#	A, C#, E	
5	**VII**^{9} (mm.13-16)	A, C#, E	B, D, F#	G
1'	**I**^{7} (mm.17-20)	B, D[42]	A	C#, E, G, F#

	Maximum Pitch Quantity and Sets	4
		{a, b}
		{a, b, c}
		{a, b, c, d}
			3
			{a}
			{a, b}
			{a, b, c}
				N/A

Table 10 Ten PCs on the five chords of the harmonic progression.

[41] When experimenting with the original ten-PCs scheme, there were additional considerations and applications considered, including: last note of a phrase can land on either a chordal 3rd or the root creating different types of cadences (PAC/IAC); avoiding early resolution of the Tonic and resolution preparation, such as the note generated before the cadence note should be close to the root or chordal 3rd, thus creating a smooth resolution; probabilities in choosing sample variations; programming MIDI notes onto different MIDI channels as the sampler function in Wwise can take multi-channel MIDI info. This approach was later abandoned because the five-PCs scheme can do all these tasks with a much simpler programming process and execution procedure.

[42] On cadences, only the root and 3rd are kept.

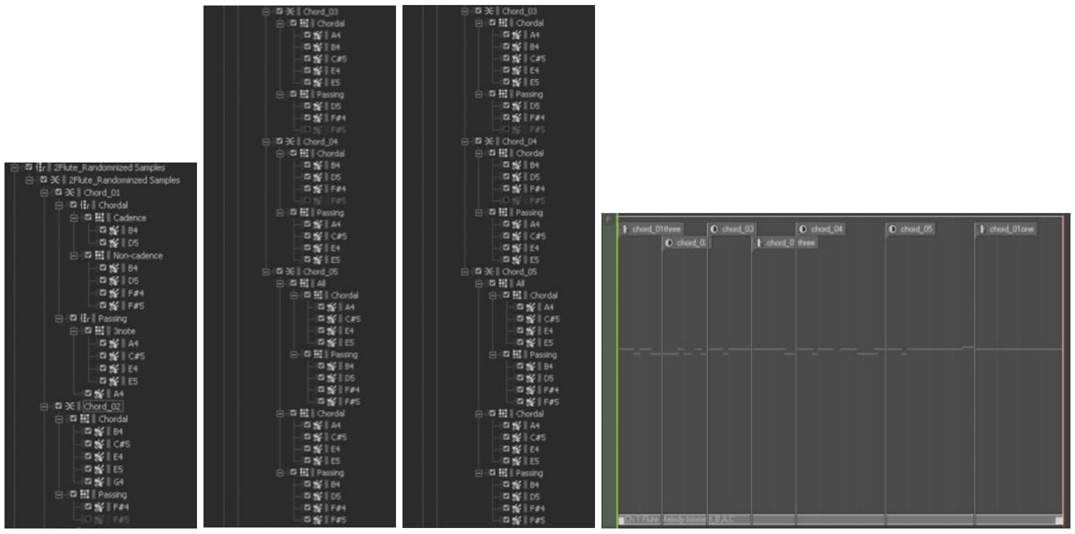


Figure 70 The original ten-PCs implementation scheme.

The next three staffs on Figure 69 show the final five PCs used. The first and third of the three staffs each show one of the PCs, while the second line represents the remaining three PCs. The first staff represents the B minor chord - {B, D, F#}. The second staff contains five pitches, forming an A-dominant 11th chord - {A, C#, E, G, B, D}.[43] The third staff shows a PC of all pitches in the above two combined - {G, A, B, C#, D, E, F#}. These three lines seem similar to the previous two lines, but they are actually dramatically different in principle and implementation.

Looking closely at the pitches on the middle line of the three staffs, the {A, C#, E, G, B, D}, the cluster of G and A together is harmonically less pleasant.[44] Consequently, I broke them into subsets, which became the remaining three PCs: {A, C#, E}, {C#, E, G, B}, and {C#, E, A, B}. I then implemented the five PCs in Wwise (Figure 71).

[43] The dominant 11th chord here only represents pitches included in the PC; it does not have dominant harmonic function in this movement.
[44] It forms a {GABC#} whole tone scale, which could be less appropriate for the style of this project.

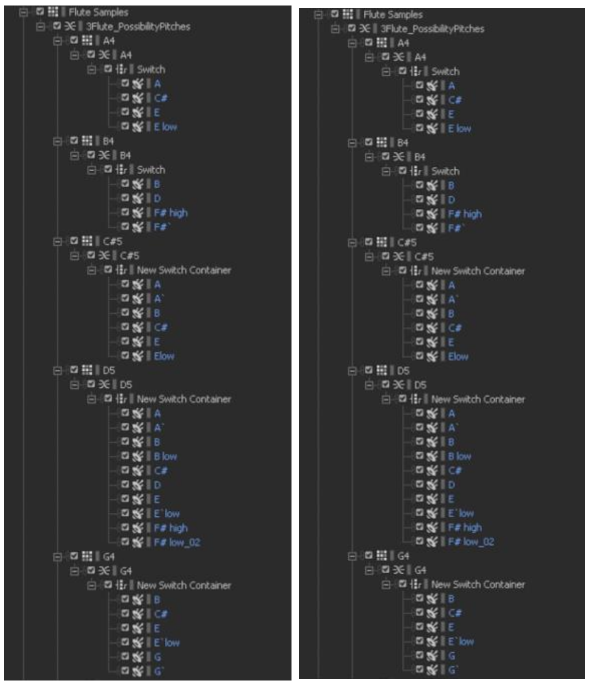


Figure 71 Five PCs implemented in Wwise.

Each group of audio samples representing the five PCs has a parent folder in Wwise that are named G4, A4, B4, C#5, and D5, as shown in Figure 71. These folder names are the MIDI notes used for triggering the samples to play. The bottom staff in Figure 69 shows the complete MIDI melody that triggers the five PCs' samples in the pattern extracted from the Original Melody. Although the generative pitches are dynamic, the rhythm remains static as programmed in the MIDI track. The rhythm strictly copies the rhythmic pattern extracted from the Original Melody, as shown in the score. However, later I applied two additional rhythmic patterns for the *Guzheng* and *Pipa*. I added two additional MIDI notes in mm.3-4 of the MIDI track, F#4 and E4, to trigger only static pitches of F#4 and E4. This small static segment will sound exactly the same every time when the track loops, thus creating a memorable, recurring motif like a musical "landmark" amid the dynamic materials in the loop. This technique of a short static motif works well at musical cadences as well if composers want the generative melody to end on a specific note. The score shown in Figure 72 shows two possible iterations of the melodic generation. The MIDI track imported to Wwise is shown in Figure 73.

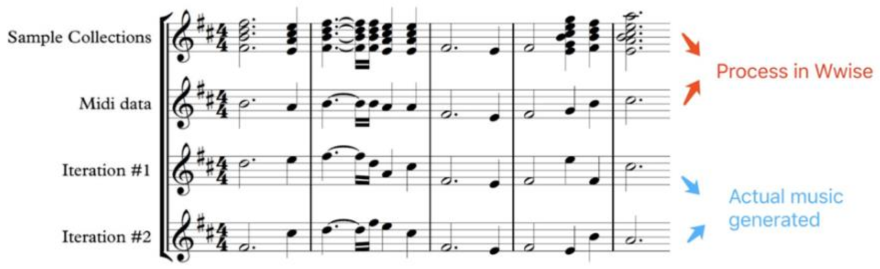


Figure 72 An example of two melody generation iterations.

Figure 73 MIDI track with rhythmic pattern that triggers the audio sample associated with the PCs.

4.2.4. Pitch and Voice-leading

All pitches in all the PCs are labeled with numbers that function as a ranking system. The algorithm utilizes these numbers to interact with the player's real time input data and play the pitches associated with the numbers. Besides pitches, the player's input also influences the voice-leading by controlling the change in ranking numbers being sent from the game engine over time.

	E4	*F#4*	*G4*	*A4*	*B4*	*C#5*	*D5*	*E5*	*F#5*	*G5*	*A5*	*B5*
D5PC	01	02	■	03	04	05	06	07	08	■	09	10
G4PC	01	■	02	■	04	05	■	07	■	08	■	■
A4PC	01	■	■	03	■	05	■	07	■	■	■	■
B4PC	■	02	■	■	04	■	06	■	08	■	■	■
C#5PC	01	■	■	03	04	05	■	07	■	■	09	■

Table 11 Ranking numbers assigned to all PCs based on D5PC.

As discussed previously, only notes that appear in the tonality of the movement need labels, and the maximum number to be labeled depends on the PC that contains the most notes. I composed *The Beginning of the Journey* in the key of D-Gong Qing Yue Yu Heptatonic Scale (太簇宫清乐羽调式)[45], which is equivalent to the B aeolian scale in Western music theory, though the notes have different roles in the two music traditions. As shown in Figure 74, the pitches in the key - B, C#, D, E, F#, G, and A - validate the five PCs. Within the predetermined two-octave range of the generative melody, the labeled notes expand to 12 notes - E4, F#4, G4, A4, B4, C#5, D5, E5, F#5, G5, A5, and B5. Since the PC in m.16 assigned to the D5 MIDI note (D5PC[46]) has a total of ten notes, the maximum number used in the ranking system will be ten. The notes in the PC will each get a number ranging from one to ten, as shown in the D5PC row in Table 11.

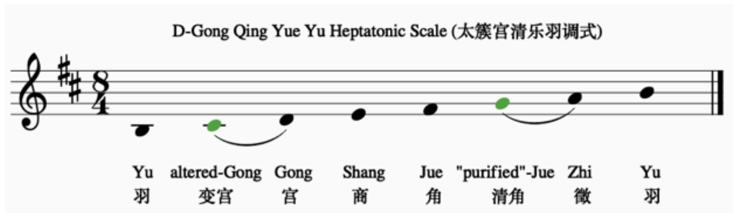


Figure 74 The scale used in *The Beginning of the Journey*.

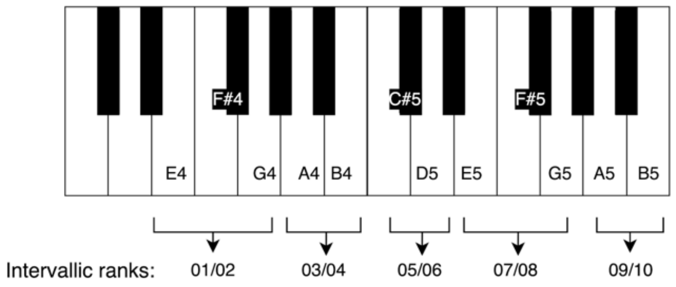


Figure 75 Ranking numbers assigned to all PCs based on D5PC - Keyboard view.

[45] In traditional Chinese music, the note D is called Tai Cu (太簇). Qing Yue(清乐), Ya Yue(雅乐), and Yan Yue(燕乐) are three types of heptatonic scales that include two altered notes added to the regular pentatonic scales. A Qing Yue (清乐) scale contains the altered-Gong (变宫), which is a minor second below the tonic note (Gong), and the Qing Jue, translated as "purified"-Jue (清角), which is a minor second above the scale degree three (Jue). Gong, Shang, Jue, Zhi, Yu are names of the notes in a pentatonic scale. The "Yu" (羽) represents the fifth note of the scale, for instance, the B note in a D pentatonic scale. When the note names are presented in the specific name of a scale, it tells the mode of the scale that the music is in. In other words, a D-Gong Yu pentatonic scale means the fifth mode of the D pentatonic scale that starts on B instead of D.
[46] From now on, I will use the assigned MIDI note to name the PCs.

Below the D5PC row in Table 11 are the ranking numbers assigned to notes of all other PCs. Since G (G4, G5) and A (A4, A5) notes never appear in the same PC, they share the same ranking number assigned to F# (F#4, F#5) and G (G4, G5). The logic behind is that their intervallic distance is the smallest (a minor second) thus having the same number label would have little impact on the control of voice-leading. A keyboard view of the ranking numbers assigned to all PCs is shown in Figure 75.

Some notes do not exist in other PCs and are therefore blacked out in Table 11. When there aren't sufficient notes in a PC for the ten numbers, the numbers will be spread out so that one note can have multiple numbers assigned. For instance, the G4PC does not contain the note A4, so B4 gets both 03 and 04 labels. If the game engine sends a "03" message while the music engine is currently playing the G4PC, the music engine will play the audio sample of pitch B4. When the music engine is playing the static segment of E4PC and F#4PC, any number will trigger the notes E4 or F#4, because they are set to be triggered by any ranking numbers. A table and keyboard view of the ten ranking numbers spread across all notes is shown in Table 12 and Figure 76. The original manuscript of the ranking system design is shown in Figure 77. In the manuscript, the bottom row in each box shows the MIDI note that a PC is associated with. The left columns show the pitches contained in the PCs, and the right columns show their assigned ranking numbers.

MIDI Note PCs	PITCHES INCLUDED (from low to high)	ASSIGNED RANKING NUMBERS
E4PC	{E4}	{all}
F#4PC	{F#4}	{all}
G4PC	{E4, **G4**, B4, C#5, E5, **G5**}	{01, 02, 03/04, 05/06, 07, 08/09/10}
A4PC	{E4, A4, C#5, E5}	{01, 02/03/04, 05/06, 07/08/09/10}
B4PC	{F#4, B4, D5, F#5}	{01/02, 03/04/05, 06/07, 08/09/10}
C#5PC	{E4, A4, B4, C#5, E5, A5}	{01, 02/03, 04, 05/06, 07/08, 09/10}
D5PC	{E4, F#4, A4, B4, C#5, D5, E5, F#5, A5, B5}	{01, 02, 03, 04, 05, 06, 07, 08, 09, 10}

Table 12 Ten ranking numbers spread and distributed to notes of all PCs.

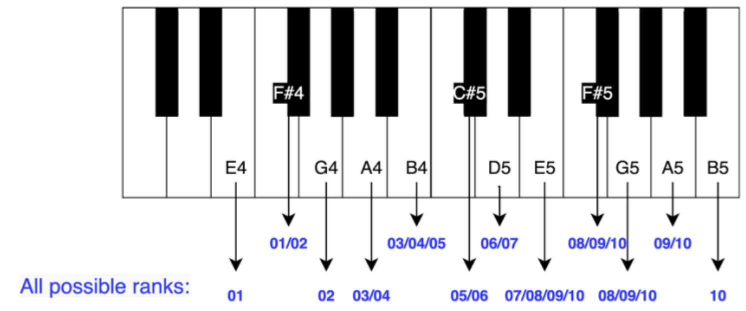


Figure 76 Ten ranking numbers spread and distributed to notes of all PCs - Keyboard view.

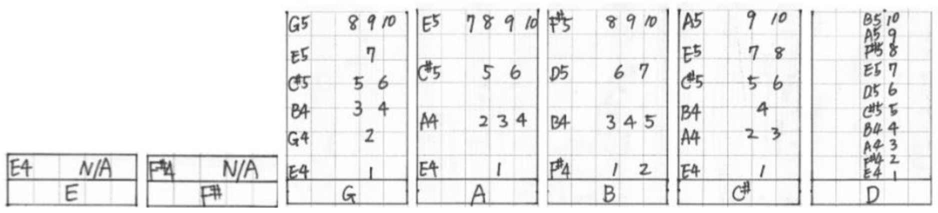


Figure 77 Original manuscript of the design of the ranking system.

Ranking System Testimony

In order to test the performance of this ranking system and its exact influences on voice-leading, I counted and collected all possible types of melodic intervals that could occur as well as their potential quantities of occurrence in one iteration of the melody generation. The voice-leading tendency and pattern of this ranking system validate my original intention of using it for real time melody manipulation.

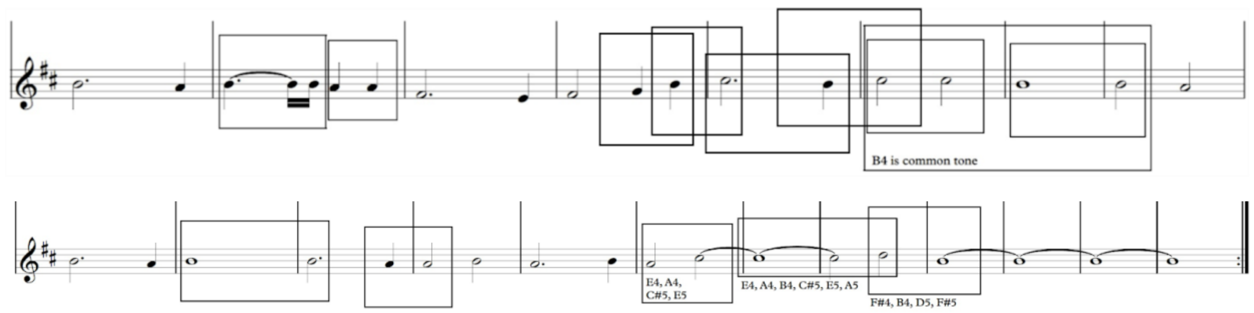


Figure 78 Boxed MIDI notes showing possibilities of unison melodic intervals.

I started counting from the smallest melodic interval, a unison, and gradually moved up to the widest interval. I then calculated how many times each type of interval may occur in this looping segment. Since every MIDI note involves a pitch generation, there are a total of 31 generation events in this loop. Between the 31 generated notes are 30 melodic intervals. MIDI notes boxed in pairs in Figure 78 share common tones between their associated PCs which implies that a unison may occur in the voice-leading. The unison intervals undoubtedly may occur when the same PCs are being repeated, such as the first half or the second half of m.2 (B4PC to B4PC, or A4PC to A4PC). A unison may also occur at five additional places when the two PCs are different. They are B4 to C#5, G4 to B4, A4 to C#5, C#5 to D5, and D5 to B4. This means that 14 out of 26 dynamic voice-leading events[47] can potentially generate a unison melodic interval.

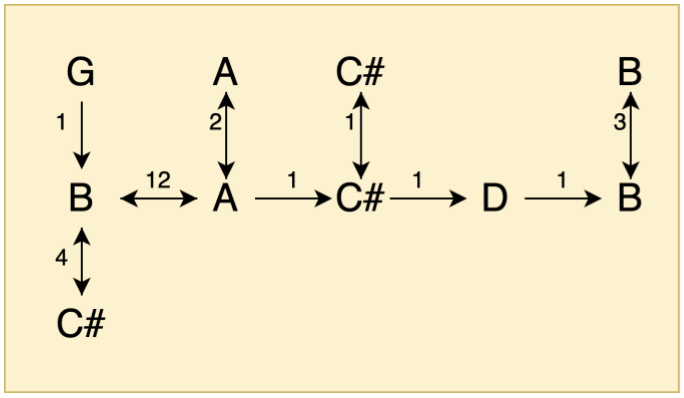


Figure 79 All voice-leading events and their number of occurrences.

I created a diagram to represent the 26 voice-leading events (Figure 79). In the figure, the letters represent the MIDI notes with their associated PCs. The arrows represent that melodic interval may occur between the two PCs. Some melodic intervals only move in one direction, while some move in both directions. Numbers next to the arrows show the total number of occurrences (out of 26) of melodic intervals in that segment. For instance, voice-leading between G4PC to B4PC only happens once in this entire segment, as written in the second half of m.4 of the score (Figure

[47] The sum voice leading places excludes the four melodic intervals happening in mm.3-4 (A4 to F#4, F#4 to E4, E4 to F#4, F#4 to G), because F#4 and E4 play static pitches not dynamically affected by the player's real time input.

78), and the direction of the voice-leading goes only one-way from G to B. Among all voice-leading events, only B4PC and A4PC have no common tones. This means even if the game engine sends the same ranking number message when these two PCs are playing consecutively, a unison melodic interval would never occur.

Without breaking the predetermined melody range of two octaves, some PCs have a much narrower pitch range. The B4PC and A4PC contain only one octave of notes, and other PCs such as the G4PC have slightly wider range than one octave. This is because I intended to avoid having the generative melody go beyond G5 in most generation events (except C#5PC and D5PC). Thus, while still preserving room for generation diversity, this restriction helps shape the contour of the melody, causing it to follow a semi-determined trajectory. For examples, by having wider range in C#5PC and D5PC, the generated melody has a greater chance of reaching a higher point on those two chords.

The ranking numbers are associated with audio samples of the specific pitches in the PCs. For instance, if the game engine keeps sending "10" during mm.5-7, the generated melody will be A5-F#5-A5-A5-F#5, and the voice-leading intervals are minor 3rd(m3)-m3-unison-m3. Since the ranking number remains the same, the resulting melodic interval is also relatively unchanged, except for the A5 being repeated in the same PC. If the game engine sends a string of different numbers such as "67429", to the music system during mm.5-7, the generated melody will be C#5-D5-B4-A4-F#5 and the voice leading intervals are m2-m3-M2-M6. Both example melodies are shown in Figure 80.

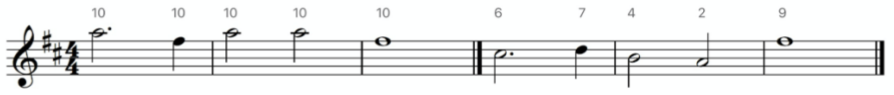


Figure 80 Numbers sent from game engine and the generated melodies in mm.5-7.

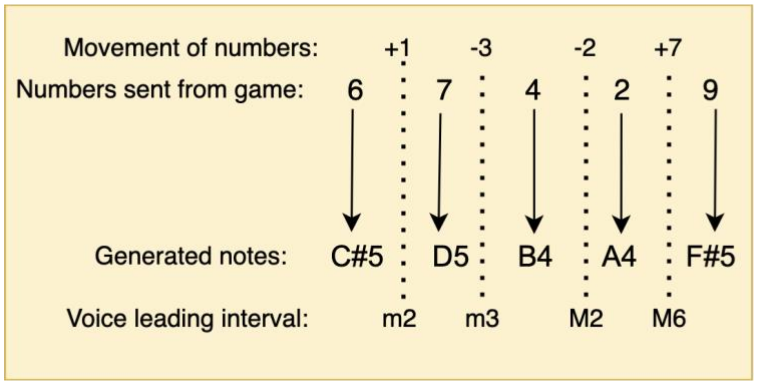


Figure 81 Relationship between the movement of the numbers and the voice leading.

The advantage of the ranking system lies not only in the ability to assign specific numbers to musical notes and trigger the samples, but also in the capability of adding mathematical formulas or calculations to control the change of numbers over time, thus controlling the voice-leading. The differences between ranking numbers sent from the game engine meaningfully reflect the resulting melodic intervals, just like the second melody in Figure 80. Among the five numbers of "67429", the four number differences, +1, -3, -2, and +7, communicate the melodic intervals, m2, m3, M2, and M6 and the direction in which they are moving (Figure 81). Therefore, this ranking system can manipulate tonal voice-leading in a generative melody effectively.

Using the ranking system to manipulate voice leading still contains indeterminacy to some degree. For example, a "+2" may result in any interval from a unison, m2, M2, or m3, depending on which underlying harmony is currently playing. However, distances of ranking numbers have strong tendencies in forming certain types of intervals. I created a chart summing all melodic intervals embodied in this MIDI track, which is included in Appendix A. It summarizes differences in ranking numbers from zero to nine.[48] Each column in Appendix A, from the left to right, displays the rank difference, types of intervals generated on which MIDI-note PC the interval happens, total possible occurrence in the segment (out of 26), and a pie diagram showing the occurrence

[48] In the movement, I avoided using differences of ranking numbers greater than nine because the maximum ranking number used in this piece is ten, the maximum difference in ranking number will be nine. Also, ranking numbers larger than a distance of nine start to sound less natural, as it creates large intervallic leaps in the melody.

probability. The chart includes both up and down movement in voice-leading, represented as "+" and "-" in the change of ranking numbers, strictly following the score.

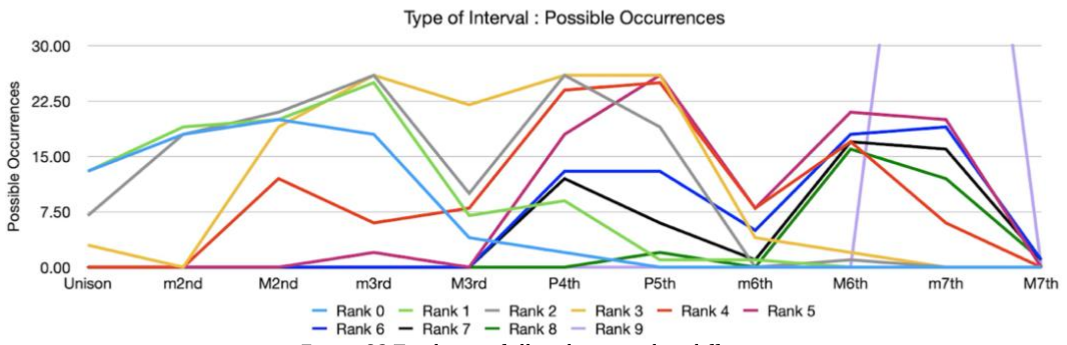


Figure 82 Tendency of all ranking number differences.

When ranking number differences get larger, the resulting distances of the melodic intervals get larger as well. For instance, ranking numbers with a distance of "0" has 75% chance of generating either a minor 2nd, major 2nd or minor 3rd, while ranking numbers with a distance of "6" has 61% chance of generating either a minor 6th, major 6th, or minor 7th and 28% chance of generating either a perfect 5th or 4th. Since we initially filtered pitches of all PCs to the tonality, the qualities of these melodic intervals become less important, as they automatically follow the diatonic plan. Starting from ranking numbers with a difference of "5" and beyond, the occurrence of melodic intervals smaller than perfect 4th significantly drops in probability. A summarizing graph laying out all ranking number differences and how intervals shift is shown in Figure 82.

4.2.5. Dynamic Instrumentation and Sequencing

Among the three groups of melodies - primary melodies, generative counter melodies, and non-generative counter melodies - each group has three instruments. Different instrumentations are associated with different weapons the player uses in the gameplay. The *Foothill* level includes three unlocked weapon types from the Five Elements: Earth, Water, and Wood. The dynamic instrumentation switches with smooth crossfades executed by Wwise states that are triggered by the game code.

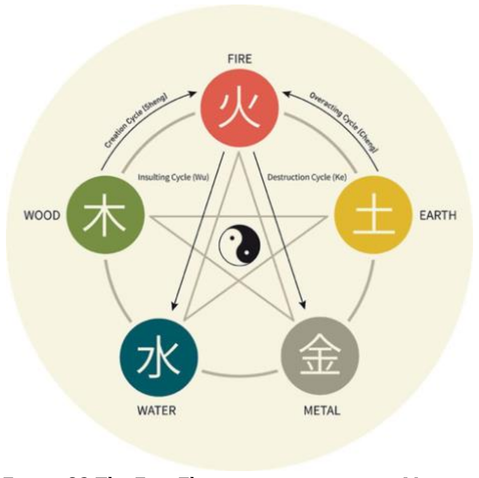


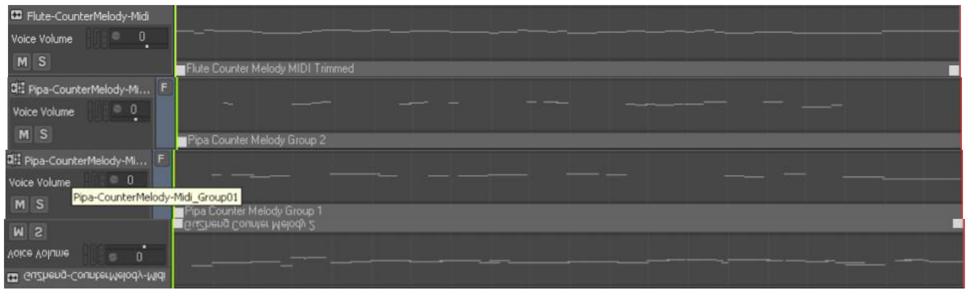
Figure 83 The Five Elements as weapons in *Mastery*.

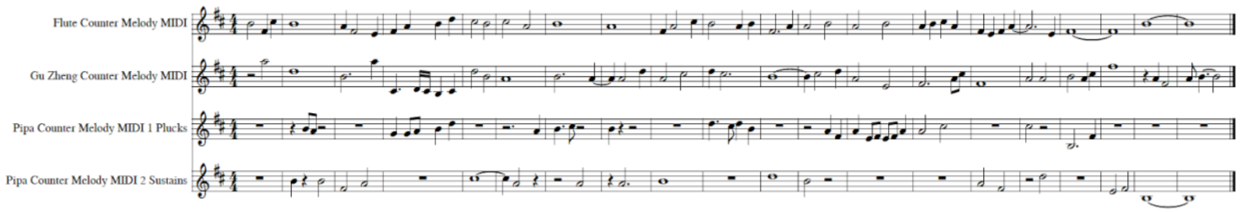
	Erhu **(high to higher register)**	***Pipa*** **(mid register)**	***Guzheng*** **(low to mid register)**	***Xiao*** **(high register)**	**Orchestral Strings**
Earth	Primary melody (static audio)	Counter melody (generative by MIDI)	N/A	Counter melody (static audio loop, 1/2)	Play on all weapon types
Water	Counter melody (static audio loop, 1/2)	Counter melody (static audio loop, 1/2) *Using *Ruan* Samples	Counter melody (generative by MIDI)	Primary melody (static audio)	
Wood	N/A	Primary melody (static audio)	Counter melody (static audio loop, 2/3)	Counter melody (generative by MIDI)	

Table 13 Dynamic instrumentation associated with different elements.

In Table 13, I describe the dynamic instrumentation associated with each weapon type. This table shows whether an instrumental layer uses MIDI track for generative content, as well as lists each instrumental layer's registral design. The numerators of the fractions represent the number of variations of the static audio loops, while the denominator represents the total number of sub-tracks in Wwise. When the numerator is smaller than the denominator, the remaining track is an empty track that contains no audio file. For instance, the *Xiao* counter melody played on the Earth weapon has a total of two sub-tracks; one is the non-generative audio track, while the other is an empty track. In Wwise, having an empty sub-track is a way to create a probability of whether the music track plays. Because of the empty sub-track, the static audio track has 50% of chance to be played in gameplay. The *Guzheng* playing on Wood has two static audio tracks and one empty track, thus, the two audio tracks each have a one third chance of playing. The "N/A" expresses that an instrument does not play on a certain weapon. The Orchestral Strings play on all weapon types. The design of instruments having different timbral registers (high, mid, low) keep spectral clarity between instruments.

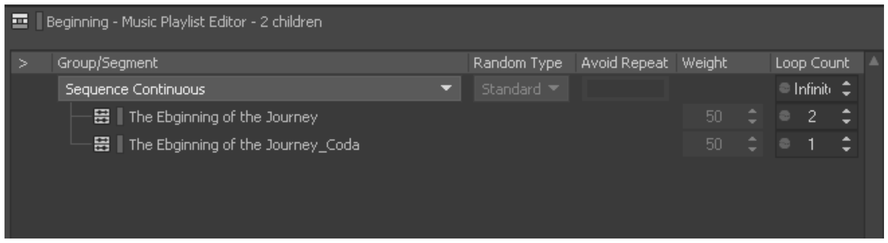
Dynamic instrumentation of the generative counter melodies each have unique rhythmic patterns, though they share harmonic progression thus use the same PCs.[49] A Wwise view of the MIDI tracks is shown in Figure 84, and a score showing the MIDI-note PCs, as well as rhythmic pattern, is shown in Figure 85. All three primary melody instruments play the same melody, while three non-generative counter melodies play varying counter melodies with different rhythmic and melodic patterns embodied in their MIDI tracks.


Flute-CounterMelody-Midi
Voice Volume
Flute Counter Melody MIDI Trimmed
Pipa-CounterMelody-Mi...
Pipa Counter Melody Group 2
Pipa-CounterMelody-Mi...
Pipa-CounterMelody-Midi_Group01
Pipa Counter Melody Group 1
GuZheng Counter Melody 2
GuZheng-CounterMelody-Midi
Figure 84 MIDI tracks of generative counter melodies in Wwise.


Flute Counter Melody MIDI
Gu Zheng Counter Melody MIDI
Pipa Counter Melody MIDI 1 Plucks
Pipa Counter Melody MIDI 2 Sustains
Figure 85 Score of MIDI tracks of the generative counter melodies.

[49] *Pipa* generative counter melody has two separate MIDI tracks. One track triggers the short pluck samples on *Pipa,* while the other triggers different long note articulation samples. I explain the dynamic articulations on *Pipa* in a later section.

The music sections of this movement also involve procedural formalistic arrangement. The main section of the movement is a 20-bar looping segment that alternates with a brief coda section after every two loops, as shown in Figure 86. The coda section is also 20 bars in length, containing no generative melodies but rather only static audio tracks with real time dynamic instrumentation. It consists of a different primary melody (Figure 87) layered on the orchestral strings. This alternative coda section does not loop and contains heavy reverb in the first six bars, conveying the feeling of hearing music played from far away. This section brings formalistic and motivic variation to the movement.


Beginning - Music Playlist Editor - 2 children

> Group/Segment	Random Type	Avoid Repeat	Weight	Loop Count
Sequence Continuous	Standard			Infinit
The Ebginning of the Journey			50	2
The Ebginning of the Journey_Coda			50	1

Figure 86 Sections of *The Beginning of the Journey* in Wwise.

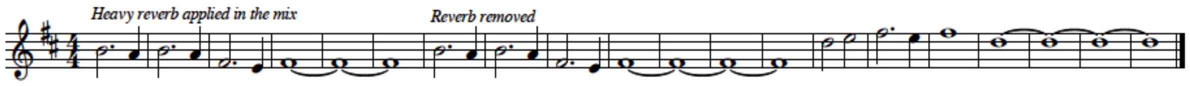


Figure 87 The primary melody in the coda section of *The Beginning of the Journey*.

4.2.6. Static Notes in Counter Melodies

Instrument	**MIDI notes triggering static notes**
Xiao	E4, F#4
Pipa	E4, F#4, B3 (the actual pitch played is B4)
Guzheng	B3 (actual pitch: B3), D4, E4, F#4, A5, C#4(actual pitch: C#5)

Table 14 MIDI notes triggering static notes in all three instrumental layers.

