AUDIOVERDRIVE:
EXPLORING BIDIRECTIONAL RELATIONSHIPS BETWEEN MUSIC AND GAME

By

Nils Iver Holtar

Masters thesis, *IT University of Copenhagen*, April 2013

Supervisors: *Mark Jason Nelson* and *Julian Togelius*

ABSTRACT

This thesis explores a method to make music an integrated part of digital games, showing strategies in elevating its role to more than an aesthetic contributor. By looking at existing terminologies, I offer a useful vocabulary to aid in understanding existing dynamic music implementations. This vocabulary will be used when looking at examples of games that attempt to push both music and game equally into the foreground. From here, the conceptual idea of music and game forming a bidirectional relationship will be explored. I present a mapping framework consisting of a custom created application that connects parameters from *Audioverdrive*, a game designed specifically for this thesis, with parameters from the commercial music software *Ableton Live*. Through the presentation of one possible configuration of this framework, I hope to highlight the potential for novel gameplay experiences and new composition and game design workflows found in such a system.

TABLE OF CONTENTS

1. Introduction 5

2. The activity based framework for describing games 6

3. Background – games and music 9

3.1 Dynamic music in games 9

3.1.1 Game time and music time 9

3.1.2 Linearity and non-linearity 13

3.1.3 Quantization 15

3.1.4 Discrete and continuous changes 16

3.2 Tighter relations between game and music 17

3.2.1 Diegesis in games 17

3.2.2 The composition-instrument 18

3.2.3 Bidirectional relationships 20

4. Game analyses 22

4.1 Otocky 22

4.1.1 Music implementation 23

4.1.2 Game state influence on music 24

4.1.3 Music influence on game state 25

4.2 Sound Shapes 25

4.2.1 Music implementation 26

4.2.2 Game state influence on music 27

4.2.3 Music influence on game state 28

4.3 Moving forward 29

5. The mapping framework 30

5.1 Ableton Live 30

5.2 The mapping application 32

5.3 A parameter 32

5.4 A mapping 32

5.5 Audioverdrive 33

6. The demo configuration 37
6.1 Live-to-game mappings 37
6.2 Game-to-Live mappings 38
6.3 Composition of gameplay 39
6.4 Examination 40
6.4.1 Game state influence on music 40
6.4.2 Music influence on game state 41
6.4.3 Closing remarks 42
7. Conclusion 44
Acknowledgements 45
Bibliography 45
Supplemental material 46

1. INTRODUCTION

In the 1980s, digital game music was realized through the use of simple synthesis techniques offered by various chips featured in the available hardware. Even though timbre and polyphony was limited, creativity was not, which is evident in how many commemorated melodies in gaming history come from this era[1]. In present day, chip-tune aesthetic has experienced an increased revival, for example in its appearance in popular electronic music genres like Dubstep and Hiphop. Comparing hardware from the 1980s to current off-the-shelf hardware reveals a quantum leap in computer processing power, and with this, the possibility space of sonic expression in digital games has also increased drastically. Sample rates have reached sufficient heights, enabling the full traversal of the frequency range perceivable to humans. Complex audio processing and manipulation can now be achieved in realtime, and synthesis techniques have become more complex and expressive. To top it off, most of these techniques can now be realized through off-the-shelf hardware, making this a very lucrative time for people interested in exploring and creating audio.

Being both a game designer and a musician, the combining of these two creative practices has been a big motivation for this thesis. The game ideas I come up with tend to be initially inspired from musical ideas, but as the game is solidified, the two tend to separate themselves as the musical idea is reduced to a sole aesthetic contributor. This thesis will mainly focus on the relationship between music and games, and how these two separate processes can be combined in ways that challenge their usual assumed roles. By looking at existing literature and existing games, I aim to offer a useful summary of the common challenges in the field of dynamic music while hopefully also demystifying the occasional black-box nature of some of their implementations. From there, the concept of a *bidirectional relationship* between music and game will be presented. Two games, namely *Otocky* and *Sound Shapes*, will undergo closer examination, showing some consequences of applying this concept in practice. I will then present my main contribution; the mapping framework. The mapping framework is realized as an application that sits between *Audioverdrive*, an iPad game programmed for this study, and the commercial *DAW*[2] software *Ableton Live*. The application lets users create mappings between game and music parameters, bringing the well-established workflow conventions of Ableton Live into the realm of game and level design. This offers the creative exploration of bidirectional relationships between music and game, and to illustrate this, I will showcase an example of a possible configuration created using this tool.

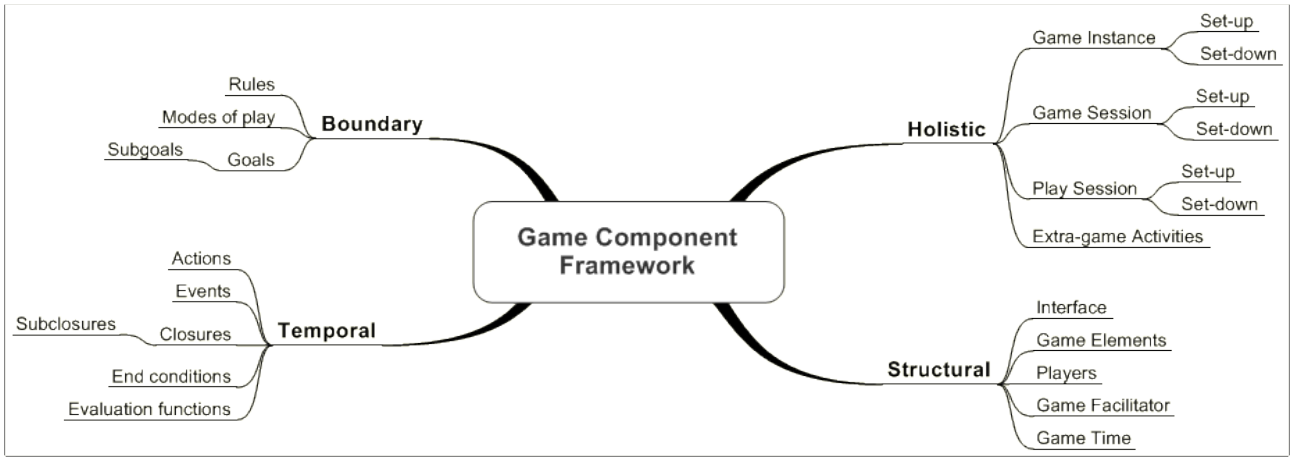
1 *The Nintendo Entertainment* system accommodates many widely-known melodies such as the main stage theme from *Super Mario Brothers* (1985), the main theme of *The Legend Of Zelda* (1986) and various music from the *Mega Man* (1987) series.

2 *DAW* is an abbreviation for Digital Audio Workstation, and is a term used to describe a software that can record, process, mix and export sound. DAWs are also expected to support third party plugin standards like VST.

2. THE ACTIVITY BASED FRAMEWORK FOR DESCRIBING GAMES

When looking closer at games, it is useful to have a point of departure. In the following chapters, terminologies and attributes pertaining to games are taken from Björk and Holopainen's activity based framework, introduced in their book *Patterns in Game Design* [1]. The framework assumes that playing a game is the equivalent of performing quantifiable changes to its *state*, where a game's state is defined as the full collection of values, elements and relationships between them. This assumption is especially useful in this study, as big focus is found in examining the various possible correlations between music and game at points in time.

The framework divides game components into four categories, namely *holistic*, *boundary*, *temporal* and *structural*. The figure below shows all components of this framework and how they are divided. As not all components bear equal amounts of significance in this study, the following summary will primarily be focused on those that do, while still aiming to provide a sufficient overview of the whole.



Holistic components are primarily used to separate the activity of playing a game from other activities, setting some boundaries for what is considered as *part of playing a game*. A useful separation is made between game *instance* and game and play *sessions*. A game instance exists from the point where the game begins to the point where it ends. A game session is the total duration of one players involvement in a game instance, while a play session is the duration of each stretch where the player is actively playing the game. Often, game sessions and play sessions are equivalent, except cases where the players' presence in the game can be said to overlap between play sessions. In this study, I will primarily use *game session*, as the games that will be looked at do not require this conceptual split.

Boundary components are what explicitly state what can and what cannot be done within the game. *Rules* are the limitations imposed on players and other game objects at any time, actively

dictating what is possible to do and when. They also govern by which means the other parts of the framework are instantiated, making this component usually the most significant when picking apart the inner workings of a game. But while rules do describe the way a game works, they do not describe *why* one should play it, which is why *goals* are introduced. *Goals* primarily describe favorable game states for individual players, while the rules describe how these states are reached. Finally, the *modes of play* component can essentially be described as various configurations of *rules* and *goals*. While the possibility of changing a *mode of play* is arguably part of the rule itself, the separation is useful when describing the game. A good example of this is found in the recently released *Knytt Underground*[3], a two-dimensional platform game where the player can switch between controlling a ball with bouncing physics and a girl with conventional platformer physics.

Temporal components are used to describe the development of a game state over time. They are introduced primarily as a means to make a detailed breakdown of the unfolding of a game session, ideally identifying some high level reasons or tendencies behind the game state changes. Low level temporal components are *actions* and *events*. Events are isolated, perceivable changes to the game state, while actions are events that occur as a direct consequence of player input. In this study, actions will consequently be referred to as *player actions*, as this is more convenient for showing the context. Closures are used as a higher level grouping of game events, detailing an event sequence that conveys some information relevant to the game. An obvious example of a closure would be the point where a player reaches a goal or subgoal. Another example could be when the player realizes that a goal cannot be reached, something that could result in defeat or a change of strategy. *End conditions* are the collection of game states that effectively end the game, and *evaluation functions* are the operations that determine the outcome of that particular game. Determining the winner, or the total score, are both examples of what an evaluation function would do. While end conditions and evaluation functions are inherent properties of rules and goals, there are many cases where it makes sense applying them in the temporal domain – for example in cases where, as the game session unfolds, the current game state is frequently evaluated in relation to the likelihood of winning.

Finally, *structural* components deal with exactly how game states are represented to the player, dividing elements into those under player agency and those under system agency. *Players* are the objects in the game that offer the most direct agency. As there is some ambiguity to whether *Players* refers to the actual human playing the game, or the actual in-game object, I will consistently replace this term with *Avatars* instead. *Game elements* refer to those elements under system agency.

3 *Knytt Underground* is a game by *Nifflas Games*, and saw its first release in 2012. It is currently available on *Sony PS3*, *PSVita* and *FastSpring*. (Official website: http://www.knyttunderground.com/)

Game time is a component dealing with how game state changes relate to "real time" changes. In turn based games like card games, for instance, the game state changes each time a player uses a turn, but the amount of "real time" spent on a turn is very variable.

Note that this framework is meant to cater for all kinds of games, whereas this thesis will only focus on *digital games*. The reader can therefore safely assume that whenever the term *game* is used, it will always be a *digital game*.

3. BACKGROUND – GAMES AND MUSIC

The presence of music enhances the gaming experience through complementing the on-screen action is a common assumption. This is also often the advocated reason as to why one should strive for implementing music in ways that make it closely and accurately follow the game state. An example of this can be found in an article by Van Geelen[2], where he enthusiastically gives vivid descriptions of envisioned experiences. The following sections will study the relationship between music and games, with a primary focus on implementation strategies, using some existing literature and terminologies that have appeared in the field.

