Generating Real Time Adaptive Music for a 2D Platformer

Nuno Costa
Faculty of Engineering of the University of Porto
Porto, Portugal

Abstract

The immersion of players in videogames is greatly affected by the presence of adaptive music. This report delves into the generation of music in real-time using Markov chains, with the objective of dynamically adjusting the music to align with the current state of the game through the modification of note properties before their execution. Furthermore, unsuccessful approaches in addressing this issue are also examined and expounded upon, followed by the presentation of key findings and main takeaways.

Keywords: interactive music, markov chains, music generation

1 Introduction

The profound impact of music in evoking emotions and enhancing immersion is widely recognized. It is rare to find communication mediums that do not harness this power to some extent, as music adds an additional dimension of sensory experience. One such medium that can potentially leverage this dimension effectively, given its interactive nature, is video games.

In the realm of video games, the presence of music is essentially indispensable. It is exceedingly rare to encounter games without a musical soundtrack, and for good reason. However, the task of creating music specifically tailored for video games is not without its challenges. Composing a unique soundtrack alongside storyline development, artistic design, and level creation requires significant skill and iteration. Consequently, even with the necessary expertise, it is unlikely that the duration of the soundtrack will match that of the gameplay due to the high financial costs associated with such an extensive compositional undertaking.

The primary objective of this project is to address this issue by developing a model based on Markov Chains, which simplifies the music composition process through automatic generation. The secondary aim is to apply this model within a demo game and adapt the generated music to the gameplay, by controlling the pitch and velocity of the notes in the generated music. For simplicity, the chosen game was *Cavyn*, a 2D Platformer.

This report is structured into five chapters. The first chapter provides an overview of music generation techniques. The second chapter elucidates the design considerations of the game that allow the adaptive nature of the generated music. The third chapter, in essence, bridges the preceding chapters by exploring the integration of music and gameplay. The fourth chapter analyzes unsuccessful approaches encountered during the music generation process and the subsequent real-time implementation of the model within the video game. Finally, the fifth and final chapter examines the results, while also proposing ideas for future endeavors.

2 Music Generation

One of the goals of this project was to expand upon a previous endeavor by enabling the generation of multi-instrument music using Markov models. To achieve this, the following approach was undertaken.

Initially, a collection of video game music was gathered and imported using the Python library music21 [2]. All the music obtained was in MIDI format, which facilitated extensive property management and control. Using this library, the music was divided into its individual instrument components, resulting in separate musical scores for each instrument. Considering that the underlying musical data adhered to a 4/4 time signature, the scores for different instruments were further divided into measures. These measures were subsequently organized into groups based on their temporal placement. Consequently, the output of this step was a list of measure groups, wherein each group contained the measures for each instrument that, when played together, constituted a single measure of the original music. Playing through the entire list of measure groups sequentially would reproduce the initially imported music.

This representation allowed for the creation of a Markov Model for generating multi-instrumental music, as the measures could be grouped and maintain their rhythmic properties. However, to construct the model itself, it was necessary to evaluate the similarity between two measure groups.

To accomplish this, a similarity matrix was generated. When comparing two measure groups, each consisting of percussion and piano measures, for instance, the comparison was performed by individually assessing the similarity of the percussion measures and the piano measures. The textual representation provided by the music21 library proved useful for comparing same-instrument measures. By converting the measures into textual representations, the *Levenshtein* distance algorithm was employed to determine their dissimilarity. This algorithm calculated the minimum number of single-character edits required to transform the first measure into the second one. Utilizing this information, the similarity of two measure groups was derived by summing the dissimilarities of their individual measures and calculating the inverse value. To prevent division by zero, two measures were considered similar if their dissimilarity distance was 1.

Having established the similarity matrix, generating a transition matrix became straightforward. Given that the natural progression of the song involves playing the measure groups sequentially, the probability of transitioning from measure group i to measure group j was determined based on the similarity between the measure group j and the subsequent measure group $i + 1$.

