Learning Adaptive Game Soundtrack Control

Aaron Dorsey, Todd W. Neller, Hien G. Tran, Veysel Yilmaz

Gettysburg College
{dorsaa01, tneller, tranhi01, yilmve01}@gettysburg.edu

Abstract

In this paper, we demonstrate a novel technique for dynamically generating an emotionally-directed video game soundtrack. We begin with a human Conductor observing gameplay and directing associated emotions that would enhance the observed gameplay experience. We apply supervised learning to data sampled from synchronized input gameplay features and Conductor output emotional direction features in order to fit a mathematical model to the Conductor's emotional direction. Then, during gameplay, the emotional direction model maps gameplay state input to emotional direction output, which is then input to a music generation module that dynamically generates emotionally-relevant music during gameplay. Our empirical study suggests that random forests serve well for modeling the Conductor for our two experimental game genres.

Introduction

We believe that music can enhance a player's emotional experience of a game in the same way a movie soundtrack cues and heightens emotions of the viewer. A good soundtrack can increase player engagement, whereas a bad soundtrack can be detrimental to the player's experience. In this work, we perform two empirical studies that focus on the use of AI and machine learning techniques to dynamically create and adapt a game soundtrack to enhance a player's experience by attempting to model a human Conductor's emotional direction of the soundtrack to the benefit of the player.

First, we survey related work and overview the architecture of our approach in the field of adaptive and dynamic music composition. Next, we describe the data collection and processing methodology, algorithmic parameters, and results of our experiments along with the music generation models we employed for each experiment (Yilmaz 2022). Finally, we discuss potential future work and share our conclusions.

Related Work

Dynamism was a key point in our research. With respect to Redhead's categorizations regarding dynamism in video game soundtracks, we ultimately wanted the music to adapt to the game-play (adaptivity) as well as be generated by an algorithm, making it generative and algorithmic (Redhead 2018). To that end, Léo provided us the tools to adjust musical elements to reflect the intended tone (Léo 2020). Léo's vertical re-orchestration technique that modified instrumentation while keeping the same melodic and harmonic structure, shaped our first music algorithm approach in this paper.

Another influential work by Rayman provided examples that used dynamic and adaptive music generation algorithms that shaped our first approach (Rayman 2014; Casella and Paiva 2001). The detailed depiction of the adaptive music structure in Mario Kart Racing that was described in the paper influenced our Player-Conductor duality as well as parameterization techniques. Junior on the other hand, gave us the idea of using probabilistic decision with contrapuntal music composition, influencing our own Markov chain implementation (Escarce Junior et al. 2021). The source limited itself to an arpeggio-oriented musical structure while we extended it using the contrapuntal fundamentals, believing that a probabilistic approach may prove useful.

Plut (Plut and Pasquier 2020) lists out the current limitations of adaptive and generative game soundtrack composition techniques while providing an overview on some adaptations in the gaming industry. It is pointed out that some games can take up to 100 hours to complete, making manual composition of novel music prohibitively expensive. Further, Plut explains that ". . . adaptive music cannot necessarily match the extreme breadth of gameplay possibilities." He further explains how the current level of generative music is deemed too risky to be adopted even by large game companies with good financial means, which in turn limits pursuit of this academic field. While this resource helps us define the problem of diversity in game soundtracks, it also helps us with some tools to address this problem as well as help us define some useful terms such as "vertical composition" and "instrument groups". We utilize such concepts in producing adaptive and generative game soundtrack algorithms.

Architectural Overview

In this section, we overview our approach to learning the adaptive AI controller for a game's soundtrack and the main soundtrack loop using that controller. The learning architecture is depicted in Figure 1. There are two human roles: Player and Conductor. The *Player* acts as a playtester,

