Entertainment Computing 33 (2020) 100337

Contents lists available at ScienceDirect

Entertainment Computing

journal homepage: www.elsevier.com/locate/entcom

Generative music in video games: State of the art, challenges, and prospects

Cale Plut*, Philippe Pasquier

Metacreation Lab, School of Interactive Arts and Technology, Simon Fraser University, Surrey, BC V3T 0A3, Canada

ARTICLE INFO

Keywords:
Games
Ludology
Generative Music
Music
Audio
Game Music

ABSTRACT

Music is a common element in almost all video games. Most music in games is written by a human composer, and played as a linear piece behind gameplay. Adaptive and/or Generative music systems can be used to extend the musical content or create new musical content using algorithms and AI. While there is research into these systems, there has yet to be an organized examination of their architecture and use. We present a taxonomy of generative music in games, to allow for examination and discussion of generative music systems. In doing so, we also present a survey of the current state of the art of generative music systems in games, and discuss challenges and prospects of generative music for games.

1. Introduction and motivation

1.1. Game audio and music

The video game industry is one of the largest media industries in the world, with 65% of American adults reporting playing video games [1]. As the games industry becomes larger, more and more attention is being paid to the rigorous study and examination of games. While much of this study centers around the design of interaction, or the visual aspects of games, one of the most key components of games is audio. Even before the advent of digital games, audio has been a key component in the pure foundations of play. Audio is so key to play that it transcends human designed play – young animals vocalize their play with yips and growls [2,3]. Audio is so fundamental to games that in 1958, when the first video game *Tennis for Two* was developed, its gameplay was accompanied by audio.

Game audio is most commonly classified into the categories of *speech*, *sound/ effect*, and *music* [4]. *Speech* refers to voiced game audio that emanates from a character in a game. This includes the Player Character (PC) and Non-Player Characters (NPCs). Recordings of human voices are the most common source for speech, but speech may also be synthesized [5]. *Sound/Effect* generally refers to audio elements that are aperiodic and nonmusical, often inspired by real-world sound effects such as a gun firing or wind rustling through trees. Sound/Effect elements may also be abstract and artificial, such as Pac-man's "wakka-wakka", the sound of Mario jumping, or a simple beep when a button is pressed [6].

The final classification of game audio is *music*, and is our focus. While defining "music" is a contentious issue, we use Luciano Berio's definition of music as anything that intends to be music [7]. Generally this is pitched, and has some regular division of time, but these features are not necessary. It is important to note that while the speech/sound/music divisions are useful to describe audio content in games, the divisions are descriptive rather than prescriptive, and the lines between these classifications are not set in stone. In short, while we restrict our scope to game music, we take an inclusive perspective on what constitutes game music.

1.2. Generative and adaptive music

The most common use of music in games is to play a linear, composed piece of music during the gameplay. In many games, music is directly tied to the current level[1] and/or game state. With linear composed music, the music begins playing through a musical piece when the associated level is loaded. If the music reaches the end of the piece, the music loops. When a new level is loaded, the music either abruptly changes or quickly fades out, and is replaced with the new level's associated music. This use of music is often associated with older games such as 1985's *Super Mario Bros.* [8], 1986's *Castlevania* [9], and 1987's *Mega Man* [10]. However, lienar composed music is still in use in games such as 2009's *Final Fantasy XIII* [11], 2014's *Shovel Knight* [12], and 2017's *Pyre* [13].

* Corresponding author.
E-mail addresses: cplut@sfu.ca (C. Plut), pasquier@sfu.ca (P. Pasquier).

[1] "Level" refers to a section or area of a game that is delineated from other sections or areas of the game. There are many terms for "level" across game genre and development tool. For instance, in fighting games, each fight takes place on a "stage". In the Unity engine, assets are grouped and loaded by "scene". These terms are interchangeable. When discussing levels, we will use the internally agreed upon term from the genre, tool, or community

https://doi.org/10.1016/j.entcom.2019.100337
Received 8 July 2019; Received in revised form 2 November 2019; Accepted 6 December 2019
Available online 16 December 2019
1875-9521/

Linear composed music can also be used without clear level delineations. As computer memory has become more plentiful and programming tricks allow for seamless loading of content, games have relied less on clear level delineations between game states. The "open-world" genre type exemplifies this, with open-world games minimizing or eliminating level delineations all together. Such games still often have clear delineations between game states. The most common state change is one between "combat" and "non-combat" gameplay. 2017's *Hellblade: Senua's Sacrifice* [14] is an example of this state change. In *Hellblade*, there is a clear change between combat and non-combat – Senua draws her sword, the environment changes to create a small inescapable arena, the camera slightly changes its angle, and the players controls change to allow new actions. The music in *Hellblade* also changes with this state change. The "non-combat" music quickly fades out and the "combat" music quickly fades in. Games such as 1998's *Baldur's Gate* [15], 2009's *Batman: Arkham Asylum* [16], and 2016's *XCOM 2* [17] use this technique as well. It is important to note that this technique is not purely a more advanced game design or use of music, but an alternative structure. In *Final Fantasy X*, there is a level change between non-combat and combat. In *Final Fantasy XII*, there is simply a state change. In *Final Fantasy XIII*, the level change returns, and in *Final Fantasy XV*, there is no level change.

There are two primary techniques to extend linear, composed music in games. The first technique – *adaptive* music – addresses the *linear* use of music. Adaptive music is sometimes called "interactive music", and is music that reacts to a game's state [18]. Adaptive music can provide large amounts of unique music from limited musical content. Adaptive music directly connects musical features to game variables. These features can include adding or removing instrumental layers, changing the tempo, adding or removing processing, changing the pitch content, etc. These changes in adaptive music are directly linked to gameplay variables. The adaptivity of music can be understood as a dimension. Low levels of adaptivity may only adapt to a small set of in-game variables, while higher levels of adaptivity may adapt to tens or hundreds of in-game variables.

One use of adaptive music can be seen in *Luftrausers* [19]. In *Luftrausers*, composer Julio "Kozilek" Kallio wrote a single 120 s musical piece. This piece of music is split into 3 groupings of instruments, each of which has 5 different arrangements, for a total of 125 different arrangements that can provide 4 unique hours of music. These arrangements are linked directly to the player's selection of parts that makes up their avatar ship, as seen in Fig. 2. Adaptive music has been shown to increase a players' perceived and experienced tension during gameplay [20].

The other technique – *generative* music – addresses the creation of musical content itself. Most music is composed by an individual or team of human composers. Computational creativity is a field that explores the automation of creative tasks, and Musical metacreation (MuMe) is a subfield of computational creativity that addresses automating the creation of music [21]. *Generative* music [22] is music that is created via systemic automation, and is sometimes called procedural music, musical metacreation, or algorithmic music. These terms are mostly synonymous and can be used interchangeably, but we will use "generative music" for simplicity.

Generative music can provide endless unique music in a game, and can be adaptive on a much deeper level than composed adaptive music, providing music that is individually tailored to the player's actions in a game [23]. Despite these potential benefits, generative music has not yet achieved widespread use in video games.

There is debate as to whether all game music can be considered generative [24,25]. Because games are interactive, the exact timings of events is different for each player [3], which means that the musical timings will also be unique to each player. We believe that a refinement on what constitutes systemic autonomy is necessary to make meaningful distinctions for games. As an example, Mozart's *Musikalisches würfelspiel*, or musical dice game, is a well-known piece of generative music [26] in which the score is made up of multiple musical sections, each of which can transition to any other section. To play the piece, a performer/player rolls dice to determine which sections to play in which order. If we consider Mozart's dice game as a *game* rather than a music performance, we can consider each dice roll to be a game state change. Because the music is directly responding to the game state change, this does not present systemic autonomy from the game state and therefore the dice game could be described as having *composed adaptive* music, not generative music.

One problem with this understanding of the *Würfelspiel* is that the music and the gameplay of the game are inseparable – The game has no gameplay outside of constructing a musical piece. While there are video games in which the music is a core component of the gameplay loop, these games generally provide gameplay that is not purely musical. In "musical exploration game" *Fract OSC* [27], the player solves puzzles by moving and interacting with abstracted physics objects, each of which directly controls parameters on a virtual synthesizer. While the music is reactive, and is directly changed based on the gameplay, the music is not the only component of the gameplay. In the *Musikalisches würfelspiel*, there is no gameplay other than the arrangement of the music. This differentiation of gameplay and musical construction is key to our understanding of generative music in games.

For our purposes, music can be considered generative within a video game if the music is produced by a systemic automation that is partially or completely independent of the gameplay. This independence can have a large range of possibilities. A generative linear system may be almost completely independent of the gameplay – a piece of music can be requested, and is then linearly played through regardless of the gameplay. A highly adaptive generative music system may use a large array of game variables to inform the generation of musical content.

Fig. 1 gives examples of games that use either generative, adaptive, or both techniques in their music. The two most common uses of music in games are *composed linear* and *composed adaptive* music. We focus this survey on uses of both *generative linear* and *generative adaptive* music.

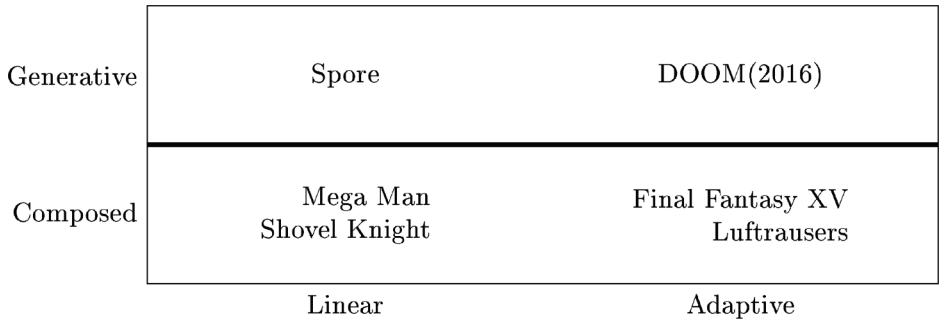

Fig. 1. Examples of games with varying degrees of adaptive and generative music.

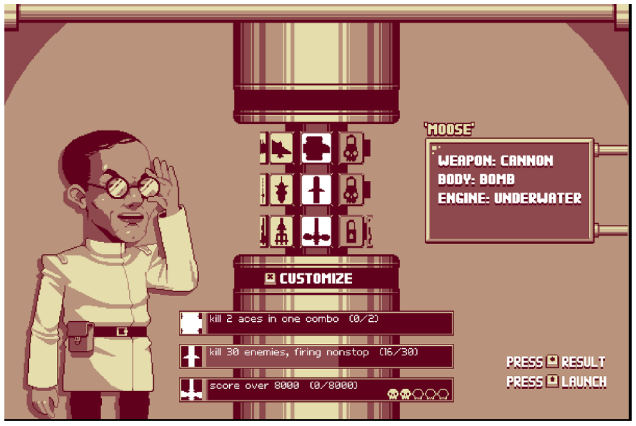

Fig. 2. *Luftrausers* selection of gameplay parts that influence music [19].

1.3. Motivation

Most games have a soundtrack with between 1 and 4 h of music, but gameplay time can range from 6 to over 100 h [28]. This often leads to the player hearing musical tracks repeated many times. While repetition is important in music, too much repetition can break immersion and become grating to the player [29,4]. One contributing factor to the repeated music in games is cost. As an example, *Pillars of Eternity* [30] takes an average of 60 h to complete. If all 60 h of gameplay were to be filled with unique composed music, the budget of the game would roughly double [31].

A generative music system can provide an amount of content well beyond what a composer is capable of at a much lower cost-per-minute of music. This would allow the designers of a game like *Pillars of Eternity* to fill all 60 h of gameplay with unique music at a cost well below the cost of 60 h of composed music. Generative music may even be used to provide additional music without real-time generation – a composer may use generative music to create a large library of music for an adaptive score.

Generative music also allows the player to customize and personalize their music. Adaptive music allows for music to more closely align with the actions of an interactive game, and has been shown to have a greater effect on the player's experienced tension than linear music [20]. However, adaptive music cannot necessarily match the extreme breadth of gameplay possibilities. Generative music can be composed in real-time as the player interacts with the game, allowing it to adapt more completely to the player actions.

Finally, generative music can empower and assist human composers. While constraints on creative freedom often paradoxically allow for greater creativity [32,4], too many creative constraints can be frustrating for human composers, and can limit their expressive range. When composing for adaptive music, composers must ensure that any of the possible combinations of music that can occur will sound acceptable together. This highly restricts the composer's possible expressive range. Generative music can allow composers to focus their attention on the artistic aspects of composing music, and not on the technical details of preparing music for adaptivity. This will allow for greater creative agency for game designers, audio designers, and composers.

1.4. Challenges

Although generative and adaptive music is becoming more popular in the industry, linear composed music represents a majority of games music, and generative music remains a niche field in both the industry and academia.

Generative systems can also be resource-intensive, and games are often pushing the limits of computing power without generative music [33]. While independent "indie" games often focuses on non-technical artistry, AAA (large-scale, big-budget) games continue to often push the limits of technology, with computational resources primarily directed towards graphical fidelity and fidelity of computational simulations [34]. While historically the technology for games music has kept pace with hardware, the relative resources allocated for music in games remains slim [4].

Another reason that generative music may not have received widespread attention in the games industry is that it is often unpredictable and can be difficult to control. The audio director of *No Man's Sky* (26), Paul Weir, notes that generative music was used in the game with an acknowledgment that it could produce "worse" music than composed music [5]. Generative music can also display the "10,000 bowls of oatmeal" problem [35,36], where the music is acceptable, but monotonous.

Games are expensive to make, with some game budgets reaching into the tens and hundreds of millions of dollars [4], and emerging technology also requires investment. It is understandable that games companies are not investing in a more expensive, less performative technology that may result in worse music, compared to simply using linear composed music. This lack of industry investment also means that while there is academic research in this area producing advanced systems, the research often takes place without industry collaboration, which limits the academic systems.

