MUSIC GAME MAP GENERATION: MELODY SABER

Chenhe Gu
New York University Abu Dhabi
cg3293@nyu.edu

Keyin Wu
New York University Shanghai
kyw259@nyu.edu

Ziying Wang
New York University Shanghai
zw1745@nyu.edu

ABSTRACT

Melody Saber is an immersive music game inspired by the popular rhythm game Beat Saber [1]. Different from Beat Saber which plays out the entire music piece regardless of the player's performance, our Melody Saber allows the user to generate a customized game map of pop songs and play different melody clips depending on how they hit the game block. The user can generate accompaniments based on deep learning models Chorderator [2] and Accomontage [3], choose a track to chop into blocks, and generate the music game map in a Virtual Reality (VR) environment through Unity. By enabling players to generate their unique melody track, Melody Saber creates an innovative music game experience combined with music game map generator.

1. OVERVIEW

We build the Melody Saber, an immersive music game in VR which, unlike traditional music games, allows the user to experience alternative renditions of pop songs and perceive real-time modulations to the music they hear as a result of their actions. Specifically, we map notes from Deep Learning-generated melody sequences to blocks which the player hits along with the music. Moreover, we alter the perceived music if the hit direction is incorrect.

Via Melody Saber, we intend to complete the circle of Music Information Retrieval (MIR) and music synthesis. This is manifested in our input processing workflow (Fig. 1), where we generate completely new accompaniment arrangements from the main melody of famous pop songs and map the results to playable game objects. We further divide our process into three sections as follows:

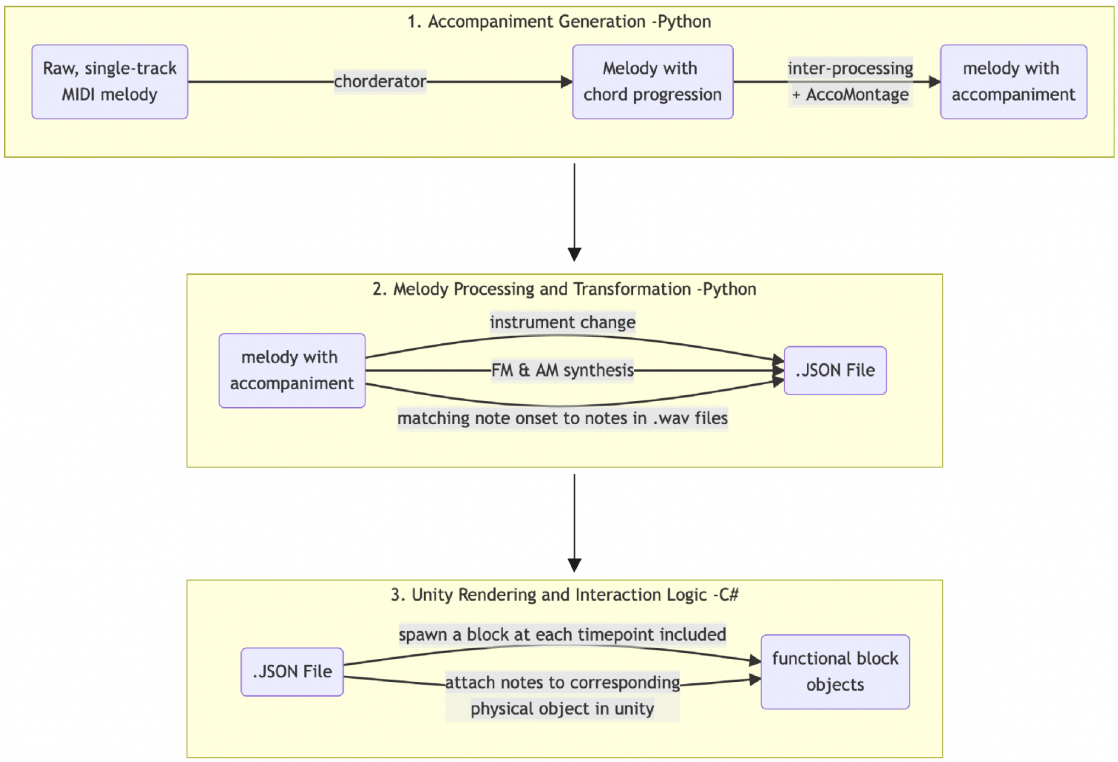

Figure. 1: System Summary

- Accompaniment Generation, which handles raw melody input and adds to it a complex accompaniment sequence;
- Melody Processing and Transformation, which takes the generated song rendition and creates a corresponding .JSON file mapping notes from the accompaniment sequence to appropriate spawn times for game blocks. Along with the original notes, 3 different modulations of the notes are also generated corresponding to 3 incorrect hit directions by the user;
- Unity Rendering and Interaction Logic, which generates blocks so the user could hit them according to the melody progression as it is being played.
-

We elaborate on each of these processes in the following sections.