With all the necessary calculations in place, generating multi-instrument music became a straightforward process. By selecting an initial index to indicate the first measure group and utilizing the transition matrix, which accounted for the probability of transitioning from measure group i to measure group j, the subsequent measure was chosen. After specifying the desired number of measures to generate, the resulting music was outputted into a MIDI file for auditory evaluation. The generated song, consisting of 100 measures, can be found in the folder "generative-model" under the filename "test.mid," which can be compared to the original sample used, "original.mid."

3 Adaptive Game Properties

The selected game for this project was *Cavyn* [1], an open-source 2D Platformer seen in Figure 1. *Cavyn* was chosen as the foundation for the proof-of-work due to its simplicity, implementation in Pygame [5] (which facilitates compatibility with the music generative model), and its adjustable game properties that enable greater control over the music, thereby enhancing immersion.

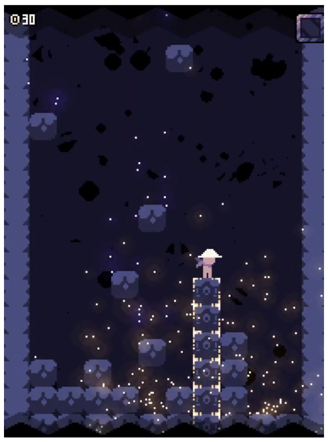

Figure 1. Cavyn, an open-source 2D Platformer

In *Cavyn*, the player's objective is to avoid falling blocks while collecting coins and special power-ups. These power-ups introduce specific parameters that impact gameplay and provide opportunities for the music adaptability process. The following power-ups are available:

- "Warp" teleports the player to the stack of blocks with the greatest height.
- "Jump" enables the player to perform higher jumps.
- "Cube" allows the player to construct blocks underneath their current position.

These properties create openings for the music adaptability process. The exploration of how these properties were utilized is further detailed in the subsequent section.

4 Music Integration

Given the dynamic nature of the game, real-time adaptations were implemented to modify the generated sounds. These adaptations primarily involved note transpositions and tempo changes. The following interactions were introduced within the game:

- When the player possesses the warp power-up, a continuous synth voice note is played, and all other generated notes are transposed down by 2 octaves.
- When the player possesses the jump power-up, a continuous synth pad note (new age) is played, and all other generated notes are transposed up by 2 octaves.
- When the player possesses the cube power-up, a continuous synth effects note (echoes) is played, and all other generated notes have their velocity increased by 20.
- For each collected coin, the generated music's beats per minute (BPM) is increased by 1, with an initial value of 50 BPM.

These changes provide a sense of progression for the player throughout the game, directly impacting their immersion by enhancing the connection between gameplay events and the accompanying music.

However, a significant challenge in this project was achieving real-time music generation. To accomplish this, the music21 scores were converted into MIDI tracks, consisting of a sequence of MIDI events that are executed by a MIDI synthesizer. However, directly converting these events into actual sounds in real-time proved to be problematic, as explained in the subsequent chapter. Therefore, it was necessary to introduce alterations to these events.

To address this issue, all MIDI events were iterated through. A timestamp was maintained to determine when a note should be played. During the iteration of the MIDI track, three possible cases could occur: a DeltaTime event, indicating a pause between notes in MIDI ticks, which had to be converted to milliseconds; a NOTE_ON event, indicating the execution of a specific pitch with a given velocity in a particular MIDI channel; and a NOTE_OFF event, serving as the inverse of a NOTE_ON event. These NOTE_ON and NOTE_OFF events were placed in a priority queue based on their respective timestamps.

With the priority queue properly established, the next step was to check, during each game loop, whether the top note in the priority queue should be triggered at that particular time, considering the current time in Pygame's MIDI clock. However, implementing this check within the game loop resulted in significant lag due to the computational demands involved. To overcome this challenge, a multi-threaded approach was adopted, with separate threads dedicated to music generation and music output (note triggering).

With the implementation of this multi-threaded system, the generative music became interactive, successfully achieving the project's goal of real-time music generation.

