An adaptive music generation architecture for games based on the deep learning Transformer model

Gustavo Amaral Costa, Augusto Baffa, Jean-Pierre Briot, Bruno Feijo, Antonio Luz Furtado

An adaptive music generation architecture for games based on the deep learning Transformer model

Gustavo Amaral Costa dos Santos[1]   Augusto Baffa[1]   Jean-Pierre Briot[2,1]

Bruno Feijo[1]   Antonio Luz Furtado[1]

[1] Dept. of Informatics, PUC-Rio, Rio de Janeiro, Brazil
[2] Sorbonne Université, CNRS, LIP6, F-75005 Paris, France

gustavoacs99@gmail.com, abaffa@inf.puc-rio.br, Jean-Pierre.Briot@lip6.fr,
bfeijo@inf.puc-rio.br, furtado@inf.puc-rio.br

Abstract: This paper presents an architecture for generating music for video games based on the Transformer deep learning model. The system generates music in various layers, following the standard layering strategy currently used by composers designing video game music. The music is adaptive to the psychological context of the player, according to the arousal-valence model. Our motivation is to customize music according to the player's tastes, who can select his preferred style of music through a set of training examples of music. We discuss current limitations and prospects for the future, such as collaborative and interactive control of the musical components.

Keywords: video game music, adaptive music generation, deep learning, Transformer, layering music.

1 Introduction

Music is essential in video games. It provides an embedding context for the players and complements the scenario [33]. Music can also offer some ways of controlling the player [13]. Meanwhile, a recent observation is that many players replace a game's music by listening to some musical piece of their choice. We postulate that this is because of the absence of enough personalization of the music the game offers. Therefore, we started investigating the possibility of generating personalized music based on player preferences (music style, as defined by a corpus of music samples). Deep learning techniques are effective in learning a music style from a corpus. Remain issues on how to customize, control and orchestrate the generation of music in the function of the situation (game and player). In the following, we will introduce the design, implementation and preliminary experiments with a prototype architecture for generating personalized and adaptive music for games.

However, at a time of so few works in the game music generation and so many open questions, we believe in the importance of searching for simple models that explore more fundamental aspects of adaptive music generation. More specifically, it becomes essential to look for models that, more straightforwardly, represent the modes of emotion and levels of emotional intensity involved in video game music generation. Also, we must look for deep learning techniques that are especially adequate to the musical narrative. And, perhaps even more importantly, we should seek a model to support instrumental layers used in video game composition (such as the practice of "striping", which is to record orchestral sections separately for future mixing according to the whims of the composer)[1]. In this context, instead of looking for alternatives or improvements in the existing complete models for adaptive music composition (such as the excellent work by Hutchings and McCormack [15], to be analyzed in Section 2.2), we decided to explore more straightforward and flexible models to support layering music. Also, we believe that our model can facilitate the control and orchestration of music for video games in a collaborative environment.

With the above mentioned principles in mind, after several experiments, we opted for the Transformer architecture [38], because it better captures the long-term structure of music [12]. Our following insight proposed a two-dimensional space of emotions to simplify the metric space and help the practice of layering music. For that, we adopt the arousal/valence model [32]. In this model, we associate arousal (i.e., intensity) with the

---

[1] As film and game music composer Eímear Noone explains: "we might record the strings separately, for example, but we'll compose in a way that the strings on their own provide a functioning piece of music. Then, if our character triggers something in the world, perhaps a battle, we can land the wood winds or brass on top of that to increase the intensity. Each part must be self-contained yet work with others – you need to be able to kick in the brass, kick in the percussion, whenever it's triggered by gameplay." [36]. See also, e.g., [16] for a general introduction to layering.

number of active layers (e.g., the system can add a layer with woodwinds or brass to increase the intensity level if a battle starts in the game's world). And the valence corresponds to the emotional modes of the generated music. We also discuss, in this article, future extensions that this simplified approach makes easier to implement. In particular, we want to move towards collaborative and interactive control of the music components generated by the Transformer-based architecture.

The following sections introduce the design, implementation and preliminary experiments with a prototype architecture for generating personalized and adaptive music for games aligned with the above mentioned principles.