3.1 Dynamic music in games

Collins [3] summarizes some useful game-centered terminologies when classifying audio in games. *Interactive* audio is audio that occurs as a direct result of player actions. Some notable examples of this would be the sound effects tied to actions such as jumping, walking and shooting. *Adaptive* audio is audio that follows game state changes that occur outside player agency. Finally, *dynamic* audio is audio that changes in tandem with the game state, including changes that result from player actions. Narrowing it down, dynamic game music is music that at any point is expected to some how represent the game state as the player inflicts changes upon it. It is dynamic game music that will be the main object of interest in moving forward. The following section will attempt to offer a conceptual overview regarding the composition of dynamic game music, as it inhabits some fundamental differences from more traditional forms of music composition. Terminologies from Jesper Kaae's “Theoretical approaches to composing dynamic music” [4] will the be ones primarily used.

Dynamic game music implementations are always concrete - the exact workings are always fully defined in the game code. However, since most implementations are proprietary, the actual workings are rarely completely transparent, so some assumptions have to be made. It is also probable that many implementations are bound to be finely tuned to their specific game and musical structure/aesthetics, employing special cases and workarounds. So while overall strategies will be covered, do not regard this as complete absolutes in regards to how developers concretely pursue dynamic music.

3.1.1 Game time and music time

An inherent property of sound is that it only occurs through the passing of time. Variations in air-pressure result in audible frequencies and a frequency can only be perceived when enough air pressure variations has formed a period. Traditionally in films, image and audio are fitted along one

common timeline of fixed-length. Upon pressing the pause button, the image freezes and the sound stops. If we were to apply dynamic music to movies in the same sense that we expect it for games, pausing the movie would still freeze the image while the music would continue playing. Whatever state the film was in at the time of pausing, this state would be expected to be upheld indefinitely until the film was resumed. And upon resuming, the music would still be expected to fit alongside the rest of the film, despite the two timelines being initially disjointed. One could say that dynamic music implementation is pursuit of create the illusion of a closely unified timeline of music and image, even though one is not completely in control of the exact unfolding. This is the biggest challenge in dynamic music, both in composition and in implementation. In games, player motivation and skill level have a major role in the duration of a game session. In this sense, the music and the game have to be viewed as two separate processes. The technical implementation of a dynamic music system is the creation of a meditator between these processes, where this mediator must translate states of one process (the game) into states within the other process (the music). Most importantly, this mediator is also expected to account for considerations and constraints in both processes in order to maintain perceived coherence in both.

The temporal components from the activity based framework make the distinction between events and closures. Referencing philosophical discussions of time perception, Kaae highlights the separation of *events* and *processes*, illustrating their meaning through the analogy of travel between two points, where the departure and the arrival are separate *events* that together form a *process*. One should take note of one inherent similarity found in the separations between discrete *moments* in time, and *courses* of time. This separation also holds relevance when breaking down music. Discrete events could be things like notes, instrumentation changes and percussive hits. Processes are when these events group together to form meaningful segments like rhythms, melodies and form. Bridging back to events and closures in games, a similar kind of relationship is seen here, as closures and sub-closures are groups of game events that together form meaning. Identifying these analogies does aid in coming up with strategies in how to mediate between music and game, but we are not yet discussing how such a mediator would maintain coherence in both processes. If one directly maps game events to music events, there is no inherent guarantee that a meaningful closure in the game would be a meaningful process in the music.

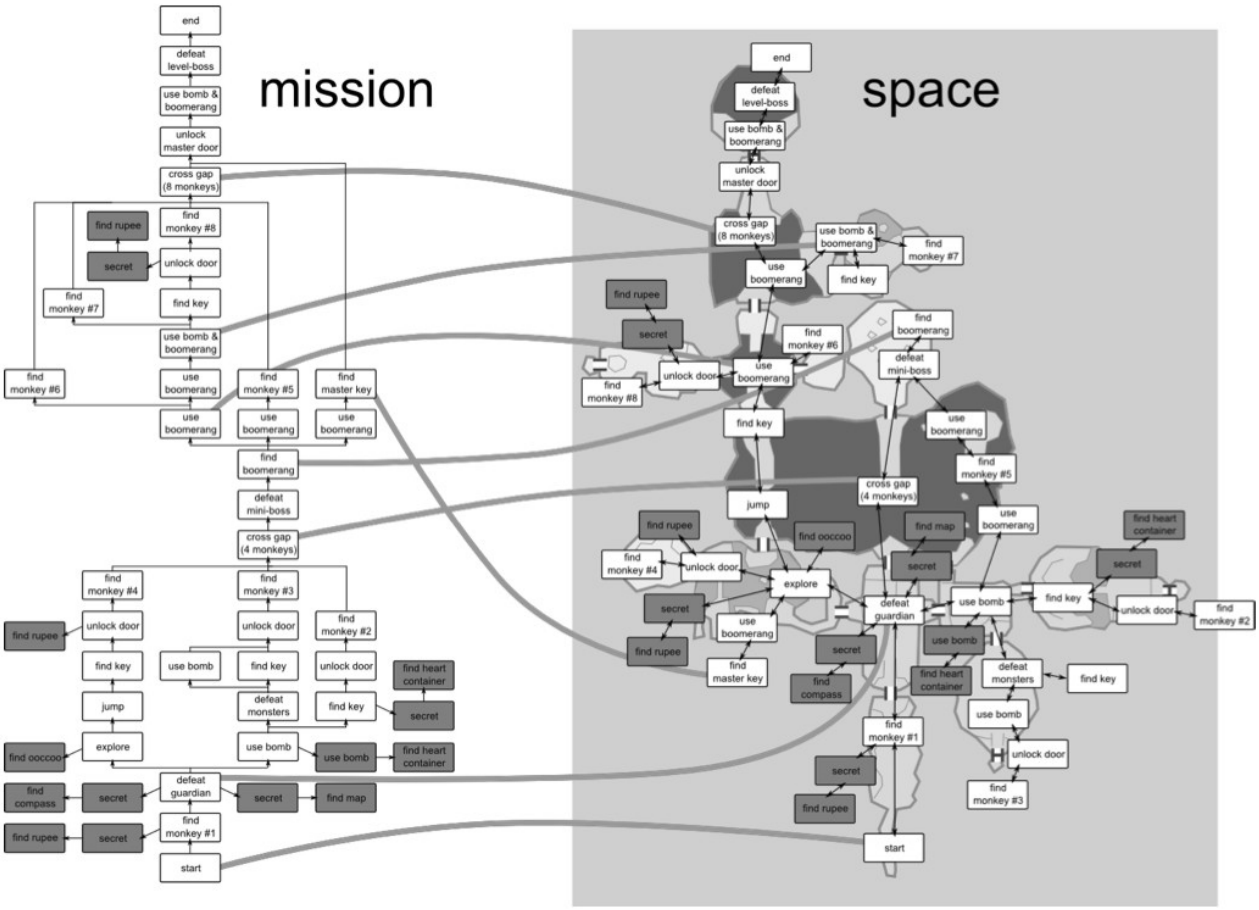
There is no “right” way of approaching this, as it is highly dependent on the designer's intentions for the resulting experience. A direct approach is identified *Guitar Hero*[4], where the gameplay is essentially pressing the right button at the right time, and these “right times” are

4 *Guitar Hero* is game series first published by *RedOctane* and *Harmonix Music Systems* and distributed by *Activision*. The first installment was released in 2005 for the *Sony Playstation 2*.

dictated by the music. Guitar notes can be said to “cause” game events, and the different song parts are both musical processes as well as closures in the game session. Here, however, music and game are more or less one process, as there is no disjoint in time between the two. Pausing a game of guitar hero freezes the image and stops the sound. This is an example of gameplay where an advanced mediator is not really required, as the music and game link is already inherent within the gameplay itself.

A more common scenario, however, are the cases where the gameplay is its own process entirely, and the music is expected to follow it. Kaae touches on some conceptual strategies and considerations. Kaae uses the term “hyperstructure of an adventure game”, by which he means the segmentation of an adventure game structure into nodes and links between them. This way of viewing a game is useful in many game research domains, the figure below, taken from Joris Dormans' paper on generating missions and levels in adventure games [5] being one example. In this particular case, it is a conceptual representation of a mission structure from *The Legend Of Zelda: Twilight Princess*[5].

5 *The Legend of Zelda: Twilight Princess* is a game by *Nintendo*, and was released in 2006 on the *Wii* and *Gamecube* consoles.



Playing the game is the equivalent of the traversal of this hyperstructure. The discrete transition from one node to another could be seen as an event, also signifying a closure (the completion of a node). Assigning a musical loop to each of these nodes and letting the events trigger a transition between these loops, is a very common strategy for realizing game music in general. Games often use unique looping musical pieces for each node, like in *Kingdom Hearts*[6], where a separate battle theme interrupts the "field music" loop as the player moves into combat. This is not really dynamic music in the way it is advocated, as the desired result in dynamic music is usually perceived musical continuity. One attempts to give the illusion of the music *changing*, rather than being *replaced*. However, as Kaae mentions, this conceptual strategy is also applied when implementing dynamic music. So the key difference here between dynamic and non-dynamic music in these cases are therefore not so reliant on the actual implementation, but rather on the actual musical content within these loops. This further highlights that dynamic music is just as much a compositional challenge as a technical one.

6 *Kingdom Hearts* is a game made by *Squaresoft* (later *Square Enix*) and *Disney*, and was released in 2002 for the *Sony Playstation 2*.

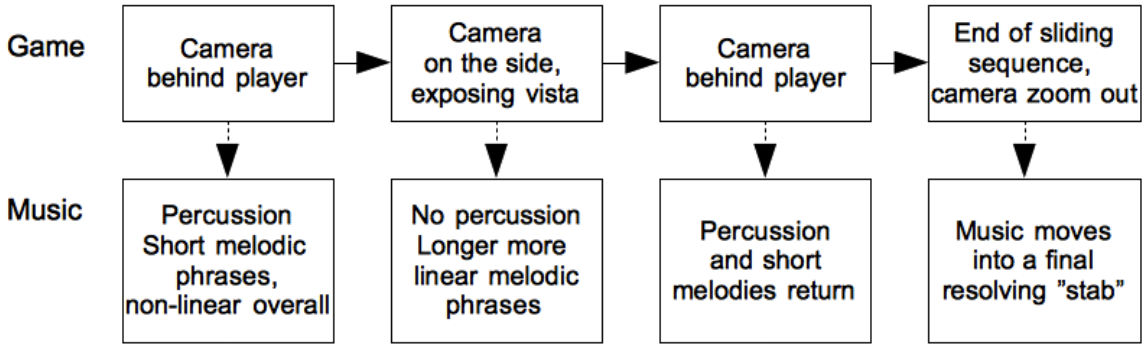
3.1.2 Linearity and non-linearity

When composing dynamic music, it is useful to look at some concepts that deal with how music is perceived. As discussed, music always occurs through time, and a compositional challenge comes in attempting to pause (metaphorically speaking) or prolong the music at any point for an indefinite amount of time. Referencing Kramer [6], Kaae outlines a distinction between *linear* and *non-linear* time in music, highlighting its relation to the mathematical concept of Markov chains. Markov chains are value sequences where any value is assumed to have a relation to its preceding values. In the case of a melody that adheres to popular western tonality, the probability of a note being of a certain pitch is dependent on the pitch of the preceding note. The *order* of the Markov chain is used to describe how many preceding values are taken into account when evaluating the current, and so it goes that "the higher the Markov order of the melody, the higher the *linearity"*. *Linearity* is the perception of music moving from a beginning towards an end. *Non-linearity*, in contrast, would be the perception of music metaphorically "standing still". A musical piece is not either or, but a combination of both these aspects. Persistent, overarching instrumentations and/or motifs can be said to be non-linear aspects of a musical piece while clearer musical progressions and forms are linear aspects. It must be made very clear that different listening habits and cultures also widely affect exactly how music is perceived in this regard. One example is found in several extreme musical genres, like *Grindcore*. For people with no prior listening experience in this genre, it is highly likely that it will be mostly perceived as a continuous wall of screaming and distortion (non-linear). Someone familiar with the genre will probably be able to distinguish the riff progression (linear) as they have adapted their way of listening through increased exposure. Therefore, there will never be an all-knowing function that can take an arbitrary sound as input and outputs a "linearity value". However, the conceptual difference between linear and non-linear turns out to be a useful aid when both analyzing and composing dynamic music for games, in spite of the absence of definitive and concrete solidifications. The tension between the moving and non-moving are often the kinds of tensions that one wishes to emphasize in dynamic game music. Like mentioned, if the player is standing still, the music is also expected to "stand still". Keywords like this, while abstract, do inform the creative composition.

