Proposal for RAP: Vertical Scrolling Rhythm Game (VSRG) Difficulty Estimation with DNNs

John Chang, Nanyang Technological University

May 16, 2022

Chapter 1

Introduction

1.1 Abstract

Rhythm Game is a popular genre that utilizes music to challenge the players' sense of rhythm and visual pattern recognition. Vertical Scrolling Rhythm Game (VSRG) , the most popular sub-genre, sees exponential growth in popularity in current years. However, it faces a problem in predicting the difficulty of play due to its extremely complex nature. In this proposal, I show how I plan to estimate this difficulty using DNNs and the possible use-cases.

Take for example in Figure 1.4, rapidly alternating the index and middle finger (2, 3) is simpler than alternating the ring and pinky (4, 5). Imagine a note sequence with the same density, pressing (4, 5) in repeat will be immensely more difficult than (2, 3).

1.1.1 Difficulty Estimation Proposal

With advancing technologies in Machine Learning, I propose an alternative way using Deep Neural Networks to estimate difficulty. We firstly encode note patterns as the input, and utilize online VSRG player performance databases "Etterna", n.d.; "osu!", n.d.; "Quaver", n.d. to train on.

Finally, we will evaluate through prediction loss and deploy an online satisfaction survey on the performance in various VSRG communities.

In the next section, I proceed to outline key milestones in the project, detail procedures and ultimately justifying steps taken.

1.2 Background of VSRGs

A Vertical Scrolling Rhythm Game (VSRG) is simply a rhythm game where notes travel from top-to-bottom or bottom-to-top to hit a receptor. The timing when it hits a receptor is usually related to an instrument in the music, thus the player mimics the play of the actual instrument. There are several ways to input, either using a keyboard, a pressure-sensitive device, a custom controller, and many more. To limit our scope, we will only consider keyboard inputs, one of the most popular way to play this genre.

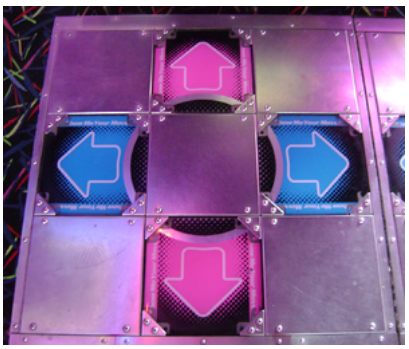


Figure 1.1: Pressure Sensitive Dance Dance Revolution device to be played with feet.

Figure 1.2: Beatmania IIDX Controller to be played with hands.

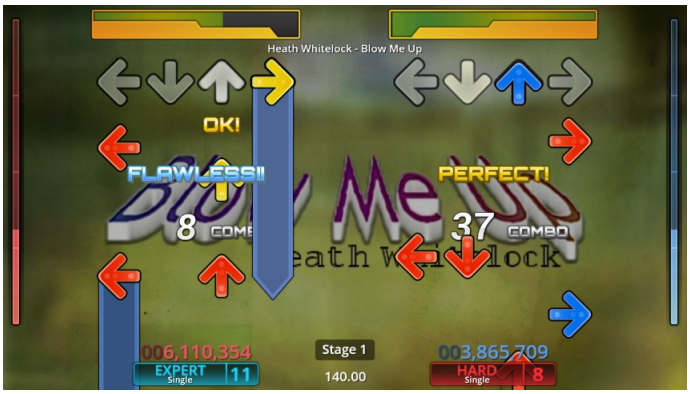


Figure 1.3: Image of StepMania 5.0.5. Shows gray arrow receptors on top, and red, yellow, blue notes travelling upwards towards the receptor.

To illustrate the gameplay, I show StepMania in Figure 1.3, one of the more popular old VSRGs. We can observe the notes in red, blue, yellow, indicative of their rhythm, travelling upwards to reach the receptor in gray. Upon reaching it, the player is prompted to input correspondingly, where they will be judged on their rhythmic precision.

One of the major pitfalls of rhythm games such as this, is difficulty estimation. Due to its complexity, creators usually resort to manual annotation. Alternatively, most popularly, difficulty is estimated by note density, for it has high correlation to difficulty. This method, however, neglects the order of the notes, note patterns, thus leading to many anomalous density-estimated maps.

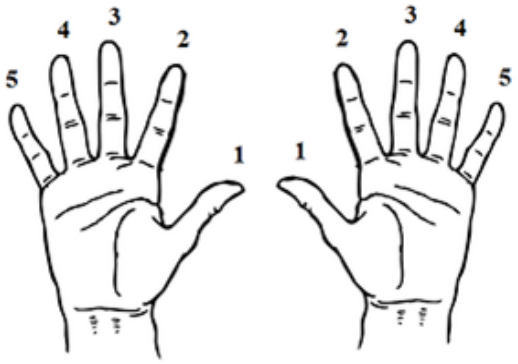


Figure 1.4: Annotated Finger positions

1.3 Project Management

A key make or break requirement in a large project is its project management.

