We describe a system for bidirectionally coupling the mu-
sic and gameplay of digital games, so gameplay proce-
durally varies the music, and music procedurally varies
game. Our goal is to inject game music more closely
into the core of gameplay, rather than having the music
serve as an aesthetic layer on top; traditionally, music re-
sponds to game state, but not vice versa. Such a coupling
has compositional and design implications: composing
game music becomes a kind of \emph{composition of gameplay},
and furthermore, game-design decisions feed back into
the music-composition process. We discuss an arcade-
style 2d side-scrolling game, AudioOverDrive, demonstrat-
ing this integrated music/game composition approach.

1. INTRODUCTION

Music is a key part of the culture and aesthetics of digital
games, so much so that it often spills out of games proper,
and into popular music culture. Games’ soundtracks form
a large part of their overarched and the cultural attach-
tment they engender is such that fans flock to see sym-
phonies perform the soundtracks\cite{6}. Meanwhile, the par-
cularistic aesthetic sound signatures of the sound chips
states in system like the Atari VCS have inspired modern-day
chiptune and bitpop musicians to repurpose the hardware
by experimenting with gameplay rather than matching
the influence is the other
dimension\cite{10,11}.-To include the
distinction is admittedly not a clean one\cite{10,11}, but we
find it productive here to start with existing game-
design- and game music composition practices, and exper-
iment with adding bidirectional coupling between the two.
It’s possible the result may converge nearer to electronic
art, especially in particular designs using feedback loops
musically intertwined with those common in cybernetic art.
But so far, in our own use of these experimental tools (see
Section 4), the result still feels much like game design and
music-game composition, albeit in a weirdly coupled way.

2. BACKGROUND

Our goal of architected the gameplay-music coupling as
a bidirectional feedback loop will no doubt sound famil-
lar to audio installation artists. In contrast to games’ lim-
ited experimentation with such feedback loops, feedback
between auditory layers and generative music systems (often
conceptualized as a form of cybernetic coupling) is a com-
mon strategy deployed and explored by interactive sound
installations. Therefore, in a sense our work can be seen as
part of a recent trend in experimental digital games, which
refines electronic art for ideas and design elements to
selectively borrow\cite{8}.

Despite intriguing connections to audio installations
and interactive electronic art more broadly, we see our-
selves as situated primarily in game design as a starting
point. The distinction is admittedly not a clean one\cite{10,11},
but we find it productive here to start with existing game-
design- and game music composition practices, and exper-
iment with adding bidirectional coupling between the two.
It’s possible the result may converge nearer to electronic
art, especially in particular designs using feedback loops
musically intertwined with those common in cybernetic art.
But so far, in our own use of these experimental tools (see
Section 4), the result still feels much like game design and
music-game composition, albeit in a weirdly coupled way.

2.1. The composition-instrument

The existing style of game design closest to our goal of
bidirectional coupling is probably the one theorized by
Herber\cite{7} as a \emph{composition-instrument}. In a composit-
ion-instrument, the player can “play” or “compose” music in
real-time while playing the game. In contrast to music-
matching games such as \emph{Guitar Hero} the player generates
(part of) the music through gameplay rather than matching
gameplay to pre-defined music.

A particularly intriguing example from 1987 stands out.
\emph{Otocky}.\footnote{2} A sidescrolling arcade shooter for the
Nintendo Famicom Disk System, it places players in con-
trol of a flying avatar that fires a short-range ball pro-
jectile in order to deal with enemies. Each of the eight
possible shooting directions has a musical note attached

2This style of gameplay-driven, state-based music transition was pi-
tioned by the LucasArts \textit{Muse} system in the early 1990s\cite{12},
and remains the dominant mode of game-music coupling\cite{1,3}.

Game engine for the iPhone and iPad platforms, with gen-
erative spaces of music composed in Ableton Live, a pop-
ular piece of digital audio workstation software aimed at
live performance, via a composer/designer-configurable set
of mappings. AudioOverDrive’s aesthetics are loosely
based on the first author’s experiences—previously sepa-
rate ones—as composer/keyboards for the game-music-
fluenced systems hand Redacted For Blind Review, and
designer of procedurally varying arcade games. One of
the goals in that regard is to produce an actually playable
version of the coupled gameplay-music experiences one
often finds in noticed music in this genre.

Figure 1. \emph{Otocky}, in which notes and instruments are tied
to weapons.

In addition to the experiments with dynamic and genera-
tive game audio already discussed, there has also, since
the early 1980s, been work on procedurally generating
game levels and other content. For example, classic games
like \emph{Civilization}, \emph{Rogue} and \emph{Elite} feature content that
is automatically generated rather than created by a human
designer. In some cases, levels are randomly generated,
while in other cases a human player or designer is given
some kind of control over the generated content\cite{9}.

In other cases, content is generated based on external data.
In one example, Monopoly boards are generated based
on demographic data for a given geographical area, using cri-
Audioverdrive: Exploring Bidirectional Communication Between Music and Gameplay

Nils Iver Holtar, Mark J. Nelson, Julian Togelius
IT University of Copenhagen

ABSTRACT

We described a system for bidirectionally coupling the music and gameplay of digital games, so gameplay procedurally varies the music, and music procedurally varies the game. Our goal is to inject game music more closely into the core of gameplay, rather than having the music serve as an aesthetic layer on top; traditionally, music responds to game state, but not vice versa. Such a coupling has compositional and design implications: composing game music becomes a kind of composition of gameplay, and furthermore, game-design decisions feed back into the music-composition process. We discuss an arcade-style 2d side-scrolling game, Audioverdrive, demonstrating this integrated music/game composition approach.