While there is some academic research into generative music for games, the research is at a very early stage. Often, systems with a stated goal of integrating into games will not have any integration into games [28]. Additionally, the design decisions for many systems are based entirely on practice and theory [37,33]. Finally, the evaluation of generative systems often takes place with either very limited video games [38] or without any integration into a video game at all [39]. Without clear and informed design goals or formal in-game evaluation, the benefits of generative music in games have not been demonstrated yet. This in turn results in less opportunity for academic collaboration with industry to advance the field.

A challenge in discussing generative music in games is that there is a wide range of poorly defined terminology in use in the field, and this terminology is often used incorrectly [5,4]. We present a typology of generative music in games, in the interest of allowing for more structured discussion of the state of the art. We use this taxonomy to present the state of the art as we know it, including peer-reviewed and published papers, public presentations and interviews, and industry uses. We survey both the research and the industrial implementations of generative music in games, and discuss the challenges and prospects of using generative music in games. When discussing a game that uses a system, or a system itself in the text, we refer to Table 1 using the convention *system name* (#).

2. Typology of generative game music systems

For our typology, we adapt the language used in the MuMe community [21]. The alterations that we make are driven by the interactive nature of games, and the unique requirements that games have for music. Our identified dimensions are shown in Fig. 3, and as before are to be understood as descriptive rather than prescriptive. In Fig. 3, the musical dimensions are displayed with their contextual relationships to each other, as we will discuss in Section 4. We note that many dimensions are not mutually exclusive, and a single system may address multiple aspects of many dimensions.

We have examined 34 generative musical systems from games, and have identified 10 dimensions that form a typology. These dimensions can be grouped into three dimensional types:

- **Sections 3 and 4: Musical Dimensions**. Due to the complex interactions between musical dimensions, we will first define the terms in Section 3, and then describe their interactions in Section 4.

Table 1
Extant game and academic systems examined within taxonomy.

Identification			Musical dimensions				Gameplay dimensions			Architecture dimensions			
Number	Game/System	Year	Generative Task	Directionality	Granularity	Grid	Diegesis	Ambience	Adaptivity	Generality	Gen. Algorithm	Musical Rep.	Musical KS
1	Ballblazer	1984	All	Horizontal	Phrase, Note	On	Non-diegetic	Ambient	Linear	Specific	Rule-based	Symbolic	External
2	Otocky	1987	Performance	N/A	Note	On	diegetic	Sourced	Adaptive	Specific	Rule-based	Symbolic	External
3	iMuse	1991	Arrangement	Horizontal	Measure	On	N/A	N/A	Linear	Generic	Rule-based	Symbolic	External
4	DirectMusic	1996	Arrangement	Mixed	Measure	On, Off	N/A	N/A	Adaptive	Generic	Rule-based	Symbolic, Audio	External
5	GhostWriter	1998	Composition	Horizontal	Note	On	Non-diegetic	Ambient	Adaptive	Specific	Rule-based	Symbolic	External
6	Munge/MNG	1998	Arrangement	Horizontal	N/A	Off	Non-diegetic	Ambient	Adaptive	Generic	Rule-based	Audio	External
7	No One Lives Forever	2000	Arrangement	Horizontal	Beat	On	Non-diegetic	Ambient	Adaptive	Specific	Rule-based	Symbolic	External
8	Rez	2001	Performance	N/A	Note	On	Non-diegetic	Sourced	Linear	Specific	Rule-based	Audio	External
9	Anarchy Online	2001	Arrangement	Mixed	Phrase, Instrument	On	Non-diegetic	Ambient	Adaptive	Specific	Stochastic	Audio	External
10	Diner Dash	2004	Arrangement	Horizontal	Phrase	On	Non-diegetic	Ambient	Linear	Specific	Rule-based	Audio	External
11	Halo Series	2004–2015	Arrangement	Vertical	Instrument	On	Non-diegetic	Ambient	Linear	Specific	Rule-based	Audio	External
12	Agate/AGMS	2008	Composition, Arrangement	Mixed	Note, Inst.	Off	N/A	N/A	Linear	Generic	Rule-based	Mixed	External
13	Spore	2008	Composition, Performance	Horizontal	Note	On	Non-diegetic	Ambient	Linear	Specific	Rule-based	Symbolic	External
14	Tom Clancy's EndWar	2008	Arrangement	Mixed	Phrase	On	Non-diegetic	Ambient	Adaptive	Specific	Rule-based	Audio	External
15	Uncharted 2: Among Thieves	2009	Arrangement	Horizontal	Phrase	On	Non-diegetic	Ambient	Linear	Specific	Rule-based	Audio	External
16	Red Dead Redemption	2010	Arrangement	Mixed	Phrase, Instrument	On	Non-diegetic	Ambient	Linear	Specific	Rule-based	Audio	External
17	Dark Void	2010	Arrangement	Vertical	Inst. Group	On	Non-diegetic	Ambient	Linear	Specific	Rule-based	Audio	External
18	The NLN Player	2011	Arrangement	Horizontal	Phrase	On	Non-diegetic	Ambient	Linear	Generic	Stochastic	Audio	External
19	Child of Eden	2011	Performance	N/A	Note	On	Non-diegetic	Sourced	Linear	Specific	Rule-Based	Audio	External
20	Bit.Trip Runner	2011	Performance	N/A	Note	On	Non-diegetic	Sourced	Adaptive	Specific	Rule-Based	Audio	External
21	Remember Me	2013	Arrangement	Vertical	Inst. Group	Off	Non-diegetic	Ambient	Adaptive	Specific	Stochastic	Audio	External
22	AudiOverdrive	2013	Arrangement, Performance	Mixed	Phrase, Instrument	Off	diegetic	Sourced	Adaptive	Specific	N/A	Audio	External
23	Cullimore, Hamilton, and Gerhard	2014	Composition	Horizontal	Chord	On	N/A	N/A	Linear	Generic	Stochastic	Symbolic	Learned
24	Engels et al.	2015	Composition	Horizontal	Note, Chord	On	N/A	N/A	Linear	Generic	Stochastic	Symbolic	Learned
25	Sonancia	2015	Arrangement	Horizontal	Phrase	On	Non-diegetic	Ambient	Linear	Specfic	Rule-based	Audio	Learned
26	No Man's Sky	2016	Arrangement	Mixed	N/A	On	Non-diegetic	Ambient	Linear	Specific	Rule-based	Audio	External
27	DOOM (2016)	2016	Arrangement	Horizontal	Phrase	On	Non-diegetic	Ambient	Linear	Specific	Rule-based	Audio	External
28	Prechtl	2016	Composition, Performance	Horizontal	Chord	On	Non-diegetic	Ambient	Adaptive	Generic	Stochastic	Symbolic	External
29	The Audience of the Singular	2017	Composition	Horizontal	Note, Chord	On	diegetic	Sourced	Adaptive	Generic	Stochastic	Symbolic	Learned
30	MetaCompose	2017	Composition, Performance	Mixed	Note, Chord, Inst. Parameters	On	N/A	N/A	Adaptive	Generic	Stochastic, GA, Rule-based	Symbolic	External
31	Melodrive	2018	Composition, Arrangement	Mixed	N/A	N/A	N/A	Ambient	Adaptive	Generic	N/A	Symbolic	External
32	Chuchel	2018	Performance	N/A	Note	Off	diegetic	Sourced	Linear	Specific	Rule-based	Audio	External
33	Ape Out	2019	Performance	N/A	Instrument	On	Non-diegetic	Ambient	Adaptive	Specific	Rule-based	Audio	External
34	Adaptive Music System/ AMS	2019	Arrangement	Horizontal	Note	On	Non-diegetic	Ambient	Adaptive	Generic	GA, RNN (Stochastic)	Symbolic	Mixed

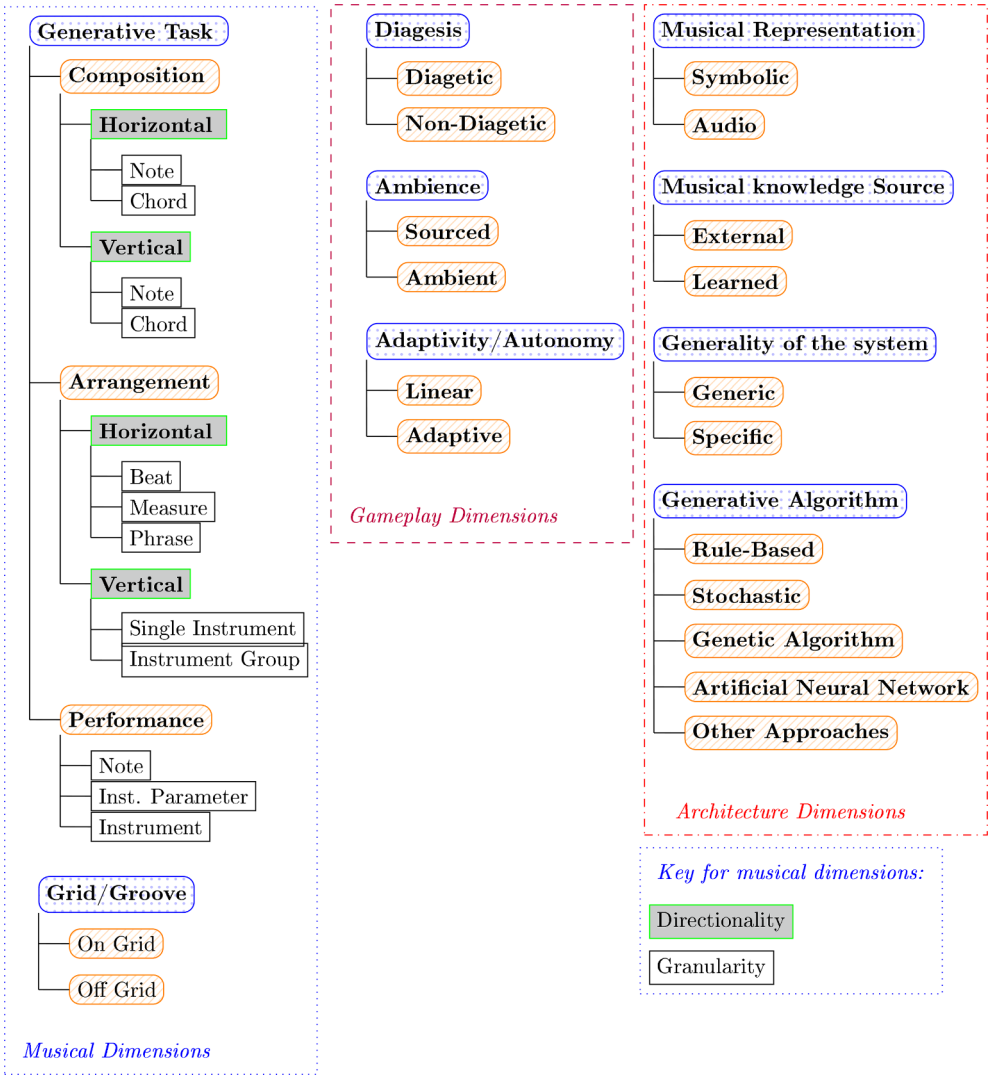

Fig. 3. Typology of generative music systems for games.

These dimensions describe how a system structures music.

- **Section 5: Gameplay dimensions**. These dimensions describe both how the generated musical content is used in the game, and how the musical generation is informed by the gameplay.
- **Section 6: Architecture dimensions**. These dimensions describe inner structure, data, and algorithms used by the generative music system.

3. Musical typology definitions

The musical dimensions address a system's relation to its musical output. Our four identified musical dimensions are *generative task*, *directionality*, *granularity*, and *grid/groove*. Music is multifaceted, and examining any dimension out of its musical context provides only a partial understanding. We begin our descriptions of musical dimensions by introducing and defining common musical terminology. We then discuss the interactions between these musical features that constitute the musical dimensions of our taxonomy.

While there are many ways to analyze music, we use western music theory to describe our musical dimensions, as we believe that it provides a structure for describing and taxonomizing the musical output of examined systems. Most music for games falls within the western tradition of music, with non-western instruments or harmonies primarily used as a special musical effect [4].

3.1. Generative task

The first musical dimension to consider for a generative system is the generative task that the system addresses. This dimension describes what the system generates. While there are many tasks that a system can address, there are three over-arching families of tasks that generative music systems for games can address:

3.1.1. Composition

The composition task addresses the creation of new music, and the creation of new musical structures, entirely through some process.

3.1.2. Arrangement

The arrangement task addresses the recombination of extant musical elements in new ways.

3.1.3. Performance/interpretation

The performance task addresses the interpretation of extant music. The line between the performance and composition tasks is not a definite one as musical elements such as dynamics (volume) may be included in the composition or left to interpretation. A generative music system addresses the performance task if the music that it alters and interprets exists separately from the interpretation.

These tasks are not mutually exclusive, and a single system may

address any combination of tasks. Most of the systems that we have surveyed address a single task, though *Ballblazer* (1), *Agate/AGMS* (12), *Spore* (13), *Audioverdrive* (22), *Anthony Prechtl system* (28), *MetaCompose* (30), and *Melodrive* (31) address multiple tasks.

3.2. Directionality

The second musical dimension to consider for a generative system is the directionality of the systems manipulation. Sheet music is arranged with time progressing from left to right in each musical system. Musical events that happen at the same time are represented in sheet music by appearing at the same horizontal position within the musical system. Because of this arrangement on the page, music can be examined horizontally or vertically, which is common in music theory. This is reflected as well in the concepts of horizontal resequencing and vertical layering [4] – horizontal resequencing modifies music across time, and vertical layering modifies music at points in time. The horizontal dimension of music represents the way that the music unfolds over time. The vertical dimension of music represents the way that the music fits together with itself at any single point in time. Systems can also act in mixed or hybrid directionality, manipulating both the horizontal and vertical dimensions of music.

3.3. Granularity

In music, individual elements combine to create complex groupings of features. These groupings also combine to form more complex meta-groupings, which can continue to combine in increasing complexity. The granularity dimension addresses what level of groupings the system manipulates. The exact manipulation of different granularity levels depends on both the task and direcitonality of a system, though there are a few commonalities across music:

3.4. Horizontal properties

3.4.1. Note

A note is a single musical event. Notes usually have a pitch, but this is not necessary. Notes usually have a duration between 0.125 and 2.0 s, though may be longer or shorter. While other terminology exists for musical events, the differences are purely semantic. We continue to use western music theory inclusively in this case and will refer to any single musical event as a “note”. Each note in music has several parameters that describe the note. These include but are not limited to: pitch, onset time, duration, volume, attack, decay, sustain, and release.