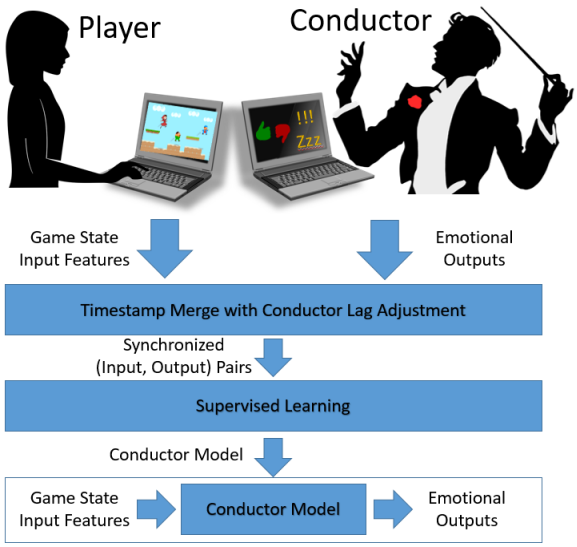

Figure 1: Learning architecture

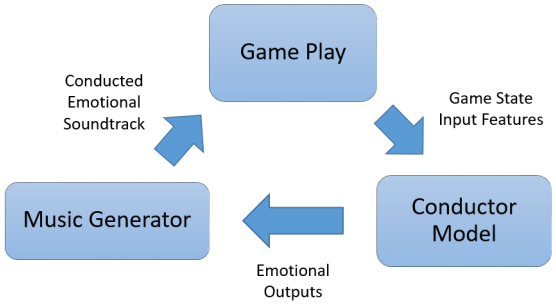

Figure 2: Soundtrack generation cycle.

playing so as to broadly sample a range of possible game states. The *Conductor* observes Player gameplay and provides soundtrack emotional cues to be paired with game states they observe. Timestamped state input features are sampled and recorded from gameplay, while timestamped emotional output features are recorded from the Conductor observing the gameplay.

These input and output features are then merged according to their asynchronous timestamps as described in the section "Experiment 1: Bouncy Shots". Timestamps of the emotional output features are adjusted for both computer clock synchronization plus a delay to account for Conductor feedback reaction time. With timestamp-merged input and output feature pairs, we then apply supervised learning to produce a predictive model from inputs (game-play states) to outputs (the Conductor's emotional associations).

A separate music generation module (Figure 2) takes emotional direction predicted by the learned model and produces the soundtrack in real-time in response to gameplay states. For example, we might prefer major or minor scales and chords for states tending towards winning or losing, respectively. Also, we might prefer greater volume and more

Model	**RMSE (Intensity)**	**RMSE (Winning)**
Baseline	0.435	0.196
Linear Regression	0.245	0.192
Gradient Boosting	0.389	0.179
Bayesian Ridge	0.387	0.193
Random Forest Regression	0.199	0.161
Lasso (alpha=1)	0.299	0.229

Table 1: Bouncy Shots Model Statistics

frequent percussion for states that are more intense.

It is important to observe that the Conductor AI is aware of the Conductor's intended emotional state for the Player as opposed to the Player's actual emotional state. We intend that the Conductor be free to influence the emotional experience of the game through adaptive soundtrack control.

Experiment 1: Bouncy Shots

Bouncy Shots (DaFluffyPotato 2022) is a simple single player game where the objective is to push the ball past an AI opponent into the other side's goal to score. Both players can knock the ball to their opponent's goal through collisions with projectiles shot from their own base. Over time, the game gets more complicated and hectic. If enough time passes by without a point being scored, additional balls will be sequentially added to the field. Also, the precision and accuracy of the opponent increases with the player's score, so the game becomes increasingly difficult over time. The game ends once the opponent scores.

Experimental Design

For our first experience, we collected over 120,000 timestamp-merged input-output pairs from over 2.8 hours of gameplay (which was enough to thoroughly explore a small game such as Bouncy Shots). The gameplay input features that we collected from Bouncy Shots are:

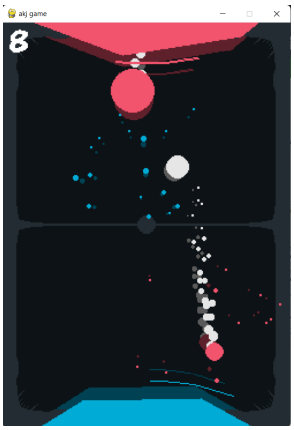

Figure 3: Example of the game Bouncy Shots.