1. INTRODUCTION

Music is a key part of the culture and aesthetics of digital games, so much so that it often spills out of games proper, and into popular music culture. Games' soundtracks form a large part of their overhead and the cultural attachment they engender is such that fans flock to see symphonies perform the soundtracks [6]. Meanwhile, the particular aesthetic sound signatures of the sound chips in systems like the Atari VCS have inspired modern-day chipmusic and bitpop musicians to repurpose the hardware for music-making outside the gameplay context [4].

We refer to the latter as gaming as situated primarily in game design as a starting point. The distinction is admittedly not a clean one [10], but we find it productive here to start with existing game-design and game music composition practices, and experiment with adding bidirectional coupling between the two. It’s possible the result may converge nearer to electronic art, especially in particular designs using feedback loops aesthetically modeled on those common in cybernetic art. But so far, in our own use of these experimental tools (see Section 4), the result still feels much like game design and game-music composition, albeit in a weirdly coupled way.

1.2. Bidirectional game–music communication

Why not throw music right into that vortex of multidirectional coupling and feedback that makes up the heart of gameplay? That’s our long-term goal: game music drilled into the core of a game’s dynamics. In this paper, we ask something closely related yet architecturally simpler. We do maintain the nicely sealed computational boundary between “the game system” and “the music system”, in part so we can reuse existing technology on each side. However, we aim to break the compositional boundary: the two systems communicate in a pervasively bidirectional manner, with neither layer treated as subsidiary. This produces a closely coupled system with complex interaction patterns and feedback between gameplay and music. Our particular interest is in treating this closely coupled system as a unified compositional situation. In a quite direct sense the composer of game music becomes a composer of gameplay—and in the other direction, gameplay design becomes a kind of music design. Our contributions enable a bidirectional game–music composition process, and Audioverdrive, an arcade-style game designed and composed using the framework. The framework connects games programmed in Cocos2D, a game engine for the iPhone and iPad platforms, with generative spaces of music composed in Ableton Live, a popular piece of digital audio workstation software aimed at live performance, via a composer/designer-configurable set of mappings. Music’s aesthetics are loosely based on the first author’s experiences—previously separated ones—as composer/keyboardsitter for the game-music–influenced synth band Redacted For Blind Review, and designer of procedurally varying arcade games. One of the goals in that regard is to produce an actually playable version of the coupled gameplay–music experiences one often finds imagined in music videos in this genre.

2. BACKGROUND

Our goal of architecting the gameplay–music coupling as a bidirectional feedback loop will no doubt sound familiar-to audio installation artists. In contrast to games’ limited experimentation with such feedback loops, feedback between audience and generative-music systems (often conceptualized as a form of cybernetic coupling) is a common strategy deployed and explored by interactive sound installations. Therefore, in a sense our work can be seen as part of a recent trend in experimental digital games, which canvases electronic art for ideas and design elements to selectively borrow [8].

Despite intriguing connections to audio installations and interactive electronic art more broadly, we see ourselves as situated primarily in game design as a starting point. The distinction is admittedly not a clean one, but we find it productive here to start with existing game-design and game music composition practices, and experiment with adding bidirectional coupling between the two.

2.1. The composition-instrument

The existing style of game design closest to our goal of bidirectional coupling is probably the one theorized by Herber [7] as a composition-instrument. In a composition-instrument, the player can “play” or “compose” music in real-time while playing the game. In contrast to music-matching games such as Galactic Strawberry, the player generates (part of the) music through gameplay rather than matching gameplay to pre-defined music.

A particularly intriguing example from 1987 stands out, Otoky. A sidescrolling arcade shooter for the Nintendo Famicom Disk System, it places players in control of a flying fighter that fires a short-range ball projectile in order to deal with enemies. Each of the eight possible shooting directions has a musical note attached to it, making the soundtrack directly player-influenced. Otoky’s playable notes always map to a note belonging to the mode and harmony in the current accompanying layer, so that player-created melodies will never contain notes that sound “wrong” or “off” from the perspective of traditional harmonic structures. Furthermore, shooting is quantized to the beat so that all notes played will fit the rhythmic structure.

The mapping here is still mostly one-directional: music is dynamically generated from player actions, but does not feed back into the gameplay. However, musical considerations implicitly feed back into gameplay design through the constraints that were added to make “playing” produce the desired effect. This is seen most clearly in the shooting quantization. Although implemented straightforwardly in the gameplay domain as a quantization of shots, which in turn results in the notes produced by the shots being quantized, clearly the purpose of the constraint is the audio-domain quantization. It is therefore best thought of conceptually as a constraint in the audio domain, which travels “backwards” through the shot-to-note mapping to produce a constraint in the gameplay domain. The constraint here is fairly simple to hand-code in either domain, but with more complex musical constraints it is easy to see how less obvious interplay may arise.

2.2. Bidirectional procedural content generation

In addition to the experiments with dynamic and generative game audio already discussed, there has also, since the early 1980s, been work on procedurally generating game levels and other content. For example, classic games like Civilization, Rogue and Elite feature content that is automatically generated rather than created by a human designer. In some levels, cases are randomly generated, while in other cases a human player or designer is given some kind of control over the generated content [9]. In other cases, content is generated based on external data. In one example, Monopoly boards are generated based on demographic data for a given geographical area, using cri-
teria for inclusion specified by the player [5]. An example of procedural game level generation based on music is Audioverdrive, where the player can supply MP3 files which are automatically analyzed by the game’s software and turned into tracks for a form of racing game.