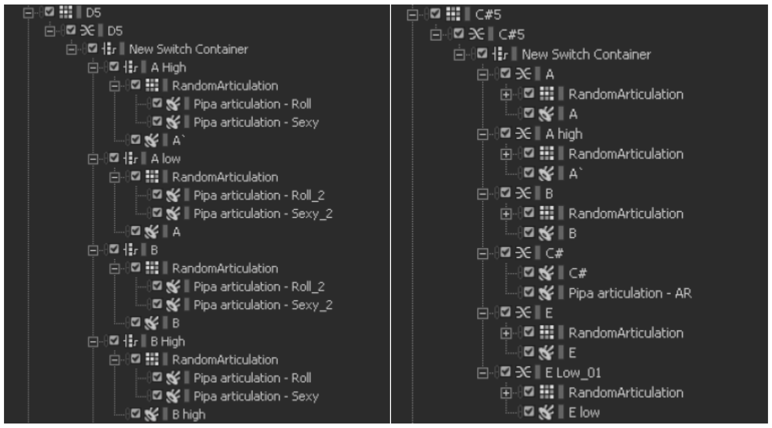
As discussed above, there are static notes in the *Xiao* generative counter melody: the notes E4 and F#4. The other two instruments in the generative counter melody group, *Pipa* and *Guzheng*, have additional music segments formed by static notes as listed in Table 14.

4.2.7. Dynamic Articulation on *Pipa*

Pipa has dynamic articulations in the generative counter melody. As shown in the score of Figure 85, two groups of articulation are notated in two MIDI tracks. The first *Pipa* line triggers the samples of plucking. Every MIDI note in this track lasts shorter than a half note which is perfect for the plucking articulation. The second *Pipa* line includes four different articulations to be triggered under various conditions. Table 15 lists all types of articulation, triggering conditions, and which MIDI track they belong to.

MIDI Track #	**Name/Abbreviation of articulations**	**Description of Articulations**	**Triggering Conditions**
1	Pluck (P)	Plucking on the strings	All notes on *Pipa* Counter Melody in MIDI track one
2	Roll (R) (挑轮)	A type of tremolo on *Pipa* which starts with a plucked note on right hand	On any half note or longer notes including B3 but except C#5, and aleatorically alternate with AP
	Appoggiatura Pluck (AP)	A slide on the string goes from one pitch to another on left hand while right hand plucks the string	On any half note or longer notes including B3 but except C#5, and aleatorically alternate with R
	Appoggiatura Roll (AR)	An appoggiatura on left hand and rolling on right hand combined	On C#5
	Sliding scale (S)	Left hand slides up and down on the string forming a one-octave scale while right hand playing a roll	On B3 (the sounding pitch is actually B4)

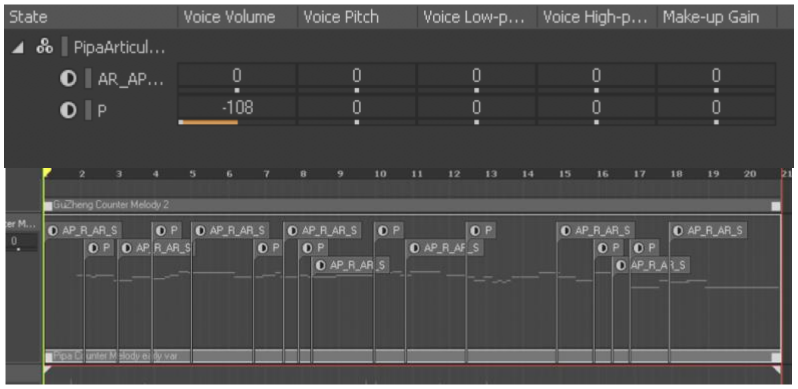
Table 15 Dynamic articulations on *Pipa*.



```
D5
  D5
    New Switch Container
      A High
        RandomArticulation
          Pipa articulation - Roll
          Pipa articulation - Sexy
        A`
      A low
        RandomArticulation
          Pipa articulation - Roll_2
          Pipa articulation - Sexy_2
        A
      B
        RandomArticulation
          Pipa articulation - Roll_2
          Pipa articulation - Sexy_2
        B
      B High
        RandomArticulation
          Pipa articulation - Roll
          Pipa articulation - Sexy
        B high
```

```
C#5
  C#5
    New Switch Container
      A
        RandomArticulation
        A
      A high
        RandomArticulation
        A`
      B
        RandomArticulation
        B
      C#
        C#
        Pipa articulation - AR
      E
        RandomArticulation
        E
      E Low_01
        RandomArticulation
        E low
```

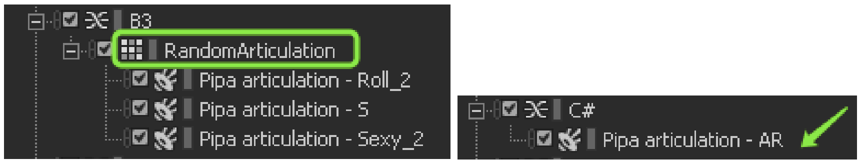
Figure 88 Triggering *Pipa* articulations using switch container.



State	Voice Volume	Voice Pitch	Voice Low-p...	Voice High-p...	Make-up Gain
PipaArticul...					
AR_AP...	0	0	0	0	0
P	-108	0	0	0	0

Figure 89 Triggering *Pipa* articulations using states on one MIDI track.

I started using Wwise's switch containers to trigger the two groups of articulations using only one MIDI track (Figure 89), but there is a significant latency problem with the switches triggering samples. I later placed all samples in blend containers instead, so that they would be triggered by Wwise's states marked on the MIDI track (Figure 88). The transition between articulation was still not perfect, thus different types of articulation occasionally overlapped with each other.



```
B3
  RandomArticulation
    Pipa articulation - Roll_2
    Pipa articulation - S
    Pipa articulation - Sexy_2
```

```
C#
  Pipa articulation - AR
```

Figure 90 Randomly choosing articulation or with fixed MIDI note association.[50]

[50] The "Sexy" sample in the figure actually represents AP.

Eventually the solution was to use two separate MIDI tracks, as discussed previously. This approach has better performance, because the two groups of articulation are each linked with a specific MIDI track designed for triggering them. Consequently, the computer no longer needs to "decide" which group of articulations a MIDI note should play all the time. This solution is also better because the four types of articulation included in group two are either played randomly or on a predetermined pitch (Figure 90), allowing more time for Wwise to "prepare" the appropriate samples in its backend process, instead of reading the switches and states marked on the MIDI track in the front user-view process. The resulting performance of this method has no glitches in switching articulation and transitions smoothly.

4.2.8. Programming the Generative Melodies

After deciding the PCs and ranking numbers, the next step was to map the musical parameters to game parameters. Figure 91 shows the original manuscript of the design of the ranking numbers' movement in real time gameplay. The vertical axis shows the value of the ranking numbers, ranging from 1 to 10.[51] The generated pitches move by different melodic intervals, which I named "distance of voice-leading" (DVL)[52] in the manuscript and which are represented by the *v* values. The horizontal axis shows the beats in which pitch generation occurs in the main looping section of this movement. Each graphical line formed by values on the two axes represents the contour of a possible generated melody, notated with the ranking numbers. For instance, the "*v*=1" line shows a potential melody where the ranking numbers can only move one step up or down during real time generation. The top line shows that the melodic movement moves four steps apart every time, thus having a value of four in DVL(*v*=4).

[51] The vertical axis starts at "1" instead of "0", because a voice-leading distance of "0" appears only occasionally. I will explain this in a later section.
[52] DVL=0 is actually included in the chart of Appendix A. DVL, voice-leading intervals discussed in Figure 83, and Ranking Number Differences in Appendix A all refer to the same thing.

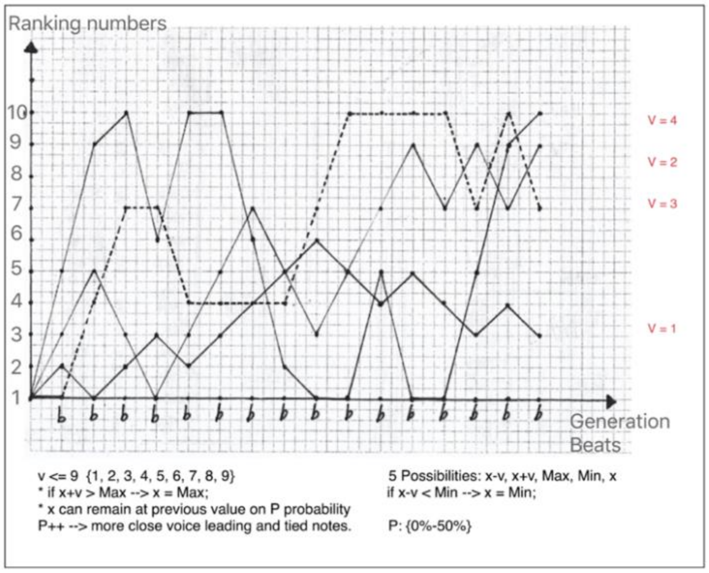


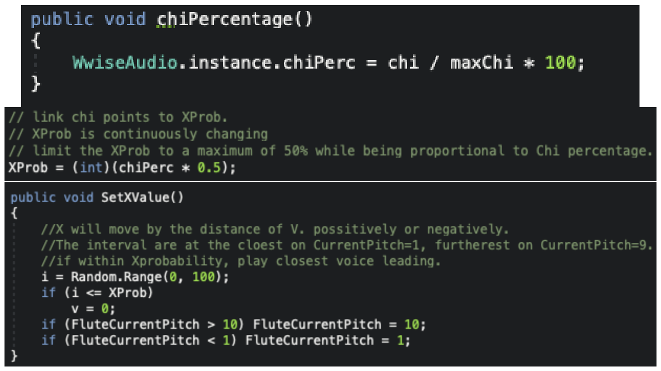
Figure 91 Design of the movement of ranking numbers in – Manuscript.

After hand-drawing this collection of melodic contours, I developed several rules for the algorithm to perform the way I wanted. The rules include setting the range of DVL (1-9), what to do if there aren't sufficient values to reach, and probabilities of the occurrence of *v*=0.

As already explained, the maximum value of *v* is nine, thus making movement from ranking number one to ranking number ten possible. When there isn't sufficient room to move upward or downward, the resulting pitch will play the maximum or minimum values. This is why the second to last beat on the *v*=4 line only moves one step up instead of four. Similarly, when there aren't enough ranking numbers to go lower, the resulting pitch will stay at one.

I coded this process by comparing the most recent ranking number to the maximum (*Max*) or minimum (*Min*) values. If the result exceeds the maximum value, it will be replaced by the maximum value and vice versa. The resulting value of the next pitch is represented by the *x* in Figure 91. In conclusion, *x* may have five different values: *x*-*v*, *x*+*v*, *Max*, *Min*, and *x* (when *v*=0).

DVL=0 or *v*=0 may occur with 0% to 50% probability, represented by the *P* value in Figure 91. This voice-leading movement may produce an effect of tied note, repeated note, and other smaller melodic intervals. The *P* value could be an interesting musical effect to link with a specific game parameter that carries the same aesthetic character. As shown in Figure 92, the *P* value (XProb in the code) is linked with the percentage of *Chi* (or mana) value (chiPerc in the code), by comparing *Chi* with its possible maximum value (maxChi). The value of XProb is then scaled to 50% so that the maximum probability never exceeds 50% as intended. An integer number (i in the code), ranging in value from one to 100, constantly generates a random number within its set range. If the value of i is smaller than XProb, the DVL value (v in the code) will be set to zero, otherwise it equals half of chiPerc.



```
public void chiPercentage()
{
    WwiseAudio.instance.chiPerc = chi / maxChi * 100;
}
```

```
// link chi points to XProb.
// XProb is continuously changing
// limit the XProb to a maximum of 50% while being proportional to Chi percentage.
XProb = (int)(chiPerc * 0.5);
```

```
public void SetXValue()
{
    //X will move by the distance of V. possitively or negatively.
    //The interval are at the cloest on CurrentPitch=1, furtherest on CurrentPitch=9.
    //if within Xprobability, play closest voice leading.
    i = Random.Range(0, 100);
    if (i <= XProb)
        v = 0;
    if (FluteCurrentPitch > 10) FluteCurrentPitch = 10;
    if (FluteCurrentPitch < 1) FluteCurrentPitch = 1;
}
```

Figure 92 Programming of dynamic value of DVL linked with *Chi*.

4.2.9. Mapping to Player Input and Game States

In addition to the *P* value being linked to the percentage of *Chi*, other musical parameters are also linked with specific player input and game states. Player inputs and game states can interactively influence the generative melody in two specific ways:

a. Directly setting a specific ranking number in order to trigger an immediate pitch (IP)

b. Setting a DVL value (either specific or random) to directly control the voice-leading and indirectly how ranking numbers change over time

Every type of player's direct input sent via the game controller can influence the music in either of the two ways. In Table 16 and Table 17, the first columns show the types of controller input data or game states used, while the second columns show how they influence the generative melody, and the third columns explain the musical meaning of the interaction between the game and music systems.

Types of player input	**Influence on the melody**	**Musical meaning/expression**
Go Left (trigger on key-off)	DVL=0	Moving left means the player is moving to known areas in the platform that is less dangerous thus less nervous emotionally
Go Right (trigger on key-off)	DVL=1	Moving to the right direction represents proceeding in the gameplay and exploring new and unknown areas in the platformer which should convey a risky, mysterious, and dangerous feeling
Jump	IP = 8-10	Generating immediate high pitches in the melody creates a musical metaphor imitating the dramatic movement of the player
Dash	IP = 8-10, DVL 7-8	Similar to Jump, generating immediate high pitches represents the dramatic movement of the player but the melodic peak is also followed by wide leaps in voice-leading in order to express the movement of dashing which seems much more rapid than walking or jumping
Land (after jump)	IP = 3-4	Pitches falling down to medium register after jump imitates a landing motion. (there is no landing after a dash)
Attack	DVL 2-3	Attacking is a more intensive movement than walking, thus the melodic movement should also be more intensive than walking (Go Left or Go Right)
Crouch	IP = 1-2	Suddenly playing the music at a low register of pitches imitating the crouch movement musically
F skills (occur more often than C skills)	DVL 4	DVL=4 contains a greater amount of variety of intervals between fourths to seventh, and is less evenly spread than DVL=5
C skills	DVL 5	A relatively more evenly spread variety of intervals between fourths to sevenths

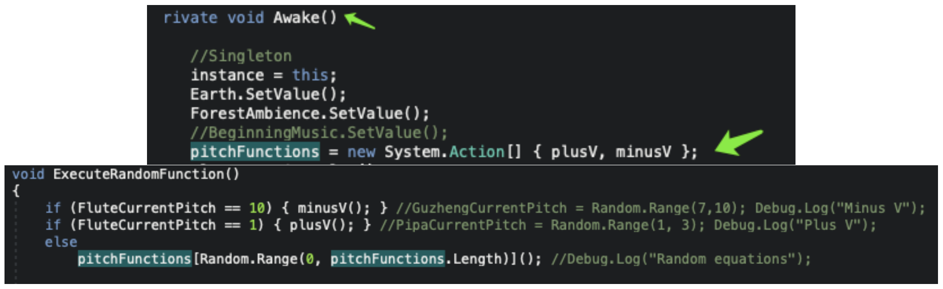
Table 16 IP and DVL mapped to player's input data.

The upward and downward motions (minusV() and plusV() in the code) of voice-leading are programmed as part of an array (pitchFunctions[] in the code) in the Awake() function, as shown in the top part of Figure 93. The bottom part of Figure 93 shows that voice-leading will move in the opposite direction when the maximum or minimum value is reached; when the maximum and minimum aren't reached, the pitchFunctions[] decides a random direction so that voice-leading moves up or down randomly.

Types of game states	**Influence on the melody**	**Musical meaning/expression**
On Mud	DVL 0-1	Player walks slower on sticky ground, which is accompanied by less active melodic movement
On Ice	DVL 9	Player walks on icy and slippery ground, which is accompanied by larger melodic intervals in the voice-leading
Player Hurt	DVL 6	Momentarily playing medium-wide melodic intervals
Player Idle	DVL 0-1	When the player has no activity for a while, the game character will switch to the idle mode, thus the voice-leading also moves with low intensity
Player die	IP = 1 DVL 0	The melody pitch immediately falls down to the lowest pitches of the currently playing PC as well as with lowest intensity of voice-leading movement
Win	IP = 1 DVL 0	The melody plays at the lowest register and the closest to the tonic, creating a closer melodic transition to the "Win" tail music segment showing a leaning towards a harmonic resolution.

Table 17 IP and DVL mapped to game states.

The DVL values gradually increase from being triggered by player's moving left, moving right, attacking, using F and C skills, and dashing. In *Mastery*, the F key casts the elemental abilities that come with each weapon type. The abilities, such as building a wall while using the Earth element or having an eddy shield surround the player for extra protection when using the Water element, are essentially supplemental to the player's basic attack skills. The C key casts special attacks that costs *Chi* and can cause more damage to the enemies than the regular attacks.



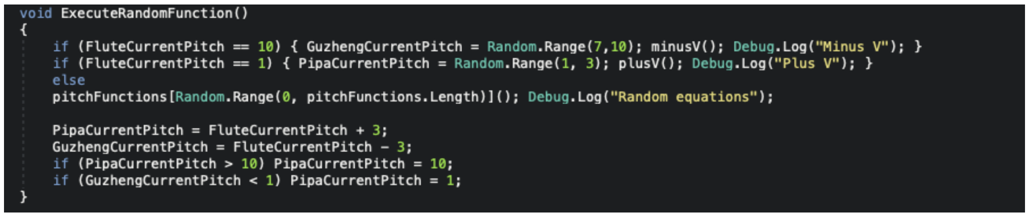
```
rivate void Awake()

    //Singleton
    instance = this;
    Earth.SetValue();
    ForestAmbience.SetValue();
    //BeginningMusic.SetValue();
    pitchFunctions = new System.Action[] { plusV, minusV };
```

```
void ExecuteRandomFunction()
{
    if (FluteCurrentPitch == 10) { minusV(); } //GuzhengCurrentPitch = Random.Range(7,10); Debug.Log("Minus V");
    if (FluteCurrentPitch == 1) { plusV(); } //PipaCurrentPitch = Random.Range(1, 3); Debug.Log("Plus V");
    else
        pitchFunctions[Random.Range(0, pitchFunctions.Length)](); //Debug.Log("Random equations");
```

Figure 93 Voice-leading is programmed to move up or down randomly.

Though not implemented in this project, the three instruments playing the generative counter melody can each have their own generation rules for ranking numbers and voice-leading.

As shown in Figure 94, I have experimented with having the *Pipa* and *Guzheng*'s pitch ranking numbers generated by modifying the *Xiao*.



```
void ExecuteRandomFunction()
{
    if (FluteCurrentPitch == 10) { GuzhengCurrentPitch = Random.Range(7,10); minusV(); Debug.Log("Minus V"); }
    if (FluteCurrentPitch == 1) { PipaCurrentPitch = Random.Range(1, 3); plusV(); Debug.Log("Plus V"); }
    else
    pitchFunctions[Random.Range(0, pitchFunctions.Length)](); Debug.Log("Random equations");

    PipaCurrentPitch = FluteCurrentPitch + 3;
    GuzhengCurrentPitch = FluteCurrentPitch - 3;
    if (PipaCurrentPitch > 10) PipaCurrentPitch = 10;
    if (GuzhengCurrentPitch < 1) PipaCurrentPitch = 1;
}
```

Figure 94 Pipa and Guzheng's pitch generation work in relation to Xiao pitches.

4.2.10. Thoughts and summaries: the two timelines

It is worth mentioning that the game engine and the music system each has their own distinctive timelines playing out during the gameplay. Lining up the two timelines elegantly becomes one of the biggest challenges in composing dynamic game music. On the one hand, the player continuously sends data to influence the gameplay, while the game engine continuously checks player's input and updates game states on every frame. All these data are then sent to the music system on every frame as well. On the other hand, the generation events in the music system need to be rooted in its own predetermined timing that is musically meaningful (e.g., meter, beat, note). [53]

The music system cannot predict any generation requests *before* receiving the game triggers. As a result, the music system may miss responding to a game trigger if it receives the game data before or after a node. This trigger will have to be either delayed until the next available node or replaced by the most recent game data received before a node, because there is no way for game data to influence the music generation between two generation nodes. The game's timeline is constantly "trying" to be synced with the music's timeline, which can be viewed as a type of performance coordination similar to how human musicians try to play music perfectly together

[53] One of the only few exceptions is dynamic crossfading between multiple layers of music with fixed musical content, such as the dynamic instrumentation discussed in 4.2.5. Dynamic Instrumentation and Sequencing.

through rehearsals. This game-music interaction differs from real musicians playing together in that it does not need rehearsal, because only the perfectly synced materials are being executed and heard. This feature, however, can also cause problems, because if the transition between nodes sounds odd, there is no way to fix it. Game composers have to work hard to make the two timings match as closely as possible. This is still an inevitable challenge in games that aren't rhythm games. [54]

This challenge also suggests that the smaller the space between the music's generation nodes, the tighter the synchronization between timelines of the game and music. In this movement, the melody notes of the generative counter melodies are being generated one at a time in real time, so every note is a generation node. Having generation nodes on every single note is quite dense. It allows the game data to sync with music generation more tightly, increases user agency via the music being more responsive, and thus creates a better experience for players.

Expanding the Pitch Ranking System

Although not implemented in this project, there are extensive possible expansions of this ranking system. For instance, 12-tone techniques can be applied to the algorithms. The pitch ranking system in this project has similarities with 12-tone techniques in that they both represent pitches with numbers, though 12-tone techniques exist more traditionally in non-tonal music. Some of the most characteristic features and contributions of 12-tone music include various manipulations of the tone rows (inversion, retrograde, and retrograde inversion), as well as the total-serialistic expansion of 12-tone techniques, which quantifies rhythm and dynamic. The various manipulations of a tone row can be applied to a pitch ranking system to create interesting melodic and motivic variations within the scope of tonal music. Techniques of total-serialism can

[54] Rhythm games are rhythm-based games which can be similar to music games but not exactly the same. While both genres incorporate musical timings to drive the gameplay, music in rhythm games does not necessarily function as the actual gameplay as it does in music games. For instance, a first-person shooter game can be a rhythm game if the game rhythm is based on the musical timing of the soundtracks.

generate dynamic rhythms, such as triggering samples of notes with various duration or applying fade-in and fade-out to long notes in order to assign them dynamic rhythmic durations. Existing limitations of the two game projects, as well as expansions of the ADM techniques introduced in this paper, will be discussed more in detail in the final chapter.

4.3. Dynamic Phrasing, Meter, and Diminution

4.3.1. First Task

The instrumentation of the second movement *First Task* involves: the traditional Chinese wind instrument *Dizi* (笛子), lute *Sanxian* (三弦), seven-string plucked instrument *Guqin* (古琴), and a set of percussion instruments including *Bangzi* (梆子), *Luo* (锣), and a type of drum called *Dagu* (大鼓). The set of percussion instruments functions as the rhythmic support to the entire movement. *Guqin* has three patterns composed in the three module lengths, set to play on loop as the movement's accompaniment. *Dizi* and *Sanxian* play in alternation like a dialogue as the melody lines.

Three formalistic sections are embodied in this movement and are played in order and contain looping structure. This soundtrack begins with an introduction section, followed by the *Dizi* section, followed by the *Sanxian* section, which functions like a response to the *Dizi*. After the initial playing of all the three sections, the *Dizi-Sanxian* sectional conversation loops to accompany the indeterminate time players spend in the associated game area in *Foothill*. Although all three sections involve dynamic musical content, only the *Dizi* part in the *Dizi* section involves direct interaction with the player input data.

This movement involves dynamic phrasing variations, which manipulate three phrasing types, three meters, four episodes[55], and five levels of diminution.[56] The three phrasing types are sentence, period, and parallel period. The three meters include 4/4, 5/4, and 6/4, for which music modules are bounced in length of four, five, and six beats. In this movement, I call the alternation between one *Dizi* section and one *Sanxian* section an *episode* and there are a total of four episodes. *Dizi* has different content for the four episodes, while the *Sanxian* section alternates between two

[55] In Baroque contrapuntal music, especially the form *Rondo*, when a repeating principal theme (sometimes called the "refrain") alternates with a contrasting theme, this is generally called an "episode (Wikipedia, 2021d)."
[56] A diminution in traditional Western classical music is a technique used to embellish or decorate long notes with shorter and fragmented notes for melodic variations.

patterns. The levels of diminution represent intensity of ornamentation and fragmentation embodied in the instrumental layer of *Dizi* in the *Dizi* section.

This movement centers in the F-Gong Ya Yue Yu heptatonic scale (仲吕宫雅乐羽调式)[57], or D Dorian scale in Western music theory, as shown in Figure 95. The two added altered tones (colored in green) in the heptatonic scale have significant roles in the *Dizi* part, which will be discussed later in this chapter. The entire movement stays on one chord--in other words, has one associated PC. This lack of harmonic movement helps to clearly demonstrate the dynamic phrasing, as it is not mixed with any PC change.

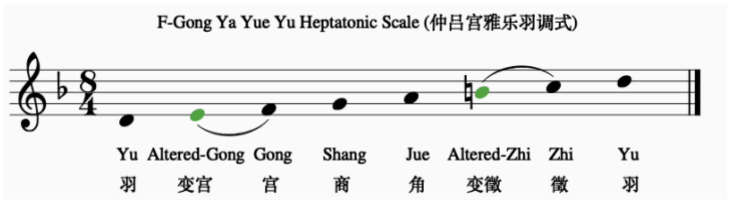


Figure 95 The key and scale used in *First Task*.

Call and Response

The musical conversation between the *Dizi* and *Sanxian* sections creates a call and response effect. The *Dizi* section carries user-driven generative phrasings with random lengths between 16 and 32 measures. The *Sanxian* section, in contrast, always has four measures in length played unchangeably in the meter of 4/4. The *Sanxian* section adopts the phrasing principal of Qi-Cheng-Zhuan-He (起承转合) that is typical in traditional Chinese music.

[57] In traditional Chinese music, F in the twelve chromatic tones is called Zhong Lv (仲吕). A Ya Yue (雅乐) scale is a type of heptatonic scale with altered-Gong (变宫) and altered-Zhi (变徵) added to the original pentatonic scale. The altered-Zhi (变徵) is a minor second below the Zhi (徵) note (Wikipedia, 2020b, 2021a).

4.3.2. Nodes for Music Generation in the *Dizi* Section

Different from the generative counter melodies in the first movement in which every note represents a generation node, the *Dizi* section in this movement has generation nodes set to four different intervals. First, the dynamic phrase types are generated by compiling modules of short music motifs. Each module is one quarter of the length of an episode. I labelled them *quarter phrases* distinguished with letters X, Y, Z, and C. The music system randomly chooses a first module, and the player's input decides the second module and thus the phrase type. Second, the computer randomly chooses a meter for the *Dizi* section before the first beat of every episode. Third, though episodes play in order instead of interactively ordered by player's input, samples of the four episodes are procedurally triggered by Wwise. The four episodes function similarly with the PCs in the first movement, as they both function to select the correct collection of audio samples for music generation. In the first movement, the generated pitches can come only from the currently playing PC's associated sample group. Similarly in the second movement, Wwise uses the sample group containing the currently playing episode's associated quarter phrases for phrasing generation. Finally, the five levels of diminution can be interactively changed on any frame of a second during the gameplay[58] depending on the computer's running power and capacity. There is a 0.3 second crossfade applied to the audio samples during the transition between the previous assigned level and the new assigned level for smoothness. The different intervals of generation are also summarized in Table 18.

[58] It is worth mentioning here that the interactive diminution level does not execute on a musical timing but on frames, depending on the computer's frame rate or frame per second (FPS) (Wikipedia, 2021b). Thus, there are many nuances of melodic and rhythmic activities occurring at transitions of diminution levels between the frames. These nuances bring indefinite musical variations to the piece that are not possible to be notated out.

Generation Nodes	**When Generate Occurs**	**Who Generates**
Phrase types	At first and second quarter phrases on every episode	Computer generates a random first quarter phrase, player decide the second phrase and phrasing type
Dizi Meters	Before first beat of every episode	Randomly chosen by computer
Episodes	Four episodes play in order, samples selected before the start of every episode	Computer
Diminution Levels	On every frame with 0.3s crossfade	Player

Table 18 Intervals of generation nodes in *First Task*.

4.3.3. Introduction Section

The introduction section has a random length generated by Wwise. It has two patterns of a one-bar phrase played on *Guqin*, labeled as Pattern *a* and Pattern *b* (Figure 96). Pattern *a* can be repeated one to three times, followed by Pattern *b* playing one time. On the last beat of Pattern *b*, *Luo* joins the music ensemble and triggers the *Dizi* section. This musical procedure programmed in Wwise is shown in Figure 97.

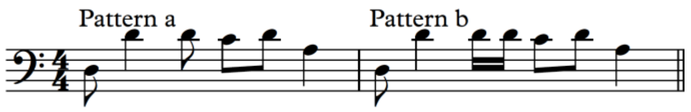


Figure 96 Introduction *Guqin* patterns.

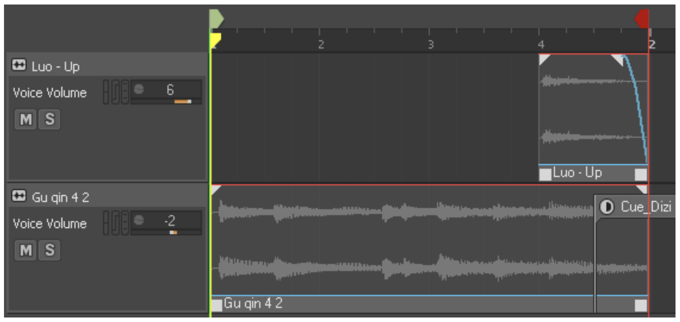


Figure 97 *Luo* joins on the last beat of Pattern *b* and triggers the *Dizi* section.

The implementation of this procedure involves several steps. Pattern *a* is duplicated as two sub-tracks where one holds a music event cue called "cue-16" to trigger Pattern *b,* while the other one does not. We will label them as Pattern a_1 and Pattern a_2 (a_2 is the one that triggers Pattern b), as shown in Figure 98 and Figure 99. Next, both Pattern *a*'s are placed in a playlist container (Figure 100), which functions to select one of three music sequences included in the playlist to randomly play Pattern *a* (also named "8_Cue_16" in Wwise) one, two, or three times. The three procedures are: playing Pattern a_2 for one time which then triggers Pattern *b*; playing Pattern a_1 for one time followed by Pattern a_2 which then triggers Pattern *b*; and playing Pattern a_1 for two times followed by Pattern a_2 which then triggers Pattern *b*. The three sequences also have their own probability of chance being selected as 35%, 45%, and 20%.

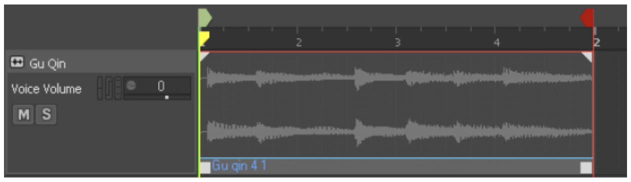


Figure 98 Pattern a_1.

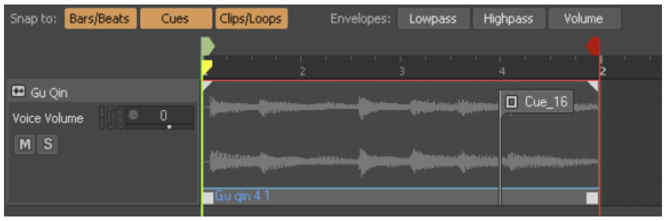
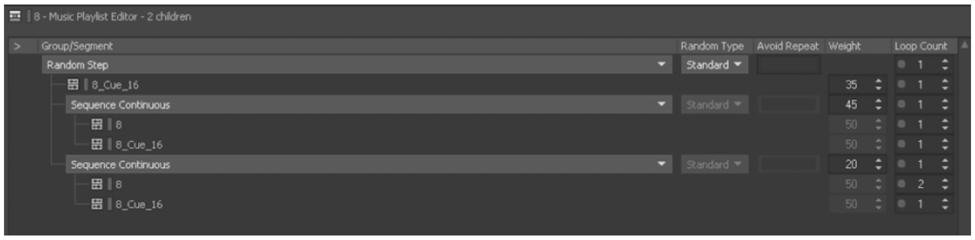


Figure 99 Pattern a_2 has a music event "Cue_16" which triggers Pattern *b*.


8 - Music Playlist Editor - 2 children

Group/Segment	Random Type	Avoid Repeat	Weight	Loop Count
Random Step	Standard			1
8_Cue_16			35	1
Sequence Continuous	Standard		45	1
8			50	1
8_Cue_16			50	1
Sequence Continuous	Standard		20	1
8			50	2
8_Cue_16			50	1

Figure 100 Playlist container randomly plays Pattern *a* for one to three times.

4.3.4. The *Dizi* Section

Percussion Instruments and Guqin

In the *Dizi* section, *Guqin* continues to randomly play Pattern *a* or *b*. While preserving the initial patterns used in the introduction section for the 4/4 meter, the patterns also adjust to the two other meters (5/4 and 6/4), as shown in Figure 101. *Bangzi* plays continuously on the upbeats throughout the entire track when the music plays in 4/4. When the music plays in 5/4, each 5-beat measure can be considered as either a 2+3 or 3+2 pattern where *Bangzi* plays on the upbeats in 1+2 or 2+1 pattern (the subtracted beats are the downbeats). *Bangzi* always alternates between 1+2+2+1 and 2+1+1+2 patterns in 5/4. When the music plays in 6/4, *Bangzi* starts with the 2+1+1+2 and alternates with 1+2+2+1 in similar fashion as in 5/4 but has extra rests added after the "1" part, as demonstrated in Figure 102. *Dagu* only plays on the down beats on every other episode regardless of the chosen meter.

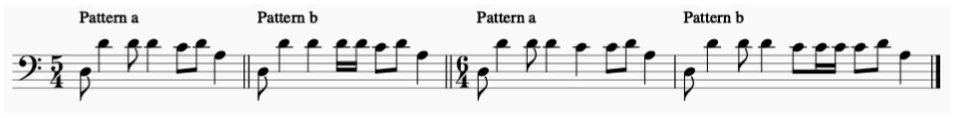


Figure 101 *Guqin*'s Pattern *a* and *b* used in the *Dizi* section.

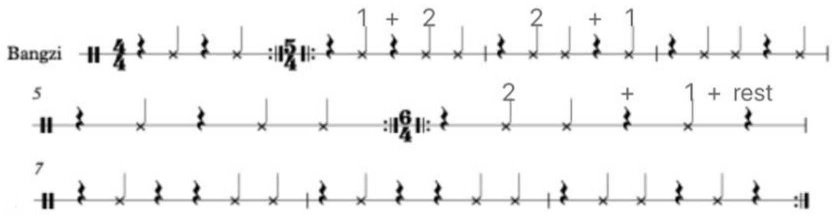


Figure 102 *Bangzi*'s patterns when playing in different meters.