2. METHODOLOGY

2.1 Accompaniment Generation

We aim to create music pieces that resemble popular hits yet are different, making them refreshing for the player to hear and play with.

We define the input MIDI track as the melody track, which contains a simple melody or the hook of a song. We add a generated MIDI track which serves to enrich and add layers to the main melody. The beginning of a naïve input track is shown in Fig. 2.

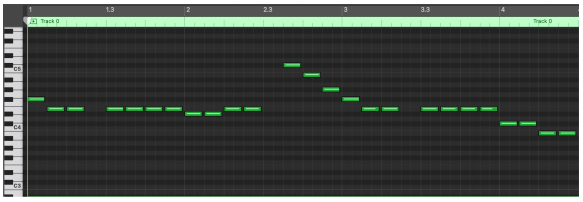

Figure. 2: Piano Roll of the input main melody (from Levels by Avicii) (GarageBand).

We adopt Chorderator, a rule-based chord generation tool that leverages dynamic programming techniques to match melody phrases to chords, which it draws from a collection of professionally-made chords [2]. We additionally provide the program with the tempo and key information which we look up in a DAW. It is also notable that Chorderator requires in its input a custom division of bars in order to distinguish each phrase, which we also supply. We observe that Chorderator achieves the optimal results with simplistic melodies with fixed phrase lengths. As illustrated in Fig 3, the resulting chord sequence for the Fig. 2 input is a series of complex chords. However, the result is still not sophisticated enough to be directly used as the accompaniment for the main melody. We therefore leverage AccoMontage [3], a deep-learning-based generative tool combining phrase selection and recombination via CNN as well as music style transfer using an VAE framework.

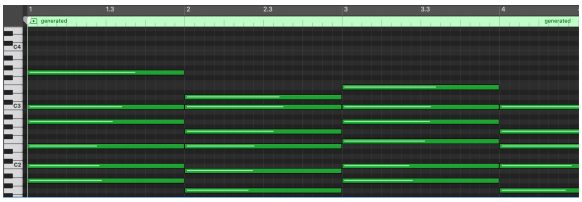

Figure. 3: Piano roll of chord progression generated from Chorderator (GarageBand)

We process the Chorderator output according to the input restrictions of AccoMontage as follows:

- We adjust the lengths of notes such that all notes in a chord share the same length.
- AccoMontage specifies a set of differences in partials between notes in a chord, which all chords need to satisfy. Additionally, only chords with 3 or 4 notes are considered. For each chord generated with more than 4 notes, we iteratively compute all differences in partials of 4-note and 3-note combinations and select the optimal one satisfying AccoMontage requirements. We prioritize 4-note chords over 3-note ones and chords with higher pitches over low-pitched chords in order to best preserve the musical texture of the Chorderator output.
- When no suitable chords can be found, we look up the corresponding note from the main melody and fill in a generic 3-note major chord based on that note, as it theoretically harmonizes with the main melody.

An example of processed output can be seen in Fig 4.

Figure. 4: Piano roll of processed chorderator output (GarageBand)

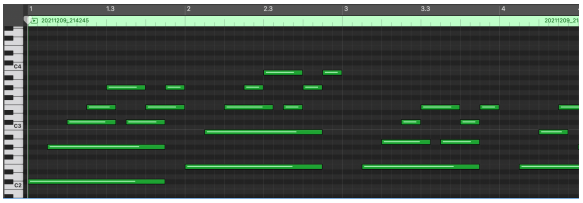

Figure. 5: Piano roll of generated accompaniment sequence (GarageBand)

We therefore are able to obtain a more sophisticated accompaniment sequence given the processed output.

2.2 Melody Processing and Transformation

In order to input the generated music into the game interface, we convert each .MIDI file into small .WAV clips, along with a corresponding .JSON file that can be read in Unity. The conversion will also alter the original music by modulation and instrument replacement to map different user actions in future game implementation.

