Composing Video Game Levels with Music Metaphors through Functional Scaffolding

Amy K. Hoover
Institute of Digital Games
University of Malta
Msida, Malta
amy.hoover@gmail.com

Julian Togelius
Dept. Computer Science and Engineering
New York University
New York, NY, USA
julian@togelius.com

Georgios N. Yannakis
Institute of Digital Games
University of Malta
Msida, Malta
georgios.yannakakis@um.edu.mt

Abstract

Artists and other creators naturally draw inspiration for new works based on previous artifacts in their own fields. Some of the most profound examples of creativity, however, also transform the field by redefining and combining rules from other domains. In procedural content generation for games, representations and constraints are typically modeled on the target domain. In contrast, this paper examines representing and generating video game levels through a representation called functional scaffolding for musical composition originally designed to procedurally compose music. Viewing music as a means to re-frame the way we think about, represent and computationally design video game levels, this paper presents a method for deconstructing game levels into multiple "instruments" or "voices," wherein each voice represents a tile type. We then use functional scaffolding to automatically generate "accompaniment" to individual voices. Complete new levels are subsequently synthesized from generated voices. Our proof-of-concept experiments showcase that music is a rich metaphor for representing naturalistic, yet unconventional and playable, levels in the classic platform game *Super Mario Bros*, demonstrating the capacity of our approach for potential applications in computational creativity and game design.

Introduction

In what way is a game level like a musical composition? In several ways, as it turns out. To begin with, let us make an assumption that we are talking about linear levels, where the player characters freely traverse forward or backward, but little variation exists in other dimensions of player movement (path changes are rare or inconsequential). Such levels are common, for example in platform games such as *Super Mario Bros*, racing games such as *Need for Speed*, and fighting games such as *God of War*. Interestingly, while many games at first appear to allow players to choose where to go and feature intricate 3D graphics representing seemingly expansive worlds, often they are in fact linear experiences at their core. This goes particularly for the campaign modes of first-person shooter games such as *Halo* and *Call of Duty*, where the player's three-dimensional movements are carefully funneled into an effectively one-dimensional corridor. While certain types of games feature completely non-linear levels, in particular open world games such as *Grand Theft Auto* and *Skyrim* have fairly linear levels.

Like these game levels, musical pieces are compositions that proceed linearly over time. They have beginnings and ends, and the musical ideas or content in between is experienced in a particular order. Several other concepts in music have analogues in level design. Take for example intensity, where intensity in music (typically expressed by frequency of notes, layering of instruments etc.) can be more or less directly translated to intensity in a (part of a) level (typically expressed by number of opponents, frequency of complicated jumps etc). Themes and other repeating melodic structures, i.e. short segments that can be repeated with variations, are the equivalent of patterns in level design as defined by Björk and Holopainen (2004). Music is almost always represented as a discrete time series of notes; it is rare that music time windows are near-continuous (such as in the case of a glissando) or a note lies outside the standard-defined note pitches (e.g. an indefinite pitch). Likewise most game levels adopt discrete space and time scales and use standardized and pre-defined pieces of content placed appropriately in the level (as standard notes placed within a discrete time scale).

Similarities between linear levels and musical compositions is perhaps most apparent in games where the player is moved through the level structure at a set pace; this includes classic scrolling arcade games such as *R-Type* and newer "infinite runners" such as *Canabalt* or *Flappy Bird*. Here, the forced movement of the screen and player character mirrors the forced progression when listening to music.

Assuming that levels are linear structures, methods for algorithmically analyzing and generating music can also analyze and generate game levels. For example, a common technique for analysis and generation is n-grams. This simple method essentially counts the occurrences of sequences of length n within a sequence, and creates a table of subsequence frequencies that can then be used to statistically generate new sequences that are similar to the original sequences. If an n-gram is trained on one or many melodies, it can then produce any number of new melodies that embody regularities in the melodies on which it has been trained. N-grams and similar sequence mining methods can successfully capture and reproduce local structure in musical composition, but typically face difficulties capturing long-term dependencies and other global structures (Ames (1989)). The very same methods can be used for games. For example, Dahlskog, Togelius, and Nelson (2014) trained n-grams on Super Mario Bros levels, and showed that they could generate levels in the same style as those the n-grams were trained on.