3.4.2. Beat

A beat is a regular division of time in music. Beats generally have a length of between 0.3 and 1.0 s. This is measured in music by the number of beats in a minute. While this range falls within the range of individual notes, a key differentiation between beats and notes is that beats generally maintain a similar duration for longer periods of time, while note are much more variable. This means that a single note may have a duration anywhere from a fraction of a beat to multiple beats. A beat may also be understood as any regular tick, and the terms may be used interchangeably.

3.4.3. Measure

A measure is an organized collection of beats. Most commonly, a measure has either three or four beats.

3.4.4. Phrase

A musical phrase is a collection of notes that combine to form a musical idea. Phrases are most often between four and eight measures long, and can also be combined together to create longer phrases.

3.4.5. Chord

A chord is any selection of two or more notes that are meant to sound together. These notes may either play at the same time or may *arpeggiate*, playing the notes of the chord across time with the intention that they are heard as a single unit. While the creation of a chord is within the vertical dimension, chords may also be treated as a musical unit – the notes C-E-G can be described as simply a C Major chord. When chords are arranged horizontally in series, this is called a *chord progression*.

3.5. Vertical properties

3.5.1. Instrument parameter

In acoustic music performance, a performer of an instrument has a variety of ways to alter the timbre, dynamics, envelope, and more aspects of the sound of their instrument. Digital instruments also have parameters that may be similarly altered.

3.5.2. Instrument

An instrument is a single, internally consistent source of sound, that can be heard as a single musical entity across time. In most cases, a musical phrase will not change instruments midway through, though there are exceptions.

3.5.3. Instrument group

Multiple instruments are often grouped together in music. Instruments are often grouped together due to having similar sonic qualities or similar musical function [4], but may be grouped in any combination.

3.5.4. Chord

As mentioned, a chord is any selection of two or more notes that are meant to sound together. When chords are manipulated along the vertical dimension, individual notes are combined to create or manipulate chords, rather than using common, pre-defined chords.

3.6. Grid/groove

While the previous dimensions describe the way that systems generate music through time, the Grid dimension describes the relationship that the music has with time. One of the most common interfaces for building drum tracks is a step sequencer, which divides each measure into 16 equal steps, equivalent to one 16th note each. Trackers, another common interface for music composition for video games, also arrange the music as a grid of equal steps. The grid dimension describes whether a system plays musical events at regular, perceivable divisions of time. Another way to describe this dimension is whether the time deltas between musical events maintains a consistent, even ratio with each other.

3.6.1. On grid

Systems that are on grid restrict the timings of their musical events to some regular division of time. Systems that are on grid are sometimes described as having a “groove”, though the term is poorly defined. *Rez* (8) organizes its musical events on a grid – when the game triggers a chord cluster from player events, the exact timing is changed to fit within an 8th note groove.

3.6.2. Off grid

Systems that are off grid do not restrict the timing of musical events to any regular division of time. Such systems generally do not restrict the timing of musical events to a regular pattern. The generative system used in *Spore* (13) sometimes plays patterns with rhythmic regularity, though the beginning point of any pattern is not restricted to a regular division of time.

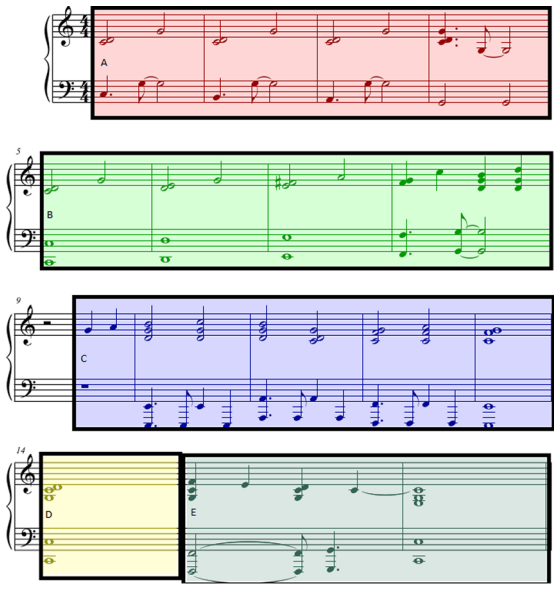

Fig. 4. Individual phrases for use in horizontal arrangement.

4. Interactions between musical dimensions

When the dimensions of Generative Task, Directionality, and Granularity are taken out of musical context, they provide an insufficient and incomplete understanding of the musical structures that a system manipulates. To understand the relationships and dependencies of these dimensions, we must also examine several common musical interactions that place these dimensions in the proper context.

Our examined interactions are the most common combinations of the dimensions of task, directionality, and granularity. As before, these common combinations are not mutually exclusive. It is possible for a single system to automate both horizontal and vertical composition, or to automate horizontal composition on a note level of granularity, vertical arrangement on an instrument level of granularity, and the instrument parameters of performance. We also note that the grid dimension does not depend on nor influence these interactions, and is a separate musical dimension.

4.1. Horizontal composition

Horizontal composition systems automate the creation of music across time. This may be on one of two levels of granularity:

4.1.1. Note

Horizontal composition systems that function at the note level of granularity automate the creation of new music by selecting a series of individual notes as they will be heard through time. Note that some note-level systems may also create larger groupings of notes that can later be used by an arrangement system, which is similar to the use of *leitmotifs* in composed music. The system by *Cullimore, Hamilton, and Gerhard*(23) addresses the composition task and functions at the horizontal note level.

4.1.2. Chord

Horizontal composition systems that function at the chord level of granularity automate the creation of chord progressions. These chords may either be selected from common chords (e.g. C Major), or may be built procedurally with vertical composition. *MetaCompose* (30) generates chord progressions by choosing common chords from a tree, while addressing the composition task.

4.2. Vertical composition

Vertical composition systems automate the construction of music at points in time. Such a system exclusively functions on a note level of granularity, combining notes to create chords. We have found no systems that fit our scope of generative music for games that exclusively address the composition task in the vertical direction. However, there are systems that include a chord-building element. *The Audience of the Singular* (29) considers the vertical note dimension while composing music.

4.3. Horizontal arrangement

Horizontal arrangement systems automate the combination and recombination of composed musical beats, measures, and phrases in new ways.

4.3.1. Beat

Horizontal arrangement systems that function on a beat level of granularity may either combine individual composed beats together to form larger collections of beats. They may also instead combine larger musical groupings together, but have the option to alter the music at each beat. Therefore, a beat-granularity horizontal arrangement system may play through phrases completely if left unattended, but choose to change which phrase is playing mid-way through the phrase at a specific beat. In *No One Lives Forever* (7), the system plays through phrases entirely while in a single game state, but can transition to different musical phrases on any beat.

4.3.2. Measure

Horizontal arrangement systems that function on a measure level of granularity are almost identical to those that function on a beat level of granularity. These systems can either arrange composed measures together, or have the option to alter longer phrases at any measure. The former is more common at the measure level of granularity than it is at the beat level of granularity. Mozart's previously mentioned *Musikalisches würfelspiel* is an example of a horizontal arrangement system at the measure level of granularity.

4.3.3. Phrase

Horizontal arrangement systems that function on a phrase level arrange composed phrases together across time to create more completed musical pieces. Figs. 4 and 5 provide an example of how horizontal phrase arrangement functions. In Fig. 4, various musical phrases are provided. In Fig. 5, these phrases are combined across time to create a novel musical piece. Complete musical phrases generally have a duration of 8–32 s. Because changes in game states may occur at any

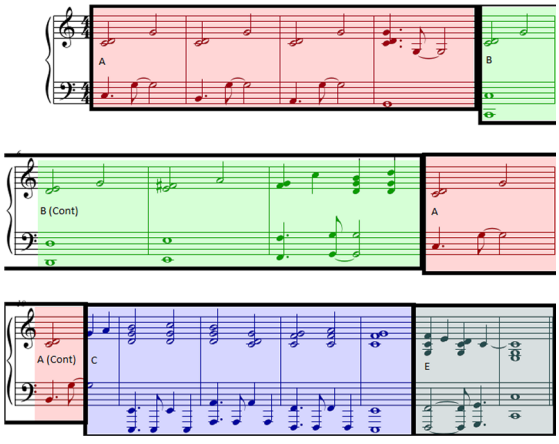

Fig. 5. Sample horizontal arrangement.

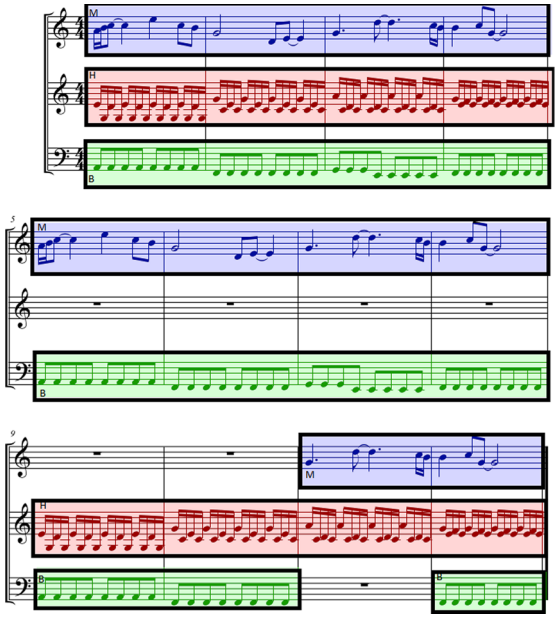

Fig. 6. Vertical Arrangement.

time, if a horizontal arrangement system can only change between musical cues at the end of a phrase, there is a possibility that the music may feel disconnected from the gameplay [4]. Because of this, systems that exclusively address horizontal arrangement in the phrase granularity are rare. For all of the music in Figs. 4–6, a recording can be heard at `https://bit.ly/2NKI7rk`. These examples were composed by the first author for the purposes of demonstrating these concepts.

4.4. Vertical arrangement

4.4.1. Instrument

Vertical arrangement systems that function in the instrument level of granularity arrange single instrumental lines into and out of a fuller orchestration. Fig. 6 demonstrates how instrumental vertical arrangement may work. In the first system, all musical parts (Melody (M, blue), Harmony (H, red), and Bass (B, green)) are playing. In the second and third systems, individual instruments are removed and re-introduced to the total arrangement, creating a new combination of musical elements.

4.4.2. Instrument group

Vertical arrangement systems that function in the instrument group level of granularity are almost identical in function to those that function on the instrument level of granularity. However, instead of removing or adding individual instrumental lines from an orchestration, such systems instead remove or add instrumental families from the mix. The system in *Dark Void* (17) functions on an instrument group level of granularity – the generative system cannot add or remove a single insturment, but instead adds or removes combinations of instruments together.

4.5. Performance

Performance systems automate the interpretation of music. Performance systems are not differentiated by their directionality. This is because performance systems alter properties that describe moments of time, but alter the dimensions across time. That is to say, performance systems alter vertical properties horizontally. Performance systems also often alter several properties at once. For simplicity and clarity, we will describe the smallest granular unit that a system manipulates when discussing performance systems.

Note Performance systems may alter the pitch of notes during gameplay. Pitch alterations are generally limited to only one or two semitones. The pinball game *Black Knight 2000* [40] alters the pitch and timing of musical sound effects based on the surrounding musical context [4]. In *Chuchel* (32), most of the sound effects and voice sounds are pitched or semi-pitched. The system alters the pitch of the PCs audio to fit within the surrounding musical context. Performance systems may also alter the timing and duration of notes, or may choose to omit or add notes during gameplay. This is sometimes part of a larger gesture such as a tempo change, or a smaller gesture such as a fermata. Performance systems that add or omit notes often to so to develop or simplify a phrase, or may add musical ornaments. *MetaCompose* (30) can both alter the duration of notes, as well as adding or removing notes from a generated melody.

Instrument parameter Performance systems may automate the parameters of a single or multiple instruments. Some examples of parameters that performance systems may alter are dynamics (velocity/volume), ADSR envelope, and spectrum (using filters or changing the waveform of the sound).

Instrument Effect Performance systems may automate the presence and parameters of audio effects on instruments. Effects are commonplace in composed music, but are generally underused in generative systems. Effects are often used together. We borrow and modify Michael Sweet's taxonomy of audio effects [4], which are separated into three categories:

- **Time-based effects** generally add some form of echo to shape the way that a sound evolves over time. These effects include reverb, delay, chorus, flange, and phase effects.
- **Frequency-based effects** alter the spectrum of the frequency, with effects such as filters, equalizers, or resonators. Vibrato is also an example of a frequency-based effect, where a low-frequency oscillator alters the frequency of a note subtly.
- **Volume-based effect** change the dynamics of music. These effects include a Tremolo, which uses a low-frequency oscillator similar to vibrato, but alters the dynamics instead of the frequency. Other volume-based effects include limiter, compressor, gate, and expander.

Instrument Performance systems may automate which instrument is playing a line. This is distinct from arrangement systems selecting instruments, as a performance system will select *which* instrument to use when playing through a composed line, not *whether* the instrument will play. *Otocky* (2) adaptively changes instruments during gameplay.

These dimensions and common structures describe the musical construction and output of generative systems for games. The next set of dimensions describe the interaction between the music system and the gameplay itself.

5. Gameplay dimensions

Our first gameplay dimension is the adaptivity, or autonomy, that the system has. This dimension describes how much the system adapts to the gameplay. Adaptivity can be understood as the dimension that describes how the system deals with input from the game.

We borrow Richard van Tol and Sander Huibert's "IEZA" framework for classifying game audio [41], which describes how the system's output interacts with the game. The IEZA framework is intended to describe all aspects of game audio, and we adapt the framework to focus on music. Fig. 7 shows our adapted framework, and examples of where common audio elements exist within the framework. The two dimensions of our framework are *diegesis* and *ambience*.