We focus on a mapping framework for the composer to manually link gameplay elements to musical elements and vice versa, taking a composition-oriented approach to the feedback loop. It is also possible to conceive of a more artificial-intelligence-oriented version of the approach: two procedural-generation systems, one game-to-music and the other music-to-game, hooked into each other in a loop, each one’s output serving as the other’s input. This would add a more substantial computational layer to the mappings. Where ours are quite direct, if the mappings were full-blown procedural generators, an additional algorithmic element arises, where it is not only the mappings but the algorithmic processes by which they’re computed that become aesthetically and experientially relevant. On the other hand, complex mappings risk breaking the aesthetic coupling by producing such a chaotic algorithmic coupling that the player no longer sees or is able to interpret the linkage.

2.3. Ableton Live

The audio side of our mapping framework is provided by Ableton Live, a popular digital audio workstation (DAW) package, i.e. software that allows arranging and processing of audio and MIDI, along with support for third-party plugins. Ableton Live is commonly used by producers, DJs, and engineers, and is particularly focused on live audio manipulation. Most notably, it features a workflow mode called the session view, where the typical horizontal time-line view is replaced by a matrix of cells, named clips. Clips are grouped into scenes and spread across individual tracks. A scene is a row, while a track is a column of clips. Tracks function like in traditional DAW software, where it is an output (stereo or mono) that can be assigned a chain of DSP units. Clips host the actual content played back to the track output, and only one clip can play per track at a time. Clips can either be launched individually or in groups, the most accessible way being through launching entire scenes (Figure 2 shows an example session view).

Ableton Live’s setup suits us well when working with dynamic game music, since a state change in the game can be mapped to trigger a specific scene or clip: the “live” part of the performance is here being played by our mapping application rather than directly by a human performer. Live also offers various quantization options, meaning that one can ensure that every action will be quantized to the next specified time unit. Out-of-the-box, there are many MIDI routing and mapping possibilities, enabling external MIDI controllers to be configured to add a substantial amount of control. The possibility space is further expanded when the official Max For Live extension is included, as this enables the creation of custom instruments and effects. Although we don’t explore it in our current experiments, the support for live audio manipulation/composition opens up the further possibility of a second person “livecoding” the game as it’s being played, by manipulating the musical composition and audio parameters which are in turn mapped into gameplay.

3. THE MAPPING FRAMEWORK

Our mapping framework is an application that sits between Ableton Live and the game, communicating with Ableton Live over the Open Sound Control (OSC) protocol and serving as the hub of gameplay–music coupling (see Figure 3). Ableton Live will report and alter its state based on received OSC messages, and the mapping application uses this functionality to create a tree list mirroring the structure of the current Live project. If the Live project is set up to explicitly send MIDI output to the mapping application (which will appear as a selectable MIDI output destination), some basic MIDI messages such as note-on and modulation will also be mappable.

Any program that exposes itself as an OSC node on the same network as the application will be selectable as a “game location”. Upon selecting a game location, the mapping system will query the IP address for mappable game parameters and if successful, all mappable game parameters will be registered in the application. The composer selects which parameters to use in the current configuration. The application will in turn inform the game about which parameters are used, limiting the amount of OSC traffic to the bare necessities.

3.1. The anatomy of a parameter

Each parameter is a floating-point number, though arbitrary semantics can be encoded into it. It might be a scaling property only assuming values between 0.0 and 1.0, or it could be a numerical representation of a color. We also distinguish between two types of parameters from a temporal perspective: continuous and discrete parameters. A continuous parameter is one that updates very frequently, and should be thought of as a continuous stream of values. A discrete parameter is one that updates less frequently, and would usually be linked to the initiation of a process of some kind, or an event.

In Ableton Live, this distinction is very clear as continuous parameters are the equivalent of knobs or sliders while discrete parameters are usually the triggering of clips and scenes. In a game, the distinction can quickly become a little more fuzzy. A parameter that reports the amount of enemies on screen could be seen as a continuous value, but the moment in time where an increase or decrease of this number occurs could in some cases also be viewed as a discrete parameter change, or event. To simplify, we let the frequency of updates loosely dictate the parameter type. In the actual source code, there is no significant functional difference between these two; the distinction is only made in the mapping interface as a compositional aid for classifying parameters.

3.2. The anatomy of a mapping

A valid mapping consists of one or more input parameters and one output parameter. The direction of the mapping can either be from Ableton Live to the game or vice versa. When one or more of the input parameters changes, a function will execute, and its 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 global variables created via the “Global Variables” button. This basic set of building blocks opens up many possibilities. The result of one mapping could for example scale the results of another. An output parameter, however, 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.

4. AUDIOVERDRIVE

Audioverdrive is an iPad game, using the Coco2D game engine. Ableton Live and the mapping software run on an OS X computer, communicating with the iPad app using OSC over a wireless network. The game is inspired by the genre of space shooters (which includes Gradius III and PixelJunk SideScroller). The player controls a side-scrolling ship, 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 act on 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 ship absorbs a different orb, the ship is hit by a bullet, or the ship is destroyed.

4.1. The game parameters

When run without OSC communication, little happens. 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 no incentive for the player to interact. An analogy could be made in viewing this state of the game as a synthesizer waiting to be played. Not surprisingly, the game takes a turn for the more interesting once the game parameters are put into use.

Table 1 lists all input game parameters that Audioverdrive accepts over OSC from the mapping application.

Table 1: Input game parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Health</td>
<td>Player’s health points</td>
</tr>
<tr>
<td>Score</td>
<td>Player’s score points</td>
</tr>
<tr>
<td>ammo</td>
<td>Number of bullets</td>
</tr>
<tr>
<td>shield</td>
<td>Shield strength</td>
</tr>
</tbody>
</table>


5 Konami, 1985

Ableton Live

The audio side of our mapping framework is provided by Ableton Live®. A popular digital audio workstation (DAW) package, i.e. software that allows arranging and processing of audio and MIDI, along with support for third-party plugins. Ableton Live is commonly used by producers, DJs, and engineers, and is particularly focused on live audio manipulation. Most notably, it features a workflow mode called the session view, where the vertical horizontal timeline view is replaced by a matrix of cells, named clips. Clips are grouped into scenes and spread across individual tracks. A scene is a row, while a track is a column of clips. Tracks function like in traditional DAW software, where it is an output (stereo or mono) that can be assigned a chain of DSP units. Clips host the actual content played back to the track output, and only one clip can play per track at a time. Clips can either be launched individually or in groups, the most accessible way being through launching entire scenes (Figure 2 shows an example session view).