Another feature of most musical compositions is that they feature multiple voices, typically played by different instruments or sung by different humans. Most Beatles songs feature one drum track, one bass track, two guitar tracks, and one voice track. These voices are highly dependent on each other; if they fail to take each other into account, the music will sound disharmonious or out of sync. But when well executed the interplay of different voices adds richness and depth to a composition beyond what a single melody could.

We can use the same analysis model for game levels. A game level could be seen as consisting of several different structures that depend on each other. The interplay of open areas, walls, items, NPCs, start and exit locations and so on is what creates the dynamics of a level together; only walls will not make a level, and the placement of items has no meaning except in the context of the placement of walls. The effectiveness of the placement of one type of in-level content depends crucially on the other content in the level; for example, if the power-up is placed before or after an end boss radically changes its meaning, and some modes of relative placement are clearly superior to others given reasonable assumptions of playing styles. We can therefore talk about a game level as a composition of several voices, where each voice is a sequence of positions for a type of object.

The list of trans-medial analogues could be made much longer but the presented analogies already make the core message of this paper rather clear: music can be used as a metaphor for efficiently representing game levels (and possibly vice versa) and that such a trans-medial way of reframing game level representation might afford new expressive possibilities to computational game level designers.

An Example: Super Mario Bros

Let us develop the idea suggested above in the context of a concrete example. We choose the classic platform game *Super Mario Bros* (SMB), as SMB levels are highly linear and well-studied within procedural content generation. An SMB level could be seen as a matrix with a given length and height, where each cell corresponds to a *tile*, which could be a brick, air, an enemy etc. In the following, we will assume that we traverse the level from left to right in steps of a single tile (Mario himself is one tile wide). The first voice, analogous to a bass or drum track, would be the height of the ground. As remarked on by other authors (e.g. Smith, Whitehead, and Mateas (2011)), the interplay of ground and gaps in the ground (which the player must jump over in order to progress) defines a basic rhythm of the level. When extracting voices from an SMB level, the first voice could therefore be the height of the ground at each x-position in the level, with 0 signifying a gap. (The mapping of a bass tone to the height of the ground has previously been suggested in the context of a side-scrolling shooter by Holtar, Nelson, and Togelius (2013).) A good choice for the second voice would be the height of the highest non-ground platform at each position, with 0 indicating no platform. We could continue this way until we have one voice for each tile type; the particular representation we use is detailed in .

Overview of the Paper

This paper describes a proof-of-concept experiment in which levels for Infinite Mario Bros, a clone of Super Mario Bros, are generated through a process of analyzing levels into 'voices', learning the relationship between voices, generating accompanying voices based on a new voice and synthesizing new levels based on the generated voices. To put this work into context, the next section will describe related work on automatic and computer-assisted composition, functional scaffolding, and procedural content generation in games. The section after that describes our methods: how the Infinite Mario Bros level format was split into voices, and the functional scaffolding method was adapted to learn relations between these. The results section describes several different attempts to generate levels using this method and showcases generated levels. We finally discuss the applicability of this method across game genres and use cases.

Background

Automatic Composition and Functional Scaffolding

Like many approaches for procedurally generating artifacts, methods for representing and generating music are often inspired by domain-specific knowledge.For instance, for a more natural melodic feeling, Holtzman (1981) focuses on the physical limitations of harpists when developing rules for constraining harp melodies. Similarly, Keller and Morrison (2007) restrict potential melodies through grammars designed to replicate the styles of different jazz artists (Keller and Morrison, 2007).Likewise, another approach called functional scaffolding for musical composition (FSMC) generates music by proposing and exploiting two deeper musical principles: 1) music can be represented as a function of time and 2) the musical voices in a given piece can be represented as functions of each other (Hoover and Stanley, 2009; Hoover, Szerlip, and Stanley, 2014).By inputting as little as a single monophonic voice or human *scaffold*, complete polyphonic pieces can be generated through the functional transformation of the scaffold.

Inspired by FSMC, this paper presents a method for composing complete video game levels based on an initial, human composed scaffold or in-game items. Levels are first conceived as a combination of voices, each representing an in-game item. Then, to generate a completely new level several of the voices are selected to scaffold the generation of additional assets (e.g. the tubes, enemies, bricks, etc.).