Episodes

The four episodes in the *Dizi* section are triggered by the *Bangzi* tracks using a State Group called "T2_Episode," as shown in Figure 103. The "T2_Episode" state group is also used to select samples of quarter phrases associated with each episode, as shown in Figure 104. For instance, the x module in episode one will always play as the x module of episode one and is never available for phrase generating in episode two, three, and four.

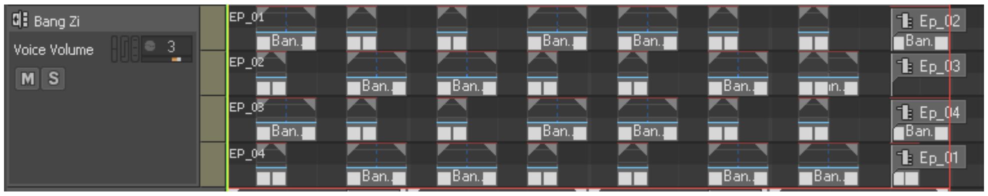


Figure 103 *Bangzi* tracks are used for triggering the four episodes in order.

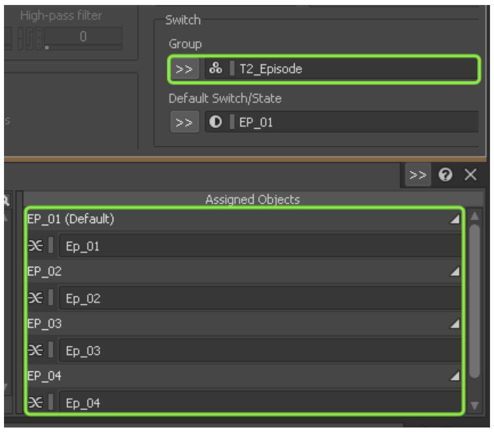


Figure 104 The "T2_Episode" state group also triggers the associated quarter phrase samples.

Each episode carries a specific melodic shape formed by its quarter phrases. The complete four-episode contour, formed by a period phrase type, may show a melodic trajectory listed below:

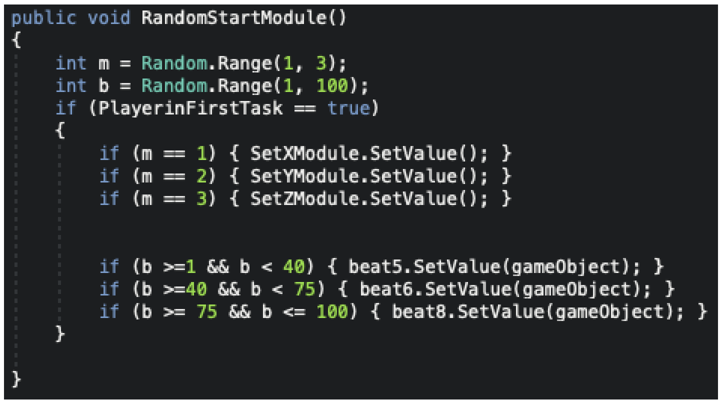
- Ep.1 starts on D4 and ends on D3
- Ep.2 starts on D3 and moves up and down to finally get back to D3
- Ep.3 starts on D3 and goes up to G3
- Ep.4 starts on F3 and goes down to D3

Besides the above-mentioned melodic contours, any episode may end on A3 representing a half cadence in a parallel period phrase type. Since phrases are dynamically generated in real time gameplay, the melodic contour may begin with the first note of either the X, Y, or Z quarter phrase. The ending notes of an episode, however, will always remain identical to the period example discussed above. While there are also the five levels of diminution, higher levels of diminution may start with a neighboring note instead of the initial note in diminution level one.

Meters

The dynamic phrasing on *Dizi,* formed by four quarter phrases, are bounced in audio files with lengths of five, six, and eight beats. Implementing these samples of the three module lengths help create an effect of having dynamic meters (5/4, 6/4, 4/4 accordingly) in the music. When music is playing in 5/4 or 6/4, every module represents one measure in the actual music, therefore five or six beats. If the music is playing in 4/4, an eight-beat module equals two measures in the meter. The Wwise engine prepares and executes the upcoming module after every five, six, or eight beats, respectively. The meter selection occurs before the first module of every episode.

The Wwise engine chooses a module length to play based on their probabilities programmed in the game engine. As shown in Figure 105, 5/4 has 40% probability to be chosen, 6/4 has 35%, and 4/4 meter has only a 25% probability.



```
public void RandomStartModule()
{
    int m = Random.Range(1, 3);
    int b = Random.Range(1, 100);
    if (PlayerinFirstTask == true)
    {
        if (m == 1) { SetXModule.SetValue(); }
        if (m == 2) { SetYModule.SetValue(); }
        if (m == 3) { SetZModule.SetValue(); }


        if (b >=1 && b < 40) { beat5.SetValue(gameObject); }
        if (b >=40 && b < 75) { beat6.SetValue(gameObject); }
        if (b >= 75 && b <= 100) { beat8.SetValue(gameObject); }
    }

}
```

Figure 105 Probability of the three meters.

Phrase Types: quarter phrases

The three phrase types take inspiration from the common practice of phrase construction in Western classical music. The phrasing primarily concerns the combining of the X, Y, Z, C quarter phrases in various ways to form a complete episode.

In Western classical music, especially classical era sonatas, there are very specific compositional techniques for phrasing.[59] The two commonly used types of musical phrasing can be seen frequently in a classical era sonata: the well-known sentence and period, in which smaller music segments can be placed in different orders, repetitions, fragmentations, and combinations. Other phrasing structures include all phrase types that are not sentence or period, such as sequential passages, a fugal structure, or motifs being heard only once, among others.

Presentation and continuation are components of a sentence, while antecedent and consequence are components of a period. Each component is further divided into two parts that are usually equal in length, such as two basic ideas in a presentation. Each part of a component is roughly equivalent to one quarter of the complete sentence or period. Although continuation does not always contain two equal parts, it needs to contain a strong cadence - often perfect authentic cadences - to occur at the last quarter of a complete sentence. While a regular period does not have to contain repetitive motifs like the basic ideas, a special type of period called parallel period often has its antecedent and consequence start with similar motifs but end in softer and stronger cadences, often a half cadence and a perfect authentic cadence.

These characteristics of traditional phrasing inspired me to create a collection of quarter phrases as the foundational building blocks to form generative phrases of either a sentence, regular period, or parallel period in this movement. The building blocks recycle for continuous generation, thus little material can produce a massive amount of dynamic musical content.

Presentation and continuation in a sentence usually have equivalent length. The presentation contains one basic idea that is repeated in identical or similar form one more time. While the presentation carries two similar parts, the continuation has no repeated parts and no

[59] Within the scope of this dissertation work, I only introduced the adoption of several traditional compositional techniques in Western classical music to ADM. I hope to expand this literature of compositional techniques in future studies. I will discuss this more in the last chapter.

similar parts with the presentation. If we take the length of the basic idea as the measurement of a building block, a sentence needs three different blocks, as shown in Figure 106.

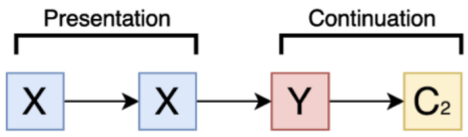


Figure 106 A sentence compiled by three modules.

The X and Y modules are available for the generation system at any time, except at the time of the last quarter phrase, while the C module can be selected only at the time of the last quarter phrase since it functions as the cadential motif.[60]

Antecedent and consequence in a music period usually have equivalent length as well. Antecedent and consequence can begin with similar or different motifs depending on whether it is a regular period or parallel period. Taking the same length of a quarter phrase for a sentence, a regular period needs four different modules, while a parallel period needs two different modules, as shown in Figure 107.

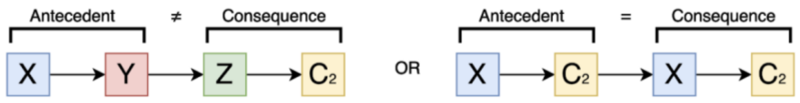


Figure 107 Periods compiled by four or two modules.

Different from a sentence, both antecedent and consequence in periods end with a cadence. Thus, I created two types of C modules: C_1 and C_2. The only difference between them is that C_2 ends on the tonic note representing an authentic cadence (or G3 on the third episode representing a plagal cadence), while C_1 ends on the dominant note representing a half cadence. C_1 modules only

[60] The C modules are composed with ending on a longer note that belongs to the last harmony in either the half cadence or authentic cadence.

appear in periods. Thus, the Z modules join the module library since a regular period needs four unique quarter phrases.

All possible phrasing patterns of sentence and period that can be generated are listed in Figure 108. Among the total of 15 phrasing patterns, there are six sentences, six regular periods, and three parallel periods.

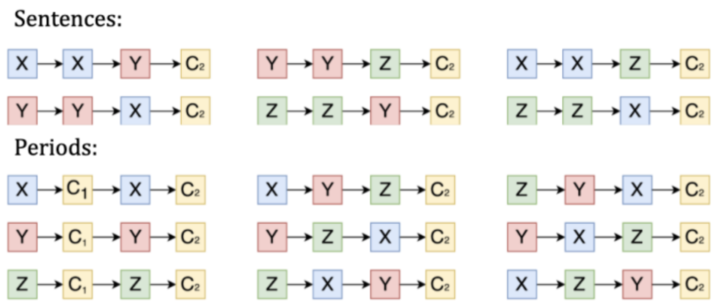


Figure 108 All possible patterns of sentences and periods by four modules.

As discussed above, quarter phrases are bounced in three different lengths: five, six, or eight beats. Figure 109 shows a demonstrative score showing all 15 phrasing patterns using the eight-beat samples (representing a 4/4 meter).

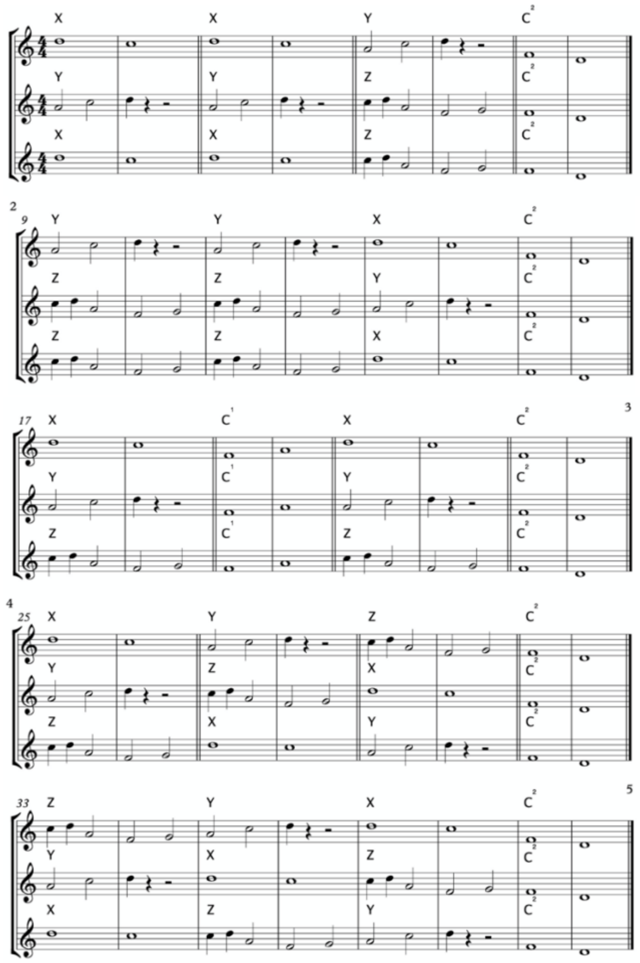


Figure 109 All 15 phrasing patterns can be generated by the XYZC modules in 4/4.

Phrasing Controlled by Game Data

Whether the *Dizi* line plays a sentence, period, or parallel period is controlled by three different game states: whether the player is attacking an enemy, attacking a breakable object[61], or not attacking anything in the *First Task* area. The phrasing type is determined when the second quarter is chosen, since sentence and period start to differ from that point (e.g., a sentence if repeating the first quarter phrase, or a period if playing a different quarter phrase). When the third

[61] In *Foothill*, breakable objects include rocks, fire rocks, and the wall built by the F skill with the Earth element while enemies include boars, flowers, wolves, pangolins, and the boss.

quarter phrase is chosen, it determines whether it is a regular period or a parallel period. The interactive generation is designed in the following steps:

- If the player is attacking an enemy, generate a sentence, meaning the first quarter phrase should be repeated.
- If the player is not attacking anything, generate a regular period, meaning the second quarter phrase should be a half cadence and the third quarter phrase should be different from the first quarter phrase.
- If the player is attacking a breakable object, generate a parallel period, meaning the second quarter phrase should be a half cadence and the third quarter phrase should be the same with the first quarter phrase.

The process of compiling a complete phrasing pattern in one episode with the four corresponding quarter phrases is shown in Figure 110.

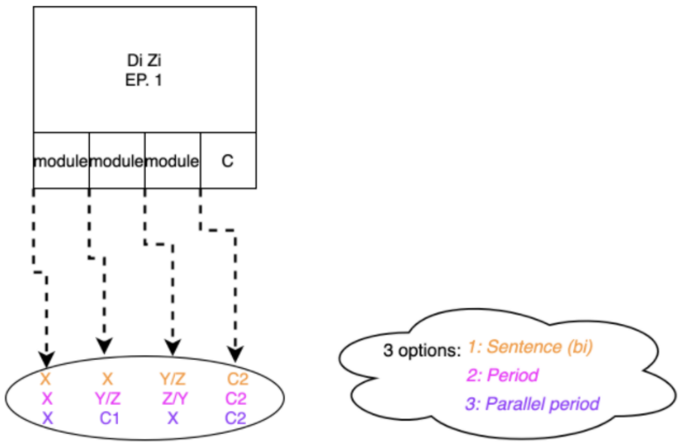


Figure 110 The state machine of the generative phrasing in the *Dizi* line in the *Dizi* section.

Diminution Levels

From the Baroque period in Western classical music history, a popular technique of music ornamentation is the diminution, which divides a long note into a series of shorter notes that function as an embellishment. I applied this technique to create five variations of the modules as five levels of diminution intensity. From diminution level one to five, the melody gradually gets more fragmented and ornamented and gradually shifts from using the initial pentatonic scale to the full heptatonic version of the scale.

The five diminution levels are interactively controlled by the player's movement in the game world, controlled from the joystick or keyboard. The initially composed quarter phrases are considered level one. From there, I composed another four levels of variation based on the initial level. As soon as a quarter phrase (X, Y, Z, or C) is selected in Wwise, samples of all five levels of diminution are also selected to play. Depending on the type of the player movement, Wwise will immediately crossfade and introduce the sample of the associated diminution level. The switching between diminution levels does not interfere with the selection of quarter phrases; the two types of dynamic content can happen simultaneously and independently.

Figure 111 shows the entire state machine where the music generation nodes represent the quarter phrases. The arrows show the sequencing of the quarter phrases forming the 15 patterns of sentences and periods. The top left X node represents that the five diminution levels are embedded in every quarter phrase and happen independently to the phrasing generation during gameplay.

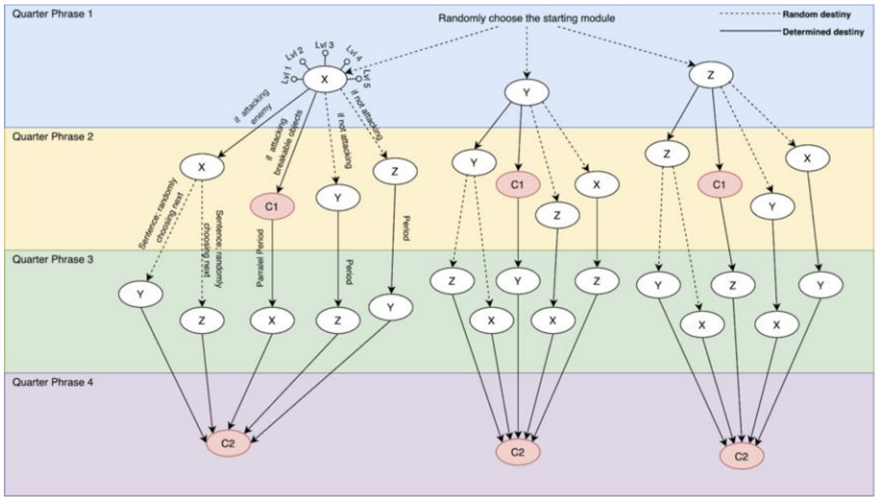


Figure 111 Diagram of the entire state machine of the phrase and diminution generation on *Dizi*.

Module creation rules and procedure

In order to assure the effectiveness of the generative phrases, I made a list of rules for composing good diminutions:

1. Quarter phrases do not have to have fragmentation and variations on every note, so that bits of the modules can sound the same as its adjacent levels, such as episode three's third and fifth levels of the X module in Figure 112.
2. Notes in levels two through five will be kept on the same beat as level one as much as possible. However, they can be occasionally moved to other beats in the measure or omitted or replaced by a nearby pitch.
3. Connections between modules should be carefully considered and designed, regardless of the ordering of the modules in real time. In any way two modules can be placed in succession, the last note of a module and the first note of the following module will never be the same. If a module ends with a longer note, its following module should start with a shorter note for rhythmic contrast. Voice-leading and rhythmic intensity are the two devices I used for designing the connections between these dynamic modules. At the transition, at least one of the two devices should be present.
4. Since X, Y, and Z modules may become the basic idea of a sentence which is repeated in the presentation, careful consideration should be made to the pattern of the motif in order to avoid monotonous repetition.
5. The intensity of diminution in levels two to five needs to be compared only to level one; they can, but not necessarily need to, be compared with each other.
6. As ornamentation level increases, the underlying scale turns from a pentatonic scale to a heptatonic scale by gradually adding more pitches to the melody. On level three, the altered-Gong, or Bian Gong (变宫, see footnote 48) is added to ornament the

melody.[62] On level four, altered-Zhi or Bian Zhi (变徵, see footnote 60) is added. On level five, both altered notes are included so that the maximum diminution level plays with a seven-note scale. The added notes are colored in orange in Figure 112.

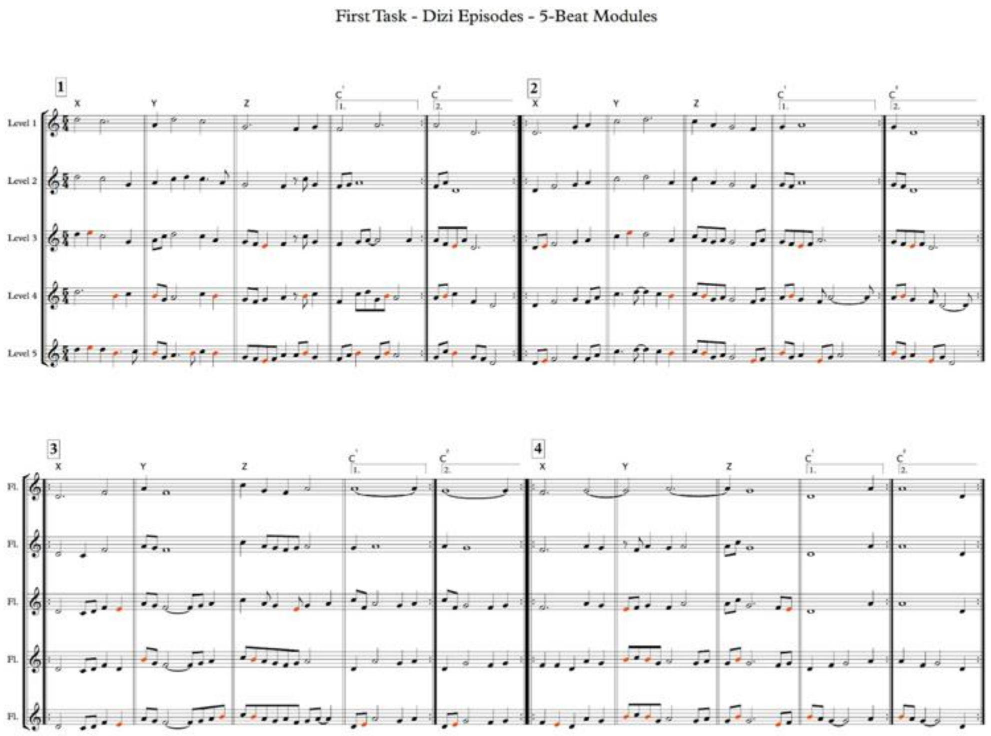


Figure 112 Score of the five diminution levels on the *Dizi* line episode three and four in 5/4 meter.

The scores of all *Dizi* episodes in all three meters and modules' diminution levels are included in Appendix B. Following the written score, I produced all the musical modules in Logic Pro X as shown in Figure 113.

[62] Pentatonic scale with the altered-Gong is also a typical hexatonic scale in traditional Chinese music.

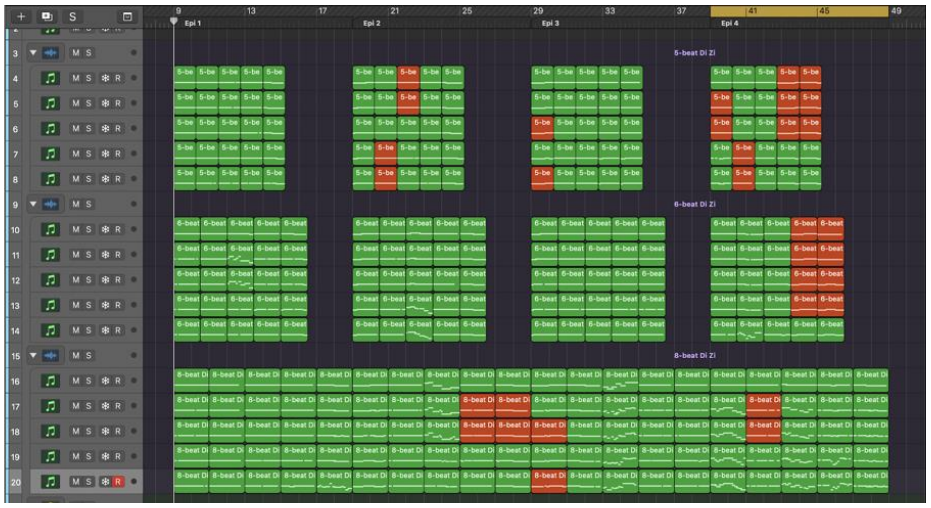


Figure 113 All audio files of *Dizi* modules created in Logic Pro X[63].

Mapping Player Control to Diminution Levels

The player controls the movements of the main game character Yan Ching, as discussed in 4.2.9. Mapping to Player Input and Game States. I grouped the controls into five pairs of states and mapped them to the diminution levels in *First Task*. Because the diminution levels have a logic of expressing different intensities of the ornamentation, I utilized them to reflect the movement intensity of the player.

In order to track the patterns of the player's movement, I created a playtest script that records the total time spent in each group of states. A graphic demonstration of the playtest script is shown in Figure 114. Each pair of states has a timer, which documents the time between when the player enters and leaves the states. The differences between the entering and leaving time are then summed up which becomes the total time spent in that state.

[63] Vertical pairs of red modules represent that they share the exact same diminution pattern.

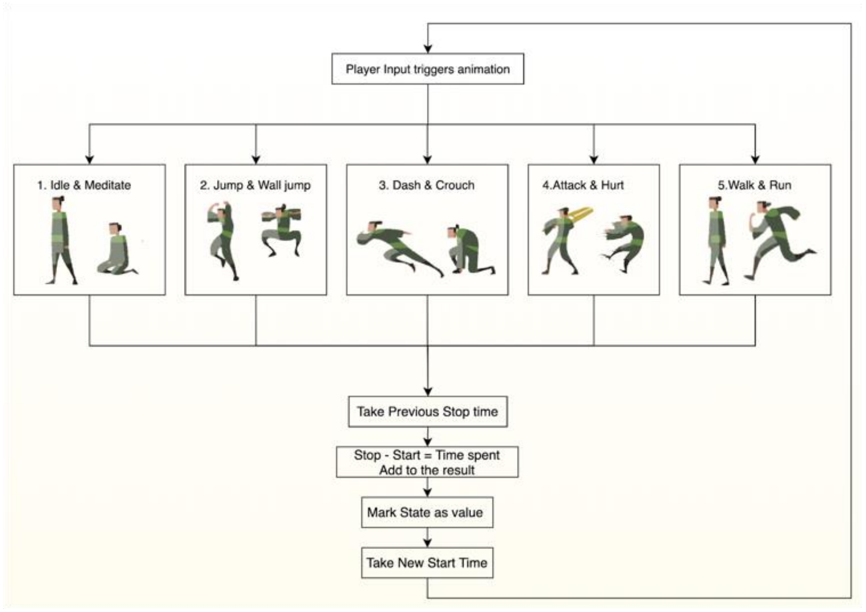


Figure 114 Graphic demonstration of the playtest script.

Although players may decide to end the game early or may fail in the middle of the level due to zero health, duration of the gameplay has no impact on the accuracy of the playtests. Data of two users' three playtests are shown in Figure 115 and Table 19. Table 19 first lists the total time spent in each state group, documented in seconds. In Figure 115, time spent in each pair of states also has a percentage view showing the distribution across the five groups of states in relation to the entire time of the playtest in pie charts. The last row for each player in Table 19 translates the average time spent among their three playtests to the percentage among all types of states. The last row of Table 19 shows the final average percentage of each state among the two players.

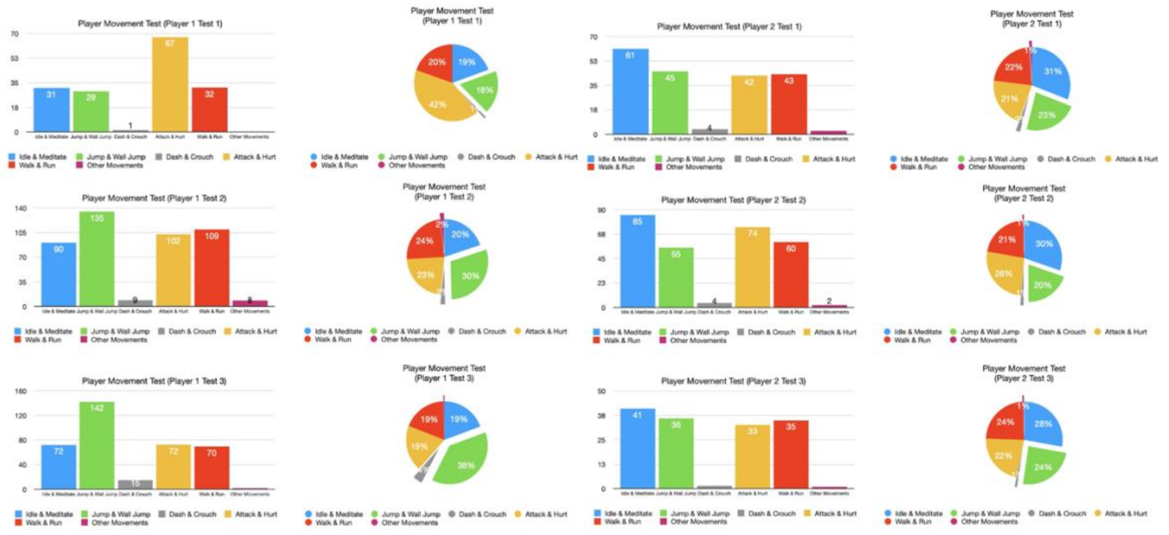


Figure 115 Data from the two users' playtests.

(Duration in seconds)	Playtest #	Idle & Meditate	Jump & Wall Jump	Dash & Crouch	Attack & Hurt	Walk & Run	Other movement
Player 1	1	31.29095	29.07928	1.376404	67.48779	31.50568	0
	2	90.4672	134.7542	9.054383	102.4653	109.2941	8.475677
	3	72.2118	142.4099	15.08417	72.3763	69.50981	1.811478
	Average %	19.3%	28.6%	2.3%	28%	21%	1%
Player 2	1	61.13615	45.3606	3.612754	42.3661	43.37753	2.431263
	2	85.83834	55.24161	3.947807	74.13653	59.65994	2.431263
	3	41.24911	36.02382	1.518918	32.79308	35.25814	0.700779
	Average %	29.6%	22.3%	1.3%	23%	22.3%	1%
Average between 2 players		**24.4%**	**25.4%**	**1.8%**	**25.5%**	**21.6%**	**1%**

Table 19 Data collection of playtests on player movements.

Before the playtests, I ordered the movement intensity among the game states based on real-life human movements. They are, from the least to the most intense movement: Idle and

Meditate, Walk and Run, Jump and Wall-jump, and Attack and Hurt. What's interesting was that the playtest result shows a similar ordering of the in-game durations of the four state pairs.

Although I thought Dash and Crouch would occur frequently, the playtest data show that they occur much less often than Idle and Meditate. One potential reason is that the animation of Meditate is much longer than Dash and Crouch. Another reason is that Idle is the default state, which can be automatically triggered after any other movements or anytime the player stops controlling the game character. Eventually, I merged Dash and Crouch with other states such as Die and set them to the first level of diminution. The following list shows all five levels of diminution with their associated player states:

- Dash & Crouch and other movements = level 1
- Idle & Meditate = level 2
- Walk & Run = level 3
- Jump & Wall Jump = level 4
- Attack & Hurt = level 5

Demonstration of the complete phrase generation procedure in one *Dizi* section is shown in Figure 116. The orange, green, and blue arrows show clearly how phrase structure, diminution level, and module length can be triggered at various time points in real time.

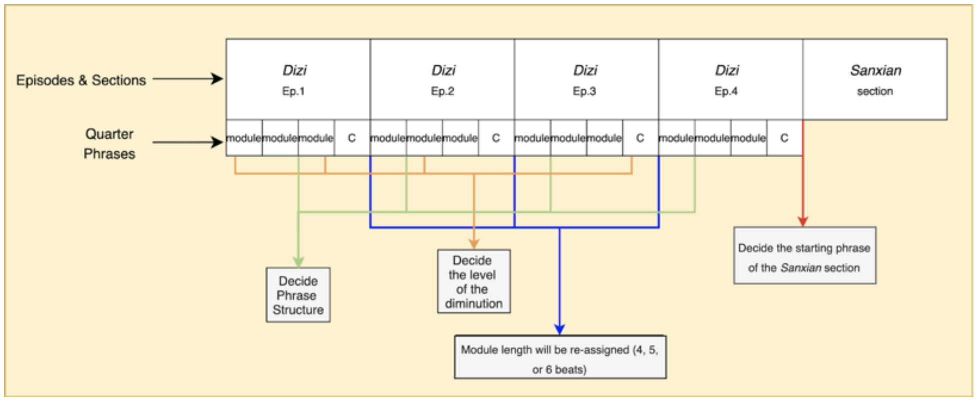
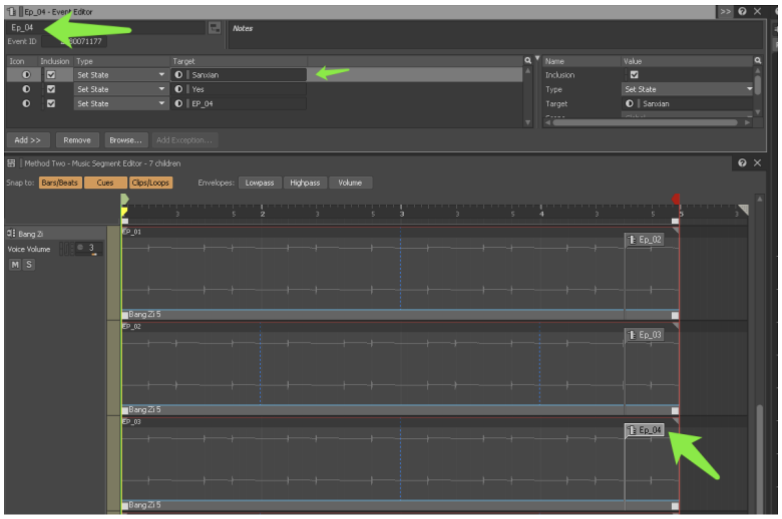

Episodes & Sections
Quarter Phrases
Dizi Ep.1
Dizi Ep.2
Dizi Ep.3
Dizi Ep.4
Sanxian section
module
C
Decide Phrase Structure
Decide the level of the diminution
Module length will be re-assigned (4, 5, or 6 beats)
Decide the starting phrase of the Sanxian section

Figure 116 Complete phrasing generation flow of one *Dizi* section in *First Task*.

4.3.5. The *Sanxian* Section

After the *Dizi* section, the *Sanxian* section is triggered when Wwise sets the "Ep_04" event, as shown in Figure 117. *Sanxian* plays two alternating phrases that adopt the compositional principle of Qi-Cheng-Zhuan-He (起承转合)[64] from traditional Chinese music. *Luo* in this section always plays on the third beats of the last measure. The *Sanxian* layer in this section embodies the Qi-Cheng-Zhuan-He melodic contour. This melody-contouring method fits well here, because its four parts forming one larger phrase perfectly aligns with the structure of four quarter phrases in this movement. This technique has a stylistic root in traditional Chinese music which makes it suitable for arranging melody using the heptatonic scales that come out of the progressive alteration of pentatonic scales.


Ep_04 - Event Editor

Ep_04

Event ID 80071177

Icon	Inclusion	Type	Target
◐	✓	Set State	Sanxian
◐	✓	Set State	Yes
◐	✓	Set State	EP_04

Name	Value
Inclusion	✓
Type	Set State
Target	Sanxian

Method Two - Music Segment Editor - 7 children

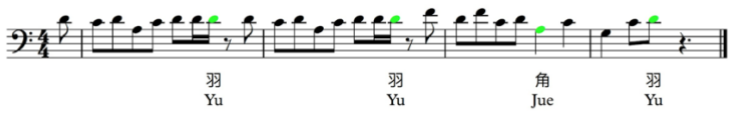
Figure 117 Wwise event "Ep_04" also triggers the *Sanxian* section.