2 Background and Related Work

2.1 Adaptive versus Generative

In [31], Plut and Pasquier present a survey about various approaches and challenges for the generation of music for video games. They consider two primary techniques:

- *adaptive music* (also named *interactive music*), where music is organized in order to be able to react to a game's state [7]. Some musical features (e.g., adding or removing instrumental layers (such as for striping), changing the tempo, adding or removing processing, changing the pitch content...) are linked to game play variables.

  An example of adaptive music is the "Luftrausers" game [39], where the composed music has been split into 3 groupings of instruments, each of which has 5 different arrangements, which a player may select for his avatar (see more details in [31, Section 1.2]).

- *generative music*, where music is not preexisting and dynamically adapted, as for adaptive music, but is generated on the fly. It is created in some systemic way by the computer and is sometimes called procedural music or algorithmic music [21]. The musical content is generated from some model.

  The model can be specified by hand[2]. The limits are that specifying the model is difficult and error prone. The progress of machine learning techniques made it possible to learn models from examples (in other words, specify a model by extension rather than by intention). All but one of the generative music systems surveyed in [31] are using Markov chains models. One example is presented in [8]. Markov models are indeed simpler than deep learning/neural networks models, but they face the risk of plagiarism[3] [5, Section 1.2.3]. The only surveyed generic music system based on artificial neural networks is Adaptive Music System (AMS) by Hutchings and McCormack [15] and it will be summarized in next section (Section 2.2).

Generative music is more general and adaptive than pre-composed composed adaptive music, but is also more difficult to control and more computing demanding. As, noted by [31]: "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" game, Paul Weir, notes that generative music was used in the game with an acknowledgment that it could produce "worse" music than composed music." [40]. Actually, that distinction between adaptive and generative music is not that clear, as often systems classified as generative are not completely generative and include adaptation components. This is for instance the case of the Adaptive Music System[4]. We will now summarize it in next section (Section 2.2) in order to illustrate some issues and also for its own merits.

2.2 Architecture of the Adaptive Music System

The architecture of (AMS) [15] is multi-agent and multi-technique:

- the *harmony role agent*, which generates a chord progression, using an RNN (trained on a corpus of symbolic chord sequences, actually an extension of the harmony system from the same authors [14]);

---

[2]This was for instance the case for the first piece of music composed in 1957 by a computer (the "ILLIAC I" computer at the University of Illinois at Urbana-Champaign (UIUC) in the United States), and therefore named "the Illiac Suite" [11]. The human "meta-composers" were Hiller and Isaacson, both musicians and scientists. It was an early example of algorithmic composition, making use of stochastic models (Markov chains) for generation as well as rules to filter generated material according to desired properties.

[3]By recopying too long sequences from the corpus. Some interesting solution is to consider a variable order Markov model and to constrain the generation (through min order and max order constraints) [25].

[4]Note that it is classified as generative in [31], although its very name claims it as adaptive!

- the *melody role agents* (one for each instrument), which instantiate characteristics (length, pitch, proportion of diatonic notes...) of pre-existing melodies, using an evolutionary rule system (XCS, for eXtended learning Classifier System [41]) and adapting them to the harmony;

- the *rhythm role agent*, which uses another RNN model.

AMS considers a model of 6 emotions: sadness, happiness, threat, anger, tenderness and excitement[5], whose selection is triggered by current game state (every 30ms, the list of messages received by the Open Sound Control (OSC) is used to update the activation values). Emotions in turn will modulate the selection among pre-melodies (choosing the melodic theme assigned to currently highest activated concept or affect), with the instantiation of their characteristics being managed by a spreading activation model[6].

As we can see, AMS actually proposes a sophisticated (and clever) generation model which includes both adaptive and generative aspects (for instance, harmony is generated and melodies are adapted). AMS has been tested in two games: Zelda Mystery of Solarus (MoS) (actually an open-source clone version) [35] and StarCraft II [3].

The comparison of our model with the much more complete and robust AMS architecture is twofold. Firstly, we more straightforwardly represent the emotional intensity in music generation, and secondly, we better support layering music. The simplicity of our approach aims to facilitate future prospecting in collaborative and interactive environments. Furthermore, we use a deep learning architecture (Transformer) better suited to capture long-term coherence in music.

