GenerationMania: Learning to Semantically Choreograph

Zhiyu Lin, Kyle Xiao and **Mark Riedl**
Georgia Institute of Technology
Atlanta, Georgia, United States
{zhiyulin,kylepxiao}@gatech.edu, riedl@cc.gatech.edu

Abstract

Beatmania is a rhythm action game where players play the role of a DJ that performs music by pressing specific controller buttons to mix "Keysounds" (audio samples) at the correct time, unlike other rhythm action games such as Dance Dance Revolution. It has an active amateur Chart (Game stage) creation community, though chart authoring is considered a difficult and time-consuming task. We present a deep neural network based process for automatically generating Beatmania charts for arbitrary pieces of music. Given a raw audio track of a song, we identify notes according to instrument, and use a neural network to classify each note as playable or non-playable. The final chart is produced by mapping playable notes to controls. We achieve an F1-score on the core task of *Sample Selection* that significantly beats LSTM baselines.

Introduction

Rhythm action games such as Dance Dance Revolution, Guitar Hero, and Beatmania challenge players to press keys or make dance moves in response to audio playback. The set of actions timed to the music is a *chart* and is presented to the player as the music plays. Charts are typically handcrafted, which limits the songs available to those that have accompanying charts. *Learning to choreograph* (Donahue, Lipton, and McAuley 2017) is the problem of automatically generating a chart to accompany an *a priori* unknown piece of music.

Beatmania IIDX (BMIIDX) is a rhythm action game, similar to Dance Dance Revolution, with an active community of homebrew chart choreographers (Chan 2004). Unlike Dance Dance Revolution, players play the role of a DJ and must recreate a song by mixing audio samples by pressing controller buttons as directed by on-screen charts. In BMIIDX, some notes from some instruments are played automatically to create a complete audio experience. That is there are "playable" and "non-playable" notes in each song. A **playable object** is defined as that which appears visually on the chart and is available for players to perform, with a one-to-one correspondence to an audio sample; on the other hand, a **non-playable object** is one that is automatically played as part of the background music. In order to get a high score as well as reconstruct the original music, a player needs to press the correct button at the correct time for playable objects, as well as not press any button when not being instructed. The controller used in this game series is also unique: It features both 7 buttons and a "turntable" control which the player scratches instead of presses.

A fundamental difference between BMIIDX and many other rhythm action games like DDR is that BMIIDX is considered a game with *keysounds*, which means every object in the chart has an audio sample counterpart which plays if and only if the corresponding action is executed. This even includes non-playable objects; their actions are automatically executed. For comparison, Guitar Hero (Miller 2009) is another keysound based rhythm action game where each note in the game represents a guitar maneuver. In BMIIDX, however, each note can represent an audio sample from different instruments.

These differences in the underlying mechanics yields a unique paradigm for creating BMIIDX charts. Requiring a clear binding between objects and instrument placement based on an underlying score means BMIIDX charts cannot be *overmapped*. Overmapping, which happens frequently in DDR, describes the situation where patterns of actions are unrelated to instruments being played or occur when no note is being played by any instrument at that moment. That is, the creation of BMIIDX charts is *strictly* constrained by the semantic information provided by the underlying music. We refer to this challenge as "Learning to *semantically* choreograph" (LtSC).

Due to the strict relationship between chart and music—as well as other differences such as charts with several simultaneously actions—the prior approach used to choreograph charts for Dance Dance Revolution (Donahue, Lipton, and McAuley 2017) cannot be used to generate BMIIDX charts.