The processing is track-based using pretty_midi library in Python. Normally the generated music will have two tracks. The user can decide which track to be processed by inputting a track number. Based on the list of notes on the chosen track, we group the notes depending on their start time interval. Each group of notes outputs a .WAV file that will be mapped to each block that the player hits in the game. Since there should leave some space between every two blocks, the time interval of groups should have a minimum, which, for example, was set to one second in our demo. After we fetch the start time of the first note in each group, we are able to bond the time when a game block should be initiated with the .WAV file that the block carries.

The clip of original music will be attached to the correct direction of each game block. The synthesized music clip by Frequency Modulation (FM) and Amplitude Modulation (AM) will be respectively attached to the relative left and right direction of the correct direction. The altered music with an instrument change will then be attached to the opposite of the correct direction. We have tested on a few tracks which shows both the melody track and the accompaniment track can produce relatively good results where the music can be recomposed through the game. In our demo, we chose to process the main melody track to generate a simpler map and decrease the difficulty. The information of each group of notes is stored in a dictionary, which can be passed as a .JSON file to Unity for game implementation. The .JSON file gives the timestamp when each game block should be hit and identifies the four music clips that each block may play according to different user actions. The structure of the .JSON file is as follows:

```
{
  "3.25":[
    {
      "wav_top" : "0_3.25.wav",
      "wav_left" : "0_3.25_left.wav",
      "wav_right" : "0_3.25_right.wav",
      "wav_btm" : "0_3.25_btm.wav"
    }
  ],
  "4.25":[
    {
      "wav_top" : "0_4.25.wav",
      "wav_left" : "0_4.25_left.wav",
      "wav_right" : "0_4.25_right.wav",
      "wav_btm" : "0_4.25_btm.wav"
    }
  ],
  ……
}
```

2.3 User Interactions and Design

We import both the .JSON file with note information and the folder of the chopped .WAV file into Unity. We initiate a timer at the beginning of each map and instantiate the block(s) every time the timer reaches a timestamp according to the .JSON file. When the block(s) is being generated, the audio file path for each side will be written into the four empty audio sources (each tagged with one of the four directions) tied to the block(s).

To detect the face the user sliced in, we captured the instantaneous position of the saber the moment it touches the block and calculated its position change within a frame count. The audio file linked to the

corresponding direction will play if the player hits the block with their saber of the same color.

The unsliced track of the two (accompaniment track in our demo) is written into the background music audio source. We get the spectrum data of the background music track, separate the spectrum into 8 different bandwidths and create audio visualization that would animate when the track is being played according to the amplitude of each bandwidth.

Figure 6: Audio visualization in game.

To improve the gaming experience, we built a neon light environment, sabers, and blocks. If sliced from the correct direction (corresponding with the arrow direction), the block will burst into multiple cubes. We also create a menu scene for track selection. Through a ray cast from the right-hand saber, the player can point at one of the three album covers, press down the selection trigger, and start playing the corresponding map.

3. EVALUATIONS

We conducted multiple user-testing for Melody Saber. Most users find the interaction of Melody Saber extremely entertaining. Compared with the classical Beat Saber, our players are intrigued by the diverse styles of melody that can be played by hitting the blocks from different directions.

Figure 7: Professor Gus Xia trying out MelodySaber.

Other than the positive feedback we received, our users also claimed that the map is beginner level and that there are occasional long gaps between two waves of blocks where they find dull.

In order to enrich the player experience and complicate the game map, we need to add layers onto the track that is later split and mapped onto individual blocks. Our current accompaniment generating algorithm cannot create an accompaniment beautiful enough to be layered with the main melody.

In future iterations, we aim to:

- Modify our algorithm and generate a richer accompaniment consisting of multiple layers of melodies and instruments.
- Adopt MIR technique to identify music that will be mapped on the blocks from the music that will construct the background track.
- Optimize block instantiation algorithm to improve hitting movement experience.

4. REFERENCES

[1] Beat Games, *Beat Saber*, May 2018.

[2] Yi, B., 2021. [online] Available at: <https://billyyi.top/research/chorderator/> [Accessed 15 December 2021].

[3] Zhao, J. and Via, G., 2021. *ACCOMONTAGE: ACCOMPANIMENT ARRANGEMENT VIA PHRASE SELECTION AND STYLE TRANSFER*. [online] Arxiv.org. Available at: <https://arxiv.org/pdf/2108.11213.pdf> [Accessed 15 December 2021].