5.1. Diegesis

The dimension of diegesis describes whether the music is diegetic or non-diegetic. Diegetic music is music that exists within the game world,

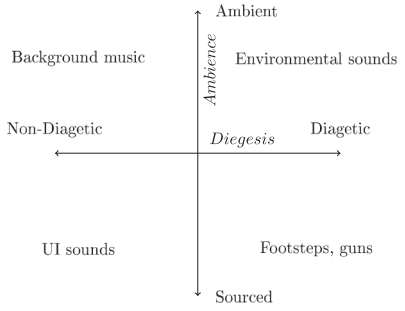

Fig. 7. Adapted IEZA model of game audio.

while non-diegetic music exists outside and along the game world.

5.1.1. Diegetic

Diegetic music originates from within the game world. This may take the form of audio emanating from an in-game object such as the radio stations on a "pip-boy" in *Fallout 3* [42], it may take the form of an in-game musical instrument such as in *The Legend of Zelda: The Ocarina of Time* [43], and it may even take the form of in-game sound effects having musical pitches, as in the case of *Pole Riders* [44].

5.1.2. Non-diegetic

Non-diegetic music does not originate from within the game world. Most games music is non-diegetic, where a musical score simply plays during gameplay. The in-game characters are not aware of non-diegetic music, it is provided exclusively for the player's benefit.

5.2. Ambience

The dimension of ambience describe whether the music is connected to a source, or is ambient. Most music in games is ambient and not connected to a source.

5.2.1. Sourced

Music that is sourced is linked to a specific source in the game, though not necessarily a source in the game world. A simple example of non-diagetic sourced music is a musical response when the player clicks on a UI button. As an example, in *Chuchel* (32), the only sounds that the player's character makes are abstract non-language vocal sounds, and are pitched to fit in with the musical surroundings. This is an example of a diegetic sourced music.

5.2.2. Ambient

Ambient music is not sourced, and instead emanates from the environment. Most music in games is non-diegetic and ambient. We do not use "ambient" to refer to a musical style, but rather to the use of the music. A non-musical example of a diegetic ambient sound is ambient weather sounds. A musical example of the common non-diegetic ambient sound is in *Halo 2* (11), in which the music is not connected to any source, and does not emanate from within the game world.

5.3. Adaptivity/autonomy

The level of autonomy that a system has from a game describes how the system reacts and responds to the events and state of a game. As we have mentioned in Section 1.2, to qualify under our definition of generative, a system must have some degree of autonomy from the gameplay. Because generative music systems for games exclusively output audio of their compositions in real-time with the gameplay, this dimension addresses the adaptivity of the music in the game. The amount of autonomy that a system has is inversely related to how adaptive the system is. The autonomy dimension is also distinct from many other dimensions, because it is a continuous dimension, while other dimensions are more categorical. There are two main divisions of autonomy for game generative music systems:

5.3.1. Linear

Linear systems have a high degree of autonomy from the gameplay. Such systems generate music with little input from the gameplay. This use of music can involve generating a single piece of music for a level/game state as the level loads, that is then used as a linear piece of music. This can also involve creating a musical composition in real-time as the gameplay unfolds, but only using variables that were set at the beginning of the generation. *Spore* (13) presents an example of linear generative music. In *Spore* (13), a piece of music is generated at the same time that the game environment loads. Once the music is generated, it is used as though it is any other linear piece of music – looping through the music from beginning to end.

5.3.2. Adaptive

Adaptive systems have a lower degree of autonomy from the gameplay. Such systems generally generate their music in real-time, continuously updating the features of their generation to match with the constantly updating game state. This theoretically allows for a generative system to be adaptive on a much deeper level than with composed adaptive music, because the music can be altered more completely than with composed adaptive music. *Melodrive* (31) describes this use of music as "Deep adaptive music".

5.3.3. Reactive

Reactive systems have no autonomy from the gameplay. In *Fract OSC*[27], the gameplay acts as an interactive synthesizer. The music in *Fract OSC* reacts directly to the position and properties of in-game objects, that are directly controlled by the player. While reactive systems are considered generative by some definitions [24], a purely reactive system does not fall within our scope because the system has no autonomy from the gameplay.

6. Architecture dimensions

The musical and gameplay dimensions describe the way that a generative game system interacts with a game, and how its musical output is structured. While these dimensions describe the practical dimensions of a system, there are also architectural dimensions that describe the inner workings and knowledge of a system.

Architecture dimensions describe how a system is internally structured. These dimensions describe the way that a system organizes and understands its data, in contrast to the previous dimensions that describe the way the system manipulates its data.

6.1. Generality of the system

As mentioned, generative music does not yet have widespread use in the games industry. Most industrial systems that use generative techniques do so to extend and expand a game's composed music to fit within the game. This also means that industry systems generally are designed specifically for the music and the game that they are integrated into. In contrast, academic systems are generally designed to provide a generic platform that can be integrated into many different games without fundamental changes. The generality dimension measures whether the generative system is designed as a generic platform, or whether the system is designed specifically for a single game.

6.1.1. Generic

Generic systems attempt to create a framework that is game-agnostic. Generic systems are independent systems, that may be integrated into multiple different games without requiring major systemic overhaul. Generalist systems are most common in academic systems, but there are systems used in the game industry as well. *DirectMusic* (4)

and *Agate/AGMS* (12) are both examples of generic music generation systems.

6.1.2. Specific

Specific systems are designed explicitly for the game that they provide music for. Such a system is intrinsically linked into the game that it provides music for. These systems generally are designed based around the surrounding musical context and gameplay events and variables for the game that they provide music for. Game-specific systems are most common in industrial uses. *Red Dead Redemption* (16) provides an example of a game-specific system. The musical system in *Red Dead Redemption* (16) takes important game variables as input. Some of the tracked variables and musical reactions are generic and applicable to many games, such as playing faster and more active music during combat sequences. However, many of the tracked variables and reactions are specific to *Red Dead Redemption* (16), such as changing the music when the player mounts a horse. While the core idea of an adaptive system that reacts to game events could be used elsewhere, removing the game context from this system would fundamentally change the workings of the system, and as such it is not generic.

6.2. Generative algorithm used by the system

While the generality of a system describes how it fits into the larger world of the game, the algorithm of a system describes how it creates music. Almost any AI algorithm can and has been used for computational creativity purposes [21]. Listing all of the possible generative musical algorithms far exceeds the scope of this survey. Instead, we focus on the four most common algorithms for use in MuMe. Of these four algorithms, only two are common in generative systems for games.

6.2.1. Rule-based

A rule-based algorithm uses a set of rules, either learned or programmed in, to generate its output. A simple example of a rule based system in the symbolic music domain is one that uses species counterpoint[2] to create harmony lines. *Rez* (8) uses a rule-based system where the exact timings of musical stings depend on a set of rules determined by the surrounding musical context. Some rule-based systems are purely deterministic – given identical input they will produce identical output. Some rule-based systems instead are non-deterministic, often using elements of stochasticity.

6.2.2. Stochastic

A stochastic system uses a pseudorandom process to generate its output. This randomness may be evenly distributed or be the result of a statistical model providing weighted changes. A Variable-order Markov model (VOMM) is an example of a statistical model that uses stochastic methods to generate music. In a VOMM, the generated music is determined by a probability that is based on the previous music. The VOMM then selects which note to choose next by a random or weighted random chance. *The Audience of the Singular* (29) uses a varaible-order Markov chain with shifting probabilities to generate its music.

6.2.3. Genetic algorithm

A Genetic Algorithm (GA) is modeled after the natural selection process as seen in nature. A GA begins with a set of randomly generated states. Each state is then evaluated by a fitness function. The fittest states are then combined using some process. This process can include random mutations as well [46]. Interactive Genetic Algorithms, which use a human subject to determine fitness, have been used in generative music systems [47]. We have identified one system, *MetaCompose* (30), which partially uses a GA to address the composition task.

6.2.4. Artificial Neural Network

An Artificial Neural Network (ANN) is an algorithm that is modeled after the human brain. ANNs are composed of neurons, connected by links. These neurons may be in a single layer, or may have hidden layers of perception and activation [46] ANNs are capable of forming complex statistical distribution models, and are differentiated from other stochastic methods because they form *connections* rather than *rule-based distributions* [48]. While ANNs are gaining popularity in MuMe, we have only identified one generative system for game music that uses ANN algorithms.

6.2.5. Other approaches

There are many other algorithms that can be used for generating music, and describing every possible generative algorithm for music is far outside of the scope of this survey. The listed algorithms are the most common algorithms in generative music systems, with almost all generative music systems for games using rule-based or stochastic algorithms. Further information on generative music algorithms can be found in an online class on *Kadenze* [49].

6.3. Musical representation

As mentioned, almost any AI algorithm can be used for generative music. While the algorithm that is used describes how the system manipulates its musical knowledge, the musical representation dimension describes how a system stores its knowledge of music. Because a generative system for game music will at some point produce audio, there are limits to the range of possibilities for the knowledge representation of systems. Surveyed systems either store their knowledge *symbolically*, and contain the ability to synthesize audio, or they contain *audio clips*, which are streamed from the storage media.

6.3.1. Symbolic

Systems that represent their knowledge symbolically can use any symbolic notation to represent the music. The most common notation in symbolic music representation for computing systems is MIDI, in which each musical event is represented by a series of variables such as pitch, velocity, channel, on/off, etc. *The Audience of the Singular* (29) represents music using MIDI values. *AOTS* (29) uses symbolic representation both in the corpus that supplies its knowledge, and in the representation of its output, which is then synthesized in real-time.

6.3.2. Audio

Systems that represent their knowledge with audio combine pre-existing samples of sound together into musical pieces. These systems are more common in industry uses of generative arrangement of music, where recording composed samples of music and arranging them adaptively has a similar work-flow to using composed adaptive music.

6.4. Musical knowledge source

While the algorithm dimension describes how a system manipulates its musical knowledge, and the musical representation dimension describes how a system stores its knowledge, the musical knowledge source describes where the knowledge originates. This knowledge can come from one of two sources:

6.4.1. External

Systems with external knowledge have their features, parameters, and values input by either their user or creator. The “user” in this case may describe the game’s player, but often describes the composer or audio designer, if they are not the creator of the system as well. In generative systems for games, external knowledge is exclusively provided by the system’s creator. One example of a system that uses

[2] “Species counterpoint” is a strict adherence to certain musical relationships in melody and harmony. It is generally used as a pedagogical method to teach melodic and harmonic writing [45]

external knowledge is seen in *Anarchy Online* (9). In *Anarchy Online* (9), each musical transition was hand-coded by the composers.

6.4.2. Learned

Generative systems may also take their knowledge from analyzing a corpus of musical input, and building a statistical model based on the input. Such systems may analyze either audio or symbolic data, though in the examined systems, only symbolic corpora have been used. *The Audience of the Singular* (29) builds a Variable-order multiple-viewpoint [50] Markov chain by analyzing a corpus of MIDI files. The MIDI files for *The Audience of the Singular* (29) were arrangements of music from the Super Nintendo Entertainment system and the Nintendo Entertainment system. The Markov chain for *The Audience of the Singular* (29) is learned a single time, before gameplay. During gameplay the system uses the existing, pre-trained Markov chain.

This concludes our taxonomy of generative music systems for games. In the next section, we will now examine the systems that fit within the scope of generative systems for games along this taxonomy.

7. Examination of musical systems

Table 1 shows the examined extant systems for generative music and audio in games. These games are listed in chronological order of release. It is important to note that while we have done our best to organize and collect data on these systems, in many cases the systems are used in commercial products. Because of this, the information concerning the systems is not always complete. Finally, for clarity we will simply be referring to the games that use a generative music system by the title of the game.

7.1. Composition systems

7.1.1. Horizontal composition

GhostWriter (1998) [51] The proposed academic education-oriented game *Ghost-Writer* (5) uses a musical system that maps in-game tension to musical tension using a rule-based system with random pitch selection. As far as we can determine, neither the musical system nor the proposed VR game were developed beyond the proposal phase.

As with many other academic musical systems, *Ghost-Writer* (5) attempts to map in-game activities to a dimensional model of affect [52], and specifically targets the dimension of tension. *Ghost-Writer* (5) does not attempt to use any automatic recognition of affect during gameplay, and instead uses a human director, who observes the gameplay and attempts to match the level of tension to the gameplay events. Because *Ghost-Writer* (5) has a stated design goal of acting as a classroom activity, Robertson assumes the presence of a teacher or facilitator who can mediate and navigate the experience and music.

Ghost-Writer (5) uses a three step process to generate it's music. The first step is the creation of a high-level form, though the authors do not provide further information on this step. Once a form is created, the system generates rhythmic data by selecting rhythmic "feet", based on the rhythmic feet of poetry. These feet are provided tension levels by the creators of the system. Once rhythms are generated, the system uses a version of Arnold Schoenberg's Theory of Harmony to create first a chord type (major, minor augmented, diminished). The system then creates a melody by choosing random pitches that fall within a set of constraints as decided by the harmony and rhythm. The system then creates an accompaniment using random generation within a simpler, more harmonic-focused set of constraints. Finally, the system assigns instruments to all parts, based on the provided tension level. The rules and constraints for each of these generative steps were hand-crafted by the system authors.

Spore (2008) [53] The generative system in *Spore* (13) automates both the composition task and performance task, using a modified version of PureData, called "EA-PD". *Spore*'s generative system was created in consultation with Brian Eno, a well-known pioneer of generative music. Theoretically, the music system in *Spore* could be used as a generic system, as it does not require game-specific messages or information, but publisher Electronic Arts has not published the system for external use. The system in *Spore* creates its music primarily by generating multiple independent musical phrases randomly, with randomness controlled via seed manipulation. This is a multi-agent approach to music generation [54], where simple individual generative agents combine to create complex combinations.

The music system for *Spore* is both reactive and linear, but not adaptive. The generation parameters are directly controlled by game state changes, but the music that is generated is the played back as linear music. *Spore* represents the only linear generative system that we are aware of. The mappings of the game state and generative parameters are externally provided by the composers and audio designers for *Spore*.

Little information is available about how *Spore*'s system addresses the performance task, though the creators describe the system's capability to apply Digital Sound Processing (DSP) effects in real-time [55].