We approach the challenge of learning to *semantically* choreograph charts for BMIIDX as a four-part process. (1) we train a neural network to identify the instruments used in audio sample files and the timing of each note played by each instrument. (2) we automatically label the difficulty of charts in our training set, which we find improves chart generation accuracy. (3) We train a supervised neural network that translates a musical context into actions for each time slice in the chart. Unlike Dance Dance Convolution (Donahue, Lipton, and McAuley 2017), which used an LSTM,

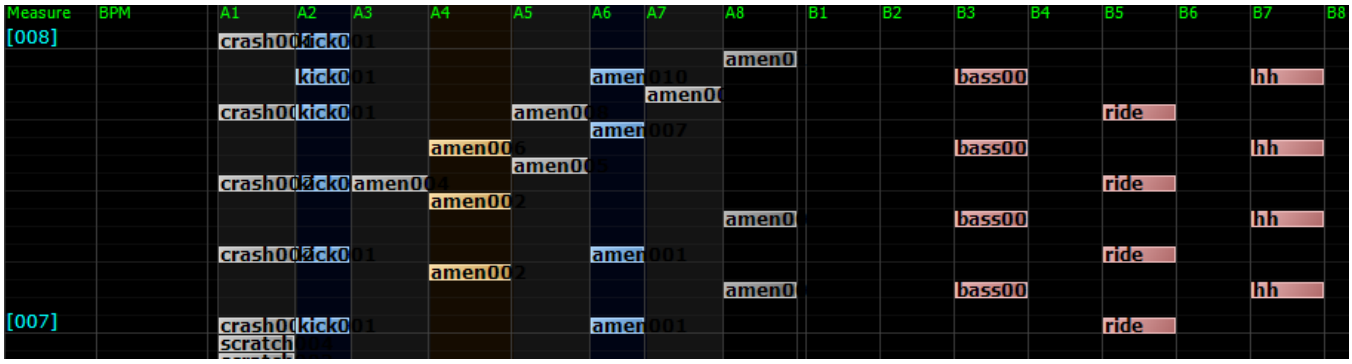

Figure 1: A visualization of a Beatmania IIDX homebrew chart, *Poppin' Shower*. Notes in this screen shot are labeled with their author-created filenames. Note that only objects in the columns starting with A are **playable objects** that is actually visible to players; Others are **Non-playable objects** used as background.

we find a feed forward model works well for learning to semantically choreograph when provided a context containing the instrument class of each sample, intended difficulty label of each sample, the beat alignment, and a summary of prior instrument-to-action mappings.[1] (4) Notes predicted to be playable by the network are mapped to controls and the final chart is constructed. In addition, we introduce the BOF2011 dataset for Beatmania IIDX chart generation.

Background and Related Work

Procedural Content Generation (PCG) is defined as "the creation of game content through algorithmic means". Machine learning approaches treat content generation as (a) learning a generative model and (b) sampling from the model during creation time (Summerville et al. 2017; Guzdial and Riedl 2016; Summerville and Mateas 2015; Hoover, Togelius, and Yannakis 2015; Summerville and Mateas 2016).

Beatmania IIDX

The homebrew community of BMIIDX is arguably one of the oldest and most mature one of its kind (Chan 2004), with multiple emulators, an open format (Be-music Source[2], BMS) and peer-reviewed charts published in semi-yearly proceedings. Despite the community striving to provide the highest quality charts, the process of generating such a chart is considered a heavy workload, and due to the strict semantic bindings, usually the author of the music or a veteran chart author has to participate in the generation of the chart. Many aspiring amateur content creators start by building charts for rhythm action games without keysounds (i.e. Dance Dance Revolution charting is considered by the community to be easier). Furthermore, there is a strong demand for customized charts: players have different skill levels and different expectations on the music-chart translation, and such charts are not always available to them.

[1] A fixed window feed-forward network can often outperform a recurrent network when short-term dependencies have more impact than long-term ones (Miller and Hardt 2018).

[2] `https://en.wikipedia.org/wiki/Be-Music_Source`

Figure 1 shows an example of a BMIIDX homebrew chart. The objects in the "A" columns are playable objects with keysounds.