Procedural Content Generation

Procedural content generation (PCG) in games refers to the algorithmic creation of game content with no or limited human input. Shaker, Togelius, and Nelson (2014) survey the field, discussing both methods for generating content and types of content that can be generated. While game content can refer to a large variety of classes or artifacts, including quests, characters, game rules, items and texture, this paper concentrates on the procedural generation of game levels.

PCG has existed in games since the early eighties, with games such as Rogue pioneering runtime generation of levels. Reasons for generating game levels include reducing designer effort, reducing required storage space, and providing infinite replayability. In recent years, many games have built aesthetics around particular methods of content generation, such as highly acclaimed platformer *Spelunky*, blockbusters *Diablo III* and *Borderlands*, and much anticipated space exploration game *No Man's Sky*.

At the same time, there has been intense research activity in PCG within academia. This has taken the form of exploring new methods for PCG, such as the search-based paradigm where evolutionary algorithms are used (Togelius et al., 2011), the solver-based paradigm where PCG problems are formulated as constraint satisfaction problems (Smith and Mateas, 2011) and the experience-driven paradigm where content is generated to optimize models of player experience (Yannakakis and Togelius, 2011). Further, the role of PCG within the game development process or as a part of the game has been explored. In particular, mixed-initiative methods have been proposed where PCG methods augment human editing of game levels. One example of this is the *Tanagra* platform game level editor (Smith, Whitehead, and Mateas, 2011), which uses PCG to accompany human editing.

Relatedly, the challenge of generating multiple types of game content in a unified manner using multiple disparate generative methods has been posed as a grand challenge in computational creativity (Liapis, Yannakakis, and Togelius, 2014a). While in the current study we are focusing only on levels for a platform game, we are exploring something analogous to multifaceted content generation in that we are breaking down the challenge of generating a particular artifact into the challenges of generating a set of constituent content types, namely, the individual "voices".

Methods and Experiments

This section describes the approach for representing game levels as FSMC voices and the artificial neural network (ANN) training method for predicting appropriate voices for a given human input, or *scaffold*.

Level Representation and Voice Extraction

We used 8 of 23 available levels from the original Super Mario Bros game, which had been encoded into the Infinite Mario Bros level format as training data (Horn et al., 2014). The encoding was done by Dahlskog and Togelius (2014).

Infinite Mario Bros levels are represented as matrices of height 14 and lengths of a few hundred (most common level length is 200). Each cell in the matrix represents a single block/tile, such as a goomba, brick block, or pipe. Almost everything in the game has an extension of a single tile, except Mario in large mode who is two tiles tall. In this representation, each cell contains a single character, which is then converted by the IMB game engine to the corresponding tile when a level is loaded. Thus, space means an empty tile, "g" means a ground tile, "p" a platform, "k" a red koopa etc.

To convert a level into voices the following procedure was

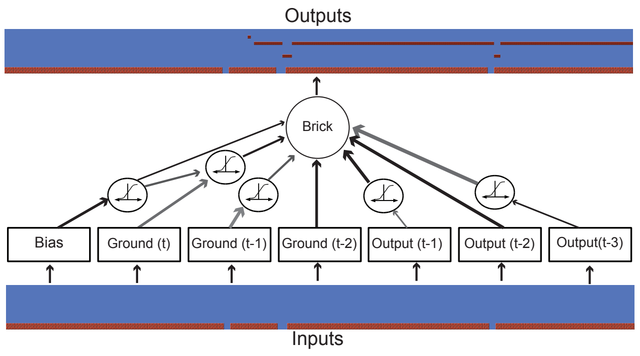


Figure 1: **ANN Representation**. Each ANN takes as input voices from one or several levels Super Mario Bros. Through NEAT, these ANNs are evolved to predict actual in-game tiles (in this example, the brick placements in a level). To best capture the regularities in a level, the inputs and outputs are fed back into the ANN at each subsequent tick. Once trained, ANNs can potentially suggest reasonable placements for new human composed in-game tiles.

followed. One voice was created for each of the following eight tile types: ground, platform, question mark block, brick block, stone, goomba, pipe, and koopa. A voice is a one-dimensional array with the same length as the level's width. For each voice, each column of the level matrix is inspected to see whether there is a tile of the specified type. If the column contains a tile of right type, then the value of the voice at that position will be the height (y-position) of the tile; otherwise it will be zero.