Instead of concerning the melodic contour as in a sentence or period in Western classical practices, Qi-Cheng-Zhuan-He concerns the last note in each quarter phrase, as explained by Liu (2012). The ending tones usually form an AABA[65] or ABCB pattern, with the tonic note repeated

[64] Each Chinese character of Qi-Cheng-Zhuan-He (起承转合) can be translated as: introduction, elucidation of the theme, transition to another viewpoint, and summing up. The four characters together describe a type of melodic shape. Each character represents one quarter of a larger phrase.
[65] Here a letter represents one ending note, not an entire quarter phrase.

most frequently. Like the example given by Liu, a pentatonic scale in the Zhi (徵) mode[66] requires the four quarter phrases to end with the pattern of Zhi-Zhi-Shang-Zhi (徵徵商徵) or Shang-Zhi-Yu-Zhi (商徵羽徵) where a strong emphasis is on the Zhi note. In a melody composed with Qi-Cheng-Zhuan-He type of phrasing, quarter phrases can take any notes from the scale and create any melodic shape and pattern as long as the last notes strictly follow the AABA or ABCB pattern. Although this last-note concern allows using Qi-Cheng-Zhuan-He in combination with Western classical music phrasing of a sentence and period, it is also stylistically common that no repeated motifs exist among all four phrases, in contrast to the repeated basic ideas in a sentence or motifs in a parallel period.

Sanxian Phrase 1



Sanxian Phrase 2

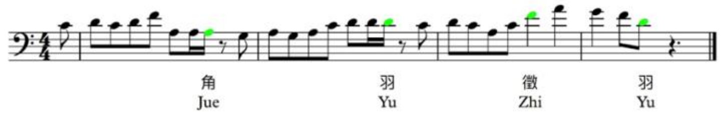


Figure 118 The two *Sanxian* melodies.

The *Sanxian* section in *First Task* takes both the AABA and ABCB forms, constructing two melody variations with the same rhythmic pattern. Since this section centers in D in the Yu mode (refer back to Figure 95) - the fifth mode of F pentatonic scale - the last notes of the four quarter phrases constitute a Yu-Yu-Jue-Yu (羽羽角羽) and Jue-Yu-Zhi-Yu (角羽徵羽) pattern. The Yu note repeats to establish its significance in the scale. In order to show contrast to the *Dizi* section, the *Sanxian* section plays in a fixed meter of 4/4, where each quarter phrase has four beats and no

[66] Zhi mode is the fourth mode of a pentatonic scale. In the fourth mode of C major pentatonic scale (G-A-C-D-E) if explaining in Western classical theory, a Zhi-Zhi-Shang-Zhi (徵徵商徵) pattern means that the four quarter phrases end with G-G-D-G.

diminution levels. The Wwise system will randomly choose one from the two variations when this section is triggered. A score of the two *Sanxian* melodies are shown in Figure 118.

4.3.6. Implementation Methods

There are three methods to implement this generative music procedure in Wwise and Unity. Each method has its own unique highlights and specialties. The first two methods import the XYZC modules as audio samples in the Actor-Mixer Hierarchy and are triggered to play by MIDI files imported in the Interactive Music Hierarchy. The two methods differ, however, in that the XYZC modules are triggered by four specifically assigned MIDI notes in the first method, while the MIDI files contain only two notes in which one triggers the beginning quarter phrase and the other triggers the remaining three modules combined in the second method. The third method places all XYZC modules as sub-tracks in the Interactive Music Hierarchy without a triggering MIDI file.

The strength of the first method is its flexibility: a player's input controls the phrase generation of every quarter phrase. The possible weaknesses of this method, however, reside in both the generation result and the production process. It may generate phrase patterns other than sentence, regular period, and parallel period, since any of the XYZC modules can be placed in the position of four quarter phrases. For instance, a Y-C_1-X-C_2 phrase or a Y-Y-Y-C_2 phrase could be generated. Other possible patterns that can be generated are shown in Figure 119. The difference with phrase structures listed in Figure 111 is clear. The other weakness lies in the production process in that it needs more assets. This method requires a larger number of MIDI files (a total of 45 MIDI files) and audio samples (300 audio samples of XYZC quarter phrases in all episodes, diminution levels, and meters). This method could be a good choice if the composer is open to phrase patterns formed by any combination of the modules, regardless of whether it is forming a sentence or period or of whether there are any cadences.

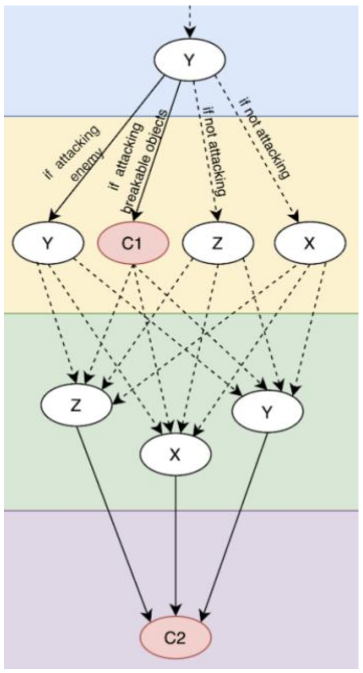


Figure 119 Possible phrase generation paths of implementation method #1.

Both the second and third method of implementation can achieve the pattern result that strictly follows the 15 phrase patterns I designed. Being able to precisely reflect the musical effect the composer was looking for can be a huge advantage. Although having the same musical outcome, the second and third methods still differ dramatically in the way their modules are implemented in Wwise and phrase generation procedure. The second method takes MIDI notes to select and trigger quarter phrases, while in the third method phrase patterns are manually implemented as audio tracks (instead of samples). The sequencing of audio tracks in the third method looks similar to sequencing in DAWs. The timing of switching between different phrase types requires extra programming work using Wwise's states, switches, and events, in comparison to the first two methods.

In the production process, the second method needs a significantly smaller number of assets, which saves space for disc storage - only 216 audio samples and 18 MIDI files. The third method only needs audio samples (300 samples) without any MIDI files, but the implementation procedure involves significantly more work than the other two methods, as samples need to be manually placed into the audio stems with correct algorithmic procedure settings in Wwise.

The third method does have a unique feature that none of the other two methods have: the switching between levels of diminution can be set to musical timing, such as next beat, bar, or random custom cues, instead of responding to game data on every frame. This feature could be useful depending on the composer's preference.

After experimenting with all three methods, I eventually decided to use the second method for the final game build. In summary, the second method involves the least amount of work in the production process, takes the least amount of disc storage, and produces the musical outcome that aligns with my phrasing design of the movement.

Implementation method #1

Although the second method was chosen to be used eventually, I hope to start explaining the process with the first method because it sets the foundational algorithmic structure of the second method.

Each quarter phrase in Wwise is triggered by a specific MIDI note: X module is triggered by C5, Y by D5, Z by E5, and C_1/C_2 by F5, as shown in Figure 120. Each MIDI file contains four MIDI notes representing a full episode. The quarter phrase samples independently follow their episodic order parallel to the MIDI triggering. For example, if MIDI note C5 plays at the very beginning, it will trigger the X module from the episode one sample group. If C5 starts another MIDI file again at the time of the second episode, it will trigger the X module placed in the second episode sample folder. MIDI files also represent the three different meters, with their note duration respectively.

A total of 45 MIDI files were produced for the first implementation method (Figure 121) in order to cover the 15 phrase patterns in all three meters. Wwise selects the appropriate MIDI track by muting or unmuting. This approach allows the change of MIDI tracks to have maximum flexibility with regard to interacting with real time game data. The muting (or unmuting) has no influence on the MIDI file's content (for instance, it cannot send a "Note Off" order). Therefore, muting a MIDI track cannot actually stop the currently playing sample, as the triggering MIDI note remains in "Note On" mode. This feature is a way to compromise the inflexibility of the non-generative MIDI data, while still producing the intended dynamic effect.

I created a soundcaster session in Wwise to test out the first implementation method as shown in Figure 122. I then set up my MIDI keyboard to perform the soundcaster session, since the soundcaster function can be used with external control devices. This is a perfect example of how Wwise can serve as a live performance tool for algorithmic music. A live performance with the soundcaster session of Wwise is recorded in a screen capture video.

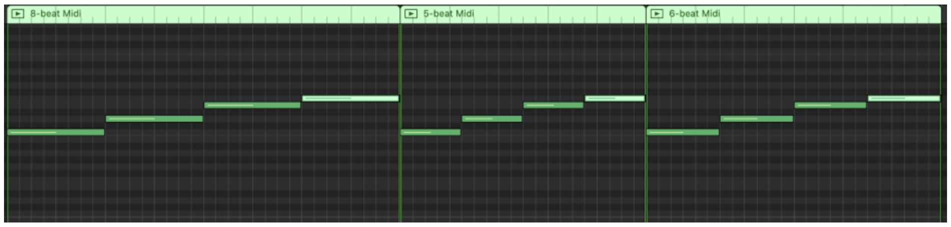


Figure 120 Different MIDI notes triggering XYZC modules.

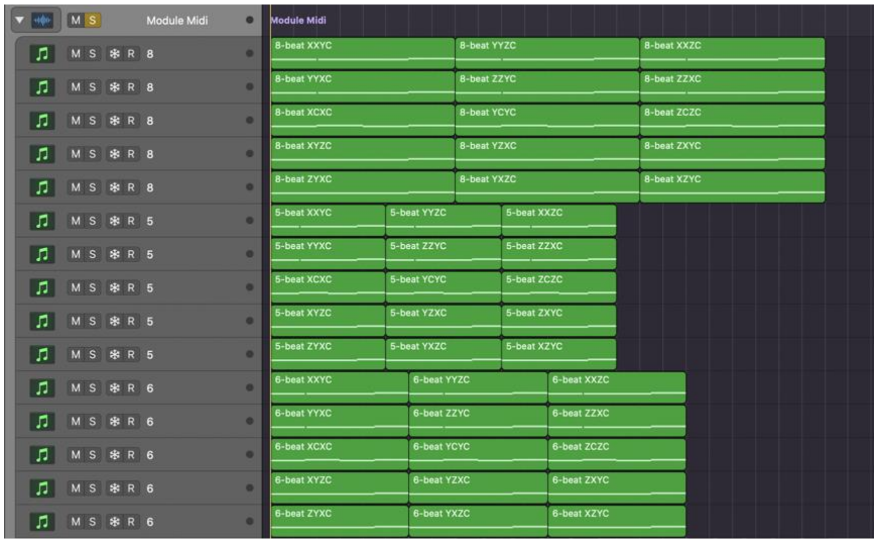
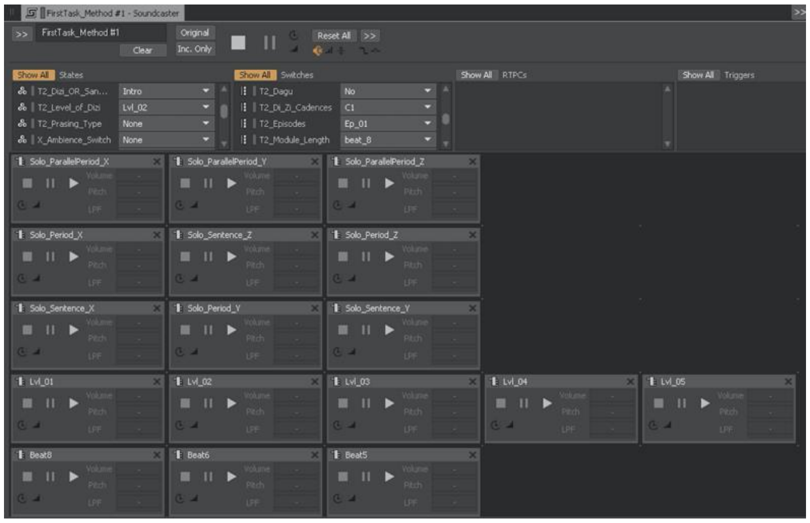


Figure 121 All 45 MIDI files for generative phrasing on *Dizi* used in implementation method one.



T2_Dizi_OR_San...	Intro
T2_Level_of_Dizi	Lvl_02
T2_Prasing_Type	None
X_Ambience_Switch	None
T2_Dagu	No
T2_Di_Zi_Cadences	C1
T2_Episodes	Ep_01
T2_Module_Length	beat_8

Figure 122 The soundcaster session of implementation method one.

Implementation method #2

The second implementation method uses two MIDI notes to represent a complete episode. While the first quarter phrase is randomly chosen by the Wwise engine, real time game data decides the second and remaining quarter phrases. In this method, I placed MIDI notes into two sets of tracks, because they represent different parts of an episode, as shown in Figure 123. The first set of MIDI tracks includes three note values (C5, D5, and E5), representing the X, Y, and Z modules as the starting quarter phrase. These are triggered by Wwise switches instead of mute/unmute, and they hold and trigger a group of Wwise states (Select_X, Select_Y, or Select_Z), which is used for remembering the first quarter phrase when a parallel period phrase type is selected and the first quarter phrase need to be repeated. The second set of MIDI tracks represent the three types of phrase structure (parallel period, sentence, or period). Each track plays one unique long MIDI note: G5 (for triggering sentence), A5 (period), and B5 (parallel period). Different from the first set of MIDI tracks, in which individual tracks are selected to play, all three tracks in the second set play in mute simultaneously, while the phrase type chosen by the game data will be unmuted (Figure 124).

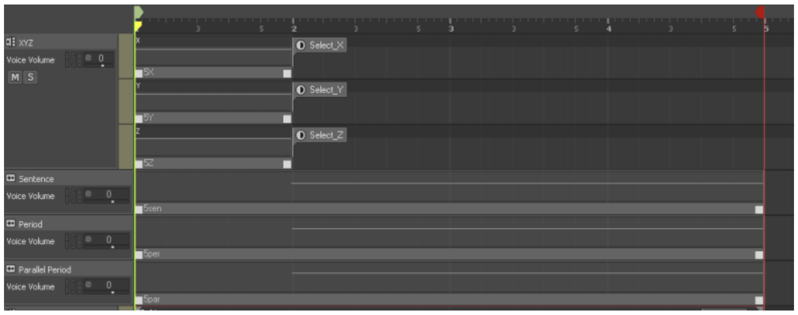
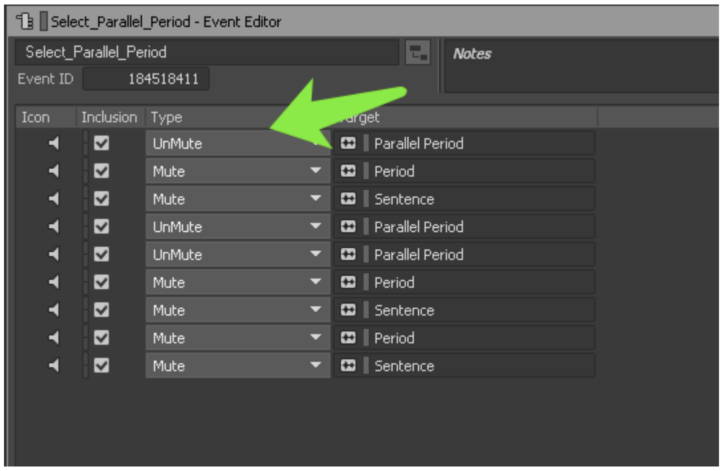


Figure 123 Two sets of MIDI tracks representing the starting quarter phrase and the phrase type.


Select_Parallel_Period - Event Editor

Select_Parallel_Period

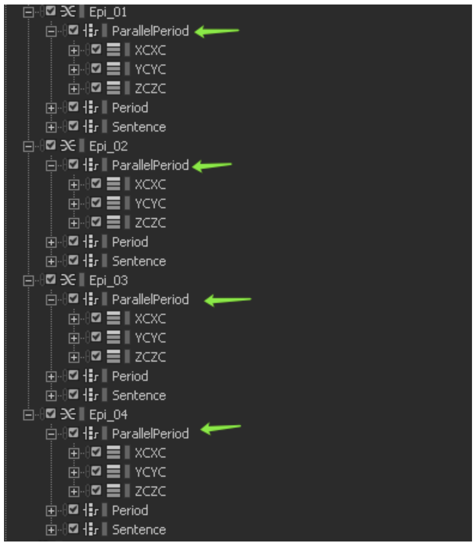
Event ID 184518411

Notes

Icon	Inclusion	Type	Target
	✓	UnMute	Parallel Period
	✓	Mute	Period
	✓	Mute	Sentence
	✓	UnMute	Parallel Period
	✓	UnMute	Parallel Period
	✓	Mute	Period
	✓	Mute	Sentence
	✓	Mute	Period
	✓	Mute	Sentence

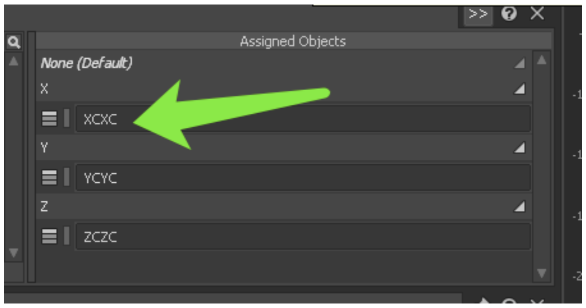
Figure 124 Unmuting associated MIDI track for selecting phrase type.

Here is an example of how a complete episode is generated in real time: if the Wwise engine randomly selects the X module as the first quarter phrase, it will first play the "5X" MIDI track (Figure 123). If the game data requests a phrase type of parallel period, the "Select_X" state will activate the sample group called "ParallelPeriod," in the respective episode folder (Figure 124) in the Actor-Mixer Hierarchy. The "ParallelPeriod" contains all possible music sequences that form a parallel period. Since the "Select_X" state has already set the starting quarter phrase to be the X module, it also selects the parallel period that repeats the X module (Figure 125), forming a complete episode in the pattern of XC_1XC_2.



```
Epi_01
    ParallelPeriod
        XCXC
        YCYC
        ZCZC
    Period
    Sentence
Epi_02
    ParallelPeriod
        XCXC
        YCYC
        ZCZC
    Period
    Sentence
Epi_03
    ParallelPeriod
        XCXC
        YCYC
        ZCZC
    Period
    Sentence
Epi_04
    ParallelPeriod
        XCXC
        YCYC
        ZCZC
    Period
    Sentence
```

Figure 125 Pre-programmed music sequences forming the phrase types in their respective episodes.



```
Assigned Objects
None (Default)
X
    XCXC
Y
    YCYC
Z
    ZCZC
```

Figure 126 The parallel period sequence that repeats the X module.

A phrase that starts with either the X, Y, or Z module as the first quarter phrase has two patterns of sentence and two patterns of period. For instance, if Y is the starting module, the two patterns of sentence are: $YYZC_2$ and $YYXC_2$, and the two patterns of regular period are $YXZC_2$ and $YZXC_2$. These patterns are also programmed as music sequences but are placed inside random containers in Wwise (Figure 127) in order to play randomly following the initial design explained in Figure 111. Thus, six MIDI tracks are necessary to properly produce this generative phrasing in each of the three meters. This method needs a total of 18 MIDI files for the generative phrasing covering all three meters. The entire generation process is also demonstrated in Figure 128.

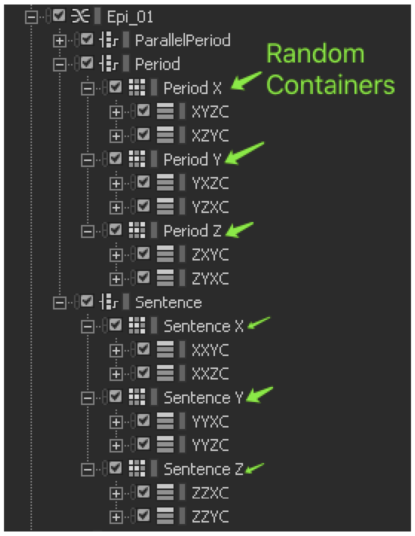


Figure 127 Music sequences of sentence and regular period in random containers.

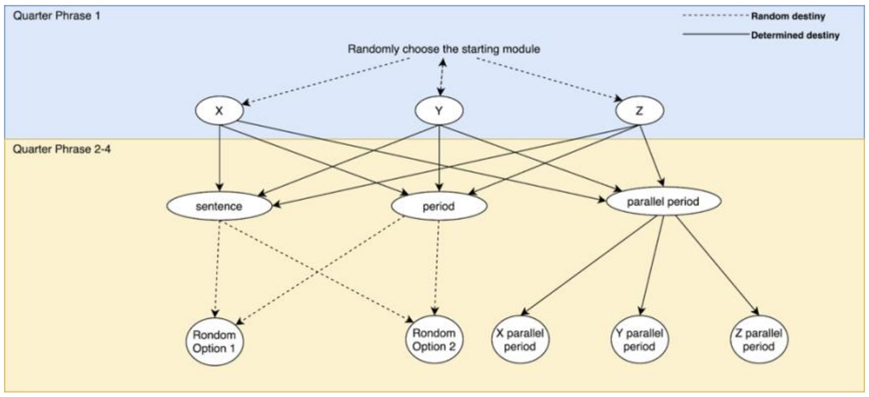


Figure 128 The second method of Implementing generative phrasing in *First Task*.

4.3.7. All Music Parameters in *First Task*: Timing of Changes

Every music parameter has its assigned algorithmic behavior interactively controlled by an associated real time game event. Each dimension of generative content has its unique timing of change. All these timings of change are summarized in Appendix C.

4.4. Real Time Mixing of Music Arrangement

Almost every movement in this project uses the real time mixing technique. Examples of this are the dynamic instrumentation in *The Beginning of the Journey* and switching between diminution levels in *First Task*. In the third movement, *Main Theme,* and the sixth movement, *The Boss*, game-driven interactive real time mixing creates different musical arrangements by combining instrumental layers in various ways. Real time mixing in this movement is similar to the first movement in that different combinations of instrumentation are dynamically triggered. The movements differ, however, in that the content played on each layer of instruments may also vary. In the first movement, every instrument in the primary melody group (*Erhu*, *Pipa*, and *Xiao*) are static audio loops playing the same melody, while in the third and sixth movements, the real time dynamic instrumentation also involves different motifs played on each type of instrument. This creates different arrangements of the piece instead of simple changes in the instrumental timbre.

4.4.1. Main Theme

Four mixing states trigger the four different combinations of instrumental layers in *Main Theme*. The mixing states are also interactively controlled by the player's geographical location in the game world. The instrumental layers include:

1. First *Erhu* playing the melody
2. Second *Erhu* playing a counter melody
3. A variation of the melody played by a *Ruan* and *Guzheng* duet (at the same register as the *Erhu*)
4. A variation of the melody played by a *Ruan* duet (at one octave lower than *Erhu*)
5. An accompanying pattern plucked on *Guzheng*
6. Strings playing in *pizzicato*

7. Strings playing in sustaining long notes
8. A set of percussion instruments playing a groove

The four mixing states are: *Erhu* without Drum, *Erhu* with Drum, *Erhu* Low, and *Erhu* with All, each consisting of different combinations of the instrumental layers, as shown in Table 20.

	Erhu without Drum	*Erhu* with Drum	*Erhu* Low	*Erhu* with All
Strings *pizz*.	☑	☑	☑	☑
Strings sustain	☑	☑	✗	☑
Erhu melody	☑	☑	✗	☑
Erhu counter melody	☑	☑	✗	☑
Ruan & *Guzheng* melody duet	☑	☑	✗	☑
Melody duet by two *Ruans*	✗	✗	☑	☑
Guzheng plucking accompaniment	✗	✗	☑	☑
Percussion set	✗	☑	✗	☑

Table 20 Instrumental layers assigned to each of the four mixing states.

As shown in Figure 129, mixing states are associated with different areas of the *Main Theme* zone. Musical layers are gradually added as the player progresses through the game. The player first enters the *Main Theme* zone at the area boxed in blue where the initial mixing state *Erhu* without Drum plays, introducing a memorable melody (also heard in the title scene) marked by a neutral tone without beats on the drums. The player successfully discovers the *Erhu* with Drum area by executing a challenging jump and climbing on the left wall. If the player discovers the "nice boar" enemy sleeping in the *Erhu* Low area, the music plays a tender and intimate version of the melody at a lower octave on *Ruans* and *Guzheng* with strings *pizz*. to reflect the fact that the nice boar enemy does not attack the player unless it is awakened. When the player progresses to the *Erhu* with Drum game area, the added percussion groove is added, illustrating game progress and encouraging and celebrating the player's success in locating this area. When the player moves to the

Erhu with All area, which is the last area in the *Main Theme* zone and which connects to *The Wolves* zone, the music will play in *tutti* for the effect of a musical climax. This climax foreshadows that the player will soon unlock a new zone.

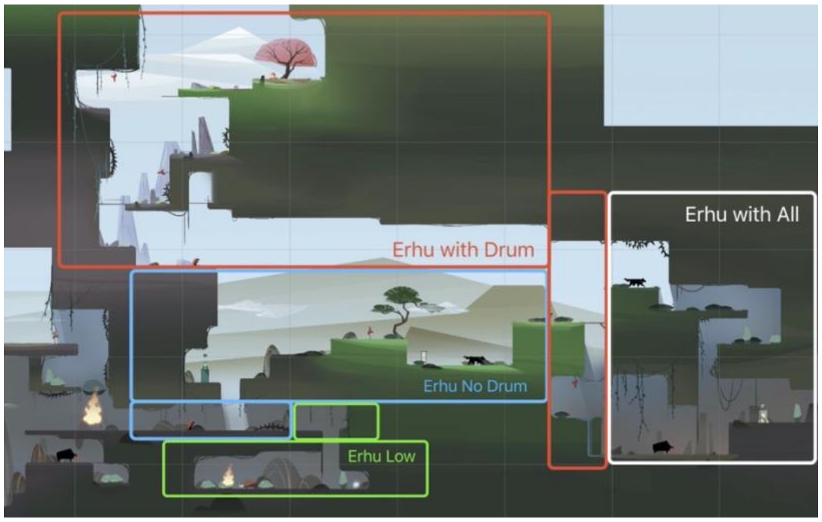


Figure 129 Mixing states with the associated game areas.

4.4.2. Title Theme

The *Main Theme* is also used for the game's opening scene as the *Title Theme* soundtrack. This track is the *Erhu* without Drum mixing state adding an additional layer of percussive patterns played on *Bangzi*, as shown in Figure 130.

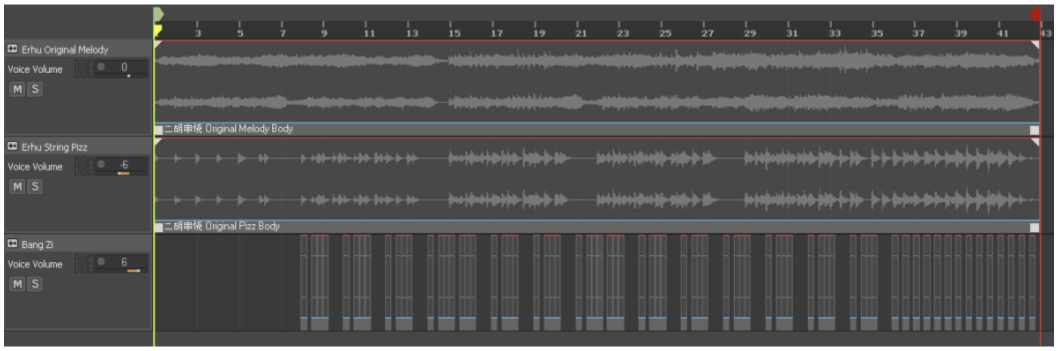


Figure 130 The stem view of all instrumental layers in *Title Theme.*

4.4.3. The Boss

Although *The Boss* movement is the last movement in *Foothill*, it will be discussed in this section because it also utilizes the dynamic mixing technique. In this movement, layers of music are associated with the boss enemy's phases instead of being linked to the player's geographical location. The boss enemy starts in an initial phase but upgrades to a second phase when it loses half of its health (Figure 131). At this point, the boss becomes more aggressive and attacks the player with additional skill. The dynamic mixing musically reflects these two phases. In a horizontal structure, the music contains three sections: an intro, a looping middle section accompanying the indeterminate duration of the battle, and a tail segment. We will discuss the tail segments in chapter 5.8. The intro section consists of four bars of rhythmic pattern played by strings. The looping middle section, which is also where the dynamic mixing is incorporated, contains three layers of instruments (Figure 132). These are:

- Melodies played on *Erhu* and *Pipa* with low strings playing in sustaining notes
- Upper string sections playing a rhythmic pattern of eighth notes
- *Pipa* playing a solo-like, free and improvisatory passage accompanied by dramatic percussion patterns

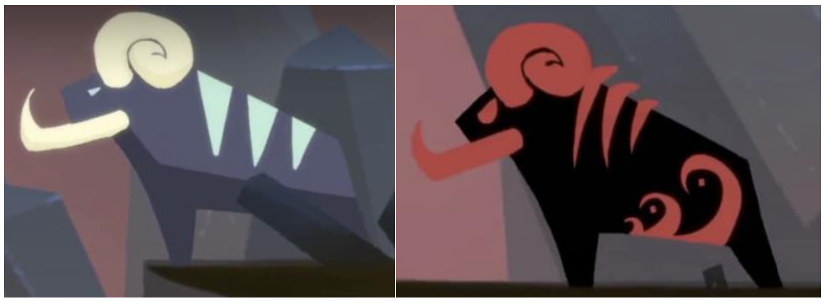


Figure 131 Boss in phase one (left) and phase two (right).

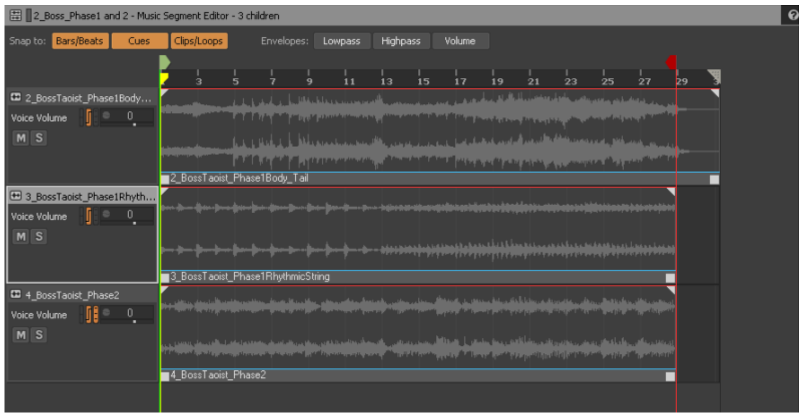
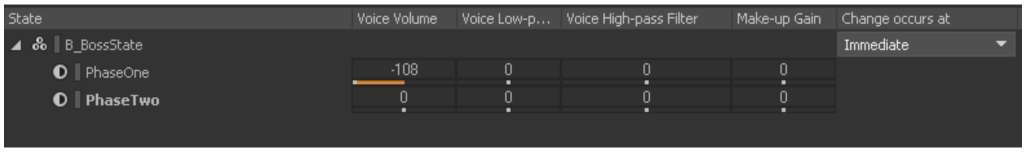

2_Boss_Phase1 and 2 - Music Segment Editor - 3 children
Snap to:
Bars/Beats
Cues
Clips/Loops
Envelopes:
Lowpass
Highpass
Volume
2_BossTaoist_Phase1Body...
Voice Volume
M
S
2_BossTaoist_Phase1Body_Tail
3_BossTaoist_Phase1Rhyth...
3_BossTaoist_Phase1RhythmicString
4_BossTaoist_Phase2
4_BossTaoist_Phase2

Figure 132 Three instrumental layers in *The Boss*.

The looping section of this movement begins with the first two layers of instruments. When the boss enters phase two, the game engine sends a call to trigger the Wwise state "PhaseTwo" which immediately fades in the *Pipa* solo passage with the dramatic percussion pattern (Figure 133). The melody layer is calm and expository and helps to gradually introduce the player to this last and most difficult battle of the level. The rhythmic pattern of eighth notes played on the upper string section brings tension to the movement. When the *Pipa* solo layer and drums come into the mix, they energize the movement and help communicate that the battle has reached a climax. At this point, the boss has lost significant health due to the player's attacks. Additionally, the path to victory has also become more challenging for the player, as the boss begins to attack with more skill and moves much more rapidly than in phase one.

Besides interacting with the boss' phases, the three layers of instruments also gradually fade out as the player loses health. The three layers each have a unique fade-out shape. While the rhythmic strings and the *Pipa* solo with drums can fade out completely, the melody on sustaining lower strings reaches its minimum volume at -2.5 dB, as shown in Figure 134. The three horizontal axes in Figure 134 represent the player's health value. This dynamic velocity design brings interesting musical effects and variety to real time gameplay as well as closely representing the progress of the battle. For instance, even when the music reaches the most fierce and intense mixing state after the boss enemy enters phase two (all three layers playing together), the music can still go back to two layers, or even down to one layer, because the player can lose health under the mixing state thresholds at any time. When the music switches from playing all three layers to only playing the melody with sustaining strings, the musical intensity also drops significantly to express that the player is near to zero health. In this way, soundtrack mixing changes adaptively follow and musically enrich real time nonlinear moving gameplay images.



State	Voice Volume	Voice Low-p...	Voice High-pass Filter	Make-up Gain	Change occurs at
B_BossState					Immediate
PhaseOne	-108	0	0	0	
PhaseTwo	0	0	0	0	

Figure 133 Wwise states for two phases of the boss enemy.