Rhythm Action Game Chart Choreography

There is a handful of research efforts in chart choreography for rhythm action games, including rule-based generation (OKeeffe 2003; Smith et al. 2009) and genetic algorithms using hand-crafted fitness functions (Nogaj 2005). Dance Dance Convolutions is the first deep neural network based approach to generate DDR charts (Donahue, Lipton, and McAuley 2017). Donahue et al. refer to the problem of learning chart elements from data as *Learning To Choreograph*. Alemi et al. (2017) suggest that this approach can reach real-time performance if properly tuned.

Dance Dance Convolutions uses a two-stage approach. *Onset detection* is a signal analysis process to determine the salient points in an audio sample (drum beats, melody notes, etc.) where steps should be inserted into a chart. *Step selection* uses a long-short term memory neural network (Hochreiter and Schmidhuber 1997) to learn to map onsets to specific chart elements. BMIIDX chart generation differs from DDR in that the primary challenge is determining whether each note for each instrument should be playable as a stage object or non-playable (i.e., automatically played for the player).

Data

We compiled a dataset of songs and charts from the "BMS Of Fighters 2011" community driven chart creation initiative. During this initiative, authors created original music and charts from scratch. The dataset thus contains a wide variety of music and charts and was composed by various groups of people. Although the author is not required to create a defined spread of different charts for a single piece of music for the event, authors frequently build 3 to 4 charts for each song. The dataset, which we refer to as "BOF2011", consists of 1,454 charts for 366 songs. Out of 4.3M total objects, 28.7%, or 1.24M of them are playable ones. Table 1 summarizes the dataset.

Table 1: BOF2011 dataset summary.

# Songs	366
# Charts	1,454
# Charts per song	3.97
# Unique audio samples	171,808
# Playable objects	1,242,394
# Total objects	4,320,683
Playable object %	28.7

We find that modeling the difficulty of charts plays an important role in learning to semantically choreograph, an observation also made by (Donahue, Lipton, and McAuley 2017). Many of the charts in our dataset are easier ones, in which non-playable objects dominate. Furthermore, a vast majority of samples are repeatedly used, such as drum samples placed at nearly every full beat throughout a chart, resulting in only 171k unique audio samples in our dataset. The ratio of playable objects to the total objects is not the only factor that determines the difficulty of the chart. Perceived difficulty of charts can also be influenced by:

- Added group of notes representing more rhythmic elements;
- Special placement of a group of notes that requires specific techniques to play accurately;
- Strain caused by a long stream of high density patterns;
- "Visual effects", perspective changes causing suddenly accelerating/stopping notes;
- A combination of the above in a small time window.

Chart authors label their charts according to difficulty level. However, such labels are based entirely on the author's perception of the chart difficulty. For example, it is common for some expert authors' charts to be labeled as "normal" levels despite being more difficult than others' "difficult" levels. Although the original Beatmania IIDX labels used the monikers "normal", "hyper", and "another", authors can assign any label to describe the difficulty of the chart.

Methods

Our chart generation system for BeatMania IIDX, which we call *GenerationMania*, uses a pipeline consisting of the following tasks:

1. Sample Classification — Identifying the instrument used in audio samples;
2. Challenge Modeling — Establish structure of each part in the chart;
3. Sample Selection — Classifying audio samples into playable and non-playable;
4. Note Placement — Assigning controls to each playable keysound.

We realize that the task of generating a music score from raw audio (which is essentially Audio Segmentation) can be well decoupled from generating BMIIDX charts from a music score. Based on the assumption that the music score is available, we focus on *Sample Classification*, *Challenge Modeling* and *Sample Selection* which is unique to BMIIDX stage generation.

Sample Classification

Sample classification is a process by which notes from different instruments in the audio samples are identified. The BMS file format associates audio samples with timing information. That is, a chart is a set of sample-time pair contains a pointer to the file system where an audio file for the sample resides. Unfortunately, in the BMS file format there is no standard for how audio samples are organized or labeled. However, many authors do name their audio sample files according to common instrument names (e.g., "drums.ogg"). The goal of sample classification is to label each sample according to the instrument based on its waveform. The predicted labels will be used to create one-hot encodings for each sample for the sample selection stage on the pipeline.