The set of voices extracted from each level faithfully and completely represents the level, provided that there is maximally one tile of each type in each column. If there are several tiles of the same types at different heights in a column, the lower tile(s) will be disregarded. In the original Super Mario Bros levels used for training data in this paper such situations are relatively rare, and mostly concern situations when multiple platforms overlap at different levels. These situations could in the future be addressed by adding extra voices or something akin to overtones, but for the current paper we simply disregard them.

Translating from a set of voices back to a level is straightforward. Starting from an empty level, voices are added one at a time. A voice is added through simply drawing tiles at the given height at each position (or not drawing a tile if the value of the voice is zero). For tubes and stones, positions below the given position are filled in as well.

Training Artificial Neural Networks

Once a set of levels has been analyzed into voices, neural networks were trained to predict a "target" voice based on the value of one or several "given voices" (as shown in figure 1). In each case, what was predicted was the value of one voice at position ("time", following the music metaphor) t based on the value of the given voice at time t as well as earlier times, and optionally the output of the predictor itself at time $t - 1$ or earlier.

For prediction we use neuroevolution, i.e. neural networks trained through evolutionary algorithms. In particular, we use NeuroEvolution of Augmenting Topologies (*NEAT*), a state-of-the-art neuroevolution algorithm capable of evolving both the structure and connection weights of neural networks (Stanley and Miikkulainen, 2002). Experiments in this paper are implemented in Colin Green's NEAT framework, *SharpNEAT* version 2.2 [1].

Several different sets of inputs to the networks were investigated. In the most common combination, the value of the input voice at time t is used, as well as the value of the same voice at $t - 1$ and $t - 2$, the output of the network at time $t - 1$ and a bias input with the constant value 1. This gives a total of 5 inputs for a single voice. When several voices are used, commensurately more inputs are used. It should be pointed out that the NEAT algorithm is capable of selecting which inputs to use for training, and so adding inputs should not have adverse effects on learning capability (within reason).

In the network configuration, each network predicts a single voice. The output o, which is constrained to the range $0..1$ is interpreted as the predicted value of the target voice at time t thus: if $o < 0.5$ then the voice value is 0, otherwise it is $(o - 0.5) * 28$ so that all 14 possible height values can be represented by the single network output.

The fitness function was designed to reward correct reproduction of the target voice(s). It steps through the given voices from $t = 0$ until the end of the voices ($t = 199$ for many of the example levels). At each step, it compares the predicted voice value (v_p) with the actual value of the target voice (v_t). If p_v is zero and p_t non-zero, or vice versa, the reward at this point is zero; if both p_v and p_t values are zero the reward is one. If both the predicted and the target value function is non-zero, the reward is $1 - abs(v_p - v_t)/14$. This scheme rewards networks in proportion to how close the predicted height of a voice is to the actual voice. To obtain the fitness of the network, all rewards are summed and normalized by the length of the voice, so that perfect reproduction yields a fitness of 1.

Five different networks were trained on the first level of each of the eight worlds of Super Mario Bros, i.e. levels 1-1, 2-1, 3-1, 4-1, 5-1, 6-1, 7-1, and 8-1. Each network is trained on all eight levels, but only predicts one voice. The first network predicts *goombas* based on the voices *ground*, *bricks* and *question marks*. The second network predicts *powerups* based on the four previous voices. The three subsequent networks predict *platforms*, *tubes* and *stones* in that order. Each of the five networks considered inputs all the previous voices including those voices generated by the previous networks. Figure 2 shows the fitness growth in an example training session.

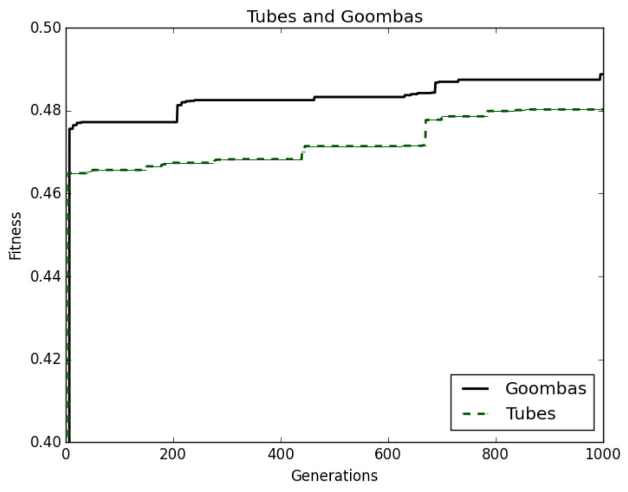


Figure 2: **Example Training Session for Goombas and Tubes Showing a Fitness Increase**.