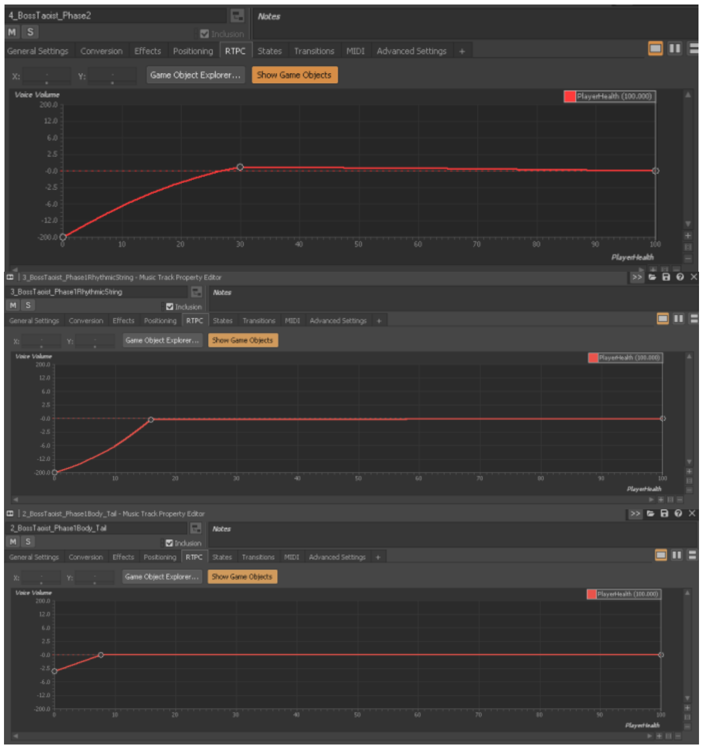


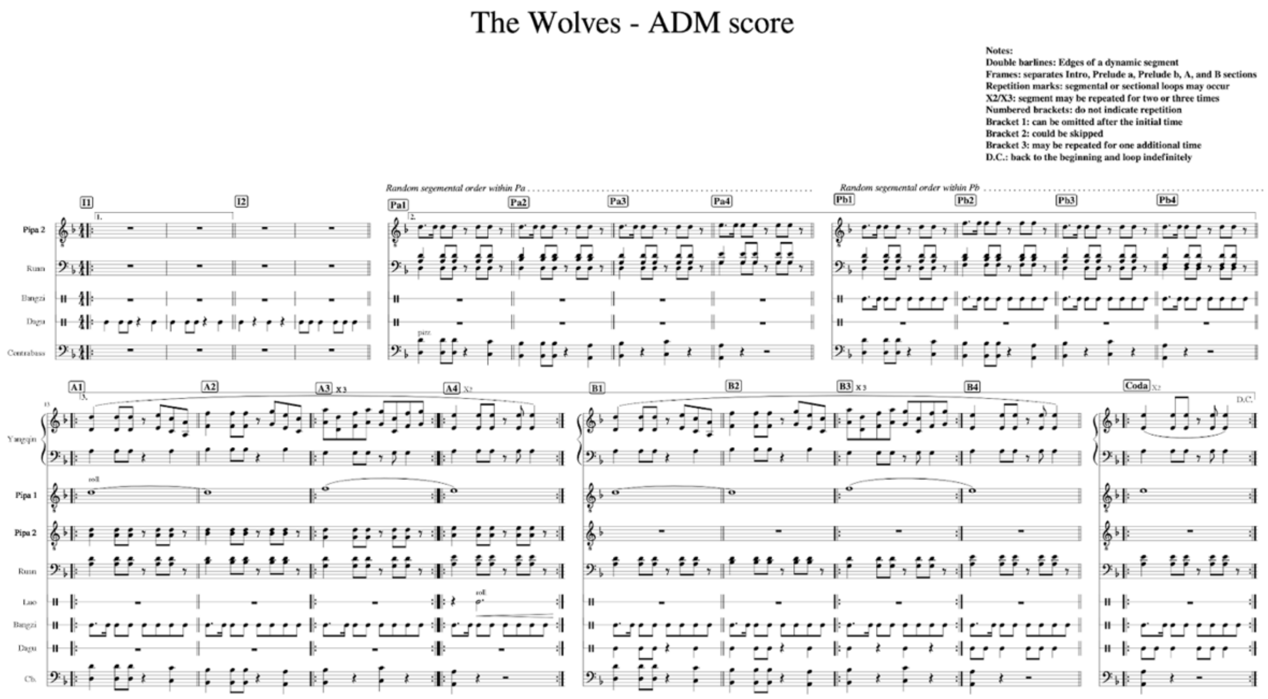
Figure 134 Dynamic velocity of three instrumental layers[67].

[67] Top: *Pipa* solo with drum; middle: rhythmic strings; bottom: melodies with sustaining strings.

4.5. Dynamic Music Sequencing

4.5.1. The Wolves

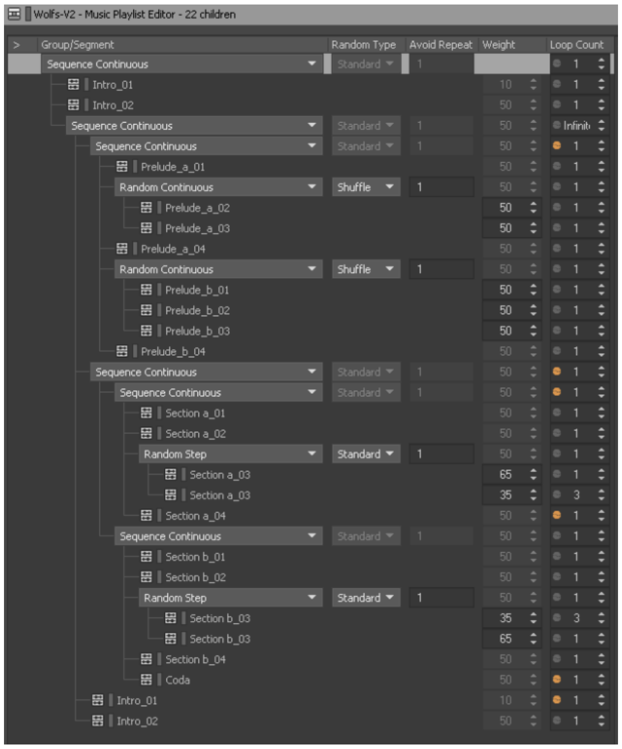
The fourth movement, *The Wolves,* incorporates dynamic arrangements of musical segments and sections in different combinations and sequences in order to form a complete piece. This track involves six musical sections: intro, prelude a, prelude b, section A, B, and coda. Within each section there are also smaller musical segments that can be further arranged dynamically. Other than the fact that the dynamic segments in the intro are two measures long, dynamic segments in other sections are always one measure in length, as shown by the double bar lines in Figure 135. The instrumentation of this piece includes Yangqin, *Pipa*, *Ruan*, *Luo*, *Dagu*, *Bangzi*, and the contrabasses. Although a major part of this movement does not interact with the player's real time game input, it still contains the interactive transition and tail segment which will be discussed in 4.7. All Movements: Transitional Passages and Tail Segments.



Notes:
Double barlines: Edges of a dynamic segment
Frames: separates Intro, Prelude a, Prelude b, A, and B sections
Repetition marks: segmental or sectional loops may occur
X2/X3: segment may be repeated for two or three times
Numbered brackets: do not indicate repetition
Bracket 1: can be omitted after the initial time
Bracket 2: could be skipped
Bracket 3: may be repeated for one additional time
D.C.: back to the beginning and loop indefinitely

Figure 135 ADM score of *The Wolves.*

The algorithmic procedure can be indicated by the ADM score but is more clearly shown by the playlist container edited in Wwise (Figure 136). The movement starts with playing the two modular segments of the intro section in order. These are labeled I1 and I2 in Figure 135 and "Intro_01" and "Intro_02" in Figure 136, as placed in a Sequence Continuous folder. After I2 finishes, the music switches to another Sequence Continuous folder containing two sub–Sequence Continuous folders for the prelude a and prelude b sections (Pa and Pb in Figure 135). Within the two prelude sections, the Pa2 and Pa3 segments are also placed in a Random Continuous folder, which results in the random ordering of this segment in real time. A similar setting was applied to Pb1, Pb2, and Pb3 segments. The yellow dot next to the lines of musical processes represent randomness in its loop count. The yellow dot above Pa1 sets the prelude section to randomly loop once or not at all. The music then continues to the A and B sections which may loop for one or two times. Within the A and B sections, A1 and A2, A4, and the Coda may each be looped either once or twice. The A3 and B3 segments can be played either once or three times, as the two options are contained in Random Step folders. Once the playlist finishes playing the Coda, the intro segments are brought back, but I1 has a 50% chance to be omitted. The section from the prelude to the intro sections repeat will then loop indefinitely as long as the player stays in the soundtrack's associated game area.



Wolfs-V2 - Music Playlist Editor - 22 children

Group/Segment	Random Type	Avoid Repeat	Weight	Loop Count
Sequence Continuous	Standard	1		1
Intro_01			10	1
Intro_02			50	1
Sequence Continuous	Standard	1	50	Infinit
Sequence Continuous	Standard	1	50	1
Prelude_a_01			50	1
Random Continuous	Shuffle	1	50	1
Prelude_a_02			50	1
Prelude_a_03			50	1
Prelude_a_04			50	1
Random Continuous	Shuffle	1	50	1
Prelude_b_01			50	1
Prelude_b_02			50	1
Prelude_b_03			50	1
Prelude_b_04			50	1
Sequence Continuous	Standard	1	50	1
Sequence Continuous	Standard	1	50	1
Section a_01			50	1
Section a_02			50	1
Random Step	Standard	1	50	1
Section a_03			65	1
Section a_03			35	3
Section a_04			50	1
Sequence Continuous	Standard	1	50	1
Section b_01			50	1
Section b_02			50	1
Random Step	Standard	1	50	1
Section b_03			35	3
Section b_03			65	1
Section b_04			50	1
Coda			50	1
Intro_01			10	1
Intro_02			50	1

Figure 136 The playlist container for *The Wolves* in Wwise.

4.6. Dynamic Modulation

4.6.1. Before Boss

The fifth movement *Before Boss* incorporates dynamic modulation that is triggered interactively by the player's geographical location in the platform. The instrumentation of this movement includes *Dizi*, *Xiao* (solo and duo), *Bawu*, and *Guzheng*.

This movement builds on a music segment of 48 bars in length with the note A starting the top melody on *Dizi* accompanied by arpeggios played on the *Guzheng* (Figure 137). The 48-bar segment loops indefinitely for the indeterminate duration the player spends in the associated game area. The 48 bars are divided into four eight-bar harmonic progression repeated six times. Chord symbols are labeled in the score of Figure 137. A counter melody played on a solo *Xiao* is introduced when the eight-bar harmonic progression is looped for the second time, or at m.9. This counter melody continues until m.24. A melodic dialogue between two *Xiaos* is introduced between mm.33 - 40, the second to last loop of the eight-bar harmonic progression in the 48 bars. Lastly, a melodic phrase played on *Bawu* is introduced at the last loop of the eight-bar looping harmonic progression in mm.41-48. This 48-bar segment is later modulated to seven different keys where the *Dizi* melody starts with notes C, D, E, B, F#, G, and Bb[68] in each key, as shown in Figure 138.

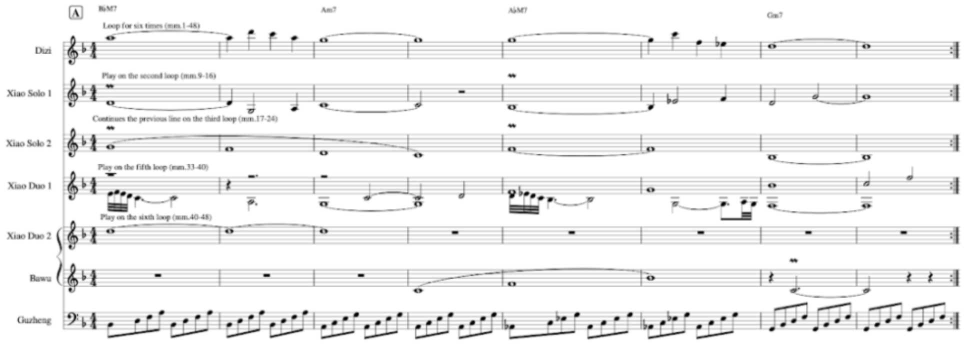

A
BbM7
Am7
AbM7
Gm7
Loop for six times (mm.1-48)
Play on the second loop (mm.9-16)
Continues the previous line on the third loop (mm.17-24)
Play on the fifth loop (mm.33-40)
Play on the sixth loop (mm.40-48)
Dizi
Xiao Solo 1
Xiao Solo 2
Xiao Duo 1
Xiao Duo 2
Bawu
Guzheng

Figure 137 ADM score of *Before Boss* in A.

[68] In the following sections, I will their first melodic note played on the *Dizi* to label a key, instead of the chordal root of the chord playing beneath. This is because this musical segment involves sequential passage that can be argued to center in multiple tonic chords. The starting melodic note is actually the chordal seventh of the first harmony - a major seventh chord. This labeling of keys also highlights the importance of melody so that each starting note in the melody represents a tonic-feel role to its underlying harmonic structure.

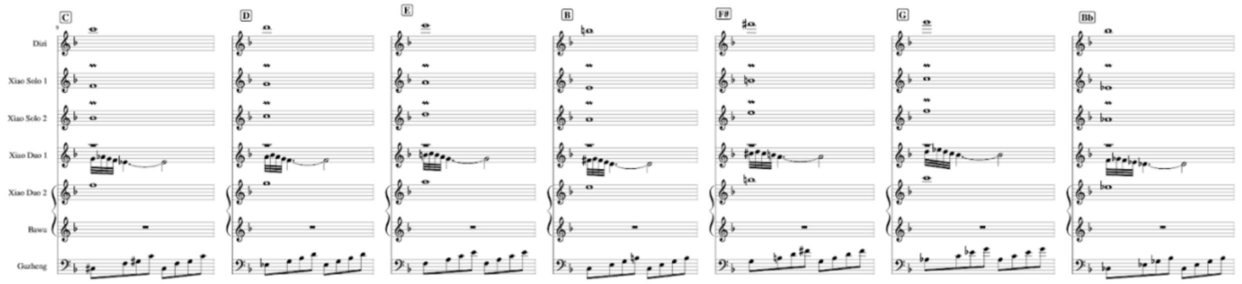


Figure 138 ADM modulation of *Before Boss*.

4.6.2. Triggering Conditions

The eight different keys are triggered by the player's geographical location in the game, a map of which is shown in Figure 139. As the player moves to the right and gets closer to the boss area, the music shifts from lower keys to higher keys, and does the opposite when the player moves away. This ascension in modulation also reflects the level design of this area that the player ascends until reaching the boss enemy area. In the areas of the red, blue, and yellow boxes in Figure 139, one out of three assigned keys is randomly chosen to play. The movement begins in the key of A directly following the conclusion of *The Wolves* movement because it is the lowest key among the eight.

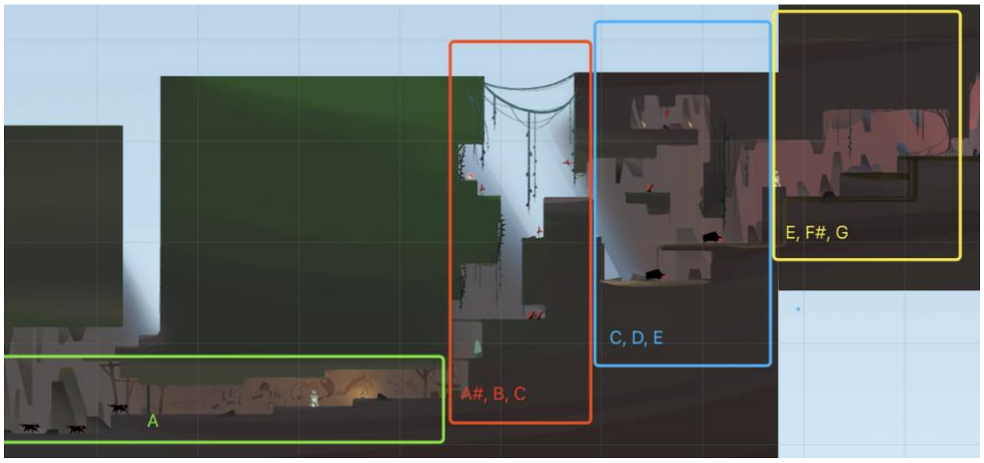
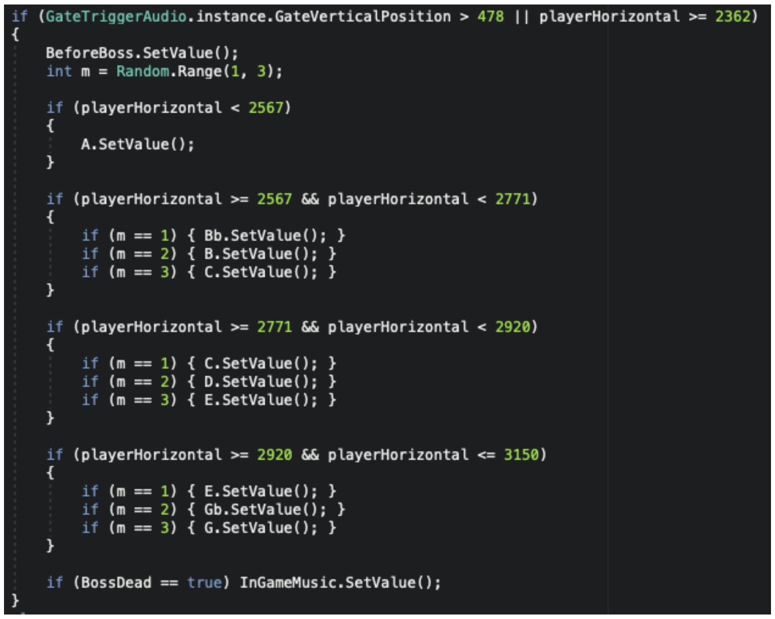


Figure 139 *Before Boss* area with eight modulations.

The game engine is programmed so that it will continuously update the player's location. When the location data meet one of the music triggering conditions, a call is sent to the Wwise engine which triggers the associated modulations (Figure 140). Six Wwise' "custom cues" are placed at every eight bars to execute the modulations (Figure 141). This step helps to allow the music to modulate only after a complete six measures. As one quarter note in BPM120 is 500 milliseconds, a six-bar phrase is thus about 12 seconds in length. A six-bar phrase gives enough time for a complete musical statement (uninterrupted phrasing) and suits the gameplay timing well. When the player is exploring inside the area of these colored boxes, at least one modulation or one key change occurs, but the frequency of modulation is not too rapid so as to sound tonally unstable or ambiguous. If players spend longer than 12 seconds in a colored area, they will hear additional random modulations that are assigned to the region.



```
if (GateTriggerAudio.instance.GateVerticalPosition > 478 || playerHorizontal >= 2362)
{
    BeforeBoss.SetValue();
    int m = Random.Range(1, 3);

    if (playerHorizontal < 2567)
    {
        A.SetValue();
    }

    if (playerHorizontal >= 2567 && playerHorizontal < 2771)
    {
        if (m == 1) { Bb.SetValue(); }
        if (m == 2) { B.SetValue(); }
        if (m == 3) { C.SetValue(); }
    }

    if (playerHorizontal >= 2771 && playerHorizontal < 2920)
    {
        if (m == 1) { C.SetValue(); }
        if (m == 2) { D.SetValue(); }
        if (m == 3) { E.SetValue(); }
    }

    if (playerHorizontal >= 2920 && playerHorizontal <= 3150)
    {
        if (m == 1) { E.SetValue(); }
        if (m == 2) { Gb.SetValue(); }
        if (m == 3) { G.SetValue(); }
    }

    if (BossDead == true) InGameMusic.SetValue();
}
```

Figure 140 All eight keys and their triggering conditions in Unity.

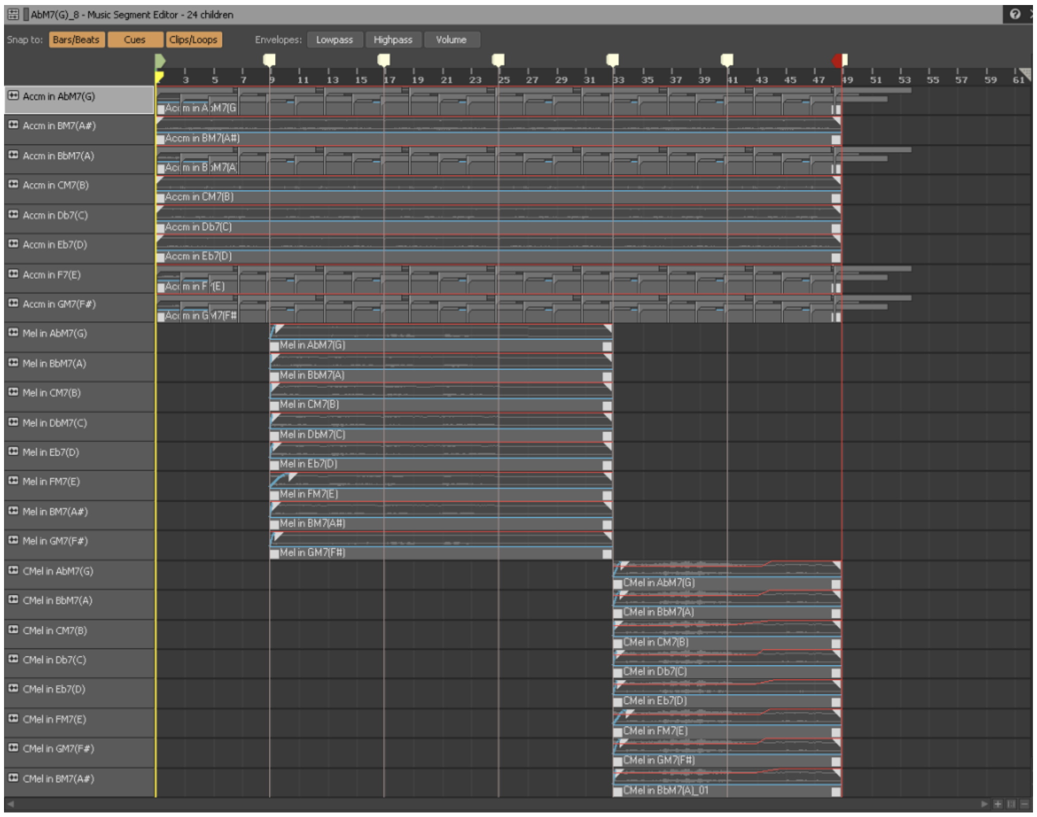
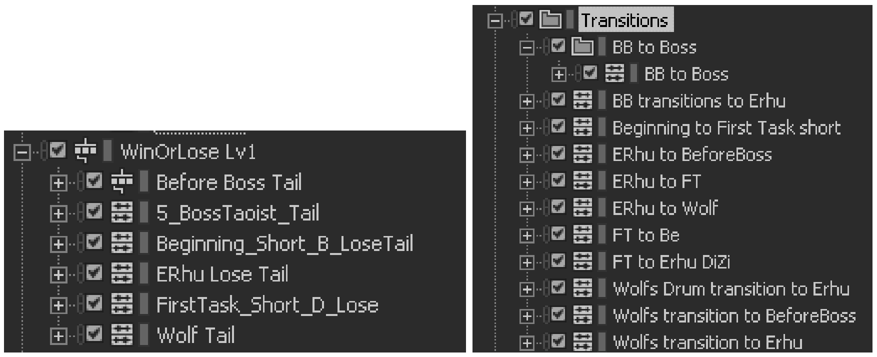


Figure 141 Custom cues placed on every eight bars.

4.7. All Movements: Transitional Passages and Tail Segments

There are dynamic transitions and tail segments incorporated in all six movements in *foothill*. Dynamic transitions connect two movements in any direction, whether backward or forward. Thus, between every two movements two transitional passages occur. They function to transit forward from track A to track B and also backward from track B to track A, as the player can decide to move in either direction at any time of their choosing. When a movement involves dynamic instrumentation, meaning that multiple versions of the movement will need transitions such as *The Beginning of the Journey*, the transitional passages also use the same instrumentation and start with the same tempo in order to transition smoothly. Transition may gradually adjust to the new track's tempo. A transitional passage may also function to finish a movement, thus creating a short break after the conclusion of the movement before the new track begins to play. When the initially triggered movement contains MIDI tracks, the MIDI tracks will be muted when the transitional passage is triggered in order to avoid undesired music discord.

Tail segments function as an end to every movement when the player fails the game (reaches zero health). Each movement has one tail segment except *The Boss*, which has an additional tail segment for the scenario wherein the player defeats the boss enemy. All transitional passages and tail segments are implemented in Wwise as shown in Figure 142.


WinOrLose Lv1
- Before Boss Tail
- 5_BossTaoist_Tail
- Beginning_Short_B_LoseTail
- ERhu Lose Tail
- FirstTask_Short_D_Lose
- Wolf Tail

Transitions
- BB to Boss
  - BB to Boss
- BB transitions to Erhu
- Beginning to First Task short
- ERhu to BeforeBoss
- ERhu to FT
- ERhu to Wolf
- FT to Be
- FT to Erhu DiZi
- Wolfs Drum transition to Erhu
- Wolfs transition to BeforeBoss
- Wolfs transition to Erhu

Figure 142 All Transitional passages and tail segments programmed in Wwise.

4.7.1. Transitional Passages and Triggering Conditions

There are a total of ten transitional passages inserted between the six movements, and these are interactively triggered by gameplay data. In order to create a seamless blend between the transitions and the movements, the two overlap (Figure 143). For instance, if the transition ends on a single note played on the *Pipa*, the intro section of the following movement begins during the same measure. Transitional passages' linked game events may occur at any time, whether the player has crossed the border of two game areas or has successfully defeated all the enemies in a room. Thus, the triggering of the transitional passages may also occur at random points during the soundtrack (e.g., measure three beat two, or beat four of measure 11). The transition passages need to sound musically smooth no matter which measure or beat they are triggered on. Additional procedures help to smooth out the seams, these include crossfades between the two connected pieces of music.

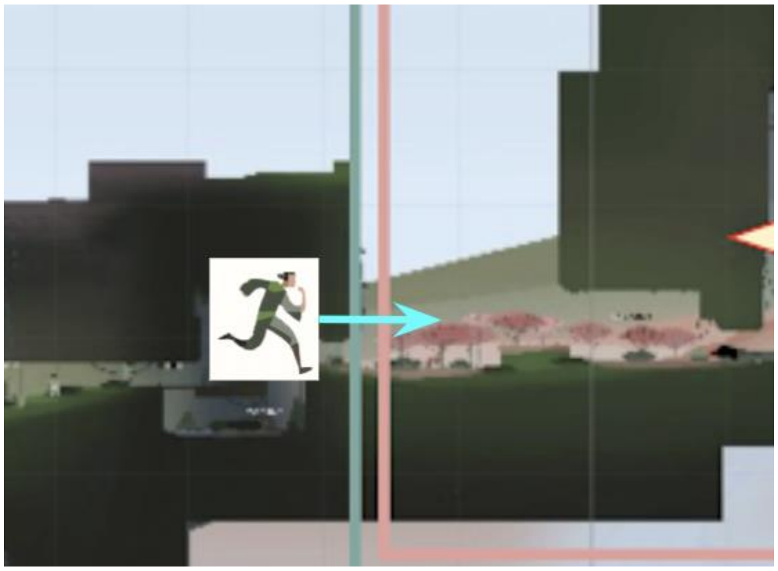


Figure 143 An example of seam between two movements located in the platform.

Table 21 lists and describes the ten transitional passages, including those with dynamic content within the transition itself, their respective triggering conditions, and other features. Among the ten transitions, only transition #1, #2, #3, and #10 contain dynamic content. Additionally, transition #7 is the only one that uses a game trigger other than the player's location in the platform. This is because the game area associated with *The Wolves* is a locked room which only allows players to proceed further after they have successfully defeated each wolf enemy. Thus, hearing the transitional passage to *Before Boss* also communicates that the player has made another achievement and is unlocked from the wolves' room. Whether a transition starts on an upbeat or ends with an overlap to the next track (or break) is also explained under the table's last column.

Transition #	**Dynamic content**	**Triggering conditions**	**Other features in the transitional passage**
#1. *The Beginning of the Journey → First Task*	Dynamic instrumentation on melody: *Erhu*, *Pipa*, or *Dizi*	Player's location in the platform	Unmutes MIDI tracks in *The Beginning of the Journey* for a clean transition. This transition starts with 1.5 beats of pickup thus it also subtracts 1.5 beats from *The Beginning of the Journey*
#2. *First Task → The Beginning of the Journey*	Dynamic instrumentation on melody: *Dizi* or *Sanxian*	Player's location in the platform	The track has a *Ritardando* to connect the faster tempo in *First Task* with the slower tempo in *The Beginning of the Journey*
#3. *First Task → Main Theme*	Dynamic instrumentation on melody: *Dizi* or *Sanxian*	Player's location in the platform	The transitional passage concludes the *First Task* track, and begins the *Main Theme* track after a short break
#4. *Main Theme → First Task*	Static material within the transitional passage	Player's location in the platform	The transitional passage starts with two pickup beats, so it enters on the third beat of a measure in *Main Theme*. The transition concludes the *Main Theme* track and begins the *First Task* track after a short break. The transition also contains a *ritardando*.
#5. *Main Theme → The Wolves*	Static material within the transitional passage	Player's location in the platform	The transitional passage starts with two pickup beats, so it enters on the third beat of a measure in *Main Theme*. The last note of the transitional passage overlaps with the first note of *The Wolves*. The transition also contains a *ritardando*.
#6. *The Wolves → Main Theme*	Static material within the transitional passage	Player's location in the platform	The transitional passage ends with a drum phrase where its last note overlaps with the first note of *Main Theme*.
#7. *The Wolves → Before Boss*	Static material within the transitional passage	The door of the wolves' room reopens after player defeated all wolves	This transition ends with an *Erhu* melody that helps to smoothly blend into the *Before Boss* track. The last note of the *Erhu* melody overlaps with the first note of *Before Boss*.

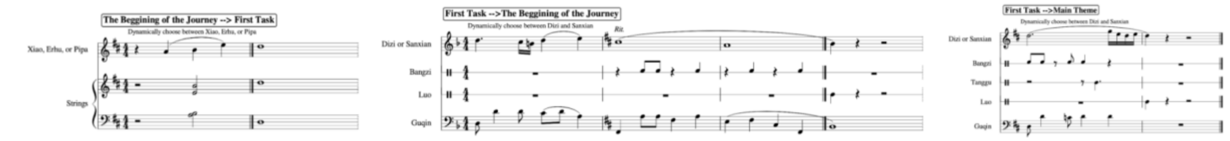
**Before Boss → The Wolves*	N/A	N/A	This transition is omitted because *The Wolves* is only played once when there are wolf enemies. When the music switches to *Before Boss*, it implies that the wolves are all defeated. The wolf enemies will not be respawned after death. Thus, when the player moves from the *Before Boss* area backwardly, the next soundtrack to transition to is the *Main Theme.*
#8. *Before Boss → Main Theme*	Static material within the transitional passage, in the key of A	Player's location in the platform	This transitional passage uses the same drum phrase on transition #6. The last note of the transitional passage overlaps with the first note of *Main Theme*.
#9. *Main Theme → Before Boss*	Static material within the transitional passage	Player's location in the platform	Since *The Wolves* will be skipped once the player defeats all wolf enemies, this transition is also needed. This transition uses the same *Erhu* melody with transition #7 as the ending in order to connect smoothly to *Before Boss.*
#10. *Before Boss → The Boss*	Dynamic transitional passages for the three keys (E, F#, G) and their four harmonies	Player's location in the platform	The last note of this transition overlaps with the first note of *The Boss*.

Table 21 Information of all the transitional passages between movements.

Any transitional passages that do not start on an upbeat have been programmed using Wwise to only execute after a complete measure of the current movement. This procedure helps to maintain consistency of measure length and soundtracks' rhythmic groove. The overlap between transitional passages and the subsequent movements is set by placing the "Entry Cue" of the "Exit Cue" cursors to their appropriate positions in the tracks.

Figures 144 to 148 show all transitional passages in written notation. The final bar lines in each score mark the last measure that the transition passage plays alone. The dotted bar lines on the following measures represent the measure at which the following movement enters. No measure following a final bar line indicates a break in the transition. Just like in other movements discussed in previous sections, dynamic instrumentation is used in the transition. Specific instrumental layers have been programmed in Wwise to adjust up or down in volume during the transition.

Because the yellow region (Figure 139) of the *Before Boss* area includes three dynamic keys, the transition passage to *The Boss* movement also has three dynamic keys. Additionally, the 48-bar looping section of the *Before Boss* movement has four distinct harmonies within each key. Thus, a unique transition passage is specifically composed for each harmony. This is illustrated in Figure 147, which contains a total of twelve transitional passages for the key of E, F#, and G, and a total of four transitional passages for the key of A shown in Figure 148.



The Beginning of the Journey --> First Task
Dynamically choose between Xiao, Erhu, or Pipa
Xiao, Erhu, or Pipa
Strings
First Task -->The Beginning of the Journey
Dynamically choose between Dizi and Sanxian
Rit.
Dizi or Sanxian
Bangzi
Luo
Guqin
First Task -->Main Theme
Dynamically choose between Dizi and Sanxian
Dizi or Sanxian
Bangzi
Tanggu
Luo
Guqin

Figure 144 Transitional passages involving *The Beginning of the Journey* and *First Task*.

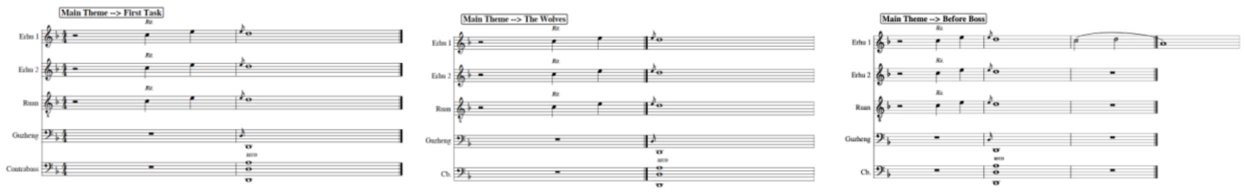


Figure 145 Transitional passages involving *Main Theme*.

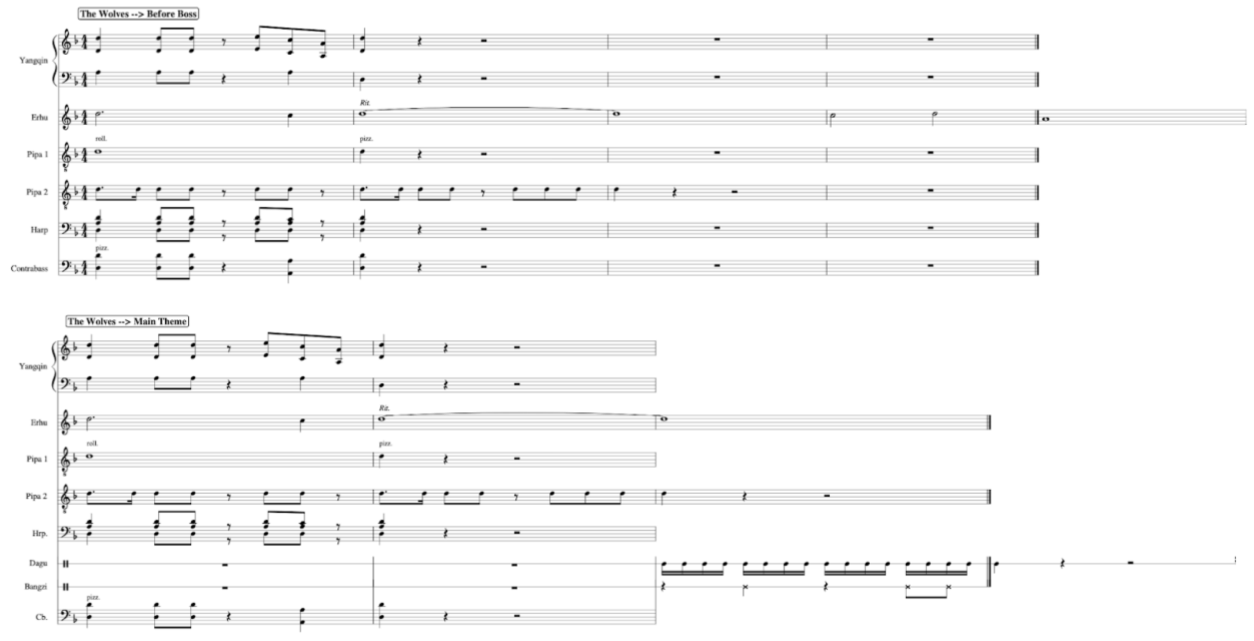


Figure 146 Transitional passages involving *The Wolves*.