We construct a training set by gathering audio samples together with similar instrument names according to a dictionary and use the most general instrument name as the supervision label. We use the 27 most common categories for labeling. To ensure that we don't overfit our classifier we train on an alternate dataset, "BMS of Fighters Ultimate" (BOFU) that does not share any music or charts with BOF2011, with a partially labeled dataset having a total of 60,714 labeled samples. Not every audio sample have a classifiable name, which we count as unlabeled samples.

We decompose the audio samples to their "audio fingerprints," which consists of a vectorized spectrogram representation of the audio using the normalized wave amplitudes over time. We also fix the bit rate of the sound to 16k, so that the representation has a consistent temporal resolution.

For our model, we followed the method described in (Sainath and Parada 2015). We feed the fingerprints through two 2D convolutional layers, each with a bias. Each of the layers is followed by a Rectified Linear Unit (ReLU) activation function with the next abstracted layer generated via max-pooling. Finally, we feed the results into another fully connected layer, which then outputs a one-hot encoding of the predicted category. We use a gradient descent optimizer with 50% dropout rate and a step size of 0.01.

After training on the BOFU dataset, we achieved an 84% accuracy based on a 10% testing set.

Challenge Modeling

We compute the difficulty level of each object in each chart in the training set. We use a rule-based technique for assessing the difficulty of each object in a chart. This technique is adapted from the *Osu!* rhythm action game.[3] The difficulty for each object of a given chart is weighted sum of the *individual strain* and the *overall strain*. Individual strain calculates as the interval between keysounds mapped to the same control on an exponetiated scale such that short intervals have exponetially higher strain values than long inter-

[3] https://github.com/ppy/osu/blob/master/osu.Game.Rulesets.Mania/Difficulty/ManiaDifficultyCalculator.cs

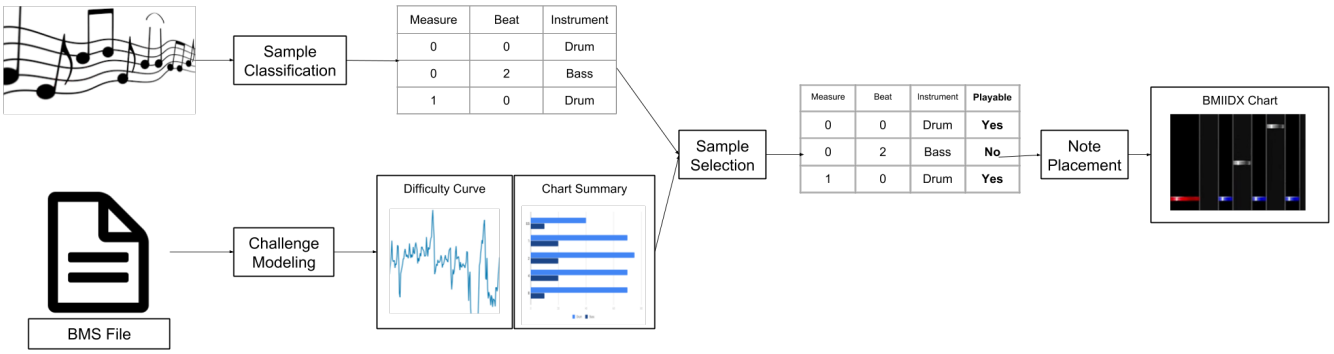

Figure 2: The GenerationMania pipeline.

vals. Overall strain calculates as the number of controls that must be activated simultaneously. In addition, challenging patterns have prolonged effects on both strain values for objects directly after them. For each 0.4 second window, the maximum strain value becomes the difficulty of that window and every object receives that difficulty. An overall difficulty of the chart is generated by weighted sum of highest local maximum strain values throughout such chart.