In many current implementations of dynamic music, non-linearity is very often a prominent characteristic. In *Dead Space*[7], for example, one is never really exposed to any melodies (in the traditional sense). Instead the music consists of deep brass cluster chords and various drones building and relieving tension as the player progresses through the game. While the music is not

7 *Dead Space* is a action-horror game by *EA Redwood Shores* (later renamed to *Visceral Games*), and it was released in 2008 for *Sony PS3*, *XBOX360* and *Microsoft Windows*.

"catchy" in the sense that it is memorable or hummable (which usually is the case with linear music), it is perceived as responding to the changes in the game state while also being "one" continuos stretch of music. This does not mean that this implementation strategy in dynamic music must be void of melodies and/or other linear properties. A sudden highly linear segment could be used as a an effective tool to contrast. Also, one must keep in in mind that while the musical parts linked to the game's nodes may be highly non-linear when heard individually, the act of playing the game might arrange these segments in way that results in more linear properties. An example that illustrates this is found in *Journey*[8]. In the fourth level of the game, the player slides through the sand for the most part. The music that plays while sliding has a distinct percussive layer that contributes to the sensation of moving forward. Melody-wise it tends more towards the non-linear, as it doesn't offer long melodic stretches, but shorter melodic stabs. Near the end of the level the camera suddenly moves to view the avatar from the side, positioning the avatar between the camera and the sun, resulting in a unique vista caused by the striking way which the sand reflects the light. In this instant, the music looses its percussive layer, and longer notes and chords replace the shorter melodic stabs. When this sequence ends, the camera moves back behind the avatar, and the previous music resumes. The avatar then slides over a cliff, where the music builds up to a final stab, signifying the end of the sequence. The figure below uses a hyperstructure to illustrate this sequence conceptually.



8 *Journey* is a game made by *ThatGameCompany* and was released for Sony PS3 in 2012.

Screenshot from the unique vista in Journey's fourth level.

3.1.3 Quantization

While some compositional strategies have been identified in relation to composing the various musical segments, a challenge still remains in solving the actual transitions between them. Crossfading is an easy way to achieve this, but this can hold a risk of disrupting the perceived continuity of the music, as one can't be sure of at exactly which point in time the crossfade will take place. Collins mentions a strategy where transitions are implemented so that they snap to predefined *cue points* within the music segments. This is a key concept in dynamic music implementations, and is termed *quantization.* This term is actually for the most part mostly used in commercial DAW software to signify the placement of notes on a defined grid. Pichlmair and Kayali [7] also touch upon this term, highlighting its main function to be a way of making the resulting music sound "correct". Looking back at how dynamic music implementations are mediators between game and music, this is a practical example of how a mediator can cater to musical considerations. There is, however, a very important consequence that occurs when applying quantization in realtime. Quantization becomes in practice the *controlled delay* of a discrete musical event. A game event occurs, informing the meditator which in turn will trigger the associated musical event at the next available cue point, resulting in a delay between the game event and the musical event. This has the vital implication that any quantized audio change will always be "late" in relation to the game event that triggered it. As Kaae points out, the music implementation can't know when an event occurs before it occurs, so when using any form of quantization from game events to musical events, there will always be a delay. Quoting film audio theorist Jeff Rona, Kaae points out that this could be problematic when attempting to offer a natural perceptual correlation between the event and the sound that resulted from it.

One of the first implementations of this technique, is found in LucasArts' *iMuse* [8] which

became an integrated part of their engine for point and click adventure games in the 1990s. While its exact workings are proprietary, the key feature is identified in how the system transitions between a large collection of short musical segments at defined cue points. Exactly which segments are chosen, are defined by the current game state. Later games, such as *Tomb Raider Legend*, also use a similar audio implementation. This particular composition technique is termed *microscoring*[9], a name it gets from the fact that composers are essentially creating small puzzle pieces that are pieced together during the course of a game session. There is a compositional challenge found here, especially in regards to finding the right balance between long and short segments. For the quantization delay to be less noticeable, segments are required to be of very short duration, but this does also impose a challenge if the composer aims to include longer melodies and passages.

3.1.4 Discrete and continuos changes

Kaae briefly highlights a distinction between *discrete* and *continuous* changes in music. Discrete changes are essentially the same as what has been previously referred to as musical events – a moment in time when something occurs in the music. Continuos changes, however, has not yet been tackled, at least not in the sense that Kaae exemplifies it. While one could draw an analogy between the temporal terminology of a process, this would in my opinion be taking it slightly too far. While musical processes can indeed be created through continuous changes, this does not mean that continuous changes are necessarily processes themselves. A more straightforward approach is to view continuous changes as simply a stream of value changes. These could occur at random or in more controlled ways. The steady increase of one sounds amplitude is an example of a continuous change, but so is a random fluctuation in a cutoff frequency.

Discrete and continuous changes are useful terminologies as they further assist in bridging games and music. As Kaae mentions, a game engine will report the game state through discrete and continuous changes. A discrete change would be the kinds of changes we have already covered, where for example the player would enter a new node in the hyperstructure. Continuous changes are mostly values in the game that rapidly update, like the player's distance to an enemy, or the total playtime. A dynamic music implementation can make use of these values in various ways. Some games separate the music into *stems*, meaning different layers that can be combined in numerous ways. Instead of having the music change between predefined segments, like previously shown, one can switch between different sets of stems in response to the game state. An example of this can be

9 Troels Brun Folmann describes his process of composing for *Tomb Raider Legend* (Crystal Dynamics, 2006) as *microscoring*. Full interview is found here: http://createdigitalmusic.com/2006/10/cdm-interview-tomb-raider-legend-composer-troels-brun-folmann-on-adaptive-micro-scoring/

found in the recent *Final Fantasy XIII-2*[10]. In each area of the game, a musical piece plays in a loop, but whenever enemies spawn, extra stems are faded in, adding more overall intensity. The appearance of enemies is a discrete change in the game state that triggers the musical process of fading in stems. Another example is the implementation in *Red Dead Redemption*[11], where the entire in-game score is composed of stems that are in A-minor with a tempo of 130 bpm[12]. While this is admittedly a strict limitation on the compositional side, it ensures that any of the stems can be combined arbitrarily in accordance with the game state without having to worry too much about disrupting the musical coherence. Here the player's distance to various hotspots is a continuous change that factors into the balance between the stems.

3.2 Tighter relations between game and music

So far, dynamic music has been discussed with the assumption that it is expected to be implemented in ways that closely mirror the game-state. This is also the most common way dynamic music is approached. The following section will touch on another way of approaching the interplay between game and music, advocating a deeper relationship. This relationship is also the kind that will be further studied in this thesis. First, however, I would like to direct attention to sound diegesis in games.

3.2.1 Diegesis in games

Categorizing sounds as *diegetic* or *non-diegetic* sounds is a separation widely used when analyzing audio in films. The separation is one that attempts to establish how the sound correlates with image, in the sense of defining where the sounds "belong". Quickly summarized, *diegetic* sounds are recognized as belonging to the world depicted in the film, while the *non-diegetic* sounds are sounds that usually complement, but do not belong physically to the depicted world. The sound of footsteps played in correlation with a character walking would be *diegetic*, while background music with no recognizable physical on-screen source would be *non-diegetic*. Diegesis becomes problematic in some cases, however, where the separation is not so clear-cut. Curtis [9] discusses how many early cartoons offer cases where the music is composed to emphasize the films narrative progression on such a detailed level that it borders on being actual diegetic sounds. Curtis proposes using the terms *isomorphic* and *iconic* when studying relationships between sounds and visuals in animated films. *Isomorphic* relations happen when sound has the same "shape" as a visual movement. Curtis uses

10 *Final Fantasy XIII-2* is a game by *Square Enix*. It was released in 2012 for the *Sony PS3* and *XBOX 360*.
11 This information is taken from this video clip, detailing the making of the soundtrack of *Red Dead Redemption*: http://www.youtube.com/watch?v=klkWMkrO0pg
12 In music, *BPM* is an abbreviation for *Beats Per Minute*, a unit used for measuring music tempo.

familiar conventions as examples, one being the case where "(...) a glissando[13] accompanies a "pan" across an animated landscape". Briefly summarized, it deals with emphasizing a motion with sounds and/or music. *Iconic* relations pertain to analogous relationships between visual events and the timbre, volume, pitch and tone of the accompanying sound. In Curtis' own words, "the relationship between the sound effect and its visual representation is not one of fidelity, but of analogy". In essence, these terminologies deal with perceived relations between sounds and visuals. Looking back at what dynamic music implementations attempt to achieve, it can be said that it is indeed the pursuit of an isomorphic relationship between music and game.

Sound diegesis has also been used in game audio analyses. When analyzing the acoustic ecology of first person shooters, Grimshaw [10] proposes a definition of sound diegesis in relation to games, stating that "*(...) diegetic sound in a digital game be defined as the sound that emanates from the gameplay environment, objects and characters and that is defined by that environment, those objects and characters*". This has an interesting consequence, and highlights a very important difference between game and film audio. While Curtis' terms deal with level of image and sound correlation on the perceptual level, Grimshaws definition is essentially, at its core, a technical causal correlation. When game elements perform an action, that action *causes* the assigned sound, regardless if that sound has an iconic or an isomorphic relationship with the action. The game logic has defined this sound as emanating from that particular action. If firing a gun makes a trumpet sound, that trumpet sound will be diegetic to the game. In films it does not work this way, as there is no real defined causality between audio and image, and diegesis is only achieved through a perceived and somewhat believable (or culturally acceptable) correlation. In games, striving for iconic or isomorphic relations in diegetic sounds therefore remains a purely aesthetic choice. I will now look more at how diegesis relates to music in games that offer tighter couplings between music and game-state.