One of the reasons that generative music is uncommon in video games is that generative systems are often CPU-intensive. While this presented a possible problem in *Spore* due to the large amount of other PCG that can have large CPU draw, the designers of the system solved this issue by mainly using generative music during parts of *Spore* where there is limited game logic, and therefore the CPU is more free to be used on music generation [55].

Cullimore, Hamilton, and Gerhard (2014) [28] The next system that we examine is an unnamed system from *Cullimore, Hamilton, and Gerhard* (23). While this system does address the composition task, it does not generate complete musical pieces. Instead, this system targets a weakness that is common in horizontal arrangement systems by composing short, chord-based transitions between musical cues.

One challenge in creating horizontal arrangement systems is that musical transitions can sound jarring if not properly handled [4]. Hand-creating transitions for each musical possibility requires a large amount of labor. Music may also be written for a horizontal arrangement such that any transition will sound acceptable, as in the case of *Red Dead Redemption* (16)'s musical system. Cullimore, Hamilton, and Gerhard's system attempts to computationally create chord progressions that can transition between any two bichords. The system encodes chords in a 2-dimensional space, with each state consisting of a pair of notes. The space is organized such that horizontal movement changes transposition of notes, and vertical movement changes the intervalic distance between the notes. The system is capable of creating chord progressions that link two chords together, though it is limited. The system exclusively generates bichords, and the authors do not mention whether transitions can be altered in case the game state rapidly changes. Also, this system seems to generate chord progressions without any sort of rhythm – in order to use this system in a game setting, some additional rhythm logic is required.

Engels et al. (2015) [37] Another unnamed generic system is presented by *Engels et al.* (24). The first and most basic model that Engels et al.'s system uses is a Markov chain. This Markov chain encodes musical events that occur at the same time together into a state. This has the effect of adding flexibility to the number of simultaneous voices that the system can play, as a state with a single note may lead to a state with a fully voiced chord.

Engels et al.'s system separates musical sections into different models. The Engels et al. system automatically segments music by using a support vector machine (SVM)[3] to group similar musical sections, based on pitch, duration, timbre, and volume. In addition to the Markov chain and the segmentation, Engels et al.'s system uses a Hidden

[3] A support vector machine is an algorithm that learns classifications from a pre-classified input dataset. The SVM then can use this data to classify new, unlabeled input data.

Markov Model, with chords as hidden states. These chords can be provided externally, or if not provided the system will attempt to automatically tag the chords. The hidden chord layer of the Markov chain is used to restrict individual voices so as to avoid clashes.

Prechtl (2016) [38] Anthony Prechtl created both a generative system and a game, integrating the system into the game for evaluation purposes. The game that Prechtl created is titled *Escape Point*, and is a abstract horror game.

Prechtl's system uses a Markov chain to compose a chord progression that loosely adapts to the level of tension during gameplay. As with *Ghostwriter* (5), this design is based off of Schimmack and Grob's 3-dimensional model of affect [56]. During gameplay, the system adjusts the probabilities for each chord based on the game state and tension level in the game. The tension in the game is represented by how close the player is to an enemy non-player character.

Prechtl's system has 12 parameter sets that control the generation of music. Seven parameters adjust the probabilities of each chord transition within the Markov chain. The other four parameters alter the performance of the music, altering the volume, velocity, timbral intensity, and the presence of a pulsing tone. The system uses two sets of parameter presets – one for low tension and one for high tension. The higher tension preset trends towards less major chords, more diminished chords, less tonal, and less diatonic than the lower tension presets. For the performance values, the higher tension presets are at a higher volume, velocity, timbral intensity, and pulse volume.

Prechtl's system is the only system from academia that we are aware of that has been evaluated in a video game setting. Prechtl found that for players with experience with games similar to *Escape Point* preferred the generative music to linear composed music. Prechtl also found that all players found the generative music more tense and exciting compared to the linear composed music. Finally, skin conductance responses were consistent with these findings, though Prechtl describes difficulties in analyzing the data. This provides both subjective and objective support of the strengths of generative and adaptive music.

The Audience of the Singular (2017) [57] *The Audience of the Singular*, or "*AOTS*", is similar to *Prechtl* (28)'s system in that it is a generative composition system that uses modified Markov chains to generate music. It is also similar to *Prechtl* (28)'s work as the developer also built a game around the system. *AOTS* has a symbolic representation of music that is learned from a corpus of 30 pieces of video game music from the late 1980s and early 1990s. The game for *AOTS* was built to interact with the music system specifically, but the music system itself is generic, and could function without the surrounding game.

AOTS uses a variable-order Markov for its core horizontal generation, which is learned from the corpus offline. The VOMM creates four versions of five musical lines, one phrase at a time, starting with the bass line. *AOTS* uses four different VOMM models, which are independently learned by analyzing selected musical phrases from the corpus. For all lines, a rhythm line is first constructed. The VOMM organizes rhythms into beats, with 1–4 16th notes per beat. The probabilities for beats are based only on the previous chosen beats. Each beat also may end with a tie, allowing for syncopation.

The bass line is also the most simple version of the markov chain – the bass line probabilities are based exclusively on the previous notes and the length of the phrase. As the phrase approaches a length of 16 measures, the bass line begins to weigh more heavily towards functional cadence relationships. The system then composes a primary melody whose probabilities are based on the previous notes, a learned melodic contour based on phrase length, and the notes of the bass line. A secondary melody is then constructed similarly to the primary melody, though the probability spread is heavily changed by the distance to the melody. Finally, the system composes two harmony lines whose probabilities are based on the previous notes, bass line, and melody lines. The secondary harmony line also considers the primary harmony line. Once the generation is complete, *AOTS* selects a drum part from several composed grooves. To create the variation lines, *AOTS* removes notes and extends durations of notes that occur on weaker beats, with the final variation containing long notes that begin on primarily strong beats. All of the probabilities for the VOMM are learned from the corpus.

7.2. Arrangement systems

7.2.1. Horizontal arrangement

Ballblazer (1984) [58] The earliest use of generative music in video games is 1984's *Ballblazer* (1). Ballblazer uses a generative technique that the creator calls "Riffology" [59], which is a catch-all term that can be applied to any system that uses constrained random selection of notes to generate music. *Ballblazer* (1)'s generative system is unique in that it addresses all three generative tasks on grid. The system arranges accompanimental music horizontally at a phrase granularity, and it composes and performs musical melodies at a note level of granularity.

The system is *Ballblazer* provides music between games and in menus, but does not provide music during gameplay. The system has a corpus of 16 measure melodic fragments (called "riffs" by designer Peter Langston), and 4 measure accompaniment sections containing a bass line, drum line, and chords. These corpora of melodic fragments and accompaniment sections are provided by the system's designer, and contain notes exclusively drawn from an aeolian scale based on a (a natural minor), with an added ♭5.

Details on the generation of the 4 bar accompaniment sections is limited, though Langston describes the logic as a "simpler form" of the melodic system. The melody composition generates several possible "riffs", or collections of randomly-selected notes within the provided scale. The system then chooses the riff that begins on a note that it closest to the note that ended the previous riff. Once the riff is selected, the system performs the riff by choosing to omit notes depending on variables, as well as determining the speed and volume of the performance.

iMuse (1991) [60] Surprisingly, the generative system with the largest effect on the state of the art for generative arrangement systems in games was created in 1991. *iMuse* and systems that extend its design by and large represent the state of the art in the games industry [33]. Because *iMuse* is a generic system that has been integrated into several games, it cannot be evaluated as a system along gameplay dimensions, though it is most commonly used to provide linear, ambient, non-diegetic music.

At its most basic functionality, *iMuse* plays a piece of music that is stored symbolically. If there are no game changes, *iMuse* will play the music as written, linearly. When there is a game state change, *iMuse* attempts to seamlessly transition the music to fit the new game state. This can be done by ending the music, or by transitioning between two musical pieces. Peter Silk created a short video that demonstrates *iMuse* transitioning between two musical pieces seamlessly, available at `https://bit.ly/1R39FPY` [61].

iMuse is, importantly, an arrangement system. This means that it does not compose new music, but instead resequences and alters composed music. For *iMuse* to create seamless transitions, musical content must be composed to enable the seamless transitions. This means that the music composed for *iMuse* must be in complimentary keys, at similar or complimentary tempi, and cannot use extended harmonies such as modal borrowing[4]. Also, the musical library, or the music used in *iMuse* must be annotated by hand to provide the system with information on where the music may transition.

Munge/MNG (1998) [24] *MNG* (6), pronounced and sometimes written as "Munge" is a filetype that stores generative music instructions for the games *Creatures 2*, *Creatures 3*, and *Docking Station*. These

[4] Modal borrowing is a compositional technique where notes or chords from a parallel mode are used. A common example is the use of a borrowed flat VI chord from the parallel minor of a major key

files contain a variety of rules and game states that correspond with an external corpus of audio music. The system in these games (which we will refer to as *Munge* for simplicity) interprets this data to address the arrangement task in a horizontal directionality. We are unable to determine the granularity of the system, though it is capable of both on grid and off grid generation. *Munge* (6) generates music that is non-diegetic, ambient, and adaptive. The system is generic, and uses a rule-based algorithm with external knowledge source and an audio representation of music.

The *MNG* filetype allows a programmer to set individual voices with musical variables, based on in-game variables, as well as randomness. Each *MNG* file is associated with a specific level of a game. As the game state changes, *Munge* receives as input a "mood" variable. The system then arranges the music based on a combination of the mood variable and randomness, according to the ranges and variables set up in the associated *MNG* file for the current level.

No One Lives Forever (2000) [62] *No One Lives Forever (NOLF)* (7) uses a system that refines the *iMuse* design. While *iMuse* (3) functions at a measure level of granularity, *NOLF* can transition between musical cues on any beat.

The primary refinement from *iMuse* to *NOLF*, beyond the granularity change, is that the system in *NOLF* can also alter the pitch and tempo of its musical library. This reduces the restrictions on the music, as the system can alter the tempi and keys of musical transitions to avoid jarring transitions. One weakness of this approach is that the musical library requires additional annotations to work within the system. While the system is capable of matching tempi and keys, it cannot automatically detect either. Essentially, the system in *NOLF* provides better flexibility and more adaptivity in the music, but at the cost of increased data entry labour.

Diner Dash (2004) [63] *Diner Dash* (10) uses a simple system to generate music with large amounts of variety from a small amount of composed music.. In *Diner Dash*, each musical cue is separated into phrases, any of which can lead to another phrase within the same cue. During gameplay, the system plays the phrases in a random order, based on the cue, which in turn is based directly on the game state.

Uncharted 2: Among Thieves (2009) [64] *Uncharted 2: Among Thieves* (15) uses a very simple generative arrangement system, that targets one specific problem in game music. In most games, when the player fails a gameplay segment or challenge, they return to a previous point in the game and attempt the challenge again. In games with linear composed music, this often re-starts the musical piece that is associated with the current game state. The music in *Uncharted 2* has multiple possible starting points, provided by the composer. When the player fails a gameplay segment, they are returned to the checkpoint, and the musical system selects a random starting point in the music. This avoids the player hearing the exact same music in the exact same way multiple times.

Nln-player (2011) [65] The *Nln-(Non-LiNear)-player*, which is demonstrated in the game *Shortburst*, uses a simplified version of a Markov chain to address the arrangement task. Most uses of Markov chains for generative music store individual notes or chords as single states to address the composition task. However, the *nln-player* uses a "cell-based" design inspired by a Markov chain where each state is a composed musical phrase, and transition matrices are unweighted. Essentially, the *nln-player* uses a Markovian design to ensure that transitions between states will not have any jarring or unexpected transitions. Within any game state, this design essentially means that the music is pseudorandomly shuffled, similar to other horizontal resequencing techniques [18,4].

The nln-player's design necessarily involves restrictions on the music that is composed for it. Phrases that can transition to each other must be consistent in key, tempi, and mode, as in *iMuse*. Additionally, the composer must hand-annotate a configuration file within a specific metadata format. Also, the system can only interpret specifically formatted file-names.

DOOM (2016) [66] As with many horizontal arrangement systems, the system in *DOOM (2016)* (27) presents another refinement on the architecture of iMuse, and arranges composed musical phrases together based on both the game state and the surrounding musical context.

DOOM's generative system is similar to the designs presented in the *nln-player* (18), *The Audience of the Singular* (29), and *Engels et al.* (24), in that *DOOM* draws from multiple corpora separated as musical phrases. *Doom*'s corpora divisions are based a standard song structure of intro-verse-chorus-bridge-outro. Each phrase type has approximately 30 potential composed phrases. During gameplay, the system determines which phrase type to play based on the surrounding gameplay. The system then choses a musical phrase at random from the corpus.

There are two notable aspects of the music system in *DOOM*. *DOOM*'s gameplay and music are built to be complimentary, with the flow of the game matching a standard song form, according to composer Mick Gordon [67]. This facilitates the design of the system, as the possibility space for the music is also contained.

The second notable aspect of *DOOM*'s system is that, similar to other horizontal arrangement systems, the music is heavily constrained for the composer. As seen elsewhere, because *DOOM* allows for any phrase to transition into any other phrase, and cannot alter the pitch content or rhythmic content of its corpus, all pieces must be written in complementary keys and tempi. *DOOM*'s system does not require files to be annotated as fully as other examined systems such as the *nln-player*.

Horizontal arrangement systems allow for music to more seamlessly transition between otherwise different musical states, as exemplified by *iMuse* (3) and *NOLF* (7). These systems also allow for greater variety in the musical content, as seen in *DOOM* (27) and the *nln-player* (18). We will now focus on systems that use vertical arrangement, which is sometimes referred to as "adaptive stem mixing" or "vertical layering".

7.2.2. Vertical arrangement

We have identified two systems that exclusively target vertical arrangement. Both of these systems provide increased musical variety, and both do so with less compositional restrictions than horizontal arrangement systems require. Because these systems share many elements of function, we examine them together.

Halo series (2004–2018) [68], *Dark Void (2010)* [69] The *Halo* (11) series share a music system. While the system has evolved as the games evolve, the core design remain consistent. *Dark Void* (17) uses a similar system. Both of these systems add or remove groups of instruments in real-time, which lets the player hear a single piece of music with many different arrangements, reducing the fatigue from repetition.