Ableton Live’s set-up suits us well when working with dynamic game music, since a state change in the game can be mapped to trigger a specific scene or clip: the “live” part of the performance is here being played by our mapping application rather than directly by a human performer. Live also offers various quantization options, meaning that one can ensure that every action will be quantized to the next specified time unit. Out-of-the-box, there are many MIDI routing and mapping possibilities, enabling external MIDI controllers to be configured to add a substantial amount of control. The possibility space is further expanded when the official Max For Live extension is included, as this enables the creation of custom instrumentations and effects. Although we don’t explore it in our current experiments, the support for live audio manipulation/composition opens up the further possibility of a second person “livecoding” the game as it’s being played, by manipulating the musical composition and audio parameters which are in turn mapped into gameplay.

3. THE MAPPING FRAMEWORK

Our mapping framework is an application that sits between Ableton Live and the game, communicating with Ableton Live over the Open Sound Control (OSC) protocol and serving as the hub of game-play–music coupling (see Figure 3). Ableton Live will report and alter its state about which parameters are used, limiting the amount of OSC traffic to the bare necessities.

3.1. The anatomy of a parameter

Each parameter is a floating-point number, though arbitrary semantics can be encoded into it. It might be a scaling property only assuming values between 0.0 and 1.0, or it could be a numerical representation of a color. We also distinguish between two types of parameters from a temporal perspective: continuous and discrete parameters. A continuous parameter is one that updates very frequently, and should be thought of as a continuous stream of values. A discrete parameter is one that updates less frequently, and would usually be linked to the initiation of a process of some kind, or an event.

In Ableton Live, this distinction is very clear as continuous parameters are the equivalent of knobs or sliders while discrete parameters are usually the triggering of clips and scenes. In a game, the distinction can quickly become a little more fuzzy. A parameter that reports the amount of enemies on screen could be seen as a continuous value, but the moment in time where an increase or decrease of this number occurs could in some cases also be viewed as a discrete parameter change, or event. To simplify, we let the frequency of updates loosely dictate the parameter type. In the actual source code, there is no significant functional difference between these two; the distinction is only made in the mapping interface as a compositional aid for classifying parameters.

3.2. The anatomy of a mapping

A valid mapping consists of one or more input parameters and one output parameter. The direction of the mapping can either be from Ableton Live to the Game or vice versa. When run without OSC communication, little happens. But since no enemies are spawned (the default enemy spawn rate is zero), there is no incentive for the player to interact. An analogy could be made in viewing this state of the game as a synthesizer waiting to be played. Not surprisingly, the game takes a turn for the more interesting once the game parameters are put into use.

Table 1 lists all input game parameters that AudioVerdrive accepts over OSC from the mapping application.

**Table 1**

<table>
<thead>
<tr>
<th>Input Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current Score</td>
<td>Game score</td>
</tr>
<tr>
<td>Player Lives</td>
<td>Game life</td>
</tr>
<tr>
<td>Power Ups</td>
<td>Game power</td>
</tr>
<tr>
<td>Time</td>
<td>Game time</td>
</tr>
</tbody>
</table>

AudioVerdrive is an iPad game, using the Coco2D game engine. Ableton Live and the mapping software run on an OS X computer, communicating with the iPad app using OSC over a wireless network. The game is inspired by the genre space shooter that includes Gradius® and PixelJunk SideScroller®. The player controls a side-scrolling ship, 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 a different orb, the ship is hit by a bullet, or the ship is destroyed.

4. AUDIOVERDRIVE

AudioVerdrive is an iPad game, using the Coco2D game engine. Ableton Live and the mapping software run on an OS X computer, communicating with the iPad app using OSC over a wireless network. The game is inspired by the genre space shooter that includes Gradius® and PixelJunk SideScroller®. The player controls a side-scrolling ship, 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 a different orb, the ship is hit by a bullet, or the ship is destroyed.

4.1. The game parameters

When run without OSC communication, little happens. 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 no incentive for the player to interact. An analogy could be made in viewing this state of the game as a synthesizer waiting to be played. Not surprisingly, the game takes a turn for the more interesting once the game parameters are put into use.

Table 1 lists all input game parameters that AudioVerdrive accepts over OSC from the mapping application.
Some instantiate objects in the world, while others affect the behavior of the world and objects already in it. This was done primarily with the intent of enabling the creation of gameplay curves in tandem with musical curves. Changes in these parameters are also instantaneously perceivable in the game. So while it is still up to the composer to decide the degree of directness between the audio mappings and the parameters that govern the game world, the parameters exposed were chosen so as to make for a high chance that relations will be recognizable by the player, if anything clearly audible is mapped to these parameters.

We have also chosen to expose the victory and defeat actions, the consequence of this being that the composer essentially controls what constitutes victory and defeat in the game. Making the end conditions trigger in interesting ways, however, is only achievable through clever utilization of both these input parameters and the output parameters depicted in Table 2. These values, received from the game and transmitted by the mapping framework to Ableton Live, should be made to influence the music in ways that change the way the music again influences the game. This is where the composer can begin to really experiment, as it is possible to effectively create new rules in the game this way (illustrated in the next section).