Results

This section describes a set of example results from training networks to reproduce missing voices in existing Super Mario Bros levels, and then using these trained networks both to reproduce the missing voices and then to create new voices for partial levels we created ourselves specifically for this project. We find that while our networks will not reproduce existing levels precisely — which is not the aim of the paper as that would result to unnecessary overfitting limitations — they will, however, produce reasonable accompaniment to given voices, regardless of whether these voices are part of levels they have been trained on or freshly created.

First of all, Figure 3a shows level 1-1 from Super Mario Bros. Figure 3b shows the three voices *ground*, *bricks* and *question marks* from level 1-1 merged into one level. This level was produced through analyzing the initial level into all its component voices, and then reassembling a new level based on three of these voices.

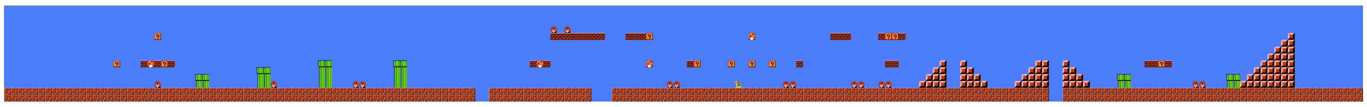
Figure 3c shows level 1-1 "reassembled" through using trained networks to reproduce the voices that were removed, thus completing the level. As can be seen, it is not identical to the original level — few of the reproduced tiles are in the right place. In particular, the stair-like stone formations are missing entirely, instead the stones are placed in columns in unexpected places. However, the reproduced voices represent interesting accompaniment to the existing voices and together they constitute a playable level.

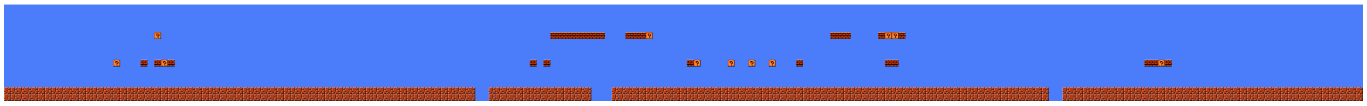
Next, Figure 3d shows a three-voice partial level created by the authors, designed to make ample use of bricks and question marks. Figure 3e shows the same level with the missing voices produced by neural networks. The generated voices produce a relatively sparse but appropriate accompaniment, and a playable level.

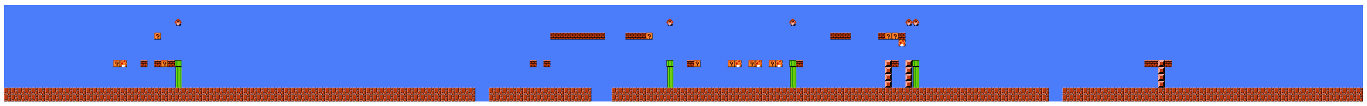
Discussion

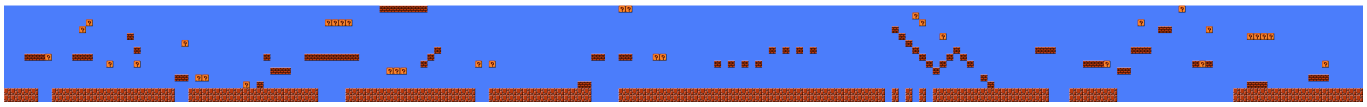
While there may be several reasons that the ANNs have difficulty precisely reproducing voices on which they have been trained including potential limitations of the neural network architecture or training regime, the most likely reason is that some of the regularities are difficult to discover. However, because fitness increases over generations on the trained networks, at least some of these regularities can be represented

[1]http://sharpneat.sourceforge.net/


(a) Original World One Level One in Super Mario Bros.