3.2.2 The composition-instrument

Norbert Herber identifies some key takeaways from audio implementations in both existing art installations and games, applying the *composition-instrument* framework to describe and conceptualize some of their shared properties [11]. The framework is described through two components, one being the ongoing music system and the other being the user interface. As the name implies, composition-instrument is used for systems that embody properties from both compositions *and* instruments. The core property lies in how the link between the audio generation

13 A glissando is a rapid succession of notes moving upward and/or downward, and is typically achieved by sliding one or more fingers rapidly over the keys of a piano or strings of a harp.

system and the user interface is realized. The audio should have some agency over the user interface, and the user interface should in turn have some agency over the audio, creating a feedback loop that effectively results in "*(...) a work that can play and be played simultaneously*". So while the framework separates user interface and audio, the goal is to implement them in such a way that the system can be regarded as one. Herber uses the term *resistance* to describe the connection complexity between the interface and the audio system. In contrast to the previous examples of dynamic music, it is not necessarily advocated that user input should have an immediate discernible effect on the generated music, but rather influence parts of the system that over time will propagate through the entire system. A particularly interesting point is made in the statement that by introducing a delay between input and output, the participants are more likely to listen carefully and engagingly in an attempt to identify what sort of repercussions their actions have on the sound.

When applying the composition-instrument to games, it becomes evident that one must approach dynamic game music with a different mentality than what has been previously discussed. In contrast with previous mentioned approaches, the music is expected to enter the foreground in a way that both the game-state and the music appear equally important. Though Herber does not mention it, it is very relevant to bring the concept of diegesis into this context. A game where many of the musical elements are diegetic to the game, will likely aid the resulting experience in adhering to the ideals of the composition-instrument.

One example of this, also mentioned by Herber, is *Electroplankton*[14] for the Nintendo DS. In this game, players can create and manipulate so-called "plankton" through use of the Nintendo DS' stylus. Depending on which mode chosen, the plankton will behave in different ways. The plankton emits various sounds and melodies as the player places them in the environment. Pichlmair and Kayali uses the term "sound agents" to describe game elements that contribute to the musical score in this way. Through continued player interaction, more sounds and more melodies combine into a larger musical piece. In this case, the music is diegetic, as each of its parts are individually caused by game elements. In Herber's own words, *"interactions with the plankton turn the Nintendo DS into an instrument that can be played purposely through the manipulation of the on-screen animations"*.

A more recent example of a game employing concepts from the composition-instrument, is *Proteus*[15]. The player begins the game on a generated island with a distinct, low-fi and pixelated aesthetic. The audio supplementing the experience is generated to match what the player is

14 *Electroplankton* is a game created by *Toshio Iwai*, and it was published by *Nintendo* in 2005 for the *Nintendo DS*.
15 *Proteus* is a game by *Ed Key* and *David Kanaga*, and was released in 2013 for *Mac OSX* and *Windows* through the Steam platform. (Official website: http://www.visitproteus.com)

currently seeing. The music is highly non-linear, consisting mostly of long-held chords supplemented with various simple waveforms fading in and out and gradually changing pitch. Advancing through the world introduces changes to the non-linear music layer, but identifying exactly how, and more interestingly, why, is not immediately clear. This helps encourage exploration as one is actively listening for more general overall changes, being an example of *resistance* in use. The player can also explicitly cause sounds to occur, for example by moving close to a "rabbit". When the player is close enough, the rabbit jumps away, emitting a special note on each jump. The note can be said to be diegetic to the game, as well as belonging to the music.

Both *Electroplankton* and *Proteus* both suffer from speculations surrounding whether or not they can be called games at all[16]. Upon closer examination of their boundary components, there are no defined goals, as no game state is more favorable than another. Whatever motivation that fuels player interaction, they are not explicitly defined in the game state. This discussion will not be pursued further, as it is a different field altogether. I would like to note, however, that one motivation in this study is to maintain clearly defined boundary components while still exploring tighter couplings between music and gameplay. I therefore propose exploring and expanding the concepts of the composition-instrument a little bit further.

3.2.3 Bidirectional relationships

Herber mostly focuses on a tight isomorphic relationship between interface and music, and highlights the exploration aspect when explaining the composition-instrument. Most of his examples are setups where musical changes slowly propagate as game elements are altered. However, he does not touch upon the possible repercussions the composition-instrument can have on gameplay on a purely mechanical level. Recapping, the composition-instrument is conceptually "*(...) a work that can play and be played simultaneously*". Looking back at how dynamic music implementations traditionally translate discrete and continuous game changes into musical changes, the composition-instrument here suggests the presence of the opposite; the translation of musical changes into changes in the game state. Like with sound diegesis in games, this can be reduced to strictly a mechanical relationship. In Herber's portrayal of the composition-instrument, the focus is directed isomorphic relations combined with interactive audiovisual exploration, and these will still be the traits associated with the composition-instrument in moving forward. To offer a clean separation of the purely mechanical implications of the composition-instrument, I propose the introduction of the term *bidirectional relationships* between gameplay and music. Whereas the

16 The following is a link to one of these discussions: http://www.gamasutra.com/view/news/185645/Is_Proteus_a_game__and_if_not_who_cares.php

composition-instrument describes the aesthetic experience, bidirectional relationship describes the concrete and mechanical.

4. GAME ANALYSES

Before moving on to describing the tools and the software created for this study, two games will be analyzed using some of the concepts introduced. These games are *Otocky*[17] and *Sound Shapes*[18], and they have purposefully been chosen because of their implementation of bidirectional relationships between gameplay and music. This chapter also serves to show how the terminologies can actively be applied, and to give a better overview of what has been done when dealing with more experimental takes on music and gameplay in the industry. The primary intention is to uncover potential traits and game design patterns in these kinds of games that will aid in contextualizing the custom created demo in this thesis.

When analyzing the games, I will first offer a general overview of the game. Then the music implementation will be briefly described, before looking closer at the how the game state influences the music and then how the music influences the game state. When studying the music, I will to some extent use linearity and non-linearity to give a sense of how the game state influences the perception of the music, occasionally touching on isomorphic relations. Take note that this will hold some degree of bias, given the fact that my own perception is used as a starting point. The composition-instrument will also be used in identifying traits, particularly in the usage of resistance.

4.1 Otocky

Otocky is an intriguing game from 1987. As it was only released for the 8-bit Nintendo Famicom Disk System in Japan, it is not a widely known game among european and american gamers. This is unfortunate because *Otocky* harbors a large pool of ideas in its bidirectional music implementation. Many of these ideas could benefit from being revisited in our time, as some aspects of the game do not come to full fruition due to the hardware it is on.

17 *Otocky* was developed by Sedic and published by ASCII Corporation in 1987 for the Nintendo Famicom Disk System. The game's creator is actually *Toshio Iwai,* the same person behind the previously mentioned *Elektroplankton*.

18 Sound Shapes is by Quesy Games, and it was released 2012 for the *Sony PS3* and *PS Vita*.
(Official website: http://www.soundshapesgame.com)

Otocky can loosely be described as a musical take on the *Shmup*[19] genre, where the player controls a flying robot-like avatar in an abstract world consisting of musical notation symbols and fish-like creatures. The available actions to the player are moving in all directions and firing a primary and a secondary weapon. The game is divided into levels, and the goal is to advance through each level by collecting sufficient amounts of musical notation symbols. Upon collecting enough notes, the level will dissolve, and the player must defeat an end boss before advancing. The primary weapon plays an integral part in the game, as it is used to collect musical notes as well as destroy enemies. When hit by an enemy, the size of the primary weapon projectiles is reduced and some of the collected notes are lost. The level ends in defeat if the player is hit sufficient amount of times. If the end boss is defeated, the player will advance to the next level. Completing a level will also unlock other, free-roaming modes to play the level in.

4.1.1 Music implementation

The unique thing about *Otocky*'s implementation is found in its emphasis on the player creating music. The player can fire the primary weapon in eight different directions, and each of these have a musical note mapped to it, resulting in the voluntarily or involuntarily creation of melodies. Viewing the music as being comprised of two layers, one layer functions as the accompanying while the other is the player created. The actual note values mapped to the eight shot directions switches in tandem with chord changes in the accompanying layer, ensuring an overall consonance[20] in the music as it is created. Some collectible orbs in the game world also affect the accompanying layer. One orb type increases or decreases the overall tempo of the music. Another orb stops the music and all game elements, with the exception of the avatar and the player's ability to fire the primary weapon. The most interesting orb type is the “record” orb, which records player actions up until a specific cue point in the music before playing back the sequence. While the note and shot sequence are played back, it is still possible to fire new shots, allowing two shots and two player-created notes to be present at once. In addition to the normal game mode, the player can play previously cleared levels in *BGM* mode and *music maker* mode. In BGM mode, no enemies will appear resulting in the player being able to freely choose which notes to play. In music maker mode, the player builds up the whole track, choosing how each instrument plays. This mode also is also offers a detailed edit screen where notes can freely be removed, changed or added.

19 *Shmup* is an abbreviation for *shoot-em-up*, a game genre where a lone avatar is tasked with shooting large numbers of enemies while dodging their attacks.

20 *Consonance* is the opposite of *dissonance*, and while one often talks about music moving between consonance and dissonance, overall consonant music is in this context used about music that sounds “right” in terms of western tonal traditions and conventions.

4.1.2 Game-state influence on music

There is a low degree of resistance when interacting with Otocky's music implementation. The musical result from firing the primary weapon is immediate and short, and does not initiate any propagations of the kind encouraged by the composition-instrument. Recording oneself through picking up a “recording” orb however, does have the potential of both aesthetic musical and game-focused strategic exploration. Musically, one can attempt to plan for two voiced passages, bringing us closer to the composition-instrument. In Otocky, however, this feature is severely handicapped due to the restrictions of the hardware. The game can only show two shots at once, and because of this, only the notes played on each 4^{th} beat are recorded, even though the game has the possibility of shooting every 8^{th}. As the game system here must heavily alter the player-recorded material, the sensation of music creation and exploration is limited. There is not a great deal of isomorphism present, as the actual note values have no metaphorical correlation with the direction or speed of the shot. It is interesting to note that in spite of this, it becomes clear very early on that the notes are caused by firing the primary weapon, making this a good example of the game phenomenon of diegesis through causality. Isomorphism is, however, present on a more macro scale, at least in the first level. As mentioned in earlier chapters, one of the most prominent motivations for implementing dynamic music in games comes from wanting the music to closely complement the action. Each level in Otocky is played like an infinite loop which is only broken when enough notes are collected. Where the collected notes used to be, enemies will appear instead, making the level become progressively harder. In the first level, this is also mirrored in the music, because the accompanying layer changes both chord progression, tempo and key as the player collects more notes. This greatly contributes to some degree of isomorphism as well as making the resulting music more linear. In the latter levels, this strategy is abandoned, making the first level stand out as the most isomorphic.

The music in Otocky is for the most part not very memorable. The accompanying layer mostly consists of fairly standard chord progressions. I would claim that Otocky's music is very non-linear, although the player's play style can play a part in this. Melodies are for the most part created by shooting, which will mostly be done with the motivation of completing a level. The pitch and temporal placement of player notes is therefore likely to be very influenced by the level layout, and these layouts are not arranged in ways that result in linear melodies. Typically, the player would use the primary weapon in many directions, resulting in a sequence of harmonically correct, but still linearly unrelated notes. From level two and onwards, though, the accompanying layer contains some voices that inhabit a greater degree of linearity, and these do usually to some extent overshadow the non-linear contributions from the player. In these cases, the player melody serves

more as a small backing to the overall musical context.