Sample Selection

Sample selection is a task of determining which objects in the music should be playable and which should be non-playable. The input features for each object are as follows:

- Difficulty: A 1×1 value of difficulty from the difficulty curve;
- Audio Features: A 1×27 one-hot representation of the instrument class that the audio sample belongs to;
- Beat Alignment: A 1×1 value ranging from 0 to 15 representing which 16th section out of a beat this note resides in, with 0 representing the note on a beat. This in most occasions represents a per-64th-note granularity in a chart having 4/4 time signature, which is around 25 milliseconds on a chart at 150 BPM (Beat Per Minute);
- Summary: A 1×270 vector summarizes the playability of different samples prior to the current object. For each instrument class, a 1×2 vector gives the probability that that instrument was playable or non-playable in a given window of time, as computed by the number of times it was playable/non-playable divided by the number of appearances. This gives a 1×54 vector for a time window. Five different time windows are provided covering 2, 4, 8, 16, and 32 beats.

Summarization is a technique popularized by *WaveNet* (Oord et al. 2016) to factor prior information from a fixed window at different time scales into a current prediction. At training time and inference time, the summary information is derived from the training data, except for the self-summary baseline, for which summary information is based on previous generation results.

Our sample prediction model is a feed-forward network consisting of 4 fully connected ReLU layers of dimensions 64, 32, 16, and 2. To perform sample selection, we pick the output node with the highest activation corresponding to playable or non-playable. Due to the class imbalance that most of the objects are non-playable we found that a weighted Mean Squared Error loss function helps improving the performance of the training.

At training time the difficulty curve is derived from the dataset so that that sample selection network can be trained to reconstruct the input data. At generation time, the difficulty curve can be provided by the user.

Note Placement

For each object at each timestep that has been classified as playable, we map it to one of 8 controls. Any process that doesn't map objects to the same control at the same time is sufficient to make a chart playable, thus note placement is not a significant contribution in this paper. We created a simple module that uses the same framework as *Sample Selection* but trained to predict note placements as labels instead of playability. A post-processing step checks and rearranges the chart so that we never map two objects that occur in too short interval to the same control.

Experiments

In this section, we evaluate variations of our sample selection model against a number of baselines. We used a supervised evaluation metric: by embedding challenge model extracted from the ground truth, we measure how similar the generated chart is compared to the original. We establish two guidelines for a good generation model: it should not only predict *playables* when they should be presented to players (high recall), but also *nonplayables* when they should be in the background (high precision).

We applied 80%,10%,10% split on training, validation and testing data. Since the charts for the same music shares similar traits, we ensured that such charts are not both in the training split and the testing split. We trained all the models using the training split and reported all the results on the testing split.

We use the following baselines:

- **Random:** classifies a given object as a playable with a probability of 0.3, chosen to give the best result;

- **All Playable:** classifies all objects as a playable;
- **LSTM baseline:** a sequence to sequence model (Sutskever, Vinyals, and Le 2014) with forget gates and a hidden and output layer size of 2. The highest activated output is selected as the prediction.

The LSTM baseline was chosen because Dance Dance Convolution (Donahue, Lipton, and McAuley 2017) used an LSTM network. However, it is impossible to directly compare the approach used in DDC to our feed forward model because the task and the inputs between Dance Dance Revolution (non-keysound based game) and BeatMania IIDX (keysound-based game) are different enough that substantial changes to the DDC algorithm are required.

Our feed-forward sample selection model and the LSTM baseline are configured with different combinations of input features drawn from: audio features (instrument labels), difficulty curve, beat alignment, and summary vectors. We also experiment with the use of summary vectors; we refer to the models without summary inputs as "free generation". There is one special case of free generation model in which we allow the model to self-summarize. That is, we use summary data based on what has been generated earlier in the chart.

For all neural network modules, we learn parameters by minimizing a weighted mean squared error (MSE) with weight of 1 to playables and 0.2 to non-playables. We used a mini-batch of 128 for the feed forward model; Due to the need of processing very long sequences, the LSTM model is trained by each sequence and is run in CPU mode. We stop training when the loss converges. The feed forward model satisfies this criteria in around 6 hours in GPU mode while the LSTM model takes far longer at around 100 hours, on a single machine using Intel i7-5820K CPU and NVIDIA GeForce 1080 GPU.