- Score (integer, range $[0, 16]$)
- Number of balls on screen (integer, range $[1, 5]$)
- Number of projectiles on screen (integer, range $[0, 9]$)
- Number of player input events per 0.1 seconds averaged over the last 0.5 seconds. (float, range $[0.0, 9.0]$)
- Average distance of balls from the middle of the screen (pixels, range $[-185.0, 185.0]$)
- Fraction of balls on the player's side (float, range $[0.0, 1.0]$)
- Average forward/backward velocity of the balls (float, range $[-3.37, 3.61]$)

The emotional outputs are:

- Intensity (float, range $[0.0, 1.0]$)
- Winning Status (float, range $[0.0, 1.0]$)

All state variable features are sampled every 100 milliseconds. Since different features were on completely different orders of magnitude, collected data was normalized with min-max scaling (scikit-learn.org 2022), where each value was proportionally scaled to the range $[0, 1]$. This is to ensure that all models can work with the same normalized data for better comparison independent of feature magnitude.

For our data collection process, one member of our group (designated the Player) generated the described input variables from their gameplay, which was automatically collected by our program, while another member (designated the Conductor) provided real-time judgment of the intensity and winning status output of the game while watching the gameplay. To achieve this, the Conductor used an interface to continually direct the perceived gameplay emotions of success and intensity on a scale of 0 to 1. Since this emotional data recording is inherently subjective, we rotated the roles of Conductor and Player among research team members in order to get a more balanced analysis of the games. Because the human Conductors conducted on different gameplays, it is not possible for us to provide a standard deviation on conductors differences.

In order to facilitate mathematical modeling of the Conductor's musical direction, we needed to first merge the timestamped sampling of gameplay feature data with the timestamped sampling of the Conductor's emotional direction. The gameplay data and the emotional input data have a different number of timestamps, and there is no guarantee that a timestamp from gameplay data will also match one of the emotional data and vice versa.

Our merge algorithm takes the timestamped input game state features and output emotional directions as *two queues of sampled events*. Initially, and when the two queue head timestamps are equal, we dequeue and merge the events of both queues into an input-output pair (initially with the greater timestamp). In all other cases, we dequeue the input/output event with the lesser timestamp, and merge it (using that timestamp) with the most recently dequeued output/input event, respectively.

Before collecting data, we synchronized timestamps across computers using the time.gov website to measure the difference between the clocks of the Player and Conductor computers. We then divided the role of Player and Conductor among ourselves, and began playing the game to generate our data. After that, we used our merging program to process the emotional data and gameplay data to supply input/output pairs for supervised learning.

From our collected data, we want to create a mathematical model to substitute the directing of the Conductor in real-time. In order to make sure our machine learning method was useful, we also tested a simple baseline heuristic that did not use AI: we predicted intensity based solely off of the number of balls on screen and winning based off of the average ball distance from the middle of the screen. We had to find a predictive analysis machine learning model that would suit our dataset. For our choice of error measure, we decided to use root mean squared error (RMSE), so we used models (with default hyperparameters) appropriate for RMSE, such as ordinary least squares linear regression, gradient boosting regression, Bayesian ridge regression, lasso regression (with alpha = 1), and random forest regression. We performed train-test-split with a size of 50-50 on these above mentioned models and all RMSE values are measured on prediction of the held-out test data.

Experimental Results

Based on the models tested (Figure 1), we can see that not only are most machine learning models better than the baseline in both respects, but the random forest regression gives the lowest RMSE and the most accurate predictions of emotional responses from game data. We used the default hyperparameters provided by scikit-learn for testing. When playing the game after incorporating our different models, we subjectively judged that the best play experience came from the random forest regression model. The music changed more fluidly and noticeably compared to other models.

One source of bias that may arise from using human-created data is the time it takes for the humans to react to a rapidly changing game state. To correct for this, we added different amounts of time to the timestamps of game data so that a potentially late reaction of the Conductor and a game-changing event can be matched up (Figure 4). We see that a lag value of 250 milliseconds provides the overall lowest RMSE for both intensity and winning status, meaning that it is the optimal amount of lag for the most predictive power for this experiment.

We also tested whether a polynomial regression of higher degree might have better predictive power than a linear regression (Figure 5). We tested from one degree up until four, and we see that the RMSE remains the lowest for both intensity and winning status at one degree. The best model for Bouncy Shots, therefore, is random forest regression where the game data it trains on has been adjusted forward 250 milliseconds, and calculates a first-degree polynomial. For one trying our system on a similar game to Bouncy Shots, these parameters would be a good place to start.