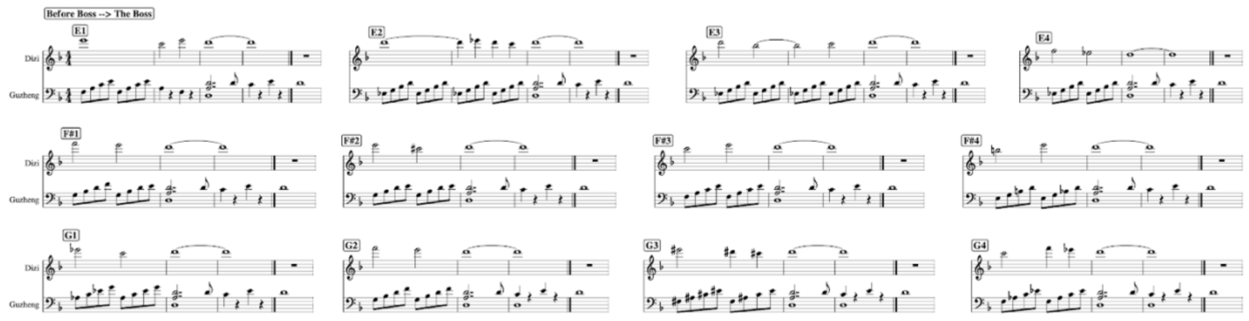


Figure 147 Transitional passages in three dynamic keys and four harmonies between *Before Boss* and *The Boss.*

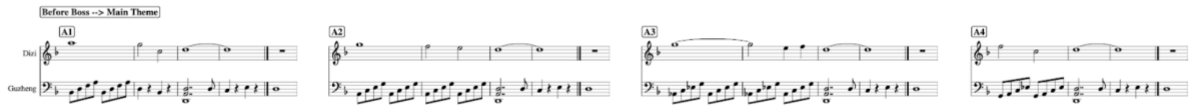


Figure 148 Transitional passages of the four harmonies in the key of A between *Before Boss* and *Main Theme*.

4.7.3. Tail Segments and Triggering Conditions

A total of seven tail segments are used in this game level. While each of the six musical movements are assigned one "defeat" tail triggered when the player dies due to zero health. *The Boss* track has an additional tail for when the player is victorious. The defeat tails contain the same instrumentation as their associated musical movements but occasionally consist of different instrumentation or additional instrumental layers. When a player dies, a brief death animation plays. After the animation finishes, the player respawns at the last saved position. Because of this, tail segments should have a similar length to the death animation in order that the background soundtrack can be cued as soon as the player is respawned. This also requires that the triggering of the tail segments be prompt.

Figures 149 and 150 show all seven tail segments in written notation. Because the *Before Boss* track contains eight dynamic tonalities, there are eight associated versions of the tail segment composed in the eight keys, as shown in Figure 149. The tail segments are set to be triggered at the "next bar" of the currently playing soundtrack in Wwise, which is the soonest triggering position that allows music to respond to game data quickly while maintaining a musically meaningful transition.

It is worth noting that after the victory tail on *The Boss* finishes playing, the music switches back to the *Before Boss* track after a short break, as shown in Figure 151. This is to continue the background music if the player decides to linger instead of immediately exiting the level.

If the player decides to go back in the game after defeating the boss, the soundtracks assigned to each game area will stay active to be dynamically triggered by the player's position. Thus, when the player goes all the way back to the *First Task* area, for example, the *First Task* track will still be triggered. However, *The Boss* as well as *The Wolves* track can only be played once and will no longer be available for playback once the player defeats the wolves and boss. When the player triggers the last meditation pavilion next to the boss area, the scene will switch to the title scene (or next level, if more levels are produced later), and the currently playing soundtrack will fade out to the *Title Theme*.

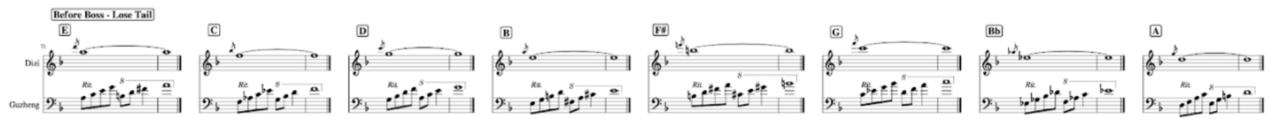


Figure 149 *Before Boss* tail segments.

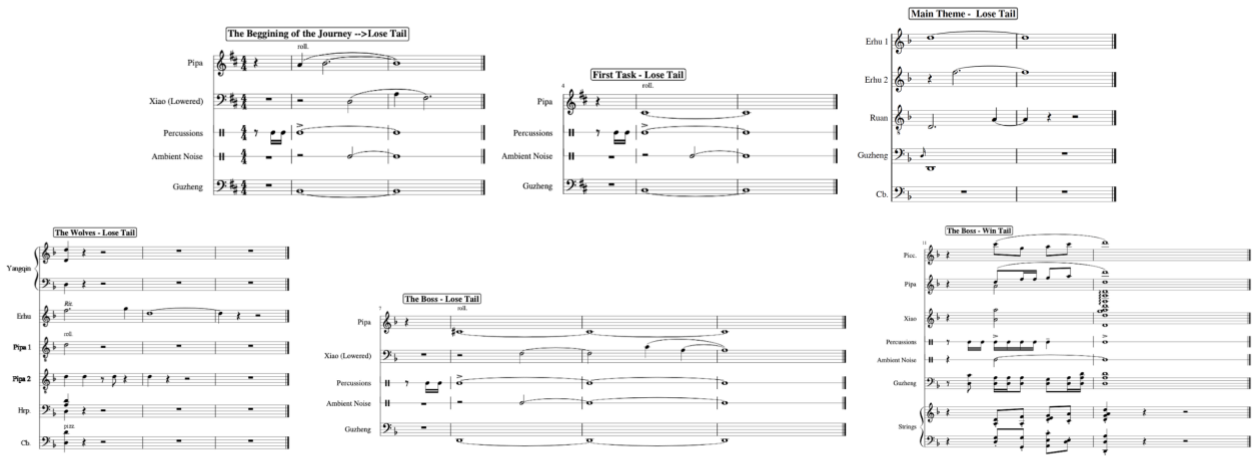


Figure 150 All tail segments except the *Before Boss* tails.

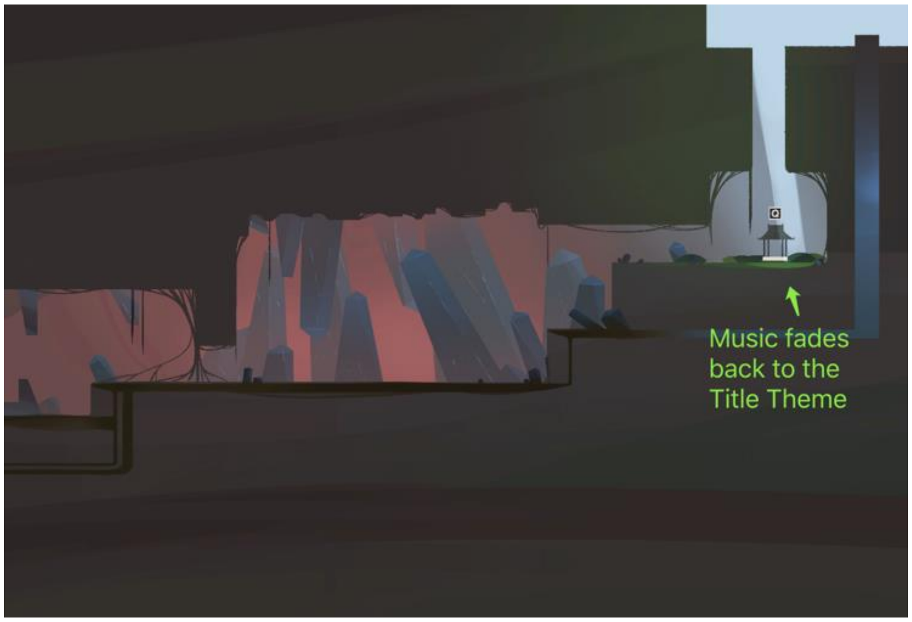


Figure 151 Music after the boss is defeated.

4.8. Summary of the Project

In this game project, I have experimented with different techniques and approaches to create ADM in each of the six musical movements. The movements have formed a sophisticated system of algorithmic composition by incorporating a variety of dynamic music procedures.

The Beginning of the Journey includes a generative counter melody whose pitch and voice-leading are interactively controlled via the use of a tonal pitch ranking system by player input as well as in-game events. The primary and non-generative melodies utilize dynamic instrumentation in response to the selection of weapon types by the player. The generative *Pipa* layer also contains dynamic articulations with either specific or random association to notes in the triggering MIDI track. The formalistic arrangement of this movement uses an algorithmic procedure to arrange the main section and the coda section. Static content (static pitch, melodic motif, and instrumentation) also helps to maintain the stylistic character of the movement.

In the *Dizi* section of *First Task*, I explored generative phrasings following the traditional phrasing patterns of sentence, period, and parallel period. This section also involves three meters that are randomly selected by the computer, four episodes of melody playing in a predetermined sequence, and five levels of diminution that respond to the player's joystick controls on every frame, all occurring in parallel. The *Sanxian* section adopts the phrasing principle of Qi-Cheng-Zhuan-He and randomly plays one of two melodic patterns. The intro section has a random length determined by the Wwise engine. The intro-*Dizi*-*Sanxian* sequence in this movement also follows a programmed procedure in Wwise.

Main Theme, *Title Theme*, and *The Boss* use the technique of dynamic mixing for the purpose of creating variations in musical arrangements. Different combinations of instrumental layers are assigned to different areas in the platform and are triggered by the player's location. The *Title Theme* adds an additional instrumental layer to the *Main Theme* for accompanying the game's title scene. Layers of instruments in *The Boss* track are interactively faded in or out to reflect the two phases of the boss enemy as well as the player's health.

The Wolves movement breaks a complete musical piece into several smaller sections and recombines them in dynamic order, while some of the sections also repeat for dynamic times. These dynamics embodied in the bigger musical sequence also loop indefinitely in order to accompany the indeterminate time the player spends in the associated game area.

The *Before Boss* movement incorporates dynamic tonal modulation in eight different keys interactively triggered by the player’s location in the platform. The eight keys modulate from low to high as the player climbs up towards the final boss area. Some of the instrumental layers are eight bars in length and loop indefinitely, while other instrumental layers form a predetermined musical sequence that is 48 bars in length. The tonal modulation can only execute after every eight bars which is close to the time a player would spend in one of the zones in the *Before Boss* area.

Finally, in all movements, there are the interactive transitional passages and tail segments that are interactively triggered by game states directly or indirectly influenced by the player’s action. The transitional passages preserve and build on the dynamic content (instrumentation, tempo, and keys) of the movements to create seamless and natural transitions. These transitional passages are triggered by associated game events (whether game-driven or player-driven), and only execute at the nearest musically meaningful moment (next measure, next beat, or a particular beat for transitions starting with a pickup).

This project’s demo videos (screen capture video with and without commentary), the full game soundtrack album, and the game’s executables (for both PC and MAC) can be found in Appendix D.

Chapter 5. Discussion: Advanced Dynamic Music (ADM)

5.1. Eight Defining Properties of ADM

Based on existing literature, there is ambiguity in describing all game music as "dynamic" and there has not been a systematic definition of a particular type of highly dynamic music system to summarize and describe game music's massive potential and possibilities. Through composing, designing, implementing and integrating the two game music composition projects, I articulate and propose a type of dynamic music called ADM with eight defining properties as discussed below.

5.1.1. Variability

The player should hear different variations of the music at each time of gameplay, regardless of music's intersection with the sound effects.

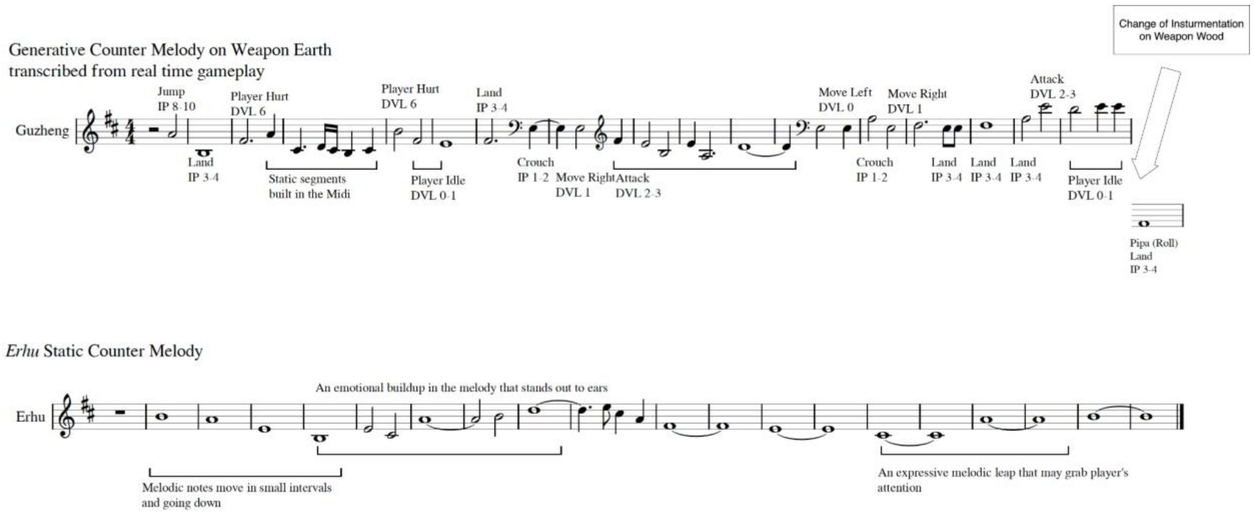
Scholars have described that "no gameplay sounds the same twice" when considering all audio elements together, including both sound effects and the music. For an ADM score, however, the game music itself should contain different variations to be played at different instances of gameplay, excluding the different combinations formed together with the sound effects.

The generative phrases played on the *Dizi* layer in the *First Task* movement in *Mastery* is a great example of how ADM can create a large number of musical variations. Several thousand musical variations are possible resulting from combining the generative phrasing types, diminution levels, and the meters. Hearing all patterns generated by only eight measures of the dynamic assets takes two hours. By modularizing and compiling a linear melody with multiple types of dynamism, quantities of variation increase exponentially.

5.1.2. User-Interaction

The player's gaming input is mandatory to make the music structure both dynamic and complete.

For instance, if a piece of game music can only move from the verse section to the chorus section when the player has performed a ten hits combo, or if the melodic line only plays dynamic pitches if the player is constantly pressing buttons on the joystick, this track fulfills the requirement of this criterion. Dynamic content that interacts with the player's input can be included in parts of a soundtrack, such as one of the instrumental layers or one horizontal section of the soundtrack.



Change of Insturmentation on Weapon Wood
Generative Counter Melody on Weapon Earth transcribed from real time gameplay
Jump IP 8-10
Player Hurt DVL 6
Player Hurt DVL 6
Land IP 3-4
Move Left DVL 0
Move Right DVL 1
Attack DVL 2-3
Guzheng
Land IP 3-4
Static segments built in the Midi
Player Idle DVL 0-1
Crouch IP 1-2
Move Right DVL 1
Attack DVL 2-3
Crouch IP 1-2
Land IP 3-4
Land IP 3-4
Land IP 3-4
Player Idle DVL 0-1
Pipa (Roll) Land IP 3-4
Erhu Static Counter Melody
An emotional buildup in the melody that stands out to ears
Erhu
Melodic notes move in small intervals and going down
An expressive melodic leap that may grab player's attention

Figure 152 Generative VS. static counter melody.

For example, I have explored this user-interaction with granular music modules at the length of single notes in *Mastery*. Figure 152 shows a transcribed score of the generative counter melody compared with the static counter melody scored in the original linear version of the movement *The Beginning of the Journey*. An audio-video demo of this comparison, which is titled Dynamic VS Static Melody, can be found in the video playlist listed in Appendix D[69]. The second staff of MIDI tracks in Figure 87 triggers the *Guzheng* generative counter melody in Figure 152, and

[69] You can also access this video at: https://youtu.be/cWTefBhsj50.

Tables 16 and 17 have explained the design of the DVL and IP values. Because the shape of the generative melody closely follows the player's direct gaming input, the music expresses the player's activities in game. In contrast, a static counter melody loops indefinitely and never changes, seeming to have no direct relationship with the interactively changing gameplay. At those moments, players may be bothered by a disconnection between the music and the gameplay (Figure 152).

5.1.3. Modularity

The music should be composed with modular structure, and the composer should create the music modules in various musical timings.

This requires that from an early stage in the creative process, game composers should be aware of their music's mobility and interactive behavior. A soundtrack should contain music modules created in more than one type of timings. It allows a variety of dynamic content to appear in the music. When module length equals the duration of one note, the compiled music played interactively in real time gameplay will sound more generative (dynamic at every note). When module length equals larger-scale musical timing, such as two measures, the compiled music in real time will sound like an ordering of several smaller musical fragments (dynamic only at the seams between modules but static content within the length of a module).

5.1.4. Game-Music Association

Music parameters should each be associated with one or more game parameters in the game code for interaction.

Music parameters come with music modules' function, size, and length. These parameters include pitch, voice-leading, phrasing, form, or meters, that are linked to game parameters such as the arrow buttons on the joystick, enemy's health, or game character's geographical location in the game world. When game parameters change their value, the associated music parameters also change their value. When a music parameter links with more than one game parameter, the dynamism of that musical parameter becomes more complex and involves more possibilities, as represented by a formula of M^G.[70]

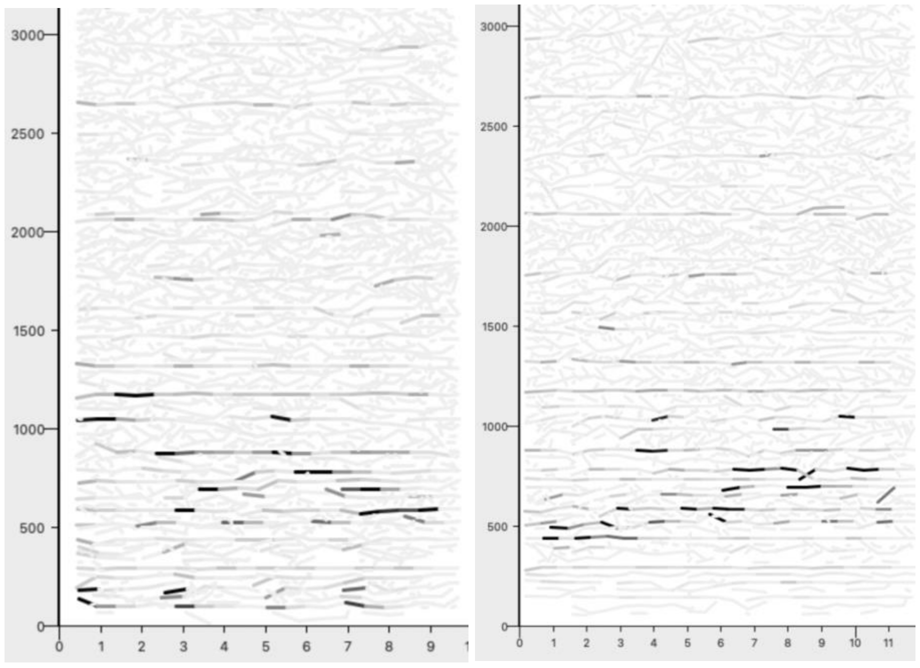


Figure 153 Spectral graphs showing Diminution Level One (left) and Diminution Level Five (right).

By associating musical parameters directly with one or more game parameters, ADM can also signal or give hints musically to the players of various in-game information. In *DOD*, for example, I created a generative rhythmic pattern constructed by algorithmically triggering one-beat percussive samples. The formed pattern can show the quantity, the type, the location, and the order of appearance of enemies being spawned on each level. In *Mastery*, the diminution levels in *First Task* can reflect the various intensities of player movement. A video titled Diminution Levels in the demo video playlist in Appendix D demonstrates the contrast between minimal player movement (Diminution Level One) and intense player movement (moving between Diminutions Levels Two to Five); Figure 153 shows these musical reflections with the spectral graphs. The Diminution Levels

[70] *M* represents music parameters and *G* represents game parameters.

Two to Five graph shows denser melodic activities happening between 500 Hz and 1200 Hz compared with the Diminution Level One graph, which uses longer notes to fill the space.[71]

5.1.5. Multidimensionality

There should be more than one type of dynamic element in one soundtrack so that the musical dynamism and game-music interaction become multidimensional.

In other words, multiple types of dynamic content should exist in one soundtrack. Different types of dynamic content may or may not occur simultaneously or carry the same timing in gameplay. For instance, the soundtrack can change its instrumentation interactively controlled by the player's choice of weapon, while the movement of the game character decides the specific pitch of the melodic layer of the track. The instrumentation changes and the pitch changes in the melody may or may not occur at the exact same time in gameplay.

In *DOD*, musical parameters have association with multiple game parameters including player location, enemy types, pickup buff types, level progression, and player class-specific abilities. In *Mastery*, musical parameters have association with player movement, choice of weapon, location, types of objects that a player can attack, percentages of *Chi* and health, and the progress in gameplay (for instance, after the wolves die, *The Wolves* movement is never heard again even if the player comes back to the wolves' area).

5.1.6. Multiplicity

At least one music behavior representing the agency of the computer improvisation should appear in the music for more musical options and indeterminacy, such as selecting elements using randomness or probabilities, regardless of whether this musical multiplicity is applied to the entire soundtrack or part(s) of the soundtrack.

[71] Please ignore the *Dagu* patterns below 200 Hz area in the Diminution Level One graph. *Dagu* plays on every alternate episode and is a different type of dynamic content in this movement.

Two primary devices to generate indeterminate selection from multiple possible options are randomness and weighted probabilities. These devices represent the computer's autonomous behavior which can trigger multiple musical options within one type of music dynamism. This multiplicity to dynamism is different from interactive content generated by the player's direct input, even though all game audio is inevitably connected to the player's indirect input. Various types of musical content can use randomness and probabilities as the source of stochasticity, such as the ordering of the sections of a soundtrack (installed on the entire soundtrack) or the chances of switching between three rhythmic patterns played on an instrument (installed on part of the soundtrack).

However, gameplay is not always under the player's control. Games involve tasks that players have no other choices but to follow and achieve, which also results in repetitive or non-variable player input patterns. In those scenarios, the computer-generated variety comes in and adds additional musical multiplicity to reduce possible musical monotony generated by recurring player input patterns.

5.1.7. Staticity

Regardless of what types of generative procedures an ADM track has, the resulting music generated by the interactive music system should be able to preserve and remain in a predetermined musical style, theme, and emotional expression through static musical content as in a linear version of the soundtrack.

Z. Whalen (2007) has expressed his concerns and reactions to a presentation given by Robert Bowen at the 2004 Princeton Video Game Conference in his article:

> In fact, Bowen has argued that the sequential production of sounds during gameplay can be considered a kind of aleatory composition such that playing a game generates a musical product. This is a fascinating argument, but the problem I see with the result is that music need not have any contact with the player… In other words, taking literally the implication of applying narrative structure to

> video-game music, it closes off the gameness of the game by making an arbitrary determination of its expressive content. (pp. 73-74)

I believe Bowen's notion of "playing a game generates a musical product" resonates with the principle of ADM. At the same time, Whalen is concerned that the "aleatory" feature would stand against or at least eclipse the more analysis-interesting "narrative" and "expressive" content.

In my opinion and my definition of ADM, the two sides do not contradict or conflict with each other. The action of playing games influencing the game music may seem aleatoric to a certain degree, but the music is never completely random. The inclusion of indeterminacy does not imply pure chance. The inclusion of aleatoric elements in some music parameters does not necessarily interfere with the music's overall narrative or emotional expression, given that static materials in other music parameters (instrumentation, key, tempo, rhythm, dynamics, meter) can help with maintaining those expressions. To give three examples: if the background music changes from the *village* to the *forest* soundtrack triggered by a player's motion occurring at an indeterminate time, this timing of the trigger does not affect the emotional expressiveness conveyed by the harmonic progressions or melodies in either soundtrack. Another example is when the game data randomly chooses a type of instrument to play the primary melody of the soundtrack, the aleatoric instrumentation does not change the musical expression of that melody. Lastly, if the primary melody involves real time pitch generation that is indeterminate in real time, a static underlying accompaniment as well as static materials played on other instruments can still convey a determinate musical vibe.

Thus, an ADM soundtrack should be able to preserve the same stylistic consistency, musical effect, and emotional expression as if it is a linear, static, and non-interactive soundtrack. In other words, among the many music parameters or parts that could incorporate dynamic content, one or more parameters or parts should remain playing static content so that the essence of a piece of music's identification can remain.[72]

5.1.8. Coherency

There should be independent condition-driven algorithmic procedures programmed for music's own purpose aside from those for game-music interaction purposes.

The code inside a game project has different purposes and functions: some functions are specifically written for gameplay (such as algorithms that take care of the leveling up of the game character); some functions are written specifically for running the animations; other functions connect the animation with game parameters. This criterion of the ADM specifies that the music should have its own algorithmic procedures either coded as "music-specific" algorithms inside the game engine (e.g., Figures 51 and 94) or programmed as algorithmic procedures in a third-party audio tool such as an audio middleware[73] (Figure 136). The algorithmic procedures may vary from simple to complex, such as generating pitches or musical sequences in a first order Markov chain, or generating a parallel melodic line that is a minor third above or below an existing melody.

This defining criterion also specifies that the music system's own algorithmic procedures has independence from interaction with game parameters. For example, when a player moves from the main city to a desert in the game world the theme music needs to switch as well. If the game engine immediately fades the Main City soundtrack and switches to the desert soundtrack audio file, it does not fulfill this criterion of ADM. However, if the music system stops the current track after a complete musical measure, and plays a short transitional passage between the two

[72] In an ADM work, the composer should carefully decide upon the proportion of static and dynamic content, in which the static part can help to immerse the player, bringing continuity, and ambiently maintaining the game flow, while the dynamic part follows the real time game events, creates musical variations, or expresses the player's individuality.
[73] Audio middleware is a software that usually sits between the game engine and the audio hardware which allows sound designers and composers to create and design audio behavior and interaction with games where minimal coding background is required.

soundtracks, it fulfills this criterion of ADM. This is because the music system itself involves an algorithmic procedure for musical purposes instead of instantly reacting to the game parameter.

Having independent algorithmic procedures internally in the music engine means also that music persists to its own timing of change, whether it happens to occur at the exact same time with a game parameter change or not. However, the gaps between the game timing and the music timing should be as little as possible.

5.2. Advantages and Potentials of ADM

Early video games were developed without large amounts of user-research or players' review to guide their production. One of the inherent charms of video games is the opportunity given to the game makers to become the author of creative content like in any other creative art form. The game makers have the ultimate responsibility and ownership of how they need the game to be designed and developed. This still continues in today's video games. Player experience is at the center of game production. A fun game experience does not only come from fulfilling the players' desires but also from surprising them.

Additionally, player's feedback comes only *after* a presentable demo or version of the game is produced. This is crucial because it implies that player feedback can help in improving but not designing the game. Like in any industry, video game consumers are not responsible for providing solutions from either a technical or creative perspective. Their feedback is highly important and encouraged, but further investigations and discussion analyzing these feedbacks are necessary. Compared with marketing which has become one of the most crucial parts of a successful commercial game, the chance of transforming player's feedback into more sales or better products is still inevitably unpredictable and requires luck. There are now professional scholars and specific teams in game companies who research user-experiences - the human-computer interaction in video games. These specialists have provided valuable analyses and insights into the reception of video game technologies and creative design by players from a professional standpoint and are becoming the primary force for game designs. However, since video games involve a great deal of collaborative work, the success of a title cannot be assured or guaranteed. This phenomenon in game production is one of the biggest challenges in creating best-selling games.

I believe that as in the case of game production more generally, combining new technological capabilities as well as creative design is the key to improve game music. There is a traceable and popular product development strategy for video games which combines technological innovations and novel game design in the service of fostering major revolutions and remarkable milestones in the history of game making. For instance, new technological capabilities enabled hosting dozens to hundreds of players playing simultaneously in the same battle in massive multi-player online games. Such capabilities contribute to the success of many big MMO titles such as *World of Warcraft* and the *Final Fantasy series*. The innovative game design and mechanics[74] of the battle arena game *League of Legend* or the battle royale game *PlayerUnknown's Battlegrounds* have also marked significant milestones in the revolution of game design.

Multidisciplinarity is another key concept to improve game music in my opinion, because it parallels and resonates with game's multidisciplinarity (the medium-specific characteristic). Game music as a component of game should inevitably follow along the development strategy of its parent medium. By incorporating and integrating ideas and cutting-edge research from the various disciplines within the realm of music (composition, music theory, improvisation, music technology, songwriting, etc.), dynamic game music can continuously find nourishment and room for improvement. The ADM framework builds upon this concept. ADM brings the technological innovations of algorithmic compositions in contemporary classical music practices to its unexplored application in video games. ADM also adopts traditional musical procedures in composing counterpoint, polyphony, orchestration, and methods of musical theoretical analysis to modularize and dismember music into smaller units and different parameters in various ways to form a highly dynamic, interactive, and non-linear structure in video games. When more game companies, scholars, and composers notice ADM's value, I envision that a new branch of human-computer interaction studies will focus on the design of dynamic music systems in games. I also envision game companies will also demand experts in dynamic music systems as irreplaceable members of the game audio department in the near future.

[74] Mechanics is one of the three components of a video game in the MDA framework suggested by Hunicke et. al., the other two elements are: dynamics and aesthetics (2004).

5.2.1. ADM as a New Game Experience

An ADM game score has the potential to bring new game experiences to the players. Scholars believe that the player has the role of co-creator of video games (Wirman, 2009). Some argue that video games are incomplete 'until the work gets a player' (Dahlen, n.d.). While game music is a central element of video games (Velardo, 2018, p. 3) which can be as interactive as the gameplay, it also needs the player to bring the music to "life" in real time. In other words, without the involvement of a player, game music is also incomplete. An ADM score can bring new gaming experiences that maximize the involvement of the player and the player's influence on the music system. Players hear game music closely following and interactively responding to their input data so as they are improvising the game score. Studies have also shown that players are aware of dynamic music and believe that it is important to enhance the interactive game experience (p. 3).

This new experience of gaming as music improvisation has not been explored widely in existing games. In many RPG games, players have the opportunity to explore and develop their own version of the game story based on their choices and decisions in different game tasks. Players also have agency in controlling the behavior, movement, and growth of their preferred game characters (Magerko, 2005). Recently, other types of game content such as urban modelling, level design, and dynamic dialogue with NPCs (Non-player character) has become procedurally generative based on analyzing user data (Blackwell, 2019; Watson et al., 2008; Youxiputao, 2020). These examples show a rising trend and the growing popularity of highly individualized and interactive content in games where each player can have a unique game experience.

5.2.2. ADM as an Expression

Player's interactions with video games are self-reflective, and are therefore expressive (Hart, 2014). By incorporating player-improvised game music into genres of single player, multi-player, or even massive multiplayer games, ADM game music could provide players the opportunity to express their personality musically through gaming. ADM allows players to exhibit their gaming techniques and to have a new form of creativity. For instance, a performance of an ADM score can serve as great content for live streaming and gameplay screen-capture videos. This is because, firstly, it is creative, meaning that each player can play and perform their unique version of the game music. Secondly, depending on the design of the interactivity between the game engine and the music engine, a great performance in musical terms can be challenging in gaming terms. Thus, it makes music improvisation through gaming virtuosic. Lastly, since improvising a game score as content for a live stream can be an attractive topic, it has the potential to bring different audience groups. These groups can include frequent game live-stream viewers or music lovers who never thought that music could be performed or "improvised" by playing games.

5.2.3. ADM as a Framework for Nonlinear Music Structure

From the compositional and technical perspective, ADM music carries many advantages which fit the game medium well, such as the capability of generating large amounts of dynamic musical content with only a small number of audio assets to accompany the indeterminate length of gameplay. This strength of ADM can also save storage space in a way that allows the game to run with better performance and smaller downloading size (often useful for meeting the needs of a game title with limited budget).

Screen scoring music[75] and algorithmic composition are not new. However, combining the two for the interactive experience of video games has not yet been explored in its full potential. For the benefit of composers, my hope in this paper on ADM is that it serves as a compositional framework for transforming music pieces composed with linear structures into a nonlinear, algorithmically procedural and dynamic structure. Through practical processes, composers get to

[75] Screen scoring music here refers to music composed for accompanying moving images, such as films, animation, and games.

know the capacity of ADM in the medium of video games to create highly dynamic and interactive musical content without losing the essence of musical styles, effects, and characteristics that can be achieved in the linear realm. The process of creating a music system embodying such transformation can be simply summarized as - deciding what in the music that should be static and what should be dynamic.

Although players' interaction in ADM plays a vital role in the *expression* of game music, the composer also holds ownership of the musical expression and meanings as the creator of the interactive system. As Lewis described, musical computer programs, like any texts, are not "objective" or "universal," but instead represent the particular ideas of their creators (Lewis, 2000).

Chapter 6. Conclusions and Future Research

6.1. Limitations and Future Work

There are some technical and aesthetical limitations in this work and these limitations suggest directions for future research. These limitations and future research areas fall into several specific topics or aspects, including: audio technology, musical arrangement, game development and design, and artificial intelligence. Audio technology involves new techniques used for integrating game audio or allowing algorithmically manipulated, game-driven sound processing. Musical arrangement refers specifically to the types of procedures and devices involved in the compositional process such as arranging generative content in polyphonic structure. Further research in theories of game development and design should be involved in the improvement of the ADM system as well. Lastly, the potential of artificial intelligence in game music needs to be evaluated and experimented with further.