(b) The Ground, Brick, and Question Mark Voices in World One Level One


(c) The remaining voices regenerated from several trained networks, using the three-voice level in (b) as input


(d) Author-created Level with Ground, Brick, and Question Mark Voices

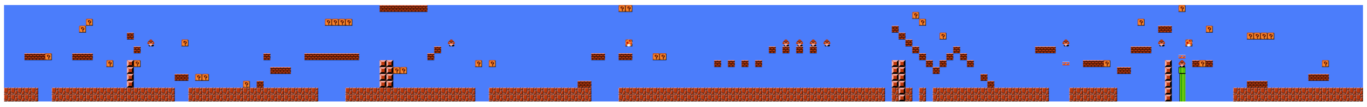

(e) Additional voices generated by trained networks for the author-created level.

Figure 3: **Visual overview of example results.**

through the training method and representation. Although perfectly trained ANNs could potentially create more plausible generated voices, the results indicate that enough is learned to enable complete level construction.

The underlying idea of this paper can be said to be analyzing a linear artifact into various linear bands or channels, and then synthesizing the artifact from such bands or channels again after some manipulation. It is interesting to note various analogies with methods from other engineering fields. One analogy is to frequency domain analysis, such as using Fourier transform to analyze a complex sound into component tones of different frequencies. (A similar idea is implemented in the Jpeg algorithm that analyzes images into channels.) Such transformations can be done in order to apply particular effects to the source content, such as removing, changing or adding frequencies; it could also be done in order to compress the source content with or without loss of information. Perhaps these ideas could carry over to the analysis we perform here. Another intriguing analogy is to multiobjectivization, the practice in optimization to transform a single-objective problem into a multi-objective one. In combination with multiobjective evolutionary algorithms, this has been shown to increase optimization performance by avoiding local optima (Knowles, Watson, and Corne, 2001). It is unclear whether a similar effect could be observed and utilized in the current domain.

A natural extension of the method and experiments described in this paper is to use this method as part of an AI-assisted authoring tool for game levels. It would allow the player to draw one or several voices (e.g. ground and enemies), and have the other voices automatically generated to accompany the drawn voices. This tool could allow the designer to "lock" certain voices in certain parts of the level, so that they become given voices and the basis for generation of other voices. It could incorporate user feedback and retrain its networks through interactive evolution to learn the user's designer preferences, similar to what is suggested in (Liapis, Yannakakis, and Togelius, 2014b). With an addition such as of our system in mixed-initiative level design we envision the generation of naturalistic, nevertheless, unconventional levels for designers to consider in the hope that their creativity is fostered.

As discussed in the introduction, many games have mostly linear levels, suggesting that the method presented in this paper could potentially generate assets for many other games. It is interesting to consider what the other voices would mean in such cases. For a side-scrolling space shooter, a mapping between musical voices and different types of level content can be found in *Audioverdrive* by Holtar, Nelson, and Togelius (2013). For a first-person shooter game, one could imagine that different voices were associated with the width of the path taken, the existence and position of cover points, power-ups, enemies, doors and other checkpoints and so on. It is highly likely that at least some "musical" structures would exist here, with cover points alternating between left and right of the level path, enemies congregating in "waves" and power-ups being placed before such waves.

Ultimately, we wonder how far we can take the central metaphor of game levels as music which underlies the work described in this paper. Would it be possible to find a good game level interpretation of concepts such as tempo, harmony and dissonance? For instance, how closely associated are the principles of tempo and rhythm-based level generation (Smith, Whitehead, and Mateas, 2011)? Perhaps certain types of content coexist harmoniously in some games but other combinations are dissonant. For example in *Halo* several Grunts frequently occur together with Elites, whereas different types of Elites rarely occur together, and enemies from different factions are very rarely encountered in the same level segment. Thus, there appears to be an analogy between frequent subsequences of events within a level and a musical chord. Game tension which is a carefully designed experience element rather common in FPS games such as *Left for Dead* can be seen as analogous to both rhythm and the musical scale. There are many other concepts in sound and music, beyond the above, that could be investigated, such as timbre and loudness.