Musical Structure for Experiments I and II with Adaptive Instrumentation Volume

Given the input intensity and winning state emotional values from the model, we designed a method to adapt com-

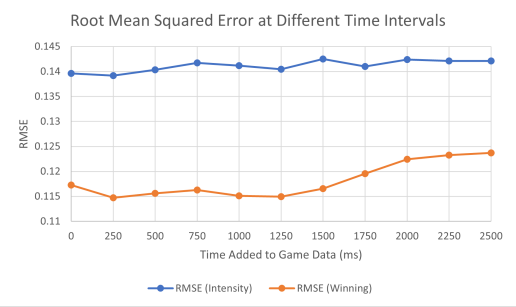

Figure 4: Lagged RMSE values of Bouncy Shots.

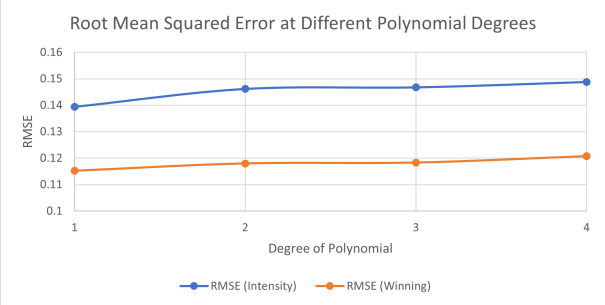

Figure 5: RMSE values of Bouncy Shots varying polynomial degrees.

posed music using those two variables. For experiment purposes, we rely on a base MIDI soundtrack that was composed by one of our members. Our approach utilized the instrumentation technique, in which the tone of the music was adjusted using different instruments playing the same melody/progression (Collins 2008). This technique, although simple, proved quite effective (Figure 6).

The chord progression is in a simple Tonic-Predominant-Dominant-Tonic form as shown in Figure 7. The simplicity of the form helps the melody easily sound major (happy) or minor (sad) only using the instrumentation technique. In total, there are 4 prerecorded tracks that play the same music with very small variances and different instruments. The tracks are as follows:

- Winning: String Section
- Intense: Percussion and Choir Sections
- Losing: Piano Section
- Winning and Intense: Brass Section

We judged that (1) the feeling of winning was best given through strings due to their bright color, (2) percussion intensifies the music by giving it a beat, (3) the choir reflects a sense of fullness due to its resonant nature, (4) the brass section gives the impression of glory, which makes it perfect to reflect both intensity and winning values, and (5) the piano part composed in the higher ranges implies sadness and sentiment. From the figure, one could see a direction going from "Losing and Calm" to "Winning and Intense". As the player is winning an intense game, the instrumentation gets tenser.

The volumes of those tracks are directly determined by the intensity and winning status values. An intensity value of 0 means that the calmness value is 1, and the same goes for winning and losing values. So, for example, when the winning value is 0.8 and the intensity value is 0.4 at a given time, the volumes of the tracks at that time are as follows:

- String Section: 0.8, the winning value of the game
- Percussion and Choir Sections: 0.4, the intensity
- Brass: 0.6, arithmetic mean of intensity and winning
- Piano: 0.2, losing value = 1 - winning value = 1 - 0.8

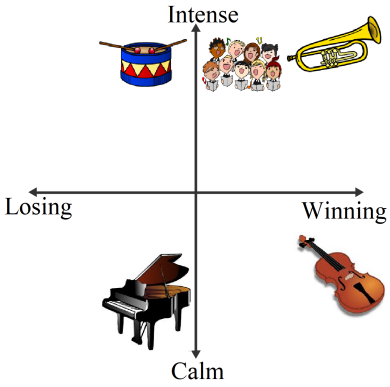

Figure 6: Graph of our instrumentation plan.

Musical Structure for Experiment I with Markov Chain Music Generation

Since counterpoint composition concerns the movement of voices with respect to each other, the principles of counterpoint can be helpful for creating an unending, evolving, adaptive musical soundtrack. Also, a probabilistic system can serve to diversify music. We speculated that a design using a Markov Chain could increase variety while preserving musical continuity. With those ideas, we implemented a 3-track system in which the tracks have distinct rhythmical structures that allow them to move at different times. This way, we preserve a harmonic foundation while also having a movement in music.