> Measure thrice, check twice, cut once

We follow loosely the idea of Agile's methodology using an web-based planning tool, Jira. Jira helps plan milestones, known as Agile Epics; within milestones are user stories, use-cases told in a story.

In this project, we have 10 separate epics to be fulfilled.

1. Cloud
2. reamberPy Preparation
3. Map Embedding
4. Score Ground Truth
5. Replay Ground Truth
6. Streamlit Demo
7. Research Report
8. Neural Net Model
9. Web API
10. Scalability

We discuss each in detail in the following sections. However, we briefly tangent to the roadmap

1.3.1 Roadmap

We estimate the project to span from March 2022 to March 2023, as follows.

	Start	End
Cloud	3 Mar 2022	1 Mar 2023
Score Ground Truth	11 Apr 2022	22 May 2022
reamberPy Preparation	23 May 2022	19 June 2022
Map Embedding	20 June 2022	7 Aug 2022
Replay Ground Truth	8 Aug 2022	11 Sep 2022
Neural Net Model	12 Sep 2022	11 Dec 2022
Scalability	19 Sep 2022	27 Nov 2022
Web API	26 Sep 2022	30 Oct 2022
Streamlit Demo	31 Oct 2022	11 Dec 2022
Research Report	12 Dec 2022	1 Mar 2023

Table 1.1: Table of Epics and Estimated Timeline

We thus discuss each epic and their purposes

Chapter 2

VSRG Terms

2.1 Notes, Long Notes, Receptors and Keys

Before delving into methodologies, we explain terms used in the paper. **Notes**: In Figure 1.3, we see arrows that travel upwards. These arrows, are known as notes, think musical notes.

Receptor: They all travel towards the **Receptor**, the grayed out arrows, where the player must press the respective keys when the notes line up with the receptors.

Long Notes (LN): There are 2 types of notes in Figure 1.3, one an arrow, the other an arrow with a long blue tail. The note with a tail means the player must press down for the whole duration of the body.

Keys: Observe Figure 1.3's 4 columns, this implies the number of keys is 4. Some games such as:

- "Quaver", n.d. has 4 and 7 Keys
- "osu!", n.d. has 1 to 18 Keys
- "Etterna", n.d. has only 4 Keys
- "O2Jam", n.d. has 7 Keys

By popular convention, n Keys are shortened to nK; 4 Keys are usually called 4K.

2.1.1 Standard ASCII Format

For the rest of the paper, to make it simple, we use a easy to understand representation in ASCII.

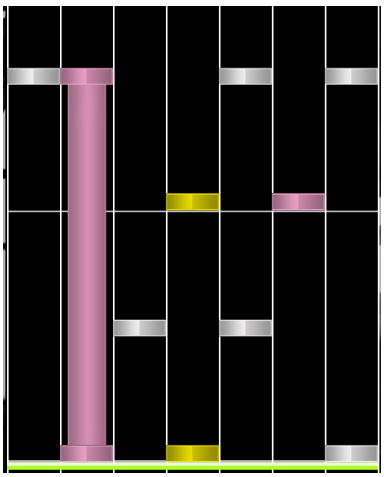
Observe the following conversion of Figure 1.3's left play field

Figure 2.1: Figure 1.3 Left Stage in ASCII. Left: Before Standardization, Right: Standardized Format.

We simplify all to `O` and make them scroll downwards [1] in Figure 2.1. Observe that they are the same pattern, just vertically flipped.

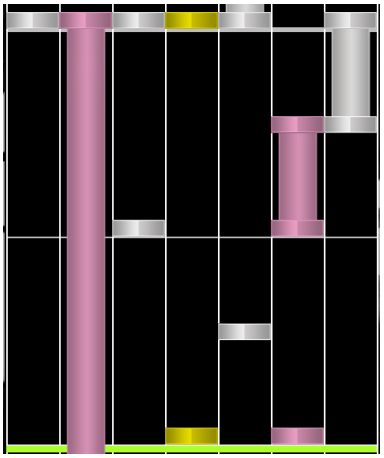
We illustrate a few more examples for your digestion.

[1] This is commonly known as down-scroll.