3 Adaptability versus Continuity and other Design Issues

A pure generative approach is some kind of ideal, as it could in principle combine personalization (learnt styles) with real-time adaptation (to the game and players situation)[7]. Using symbolic-level music models as opposed to signal-level music models brings the advantages of higher level manipulation (at the composition level) and less computer resources[8]. An important and actually difficult issue remains the capacity to generate on the fly music content[9] and be able to adapt it, while maintaining some continuity (as for a musician improvising in some jazz context, balancing between constructing and following some musical discourse and fitting into the dynamic context, in the first place, harmonic modulations). Recent progress for control strategies for Markov chains and as well for deep learning show promising results. Markov constraints have been proposed as an unifying framework for Markov-based generation while satisfying constraints [23], and has been applied to real-time improvisation [22] and to interactive composition [26]. Challenges for introducing control are somehow harder for deep learning (as explained in [4, Section 10]), but progresses are made, using control strategies such as conditioning (adding some additional input to the neural network in order to parameterize training and generation), e.g., as in [9].

4 Current Proposal

Although simpler, our current prototype shares some similarity with AMS (see Section 2.2 and [15]), in that it uses both neural network-based generation and an emotion reference model.

4.1 Design Principles

After having at first experimented with a recurrent neural network architecture of type LSTM (part of Google's Magenta project library) [20], we selected the Transformer architecture[10] [38], for its ability to enforce consis-

---

[5]The 5 first ones are the most consistently used labels in describing music across multiple music listener studies [19]. Excitement has been added as an important aspect of emotion for scoring video games.

[6]It is a graph of concept nodes, connected by weighted edges representing the strength of the association between the concepts (and is inspired from a semantic content organisation in cognitive science [6]). Activation spreads as a function of the number of mediating edges and their weights. As explained in [15]: "Spreading activation models don't require logical structuring of concepts into classes or defining features, making it possible to add content based on context rather than structure. For example, if a player builds a house out of blocks in Minecraft, it does not need to be identified as a house. Instead, its position in the graph could be inferred by time characters spend near it, activities carried out around it, or known objects stored inside it."

[7]The issue of how to combine various context information, plot, evolution, player(s) situation, etc., including statistics, e.g., average reactivity of a player, into some decision about what is the objective (adapt to current game context, or the opposite, trigger a player to engage more) and how to accordingly adapt the music is still an open issue. It is likely that it should use some aggregation/interpretation rules, as well as multi-criteria decision strategy, within some front end module in charge of mapping events and models from the game up to the control parameters for music generation or/and adaptation.

[8]Although, recent waveform-level models such as WaveNet [37] demonstrated the feasibility of real-time conditioned generation, used for instance for intelligent assistants such as Google Echo or Amazon Alexa.

[9]Hard real-time may be unnecessary, as the music does not have to adapt immediately to events, as opposed to sound effects.

[10]Transformer is an important evolution of a Sequence-to-Sequence architecture (based on RNN Encoder-Decoder), where a variable length sequence is encoded into a fixed-length vector representation which serves as a pivot representation to be iteratively

tency and structure, by better handling long-term correlations. It recently became popular for such applications as: translation, text generation (e.g., the Generative Pre-trained Transformer 3 aka GPT-3 model), biological sequence analysis and music generation [12].
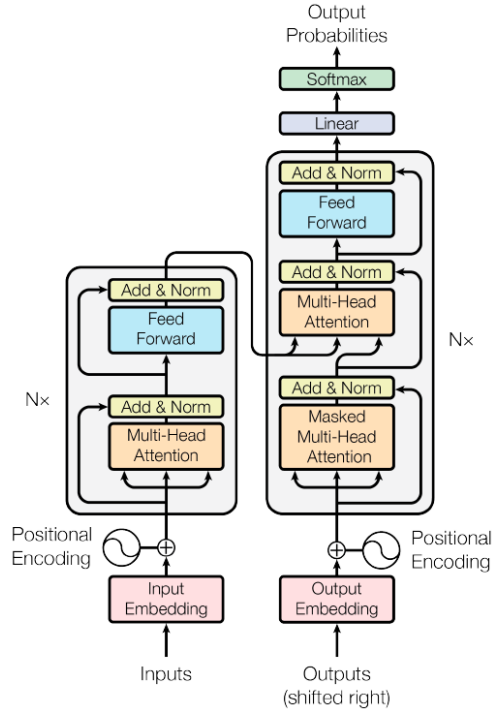

Figure 1: Transformer architecture, reproduced from [38]

The proposal by Jeffries for ambient music generation based on the Transformer [17] has also been a source of inspiration.