The key difference between the systems of *Halo* (11) and *Dark Void* (17) is their granularity. While *Halo*'s system can create much more variety due to the number of controllable instruments, *Dark Void*'s system provides more control to the composer, as the composer can decide which instrument groups will sound better together.

An advantage to using vertical arrangement is that the music can be written without any horizontal restrictions. The only restriction that vertical arrangement systems place on the musical corpus is that during any piece of music, any collection of parts must be able to play together. This is a minimal restriction, as most instrument parts are composed together in music by default. Also, unlike horizontal arrangement systems, there is only limited annotation required for the musical corpus, as different instrumental parts of a single composed musical piece can generally be assumed to play in the same tempo, with complementary notes playing at any given time.

7.2.3. Mixed arrangement

DirectMusic (1996) [70] *DirectMusic* (4) presents a major improvement on the *iMuse* (3) design. As with *iMuse* (3), because *DirectMusic* is a generic system, it cannot be evaluated along gameplay dimensions, but it is most often used to provide ambient, non-diegetic music. Unlike *iMuse* (3), *DirectMusic* is capable of adaptive music, and can generatively arrange in the vertical dimension.

DirectMusic can simultaneously play multiple musical sources, and reads from MIDI, WAV, and a proprietary format that contains both symbolic data and a wavetable for synthesis, similar to MOD files. *DirectMusic* can individually add or remove parts from any filetype, allowing for vertical arrangement. In a departure from the *iMuse* design, *DirectMusic* can also alter MIDI events, allowing it to alter pitch and tempo, and can apply DSP effects such as reverb. *DirectMusic* can also spacialize audio, providing 3d sound. These capabilities provide adaptive audio e.g. a single symbolic musical cue may change between a major and minor mode based on the game state.

Anarchy Online (2001) [71] Anarchy Online (AO) (9) uses a system called the "Sample-based Interactive Music tool", or "SIM tool" [72]. The *SIM Tool* implements a Marovian algorithm similar to the *nln player* (18), where each state represents a short musical phrase.

Anarchy Online is a Massively Multi-player online game, a type of game where thousands of simultaneous users interact with a persistent world. These games generally have high amount of game content, with players regularly spending over 100 h in the game world. This high gameplay length exacerbates a problem of linear composed music – the player may hear single musical cues multiple times, resulting in boredom [4,72]. As far as we are aware, *Anarchy Online* is the only game of this genre to use generative music to address this concern.

Music composed for the *SIM tool* is split into short clips, and each clip is individually sampled to create consistent reverb trails. The clip must be annotated with tempo, meter, "layer" (an associated game state), and a transition matrix to other clips. Clips contain three to five transitions without switching layers, and "a fair number" [72] of transitions to other layers. Layers represent both horizontal and vertical responses to game states. Audio Designer Bjørn Lagim provides an example of both – as the player moves between environments, the system will transition between layers, arranging music horizontally to differentiate the environments. When the player engages in combat, the *SIM tool* adds or removes layers from the 14 available combat layers, based on the relative health points of the player and their opponent, as well as the size of the opponent, arranging music vertically to adapt to the gameplay.

Lagim describes several drawbacks to the *SIM tool*. He notes that the clips used in *AO* are a few seconds long, and that they are only able to transition at certain points. This can cause musical transitions to occur well after the associated gameplay state change. The *SIM tool* can crossfade between clips during playback based on chord progression, though this was not implemented in the game. Another drawback to this tool is that the composer must provide the annotations for each clip by hand. If a clip is added, other clips that may transition to the new clip must be updated to include the new clip. If a clip is removed, clips that do transition to the removed clip must also be updated by hand.

Tom Clancy's EndWar (2008) [73] Tom Clancy's EndWar (14) uses a common mixed arrangement system design, one that is similar to the more recent systems used in *Red Dead Redemption* (16) and No Man's Sky (26). The music in *Endwar* is divided into individual short musical phrases, described internally as "cells". Each cell is placed into a corpus, described internally as a "pool". Within each corpus, any phrase may transition to any other phrase upon completion. The system may also adjust the density of the music by adding a constrained random pause between each phrase. This pause can be individually set for each corpus during runtime.

The system generatively arranges music horizontally by adding or removing individual corpora from the total mix, based on game state and the musical context. When a layer is removed, the system allows the current musical phrase finish playing within that layer, rather than cutting off the music. The corpora that are playing at any point are determined by the game state, which allows the system to adaptively alter the arrangement of the music.

Designer Ben Houge describes several drawbacks and constraints of this system. Houge notes that the technology for reading music off of a DVD directly was too slow in 2008 to have multiple corpora of short phrases seamlessly transition and play. This required the system to load the music into RAM, which is normally allocated to game levels and textures. Also, Houge describes the workflow of the system, which involved numerous iterations and mockups to create music for the system, as well as what Houge calls "significant time" tweaking parameters in game context [74].

Red Dead Redemption (2010) [75] The system in Red Dead Redemption (RDR) (16) presents the most extreme form of systemic flexibility at the expense of restricted expressive range. While *RDR* uses a rule-based algorithm, its architecture can also be understood as a Markov chain in which any state can lead to any other state. Music in *RDR* is divided vertically by instruments, and each instrument has an associated function e.g. "Melody" or "Bass". Within any game state, the system creates music by randomly selecting one musical phrase from each function's associated corpus. When the game state changes, the system crossfades to a new randomly selected group of phrases from the associated corpora.

This design presents an extremely flexible system, as there are no restrictions on how the corpora can be combined. However, this design presents severe restrictions on the composed library. All of the music in *RDR* is composed in the key of a minor, at a tempo of 130 beats per minute (bpm). The music does not contain extended harmonies or modal borrowing. In short, because the system can combine any collection of musical phrases together, all music composed for the system must combine well with all other music composed for the system. This severely limits the expressive range of the system.

No Man's Sky (2016) [76] No Man's Sky (26) uses multiple similar generative systems to create multifaceted generative audio. The systems in *No Man's Sky* address the tasks of world audio playback similar to *Sonancia* (25), the generation of alien-sounding speech, and the generative arrangement of music, though we narrow our focus to the music generation system.

The corpus for *No Man's Sky* is a composed score. The band "65daysofstatic" composed a linear score for the game absent the generative system. The composed music is divided into small clips of music for the generative system, and each musical piece acts as a single corpus. Because the elements of each corpus are sourced from composed music, the composed clips can be assumed to fit together musically. This does require the music to be composed without key or tempo changes within each piece, but it also reduces the need for hand annotations of music.

No Man's Sky's generative system differentiates music horizontally based on five associated game states: "Wanted", "Space", "Planet", "Map", and "AmbientMode". Each game state has associated musical pieces, which are pseudorandomly assigned to play together. Details of the rules and restrictions that govern the combinations of musical elements is unavailable. The system arranges music vertically based on a random procedural playback of each location in-game. The instrumentation depends both on the location of the player and where the player is looking [5].

Adaptive Music System/AMS (2019) [77] The Adaptive Music System (AMS) (34) has similar design elements to other academic generative systems, such as *MetaCompose* (30) and *The Audience of the Singular* (29). AMS alters pre-composed music based off of an affective mapping, that is taken from the game state. It does this using a combination of rule-based algorithms, genetic algorithms, and a Recurrent Neural Network (RNN). An RNN is a class of Neural Network that maintains past information while receiving new information.

AMS extends a categorical model of affect from music perception literature, with 6 affect categories: happiness, fear, anger, tenderness, sadness, and excitement. To connect the game state to the affective data, a model of "spreading activation" is used. This model represents affect and game concepts as weighted verticies in a 2-dimensional plane. When a vertex activates, it also activates nearby vertices. In *AMS*, because game concepts and affects are both on the same plane, when a game event activates, it will also activate the nearby affective vertices,

which influence the music generation.

To generate music, *AMS* uses a multi-agent approach with three agent roles. The first agent role is the "harmony" role, which builds a chord progression using an RNN algorithm trained on a symbolic corpus. This agent does not generate notes, but as in *The Audience of the Singular* (29), the harmony information is used to constrain the output of multiple melody agents. The melody agents use a rule-based approach to alter pre-composed musical pieces. The rules are created offline using a supervised genetic algorithm, in this case trained by a single expert composer. Finally, *AMS* creates a percussive line with a single agent that uses a similar RNN approach to the harmony agent.

One element that sets *AMS* apart from other academic systems is that while it is generic, it has been evaluated using real-world games. As part of an evaluative study, *AMS* was integrated into an open-source Zelda clone titled *Zelda: Mystery of Solarus* as well as the Real-time strategy game *StarCraft II*. A correlational analysis of the games with *AMS* and with their original score found that *AMS* significantly, if slightly, increases player immersion.

7.3. Performance systems

While composition and arrangement systems automate the creation of new music, performance systems automate the interpretation and playback of music. *Otocky (1987)* [78] *Otocky* (2) is one of the first examples of a game using generative music system. The gameplay of *Otocky* is a horizontal "shoot-em-up" or "shmup", similar to the *Gradius* games. As the player progresses, they collect various upgrades, each of which shares a name with an associated synthetic instrument. When the player fires their weapons, the associated instrument plays alongside the composed linear soundtrack, with pitch determined by the soundtrack and rhythm quantized to the nearest eighth note.

Rez (2001) [79] *and Child of Eden (2011)* [80] *Rez* (8) and it's prequel *Child of Eden* (19), have nearly identical gameplay, and use identical musical systems. While these systems are almost identical in design, they are not generic systems, but are nearly identical game-specific systems.

The systems in *Rez* and *Child of Eden* are very similar to the system used in *Otocky* (2). *Rez* and *Child of Eden* are third-person, 3d shmups that are on-rails (the player does not control the motion of their avatar). The player controls the location of a reticle on screen, and when they move the reticle over an enemy, they lock on to the enemy. When the player presses a button, their avatar fires its missile-like weapons, which automatically track and hit the locked-on enemies. When the missiles hit the enemies, a pitched cluster of notes is played. The exact timing of the notes is quantized to be on grid, and the timbre is based the player's performance.

Bit.Trip Runner (2011) [81] *Bit.Trip Runner* (20) uses a system that is nearly identical to the system in *Rez* (8) and *Child of Eden* (19), though it is simplified. A difference in *Bit.Trip Runner*'s system is that, similar to *Otocky* (2), the system provides adaptive music.

Bit.Trip Runner (20) is an infinite runner game, where the player character moves at a constant speed, and the player must take action to avoid oncoming obstacles and to pick up power-ups. When the player jumps or slides, a note is randomly selected from a pentatonic scale that matches the surrounding musical context. This note plays at the next available beat. The instrument that plays the note is directly related to the number of power-ups that the player has collected.

Chuchel (2018) [82] *Chuchel* (32) is unique among the surveyed systems because the diegetic, sourced music that system performs also functions as sound effects and even as character voices. When the player interacts with objects in *Chuchel*, the object almost always produces a sound. There may also be an auditory reaction from the player character. In many cases, these sounds are pitched, and the pitch of the sound is determined by the surrounding musical context. In *Chuchel*, unlike *Rez* (8), *Child of Eden* (19), or *Otocky* (2), this system has a low degree of player control over the music. While actions in the previously mentioned performance systems always respond musically, in *Chuchel* the musical nature of sounds is unpredictable and inconsistent.

Ape Out (2019) [83] The music system in *Ape Out* (33) is unique in our surveyed systems as the game soundtrack and the musical output of the system contain no pitched music at all. Instead, the music for *Ape Out* is provided exclusively by a virtual drummer. The player can interact with the game world in only two ways – grabbing a non-player character/object, and throwing the NPC/Object. When the player throws an NPC into a surface, the NPC explodes in gratuitous violence. During gameplay, there is a constant drum groove that plays. When an NPC is killed, there is also a cymbal accent.

The drum groove is selected based on the current gameplay level, the amount of on-screen action, and random chance. The generative system contains a library of 1000 short drum grooves and cymbal hits. Each level of the game has an associated library of drum groves. The drum grooves are also delineated by what the developers describe as "level of action". As the action in *Ape Out* becomes more intense, more active drum grooves play. The cymbal accents are chosen randomly from a library of cymbal hits.

7.4. Fringe systems

While the previously examined systems have well-defined generative tasks, we have also identified seven systems that do not fit as cleanly into our taxonomy. These systems can be described using our taxonomy, though either they fill multiple roles, or they approach music in a unique way that is not fully captured by the taxonomy. We describe these systems as fringe systems as they exist on the fringes of our taxonomy.

Agate/AGMS (2008) [84] *Agate* (12), also called *AGMS*, is a system that simultaneously addresses the composition and arrangement tasks, with both a horizontal note granularity and a vertical instrument granularity. The system has not been integrated into a game and therefore cannot be described with gameplay dimensions, though it does play linear music. *Agate* is designed as a generic system that uses a rule-based algorithm. *Agate*'s musical representation is both symbolic and audio, though the knowledge source is exclusively external.

Agate organizes its music in libraries and rule sets that are associated with "moods". *Agate's* moods are provided by an external source, and are attached to a game state, rather than being based in a more general representation of affect. *Agate* combines two simultaneous rule-based algorithms to address both the arrangement and composition task. *Agate* generatively addresses the composition task with a set of rule-based constraints on otherwise random generation. The designer can select a collection of notes that may be used, the level of randomness, the beats that notes may play on, the possible durations of notes, and the available instruments. *Agate* then creates ambient soundscape music by randomly selecting pitches.

For the arrangement task, *Agate* also uses constrained random generation. The composer or designer provide either symbolic or audio representations of short musical phrases. The designer also sets parameters that constrain the activity level of the samples. *Agate* plays these phrases at random times over the ambient soundscape that is generated, constrained by the activity level.

Sonancia (2015) [85] As with *No Man's Sky* (26), *Sonancia* (25) uses multiple different generative algorithms to generate a level, populate the level, and add audio triggers to locations in the level. We focus our examination primarily on the music generation aspects of *Sonancia* (25).

Sonancia is a game that generates a horror-game level to match a provided or machine-generated tension curve. Once the level is generated, the musical system distributes musical cues throughout the game environment to match the generated level's tension curve. Each musical cue is annotated along Schimmack and Grob's 3-dimensional model of affect [86]. The musical cues are distributed into each room in the generated level via the previously mentioned rule-based algorithm.