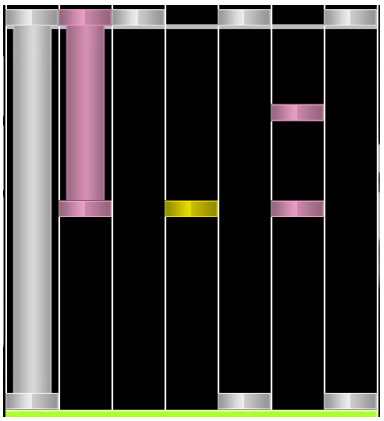
```
O |     O   O
  |   O   O
  | O   O
  O   O     O
=============
```

Figure 2.2: 7K Pattern in osu! editor



```
O | O O O   |
  |       | O
  | O     O
  |     O
  |   O   O
=============
```

Figure 2.3: 7K Pattern in osu! editor (2)



```
| | O   O   O
| |       O
| O   O   O
|
|       O   O
=============
```

Figure 2.4: 7K Pattern in osu! editor (3)

2.2 Relationship of Notes, Patterns and Beatmaps

Many notes together form patterns, many patterns form a beatmap (short: map). Each beatmap corresponds to a piece of music.

```
     ^ |         | End of Song
     | | O O   O |
     | |         |
     | |   O O   | ^
 Map | | O       | | Pattern
     | |   O O | | v
     | |       | |
     | | O O O O |
     v | ======= | Start of Song
         Receptor
```

Figure 2.5: Difference between Map and Pattern

Think of an art piece: a pattern within the art is a subset of the art piece. However, the definition is loose, the pattern may be the whole art piece.

2.3 Judgements

Judgements determine the quality of your rhythm. If you pressed the key on time, it'll give a better judgement. Judgements will dictate the overall score and rhythmic accuracy of the play.

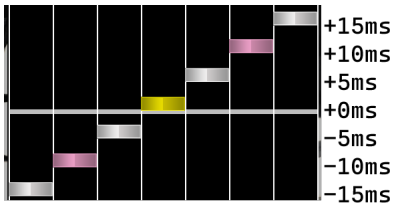


Figure 2.6: Judgements depending on how early or late the player hits.

For example, judgements in osu! are as follows for overall difficulty[2] of 8.

Window	Judgement
$\pm 16.5ms$	Best[3]
$\pm 40.5ms$	Good
$\pm 73.5ms$	Okay
$\pm 103.5ms$	Not Good
$\pm 127.5ms$	Bad
otherwise	Miss

Table 2.1: Judgement Windows for osu!

[2] overall difficulty affects strictness of these judgement windows

Chapter 3

Key Milestones & Requirements

We describe the epics in Table 1.1

3.1 Cloud

The cloud provider is the backbone of the project as we heavily depend on big data and its future scalability. We require the cloud to provide SQL support with lowest cost incurred within the year. Some candidates include **Google Cloud**, **AWS**, **Microsoft Azure** and more.

We chose Google Cloud mainly for its simplicity in converting `.sql` files to `.csv` files. [1]

3.2 Ground Truth

The ground truth is of utmost importance as with machine learning projects. One major caveat of the VSRG topic is the lack of a ground truth set, thus we create our own.

Ultimately we need a dataset that well describes the population

We can yield 3 important ground truths

1. Aggregated Map Difficulty
2. Time-Series Map Difficulty

3.2.1 Aggregated Map Difficulty

Upon completing a map, we yield a score describing their performance. This value is an aggregation of difficulty of the map. Thus, only provides an aggregated map difficulty feedback.

In order to yield difficulty of specific patterns we need real-time play analysis

Methodology

1. Yield the largest score dataset
2. Filter out anomalies
3. Predict difficulty of maps

3.2.2 Time-Series Map Difficulty

Using replays of each play, we can evaluate every action in real-time. This is better suited for pattern difficulty evaluation.

This fine-grained approach will be atomic, though will be more difficult in every aspect.

[1] The data we receive from osu! is in SQL format

Methodology

1. Yield Replays using API with `replay_id` in 3.2.1
2. Emulate Replays to find errors in plays
3. Decompose Map into Patterns
4. Link errors to patterns
5. Predict pattern difficulty

Chapter 4

Score Ground Truths

4.1 Score Prediction Methodology

If we are able to yield every players' score from every map, we can predict how difficult every map is.

E.g. If a map has a low median score, we know that it's harder, vice versa.

4.1.1 Null Scores

However, there isn't a score from every player on every map, i.e. not all players have played all maps.

	Map 1	**Map 2**	**Map 3**
Player A	600K		
Player B	800K	900K	700K
Player C		950K	

Table 4.1: Example Player Scores for Maps

By observation we estimate Player A's score on Map 2 to be $> 600,000$ and $< 900,000$ This is because Player B is better than A by observation of Map 1

The same observation can be made for Player C on Map 1.