Conclusions

This paper has presented a method for modeling and generating linear video game levels based on a musical metaphor. Levels are analyzed into separate voices, where each voice represents the existence and height of a different tile type. Conversely, levels can be synthesized by "drawing" voices onto a level canvas. Core to the method is that we see the various voices as accompanying each other, or perhaps some voices as accompanying and others leading. We used an existing neuroevolution-based method for learning and generating musical accompaniment and a corpus of Super Mario Bros levels to learn a model of accompaniment. This model could then generate complete levels based on freeform drawing of one or two voices. While results show that the trained models can generate playable levels with some interesting features based on a few given voices, capturing even more regularities is a future goal. Future work also includes building an AI-assisted level design tool based on the method demonstrated here, and extending the idea to other types of games.

Acknowledgments

This research is supported, in part, by the FP7 ICT project C2Learn (project no: 318480) and by the FP7 Marie Curie CIG project AutoGameDesign (project no: 630665).

References

Ames, C. 1989. The markov process as a compositional model: a survey and tutorial. *Leonardo* 175–187.

Björk, S., and Holopainen, J. 2004. Patterns in game design (game development series).

Dahlskog, S., and Togelius, J. 2014. Procedural content generation using patterns as objectives. In *Applications of Evolutionary Computation*. Springer. 325–336.

Dahlskog, S.; Togelius, J.; and Nelson, M. J. 2014. Linear levels through n-grams. *Proceedings of the 18th International Academic MindTrek.*

Holtar, N. I.; Nelson, M. J.; and Togelius, J. 2013. Audioverdrive: Exploring bidirectional communication between music and gameplay. In *Proceedings of the 2013 International Computer Music Conference*.

Holtzman, S. 1981. Using generative grammars for music composition. *Computer Music Journal* 51–64.

Hoover, A. K., and Stanley, K. O. 2009. Exploiting functional relationships in musical composition. *Connection Science Special Issue on Music, Brain, & Cognition* 21(2):227–251. This paper is accompanied with a set of musical samples at http://eplex.cs.ucf.edu/neatmusic.

Hoover, A. K.; Szerlip, P. A.; and Stanley, K. O. 2014. Functional scaffolding for composing additional musical voices. *Computer Music Journal* 38(4):80–99.

Horn, B.; Dahlskog, S.; Shaker, N.; Smith, G.; and Togelius, J. 2014. A comparative evaluation of procedural level generators in the mario ai framework. In *Proceedings of Foundations of Digital Games (FDG)*.

Keller, R. M., and Morrison, D. R. 2007. A grammatical approach to automatic improvisation. In *Proceedings, Fourth Sound and Music Conference, Lefkada, Greece, July.Most of the soloists at Birdland had to wait for Parkers next record in order to find out what to play next. What will they do now*.

Knowles, J. D.; Watson, R. A.; and Corne, D. W. 2001. Reducing local optima in single-objective problems by multi-objectivization. In *Evolutionary multi-criterion optimization*, 269–283. Springer.

Liapis, A.; Yannakakis, G. N.; and Togelius, J. 2014a. Computational game creativity. In *Proceedings of the Fifth International Conference on Computational Creativity*, 285–292.

Liapis, A.; Yannakakis, G. N.; and Togelius, J. 2014b. Designer modeling for sentient sketchbook. In *Computational Intelligence and Games (CIG), 2014 IEEE Conference on*, 1–8. IEEE.

Shaker, N.; Togelius, J.; and Nelson, M. J. 2014. *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer.

Smith, A. M., and Mateas, M. 2011. Answer set programming for procedural content generation: A design space approach. *IEEE Transactions on Computational Intelligence and AI in Games* 3(3):187–200.

Smith, G.; Whitehead, J.; and Mateas, M. 2011. Tanagra: Reactive planning and constraint solving for mixed-initiative level design. *IEEE Transactions on Computational Intelligence and AI in Games* (99).

Stanley, K. O., and Miikkulainen, R. 2002. Evolving neural networks through augmenting topologies. *Evolutionary computation* 10(2):99–127.

Togelius, J.; Yannakakis, G.; Stanley, K.; and Browne, C. 2011. Search-based procedural content generation: A taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games* (99).

Yannakakis, G. N., and Togelius, J. 2011. Experience-driven procedural content generation. *IEEE Transactions on Affective Computing* 2(3):147–161.