4.2. The game design

The game design is broken into three parts: the music-to-game mappings, the game-to-music mappings, and finally, an overall phase structure in which both the gameplay and music progresses from an early to a series of intermediate phases, to the culmination of the level. This phase structure can be seen as both the “level design” and the “composition”: the composer’s choices guiding how the coupled experience unfolds at the macro scale.

4.2.1. How music controls the game

The musical aesthetic is an electronic track inspired by acts like Daft Punk and Lazerhawk and some elements of soundtracks to 1980s space shooter games. The instrument lineup is fairly basic, with relatively direct mappings to game state. Enemy shots are triggered when a kick or a snare drum plays, and all enemies currently on screen will attempt to crash into the player when a clap sample plays. A distorted siren instrument is mapped to spawn enemies, and this usually happens at the beginning of four-bar periods, backed up with a crash sound. A short plucking instrument triggers player shots, 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. Figure 5 illustrates how the terrain and instrument can correlate. 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-like instrument (made up of white noise and several siren waves) plays, where the position of the spawned orb is decided by the note pitch.

4.2.2. How the game controls the music

The game controls the music mainly by triggering clips and scenes in Ableton Live when events occur that resulted from player actions (directly or indirectly). Since the clips in turn instantiate and modify game elements when played, this produces the bidirectional communication. For example, when enemies are destroyed, an explosion clip is triggered along with a three-note chord played by the sonar-like instrument. Due to its mapping, this chord will also instantiate weapon orbs in the game state. That demonstrates how bidirectional mappings can work together to create new rules in game, since the instantiation of weapon orbs becomes a direct cause of destroying enemies. Different clips will be launched in Ableton Live depending on the current set of orbs connected to the player. 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 arpeggio instrument. Exactly which clip is launched differs depending on the number and type of orbs linked to the player. Picking up or connecting with different orbs therefore has different effects in the game: bouncing orbs will indirectly change the terrain, while other orbs will influence the player’s shot patterns.

4.2.3. Composition of the gameplay–music progression

The game is structured in four sequential phases (see Figure 6). Phases are programmed in the mapping framework (in Javascript), and influence which clips and scenes in Ableton Live, and therefore which 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, which can only be achieved by completing the challenge in each phase.

The first phase starts with a simple two-note sonar clip creating bounce weapon orbs. 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 instrument graduations are 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 sound. At this point in the track, neither the arpeggio nor the drums play, so no enemy or player shots are fired. However, a clap appears at the end of each four-bar period whenever

<table>
<thead>
<tr>
<th>Player speed</th>
<th>Continuous</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enemy Speed</td>
<td>Continuous</td>
</tr>
<tr>
<td>Terrain Speed</td>
<td>Continuous</td>
</tr>
<tr>
<td>Player Shoot</td>
<td>Discrete</td>
</tr>
<tr>
<td>Enemy Shoot</td>
<td>Discrete</td>
</tr>
<tr>
<td>Enemy Mode</td>
<td>Continuous</td>
</tr>
<tr>
<td>Enemy Spawn Rate</td>
<td>Continuous</td>
</tr>
<tr>
<td>Spawn Enemy</td>
<td>Discrete</td>
</tr>
<tr>
<td>Weapon Orb Type</td>
<td>Discrete</td>
</tr>
<tr>
<td>Weapon Orb Spawn Rate</td>
<td>Continuous</td>
</tr>
<tr>
<td>Spawn Weapon Orb</td>
<td>Discrete</td>
</tr>
<tr>
<td>Terrain Top Height</td>
<td>Continuous</td>
</tr>
<tr>
<td>Terrain Bottom Height</td>
<td>Continuous</td>
</tr>
<tr>
<td>Color None</td>
<td>Continuous</td>
</tr>
<tr>
<td>Color Spread</td>
<td>Continuous</td>
</tr>
<tr>
<td>Color Hoising</td>
<td>Continuous</td>
</tr>
<tr>
<td>Color Power</td>
<td>Continuous</td>
</tr>
<tr>
<td>Color Bounce</td>
<td>Continuous</td>
</tr>
<tr>
<td>Color BF</td>
<td>Continuous</td>
</tr>
<tr>
<td>Color Terrain</td>
<td>Continuous</td>
</tr>
<tr>
<td>Game Victory</td>
<td>Discrete</td>
</tr>
<tr>
<td>Game Defeat</td>
<td>Continuous</td>
</tr>
</tbody>
</table>

Table 1. Available input parameters for the game.

| Player X Position | Continuous |
| Player Y Position | Continuous |
| Player Deaths | Discrete |
| Total onscreen enemies | Discrete |
| Total destroyed enemies | Discrete |
| Enemy1 shot | Discrete |
| Number of spread | Discrete |
| Number of homing | Discrete |
| Number of power | Discrete |
| Number of burst | Discrete |
| Current modifier | Discrete |

Table 2. Available output parameters for the game.
Some instantiate objects in the world, while others affect the behavior of the world and objects already in it. This was done primarily with the intent of enabling the creation of gameplay concepts in tandem with musical concepts. Changes in these parameters are also instantly perceivable in the game. So while it is still up to the composer to decide the degree of directness between the audio mappings and the parameters that govern the game world, the parameters exposed were chosen so as to make for a high chance that relations will be recognizable by the player, if anything clearly audible is mapped to these parameters. We have also chosen to expose the victory and defeat actions, the consequence of this being that the composer essentially controls what constitutes victory and defeat in the game. Making the end conditions trigger in interesting ways, however, is only achievable through clever utilization of both input parameters and the output parameters depicted in Table 2. These values, received from the game and transmitted by the mapping framework to Ableton Live, should be made to influence the music in ways that change the way the music again influences the game. This is where the composer can begin to really experiment, as it is possible to effectively create new rules of the game this way (illustrated in the next section).