The tracks are assigned 3 notes as a starting point, after which the notes in each track are constantly updated. The tracks' notes change two, three, and four times per measure, respectively. Every time that the note played by a track is updated, the algorithm makes a probabilistic deduction with respect to the remaining 2 notes that are kept being played on the other 2 tracks. Note that it is possible for the note to stay the same.

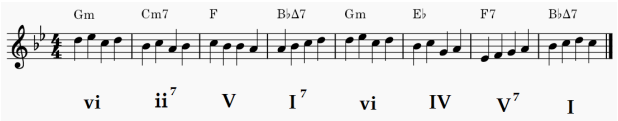

Figure 7: Chord progression.

Chordal harmony, which is the popular music theory approach in which the functions of the chords at a given instance are analyzed, did not fit our needs since we were looking for an ever-evolving and diverse harmonic foundation. On that note, as a neo-classical composer, Paul Hindemith's modern contrapuntal approach to music analysis influenced our design. Let us observe the critical points in Hindemith's work in relation to ours before we explain our evaluation of his approach.

Hindemith explains his *Series I* as follows: "the tone an octave higher stands in so close a relationship that one can hardly maintain a distinction between the two. The tone which is only a fifth higher than the given tone is the next most closely related, and there follow in order the fourth, the major sixth, the major third, the minor third, and so on . . . " (Hindemith 1945)

At this point, we make two assumptions. Firstly, we classified intervals as perfect, major, minor, augmented, and diminished, as in chordal harmony. Further, we used his opinions on the correlation between the distance of the notes and euphony. We will attempt to match the defined inputs with our classification of intervals. And secondly, we neglected the occurrence of any hierarchy between notes and assumed that an interval that is built up is the same as one that is built down. Despite our discrete classifications and definitions so far, Hindemith underlines that "We know that no point can be determined at which 'consonance' passes over into 'dissonance'" (Hindemith 1945). Our probabilistic approach helps us address this issue.

The algorithm's probabilistic structure depends on the relations between notes, called intervals. In total, there are 5 types of interval qualities which are categorized as follows:

- Minor interval $\rightarrow$ Minor Feeling $\rightarrow$ Low Winning Status
- Major interval $\rightarrow$ Major Feeling $\rightarrow$ High Winning Status
- Perfect Interval $\rightarrow$ Mild Feeling $\rightarrow$ Low Intensity
- Diminished Interval $\rightarrow$ Intense Feeling $\rightarrow$ High Intensity
- Augmented Interval $\rightarrow$ Intense Feeling $\rightarrow$ High Intensity

Another determinant in the algorithm is the distance between any two notes. For example, regardless of the quality of the intervals, if the two notes are next to each other, e.g. 'C' and 'D', then there is tension in the harmony that they would create, which is why they are called dissonant intervals. Likewise, any two notes that have more than one whole note between them would have a larger resonance, thus would indicate lower intensity. Those are called consonant intervals, with only one exception: the tritone, which is the interval between two notes that are exactly 6 half notes away from each other (e.g. 'C' and 'F#').

With that information at hand, notice that in a 12-note octave, there are only a few possible ways to make up a chord consisting of 3 notes. We will categorize any chord that has at least one dissonant interval as "dissonant chords", and others as "consonant chords". We later judge that consonant/dissonant chords imply a high/low winning status and low/high intensity, respectively.

Algorithm Initialization The algorithm is a combination of deterministic and probabilistic decisions. At a given instant of a note change, we have the information of two remaining constant notes. First, we deduce the interval of those two notes. Go to **Rule-1** or **Rule-2** if the interval is dissonant or consonant, respectively.

Rule-1 With a probability of $intensity^x$, the updated note also creates a clashing sound with one of the other tracks, resulting in a very high-intensity harmony. Otherwise, follow **Rule-3** if the winning status is less than 0.5 (losing the game), and **Rule-4** otherwise (winning the game).