4.1.3 Music influence on game state

Seen in relation to the actual gameplay, the musical matching of shots has many interesting consequences. Each shot, and therefore note, is quantized to the nearest 8th beat, meaning that there is almost always a delay between player input and the corresponding shot. This effects how most players will interact with the game, as it is very unlikely that a player is precisely on the beat when pressing the fire button. In order to deliberately hit a beat, the player must learn to press slightly earlier than the beat, but since enemies and other targets usually move pretty fast, a more likely scenario is players rapidly pressing the fire button in hoping that at least one shot will fire before the opportunity is missed. As mentioned, only two shots of the primary weapon can be on the screen at once. There are several primary shots types, where the only difference is their range. Technically though, this means that longer ranged weapons are on the screen longer effectively preventing the creation of additional shots, and causing further delays between player input and actual shot instantiation. Picking up orbs that alter the speed of the music affects the game by making the delay between player input and shot instantiation smaller or larger depending on whether the speed is increased or decreased. This is an interesting effect, because increasing the tempo enables rapid fire, essentially creating another mode of play. Self-recording adds a potential strategic layer in allowing players to plan for future segments while recording. The player might want to cover as much of the screen as possible with shots, and this can be achieved by remembering the recorded directions. This is probably the best example of a bidirectional relationship between music and game in Otocky's music implementation.

While Otocky must be given credit for presenting these ideas so early in the history of console gaming, it is evident that the technical limitations imposed on Otocky do prevent some of its ideas to fully flourish. But there lies great potential in many of them, the tempo altering and recording being personal favorites. The shot quantization, however, while ensuring rhythmic coherence in the music, do impose some counter-intuitive interaction patterns for the player, at least in the way it is implemented here.

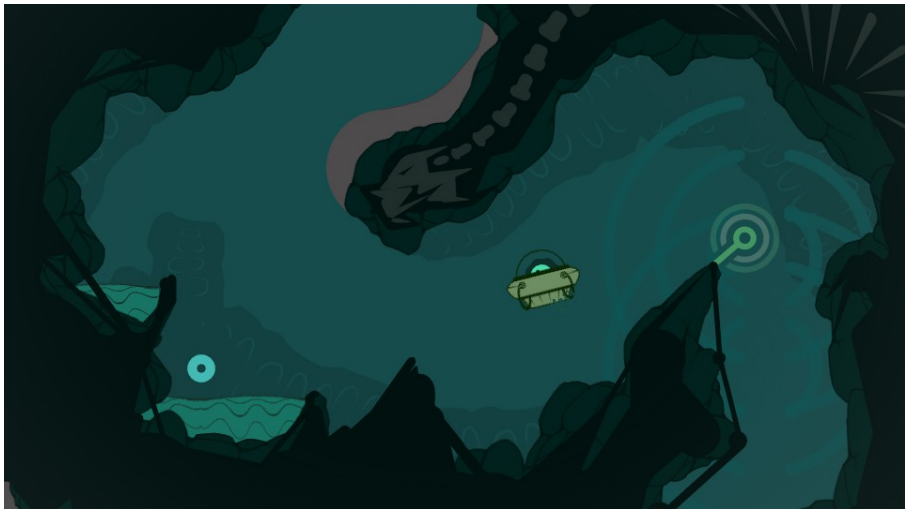
4.2 Sound Shapes

Before *Sound Shapes*' main menu appears, the player is greeted with the following message: *"Welcome! Sound Shapes is about playing and creating music"*. *Sound Shapes* is a two-dimensional side-scrolling platforming game where the player maneuvers a round blob-like avatar through many levels with different audiovisual aesthetics. Actions available to the player are moving, jumping and

transforming the avatar into a ball. The latter action serves as a switch between two modes of play, where each mode provides unique properties to the avatar. When in ball mode, the avatars speed greatly increases, while the regular mode allows the avatar to attach to any lightly colored surfaces. Some areas can only be completed by attaching to walls or ceilings, while other areas, like large gaps, can only be overcome using high-speed jumps and rolls. Much of Sound Shapes' gameplay therefore revolves around switching between these modes at the right moments.

The player will encounter a variety of game elements while traversing a level. Any red object, both moving and static, kill the avatar upon contact. When this happens, the avatar is forced to restart from the last touched checkpoint. There are also other kinds of game elements that help the avatar progress in various ways, like moving platforms and trampolines. Also, the player can collect circles that are scattered around the level. A level is completed by reaching the end (a cassette-player), at which point the player is awarded a total score. The evaluation function calculates this from the amount of circles collected and the amount time the player used to complete the level.

Sound Shapes also offers the possibility of level creation. After choosing the theme of the level, the player can place objects around to create a custom level. Beating the levels that come bundled with the game, will successively unlock new objects to use in the level editor.



4.2.1 Music implementation

The music implementation in Sound Shapes is created in a way that makes elements in the music diegetic to game elements. Levels are separated into screens, and the music played in each screen is decided by the placement of game elements. Some game elements are mapped to certain instruments, and imagining that the music is played from left to right on the screen, their physical placement on the screens x axis determines when in the bar that instrument will play notes. Each

level has an overall collection of instrument sequences, sounds and rhythms, and by advancing through the screens of level, various combinations of these are revealed. When the player collects a small circle, the circle will turn into a sound emitting object at that location. Collected circles from previous screens also follow into future screens, but are removed after two or three screens depending on the level.

Not all objects play their sound based on position. Enemies, in particular, have their musical part play at predefined intervals, which usually is linked to their attack. Some attacks are triggered in response to the player entering the enemies line of sight. In these cases, the player has direct influence over when the actual sound plays. All game events linked to game elements that are associated with a sound or instrument (apart from the player) are quantized to the beat.

4.2.2 Game-state influence on music

The actual music of Sound Shapes comes in many varieties. Established artists like *Deadmau5* are featured in the game, and each level has a unique track assigned to it. In essence, these tracks are mostly short repeating loops two or four bars long. These loops can be heard in full while selecting which level to play. It is through playing the level, that the music gets deconstructed into smaller pieces, and the tension between non-linearity and linearity becomes more complex.

After picking up a circle, its musical contribution will be added to the overall music. Before even hearing its musical contribution, the player will anticipate it. It is important to acknowledge that this happens through the playing of the game, and that this musical change could not have been anticipated in the same sense had one just listened to the music. While the musical process heard in isolation might be highly non-linear, the anticipation of expected change results in a special kind of linearity only made possible by combing the game and musical processes.

On the more macro level, the overall progression of the music, as the game is played, is completely decided by the level layout and the sequential order of screens. The music is always a direct result of the placement of game elements on the current screen plus the collected circles. Viewing a level as a hyperstructure of nodes, where each node is a screen, the resulting structure will both represent the level progression and its music progression. An interesting thing to note is how this essentially makes the actual level design largely dictate the balance between the music linearity and non-linearity. The screens could be aligned in such a way that more elements are sequentially introduced, effectively creating a form of musical crescendo. This is assuming a lot about the nature of the track, of course. In *Sound Shapes*' various levels, several strategies are pursued in this regard. Some levels are highly non-linear throughout, offering no real musical indication as to how close to the end the player is, while other levels have a clearer “direction” in

this sense.

Compared to Otocky, there is a greater presence of resistance in this music implementation and also more of an adherence to the composition-instrument. The mentioned anticipated change in the music already elevates the music into the foreground. One is not presented immediately with the audible result of picking up a circle, introducing a light degree of resistance. There are also many isomorphic and iconic relationships between the game events and their musical phrases. For example, a huge mechanical foot hits the ground, the sound of a thick snare sounds. In some cases, though, one has to devote some time to identify exactly which part of the music is causing what, and I would argue that this further increases the active listening and resistance.

Some factors of the implementation do push back the composition-instrument and musical exploration aspect somewhat. When the player is killed, all circles collected since the last checkpoint are lost, meaning that their associated musical contribution is also lost. This usually occurs within the boundaries of a screen, and this results in the initial incremental musical build experienced by gathering the circles being immediately undone. The next attempt at gathering these circles effectively looses some of that initial musical direction. Also, even though the game runs on the Playstation 3, the audio implementation also has some technical issues that interfere with the musical continuity. Upon re-spawning for example, the musical sequence "jumps" slightly, breaking the illusion of a continuous track.

4.2.3 Music influence on game state

The most notable effect that the music has on the gameplay is found in how the player must time the movement. Most game element behaviors are linked to events in the music, and the timing of these are usually defined in the music. In the screenshot below, for example, the "ahh" platforms appear as a direct result of an "ahh" voice in the music. To get past this screen, the player will first have to identify when in the music the "ahh" voice occurs, and the time the jumps in relation to this.

Looking strictly at the gameplay, this is the only major impact that the music has. While it

can be argued that the musical effect of collecting circles can largely effect how the game is played, the game rules also provide a motivation for collecting these by providing a higher score. It does seem likely, however, that the music has had great impact on how the official levels have been designed. Some of the tracks will undeniably work better when the various rhythmic sounds, melodic phrases and notes are placed in certain ways. It is therefore highly probable that this has been catered for when creation of the levels took place. So while this is not really a musical connection that occurs within a game session, it can be said that the gameplay, and then especially level design, has been impacted by the music.

4.3 Moving forward

I have up until now looked at some concepts in dynamic music and explored some concepts related to how deeper relations between game and music could be explored. With these concepts as backgrounds I have looked at two games and determined in what ways their music implementations affect the music and their gameplay. By closer study of *Sound Shapes* in particular, an important new aspect has emerged that I feel needs further solidification.

In chapter 3.2.1, I mentioned that dynamic music implementations are essentially mediators between the two processes game and music. These two processes each have their own considerations when maintaining coherence, and an implementations main challenge is identified as catering for these. When proposing the thoughts behind bidirectional relationships, I emphasized that this in essence a strict mechanical relationship. In Sound Shapes, game events occur exactly as they occur in the music, so the musical process is maintaining its own coherence while exercising direct control over the game state. Movement within the game occurs at the player's convenience, and not “on-rails” like in the previously mentioned *Guitar Hero*. What this means is that the current state of the music process *must* be considered in order to accurately describe the game state (on a mechanical level), since the music moves at it's own convenience while the game moves at the player's convenience.

Looking just at *Sound Shapes*, this might not seem like such a novelty. The only notable effect this has on gameplay is the introduction of timing, which is can hardly be called a breakthrough in game design. In *Otocky*, however, the player can exercise some degree of control over the music in ways that feed back (though admittedly modestly) into the gameplay. Potential emerges if one imagines Otocky's ideas combined with Sound Shapes' more elaborate music to game mapping. Interfering with a musical process that in turn dictates aspects of the game state is, to my knowledge, not a widely explored concept in current games. In moving forward, I will dive further into this using my custom software.