While the novelty of our framework lies in its possibility for creating system-altering mappings, some color parameters are also manipulable. This was done in an attempt to provide composers with more freedom in regards to which musical style to employ. It was our goal to avoid the potential restrictions that a too-predefined aesthetic might introduce. Also, since there is no screen overwriting the coupled experience unfolds at the macro scale.

4.2.3. Composition of the gameplay–music progression

The game is structured in four sequential phases (see Figure 6). Phases are programmed in the mapping framework (in Javascript), and influence which clips and scenes in Ableton Live, and therefore which 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, which can only be achieved by completing the challenge in each phase.

The first phase starts with a simple two-note sonar clip creating bouncing weapon orbs. 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 instrument mappings 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 sound. At this point in the track, neither the arpeggio nor the drums play, so no enemy or player shots are fired. However, a clap appears at the end of each four-bar period whenever

| Player speed             | Continuous | Enemy Speed | Continuous | Terrain Speed | Continuous | Player Shoot | Discrete | Enemy Shoot | Discrete | Enemy Shoot Speed | Continuous | Enemy Power | Discrete | Color Spread | Continuous | Color Tone | Continuous | Color Power | Continuous | Color Bounce | Continuous | Color BG | Continuous | Game Victory | Discrete | Game Defeat | Continuous |
|--------------------------|------------|-------------|------------|--------------|------------|--------------|-----------|-------------|----------|-------------------|------------|-------------|----------|-------------|------------|-----------|------------|------------|-------------|------------|-----------|------------|------------|---------|-----------|------------|

Table 1. Available input parameters for the game.

Table 2. Available output parameters for the game.

4.2.2. How the game controls the music

The game controls the music mainly by triggering clips and scenes in Ableton Live when events occur that result from player actions (directly or indirectly). Since the clips in turn instantiate and modify game elements when played, this produces the bidirectional communication. For example, when enemies are destroyed, an explosion clip is triggered along with a three-note chord played for convenience), and this instrument is 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. Figure 5 illustrates how the terrain and instrument can correlate. 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 sonor-like instrument (made up of white noise and several sine waves) plays, where the y-position of the spawned orb is decided by the note pitch.

4.2.1. How music controls the game

The musical aesthetic is an electronic track inspired by acts like Daft Punk and Lazerhawk and some elements of soundtracks to 1980s space shooter games. The instrument lineup is fairly basic, with relatively direct mappings to game state. Enemy shots are triggered when a kick or a snare drum plays, and all enemies currently on screen will attempt to crash into the player when a clap sample plays. A distorted snare instrument is mapped to spawn enemies, and this usually happens at the beginning of four-bar periods, backed up with a crash sound. A short plucking instrument triggers player shots, 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. Figure 5 illustrates how the terrain and instrument can correlate. 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-like instrument (made up of white noise and several sine waves) plays, where the y-position of the spawned orb is decided by the note pitch.

Figure 5. The lower image shows the voice of deep pad bass clip currently looping when the 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 screenshot, as can be seen in how the bottom terrain rises in steps.

Figure 6. Gameplay phases in Audovordrive.
an enemy is present, causing the enemy to charge towards
the player. When the player has succeeded in maintain-
ing 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 play-
ning, resulting in enemies shooting regularly. Since the
enemy spawn sound plays every fourth bar, the player must
actively dodge enemy 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 pat-
tern 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, in order to create
a clear contrast with the previous phase by making this
phase more spatially open and focused on combat.

After 32 bars, the player will enter phase three. Sim-
ilar to the second phase, the music here 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 cur-
rent 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 also omitted here, allowing the player the
possibility to influence the terrain generation again. This of-
fers a strategy for dealing with some enemies, as more
jagged terrain increases the likelihood of them colliding
with it. 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 sur-
vive for 32 bars. A main difference is that the terrain in-
strument plays much more elaborately, making the screen
home to many more weapon orbs. This was done in an
attempt to make the actual game experience match the cli-
matic nature of the music, as the screen is likely to be
filled with elaborate player and enemy orbs. As in the
second phase, no bounce orbs are spawned, and therefore
there is no terrain.

5. DISCUSSION

In our experience using this mapping framework to de-
sign/compose Audioverdrive, the music-composition and
gameplay-design processes are indeed intertwined (as in-
tended), though the unified composition situation often
ends up posing rather challenging constraints. Some mu-
tical choices are restricted by gameplay considerations when
in the context of particular mapping choices, and in
the other direction, some gameplay ideas must be scrapped
because of their inefficiency. On the other hand, cer-
tain combinations have a synergy where gameplay pat-
terns suggest development of a musical theme and vice
versa. For example, the terrain instrument played a much
more minor part in our initial design, but was expanded to
use more elaborate melodies because of how well the
cooperation with terrain generation worked.

A main challenge with our current approach to game-
music mapping is the lack of structure given by this very
open-ended space of possible mappings. Our framework
currently gives maximum flexibility, simply exposing in-
put and output parameters and allowing the composer to
build a coupled experience out of them by overlaying any
type of coupling structure they wish. This constitutes
a fairly low-level interface, however. The higher-level
structure that a composer can build on top of the frame-
work can take many forms, which are not reified in the
interface or architecture. Therefore, to achieve coherent
and interpretable results, the composer/designer needs to
develop a concept or method, more specific than the space
of all possible couplings, with which they'll organize the
game–music mapping in their particular piece.

The particular method we used for Audioverdrive was
to begin with a musical idea, which we used as a point of
departure for organizing the complete gameplay expe-
rience. This led to a macro-level composition oriented
around movements or phases, with transitions between
them triggered in a fairly conventional fashion by com-
pletion of gameplay objectives. Therefore these musi-
cal phases correspond one-to-one with gameplay phases.