4.2 Training Examples

We selected ambient music as the musical style for our first experiments, bearing in mind the generality of the model, where the user could arbitrarily select a corpus of music of its preference (e.g., classical, jazz, techno...). The corpus of music for training musical examples is a Spotify playlist named "Ambient songs for creativity and calm", curated by Jeffries, and containing approximately 20 hours and 165 titles [18].

The compressed audio files (mp3) corresponding to the musical examples were uncompressed into waveform (wav) files and then, by using a pitch detector, to symbolic (midi) files. For the polyphonic transcription to midi files, we used the Onsets and Frames transcription system[11] [10], developed by the Magenta project.

4.3 Layering

We consider layers of music, analogous to the production of orchestral music for games [36], with currently 3 layers:

- 1st layer, the most conservative and neutral;

- 2nd layer, to add more excitement, e.g., though some additional instrument;

- 3rd layer, to intensify the immersion and the tension.

These layers are generated from the same learning corpus, but from different seeds (starting sequences) and with different generation parameters[12], depending on the controlling model (as will be presented in Section 4.5).

---

decoded to generate a corresponding sequence (see more details, e.g., in [5, Section 5.13.3]). Its main novelty is a self-attention mechanism (as a full alternative to more classical mechanisms such as recurrence or convolution), to focus on contributing elements of an input sequence. For more details on the architecture, illustrated in Figure 1, please see the original article [38] or some pedagogical introduction [29].

[11]It uses both convolutional and recurrent (LSTM) neural networks in order to: 1) predict pitch onset events; and 2) to use predictions to condition framewise pitch predictions.

[12]Currently, we vary a temperature parameter that controls the determinism of the generation, for some more likely or more unpredictable result.

In addition to this static parameterization of their generation according to the controlling model, each musical layer will be dynamically activated and played (or not)[13], depending on the strategies of the controlling model.

4.4   Mapping Emotions

In order to have some high-level and human understandable control of the generation by the game play context (game and player(s)), we chose an emotion model, more precisely the arousal/valence model [32], in which an emotion can be mapped using two parameters:

- the *arousal*, which represents the intensity of the emotion;

- and the *valence*, which represents its quality (e.g., if it is positive, negative, neutral. . .).

In order to simplify our current prototype, we now consider only 9 (discrete) emotions, as illustrated in Figure 2.
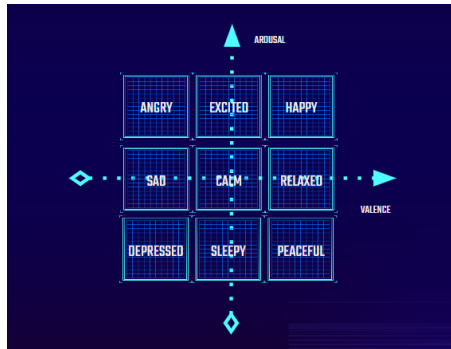

Figure 2: Arousal/valence emotion model and the 9 pre-defined emotions

The emotion model is designed as a server receiving control information from the game, in order to be able to work with various games and game values models. The game play information (events) emitted by the game may be about the game situation, player(s) situation, but also from various other sources such as quests, terrains, etc. How to aggregate these various informations is still an open issue for future work (see Section 5.1).

4.5   Strategy and Control Model

While planning for the future some more advanced state machine for mapping the emotions into generation control strategies (as will be detailed in Section 5.1), in current prototype we have implemented 9 pre-defined strategies (corresponding to the 9 emotions shown in Figure 2), with for each one different values corresponding to the parameters for the generation: which layers are activated, which instruments (sampled or synthetic sounds[14]) are used and which effects are used. More strategies/types may be added by adding strategy classes to the implementation (as shown in Figure 3).