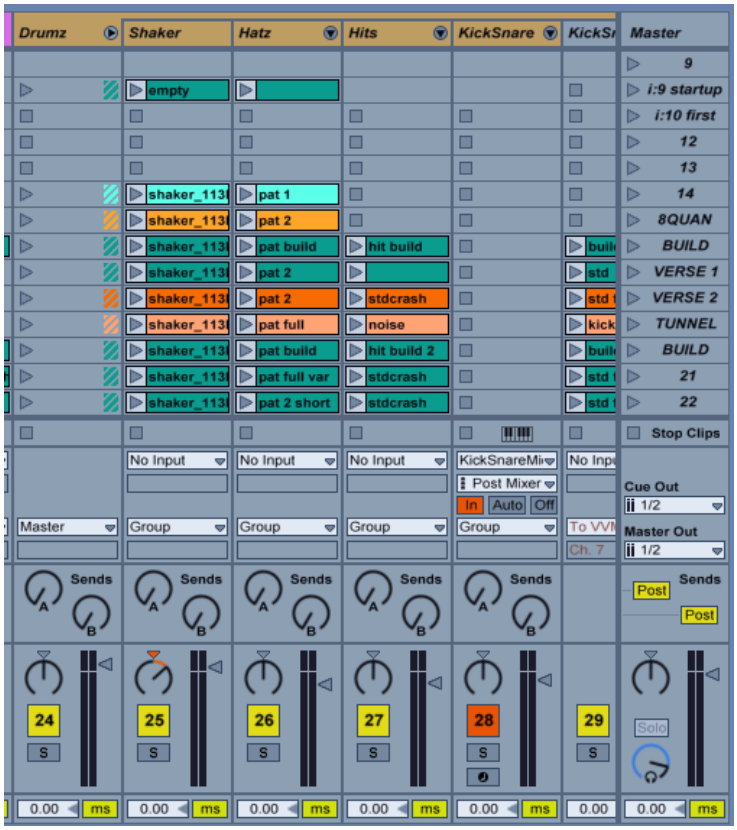
5. THE MAPPING FRAMEWORK

The mapping framework is the collection of software that together enable the exploration of bidirectional relations between music and games. *Ableton Live*[21] handles the actual audio output and composition workflow, while a custom application routes various signals between *Ableton Live* and the game *Audioverdrive,* specially designed for this study. Looking at how dynamic music implementations could be seen as the mediator between music and game, this is actually a physical representation of this – *Audioverdrive* being the game, *Ableton Live* being the music and the mapping application being the music implementation. This chapter will outline the technical details of the framework, and how one interfaces with it. Note that the described implementation is the first iteration and some of the choices have in hindsight proven to be less effective than intended. These issues will not be touched upon here, but will be integrated into the closing sections of the thesis.

5.1 Ableton Live

Ableton Live (from hereon referred to as *Live*) is, as mentioned in the introduction, a popular DAW software. Live is widely used among producers, DJs, and engineers, and stands out from its competitors due to its particular focus on live audio manipulation and customization. A most notable feature is the workflow mode called the *session view*, where the typical horizontal timeline view associated with DAW software is replaced by a matrix of cells, named *clips*. Clips are grouped into *scenes* and spread across individual *tracks*, where a *scene* is a row and a *track* is a column of clips. Tracks have an output (stereo or mono) that can be assigned a chain of DSP units, while clips contain the actual content played back to the track output. Only one clip can play per track at a time, and can either be launched individually or in groups. The screenshot on the next page shows an example of how a session view can be set up.

21 More information on Ableton Live can be found on the official website: http://www.ableton.com



The session view is especially tailored for live performances and improvisation. Each clip has a quantization parameter that decides when it will actually start playing after being launched, which allows for seamless and natural interchanging of various melodies and patterns. Out-of-the box, Live has many MIDI[22] routing and mapping possibilities, enabling Live to both control and be controlled by external MIDI-compatible hardware and software. In recent years, *Cycling 74*'s audio processing and interface creation tool *Max* has been integrated into *Live*, allowing custom creation of DSP effects, instruments and other tools that can both enhance and change the workflow. This feature, conveniently dubbed *Max for Live*, is not used in the current setup, but still worth mentioning to illustrate the notable focus on customization found in Live. Prior to *Max for Live*, the more advanced Live controllers were realized using the unofficial *LiveOSC*[23] plugin, a plugin that allows Live to send and receive OSC messages. This OSC API enables deeper control than the standard MIDI mapping possibilities, and is what I used in the implementation of the mapping framework.

Looking at some of the outlined strategies in dynamic music, one can quickly see how Live is a good candidate for handling the audio side in this kind of implementation. From a technical viewpoint, Live can be made to process and handle both discrete and continuous changes from a

22 *MIDI* is an abbreviation for *Musical Instrument Digital Interface*, and it is a communication standard introduced in 1983 that most digital instruments and music software supports.

23 More information on *LiveOSC* can be found here: http://livecontrol.q3f.org/ableton-liveapi/liveosc/

game engine and most of the previously highlighted strategies could be realized. A change in the game state can be mapped to launch a specific scene or clip, or the distance to an enemy could be mapped to control the volume of a particular track.

5.2 The mapping application

The mapping application is made to function as a communications hub between Live and the game, by running in parallel with Live. Through various OSC queries, the mapping application creates a tree structure mirroring the structure of the current Live project, where all the leaf nodes are mappable parameters. If the Live project is set up to explicitly send MIDI output to the mapping application (which will appear as a separate MIDI output device within Live), some basic midi messages will also be mappable. Any application that exposes itself as an OSC node on the same network will be selectable as a “game location”. Upon selecting a game location, the mapping system will query the ip address for game parameters, and should this be successful, all received game parameters will now be listed in the applications parameter browser. From the list, the composer selects which parameters to use in the current setup. The application will in turn inform the game about which parameters are used, to limit the amount of OSC traffic to only the bare necessities. In an ideal implementation, any game behavior that is informed by an input parameter should also have a default behavior in cases where the parameter is not used by the application.

5.3 A parameter

A parameter is technically speaking, just a float value, but the actual meaning and usage of that value can vary. It could be a scaling property only assuming values between 0.0 and 1.0, or it could be a numerical representation of a color or a game element. The previously outline separation between continuous and discrete is also applied in this case, and it is the frequency of updates that decides which group a parameter belongs to. A parameter that reports the player position in every update cycle, would be regarded as a continuous parameter, for example, whereas the instantiation of a game element or a game event would be a discrete parameter. In the actual source code, there is no significant functional difference between these two, and this distinction is only made to better inform the user about how the parameters should be used.

5.4 A mapping

A valid mapping consists of one or more input parameters and one target output parameter. The direction of the mapping can either be from Live to the Game or vice versa. Whenever one of the input parameters change, a function will execute, and it's result will be transmitted to the output

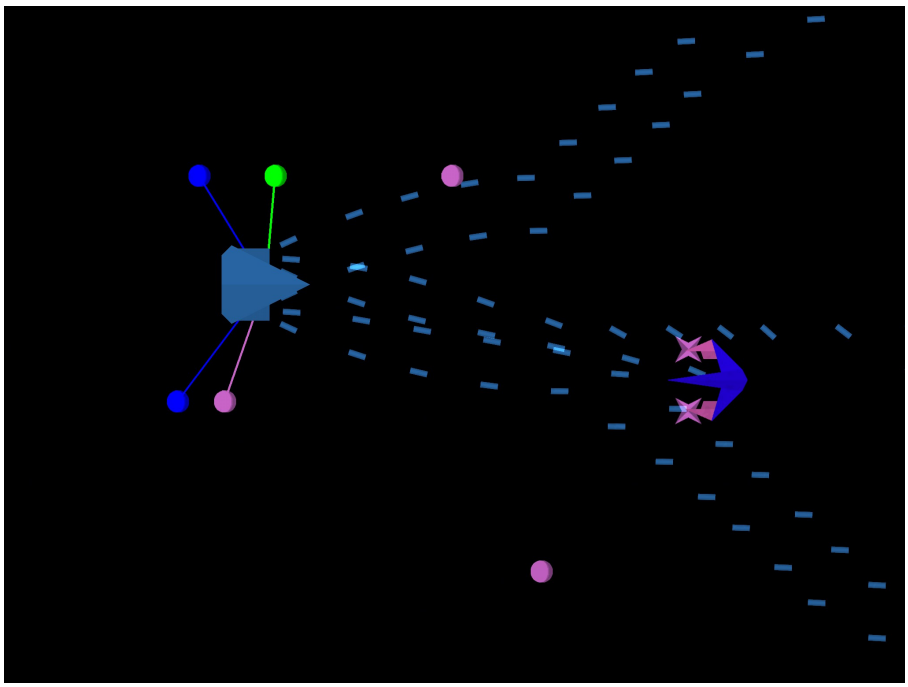
parameter. The function is a user-written javascript function where the current input values are passed as arguments. The javascript instance is shared across all mappings, allowing mappings to influence each other through user created global variables. While this system requires some programming knowledge from the composer, it also opens up the possibility of creating some larger high level rules. However, an output parameter will only transmit its value when triggered by an input parameter, so it is not possible to directly trigger the calculation of one mapping from another. In that sense, each mapping ultimately acts on its own. This was done with the intent of encouraging clean mapping approaches, having each mapping be separate processes.

5.5 Audioverdrive

The game in this study, *Audioverdrive* is developed for the iPad using the Cocos2D game engine. Live and the mapping software runs on an OS X computer, sending and receiving OSC messages to/from the iPad app over a wireless network. The game is inspired by *Otocky*'s own *shmup* genre, where my inspiration came mainly from games like *Gradius*[24] and *PixelJunk SideScroller*[25]. The player controls a spaceship through an automatically side-scrolling two dimensional world, attempting to avoid collision with enemies, enemy bullets and terrain. Also present in the world are weapon-modifying orbs that the player can either absorb or activate through proximity. The orbs come in four varieties: *power*, *spread*, *homing*, and *bounce*. When activated, an orb's property will be added to the ship's weapon, combining itself with any properties already added. Should the player choose to absorb an orb by tapping it, that orb's property will be added to the ship permanently---or until the player absorbs another orb, the ship is hit by a bullet, or the ship is destroyed. The screenshot on the next page is taken from the completed demo, and here the player is currently in proximity of four orbs, the two blue being *spread*, the green being *homing* and the purple being *power*.

24 *Gradius* is made by *Konami*, and is one of the most famous series in the *shmup* genre. The first game in the series was released in 1985 for many platforms, including the *Nintendo Entertainment System*.

25 *PixelJunk Sidescroller* is a game by *Q-Games*, and it was released in 2011 for the *Sony PS3*. (Official website: http://pixeljunk.jp/library/SideScroller/)



There is only one type of enemy in the game, but this enemy can appear in various versions. Each enemy uses two different weapon orbs infused, which greatly affects how their weapons behave. In addition to this, each enemy can assume three different modes: *regular*, *combined* and *charge* modes. An enemy in regular mode uses only one of the weapon orbs, but always fires two shots. An enemy in the combined mode will combine its two weapon orbs, but will only fire one shot at a time. Enemies in this mode will also attempt to align with the player on the y-axis before firing. Finally, the charge mode makes enemies disable their weapon and simply attempt to ram into the player.

With no OSC communication, the game will run in its "default" mode. The player can control the ship and pick up weapon orbs, but since no enemies are spawned (the default enemy spawn rate is zero), there is really no incentive for interaction. A complete list of all the parameters available to control the game with is found on the beginning of the next page.