In the aspect of audio technology, one of the limitations of this dissertation is that the utilization of generative MIDI data is currently missing. As briefly discussed in the *Mastery* project, the current algorithmic procedures only use pre-produced, fixed MIDI files. Including the function of generative MIDI data could further expand the possibilities of musical dynamism in numerous ways. For instance, audio samples in a couple of soundtracks in this paper are represented and triggered by MIDI notes. Generative MIDI data can be used for various purposes in ADM and can provide another dimension for creating dynamic content.[76]

Another limitation related to audio technology is a dynamic 5.1 or 7.1 surround sound design of the music system which could potentially bring unexpected effects to the listening

[76] I have found several audio tools that can be used for bringing generative MIDI to ADM such as Audio Helm by Matt Tytel, Midi Tool Kit developed by Thierry Bachmann, Complete Sound Suite produced by 3y3net, and Stepp produced by Pelican 7, all available in the Unity asset store.

experience in game. Currently, the two creative projects involve only stereo sound spatialization as included in the two projects. The acoustic outcome of the dynamic spatialization can be less natural and abruptive to the player's ears. Creating a dynamically moving score in a surround system has the advantage of merging all sound into one acoustic space. Thus, no matter how sound assets move in the acoustic space in real time, a sense of entirety can be preserved throughout.

In terms of musical arrangement, one of the limitations of this dissertation project is that the generative musical content remains in monophonic structure. Whether it is the generative counter melody in *The Beginning of the Journey*, or the generative phrasings in *First Task*, in *Mastery*, generating real time indeterminate musical content was only applied to one of the instrumental layers. Managing generative content in multiple layers of music can be very challenging because the composer needs to not only anticipate and control all possible combinations of voice-leading within one melodic line, but also the intervallic relationship, or the polyphonic structure that occurs between multiple melodic lines. Manipulating this polyphonic structure takes the composer's ability to make the outcome of the generation harmonious and musically elegant. The possibilities embodied in generative polyphony involves an incredible amount of indeterminacy. It might take thousands of iterations to be able to know every single possible combination. To give an example, if the primary melody in *The Beginning of the Journey* is also a generative line, thus having a two-part generative melody, the composers need to facilitate how they want the distances between notes from each line to be programmed. The composers might not want the two lines to ever clash with each other, while at the same time the intervallic relationship should be musically meaningful and desirable to listen to. The composer also needs to carefully evaluate at what time the two lines should move in a contrary motion, oblique motion, or parallel motion, then implement their plan in actual algorithms. Two-part generative lines are already very challenging, notwithstanding those instances when the composer needs to design a four-part or five-part generative structure. When generative polyphony can be controlled and manipulated easily, generative harmonic progressions can also be easily achieved.

From a compositional perspective, future research is needed to explore other ways of manipulating these musical parameters that are different from the ways suggested in this paper. Additionally, there is a need to explore other types of musical parameters that are manipulatable for ADM that were not discussed in this paper. I have experimented with musical parameters such as tempo, meter, pitch, timbre, voice-leading, phrasing, form, instrumentation, rhythm, articulation, and tonality. These parameters could be used in other ways to create an ADM score while still other parameters should be included in the ADM system because of their potential for outputting dynamic content, such as scale, ternary form, blues, sound synthesis, among others. I have explored generative rhythmic patterns in the project *DOD* where I have programmed the probability of using an eighth-note or a 16th-note. However, generating more sophisticated rhythmic patterns, such as those involving dotted rhythms, more note values, or polyrhythms, deserves further experimentation and application.

In ADM, designing musical timing involves many challenges. From the game development and design perspective, having video games developed in musical timing could be an exciting and innovative area to explore. There are existing examples of games that are developed and executed in musical timing, widely known as *rhythm games* (different from *music games*). If we consider how video games move and are executed in frames per second, which creates an illusion of continuous gameplay or the moving game images, game's timing can also be programmed and quantized in musical time units, such as on every 64th-note[77]. By quantizing frames to smaller and smaller musical time, a rhythm game can still provide smooth and continuous visuals while perfectly in sync with musical time units. This feature would revolutionize ADM in video games in my opinion, because there is no longer the conflict between game timing and musical timing that game

[77] Or even smaller musical time units (such as ticks of a 64th note).

composers have to solve. Quantizing video game's internal clock in extremely small musical time units provides the opportunity to generate sophisticated rhythmic patterns in the music. Additionally, with ADM in such games, different styles of rhythm can also be included in the musical performance, whether it is a bluesy swing or Baroque rubato. As a result, this time quantization will truly make games *musical*, as discussed by Kanaga (2012).

Since the goal of ADM is to explore more advanced applications of mobile form composition in games, this evolution may likely end in the realm of having everything as generative content utilizing the power of artificial intelligence. The rising significance of machine learning applied to music composition is exciting, especially given recent trends in models such as recurrent neural network (RNN), Variational Autoencoders (VAEs), and Generative Pre-trained Transformer 2 (GPT2). Besides the Markov process, the current use of types of machine learning algorithms in music generation will need evaluation by professional composers and their application to ADM deserves further research and exploration.

In the future, limitations must be drawn and research conducted into areas related to game design and the human-computer interaction studies in the direction of game music. Specifically, game music composition should be further investigated in depth in terms of particular game genres (puzzle, adventure, survival, etc.), taxonomies of game states (game-driven, player-driven, hybrid, etc.), music cognition in a game context, and the reception of game music specific to the "stereotypes" of players (killers, socializers, explorers, etc.) (Bizzocchi & Tanenbaum, 2011). For example, players may hear dynamic music material in single-player music resulting from player's control. However, players may also hear dynamic music material in two-player games or multi-massive players online games due to their different game goals (that two are fighting against each other) or specific classes and roles in a massive game battle (such as the tank, DPS, or heal heroes).

While working on this dissertation project, I had envisioned an evolution of algorithmic music which I believe will become a new avenue for music creativity. As shown in Figure 154, I have given a hypothesis of how dynamic music will advance in the future. While there are still games incorporated with musical dynamism that consists of only switching between a list of soundtracks, this project has already explored the move to generating transitional passages and generative melody. The immediate next stage will be generative polyphony which takes care of multiple melodies at the same time as well as a harmonic structure. Since these music generation processes are achieved through algorithms, the ultimate stage of ADM may involve generative algorithms. For instance, if each type of musical generation eventually has a universal algorithm to generate, whether it is an algorithm specifically for generating Bill Evan's type of harmonies, or an algorithm specifically for generating traditional Persian music, all these algorithms will be available to the game engine. The game engine, or the computer, can then create different combinations of these algorithms to generate new styles of music. Certainly, this process can be associated with various in-game states and player's input. I hope this hypothesis can serve as an inspirational device for future game composers, sound designers, game developers, as well as scientific researchers in this field and as a guiding direction to achieve the maximum potential of computer music.

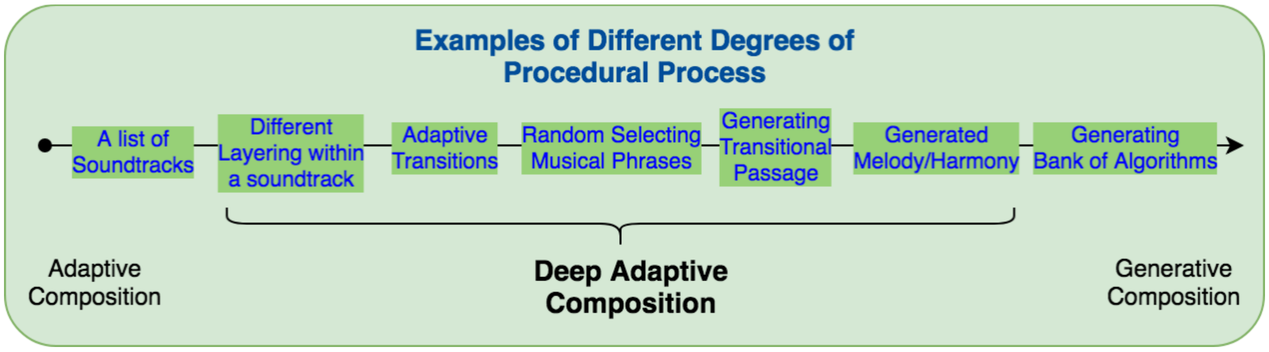

Examples of Different Degrees of Procedural Process
A list of Soundtracks
Different Layering within a soundtrack
Adaptive Transitions
Random Selecting Musical Phrases
Generating Transitional Passage
Generated Melody/Harmony
Generating Bank of Algorithms
Adaptive Composition
Deep Adaptive Composition
Generative Composition

Figure 154 An assumption of the evolution and future of ADM.

6.2. Following the Two Creative Projects and ADM Definition

Composers can apply ADM as a solution to elevate a linear score's suitability to the dynamic needs of the game medium. As introduced in 2.4. Towards ADM, there are existing game examples that have incorporated some highly dynamic and interactive musical procedures in their music systems. However, these existing game music examples do not always have all eight criteria presented.

As the initial experimentation of ADM, the two ADM projects in this dissertation have closely followed the ADM definition. The projects demonstrate the eight theoretical defining properties of ADM in real examples. As the composer, music designer, programmer, and integrator, I think the two projects have achieved the goals of solving specific game music problems through the eight ADM criteria. Though some of the musical effects or outcomes might sound subtle or insignificant to the general audience, future composers may apply ADM as a theoretical structure and solution to their own compositions in their unique artistic ways.

From working on the two ADM projects, I later noticed several issues or potential shortages of the ADM theory and the two projects, which are worth improvement or further investigation in the future. The first problem is whether or not players can clearly perceive these musical changes. Another related question is, how are players aware that they are actively influencing the music by interacting or "improvising" with the computer algorithms behind the game engine. During the compositional and designing process, it is also interesting to consider how ADM can proactively inform the players besides responding to player input data, in which case a true "bi-directional" interaction between an algorithmic music system and the player (Dobrian, 2004). In future research, auditory saliency is an informative area to investigate. This research area may aid in the design of specific types of dynamism (such as at what time in games dynamic content, static content, abrupt content, or smooth content are more preferable), as well as provide better promotion and representation of ADM. Since I was only summarizing the eight characteristics from my own compositional experience in the two creative projects, I am also aware of the possibility that other characteristics of dynamic music systems not summarized in the eight defining properties of ADM may carry similar or even better contributions to current game music practices. These other possibilities embodied in game music will also be one of my future research areas.

In the two creative projects, I have made aesthetic assumptions that smoothness, continuity, and musical coherency are more preferable musical characteristics in many scenarios, especially when dynamic music structure is compiled with music modules of small or granular size. These aesthetic principles help the compilation of these tiny music modules (e.g., modules of single notes) to sound more logical and more like “music”. However, these principles may be less important for large music modules, because sounding more like “music” may not be the primary goal.

Sometimes composers would want the players to hear disjunctions happening in the music, in which the musical changes are more sudden without smooth transitions. These sudden changes in music can show dramatic events occurred in the games and thus the aesthetics of disjunction will be more preferred. This aesthetic preference of disjunction or more dramatic, sudden musical changes also relate to the “mickey-mousing” effect used in film music and cartoon music. Sometimes composers might intend to insert a gap or pause between two music segments or modules. The “on and off” effect happening in the underlying soundtrack can help express a variety of in-game scenarios as well, such as after a huge battle, the players return back to calmer gaming activities where they just submit missions or check inventories. The emptiness between music modules may sound more dramatic than switching from a rhythmic section to a melodic section in a continuous piece of music. When these sudden and more dramatic musical changes occur, there might not even be enough time, or the necessity, to finish a complete musical measure or adding a transitional passage. Because the more important goal is to show a dramatic change, maintaining musical coherency during this musical change is not necessarily the priority.

6.3. Final Thoughts

This dissertation as a whole contributes to current game music studies with two things. First, I explored a variety of approaches and ways to create dynamic music structure documented in the two creative projects. Second, I defined a specific type of dynamic game music as ADM with the eight specific criteria.

In the first half of this dissertation, I introduced my own compositional techniques and approaches in the two creative projects, which I hope can provide specific examples and inspirations to other composers in composing ADM. These techniques and approaches include but are not limited to: 1) generating melodies with predetermined pitch collections following procedurally changing harmonies; 2) generating dynamic phrasing variations by combining melodic segments in different orders while having other types of dynamism occurring such as diminutions, meters, and episodic melodic shape intertwined; and 3) generating rhythmic pattern with one-beat samples in grids of dynamically changing note values. Although every piece of video game is unique and so is every piece of music, I hope these examples in my projects can provide inspiration for composers on the types of musical dynamism and types of game-music associations they can develop in their works.

In the second half of this dissertation, I summarized then discussed the eight defining properties of ADM. ADM is a groundbreaking establishment that helps to clear the confusion created by describing all game music as "dynamic" and differentiates highly dynamic music systems from others. Game music scholars can use ADM as a methodology for analyzing game music or a taxonomy for determining dynamic music procedures in games. Composers can take ADM as a theoretical guide, compositional framework or solution for composing dynamic game music. For each characteristic, I have tangible musical examples in my two creative projects. Based on methods of theoretical analysis in classical music and algorithmic compositions, as well as my own experience in working on the two ADM projects, each of the eight defining properties of ADM can solve a specific game music problem.

The purposes of the eight characteristics of ADM can be summarized as (in the order of the eight ADM criteria): 1) more music variations using modular structures – ADM helps to reduce monotony embodied in looped music; 2) more interaction – ADM incorporates player input and makes dynamic game music sound more lively, individualized and personalized. Additionally, this close musical responsiveness can also reduce the disconnection between the images and music; 3) better continuity - game music has seams (Medina-Grady, 2014). The staggering effect created by compiling musical modules composed in various musical timings helps to cover the musical seams, which increases the musical smoothness and continuity (less noticeable disjunction in music, especially when using more granular-sized modules); 4) meaningful expressions - associating musical parameters with game parameters gives musical expressions "meanings" to the game narratives or progress; 5) more realistic music - Realistic music performance does not change a single musical parameter (pitch, dynamics, figurations) at a time, but multiple intertwined. When having multiple music-game associations or one musical parameter associated with multiple game parameters, the music becomes more realistic and closer to live music. For games with high-fidelity graphics, more live-sounding dynamic music should be included; 6) more musical options – computer-generated aleatory (randomness and probabilities) brings additional musical variations to reduce monotonous material generated by players' recurring and non-changing input patterns (caused by their gaming habits or the game's configurative tasks), other than the monotony created by looped structure; 7) stylistic identification and consistency - having static content helps to identify a piece of ADM in specific musical styles, genre, or emotional effect. Even if the dynamic content of ADM is granular-sized and reacting to player input, the static content ambiently show the overall or higher-level stylistic characters and emotional effect of a piece of music; 8) better musical coherency - having specific and independent musical procedures in the game algorithms elevate the musical experience in games in the way that the transitions between two music modules can be musically connected. This helps maintain the game music's constant rhythmic pulsing to not be abruptly interrupted by immediately switching from one module (of any length) to another on crossfades or cutoffs.

I hope the ADM definition as well as my two ADM projects with their detailed documentation can inform insights and shortcuts for classically trained composers to enter the game music world. I hope composers can be introduced with the beginning-to-end process of game music integration and the types of tasks and skills involved in this process. And I hope composers can get a comprehensive picture about the potential of game music and the field's possible direction and capacity for musical innovations in the future.

BIBLIOGRAPHY

Åberg, E. (2017). *Game music: From composer to consumer*. 38.

acajc3. (2016, October 3). Playing with the Past in the Imagined Middle Ages: Music and Soundscape in Video Game. *Sounding Out!* https://soundstudiesblog.com/2016/10/03/playing-with-the-past-in-the-imagined-middle-ages-music-and-soundscape-in-video-game/

Adorno, T. W. (2007). *Composing for the films*. Continuum.

Ames, C. (1989). The Markov Process as a Compositional Model: A Survey and Tutorial. *Leonardo, 22*(2), 175–187. JSTOR. https://doi.org/10.2307/1575226

Aristopoulos, M. (2011a). Richard Stevens and Dave Raybould: The Game Audio Tutorial: A Practical Guide to Sound and Music for Interactive Games. *Music, Sound, and the Moving Image*, *5*(2), 197.

Aristopoulos, M. (2011b). The Game Audio Tutorial: A Practical Guide to Sound and Music for Interactive Games (review). *Music, Sound, and the Moving Image*, *5*(2), 197–202.

Aska, A. (2017). *Introduction to the Study of Video Game Music*. Lulu.com.

Audio Helm—Native Synthesizer, Sequencer, Sampler [RELEASED]. (2017). Unity Forum. https://forum.unity.com/threads/audio-helm-native-synthesizer-sequencer-sampler-released.501808/

Audiokinetic. (2010). *Wwise Tutorial 13—Creating Stingers*. https://www.audiokinetic.com/learn/videos/fLrz463kVEI/

Belinkie, M. (1999). *Video game music: Not just kid stuff*. https://www.vgmusic.com/information/vgpaper.html

Berndt, A., & Hartmann, K. (2008). The Functions of Music in Interactive Media. In U. Spierling & N. Szilas (Eds.), *Interactive Storytelling* (pp. 126–131). Springer Berlin Heidelberg.

Betts, W. (2019, April 24). Algorithms, apes and improv: The new world of reactive game music. *MusicTech*. https://www.musictech.net/features/interviews/ape-out-matt-boch-game-soundtrack/

Bizzocchi, J., & Tanenbaum, T. J. (2011). *Well Read: Applying Close Reading Techniques to Gameplay Experiences*. 16.

Black Mirror: Bandersnatch. (2019). In *Wikipedia*. https://en.wikipedia.org/w/index.php?title=Black_Mirror:_Bandersnatch&oldid=925538308

Blackwell, C. (2019). *Procedural level generation for games* (United States Patent No. US10272341B1). https://patents.google.com/patent/US10272341B1/en

Boyd, A. (2003). *When Worlds Collide: Sound And Music In Film And Games*. https://www.gamasutra.com/view/feature/131310/when_worlds_collide_sound_and_.php

Bramwell-Dicks, A., Petrie, H., Edwards, A. D. N., & Power, C. (2013). Affective Musical Interaction: Influencing Users' Behaviour and Experiences with Music. In S. Holland, K. Wilkie, P. Mulholland, & A. Seago (Eds.), *Music and Human-Computer Interaction* (pp. 67–83). Springer London. https://doi.org/10.1007/978-1-4471-2990-5_4

Burr, A. (2014). Mirroring Reality: Music in Video Games. *Student Research Day Abstracts and Posters*. https://digitalcommons.chapman.edu/cusrd_abstracts/9

Casella, P., & Paiva, A. (2001). MAgentA: An Architecture for Real Time Automatic Composition of Background Music. In A. de Antonio, R. Aylett, & D. Ballin (Eds.), *Intelligent Virtual Agents* (pp. 224–232). Springer Berlin Heidelberg.

celldweller. (2016). *Killer Instinct Season 3—Creating The Music for "Eyedol."* https://www.youtube.com/watch?v=XVh1LYGLzWU

Cerrati, M. (2005). Video Game Music: Where It Came from, How It Is Being Used Today, and Where It Is Heading Tomorrow. *Vanderbilt Journal of Entertainment and Technology Law, 8*, 293–334.

Chan, H., & Ventura, D. A. (2008). Automatic Composition of Themed Mood Pieces. *Proceedings of the 5th International Joint Workshop on Computational Creativity*, 109–115.

Chen, X. (2019). *《Sky 光遇》这款游戏有哪些不容易察觉的细节？*. 知乎. https://www.zhihu.com/question/325437663/answer/725714428?from=timeline&isappinstalled=0&s

_r=0&s_s_i=NmQwqvevxkEPNwmgF82YiqPStvepDcrmYHgiYNbXqMs%3D&utm_medium=social&utm_oi=27449421201408&utm_source=wechat_session

Childs IV, G. W. (2007). *Creating Music and Sound for Games*. Cengage Learning.

Chiricota, Y., & Gilbert, J.-M. (2007). IMTool: An Open Framework for Interactive Music Composition. *Proceedings of the 2007 Conference on Future Play*, 181–188. https://doi.org/10.1145/1328202.1328235

Clark, A. (2007). *Defining Adaptive Music*. http://www.gamasutra.com/view/feature/129990/defining_adaptive_music.php

Clinton, W. (2019). *Music improvisation in Python using a Markov Chain Algorithm*. 42.

Cole, M. S. (1974). Mozart Rondo Finales with Changes of Meter and Tempo. *Studia Musicologica Academiae Scientiarum Hungaricae*, *16*(1/4), 25–53. JSTOR. https://doi.org/10.2307/901841

Collins, K. (2005). From Bits to Hits *Video Games Music Changes its Tune*. *Film International*, *3*(1), 4–19. https://doi.org/10.1386/fiin.3.1.4

Collins, K. (2007). In the Loop: Creativity and Constraint in 8-bit Video Game Audio. *Twentieth-Century Music*, *4*(02), 209–227. https://doi.org/10.1017/S1478572208000510

Collins, K. (2008). *Game Sound: An Introduction to the History, Theory, and Practice of Video Game Music and Sound Design*. MIT Press.

Collins, K. (2009). An Introduction to Procedural Music in Video Games. *Contemporary Music Review*, *28*(1), 5–15. https://doi.org/10.1080/07494460802663983

Collins, K. (2013). *Playing with sound: A theory of interacting with sound and music in video games*. https://uci.worldcat.org/title/playing-with-sound-a-theory-of-interacting-with-sound-and-music-in-video-games/oclc/824734256&referer=brief_results

Collins, K., Cooper, D. B., Summers, C., Fuller, B., Thiel, D., Paul, L. J., & Storming the Base (Firm). (2016). *Beep: A documentary history of game sound*. Ehtonal.

Collins, K., Kapralos, B., & Tessler, H. (2014). *The Oxford handbook of interactive audio*. Oxford University Press.

Collins, N. (2008). The Analysis of Generative Music Programs. *Organised Sound*, *13*(3), 237–248. https://doi.org/10.1017/S1355771808000332

Collins, T., Laney, R., Willis, A., & Garthwaite, P. H. (2011). Chopin, mazurkas and Markov. *Significance*, *8*(4), 154–159. https://doi.org/10.1111/j.1740-9713.2011.00519.x

Comair, C., Johnston, R., Schwedler, L., & Phillipsen, J. (2004). *Method and apparatus for interactive real time music composition* (United States Patent No. US6822153B2). https://patents.google.com/patent/US6822153B2/en

Crathorne, P. J. (2010). *Video game genres and their music* [Thesis, Stellenbosch : University of Stellenbosch]. http://scholar.sun.ac.za/handle/10019.1/4355

Cycling ’74. (2014). *M: The Classic Intelligent Composing and Performing System*. https://cycling74.com/products/m

Dabby, D. S. (1996). Musical variations from a chaotic mapping. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, *6*(2), 95–107. https://doi.org/10.1063/1.166171

Dahlen. (n.d.). *From the Editor*.

Davies, H., & Saxton, R. (2015). *Towards a more versatile dynamic-music for video games: Approaches to compositional considerations and techniques for continuous music*. https://ora.ox.ac.uk/objects/uuid:3f1e4cfa-4a36-44d8-9f4b-4c623ce6b045

DeCastro, D. (2007). *Quality Video Game Music Scores, Considering the Standards Set, and Personal Reflections*. https://www.vgmusic.com/information/vgpaper3.html

Demystifying Video Game Audio Middleware | Somatone Interactive, Inc. (2015, August 17). *Somatone Interactive*. https://www.somatone.com/demystifying-audio-middleware/

Dobrian, C. (2004). *STRATEGIES FOR CONTINUOUS PITCH AND AMPLITUDE TRACKING IN REALTIME INTERACTIVE IMPROVISATION SOFTWARE*.

Donnelly, K. J., Gibbons, William, & Lerner, N. (2014). *Music In Video Games | Studying Play*. Taylor & Francis. https://www.taylorfrancis.com/books/e/9781134692040

Douglas, A. (2002). *Sound of Music: The Form, Function, and History In Video Games*. https://web.stanford.edu/group/htgg/cgi-bin/drupal/sites/default/files2/adouglas_2002_1.pdf

Driscoll, K., & Diaz, J. (2009). Endless loop: A brief history of chiptunes. *Transformative Works and Cultures*, *2*. https://doi.org/10.3983/twc.2009.096

Dunkel, M., Petermann, E., & Sauerwald, B. (2013). *Time and Space in Words and Music*. Peter Lang. https://doi.org/10.3726/978-3-653-02252-0

Eigenfeldt, A., & Pasquier, P. (2010). *Realtime Generation of Harmonic Progressions Using Controlled Markov Selection*. 10.

Eladhari, M., Nieuwdorp, R., & Fridenfalk, M. (2006). The Soundtrack of Your Mind: Mind Music - Adaptive Audio for Game Characters. *Proceedings of the 2006 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*. https://doi.org/10.1145/1178823.1178887

Elferen, I. van. (2011). ¡Un Forastero! Issues of Virtuality and Diegesis in Videogame Music. *Music and the Moving Image*, *4*(2), 30–39. https://doi.org/10.5406/musimoviimag.4.2.0030

Encabo, E. (2018). *Sound in motion: Cinema, videogames, technology and audiences*. Cambridge Scholars Publishing.

Experience-driven procedural music generation for games. (2012, July 1). [Text]. IEEE Computer Society. https://doi.org/10.1109/TCIAIG.2012.2212899

Farbood, M., & Schoner, B. (2001). Analysis and Synthesis of Palestrina-Style Counterpoint Using Markov Chains. *ICMC*. http://alumni.media.mit.edu/~schoner/papers/ICMC01_PALESTRINA.pdf

Free Spirited Gamers. (2019, July 14). *20 Great Pixel Art Games 2019 & Beyond*. https://www.youtube.com/watch?v=wqpEJo3whYc

Game Audio Table—Audio Middleware Implementation Table.htm. (2015). file://localhost/Users/yihui/Zotero/storage/CD37LULA/Game%20Audio%20Table%20-%20Audio%20Middleware%20Implementation%20Table.htm

Geelen, T. van. (2017). Realizing groundbreaking adaptive music. In *From Pac-Man to Pop Music* (pp. 93–102).

Gibbons, W. (2009). Blip, Bloop, Bach? Some Uses of Classical Music on the Nintendo Entertainment System. *Music and the Moving Image*, *2*(1), 40–52.

Gibbons, W. (2017, May 25). *Music, Genre, and Nationality in the Postmillennial Fantasy Role-Playing Game*. The Routledge Companion to Screen Music and Sound. https://doi.org/10.4324/9781315681047-34

Goel, K., Vohra, R., & Sahoo, J. K. (2014). Polyphonic Music Generation by Modeling Temporal Dependencies Using a RNN-DBN. *ArXiv:1412.7927 [Cs]*, *8681*, 217–224. https://doi.org/10.1007/978-3-319-11179-7_28

Gordon, M. (2014). *Killer Instinct—Dynamic Music Breakdown*. https://vimeo.com/84491216

Grimshaw, M., & Schott, G. (2007). *A conceptual framework for the design and analysis of first-person shooter audio—University of Bolton Institutional Repository (UBIR)*. http://ubir.bolton.ac.uk/429/

Grundberg, S., & Hansegard, J. (2014). *YouTube Star Plays Videogames, Earns $4 Million a Year—WSJ*. https://www.wsj.com/articles/youtube-star-plays-videogames-earns-4-million-a-year-1402939896

Gungormusler, A., Paterson-Paulberg, N., & Haahr, M. (2015, February 11). *barelyMusician: An Adaptive Music Engine for Video Games*. Audio Engineering Society Conference: 56th International Conference: Audio for Games. http://www.aes.org/e-lib/browse.cfm?elib=17598

Hart, I. (2014). Meaningful Play: Performativity, Interactivity and Semiotics in Video Game Music. *Musicology Australia*, *36*(2), 273–290. https://doi.org/10.1080/08145857.2014.958272

Hassani, Z., & Wuryandari, A. I. (2016). Music generator with Markov Chain: A case study with Beatme Touchdown. *2016 6th International Conference on System Engineering and Technology (ICSET)*, 179–183. https://doi.org/10.1109/ICSEngT.2016.7849646

Herrman, J. (2017). *YouTube's Monster: PewDiePie and His Populist Revolt*. 1.

Hoffert, P. (2007). *Music for new media: Composing for videogames, web sites, presentations, and other interactive media*. Berklee Press.

Horowitz, S., & Looney, S. R. (2014). *The Essential Guide to Game Audio* (1st ed.). CRC Press. http://www.crcnetbase.com/isbn/9781134595372

Hou, C. (2020). *关于动态音乐设计的思考-Part 1-设计分类学*. Audiokinetic 官方 Weixin Official Accounts Platform. http://mp.weixin.qq.com/s?__biz=MzAwMTMxMjQwOA==&mid=2247484523&idx=1&sn=f322a016b2880f88a13d8748881b0e23&chksm=9adad79cadad5e8adfa48a06bff0640a60c1638bdc66fc0aa820bd15d97d4d8e8359cd69459e#rd

Huang, W., & Zhang, Y. (2020). Application of Hidden Markov Chain and Artificial Neural Networks in Music Recognition and Classification. *Proceedings of 2020 the 6th International Conference on Computing and Data Engineering*, 49–53. https://doi.org/10.1145/3379247.3379276

Hunicke, R., LeBlanc, M., & Zubek, R. (2004). MDA: A formal approach to game design and game research. *Proceedings of the AAAI Workshop on Challenges in Game AI*, *4*(1), 1722.

Ishizuka, K., & Onisawa, T. (2008). Generation of Variations on Theme Music Based on Impressions of Story Scenes Considering Human's Feeling of Music and Stories. *Int. J. Comput. Games Technol.*, *2008*, 3:1-3:9. https://doi.org/10.1155/2008/281959

Ishizuka, K., & Onisawa, T. (2006). Generation of Variations on Theme Music Based on Impressions of Story Scenes. *Proceedings of the 2006 International Conference on Game Research and Development*, 129–136. http://dl.acm.org/citation.cfm?id=1234341.1234363

Jacob, B. L. (1996). Algorithmic composition as a model of creativity. *Organised Sound*, *1*(3), 157–165. https://doi.org/10.1017/S1355771896000222

Kaae, J. (2017, October 3). *Theoretical approaches to composing dynamic music for video games*. From Pac-Man to Pop Music. https://doi.org/10.4324/9781351217743-6

Kamp, M. (2015). *Four ways of hearing video game music*. 10.

Kamp, M., Summers, T., & Sweeney, M. (2016). *Ludomusicology: Approaches to video game music*. Equinox Publishing.

Kanaga, D. (2012). *Played Meaning (Concerning the Spiritual in Games)*. 6.

Kapsecker, M. (2018). *What is the difference between a First Order Markov Model and a Second Order Markov Model? Also, What are the differences between a Markov Model and a Hidden Markov Model (HMM)? | Data Science and Machine Learning*. https://www.kaggle.com/getting-started/47042

Koebner, T. (2004). *Musik zum Abschied. Zur Komposition von Melodramen*. 23.

Kolisch, R., & Mendel, A. (1943). Tempo and Character in Beethoven's Music—Part I. *The Musical Quarterly*, *29*(2), 169–187. JSTOR.

Kregor, J. (2017). PROGRAM MUSIC: Understanding the Leitmotif: From Wagner to Hollywood Film Music. *Music Library Association. Notes; Philadelphia*, *73*(3), 547–550.

Lagim, B. A. (2002). *The Music of Anarchy Online: Creating Music for MMOGs*. http://www.gamasutra.com/view/feature/131361/the_music_of_anarchy_online_.php

Lamp, L. (2020). *Design in Art: Emphasis, Variety and Unity*. Sophia. https://www.sophia.org/tutorials/design-in-art-emphasis-variety-and-unity?utm_campaign=share&utm_content=tutorial&utm_medium=facebook&utm_source=sophia

Lerner, N. (2018). *Origins of Musical Style In Video Games, 1977–1983—Oxford Handbooks*. 25.

Lewis, G. E. (2000). Too Many Notes: Computers, Complexity and Culture in "Voyager." *Leonardo Music Journal*, *10*, 33–39.

Li, T., Choi, M., Fu, K., & Lin, L. (2019). Music Sequence Prediction with Mixture Hidden Markov Models. *2019 IEEE International Conference on Big Data (Big Data)*, 6128–6132. https://doi.org/10.1109/BigData47090.2019.9005695

Liebe, M. (2013). Interactivity and Music in Computer Games. In *Music and Game: Perspectives on a Popular Alliance* (pp. 41–62). https://doi.org/10.1007/978-3-531-18913-0_2

Lissa, Z. (1965). *Ästhetik der Filmmusik*. http://www.filmmusik.uni-kiel.de/Beitraege/BeitraegePDFS/KB02.pdf#page=140

Liu, G. (2012). 起承转合关系在音乐作品结构中的体现—图文—百度文库. https://wenku.baidu.com/view/4ab4cf914afe04a1b171de73.html

Livingstone, S. R., & Brown, A. R. (2005). *Dynamic Response: Real-Time Adaptation for Music Emotion*. *123*, 105–111.

Magerko, B. (2005). *Story Representation and Interactive Drama*. 6.

Marguc, J. (2015). *UNITY'S AUDIO SYSTEM UNDER THE HOOD*. https://files.unity3d.com/janm/UniteEurope2015.pdf

Marks, A. (2009). *The complete guide to game audio: For composers, musicians, sound designers, and game developers*. Focal Press/Elsevier.

McCuskey, M. (2003). *Beginning game audio programming*. Premier Press.

McIntyre, R. A. (1994). Bach in a box: The evolution of four part Baroque harmony using the genetic algorithm. *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*, 852–857 vol.2. https://doi.org/10.1109/ICEC.1994.349943

Medina-Gray, E. (2014). *Modular Structure and Function in Early 21st-Century Video Game Music* [Ph.D., Yale University]. https://search.proquest.com/docview/1542023712/abstract/A685376C11BC49F9PQ/1

Medina-Gray, E. (2019). Analyzing Modular Smoothness in Video Game Music. *Music Theory Online, 25*(3). https://doi.org/10.30535/mto.25.3.2

Mitchell, G., & Clarke, A. (2007). *Videogame Music: Chiptunes byte back?* 7.

Moore, F. R. (1990). *Elements of Computer Music*. Prentice-Hall, Inc.