The complete model (Strategy/Layer/Emotion) for controlling music generation is shown in Figure 4. Current mapping is as follows: the strength (arousal) corresponds to the number of active layers, while the quality (valence) corresponds to the choice of emotional modes of the generated musical components.

4.6   Architecture

The architecture of current system is illustrated in Figure 5, and the flow logic is as follows:

1. User's client requests a music;

2. The server maps the user feeling through the arousal valence parameters;

3. It fetches, from memory, a song correspondent to the mapped emotion[15];

4. If no associated music has already being generated, it starts the generation;

5. After the music is fetched, it attaches metadata such as instruments;

---

[13]Currently within the Ableton Live platform, a real-time sequencer for live music creation and production [1].

[14]Currently selected from some instruments library for ambient music within Ableton Live.

[15]This optimization will be detailed in Section 4.7.

```
src > music-gen > music-generator > strategies > TS angry.strategy.ts > ...
  1    import { INSTRUMENTS, MODELS } from '../../../common/constants';
  2    import { IFeelingMap } from '..';
  3    import {
  4      ActiveLayer,
  5      InactiveLayer,
  6      ILayer,
  7      IStrategy,
  8      BaseStrategy,
  9    } from './strategy.interface';
 10
 11    /**
 12     * Concrete Strategies implement the algorithm while following the base Strategy
 13     * interface. The interface makes them interchangeable in the Context.
 14     */
 15    export class AngryStrategy extends BaseStrategy implements IStrategy {
 16      readonly model = MODELS.SAD;
 17
 18      get layers(): ILayer[] {
 19        return [
 20          new ActiveLayer(INSTRUMENTS.SUNRISE_WAVES), // break line
 21          new ActiveLayer(INSTRUMENTS.DARK_ONE),
 22          new ActiveLayer(INSTRUMENTS.AIR_OF_DREAD),
 23        ];
 24      }
 25
 26      isInRange({ arousal, valence }: IFeelingMap) {
 27        return (
 28          arousal >= 75 && // break line
 29          arousal <= 100 &&
 30          valence >= 0 &&
 31          valence <= 50
 32        );
 33      }
 34    }
```

Figure 3: Adding a new strategy named angry. It specifies: the instruments used for each layer (method `get`); and the range of arousal/valence values for triggering the strategy (method `isInRange`)

Figure 4: Complete Strategy/Layer/Emotion model

6

6. It delivers the request response with the music to the final user;
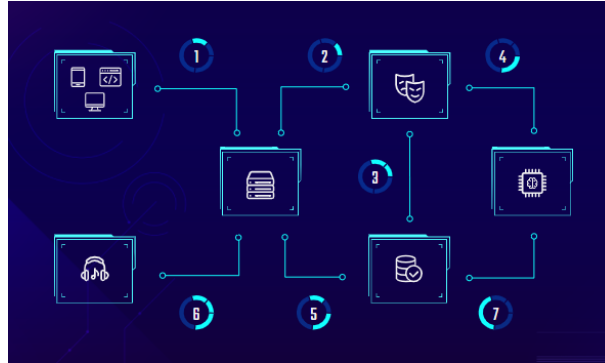
7. The memory is refreshed.

Figure 5: Final architecture flow

4.7 Implementation

To optimize the music generation process, at least one music corresponding to each strategy is saved in memory. The architecture is designed as a server responsible for music generation, for various possible game clients, based on game engines like Unity or Unreal, or specific ones. In order to automate and scale-up the machine learning life cycle, we have used the Pachyderm platform (pipeline) [24]. For the implementation, we have used Nvidia CUDA development environment for high performance GPU-accelerated applications.

4.8 Evaluation

Current architecture has been tested with an emulated game model and with music generated from a corpus of ambient music. Arousal valence values have been estimated according to possible moments of the hero's journey and the behavior of the system. We are planning the integration with a real game using Unity.

5 Prospects

5.1 Game/Music Mapping Model

At present time, input from the game play is limited, but it could benefit from many more parameters and events (e.g., plot situation, player situation, including statistics, e.g., average player reactivity...) and how to aggregate them. And, as opposed to mapping music to player state, we may want to oppose it instead, e.g., if the player is perceived to be showing to signs of abandon, you may want him to try to boost him with some positive music. Last, note that [30] proposes an additional dimension: tension, that you could compute and use to improve the system's emotion mapping.