Input parameters (*to* the game)

Player speed	Continuous
Enemy speed	Continuous
Terrain Speed	Continuous
Player Shoot	Discrete
Enemy Shoot	Discrete
Enemy Mode	Discrete
Enemy Spawn Rate	Continuous
Spawn Enemy	Discrete
Weapon Orb Type	Discrete
Weapon Orb Spawn Rate	Continuous
Spawn Weapon Orb	Discrete
Terrain Top Height	Continuous
Terrain Bottom Height	Continuous
Color None	Continuous
Color Spread	Continuous
Color Homing	Continuous
Color Power	Continuous
Color Bounce	Continuous
Color BG	Continuous
Color Terrain	Continuous
Game Victory	Discrete
Game Defeat	Discrete
Refresh current weapon descriptor	Discrete
Clear all on-screen weapon orbs	Discrete

Some of these parameters instantiate game elements in the world, while others change behaviors and some aesthetics. The intent was to expose sufficient parameters so that many varieties of gaming experiences would be realizable. Changes in some parameters are more immediately perceivable than others. Discrete parameters like *Spawn Enemy*, for example, creates a new enemy the second it is triggered. Changing the *Enemy Mode* will only have an immediate effect if there are enemies present at the time of change. The continuous parameter *Enemy Spawn Rate* controls the amount of seconds that pass between an automatic enemy creation, and a change in this parameter will only be perceived after the passing of some time, making it a less instant change in the game state. It is up to the composer in deciding how to map parameters from Live to the game, but some of the parameter differences do make certain types of configurations more suitable than others. Instantly perceivable changes might for example more suited to be triggered from clearly audible music events like the beginning of a note or the distinct playback of a unique sound.

Colors can also be changed through mappings, and there are mainly two reasons as to why I chose to enable this. One reason was to allow some aesthetic customization in order to broaden the range of musical styles that might be used. This can be seen in relation to *Sound Shapes*, where each

musical style has a specific color scheme and aesthetic tied to it. The second reason is that colors can also be used to provide other kinds of visual feedback. If one behavior is supposed to be favored over another, for instance, giving color feedback could aid in communicating this.

The victory and defeat conditions are also exposed, providing the composer with full control over the goal states of the game. Without any knowledge of the game state, however, creating end conditions is not possible. Below is a complete list of the output parameters reported from the game.

Output parameters (*from* the game)

Player X Position	Continuous
Player Y Position	Continuous
Player Deaths	Discrete
Total on-screen enemies	Discrete
Total destroyed enemies	Discrete
Number of spread connected	Discrete
Number of homing connected	Discrete
Number of power connected	Discrete
Number of bounce connected	Discrete
Currently absorbed orb	Discrete

This list is much shorter than the previous list depicting input parameters. The main reason for this is because these parameters only report values that are only influenced through player interaction. *Audioverdrive* relies mostly on external input to modulate parameters that create and modify game elements, so under the assumption that the mapping application already knows these values, the game does not report them back (something that also could result in infinite loops and deadlocks). It goes without saying that the intention is that input and output parameters are to be used together to create bidirectional setups. Since the actual editing is done by a composer through the usage of the mapping application and Live, this can yield very unusual results seen from a traditional game design perspective. Composers might be more likely to favor coherence and particular aesthetic qualities in the music, as opposed to designing from a gameplay-first perspective at all times. A musically interesting bass-line might be chosen in favor of a finely tuned difficulty curve, for instance.

6. THE DEMO

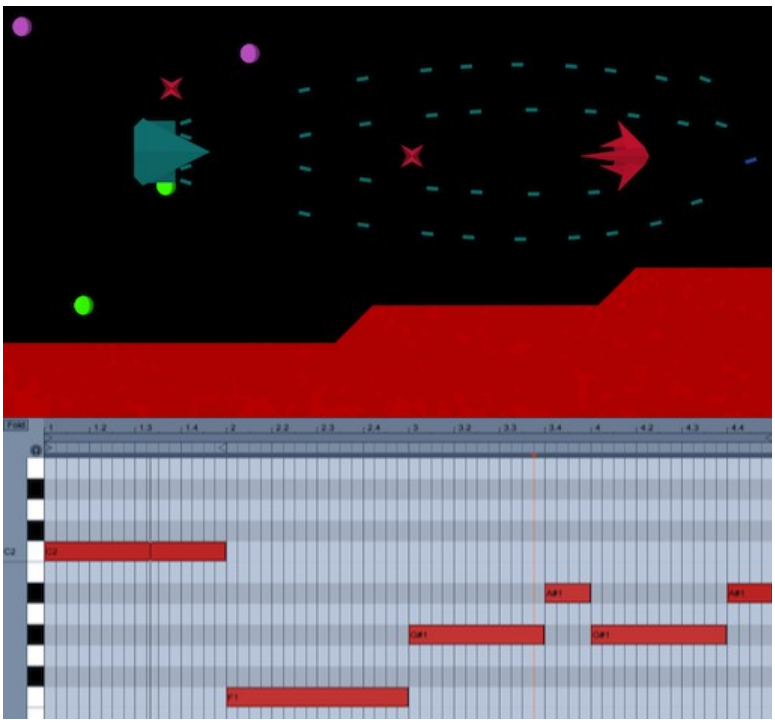
I have put together a mapping configuration for Audioverdrive, with the interest of showcasing some of its possibilities. The point of departure was a simple musical idea, an approach that seemed to resonate well with the idea of placing composition as key part of the game design. For me, this was a very unusual way to approach composition, as some musical choices had to be made as a result of gameplay considerations. While my approach to creating music is often to go for clean and simple patterns, I was inspired to go more overboard in some melodies for the purpose of creating more interesting effects in the game state. While the final result might be unclear to someone without full knowledge of how the mappings are set up, it does highlight a lot of the potential that lies within this sort of approach to game design.

6.1 Live-to-game mappings

For the demo, the music is an electronic track inspired by acts like *Daft Punk*[26] and *Lazerhawk*[27] while still retaining some of the musical touches I personally associate with shmup games. The mapping setup is fairly complex, and this section will only mention the instruments that directly influence the game state. Since imagining exactly what sound and instruments sound like is almost impossible through a textual description, examples of each of these instruments are also included in the "Audio Examples" folder on the CD that accompanies this thesis. The filenames will be shown in parentheses whenever the instrument is mentioned.

Enemy shots are triggered when a kick or a snare drum plays *(ae01-EnemyShots.wav)*, and all enemies currently on screen will attempt to crash into the player when a clap sample *(ae02-EnemyCharge.wav)* plays. A distorted sitar instrument *(ae03-EnemySpawn.wav)* is mapped to spawn enemies, and this usually happens at the beginning of four-bar periods, backed up with a crash cymbal. A plucking instrument *(ae04-PlayerShots.wav)* fires shots from the player, usually in rapid succession, as this instrument mainly plays quick arpeggios. The terrain height is controlled by the current note value of a deep pad sound (which will be referred to as the *terrain instrument* for convenience), and this instrument is played with two separate voices, where the lower voice controls the bottom height and the upper voice controls the top height of the terrain *(ae05-TerrainInstrument.wav)*. The mapping is made so that smaller intervals between the voices result in narrower passages in the game. Weapon orbs are spawned every time a sonar-sounding instrument (*ae06-WeaponOrbSpawn.wav*) plays , and the *y* position of the spawned orb is decided by the note pitch.

26 *Daft Punk*'s official website: http://www.daftpunk.com
27 *Lazerhawk*'s bandcamp page: http://lazerhawk.bandcamp.com



In this figure, the lower is taken from the terrain instrument clip currently looping when the upper image (game screenshot) was taken. The two last notes lead up to the first note in the loop (Ab-Bb-C) which is what just happened in the upper image, as can be seen in how the bottom terrain rises in steps.

6.2 Game-to-Live mappings

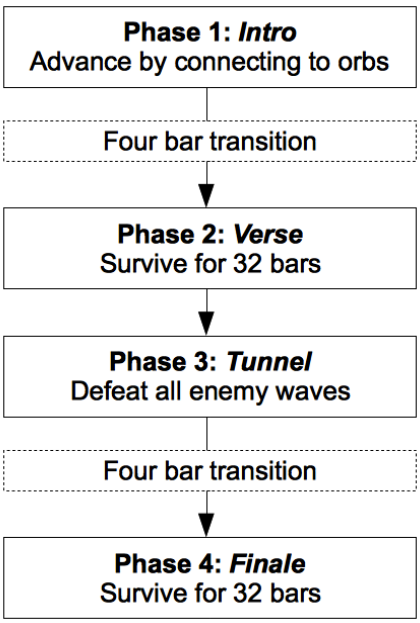
The game controls Live mainly through triggering clips and scenes in response to certain events that occur in the game state, mostly as a direct or indirect result of player actions. Since the various clips in turn instantiate and modify game elements through being played, the bidirectional relationship is realized here. As mentioned, Live offers a quantization option per clip, and this is used differently depending on the clips purpose. Whenever enemies are destroyed, an explosion clip is triggered along with a three note chord played with the orb-spawning instrument. The explosion clip has no quantization and will play immediately, but the chord is quantized to play at the next 16th beat. This was an aesthetic choice made with the intention of having the chord blend nicely and "on-time" with the music. One thing to note here is that due to the Live-to-game configuration, this chord will in turn instantiate weapon orbs. This is one example of how the framework can be used to modify the game's initial boundary components by creating new game rules. In this case, the instantiation of weapon orbs becomes a direct cause of destroying enemies.

Most of the remaining clip triggering is done when the player's current weapon orb collection changes. For this configuration, the orbs have been split into two groups, where the bounce orb is matched against the terrain instrument and the other orbs are matched against the player shot instrument. Exactly which clip that is launched is determined by which and how many

orbs are linked to the player. Picking up or connecting with different orbs therefore have different effects; bouncing orbs will indirectly change the terrain, while other orbs will modulate the player's shot patterns.

6.3 Composition of gameplay

The game is structured in four sequential phases, and this progression was achieved by letting phases influence which clips and scenes in Ableton Live (and therefore gameplay sequences) are triggered. The track is built around one short musical motif, with phases driving the musical progression built from this motif, roughly in ABAB form. The player's goal is to survive until the end, and this can only be achieved through the successful completion of each phases' challenge. Below is a figure that shows how the structure of the game experience.



The first phase starts with a very simple clip creating one bounce weapon orb in each fourth bar. Upon connecting with an orb, a terrain instrument clip is triggered. Connecting with more than one orb replaces the clip with more lively versions, resulting in more jagged terrain. The goal in this phase is to succeed in maintaining a connection of two or more orbs for a period of four bars each time. With each success, a crash sound will play while new instruments are gradually added and existing clips are replaced. This phase is meant to offer some exploration of the terrain generation, while also gradually introducing the main musical motif. The two final crash cymbals in this phase are also accompanied by the enemy-spawning instrument. At this point in the track, neither the player shot instrument nor the drums are playing, so no enemy or player shots are fired. However, a clap appears at the end of each four bar period whenever an enemy is present, causing the enemy to

charge for the player. When the player has succeeded in maintaining sufficiently long connections, the track automatically builds up and transitions into the next phase.