Results

Following the two guidelines we pointed out for evaluating the models, we report the performance of these *Sample Selection* models using precision, recall, and F1-score — a harmonic mean of them — since they are both very important. We calculate each metric per chart, then report the average of these values.

The results are shown in Table 2. Without summaries, the LSTM baseline performs the best, with a precision approaching that of models provided with summary data. LSTMs make use of history while free generation feed-forward models can only make decisions based on the current time step. With summary embedded, all the models receive a significant boost in their performance. Notably, the feed-forward models (ours) with summary data has a higher recall, which means it produces much fewer false negatives. The LSTM baseline also improved with summary data but is hampered by low recall. Although LSTM are usually used with generating long sequence of data, we believe it is because lack of data and high variance in each sequence causing it to perform worse in our task.

The information contained in the summary plays a role in the performance. We tried several different ways to create summaries which gave us different levels of performance

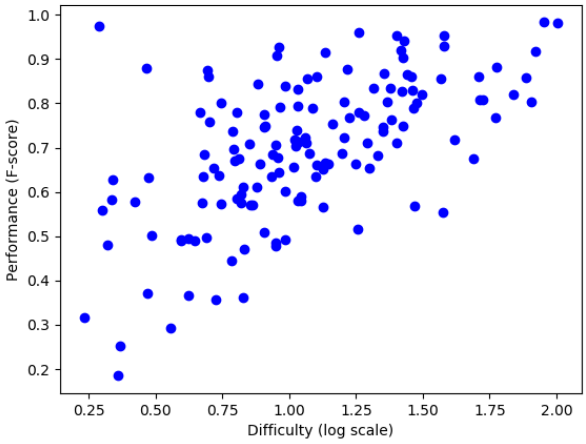

Figure 3: The performance of feed forward model (with summary) regarding difficulty of the ground truth chart.

boosts. The summary representation presented provided the best boost to both the feed forward model and the LSTM. Additionally, we considered an auto-encoder structure for LSTM model, which tries to auto-summarize the chart. We also considered multi-layer LSTM structures like in (Donahue, Lipton, and McAuley 2017). However, these models either overfit quite quickly or have unrealistic computational requirements.

In our feed-forward model with summary, per-object difficulty information accounts for a 7.7% improvement in the F1-score. As with (Donahue, Lipton, and McAuley 2017), we also observe that all generators varied in performance on charts with different difficulty. We analyzed the effect of chart difficulty on our best performing model, the feed forward model with summary. We sorted all charts in the testing set by their difficulty then examined single-chart performance. The result is summarized it in figure 3. We observed a larger variance of performance in easier charts, and a stable performance on harder charts.

Discussion and Future Work

A side effect of how beat-phase information is organized in our specific task is that we were unable to include Δ-beat information in our models. Δ-beat is a feature that measures the number of beats since the previous and until the next step. They were used in DDC (Donahue, Lipton, and McAuley 2017). However, a naive approach of "finding the next note" will not work for our task. This is mainly because (1) several semantically unrelated notes can be placed at the exact same time and (2) notes can be placed in very short intervals (such as when representing a glissando). These issues prevent effective Δ-beat detection in granularity of single note. Perhaps grouping notes based on their musical semantic relations can be a solution to this.

Our Challenge Model technique is relatively simplistic and there is room for expansion. The key assumption of this model is that for a given arrangement of objects, every player perceives exactly the same level of challenge. How-

Table 2: Results for playable classification experiments, presented in mean and standard deviation.