In addition, as mentioned in Section 4.5, we are planning to substitute current strategy scheme with some more abstract and general state machine model, analog to the AMS spreading activation model (see Section 2.2 and [15]), in order to track the transitions of the player's emotions. Better transitions between music could also be planned ahead, through interpolation.

5.2 Interactive Coordination

A more radical approach is to substitute the sequencer-like platform (currently, Ableton Live) by a more general platform for interactive and collaborative control of musical components (being generated by our current Transformer-based architecture). We are thinking of the Skini platform of our colleague Bertrand Petit [28]. This platform allows defining some kind of "orchestral blueprint? (actually, some cartography of possible paths) for activating various musical components of a piece of music[16]. Paths may be fixed or open with various choices, to be decided according to the interaction with the public (various active listeners). Figure 6 shows an example of visual orchestral blueprint (musical flow) in Skini.

---

[16]It separates the macro-level coordination from the actual micro-level components, as for architectural/coordination languages in software architectures [34] or distributed systems.
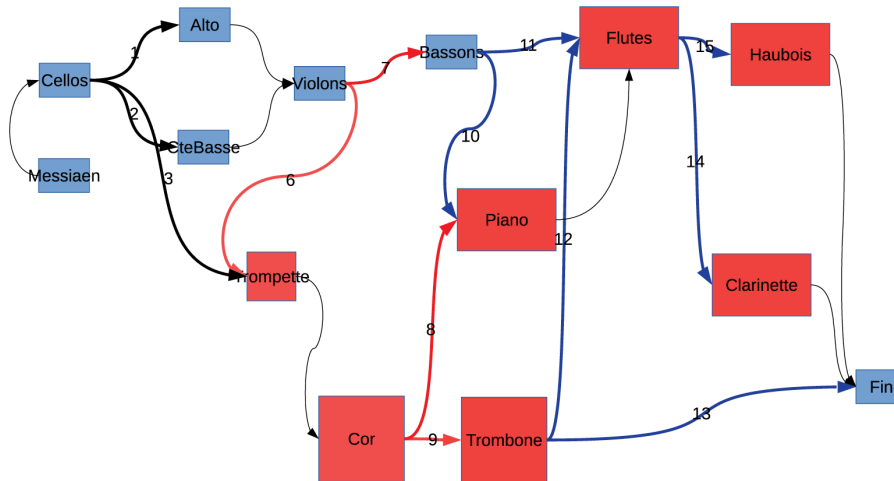
Figure 6: Example of orchestral flow in Skini (Opus1 Piece by Bertrand Petit). Plain arrows represent fixed paths and bold arrows represent paths which may be decided by the public. Each music/sound component (in blue) may be activated an unlimited number of times, except for "reservoirs" (in red) which are set to have some maximum number of activations

The control expression in the Skini platform is based on the integration of the synchronous reactive programming language Esterel [2] in JavaScript (on the Web). The advantage over a sequencer (which has a semi-rigid temporal structure) is the expressive power (Turing complete) of a language like Esterel (which, for example, is used to control Airbus planes), to program any type of coordination of real-time musical events, depending on various in-game events. Additionally, Esterel has formal semantics and property verification tools, thus offering possibilities of formally verifying properties, such as the termination or non-simultaneous activation of two arbitrary musical components. The Skini platform's capability for collaborative interactive orchestration (e.g., for simultaneous control interactions by several actors) offers us the right level of management of various interactions and controls coming from the game engine and from the different players. The Skini platform (whose architecture is shown in Figure 7) has already been tested recently, in a first scenario with a game platform (Unreal Engine 4), to control the scheduling and musical orchestration depending on the situation of the player within the game [27].

6   Conclusion

In this paper, we have presented some architecture, based on deep learning, for generating music for video games. Our current architecture is a proof of concept, but we believe that it shows the way for future developments that we have sketched. We are currently working on the design of next version architecture and its coupling with the coordination level based on Skini.

Acknowledgments

References

[1] Ableton team. Ableton Live. Ableton, Last access on 16/06/2022. Music creation and performance software.