Rule-2 With a probability of $intensity^x$, the updated note also creates a clashing sound with one of the other tracks, resulting in a very high-intensity harmony. Otherwise, follow **Rule-5** with a probability of winning status, and **Rule-6** otherwise.

Rule-3 With a probability of $winning\ status \times 2$ (since winning status is less than 0.5), the track is updated to form a perfect interval with any of the other 2 tracks. Otherwise, the track is updated to form a minor interval with any of the other 2 tracks.

Rule-4 With a probability of $(winning\ status - 0.5) \times 2$ (since winning status is greater than 0.5), the track is updated to form a major interval with any of the other 2 tracks. Otherwise, the track is updated to form a perfect interval with any of the other 2 tracks.

Rule-5 Update the current track so that the 3 tracks form a major chord. Note that this is always possible since the 2 remaining tracks form either a perfect interval or a minor/major interval.

Rule-6 Update the current track so that the 3 tracks form a minor chord. Note that this is always possible since the 2 remaining tracks form either a perfect interval or a minor/major interval.

$intensity^x$ refers to the value that is deduced by taking the intensity value to the power of any number x, and its function is to provide occasional dissonances that later musically resolve to create musical releases which help create more complex music. With respect to the innate intensity of the game, the value of x could be adjusted. For a low-intensity game, it would be wise to take a high x value to come across dissonances as rarely as possible. For our experiments, we used $x = 2$.

Note that the algorithm eliminates the deterministic chord progression system where winning the game indicates a series of major chords and vice versa. Instead, the frequencies of major and minor chords were adjusted according to the winning status so that the soundtrack consists of both major and minor chords, as is common in music.

Comparison of the Music Algorithms

The Markov-chain-generated (MCG) musical structure has the advantage of not needing a composer and the instrumentation adaptive instrumentation volume (AIV) musical structure has the flexibility to be enriched by a human musician. Both designs adapt the music to the gameplay though the adaptation process is heavily dependent on the skills of a human musician for the AIV model. The MCG method could

Figure 8: Example of the game Chaosman.

Model	RMSE (Intensity)	RMSE (Winning)
Baseline	0.578	0.377
Linear Regression	0.200	0.177
Gradient Boosting	0.261	0.161
Bayesian Ridge	0.259	0.178
Random Forest Regression	0.097	0.075
Lasso (alpha=1)	0.235	0.192

Table 2: Chaosman Model Statistics

also be influenced by the sound designer by choosing different x values for the power of $intensity$, but it is obviously quite limited with respect to the AIV model.

We found listening to the AIV system more enjoyable due to the involvement of a musician in the music's creation. Since the AIV design values human imagination in music, it would be better suitable for story-based or linear games while the MCG design would fit better with unending game genres like high-score-based games since the continuity in music is better expressed with the algorithm.

Experiment 2: Chaosman Overloaded

Chaosman Overloaded (Cheezesoft 2022) is a platformer shooter game in which the player must race through a dungeon filled with dangerous enemies and traps in order to reach a door that proceeds to the next level. To navigate through the dungeon, the player has the options to jump, run and shoot at the enemies. The most important aspect of this game is the chaos meter because if the meter fills, the player will lose a life. If the player touches an enemy, the meter will fill up, and the meter also naturally increases as time goes by. Luckily, there are a number of towers spread out in each level that will decrease the player's chaos meter if they stand near it. The player has 3 lives, and there are a total of 3 levels in this game. If the player loses all 3 lives, they will have to start over from the first level.

In the original game, there are unique features, e.g. constantly changing colored-background and a very small bubble of vision. However, these features are either irrelevant or unfavorable to our experiment, so we had to do some modifications for the game to suit the scope of our project better.