In phase two, the player needs to survive for 32 bars. At this point, the track has the full set of drum clips playing, resulting in enemies shooting regularly. Since the enemy spawn sound plays every fourth bar, the player must actively dodge their bullets while trying to connect with weapon orbs that activate the arpeggio clips, which in turn cause the player to fire shots back. Musically, this is where the track has really begun in full, with a steady bass pattern and drum beat being the main focus. Bounce orbs are omitted from the possible spawned orb types, so no terrain generation occurs in this phase. This was done with the intention of making this a clear contrast with the previous phase by making the space more open and focused on combat and bullets.

After 32 bars, the player will enter phase three, which I have dubbed the *tunnel phase*. Similar to the first phase, the music, and therefore also the game, will not advance unless the player completes a series of goals. Here, the player needs to destroy each incoming wave of enemies, and new waves will not present themselves until the current one has been dealt with. Each enemy wave is spawned with an increasingly elaborate melody being played by the enemy spawning instrument. Musically, this phase is a slightly more intense version of the motif played in the intro, with the terrain instrument being in the foreground. Bounce orbs are again made available, offering the player the possibility to influence the terrain generation again. When the final wave of enemies is dealt with, the track transitions into the last phase.

Musically, the final phase is a full realization of the second phase, and is intended to be the peak of the game experience. As in the second phase, the player must survive for 32 bars. The main difference is that the sonar instrument plays much more elaborately, making the screen home to many more weapon orbs. This is was done in an attempt to make the actual game experience match the climatic nature of the music, as the screen is likely to be filled elaborately with player and enemy bullets. As in the second phase, no bounce orbs are spawned, and therefore there is no terrain.

6.4 Examination

I will now look more closely at some of the aspects of the finished demo, and how it relates to the concepts of bidirectional relationships between music and game.

6.4.1 Game-state influence on music

In the first phase of the demo, the composition instrument has a noticeable presence. The experience begins in complete silence until the sonar instrument spawns a weapon orb, and from

here, the player must deduce the workings of the game through exploration. There are difference levels of immediacy in the player action's musical result. When the player initially connects to a bounce orb, the deep pad instrument and terrain appear close to immediately. Connecting to more bounce orbs make the terrain and melodies more elaborate, also an immediate effect, though maybe not completely apparent. Resistance is identified, however, in deducing the “right” way of progressing. While subtle clues like the increasing loudness of filtered noise does hint that maintaining connections will result in something, deducing this is in no way as immediate as the direct mappings of the previous. And *what* it results in does not become clear until the process has been repeated some times. Apart from terrain, there is also very little that can damage or destroy player at this point of the game, and with no prior knowledge of the system, the player might think the point is simply to explore. Having to maintaining connections goes a little bit against the free exploration, since one needs to limit movement to a large degree in order to maintain proximity.

Once all these relations are learned however, the same special linearity found in *Sound Shapes* emerges. Upon connecting to orbs, one expects the music to change, even though the music in itself might not hint to it. The orbs significance in this regard changes throughout the game, however. While the first phase requires maintained orb connections for the music and game to proceed, the orbs later provide musical *variation* rather than *direction*, at least in my opinion. But the anticipation of musical change is then achieved in other ways. In the third phase, the background music stays largely the same throughout the whole segment. Each enemy wave is spawned through an increasingly more elaborate melody played with the enemy-spawning instrument. Upon defeating an enemy wave, one then readily expects a variation of the music at the next bar.

Diegetic relationships are established mechanically as they occur, and there is little conscious use of isomorphic or iconic relations. Hearing the music by itself does not give a clear portrayal of what the various elements might signify. One iconic relation I identify, however, is the ominous sound of enemies spawning, a kind of convention established in early *Star Trek* movies[28]. On the macro level, there is a clear intended correlation between the overall track progression and the gameplay progression, going from minimalistic beginnings to a climatic finale.

6.4.2 Music influence on game state

The most interesting influence I see the music have o the game state comes from the fact that the music is its own process. These examples exist for the most part in the third phase. Depending on

28 In the first *Star Trek* movie from 1979, for example, these kinds of sounds where used when unknown/potentially dangerous ships appeared.

when the exact time the player chooses to connects to a bounce orb, the terrain generation will differ. In the third phase, this does create some situations where knowing the music can result in strategies when dealing with enemies. Invoking melodic passages that result in more jagged terrain increases the likelihood of enemies colliding with it, especially in the case of enemies that attempt to align with the player's y-position. However, the player will also have to dodge the terrain, so this must also be taken into account when deciding when and what melody to invoke.

Another example is found in timing when to defeat the enemy waves. The next enemy wave will always spawn at the very beginning of the next four-bar period and this means that timing becomes essential if the player aims to utilize all the orbs on screen. If the enemies are destroyed too early, the orbs may be gone before the next wave spawns, while if they are destroyed too late, the orbs might not reach the player at the time of need.

6.4.3 Closing remarks

This particular mapping configuration is the product of an interesting, but also unfamiliar workflow. While it was approached with an initial musical idea, the process had me going back and forth between gameplay considerations and musical considerations. It was never clear if the musical idea, its instruments and aesthetic actually was something that would work with the game. I will not categorize the result as a clean design, due to the presence of some obviously conflicting ideas. In the first phase, for example, connecting to orbs advance the level and the music, but then that aspect is completely abandoned later. It does however show that this tool functions as an effective way to try out, and identify strengths and flaws in various configurations of bidirectional relationships.

The space of possible configurations for *Audioverdrive* is very large, and no rules dictating overall gameplay progression are inherently defined in the game. This leaves the composer with a lot of work, and in the case of this demo, there is a very large and somewhat unmanageable amount of *if*-statements and global variables in the javascript. Also, for the progression logic within the first phase to work as intended, it was necessary to introduce a new input parameter during the creation of the demo that caused *Audioverdrive* to re-transmit the current orb count parameters, something I regard as a rather hacky way to deal with a workflow problem. Also, since each parameter is a single float, a very common operation was to pack and unpack values through bit operators. To keep this a more streamlined process, I created some hardcoded javascript functions that would take user-friendly arguments and pack them in the necessary ways.

These observations suggest some improvements that could be incorporated to make the application less focused on programming, and more on bidirectional music and game design. One idea might be to consider creating some generic gameplay progression templates, where the phase-

based progression in my demo could be one of them. To counter the need for custom javascript functions, a solution could be to enable the transmission of custom functions from the game to the mapping applications javascript instance. In the case of *Audioverdrive*, for example, this could greatly reduce the number of custom bitwise operations needed when packing and unpacking values.

7. CONCLUSION

This thesis has reviewed several concepts and terms that prove useful when dissecting dynamic game music implementations. Music and game has been described as two separate processes, and some of the consequences from various mediation strategies between these have been shown. The concept of bidirectional relationships has both been established as a term, and contributed in showing that quantifiable effects on the gameplay can occur through using it in implementation. With the intent of further exploring this concept, I built a mapping framework that allows parameters in *Ableton Live* to be linked to game parameters. To test this workflow, I designed the iPad game *Audioverdrive*, and created a demo configuration to showcase some of its possibilities.

From using the framework, I have found that this form of exploration is not without its challenges. The overabundance of possibilities makes it difficult to have a clear starting point as well as maintain an overview while working, mainly due to the large amount of scripting required to create a game progression. But while its current state is not perfect, it still shows applicability. In the area of sound perception, the framework provides an opportune way to explore the boundaries between perceived and learned relationships between image and sound in games. In the field of game and level design, interfacing the mapping framework with games of other genres could be a great way to identify exactly how diverse the space of bidirectional setups is. At the time of writing, I am the only one who has used this workflow, and so exposing it to more people would be likely to reveal more possibilities. It would be especially helpful to involve more (and maybe less technical minded) composers, as this would aid in evaluating the musical expressiveness of the framework. From there, one could attempt to establish potential differences between experiences created by composers versus experiences created by game designers. As a final statement, I feel this thesis shows that there lies a lot of potential in challenging some of the wide-spread assumptions regarding how music and game should be linked. There is a well of unexplored opportunity, and that this was one attempt at diving into that well.

ACKNOWLEDGEMENTS

I would like to thank my supervisors Mark Jason Nelson and Julian Togelius for their helpful and inspiring enthusiasm throughout the whole duration of this project. I would also like to give massive thanks to Ableton Live, VVOSC, LiveOSC, Cocos2D, Box2D and Apple Cocoa Framework. Without these tools and frameworks, it would not be possible to get the mapping framework in the working state it currently is.

BIBLIOGRAPHY

[1] S. Björk and J. Holopainen, *Patterns in Game Design.* Charles River Media, 2004.

[2] T. Van Geelen, *Realizing groundbreaking adaptive music*, In *From Pac-Man to Pop Music: Interactive Audio in Games and New Media*, K. Collins ed. Ashgate, 2008. 93 – 102.

[3] K. Collins, *An introduction to Participatory and Non-Linear Aspects of Video Games Audio*, In *Essays on Sound and Vision,* S. Hawkins and J. Richardson ed. Helsinki University Press, 2007.

[4] J. Kaae, *Theoretical Approaches to Composing dynamic music,* In *From Pac-Man to Pop Music: Interactive Audio in Games and New Media*, K. Collins ed. Ashgate, 2008. 75 – 91.

[5] J. Dormans, Adventures in Level Design: *Generating Missions and Spaces for Action Adventure Games*, In Proceedings of the 2010 Workshop on Procedural Content Generation in Games, 2010. Article no 1.

[6] J. D. Kramer, *The Time of Music.* Schirmer/Mosel Verlag GmbH, 1988.

[7] M. Pichlmair and F. Kayali, *Levels of Sound: On the Principles of Interactivity in Music Video Games*, In Proceedings of the 2007 Digital Games Research Association Conference, 2007. 424 – 430.

[8] M. Z. Land and P. N. McConnel, *Method and apparatus for dynamically composing music and sound effects using a computer entertainment system.* U.S. Patent 5,315,057, 1994.

[9] S. Curtis, *The Sound of the Early Warner Bros. Cartoons*, In *Sound Theory Sound Practice*, R. Altman ed. Routledge, London, 1992. 191 – 203.

[10] M. Grimshaw, *The acoustic ecology of the First-Person Shooter*, In *Games Computing and Creative Technologies: Dissertations*, 2007. Paper no 1.

2007

[11] N. Herber, *The Composition-Instrument,* In *From Pac-Man to Pop Music: Interactive Audio in Games and New Media*, K. Collins ed. Ashgate, 2008. 103 – 123.

SUPPLEMENTAL MATERIAL

The attached CD contains all the digital content created for this thesis. In the *Audio* folder, the complete Ableton Live project is included as well as the mapping configuration file *Demo.apmc* that must be opened using the mapping application. The *Audio Examples* folder contains the files referenced in chapter 6. The *Binaries* folder contains the compiled versions of the software, where *APMapper.app* is the mapping application and *Audioverdrive.ipa* is the game. Note that the compiled Audioverdrive app will not run on any other device than my development device or a simulator, due to the Apple requiring device permissions to be explicitly compiled into apps that are not distributed via the app store. Contact me on niih@itu.dk if you require it to run on your device, and I might be able to send you a version with your device data embedded. If you have your own Apple Development Certificate, you can compile the source code, which is found in the *Source Code* folder. A .pdf version of this document is found in the *Report* folder.