[2] Gérard Berry and Georges Gonthier. The Esterel synchronous programming language: Design, semantics, implementation. *Science of Computer Programming*, 19(2):87–152, 1992.

[3] Blizzard Team. StarCraft II: Wings of Liberty. Blizzard Entertainment, 2010.

[4] Jean-Pierre Briot. From artificial neural networks to deep learning for music generation – History, concepts and trends. *Neural Computing and Applications (NCAA)*, (33):39–65, January 2021. Special issue Neural networks in Art, sound and Design, edited by Juan Romero and Penousal Machado.
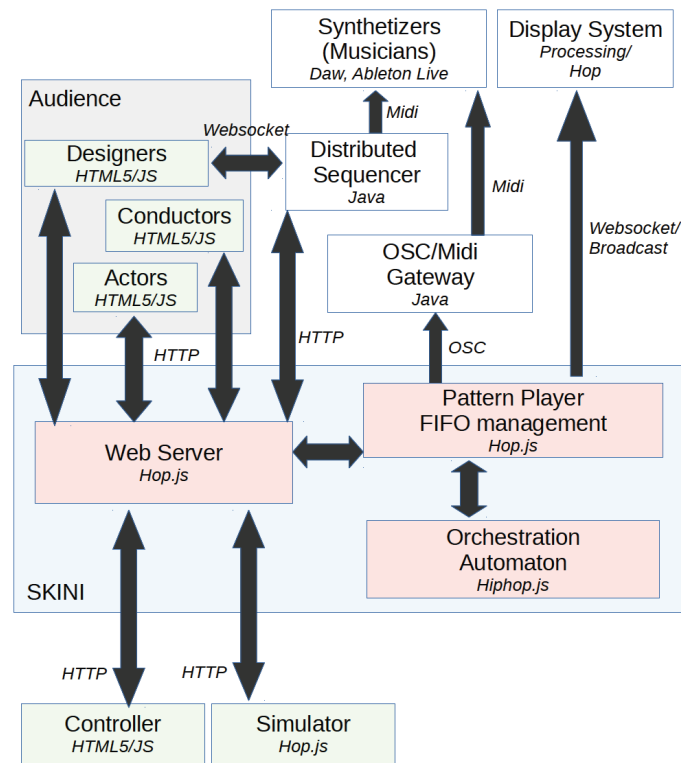
Figure 7: Skini architecture

[5] Jean-Pierre Briot, Gaëtan Hadjeres, and François-David Pachet. *Deep Learning Techniques for Music Generation*. Computational Synthesis and Creative Systems. Springer, 2019.

[6] Allan M. Collins and Elizabeth F. Loftus. A spreading-activation theory of semantic processing. *Psychological Review*, 82(6):407–428, 1975.

[7] Karen Collins. *From Pac-Man to Pop Music Interactive Audio in Games and New Media*. Ashgate Publishing Ltd, 2011.

[8] Steve Engels, Tiffany Tong, and Fabian Chan. Automatic real-time music generation for games. In *Eleventh Annual AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-15)*, pages 220–222, 2015.

[9] Gaëtan Hadjeres and Frank Nielsen. Interactive music generation with positional constraints using Anticipation-RNN, September 2017. arXiv:1709.06404.

[10] Curtis Hawthorne, Erich Elsen, Jialin Song, Adam Roberts, Ian Simon, Colin Raffel, Jesse Engel, Sageev Oore, and Douglas Eck. Onsets and frames: Dual-objective piano transcription, June 2018. arXiv:1710.11153.

[11] Lejaren A. Hiller and Leonard M. Isaacson. *Experimental Music: Composition with an Electronic Computer*. McGraw-Hill, 1959.

[12] Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Ian Simon Curtis Hawthorne, Andrew M. Dai, Matthew D. Hoffman, Monica Dinculescu, and Douglas Eck. Music Transformer: Generating music with long-term structure, December 2018. arXiv:1809.04281.

[13] Aline Hufschmitt, Stéphane Cardon, and Éric Jacopin. Dynamic manipulation of player performance with music tempo in Tetris. In *26th International Conference on Intelligent User Interfaces*, IUI '21, pages 290–296, College Station, TX, USA, 2021. ACM.