We may continue on the predictions for Map 3, we ultimately yield a filled matrix.

	Map 1	**Map 2**	**Map 3**
Player A	600K	850K*	500K*
Player B	800K	900K	700K
Player C	900K*	950K	800K*

Table 4.2: Example Player Scores for Maps, *: Estimated

We can deduce the difficulty of the maps $\text{Map}_3 > \text{Map}_1 > \text{Map}_2$.

4.1.2 Insufficient Supports

We may estimate map difficulty through Player B's score, however, do we have sufficient evidence that Map 3 is truly the hardest, just based on Player B?

A reasonable doubt: what if Player B's score on Map 3 was badly done?

Consider Map 2, we have 2 players' score, this is more convincing, the chances of both players unintentionally doing badly is lower.

The idea of supports is simply the number of non-estimated values, higher support implies more evidence.

The supports can come from 2 sides:

- Number of scores per map.
- Number of scores per player.

Insufficient support for each category implies low evidence of reliability of estimation. Thus, we simply removed them if they are below a certain threshold.

4.1.3 Analogy of Score Estimation

In section 4.1.1, we discussed score estimation, a popular, similar problem is recommendation systems. Think movie recommendations.

	Movie A	**Movie B**	**Movie C**	**Movie D**
User A	5	4		
User B	5	4.5	5	3
User C	2	2	3	5

Table 4.3: Example Movie Scores by Users

Observe the movie ratings in Table 4.3.

Would you recommend Movie C or D to User A? By averaging scores, we see that Movie C and D are equally good. However, movie tastes are not one-dimensional. A better way is to weigh opinions of users with similar tastes higher.

Based on Movie A and B, we observe User B have **similar tastes** to A, while User C doesn't. Thus, User A would likely enjoy Movie C as B enjoyed it too.

This problem is solved using **Collaborative Filtering**.

With this, we can estimate User A's rating of Movie C and D, thus recommending the better movies.

4.1.4 Method of Score Estimation

Thus, similarly, we estimate scores with **Collaborative Filtering**.

> @Prof Though we have some initial results, we'll hold onto it until it's more finalized/

4.2 Getting Score Data

We yield raw score data from osu! Database. For this project, we download
`2022_04_01_performance_mania_top_10000.tar.bz2`
This contains high score data from the top 10,000 players.

- `osu_beatmaps.sql` Contains map info
- `osu_scores_mania_high.sql` Contains score info
- ...

However, it's in `.sql`, to make it easier downstream, we convert to `.csv`

Conversion of SQL to CSV

Luckily, Google Cloud (GC) has capabilities on converting it, albeit roundabout.

1. Upload `.sql` files to GC Storage
2. Migrate `.sql` files from GC Storage into GC SQL
3. Query data using BigQuery into `.csv`

We queries as follows

```
SELECT * FROM EXTERNAL_QUERY(
    "projects/project-name/...",
    "SELECT * FROM osu_beatmaps WHERE playmode = 3;"
);
```

Figure 4.1: Beatmap Info Query

We selected `playmode = 3` to avoid machine generated beatmaps, which are not representative of our true user-made map population.

```
SELECT * FROM EXTERNAL_QUERY(
    "projects/project-name/...",
    "SELECT * FROM osu_scores_mania_high;"
);
```

Figure 4.2: Score Info Query

We yield 11,453,737 million scores alongside 10,766 beatmaps from this dataset.

4.3 Filtering Score Data

After yielding the data, we need to filter away anomalies.

1. Remove maps not from osu!mania, that is, not a VSRG play [1]
2. Remove ”SV” maps, i.e. gimmicks that change difficulty of patterns
3. Filter scores from 600K 1M, anything below 600K is not representative of a ”good play”
4. Filter maps to only 4K and 7K. As keys except these have too little samples

[1] osu! has 4 different game modes, osu!mania is one of them

Chapter 5

Annex

5.1 Annex

Annex

I have been in the VSRG community for over 10 years and have tried difficulty estimation on-and-off as an amateur, thus I have good faith in the success of this research.

There is already an ongoing unpaid community project for this, though in very early stages, utilizing manual algorithms instead of DNNs, thus there will be additional acknowledgements outside of NTU.

I estimate that this project can be done within 1 year. I can draft a timeline for this if this goes through.

References

Etterna. (n.d.). https://etternaonline.com/
O2jam. (n.d.). https://o2jam.fandom.com/wiki/O2Jam
Osu! (n.d.). https://osu.ppy.sh/
Quaver. (n.d.). https://quavergame.com/