Model	F1-score	Precision	Recall
Reference Baselines			
Random	0.291 ± 0.089	0.335 ± 0.200	0.299 ± 0.020
All Playable	0.472 ± 0.207	0.335 ± 0.199	1.000 ± 0.000
Free Generation Models			
FFAudio Features + Difficulty Curve + Beats	0.253 ± 0.143	0.523 ± 0.266	0.179 ± 0.113
FF Audio Features + Difficulty Curve + Beats + Self Summary	0.368 ± 0.198	0.422 ± 0.213	0.392 ± 0.258
LSTM + Audio Features + Difficulty Curve + Beats	0.424 ± 0.154	0.767 ± 0.176	0.353 ± 0.248
Generation with Summary			
FF Audio Features + Beats + Summary	0.621 ± 0.206	0.760 ± 0.110	0.568 ± 0.254
FF Audio Features + Difficulty Curve + Beats + Summary	$\mathbf{0.698 \pm 0.162}$	0.778 ± 0.112	$\mathbf{0.649 \pm 0.197}$
LSTM + Audio Features + Difficulty Curve + Beats + Summary	0.499 ± 0.225	$\mathbf{0.805 \pm 0.121}$	0.405 ± 0.237

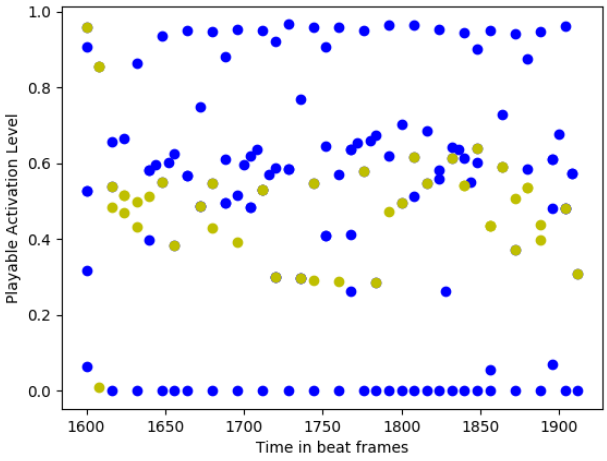

Figure 4: Around 6 seconds of playable classification result compared to ground truth human Chart for *Poppin' Shower*. An activation level higher than 0.5 is considered classification of *playable*. Blue dots identifies correct predictions, yellow dots identifies incorrect ones. On this song, we achieved 0.824 F-Score with our feed forward model.

ever, it is possible that players have different playing level, and they have individual differences. This causes problem on evaluating challenge level of asymmetric and/or hand-biased patterns since every control is treated exactly the same. A derivation of this assumption is that "easy" charts should be treated the same as harder charts, which proves to be particularly problematic and may be a cause of poor generation performance on "easy" charts. We observed that unlike harder charts, many "easy" charts are designed for newcomers to the game, which in turn have reduced challenging artifacts and focused notes representing only the melody of the music. This results in drastically different charting style, which may explain why our Sample Selection classifier have poorer performance on them. Because the Challenge Model was hand-authored using a particular dataset (*Osu!* stages), its performance on a different dataset may deteriorate. The Challenge Model is also sensitive to parameter tuning. A model-free approach or a player experience based system may help in this scenario.

Aside from that, the Challenge Model and summary can be extracted from charts provided by players to allow for a degree of controllability of the system. Our feed forward model even allows generation on-the-fly. This make it possiblefor our pipeline to be used in tasks such as dynamic challenge adaptation, where challenge level of the stage changes based on player's performance and preference (Zook and Riedl 2015) and style transfer, where two charts blend with each other(Johnson, Alahi, and Fei-Fei 2016). Furthermore, a Challenge Model that is human-understandable allows player to easily manipulate it to their will, which in turn may facilitate human participation in this process, allowing Computational Co-creativity applications which would be especially helpful to content creators. We don't know if our system meets the player's expectations yet; We leave all of these as future work.

Conclusions

Choreographing Rhythm Action Game stages is a challenging task. BMIIDX added more challenge on top of it by posing extra semantic constraints by requiring one-to-one audio-sample-to-playable-object relation. We have established a pipeline for *Learning to Semantically Choreograph*, provided a dataset for reproducible evaluations, and showed that a feed forward neural network model with challenge modeling and summary information performs well on satisfying these new constraints. We further discuss how users can inject a degree of control over the algorithm by inputting a customized or manually edited difficulty curve and biasing the summary information.