[14] Patrick Hutchings and Jon McCormack. Using autonomous agents to improvise music compositions in real-time. In João Correia, Vic Ciesielski, and Antonios Liapis, editors, *Computational Intelligence in Music, Sound, Art and Design – 6th International Conference, EvoMUSART 2017, Amsterdam, The Netherlands, April 19–21, 2017, Proceedings*, volume 10198 of *LNCS*, pages 114–127. Springer, 2017.

[15] Patrick Hutchings and Jon McCormack. Adaptive music composition for games. *IEEE Transactions on Games*, 12(3):270–280, 2020.

[16] Hyperbits. Layering music: 20 ways to layer sounds. Hyperbits, Last access on 16/06/2022. Blog.

[17] Dan Jeffries. The musician in the machine. Magenta blog, August 2020.

[18] Dan Jeffries. Ambient songs for creativity and calm. Spotify, 2022. Playlist.

[19] Patrik N. Juslin. What does music express? basic emotions and beyond. *Frontiers in Psychology*, 4:Article 596, 2013.

[20] Magenta. Make music and art using machine learning.

[21] Gerhard Nierhaus. *Algorithmic Composition: Paradigms of Automated Music Generation*. Springer, 2009.

[22] François Pachet. Musical virtuosity and creativity. In Jon McCormack and Marc d'Inverno, editors, *Computers and Creativity*, pages 115–146. Springer, 2012.

[23] François Pachet and Pierre Roy. Markov constraints: Steerable generation of Markov sequences. *Constraints*, 16(2):148–172, 2011.

[24] Pachyderm team. Pachyderm. Github, Last access on 16/06/2022. Code documentation.

[25] Alexandre Papadopoulos, François Pachet, and Pierre Roy. Generating non-plagiaristic Markov sequences with max order sampling. In Mirko Degli Esposti, Eduardo G. Altmann, and François Pachet, editors, *Creativity and Universality in Language*, Lecture Notes in Morphogenesis, pages 85–103. Springer, 2016.

[26] Alexandre Papadopoulos, Pierre Roy, and François Pachet. Assisted lead sheet composition using FlowComposer. In Michel Rueher, editor, *Principles and Practice of Constraint Programming: 22nd International Conference, CP 2016, Toulouse, France, September 5-9, 2016, Proceedings*, Programming and Software Engineering, pages 769–785. Springer, 2016.

[27] Bertrand Petit. Skini et jeu vidéo, January 2021. Blog.

[28] Bertrand Petit and Manuel Serrano. Skini: Reactive programming for interactive structured music. *The Art, Science, and Engineering of Programming*, 5(1):Article 2, June 2020.

[29] Michael Phi. Illustrated guide to Transformers – step by step explanation. Towards Data Science, April 2020. Blog.

[30] Cale Plut and Philippe Pasquier. Music matters: An empirical study on the effects of adaptive music on experienced and perceived player affect. In *2019 IEEE Conference on Games (CoG)*, pages 1–8, 2019.

[31] Cale Plut and Philippe Pasquier. Generative music in video games: State of the art, challenges, and prospects. *Entertainment Computing*, 33:100337, 2020.

[32] James A Russell. A circumplex model of affect. *Journal of Personality and Social Psychology*, 39(6):1161–1178, 1980.

[33] Timothy Sanders and Paul Cairns. Time perception, immersion and music in videogames. In *Proceedings of the 24th BCS Interaction Specialist Group Conference*, BCS '10, pages 160–167, Swindon, U.K., 2010. BCS Learning & Development Ltd.

[34] Mary Shaw and David Garlan. *Software architecture – Perspectives on an emerging discipline*. Prentice Hall, Upper Saddle River, N.Y., USA, 1996.

[35] Solarus Team. The Legend of Zelda: Mystery of Solarus. Solarus, 2011. Game.

[36] Keith Stuart. 'Mozart would have made video game music': composer Eímear Noone on a winning art form. *The Guardian*, October 2019.

[37] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. WaveNet: A generative model for raw audio, December 2016. arXiv:1609.03499.

[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, December 2017. arXiv:1706.03762.

[39] Vlambeer. Luftrausers. Devolver Digital, 2014. Game.

[40] Paul Weir. The sound of 'no man's sky', 2017.

[41] S. W. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.