More complex bidirectional feedback is then instantiated
within each phase, with game–music mappings depending
on the particular phase. The development of the mappings
within each phase took place in a more iterative fashion,
experimenting until we eventually reached a fixed point
where both the gameplay and the music “worked”.

The approach we took with Audioverdrive is only one
way of approaching the composition/design of these cou-
pled experiences. But, it may be worth focusing on and
developing a few such specific approaches in more depth,
to understand the composition/design of these coupled ex-
périences in a more structured context. In addition to al-
lowing us to gain further compositional/design experience
with specific idioms, a narrowed focus can also lead to op-
opportunities for better computational and interface support
for mappings. For example, we are considering building
interface modes around reified mapping patterns, such as
including the concept of a “phase” in the mapping inter-
face.

6. CONCLUSION

Our goal is to push game music into the core of game-
play through strong bidirectional coupling, where music
procedurally varies gameplay, and gameplay procedurally
varies music. With such a feedback loop, music composi-
tion can be treated as a separate aesthetic layer accompa-
nying game design, but is instead unified into a single
composition/design situation: composing game music
composes gameplay, and designing gameplay de-
signs game music.

In our initial work to realize coupled gameplay–music
composition, we built a mapping framework that allows a
composer to link gameplay parameters with Ableton Live elements in both directions. We designed a space-
shot game, Audioverdrive, using this framework, ex-
ploring a particular compositional style oriented around a
linear progression of “phases”, with coupled interaction,
through fairly direct mappings, within each phase.

On the experiential side, our goal is to produce an
experience that is somewhere between the composition-
instrumental and gameplay experience: on one hand, where the
player feels they are producing music; and the music-following
experience of games such as Guitar Hero, on the other,
where the player feels they are following along with pre-
written music. One of our touchstones in aiming at a
coupled experience between those two is the frequently
imagined—but not actually playable—interaction between
gameplay and music depicted in the music videos of some
electronic music groups.

On the composition side, our main interest has been
in exploring the results of constraints in each direction,
ensuring as much as possible that we are not simply layer-
ing gameplay on top of music or the reverse, but tackling
their coupling. Though the game is fairly simple, we think
that is the case in Audioverdrive. From a strictly techni-
cal perspective, in fact, the game cannot operate without
both sides of the equation. The Ableton Live setup and
Audioverdrive run on physically separated devices, con-
ected through a wireless network. When the connection
is interrupted, the game and music both clearly flounder
as their animating counterpart disappears, leaving a ship
floating aimlessly through space, accompanied by some
monotonous synths paddles.

7. ACKNOWLEDGEMENTS

Thanks the makers of modular software we strung together:
Ableton Live, VVOSC, LiveOSC, Cocos2D, and Box2D.

Figure 7. Screenshot from phase four, where the player
must simply survive for 32 bars. The sonic instrument
plays a full melody here, resulting in many weapon orbs;
the player is shown connected to four of them.

8. REFERENCES

[1] K. Collins, Game Sound: An Introduction to the His-
tory, Theory, and Practice of Video Game Music and

games,” Contemporary Music Review, vol. 28, no. 1,

sic,” Master’s thesis, University of Stellenbosch,

tory of chiptunes.” Transformative Works and Cul-
2009.0096.

esting monopoly boards from open data,” in Pro-
cedings of the 2012 IEEE Conference on Computa-
tional Intelligence and Games, 2012, pp. 288–295.

tifunctional medium,” Master’s the-
thesis, University College Dublin, 2006,
http://www.cianfurlongmusic.net/Music/Computer_
Game_MusicA_MultifunctionalMedium.pdf.

emergence and interaction,” Hor, vol. 9, 2007, http:
//www.hz-journal.de/t0/herber.html.

art, between interactive art and video games,” Mas-
ter’s thesis, University of California, Los An-
playtime.pdf.

metaphors for procedural content generation in
video games,” in Proceedings of the 2013 ACM SIGCHI
Conference on Human Factors in Computing Sys-
tems, 2013, pp. 1509–1518.

[10] O. Leino, “Re-conceptualising the play-element in
electronic art,” in Proceedings of the 17th Inter-
national Symposium on Electronic Art, 2011,
http://isea2011.sabanciuniv.edu/paper/
re-conceptualising-play-element-electronic-art.

On the principles of interactivity in music video
games,” in Proceedings of the 2007 Digital Games Research