Munday, R. (2007). Music in Video games. In J. Sexton (Ed.), *Music, Sound and Multimedia: From the Live to the Virtual* (pp. 51–67). Edinburgh University Press; Cambridge Core. https://www.cambridge.org/core/books/music-sound-and-multimedia/music-in-video-games/CEC7C37055FE87E4AC5F9768E4C530B0

Mundhenke, F. (2013). Resourceful Frames and Sensory Functions – Musical Transformations from Game to Film in Silent Hill. In P. Moormann (Ed.), *Music and Game: Perspectives on a Popular Alliance* (pp. 107–124). Springer Fachmedien Wiesbaden. https://doi.org/10.1007/978-3-531-18913-0_6

Nakamura, J.-I., Kaku, T., Hyun, K., Noma, T., & Yoshida, S. (1994). Automatic background music generation based on actors' mood and motions. *The Journal of Visualization and Computer Animation*, *5*(4), 247–264. https://doi.org/10.1002/vis.4340050405

Naushad, A. (2013). Condition Driven Adaptive Music Generation for Computer Games. *International Journal of Computer Applications*, *64*(8), 6–10. https://doi.org/10.5120/10652-5416

Nelson, C., & Wünsche, B. C. (2007). Game/Music Interaction: An Aural Interface for Immersive Interactive Environments. *Proceedings of the Eight Australasian Conference on User Interface - Volume 64*, 23–26. http://dl.acm.org/citation.cfm?id=1273714.1273718

Neumeyer, D. (2013). *The Oxford Handbook of Film Music Studies*. Oxford University Press.

Newcomb, D. L. (2012). *The Fundamentals of the Video Game Music Genre* [D.M.A., James Madison University]. https://search.proquest.com/docview/1015388890/abstract/F407C3B0580E447EPQ/1

Nick DMN. (2016). *Eyedol Komplete Dynamic Theme W/Gameplay*. https://www.youtube.com/watch?v=nEnaOmXW0Ck&feature=emb_logo

Nierhaus, G. (2009). *Algorithmic Composition—Paradigms of Automated Music Generation*. https://link.springer.com/content/pdf/10.1007%2F978-3-211-75540-2.pdf

Nuanáin, C. Ó. (2019, March 15). *A Brief History of Machine-Assisted Music in Video Games*. Medium. https://medium.com/the-sound-of-ai/a-brief-history-of-machine-assisted-music-in-video-games-4f249d6b7ef5

Ogata, T. (2003). *Music Generation by Transformation- Toward the Narratology of Music-*.

Pachet, F. (2003). The Continuator: Musical Interaction With Style. *Journal of New Music Research, 32*(3), 333–341. https://doi.org/10.1076/jnmr.32.3.333.16861

Peerdeman, P. (2010). *Sound and Music in Games*. https://peterpeerdeman.nl/vu/ls/peerdeman_sound_and_music_in_games.pdf

Pennycook, B. (1991). Machine Songs II: The “PRAESCIO” Series - Composition-Driven Interactive Software. *Computer Music Journal*, *15*(3), 16–26. https://doi.org/10.2307/3680762

Pennycook, B. (1997). Live Electroacoustic Music: Old Problems, New Solutions. *Journal of New Music Research*, *26*(1), 70–95.

Phillips, W. (2014). *A Composer’s Guide to Game Music*. MIT Press.

Pichlmair, M., & Kayali, F. (2007). *Levels of Sound: On the Principles of Interactivity in Music Video Games*.

Plans, D., & Morelli, D. (2012). Experience-Driven Procedural Music Generation for Games. *IEEE Transactions on Computational Intelligence and AI in Games*, *4*(3), 192–198. https://doi.org/10.1109/TCIAIG.2012.2212899

Potter, J. D. S., Robert F. (2018). Everything Merges with the Game: A Generative Music System Embedded in a Videogame Increases Flow. *Game Studies*, *18*(2). http://gamestudies.org/1802/articles/sites_potter

Prechtl, A., Laney, R., Willis, A., & Samuels, R. (2014). *Algorithmic music as intelligent game music—Open Research Online*. http://oro.open.ac.uk/39795/

Ramanto, A. S., No, J. G., & Maulidevi, D. N. U. (2017). Markov Chain Based Procedural Music Generator with User Chosen Mood Compatibility. *International Journal of Asia Digital Art and Design Association*, *21*(1), 19–24.

Rayman, J. (2014). *Experimental Approaches to the Composition of Interactive Video Game Music*. 32.

Reese, K., Yampolskiy, R., & Elmaghraby, A. (2012). A framework for interactive generation of music for games. *2012 17th International Conference on Computer Games (CGAMES)*, 131–137. https://doi.org/10.1109/CGames.2012.6314564

Reiser, M. (1993). Wagner’s use of the leitmotif to communicate understanding. In *Psychoanalytic explorations in music: Second series* (pp. 217–228). International Universities Press, Inc.

Reyland, N. (2012). The Beginnings of a Beautiful Friendship?: Music Narratology and Screen Music Studies. *Music, Sound, and the Moving Image*, *6*(1), 55–71. https://doi.org/10.3828/msmi.2012.6

Riedl, M. O., & Stern, A. (2006). Believable Agents and Intelligent Story Adaptation for Interactive Storytelling. In S. Göbel, R. Malkewitz, & I. Iurgel (Eds.), *Technologies for Interactive Digital Storytelling and Entertainment* (pp. 1–12). Springer. https://doi.org/10.1007/11944577_1

Robertson, J., Quincey, A. de, Stapleford, T., & Wiggins, G. (1998). Real-Time Music Generation for a Virtual Environment. *Proceedings of Ecai-98 Workshop on Ai/Alife and Entertainment*.

Ross, D. (2015). *Video game music: Classic FM Handy Guides*. Elliott & Thompson. http://public.eblib.com/choice/publicfullrecord.aspx?p=1911142

Rothgeb, J. (1987). Review of Brahms and the Principle of Developing Variation [Review of *Review of Brahms and the Principle of Developing Variation*, by W. Frisch]. *Music Theory Spectrum*, *9*, 204–215. JSTOR. https://doi.org/10.2307/746127

Rowe, R. (1993). *Interactive music systems: Machine listening and composing*. MIT Press.

Rowe, R. (2001). *Machine musicianship*. MIT Press.

Rumsey, F. (2015). Game Audio: Generative Music, Emotions, and Realism. *Journal of the Audio Engineering Society*, *63*(4), 293–297.

Schifrin, L. (2011). *Music Composition for Film and Television*. Hal Leonard Corporation.

Scirea, M., Nelson, M. J., & Togelius, J. (2015). *Moody Music Generator: Characterising Control Parameters Using Crowdsourcing*. https://link.springer.com/chapter/10.1007/978-3-319-16498-4_18

Scott, N. (2014). Music to Middleware: The Growing Challenges of the Game Music Composer. *Proceedings of the 2014 Conference on Interactive Entertainment*, 34:1-34:3. https://doi.org/10.1145/2677758.2677792

Sexton, J. (2007). *Music, sound and multimedia: From the live to the virtual*. Edinburgh University Press.

Shumate, J. (2016). *Real-Time Synthesis for Sound Creation in Peggle Blast!* Audiokinetic Blog.

Simons, P. M. (1988). Computer Composition and Works of Music: Variation on A Theme of Ingarden. *Journal of the British Society for Phenomenology*, *19*(2), 141–154. https://doi.org/10.1080/00071773.1988.11007856

Sjöblomab, M., & Hamari, J. (2017). *Why do people watch others play video games? An empirical study on the motivations of Twitch users—ScienceDirect*. https://www.sciencedirect.com/science/article/pii/S0747563216307208

smith-d-c. (2011). *The Greatest Film Composers of All Time & Their Best Movie*. IMDb. http://www.imdb.com/list/ls005359850/

Solbach, A. (2004). *Film und Musik: Ein klassifikatorischer Versuch in narratologischer Absicht*. 14.

Somberg, G. (Ed.). (2017). *Game audio programming: Principles and practices*. Taylor & Francis, CRC Press. https://doi.org/10.1201/9781315368696

Somberg, G. (2019). *Game Audio Programming 2: Principles and Practices*. 389.

Spector, L., & Alpern, A. (1995). *Induction and Recapitulation of Deep Musical Structure*. 20–25.

Stevens, R., & Raybould, D. (2015). *Game Audio Implementation* (1st ed.). CRC Press. http://www.crcnetbase.com/isbn/9781317679462

Stevens, R., Raybould, D., & Raybould, D. (2013). *The Game Audio Tutorial: A Practical Guide to Sound and Music for Interactive Games*. Routledge. https://doi.org/10.4324/9780240817279

Summers, G. M. T. (2014, March 26). *From Parsifal to the PlayStation: Wagner and Video Game Music*. Music In Video Games. https://doi.org/10.4324/9781315882697-17

Summers, T. (2011). Playing the Tune: Video Game Music, Gamers, and Genre. *Act - Zeitschrift für Musik & Performance*, *2011*, 2–27.

Summers, T. (2016). *Understanding Video Game Music*. Cambridge University Press.

Summers, T. R. D. (2012). *Video game music: History, form and genre* [Ph.D., University of Bristol]. https://ethos.bl.uk/OrderDetails.do;jsessionid=A55FBD88622EA3006043B51C0FDF99CF?uin=uk.bl.ethos.573894

Sweeney, M. R. (2014). *The aesthetics of videogame music* [Ph.D., University of Oxford]. https://ora.ox.ac.uk/objects/uuid:70a29850-0c0d-4abd-a501-e75224fa856a

Sweet, M. (2015). *Writing Interactive Music for Video Games: A Composer's Guide* (1st ed.). Addison-Wesley Professional.

Szinger, J. (2017). *On Composing Interactive Music: Let's time-travel to 1993*. 9.

Tan, S.-L., Cohen, A. J., Lipscomb, S. D., & Kendall, R. A. (2013). *The Psychology of Music in Multimedia*. OUP Oxford.

Tanenbaum, T. J. (2015). Hermeneutic Inquiry for Digital Games Research. *The Computer Games Journal*, *4*(1–2), 59–80. https://doi.org/10.1007/s40869-015-0005-9

TECHJVB. (2018). *A Brief History of Music in Video Games*. https://blog.audiokinetic.com/the-history-of-music-in-video-games/?utm_source=hs_email&utm_medium=email&utm_content=66194996&_hsenc=p2ANqtz-94bO64HCY9VuwGMSpnJspULp5efpjFIVo6G5p2JbKPB-YQYJ-Y5xwfQxatwCuvcVT9X1XsNmtjF7GOGvrTmo62338uzA&_hsmi=66194996

Thomas, C. (2017). *Composing Music for Games: The Art, Technology and Business of Video Game Scoring*. Routledge. https://doi.org/10.1201/b21203

Unity Technologies. (2020). *Unity - Manual: Order of Execution for Event Functions*. https://docs.unity3d.com/Manual/ExecutionOrder.html

Velardo, V. (2018). *Players Have Spoken: Video Game Music Must be Adaptive*. 13.

Verbeurgt, K., Dinolfo, M., & Fayer, M. (2004). Extracting Patterns in Music for Composition via Markov Chains. In B. Orchard, C. Yang, & M. Ali (Eds.), *Innovations in Applied Artificial Intelligence* (Vol. 3029, pp. 1123–1132). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-24677-0_115

Verbeurgt, K., Dinolfo, M., Fayer, M., & LINK (Online service) (Eds.). (2004). *Extracting Patterns in Music for Composition via Markov Chains*. Springer.

Vidiera, T. G., Pennycook, B., & Rosa, J. M. (2017). Formalizing Fado: A Contribution to Automatic Song-Making. *Journal of Creative Music Systems*, *1*(2). https://doi.org/10.5920/JCMS.2017.03

Watson, B., Müller, P., Veryovka, O., Fuller, A., Wonka, P., & Sexton, C. (2008). Procedural Urban Modeling in Practice. *IEEE Computer Graphics and Applications*, *28*(3), 18–26. https://doi.org/10.1109/MCG.2008.58

Wermter, S., Weber, C., Duch, W., Honkela, T., Koprinkova-Hristova, P., Magg, S., Palm, G., & Villa, A. E. P. (Eds.). (2014). *Artificial Neural Networks and Machine Learning – ICANN 2014: 24th International Conference on Artificial Neural Networks, Hamburg, Germany, September 15-19, 2014. Proceedings* (Vol. 8681). Springer International Publishing. https://doi.org/10.1007/978-3-319-11179-7

Whalen, Z. (2007). *Case study: Film music vs. Video-game music: The case of Silent Hill*.

Whalen, Z. N. (2004a). *Game Studies—Play Along—An Approach to Videogame Music*. http://www.gamestudies.org/0401/whalen/

Whalen, Z. N. (2004b). *Play Along: Video game music as metaphor and metonymy*.

Whitmore, G. (2016a). *PopCap Games composer Guy Whitmore discusses the advantages of scoring your game, within the game*.

Whitmore, G. (2016b, May 30). *Scoring Peggle Blast! New Dog, Old Tricks*. Audiokinetic Blog. https://blog.audiokinetic.com/zh/scoring-peggle-blast-new-dog-old-tricks/

Whitmore, G. (2018). *The Business of Music Design for Games and Dynamic Storytelling*.

Wikipedia. (2019). Pixel art. In *Wikipedia*. https://en.wikipedia.org/w/index.php?title=Pixel_art&oldid=922388637

Wikipedia. (2020a). CG Artist. In *Wikipedia*. https://en.wikipedia.org/w/index.php?title=CG_Artist&oldid=973169822

Wikipedia. (2020b). 中国五声音阶. In *维基百科，自由的百科全书*. https://zh.wikipedia.org/w/index.php?title=%E4%B8%AD%E5%9C%8B%E4%BA%94%E8%81%B2%E9%9F%B3%E9%9A%8E&oldid=63199226

Wikipedia. (2021a). 十二律. In *维基百科，自由的百科全书*. https://zh.wikipedia.org/w/index.php?title=%E5%8D%81%E4%BA%8C%E5%BE%8B&oldid=64708705

Wikipedia. (2021b). Frame rate. In *Wikipedia*. https://en.wikipedia.org/w/index.php?title=Frame_rate&oldid=1012344840

Wikipedia. (2021c). Chiptune. In *Wikipedia*. https://en.wikipedia.org/w/index.php?title=Chiptune&oldid=1014714338

Wikipedia. (2021d). Rondo. In *Wikipedia*. https://en.wikipedia.org/w/index.php?title=Rondo&oldid=1016202488

Wilde, M. D. (2004). *Audio Programming for Interactive Games: The Computer Music of Games*. Routledge. https://doi.org/10.4324/9780080472065

Williams, D., & Lee, N. (2018). *Emotion in Video Game Soundtracking*. Springer International Publishing.

Winkler, T. (1994). Technical and aesthetic considerations in interactive computer music systems. *The Journal of the Acoustical Society of America*, *96*(5), 3302–3302. https://doi.org/10.1121/1.410819

Winkler, T. (1998). *Composing interactive music: Techniques and ideas using Max*. MIT Press.

Winters, B. (2012). Music and Narrative. *Music, Sound, and the Moving Image*, *6*(1), 3–7. https://doi.org/10.3828/msmi.2012.2

Wirman, H. (2009). On productivity and game fandom. *Transformative Works and Cultures*, *3*. https://doi.org/10.3983/twc.2009.0145

Wooller, R., Brown, A. R., Miranda, E., Diederich, J., & Berry, R. (2005). A framework for comparison of process in algorithmic music systems. In B. David & E. Ernest (Eds.), *Creative Industries Faculty* (pp. 109–124). Creativity and Cognition Studios. https://eprints.qut.edu.au/6544/

Wray, T., Gillepsie, K., Sue, R., & Holmes, M. (2019). Composition in Theme and Variations Form. *Student Research Symposium*. https://digitalcommons.usu.edu/researchweek/ResearchWeek2019/All2019/280

Yabsley, A. (2007). *The Sound of Playing: A Study into the Music and Culture of Chiptunes*. 39.

Yeh, S., Lin, P.-J., Liu, H.-Y., & Chen, R.-M. (2002). The implementation of interactive music system for massive multi-player online game. *Fourth International Symposium on Multimedia Software Engineering, 2002. Proceedings.*, 11–16. https://doi.org/10.1109/MMSE.2002.1181590

Young, D. M. (2012). *Adaptive Game Music: The Evolution and Future of Dynamic Music Systems in Video Games* [Ohio University]. https://etd.ohiolink.edu/pg_10?0::NO:10:P10_ACCESSION_NUM:ouhonors1340112710#abstract-files

Young, D. M. (2013, February 6). *The Future of Adaptive Game Music: The Continuing Evolution of Dynamic Music Systems in Video Games*. Audio Engineering Society Conference: 49th International Conference: Audio for Games. http://www.aes.org/e-lib/browse.cfm?elib=16655

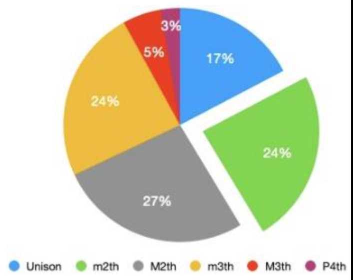
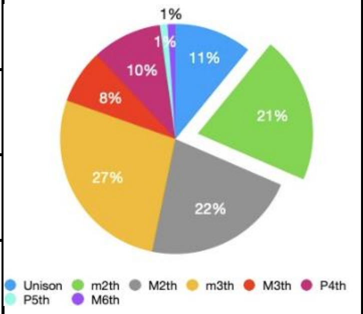
Youxiputao. (2020). 资深商务离职做研发：七人团队用虚幻引擎做了款「万人同玩」的游戏. https://mp.weixin.qq.com/s/P3xzqK-taAqr0ZdxOic1Ag

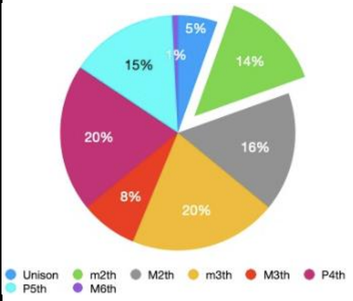
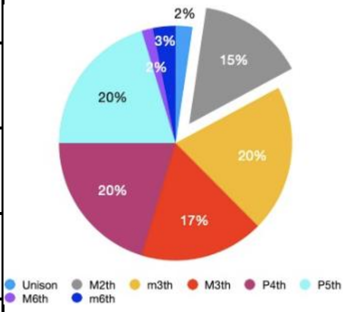
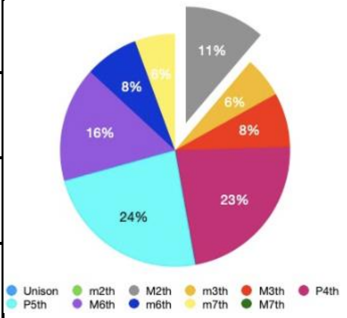
Yuhua, W. (2019). *谷歌 OwlchemyLabs 经验分享：契合的配乐对 VR 游戏设计效果非常好—映维网*. https://yivian.com/news/68250.html

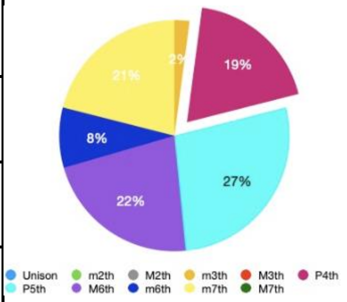
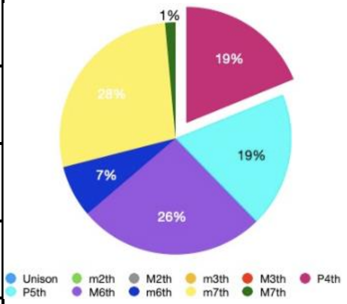
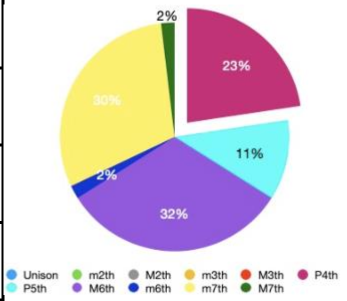
Zehnder, S. M., & Lipscomb, S. D. (2006). The role of music in video games. *Playing Video Games: Motives, Responses, and Consequences*, 283–303. https://doi.org/10.4324/9780203873700

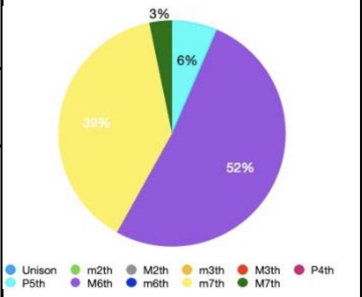
Zicarelli, D. (1987). M and Jam Factory. *Computer Music Journal*, *11*(4), 13–29. https://doi.org/10.2307/3680237

Appendix A: Ranking Number Differences and Melodic Intervals

Ranking Number Differences (*excluding* E4 and *F#4*)	Interval Types	Location in the segment (represented by MIDI notes associated with PCs)	Number of Occurrences in the segment	Probability of occurrence
0	Unison	B-B, A-A, C#-C#, C#<->B, A->C#, C#->D, D->B	13	
	m2nd	B<->A, G->B, B<->C#, C#->D	18	
	M2nd	B<->A, G->B, B<->C#, A->C#, C#->D, D->B	20	
	m3rd	B<->A, B<->C#, C#->D, D->B	18	
	M3rd	B<->C#,	4	
	P4th	A->C#, D->B	2	
1	Unison	B-B, A-A, C#-C#, C#<->B, A->C#, C#->D, D->B	13	
	m2nd	B<->A, G->B, B<->C#, C#->D, D->B	19	
	M2nd	B<->A, G->B, B<->C#, A->C#, C#->D, D->B	20	
	m3rd	B-B, A-A, C#-C#, B<->A, B<->C#, A->C#, C#->D, D->B	25	
	M3rd	B-B, A-A, G->B, D->B	7	
	P4th	B-B, A-A, C#-C#, A->C#, C#->D, D->B	9	
	P5th	G->B	1	
	M6th	A->C#	1	
2	Unison	B-B, A-A, A->C#, D->B	7	
	m2nd	B<->A, G->B, B<->C#, D->B	18	

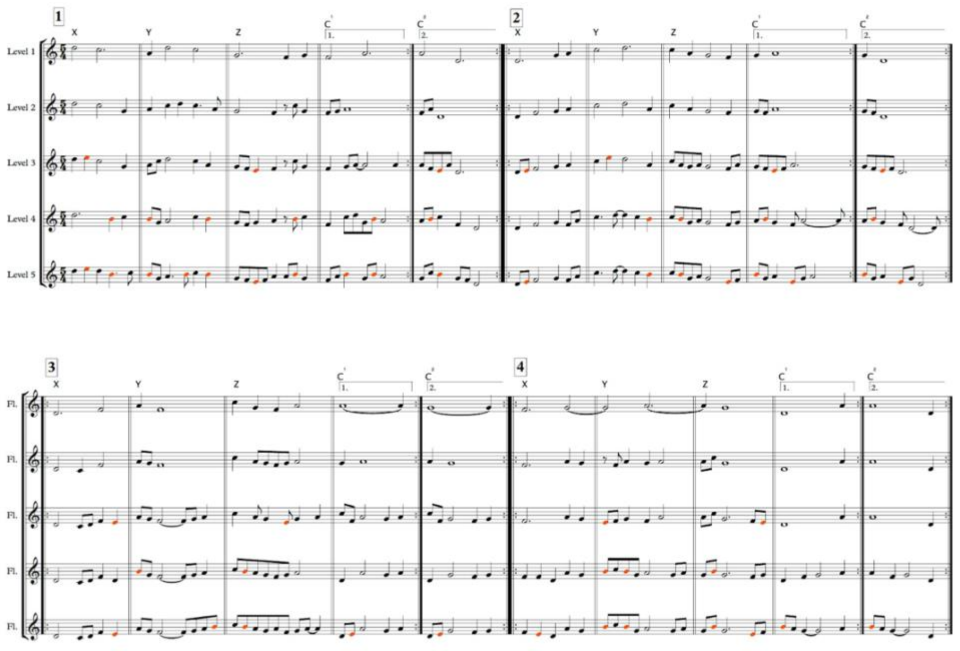
	M2nd	C#-C#, B<->A, G->B, B<->C#, A->C#, C#->D, D->B	21	
	m3rd	B-B, A-A, C#-C#, B<->A, G->B, B<->C#, A->C#, C#->D, D->B	26	
	M3rd	B-B, A-A, C#-C#, G->B, A->C#, C#->D, D->B	10	
	P4th	B-B, A-A, C#-C#, B<->A, G->B, B<->C#, A->C#, C#->D, D->B	26	
	P5th	B<->A, G->B, B<->C#, C#->D, D->B	19	
	M6th	A->C#	1	
3	Unison	A-A, A->C#,	3	
	M2nd	B<->A, G->B, B<->C#, A->C#, D->B	19	
	m3rd	B-B, A-A, C#-C#, B<->A, G->B, B<->C#, A->C#, C#->D, D->B	26	
	M3rd	B-B, A-A, C#-C#, B<->A, G->B, A->C#, C#->D, D->B	22	
	P4th	B-B, A-A, C#-C#, B<->A, G->B, B<->C#, A->C#, C#->D, D->B	26	
	P5th	B-B, A-A, C#-C#, B<->A, G->B, B<->C#, A->C#, C#->D, D->B	26	
	m6th	G->B, A->C#, C#->D, C#-C#,	4	
	M6th	A->C#, D->B	2	
4	M2nd	B<->A,	12	
	m3rd	B-B, A-A, G->B,	6	
	M3rd	B-B, A-A, C#-C#, A->C#, D->B	8	
	P4th	B-B, C#-C#, B<->A, G->B, B<->C#, A->C#, C#->D, D->B	24	
	P5th	B-B, A-A, C#-C#, B<->A, G->B, B<->C#, C#->D, D->B	25	
	m6th	B-B, C#-C#, G->B, A->C#, C#->D, D->B	8	

	M6th	A-A, C#-C#, B<->A, C#->D, D->B	17	
	m7th	B<->C#, C#->D, D->B	6	
5	m3rd	A-A	2	
	P4th	B<->A, G->B, B<->C#, A->C#	18	
	P5th	B-B, A-A, C#-C#, B<->A, G->B, B<->C#, A->C#, C#->D, D->B	26	
	m6th	B-B, C#-C#, G->B, A->C#, C#->D, D->B	8	
	M6th	A-A, C#-C#, B<->A, B<->C#, C#->D, D->B	21	
	m7th	C#-C#, B<->A, G->B, B<->C#, C#->D, D->B	20	
6	P4th	B<->A, A->C#	13	
	P5th	B-B, A-A, C#-C#, G->B, B<->C#, A->C#, D->B	13	
	m6th	B-B, G->B, D->B	5	
	M6th	B<->A, B<->C#, C#->D, D->B	18	
	m7th	C#-C#, B<->A, G->B, B<->C#, D->B	19	
	M7th	G->B	1	
7	P4th	B<->A	12	
	P5th	B-B, A-A, G->B	6	
	m6th	G->B	1	
	M6th	B<->A, B<->C#, D->B	17	
	m7th	B<->A, B<->C#	16	
	M7th	G->B	1	
8	P5th	A-A	2	

	M6th	B<->A, B<->C#	16	
	m7th	B<->A	12	
	M7th	G->B	1	
9	m7th	B<->A	12	

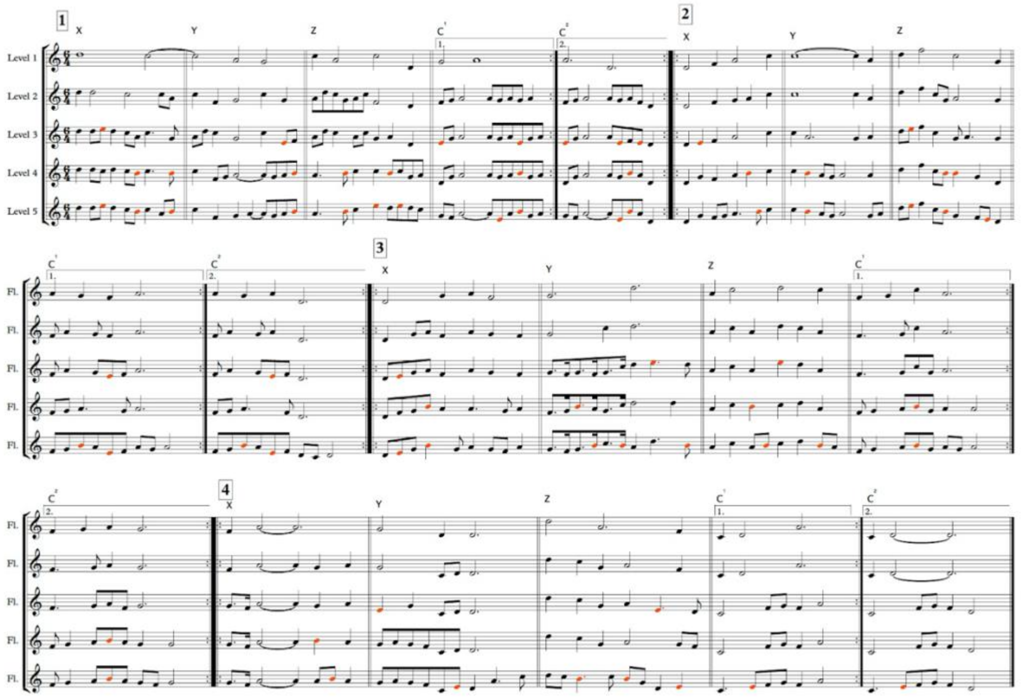
Appendix B: Scores of All Quarter Phrases for the Four Episodes with Five Diminution Levels in Three Meters Played on *Dizi* in *First Task*

First Task - Dizi Episodes - 5-Beat Modules

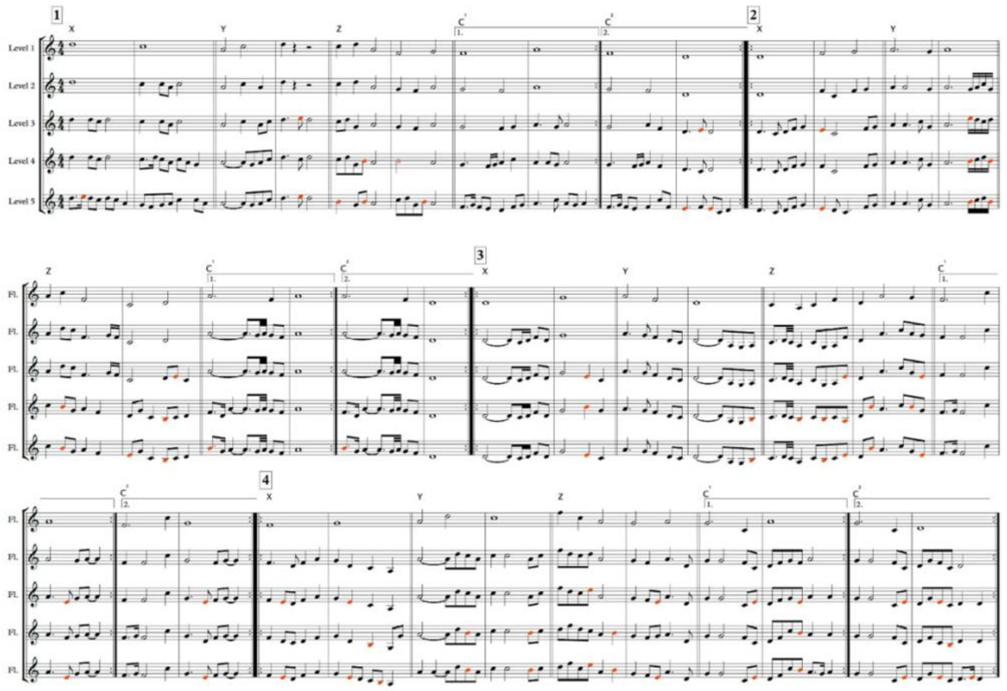


First Task - Dizi Episodes - 5-Beat Modules

First Task - Dizi Episodes - 6-Beat Modules



First Task - Dizi Episodes - 8-Beat Modules



Appendix C: Timing and Parameter Summaries in *First Task*

Instrument	Music Parameter	Generation Nodes or Dynamic Change	Change Behavior and Timing	Interacts with Game Parameters	Details
Sanxian	Response Phrase	Alternating with *Dizi*	After every *Dizi* episode	No	Programmed in Wwise
Dizi	Episodes	Episode No.1	Continuous looping in order from 1-4	No	Programmed in Wwise
		Episode No.2			
		Episode No.3			
		Episode No.4			
	Meter (Module Length)	5/4 (5-beat)	Change on every episode	No	Randomly chosen by the game code in Unity
		6/4 (6-beat)			
		4/4 (8-beat)			
	Starting Module	X	Change on every first quarter phrase	No	Randomly chosen by the game code in Unity
		Y			
		Z			
	Phrase Structure	Sentence	Change on every second quarter phrase	Yes	Player attacks enemy
		Period			Player does not attack
		Parallel Period			Player attacks breakable object
	Level of Diminution	Level 1	On every frame	Yes	Player Dash & Crouch
		Level 2			Player Idle & Meditate

		Level 3			Player Walk & Run
		Level 4			Player Jump & Wall Jump
		Level 5			Player Attack & Hurt
	Type of Cadence	No Cadence	Set by the second module phrase	No	Unmute by choosing the sentence phrase type
		Half Cadence			Programmed in Wwise
		Authentic Cadence	Set by the fourth module phrase		Programmed in Wwise
Bangzi and *Guqin*	Pattern	4-beat pattern	In the introduction section	No	
			With 8-beat module of the *Dizi* section	Yes	Randomly chosen in Unity
			With the *Sanxian* section	No	Programmed in Wwise
		5-beat pattern	With 5-beat module of the *Dizi* section	Yes	Randomly chosen by the game code in Unity
		6-beat pattern	With 6-beat module of the *Dizi* section	Yes	Randomly chosen by the game code in Unity
Dagu	Play or not play	Play	Episodes 1 & 3	No	Programmed in Wwise
		Not play	Episodes 2 & 4		
Luo	Articulation	Rising tone	On the last beat of the introduction section	No	Programmed in Wwise
		Dropping tone	On the last two beats of the *Sanxian* section		

Appendix D: Links to Project Demo Videos, Albums, and Game Executables

- DOD's project demo videos YouTube playlist:

  https://youtube.com/playlist?list=PL9A70Kv75WGcV5cCVLG9H_TAakdoEDzM-

- DOD game executables:

  https://drive.google.com/drive/folders/1rdQVdENJJv4zcK81mGxld8zQmGdaVX83?usp=sharing

- Mastery's project demo videos YouTube playlist:

  https://youtube.com/playlist?list=PL9A70Kv75WGeV5ws6clNnl6uMLaTzDrxl

- Mastery game executables:

  https://drive.google.com/drive/folders/114iFLncUwOx4ZoY5lEV_jH0pjCZvIEjr?usp=sharing

- Full album of DOD's game soundtracks: https://melismaticwitch.bandcamp.com/album/do-or-die-a-roll-playing-adventure-original-video-game-soudtracks

- Full album of Mastery's game soundtracks:

  https://melismaticwitch.bandcamp.com/album/mastery-original-video-game-soundtracks