The selection method for the music can be chosen by the designer from one of four provided methods: "Hall of Fame", which selects the top *n* pieces that match along a single emotional dimension, "Equidistant", which selects *n* pieces based on their ranking within the chosen emotional dimension, "Granular", which selects the closest emotional match to the generated room, and "Random", which selects randomly.

A core difference of *Sonancia*'s generative system, compared with other systems that require musical annotation, is that annotation for *Sonancia* does not need to be provided by an external source. *Sonancia*'s initial corpus of music is annotated via crowdsourced ranking. Support Vector Machines are then trained on the user provided annotations and a selection of audio features. The SVMs are then used to annotate new musical files based on the same selection of audio features. This use of SVMs in the prediction task allows for the automatic annotation of music files, reducing the labour of the standard arrangement-oriented pipeline.

MetaCompose (2017) [39] *MetaCompose* (30) uses two different systems to address both the composition and performance tasks. The component sub-systems of *MetaCompose* differ not only in the task that they address, but also the generative algorithm that they use.

The first system uses a combination of a stochastic algorithm, a genetic algorithm, and a rule-based algorithm. The first system begins its generation by generating a chord progression using a random walk on a directed graph of possible chord transitions. This creates chord progressions which over time resolve to the I chord. Once a progression is created, a genetic algorithm evolves a melodic line. This genetic algorithm uses a fitness function that is provided externally. The fitness function's design mirrors species counterpoint, with restrictions on large leaps, and tendencies towards chord notes during leaps and after chord changes. Finally, this composition system creates a framework for accompaniment. The accompaniment system uses two rule-based components to create a rhythm and arpeggio for accompaniment. This system uses Euclidian rhythm to create even and repeating divisions of time, and selects from pre-composed arpeggios to play the chords through time.

The second system in *MetaCompose* targets the performance task at the instrument and note levels of granularity, using a rule-based system. This final performance system takes the previously generated music, and alters and synthesizes the music according to provided Valence and Arousal values. The system directly links arousal to volume, and valence to brightness of timbre. Additionally, this system chooses an accompanying drum line to accompany the music. This drum line is more prominent and faster as arousal increases, and more regular and steady as valence increases. Finally, this system alters notes of the provided composed music, adding dissonant tones from alternative musical modes as valence decreases.

Melodrive (2018) [23] *Melodrive* (31) has limited information available, as it is an in-development system from the games industry. Because *Melodrive* is a generic system, it cannot be discussed along gameplay dimensions, though it is intended to provide adaptive music. *Melodrive* uses a symbolic representation of music with an external knowledge source, though we cannot determine the algorithm for *Melodrive*. *Melodrive* is differentiated from the other surveyed generic music systems by being integrated into the *Unity* game engine. This means that *Melodrive* can be integrated into any game built using the *Unity* engine without requiring large amounts of labour to port the system to a new engine or project.

Melodrive (39) is available as a Unity plugin, and presents a simple interface for designers. To generate original music, the *Melodrive* script must be given a style and emotion. Both of these options are categorical, with the set of possibilities determined by *Melodrive*. *Melodrive* can also interpolate between different emotions, and Russell's 2-dimensional model of emotion [87] may also be used. *Melodrive* creates fully-featured music without requiring large amounts of additional labour or musical restrictions. However, *Melodrive* also does not offer the customization possibilities in more open-ended generic systems such as *DirectMusic* (4).

8. Tools for adaptive and generative music

The two most popular publicly available video game engines in the game industry are the *Unreal* engine and the *Unity* engine. Another less popular publicly available game engine is Amazon's *Lumberyard*, based on Crytek's *Cryengine*. Large game companies often use an internally-developed game engine to create their games, such as Electronic Arts' *Frostbite*, Ubisoft's *AnvilNext*, Square-Enix's *Luminous* engine, Bethesda's *Creation* engine, and Rockstar Game's *Rockstar Advanced Game Engine*. The engines that are in use by large companies generally do not publish their specifications. In both *Unity* and *Unreal*, each audio asset is attached to at least one object in the game. To trigger an audio asset, the object must call the audio asset from code. This design does not easily allow for generative music, as the number of managed audio assets would easily become unfeasible. *Lumberyard*, as far as we are aware, does not have any audio rendering capabilities built in. As far as we are aware, most internal industry game engines have similar audio capabilities to *Unreal* and *Unity*. External audio solutions and tools can be used to facilitate generative music in games.

8.1. Audio middleware

Audio Middleware engines provide a solid base for interactive and adaptive audio. Middleware can be used to fill any of the musical and gameplay dimensions of our taxonomy. However, middleware engines are limited in architecture dimensions – currently they are only capable of creating systems specific to a game, that use a rule-based algorithm with an external knowledge source, and audio representation of music.

Audio middleware engines seamlessly integrate into game engines, and facilitate adaptive music and audio. The two most popular middleware engines are Firelight Technologies' *FMOD Studio* [88], and Audiokinetic's *Wave Works Interactive Sound Engine (Wwise)* [89]. Both middleware engines share similar functionality.

FMOD and *Wwise* act like traditional DAWs for audio editing, but with additional controllable parameters. These additional parameters can be any numeric or boolean game data, which is passed via an API call. These parameters can be used to add, remove, or alter audio effects, including volume, DSP effects, and spacialization. Middleware can also loop sections of music until game parameters are changed.

FMOD and *Wwise* can both also use indeterminacy for selecting clips. A standard implementation of this feature in non-musical audio is to provide variety in commonly-heard sounds. A single footstep that is repeated every time the PC takes a step will get grating. By creating a corpus of possible footstep sounds and randomly selecting one each time the PC takes a step, the resulting sounds are less repetitive and more believable. A musical example of this can be seen in *Tom Clancy's Endwar* (14), where the system shuffles musical phrases based on randomness, taken from a corpus of possibilities.

These capabilities, when used together, facilitate generative systems that address the arrangement task. Generative arrangement systems that use middleware generally require the same musical restrictions as seen in many of the surveyed systems: Any layers that intend to play together must be composed at identical tempi and keys as each other. Additionally, any clips that may randomly play alongside each other must fit together musically.

8.2. iMuse

iMuse (3) has been discussed previously due to the consistent use of the system in games. We now describe *iMuse* as a tool that can facilitate generative music. *iMuse* is most easily used for horizontal arrangement on-grid, and can fill any gameplay dimensions. *iMuse* can only create music from external sources, with symbolic representation of audio, using a rule-based algorithm. While *iMuse* could theoretically be used

for vertical arrangement, but this would require considerable work and the software is not designed for vertical arrangement. *iMuse* could also theoretically create adaptive music, but it's design is most suited to linear music.

8.3. DirectMusic

As with *iMuse* (3), *DirectMusic* (4) also functions as both a system in games and a tool to facilitate generative music. *DirectMusic* extends the *iMuse* possibilities, and can be used to address the arrangement task in both horizontal and vertical directionalities. *DirectMusic* is most commonly used on grid, but this is not a necessary part of its design. Systems that use *DirectMusic* can fill any gameplay dimensions, though must use a rule-based algorithm with an external knowledge source. *DirectMusic* can use both symbolic and audio representation of music.

8.4. PureData

Miller Puckette's PureData (PD) can be used as a generative music tool in games, in a limited capacity. PD is a visual programming language that targets real-time audio generation. PD can be used in a system that fills any musical, gameplay, and architecture dimensions of the taxonomy. Electronic Arts used a modified version of PD called "EAPD" for the generative soundtrack in *Spore* [53]. PD was also used to synthesize the music of *The Audience of the Singular* (29). Unfortunately, despite the potential strengths of PD, there is no official support for PD implementation in any game engine that we are aware of, and external libraries are often outdated and unstable.

8.5. Other languages

Real-time coded generation of music, such as *csound* and *Max 8*, can also be used to synthesize and perform generated music. *Chunity* [90] and *uRTcmix* [91] are examples of real-time audio/music languages that can be used as plugins for *Unity*. These plugins allow for easy use of audio generation functions for audio playback and synthesis of primitive waves. We have found no instances of these languages being used for generative music in games, though as with the other synthesizers, these languages could be used to synthesize music from any symbolic representation, or to sequence any audio representation of music.

8.6. Custom synthesizers

Both the *Unreal* engine [92] and *Unity* [93] allow programmers to access the audio data directly. This allows a designer to programmatically build synthesizers directly into the engine. These synthesizers may then interpret symbolic music data into musical sounds. Because these synthesizers act only as a playback device, a system that uses custom synthesizers may fill any dimension of the taxonomy. Perhaps because of the complexity of programming synthesizers from scratch, we find no examples of this being used in either the academic research or industry use. The *Unreal* engine contains very basic waveform synthesizer blueprints as samples, though we are not aware of any industrial or research game that uses these synthesizers.

8.7. Open Sound Control

Open Sound Control (OSC) is a protocol for computer communication. OSC can also be used locally, sending data from different programs on the same machine. Because of this, OSC can allow for programs such as Ableton Live or Max/MSP to provide audio for games. As before, because OSC is primarily a tool for communication, rather than containing any algorithm itself, a system that uses OSC may fill any role in our taxonomy. *Audioverdrive* (22) uses OSC to allow for interaction between level generation and music. We are not aware of any examples of commercial use of OSC. Most likely, this is because OSC cannot be integrated seamlessly as middleware engines can. During gameplay, a system using OSC must still be running an external program as well. Games generally run from a single executable, and players are not expected to follow long setup processes to play a game. This may be why OSC is not used in the games industry.

9. Discussion

9.1. Analysis of trends

We identified 34 systems that fit the scope of generative music in games. We acknowledge that this list may be incomplete, as public information on industrial games is limited. We again acknowledge that our taxonomy and descriptions are based upon the use of Western music theory as a descriptive lens as discussed in Section 3, and that other ethnomusicological lenses or descriptive tools may find alternate understandings of these systems.

We draw attention to our narrow scope of generative music in games. We do not discuss games with highly adaptive non-generative music, such as *Final Fantasy XV* [94]. We also do not discuss games where user-selected music is used to procedurally generate a game level, such as in *Audiosurf* or *Beat Hazard*. We also do not discuss "audio games" [95], where music and audio are used as the primary method of conveying information to the user. Finally, we do not discuss the "music game" genre of games where music is mechanically important, including games like *Fract OSC* or *Rock Band*. While these games all allow for interaction with music, our scope is limited to games that use a generative music system with some level of systemic autonomy, and we have not identified any games within these genres that use generative music.

The systems from the games industry have many commonalities with each other. These systems generally address the arrangement task on grid. They provide non-diegetic, ambient music. These systems generally are specific to their game, and use a rule-based algorithm with an external knowledge source and audio representation of music. We believe that there are several reasons for these common trends. As discussed, audio middleware is very common in the games industry, and are capable of a rule-based algorithm with external knowledge source and audio representation. Most game music is non-diegetic and ambient, and the source of the generation does not necessarily have an impact on the gameplay dimensions. Finally, creating on-grid arrangement systems most closely matches the workflow of using composed music in games.

The systems from academic research generally address the composition task with mixed directionality. These systems also primarily provide non-diegetic ambient music, and generally are adaptive. Academic systems are far more varied than industry systems in the architecture dimensions, with systems using stochastic, rule-based, and genetic algorithms, with both symbolic and audio representation, and both learned and external knowledge sources. While this does indicate that academic systems are more technologically advanced, it is important to recognize that many of these systems have not been integrated into or evaluated within actual gameplay. The academic systems also do not target a commercial release, which means that they can produce music that does not sound as "good" as a human composer, without affecting commercial success.

Generative music systems for games have trended towards audio representation of music, especially in the industrial applications. This is most likely due to an assumed dislike of MIDI sounds in the audience [96], and the higher fidelity and quality of audio representation. However, we do note that award-winning games such as *Shovel Knight*, *Celeste*, and *Luftrausers* make heavy use of synthetic instruments.

9.2. Conclusion and suggestions for future work

Generative music for games is becoming increasingly commonplace,

and is advancing quickly. Of our 34 surveyed systems, 19 of them are from the 10 years prior to this writing, while the remaining 15 are from the preceeding 26 years. However, there is still much room for advancement in the area. Current systems tend to fall into two main categories: Simple and effective systems, which are more common in industrial applications of generative music, and more advanced but untested systems, which are more common in academia.

We believe that the future of games music will involve increasing use of generative techniques. Generative systems can provide a greater variety and adaptivity to music than is possible with composed music, and with less required labour. Generative systems can also create endless amounts of music, which is well suited to longer-duration games. Generative systems can also provide large amounts of variety, which is particularly useful in run-based games.

There is a valid concern that generative music may cause harm to video game composers by rendering their work unnecessary. We note that this concern is not reflected in the current implementations of generative music, which require either a library of curated or provided music, or musical expertise in the design of the system. Current AI techniques for generative music also often require corpora of composed music to be effective. We believe that human-composed music is still capable of greater expression than computer generated music, especially when the game activity is predictable, and suggest that future work in this area continue to leverage the strengths of both human-composed and generative music together.

As we have discussed, generative music systems from the game industry generally address the arrangement task. We have identified two main weaknesses in the current state of the art for these systems. These weaknesses are linked together, and any attempt to rectify one weakness within current paradigms exacerbates the other. The first weakness is that the generative music systems are often highly restricted in expressive range. *Red Dead Redemption* (16) demonstrates this weakness – while the system is capable of producing huge amounts of music due to the large corpus that it draws from, the system is incapable of producing music that is not at a tempo of 130 beats per minute, or producing music that is in any key other than a strict diatonic a minor.

The other weakness of industrial systems is that the current architecture requires large investments of labour. *Anarchy Online* (9) demonstrates this weakness. In *AO*, each piece of music needs to be annotated with transitions in and out. Additionally, changes to any music cue requires all other cues that transition to the altered cue. This requires far more labour than composed music does, as composed music can be directly assigned to states, assuring that horizontal transitions and vertical layers will smoothly transition to each other. These weaknesses are exacerbated by each other – any attempt to increase the expressive range of an arrangement system will require increased amounts of musical variety, which will increase the amount of metadata required to ensure that the resulting music does not clash with itself. Any attempt to reduce the labour cost of these systems will restrict the composed musical library, which reduces the expressive range of the system.