game music in the 1990s,” in Music and Game: Per-
spectives on a Popular Alliance, P. Moormann, Ed.
an enemy is present, causing the enemy to charge towards the player. When the player has succeeded in maintain-
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 play-
ing, resulting in enemies shooting regularly. Since the en-
emy spawn sound plays every fourth bar, the player must actively dodge enemy bullets while trying to connect with
weapon orbs that activate the trigger 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 pat-
tern 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, in order to create
a clear contrast with the previous phase by making this
phase more spatially open and focused on combat.
After 32 bars, the player will enter phase three. Sim-
ilar to the second phase, the music here 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 cur-
rent 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 also introduced during this phase to give
the player the pos-
sibility to influence the terrain generation again. This of-
fers a strategy for dealing with some enemies, as more
jagged terrain increases the likelihood of them colliding
with it. 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 sur-
vive for 32 bars. A main difference is that the sonar in-
strument plays much more elaborately, making the screen
turn home to many more weapon orbs. This was done in an
attempt to make the actual game experience match the cli-
matic nature of the music, as the screen is likely to be
filled with elaborate player and enemy orbs. As in the
second phase, no bounce orbs are spawned, and therefore
there is no terrain.
5. DISCUSSION
In our experience using this mapping framework to de-
sign/compose Audioverdrive, the music-composition and
gameplay-design processes are indeed intertwined (as in-
tended), though the unified composition situation often ends up posing rather challenging constraints. Some mu-
sical choices are restricted by gameplay considerations when in the context of particular mapping choices, and in
the other direction, some gameplay ideas must be scrapped because of their potential constraints. On the other hand, cer-
tain combinations have a synergy where gameplay pat-
terns suggest development of a musical theme and vice
versa. For example, the terrain instrument played a much
more minor part in our initial design, but was expanded to
use more elaborate melodies because of how well the
coinciding with terrain generation worked.
A main challenge with our current approach to game-
music mapping is the lack of structure given by this very
open-ended space of possible mappings. Our framework
currently gives maximum flexibility, simply exposing in-
put and output parameters and allowing the composer to
build a coupled experience out of them by overlaying any
couple of structure that a composer can build on top of the frame-
work can take many forms, which are not reified in the
interface or architecture. Therefore, to achieve coherent
and interpretable results, the composer/designer needs to
develop a concept or method, more specific than the space
of all possible couplings, with which they’ll organize the
game–music mapping in their particular piece.
The particular method we used for Audioverdrive was to
begin with a musical idea, which we used as a point of
departure for organizing the complete gameplay expe-
rience. This led to a macro-level composition oriented around
movements or phases, with transitions between them triggered in a fairly conventional fashion by com-
pleting of gameplay objectives. Therefore these musi-
cal phases correspond one-to-one with gameplay phases.
More complex bidirectional feedback is then instantiated
within each phase, with game–music mappings depending
on the particular phase. The development of the mappings
within each phase took place in a more iterative fashion,
experimenting until we eventually reached a fixed point
where both the gameplay and the music “worked”.
The approach we took with Audioverdrive is only one
way of approaching the composition/design of these cou-
piled experiences. But, it may be worth focusing on and
developing a few such specific approaches in more depth,
to understand the composition/design of these coupled ex-
periences in a more structured context. In addition to al-
lowing us to gain further compositional/design experience
with specific idioms, a narrowed focus can also lead to op-
portunities for better computational and interface support
for mappings. For example, we are considering building
interface modes around reified mapping patterns, such as
including the concept of a “phase” in the mapping inter-
face.
6. CONCLUSION
Our goal is to push game music into the core of game-
play through strong bidirectional coupling, where music
procedurally varies gameplay, and gameplay procedurally
varies music. With such a feedback loop, music composi-
tions may be treated as a separate aesthetic level
accompanying game-design, but is instead unified into a
single compositional/design situation: composing game
music comprises gameplay, and designing gameplay de-
signs game music.
In our initial work to realize coupled gameplay–music
composition, we built a mapping framework that allows
a composer to link gameplay parameters with Ableton Live elements in both directions. We designed a space-
shooter game, Audioverdrive, using this framework, ex-
ploring a particular compositional style oriented around a linear progression of “phases”, with coupled interaction,
through fairly direct mappings, within each phase.
On the experiential side, our goal is to produce an
experience that is somewhere between the composition-
instrumentation dichotomy — on one hand, where the player
feels they are producing music; and the music-following
experience of games such as Guitar Hero, on the other,
where the player feels they are following along with pre-
written music. One of our touchstones in aiming at a
coupled experience between those two is the frequently
imagined—but not actually playable—interaction between
gameplay and music depicted in the music videos of some electronic music groups.
On the composition side, our main interest has been in
exploring the results of constraints in each direction,
ensuring as much as possible that we are not simply layer-
ning gameplay on top of music or the reverse, but tackling
their coupling. Though the game is fairly simple, we think
that is the case in Audioverdrive. From a strictly techni-
cal perspective, in fact, the game cannot operate without
both sides of the equation. The Ableton Live setup and
Audioverdrive run on physically separated devices, con-
ected through a wireless network. When the connection is
interrupted, the game and music both clearly flounder
as their animating counterpart disappears, leaving a ship
floating aimlessly through space, accompanied by some
monotonous synth pads.
7. ACKNOWLEDGEMENTS
Thanks the makers of modular software we strung together:
Ableton Live, VVOSC, LiveOSC, Cocos2D, and Box2D.

Figure 7. Screenshot from phase four, where the player
must simply survive for 32 bars. The sonar instrument
plays a full melody here, resulting in many weapon orbs;
the player is shown connected to four of them.

REFERENCES
[1] K. Collins, Game Sound: An Introduction to the His-
tory, Theory, and Practice of Video Game Music and
games,” Contemporary Music Review, vol. 28, no. 1,
sic,” Master’s thesis, University of Stellenbosch,
tory of chiptunes.” Transformative Works and Cul-
esting monopoly boards from open data,” in Pro-
ceedings of the 2012 IEEE Conference on Computa-
tional Intelligence and Games, 2012, pp. 288–295.
multifunctional medium,” Master’s the-
esis, University College Dublin, 2006,
http://www.cianburlongmusic.net/Music/Computer_
Game_Music_A_Multifunctional Medium.pdf.
emergence and interaction,” Hearing, vol. 9, 2007,
http://www.bz-journal.com/t0/herber.html.
art, between interactive art and video games,” Master’s
thesis, University of California, Los An-
playtime.pdf.
metaphors for procedural content generation in
games,” in Proceedings of the 2013 ACM SIGCHI
Conference on Human Factors in Computing Sys-
tems, 2013, pp. 1509–1518.
in electronic art,” in Proceedings of the 17th Inte-
national Symposium on Electronic Art, 2011,
http://issa2011.sabanciuniv.edu/paper/
re-conceptualising-play-element-electronic-art.
[12] ——, “An introduction to procedural music in video
games,” Contemporary Music Review, vol. 28, no. 1,
sic,” Master’s thesis, University of Stellenbosch,