Experimental Design

For our second experiment, we chose a game in the platformer genre to contrast with our first experiment, a sports video game. We did so to test our model on games with different types of variables to see how it would perform. With a platformer type of game, not only do we have variables that can indicate winning and intensity level similar to the previous games such as the score or the numbers of certain game elements on screen, but we also have to take into account hard-to-define variables, e.g. distance to the goal, the number of enemies, their average position, etc. In Chaosman, the variables that we sampled at 100 ms intervals were:

- Score (integer, range $[0, 706]$)
- Number of lives (integer, range $[0, 3]$)
- Chaos value (float, range $[0.0, 100.0]$)
- Current level (integer, range $[0, 2]$)
- Number of enemies within 80 pixel radius (integer, range $[0, 7]$)
- Number of bullets on screen (integer, range $[0, 10]$)
- Number of player input events per 0.1 seconds averaged over the last 1 second (float, range $[0.0, 13.9]$)
- Distance from goal (pixels, range $[7.0, 867.31]$)
- Distance from start (pixels, range $[0, 1214.05]$)
- Elapsed time (ms, range $[14.0, 37610.0]$)
- Position (whether at tower, $0(\text{false}), 1(\text{true})$)

We collected over 209,000 merged input-output rows of data from Chaosman, taken from over 4 hours of testing. We collected more data from Chaosman than Bouncy Shots because Chaosman is a larger game, though it can still be completed in a couple minutes.

Again, our emotional outputs consisted of Intensity and Winning Status values. The data collection process was the same; only gameplay state features differed. After collection, we again normalized the data and created the ML models of our previous experiment to look for possible differences in performance.

Experimental Results

We considered the same regression models as with the Bouncy Shots game, but for Chaosman our baseline values used number of bullets for intensity prediction and distance from goal for winning status prediction. Random forest regression was also the best performing model (Table 2) with RMSE values minimal for both intensity and winning.

Random forest regression was also used for lag and polynomial degree optimization. For the lag test, the lowest RMSE value was for 500 ms for the intensity output, but

Degree	1	2	3	4
RMSE (Intensity)	0.0969	0.0979	0.0982	0.0982
RMSE (Winning)	0.0753	0.0756	0.0749	0.0754

Table 3: Chaosman Degree Statistics

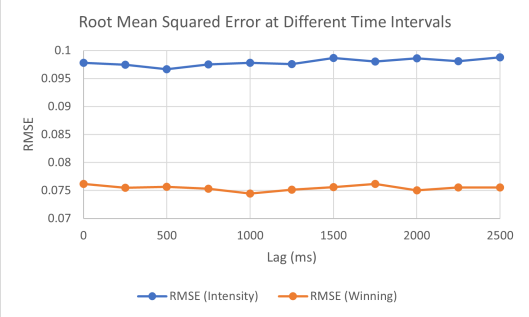

Figure 9: Lagged RMSE values of Chaosman.

for 1000 ms for the winning output (Figure 9). In this situation, we chose to use 500 ms of lag, because we can see that RMSE measures for intensity are greater than those of winning status, so we chose the value where the less consistent RMSE was best. We observe that this synchronization value is different from the best value that was concluded in experiment 1. Differences in synchronization shift are to be expected because of significant differences in game genre and time needed for human state evaluation. Bouncy shots as a fast "twitch" game is relatively simple, allowing rapid assessment. Chaosman as a platformer is more complex with many features relevant to evaluation and requires more time for human assessment of game state.

For the polynomial degree question, we can see that degree 1 has the lowest RMSE value for intensity score, but 3 has the lowest RMSE for winning status (Figure 10). To resolve this, we again note that the RMSE of intensity is greater on the whole than that of winning status, so we again chose to minimize the greater RMSE value and move forward with a first-degree polynomial. Therefore, for games similar to Chaosman Overloaded, we expect that a good starting place would be to model a first-degree polynomial with a lag of 500 ms.

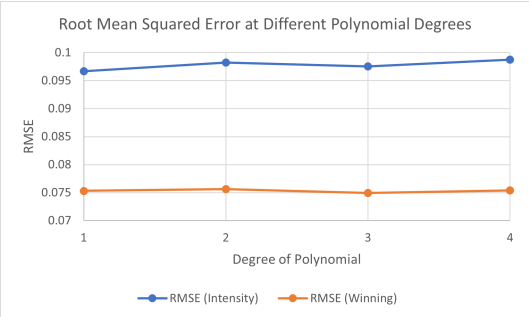

Figure 10: RMSE values of Chaosman for different degrees.

Future Work

There are a few primary ways that we would like to extend this work. First, we could minimize state variables for soundtrack loop efficiency. In these experiments, we chose state variables that we strongly suspected were relevant, but we would like this process to be general and lightweight. Thus, if the work pipeline allowed for many state features where an iterative process might minimize the number of features, we could semi-automate the ML side of the process. Iteratively applying regularization (e.g. lasso) or variable selection techniques could provide feedback to the output gameplay feature stage so as to craft a more efficient soundtrack loop.