Academic systems in contrast tend towards addressing the composition task. This removes some of the inherent weaknesses of arrangement systems, but these systems also have shared weaknesses. The biggest weakness of generative composition systems in the current state of the art is that they are limited by their isolation from the larger game industry context. This isolation often results in music composition systems that could theoretically provide music for a game, but do not engage with the interaction that differentiates games from linear media.

We suggest for academic systems to take inspiration from the related field of game-playing AI. Many online cooperative and competitive games utilize an API that allows for third-party developers to view and respond to in-game events. The company *Overwolf* [97] has also created a single, more unified interface that collects game events in real-time from a selection of games. Companies OpenAI and DeepMind use these competitive games to develop and test cutting-edge game-playing AI [98,99], and we believe that many of the same techniques for playing games could be re-purposed to create content during gameplay.

We also believe that increased investment in generative performance and interpretation is needed for generative music to gain widespread adoption. There is a legitimate fear in the industry that lower quality synthesis without interpretation will be poorly received. *DOOM (2016)* (27) presents an example of how synthetic sounds could be used to enhance musical performance, as composer Mick Gordon heavily used synthesis by hand-crafting the synthesizer sounds and parameters. Approaching performance with generative techniques will allow generative systems to match composed music in fidelity.

It is clear that generative, adaptive music systems have the potential to provide not only greater variety of game music, but also more compelling and powerful music. The academic evaluation of adaptive and generative systems demonstrates support for this – adaptive music has a greater effect on a player's subjective experienced emotion [20] than linear music, and the generative systems that have been evaluated also demonstrate that generative systems have a similar effect, and can cause objective affective responses as well [38,39].

There is interest in generative music in the games industry, but it is thus far akin to dipping a single toe into the pool of possibilities. We suggest continued cooperation between academia and the games industry, with the intent of developing systems that can address more generative tasks with more expressive range, and that can allow composers to focus on crafting musical worlds, rather than on the data-entry labour required by many current industrial systems. We believe this cooperation will also lead to these systems having access to more evaluative and design resources to smooth out the rough edges that are common in current academic systems. Ultimately, we believe that future cooperation between academia and industry in the field of generative music for games will lead to better games with better music.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

We would like to thank Osama Alsaman for his assistance in reviewing this paper.

References

[1] Entertainment Software Association, 2019 essential facts about the computer and video game industry, 2019.
[2] J. Huizinga, Homo ludens: a study of the play-element in culture, International Library of Sociology and Social Reconstruction, London, 2003.
[3] K. Salen, E. Zimmerman, Rules of Play: Game Design Fundamentals, The MIT Press, 2004 arXiv:arXiv: 1011.1669v3.
[4] M. Sweet, Writing Interactive Music for Video Games, Addison-Wesley, Upper Saddle River, NJ, 2015.
[5] P. Weir, The sound of 'no man's sky', 2017. https://www.gdcvault.com/play/1024067/The-Sound-of-No-Man.
[6] K. Collins, Playing with Sound: A Theory of Interacting with Sound and Music in Video Games, The MIT Press, 2013.
[7] L. Berio, Two Interviews/Luciano Berio, M. Boyars, New York, 1985.
[8] S. Miyamoto, Nintendo Creative Department, Super mario bros, 1985.
[9] H. Akamatsu, Konami, Castlevania, 1986.
[10] A. Kitamura, Capcom, Mega man, 1987.
[11] M. Toriyama, Square Enix, Final fantasy xiii, 2009.
[12] S. Velasco, Yacht Club Games, Shovel knight, 2014.
[13] A. Rao, G. Kasavin, Supergiant Games, Pyre, July 2017.
[14] T. Antoniades, Ninja Theory, Senua's sacrifice, Hellblade, 2017.
[15] J. Ohlen, BioWare, Black Isle Studios, Baldur's gate, 1998.
[16] S. Hill, Rocksteady Sudios, Batman: Arkham asylum, 2009.
[17] J. Solomon, Firaxis Games, Xcom 2, 2016.
[18] K. Collins, From Pac-Man to Pop Music Interactive Audio in Games and New Media, Ashgate Publishing Ltd, Farnham, Farnham, 2011.

[19] Vlambeer, Luftrausers, 2014.
[20] Music matters: An empirical study on the effects of adaptive music on experienced and perceived player affect, 2019.
[21] P. Pasquier, A. Eigenfeldt, O. Bown, S. Dubnov, An introduction to musical metacreation, Comput. Entertain. 14 (2) (2016) 1–14.
[22] P. Galanter, What is generative art? Complexity theory as a context for art theory, in: GA2003–6th Generative Art Conference.
[23] V. Velardo, Melodrive, 2018. https://melodrive.com/index.php.
[24] K. Collins, An introduction to procedural music in video games, 2009.
[25] G.N. Yannakakis, J. Togelius, Artificial Intelligence and Games, Springer International Publishing, Cham, 2018.
[26] W.A. Mozart, Musikalisches würfelspiel, 1792.
[27] Phosfiend Systems, Fract osc, April 2014.
[28] J. Cullimore, H. Hamilton, D. Gerhard, Directed transitional composition for gaming and adaptive music using q -learning, ICMC (2014) 332–338.
[29] M. Scirea, Affective music generation and its effect on player experience, Ph.D. thesis IT University of Copenhagen, 2017.
[30] J. Sawyer, B. Null, E. Fenstermaker, Pillars of Eternity, 2015.
[31] Canadian League of Composers, Commissioning Rates — Canadian League of Composers, 2015. https://www.composition.org/commissioning-rates/.
[32] I. Stravinsky, Poetics of Music in the Form of Six Lessons, Harvard University Press, 1970.
[33] D. Plans, D. Morelli, Experience-driven procedural music generation for games, IEEE Trans. Comput. Intell. AI Games 4 (3) (2012) 192–198.
[34] C. Crawford, Chris Crawford on Game Design, Peachpit, 2003 arXiv:9780201398298.
[35] T. Challies, no man's sky and 10,000 bowls of plain oatmeal, Oct 2016. https://www.challies.com/articles/no-mans-sky-and-10000-bowls-of-plain-oatmeal/.
[36] P. Hoegi, A Tabular System: Whereby the Art of Composing Minuets is Made So Easy that Any Person, Without the Least Knowledge of Musick, May Compose Ten Thousand, All Different, and in the Most Pleasing and Correct Manner. Invented by Sigr. Piere Hoegi, Printed at Welcjer's musick shop, 1763.
[37] S. Engels, F. Chan, T. Tong, Automatic real-time music generation for games, Eleventh Annual AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, 2015, pp. 220–222.
[38] A. Prechtl, Adaptive music generation for computer games, Ph.D. thesis, The Open University, 2016.
[39] M. Scirea, J. Togelius, P. Eklund, S. Risi, Affective evolutionary music composition with MetaCompose, Genet. Program Evolvable Mach. 18 (4) (2017) 433–465.
[40] S. Ritchie, E. Boon, D. Watson, J. Joos Jr., Black knight 2000, April 1989.
[41] R.v. Tol, S. Huiberts, Ieza: A framework for game audio, Jan 2008. https://www.gamasutra.com/view/feature/3509/ieza_a_framework_for_game_audio.
[42] B. Softworks, Fallout 3, October 2008.
[43] Nintendo, Y. Yamada, E. Aonuma, Y. Koizumi, The legend of zelda: Ocarina of time, November 1998.
[44] Die Gute Fabrik, Sportsfriends, 2014.
[45] P. Magnuson, Basic rules for species counterpoint, 2008. http://academic.udayton.edu/PhillipMagnuson/soundpatterns/speciesctpt/.
[46] S.J. Russell, P. Norvig, Artificial intelligence: a modern approach, third ed., Prentice Hall series in artificial intelligence, 2010.
[47] J.A. Biles, Genjam: A genetic algorithm for generating jazz solos, Proceedings of the International Computer Music Conference, 1994, pp. 131–137.
[48] E.R. Miranda, D. Williams, Artificial intelligence in organised sound, Org. Sound 20 (1) (2015) 76–81.
[49] P. Pasquier, Generative art and computational creativity. https://www.kadenze.com/courses/generative-art-and-computational-creativity-i.
[50] D. Conklin, I.H. Witten, Multiple viewpoint systems for music prediction, J. New Music Res. 24 (1) (1995) 51–73.
[51] J. Robertson, A. De Quincey, T. Stapleford, G. Wiggins, Real-Time Music Generation for a Virtual Environment, ECAI-98 Workshop on AI/ALife and Entertainment.
[52] T. Eerola, J.K. Vuoskoski, A comparison of the discrete and dimensional models of emotion in music, Psychol. Music 39 (1) (2011) 18–49.
[53] W. Wright, Spore, 2008.
[54] K. Tatar, P. Pasquier, Musical agents: a typology and state of the art towards musical metacreation, J. New Music Res. 48 (1) (2019) 56–105.
[55] K. Jolly, A. McLeran, Procedural music in spore, 2008. https://www.gdcvault.com/play/323/Procedural-Music-in.
[56] U. Schimmack, A. Grob, Dimensional model of core affect: a quantitative comparison, Eur. J. Personality 14 (2000) 325–345.
[57] C. Plut, The Audience of the Singular, Master's thesis, Simon Fraser University, Vancouver, BC, 2017.
[58] D. Levine, P. Langston, D. Riordan, G. Hare, Ballblazer, 1984.
[59] P.S. Langston, Six techniques for algorithmic music composition, in: International Computer Music Conference. Computer Music Conference Association, San Francisco, 2005, pp. 164–167.
[60] K. Winbladh, H. Ziv, D.J. Richardson, iMuse, Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering – FSE '10, 2010, p. 383. doi:https://doi.org/10.1145/1882291.1882360.
[61] P. Silk, imuse demonstration 2 – seamless transitions, May 2010. https://bit.ly/1R39FPY.
[62] C. Hubbard, K. Stephens, W. Saulsberry, G. Whitemore, C. Miller, S. Ryan, Monolith Interactive, The Operative: No One Lives Forever, 2000.
[63] Gamelab, N. Fortugno, Diner dash, 2004.
[64] Naughty Dog, B. Straley, H. Amy, Uncharted 2: Among thieves, 2009.
[65] T. Nispen tot Pannerden, S. Huiberts, S. Donders, S. Koch, The NLN Player: A system for nonlinear music in games, in: International Computer Music Conference (August), 2011, pp. 269–321.
[66] M. Stratton, H. Martin, T. Bell, J. O'Connell, B.E. Khan, H. Martin, A. Gascoine, M. Gordon, id Software, DOOM, 2016.
[67] L. Reycevick, The Brilliance of DOOM's Soundtrack, 2016. https://www.youtube.com/watch?v=7X3LbZAxRPE.
[68] Microsoft, Halo Series, 2001–2017.
[69] Airtight Games, J. Perez III, J. Lamparty, Dark Void, 2010.
[70] Archived Documents, Directmusic c/c reference, Apr 2009.
[71] Funcom, Anarchy online, 2001. https://www.anarchy-online.com/.
[72] B.A. Lagim, The music of anarchy online: Creating music for mmogs, Sep 2002. https://www.gamasutra.com/view/feature/131361/the_music_of_anarchy_online_.php.
[73] Ubisoft Shanghai, M. de Plater, Tom clancy's endwar, 2008.
[74] B. Houge, Cell-based music organization in tom clancy's endwar, 2012.
[75] Rockstar Games, Red Dead Redemption, 2010.
[76] Hello Games, No Man's Sky, 2016.
[77] P. Hutchings, J. McCormack, Adaptive music composition for games, IEEE Trans. Games (2019).
[78] I. Toshio, Otocky, 1987.
[79] United Game Artists, J. Kobayashi, T. Mizuguchi, H. Abe, K. Yamada, K. Yokata, Rez, 2001.
[80] Q. Entertainment, T. Mizuguchi, Child of eden, 2011.
[81] Gaijin Games, A. Neuse, Bit.trip runner, 2011.
[82] J. Plachy, A. Design, Chuchel, 2018.
[83] G. Cuzzillo, Ape out, 2019.
[84] J. Hedges, K. Larson, C. Mayer, An Adaptive, Generative Music System for Games, 2010. https://www.gdcvault.com/play/1012710/An-Adaptive-Generative-Music-System.
[85] P. Lopes, A. Liapis, G.N. Yannakakis, Sonancia: Sonification of Procedurally Generated Game Levels, in: Proceedings of the 1st Computational Creativity and Games Workshop, 2015.
[86] U. Schimmack, R. Reisenzein, Experiencing activation: energetic arousal and tense arousal are not mixtures of valence and activation, Emotion 2 (4) (2002) 412–417.
[87] J. Russell, A circumplex model of Affect, J. Pers. Soc. Psychol. 39 (6) (1980) 1161–1178.
[88] FMOD, FMOD Studio, 2016. http://www.fmod.org/products/.
[89] Audiokinetic, Wwise, 2017.
[90] J. Atherton, G. Wang, Chunity: Integrated audiovisual programming in unity, New Interfaces for Musical Expression.
[91] B. Garton, Rtcmix, 2019. http://rtcmix.org/.
[92] Epic Games, T. Sweeney, Unreal engine, 1998. https://www.unrealengine.com/en-US/what-is-unreal-engine-4.
[93] Unity3d, Unity, 2019. https://unity.com/.
[94] Square Enix, H. Tabata, Final fantasy xv, 2016.
[95] A. Preece, The world of audio games: A crash course, Nov 2013. https://www.afb.org/aw/14/11/15738.
[96] G. Kramer, D. Alexander, Why the Music in Dragon Quest XI is so Terrible, 2018. https://youtu.be/xfdFU3O3nf8.
[97] Overwolf, Overwolf: Tech for developers who love gaming. https://www.overwolf.com/.
[98] D. Farhi, J. Pachocki, S. Sidor, G. Brockman, Openai five. https://openai.com/five/.
[99] T. Simonite, Deepmind beats pros at starcraft in another triumph for bots, Jan 2019. https://www.wired.com/story/deepmind-beats-pros-starcraft-another-triumph-bots/.