Learning to semantically choreograph is essential to generating keysound based game charts. However, incorporating semantics may potentially also be used to improve generation on non-keysound based games such as Dance Dance Revolution, where it is possible to overmap actions and still achieve high accuracy according to automated metrics. Aside from solving a challenging creative task, intelligent systems such as GenerationMania can be of benefit to homebrew chart choreography communities by overcoming skill limitations. The ability to control the generative process is an essential part of the adoption of such systems.

References

Alemi, O.; Françoise, J.; and Pasquier, P. 2017. GrooveNet: Real-Time Music-Driven Dance Movement Generation using Artificial Neural Networks. *networks* 8(17):26.

Chan, A. 2004. CPR for the Arcade Culture.

Donahue, C.; Lipton, Z. C.; and McAuley, J. 2017. Dance Dance Convolution. In *Proceedings of the 34th International Conference on Machine Learning*.

Guzdial, M., and Riedl, M. 2016. Game level generation from gameplay videos. In *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*.

Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.

Hoover, A. K.; Togelius, J.; and Yannakis, G. N. 2015. Composing video game levels with music metaphors through functional scaffolding. In *First Computational Creativity and Games Workshop. ACC*.

Johnson, J.; Alahi, A.; and Fei-Fei, L. 2016. Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision*, 694–711. Springer.

Miller, J., and Hardt, M. 2018. When Recurrent Models Don't Need To Be Recurrent. *arXiv preprint arXiv:1805.10369*.

Miller, K. 2009. Schizophonic Performance: Guitar Hero, Rock Band, and Virtual Virtuosity. *Journal of the Society for American Music* 3(4):395429.

Nogaj, A. F. 2005. A genetic algorithm for determining optimal step patterns in Dance Dance Revolution.

Oord, A. v. d.; Dieleman, S.; Zen, H.; Simonyan, K.; Vinyals, O.; Graves, A.; Kalchbrenner, N.; Senior, A.; and Kavukcuoglu, K. 2016. WaveNet: A Generative Model for Raw Audio. In *SSW*, 125.

OKeeffe, K. 2003. Dancing monkeys. *Masters project* 1–66.

Sainath, T. N., and Parada, C. 2015. Convolutional neural networks for small-footprint keyword spotting. In *Sixteenth Annual Conference of the International Speech Communication Association*.

Smith, G.; Treanor, M.; Whitehead, J.; and Mateas, M. 2009. Rhythm-based Level Generation for 2D Platformers. In *Proceedings of the 4th International Conference on Foundations of Digital Games*, FDG '09, 175–182. New York, NY, USA: ACM.

Summerville, A., and Mateas, M. 2015. Sampling Hyrule: Sampling Probabilistic Machine Learning for Level Generation. In *Conference on Artificial Intelligence and Interactive Digital Entertainment*.

Summerville, A., and Mateas, M. 2016. Super mario as a string: Platformer level generation via lstms. *arXiv preprint arXiv:1603.00930*.

Summerville, A.; Snodgrass, S.; Guzdial, M.; Holmgård, C.; Hoover, A. K.; Isaksen, A.; Nealen, A.; and Togelius, J. 2017. Procedural Content Generation via Machine Learning {(PCGML)}. *CoRR* abs/1702.0.

Sutskever, I.; Vinyals, O.; and Le, Q. V. 2014. Sequence to Sequence Learning with Neural Networks. In Ghahramani, Z.; Welling, M.; Cortes, C.; Lawrence, N. D.; and Weinberger, K. Q., eds., *Advances in Neural Information Processing Systems 27*. Curran Associates, Inc. 3104–3112.

Zook, A., and Riedl, M. O. 2015. Temporal game challenge tailoring. *IEEE Transactions on Computational Intelligence and AI in Games* 7(4):336–346.