A demo can be found here.

5 Difficulties and Unsuccessful Approaches

Several approaches were undertaken to address the music generation objectives; however, the outcomes lacked tangibility, necessitating further exploration.

Initially, the project aimed to employ machine learning and deep learning techniques for music generation. However, based on the obtained results, the project scope was modified to focus on utilizing Markov Chains as the foundation for the generative component.

Regarding the deep learning methods, the diffusion models provided by Google's Magenta project [3] were initially investigated. These models leverage techniques such as variational autoencoders to generate complex musical compositions. Regrettably, the generated results exhibited low quality, and as attempts were made to train them with larger datasets, limitations in available RAM became a considerable obstacle. Subsequently, LSTM models were explored, but transitioning the multi-instrument nature of the input data into a MIDI sequence posed challenges. Most available LSTM models were primarily trained for single-instrument scenarios, predominantly focusing on piano music. Furthermore, the persisting RAM limitations made it impossible to train these models effectively and produce relevant results.

Consequently, the decision was made to pivot towards the utilization of Markov Chain generation, as described in Section 2, given its potential to overcome the limitations encountered with the deep learning approaches.

Additional challenges emerged when attempting real-time music generation. Specifically, efforts were made to output MIDI data through socket connections and employ API libraries like rtmidi [4], which offer functionalities for MIDI communication. These libraries aim to facilitate the integration and control of MIDI devices. However, complications during the installation and detection of MIDI synthesizers hindered the implementation of this solution. As an alternative, the conversion from MIDI events to Pygame events was explored as a workaround, but unfortunately proved unsuccessful. Consequently, the decision was made to employ a priority queue of MIDI events, as explained in Section 4 to ensure efficient real-time music generation. However, despite the implementation of this solution, certain side-effects have been observed, the underlying causes of which have yet to be fully elucidated. One notable side-effect is the occurrence of extended periods of silence, resulting from the unnecessarily long calculated timestamps for subsequent notes. Efforts were undertaken to address this issue, albeit without success. Nevertheless, the overall outcome of the project remains satisfactory.

As stated, the encountered obstacles were significant and overcoming them played a vital role in the development of this project.

6 Results and Final Considerations

In conclusion, this project aimed to address the challenge of creating tailored music for video games by developing a model based on Markov Chains for automatic music generation. The model was implemented and integrated into the *Cavyn* game, allowing for real-time adaptation of the generated music based on gameplay events. The project successfully achieved the goal of interactive and dynamic music generation within the game.

In the future, there are several areas of potential improvement and expansion. Firstly, exploring more advanced deep learning models and techniques specifically designed for multi-instrument music generation could lead to more sophisticated and realistic results. Alongside this, finding a way to handle the MIDI events in a more systematic way would also improve the final result and make sure that some unexplained side-effects of the current solution, such as the long silence breaks, would not happen.

Furthermore, expanding the scope of the project to include other genres of video games and exploring different musical styles and moods would provide more diverse and engaging experiences for players.

Overall, this project has demonstrated the potential of using automated music generation techniques, particularly Markov Chains, in the context of video games. By enabling real-time adaptability and integration with gameplay events, the project has contributed to enhancing the immersive and interactive nature of video game experiences through dynamic and tailored music.

References

[1] 2022. Cavyn by DaFluffyPotato. https://dafluffypotato.itch.io/cavyn [Accessed May 28, 2023].

[2] Michael Cuthbert, Christopher Ariza, Josiah Wolf Oberholtzer, et al. 2008. music21. https://web.mit.edu/music21/.

[3] Adam Roberts, Curtis Hawthorne, Ian Simon, et al. 2016. Magenta: Music and Art Generation with Machine Intelligence. https://magenta.tensorflow.org.

[4] Gary P. Scavone. 2003. RtMidi: realtime MIDI i/o C++ classes. https://www.music.mcgill.ca/~gary/rtmidi/.

[5] Pete Shinners and Contributors. 2000. Pygame.