Another extension would be to explore the effectiveness of our model in different types of games. In this paper we have tested 2D video sports (Bouncy Shots) and 2D platformer (Chaosman) genres. Looking at other genres of game (Horror, Offline FPS, RPG, Fighting, etc.) would provide greater insight as to the best models and common state variables that have greater impact on emotional states. The instrumentation for different genres also differs: an intense scene in a horror game should be scored differently than an intense scene in a shooter. Exploration of more genres could suggest extensions or generalizations for our process.

Thirdly, this paper practices a harmony-centered approach to create dynamic, adaptive music. However, we have not tapped the greater potentials of percussion. For example, increasing or decreasing the tempo with respect to the intensity level as well as varying percussion sections could have a massive impact on the intensive feel of the soundtrack. Likewise, approaches introduced here could be implemented in conjunction with a melody-centered sound generation algorithm, which could potentially provide samples with greater musicality, i.e. have more recognizable musical structure and melody.

One limitation of our Markovian music generation is our assumption that all the notes played by the algorithm are hierarchically equal. This would disappoint most musicians, who are generally tonal and deliberate in the selection of note patterns in musical context.

Conclusions

In this work, we have demonstrated methods for creating an adaptive game soundtrack using machine learning (ML) techniques to learn AI for emotional conducting of music. Through our Player-Conductor ML approach, we were able to collect data on a broad spectrum of game states in order to create a more robust AI Conductor model. We then merged timestamped data from the Player's game state inputs and the Conductor's emotional outputs. Testing multiple supervised learning models on this data, we found that random forest regression provided the lowest RMSE. Possible sources of error were noted and tested, such as a potential gap between in-game events happening and the Conductor's reaction in recording their emotional state and the possibility of higher-degree polynomials being of better quality, and we concluded that short lag times (250-500 ms) and a first-degree polynomial yielded lowest RSMEs.

References

Casella, P.; and Paiva, A. 2001. Magenta: An architecture for real time automatic composition of background music. In *International Workshop on Intelligent Virtual Agents*, 224–232. Springer.

Cheezesoft. 2022. Chaosman Overloaded. https://cheezye.itch.io/chaosman-overloaded. Accessed: 2022-07-1.

Collins, K. 2008. Compositional Approaches to Dynamic Game Music. In *Game Sound: An Introduction to the History, Theory, and Practice of Video Game Music and Sound Design*, chapter 8. The MIT Press. ISBN 9780262270694.

DaFluffyPotato. 2022. Bouncy Shots. https://dafluffypotato.itch.io/bouncy-shots. Accessed: 2022-08-29.

Escarce Junior, M.; Rossmann Martins, G.; Soriano Marcolino, L.; and Tavares dos Passos, Y. 2021. Emerging Sounds Through Implicit Cooperation: A Novel Model for Dynamic Music Generation. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 13(1): 186–192.

Hindemith, P. 1945. *The Craft of Musical Composition: Book 1: Theory*. Associated Music Publishers, Inc. ISBN 0901938300, 9780901938305.

Léo, S. 2020. *The Effect of Dynamic Music in Video Games: An Overview of Current Research*. Bachelor's thesis, Uppsala University.

Plut, C.; and Pasquier, P. 2020. Generative music in video games: State of the art, challenges, and prospects. *Entertainment Computing*, 33: 100337.

Rayman, J. 2014. *Experimental Approaches to the Composition of Interactive Video Game Music*. Ph.D. thesis, University of East Anglia.

Redhead, T. 2018. The Emerging Role of the Dynamic Music Producer. In *Australasian Computer Music Conference 2018: Reflecting Worlds: The Promise and Limitations of Mimesis in Electronic Music*. Edith Cowan University.

scikit-learn.org. 2022. sklearn.preprocessing.MinMaxScaler. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html. Accessed: 2022-08-29.

Yilmaz, V. G. 2022. VideoGameMusicGenerator. https://github.com/GoyaReceli/VideoGameMusicGenerator. Accessed: 2022-12-12.