Star Rating Rebirth: A VSRG Difficulty-Rating Framework (Feb 2025 Update)

[Crz]sunnyxxy

February 17, 2025

Abstract

We propose an algorithm that assesses the difficulty of an osu!mania beatmap which is almost equivalent to assessment in all VSRGs in general.

Contents

1 Introduction

VSRGs (Vertical Scrolling Rhythm Games) have been present for over a quarter of a century. Each map, consisting of notes, would have a certain level of difficulty. For a lot of games, this difficulty is assessed by humans. However, osu!mania and Malody are rhythm games where, due to their community-based nature, the difficulty can only be rated by an algorithm for ranking purposes.

The question of finding a suitable algorithm for difficulty rating has attracted a large amount of research. However, no reliable, universal algorithm exists to this date. Stepmania has a somewhat reliable algorithm, but only for 4K rice maps. In this paper we propose a generalised algorithm for 1-10K including long notes, using osu!mania mechanism as a proxy (which could be easily generalised to other games).

This paper builds on a previous algorithm in 2019 (co-authored by me and ChlorieHCl), which some have seen before. That paper was an extremely outdated draft. We shall refer to it as "the 2019 algorithm". We restructure that prototype, rewrite the code, make a variety of improvements and perform evidence-based tuning to achieve highly satisfactory results. The tuning process involved using the huismet website iteratively and I would like to give credits to Natelytle and vernonlim for huismet integration.

2 Feature Engineering

2.1 Defining a Beatmap

First, we want to define a mathematical representation of a beatmap, so we can work with abstractions precisely.

All numerical values of time or column throughout this paper are integers.

Definition 2.1 (Available Space)**.** *An available space is a pair* (K, T) *where* $K \in \{1, 2, ..., 10\}$ *and* $T \geq 0$*.*

Definition 2.2 (Note)**.** *Let* (K, T) *be an available space. A note adapted to* (K, T) *is a tuple* (k, h, t) *where* $0 \leq k < K$*,* $0 \leq h \leq T$*, and either* $h \leq t \leq T$ *or* $t = -1$ *holds.*

Remark. *Here we have made a distinction between a 0ms long note (case* $h_i = t_i$*) and a tap (where we use the convention* $t_i = -1$*).*

Definition 2.3 (Ordering)**.** *Let* $n_1 = (k_1, h_1, t_1)$ *and* $n_2 = (k_2, h_2, t_2)$ *be two notes adapted to* (K, T)*. Then we compare* n_1 *and* n_2 *by* h*, then (if* $h_1 = h_2$*) by* k *and finally (if* $h_1 = h_2$ *and* $k_1 = k_2$*) by* t*. This induces a well-defined total order on the set of notes adapted to* (K, T)*.*

Definition 2.4 (Beatmap)**.** *A beatmap is the collection of:*

1. *an available space* (K, T)*,*
2. *a non-decreasing length-*N *sequence of notes* $(n_i) = ((k_i, h_i, t_i))$ *adapted to* (K, T)*,*
3. *a positive number* $q > 0$ *called the 300-window.*

The non-decreasing part ensures uniqueness if the beatmap itself is the same (which, interestingly, is not a property held by .osu files – the order of two notes at the same s depends on which note is placed first in the editor). The 300-window is defined to be two sided, in seconds, and for a tap. For example, if $od = 8$, then we use $q = 0.081$. It is also clear that our definition of a beatmap does not include SV or other visual components that are aside from the timing of the notes.

Remark. *The reason we use milliseconds for note timing and seconds for 300-window is that the raw data of notes are all in milliseconds and are integers, so it would be trickier to convert them into floats. This problem does not exist for hit window so we can normalise it immediately.*

Definition 2.5 (Hit Leniency)**.** *Hit Leniency is defined as*

$$x = \min\{0.3q^{0.5},\ 0.6(0.3q^{0.5} - 0.09) + 0.09\}.$$

Definition 2.6 (Legal Beatmap)**.** *A beatmap is legal if there exists no $i < j$ with $k_i = k_j$ such that $h_i = h_j$ or $t_i \geq h_j$.*

From now on, we can be assured what a legal beatmap is and do all calculations on that.

2.2 Overarching Principles

First, we ask a question: what truly makes a beatmap difficult? A naive approach would be to look at its density. It is well-known that the earlier versions of osu!mania star rating rely heavily on the peak density, whereas Malody relies on the average density. We know that density captures only a very small part of difficulty. Even if we use an approach that is more sophisticated than average (L^1 norm) or peak (L^∞ norm), the results would still be very poor.

The other extreme is to try to capture every possible pattern. First, the patterns of beatmaps are often evolving and extremely hard to enumerate even just in 4K. Second, the beatmap dataset is relatively scarce. If we classify, the risk of overfitting is extremely high. Another issue is that selecting a ton of mini-features is highly prone to a (non-zero-effort) *attack* where an attacker studies the algorithm and maps in a way that intentionally inflates or deflates the rating.

Our approach, therefore, is to select some crucial features that are both quite independent to each other and central to the difficulty of a beatmap, while trying to be generalisable and not being overly prescriptive. This relies on some good judgement. Fortunately, the author has played VSRGs long enough to extract some principal features, which are introduced in subsections 2.3 to 2.7.

2.3 Same-Column Pressure: $\bar{J}(s)$

This is a value that indicates how intense the notes on that single column at that time is. It is denoted $J_k(s)$ where the point (k, s) satisfies $0 \leq k < K$ and $0 \leq s \leq T$.

If there are no notes in that column, then $J_k(s) = 0$.

Otherwise, if the point (k, s) is strictly before the first note's head in that column or at least as late as the last note's head in that column, then $J_k(s) = 0$.

Otherwise, we can find two adjacent notes in that column $(k, h_l, t_l), (k, h_r, t_r)$ such that $h_l \leq s < h_r$.

Then we define $\Delta_k(s) = 0.001(h_r - h_l)$. Here the purpose of 0.001 is to normalise milliseconds into seconds. Humans perceive ideas like densities and intensities much better in seconds and the values assigned will be much easier to read.

Definition 2.7 (Jack Nerfer)**.**

$$j_k(s) = 1 - 7 \cdot 10^{-5} \cdot (0.15 + |\Delta_k(s) - 0.08|)^{-4}.$$

Now we can assign the values of (unsmoothed) pressure at each time and for each column:

$$J_k(s) = (\Delta_k(s))^{-1}(\Delta_k(s) + \lambda_1(x)^{1/4})^{-1} j_k(s).$$

In the 2019 algorithm, $\lambda_1 = 0.11$.

Now we smooth out:

$$\bar{J}_k(s) = 0.001 \sum_{t=\max(0,s-500)}^{\min(T,s+499)} J_k(s).$$

Finally, we obtain the Same-Column Pressure:

Definition 2.8 (Same-Column Pressure)**.**

$$\bar{J}(s) = [\sum_{k=0}^{K-1} (\bar{J}_k(s))^{\lambda_n} w_k(s)]^{1/\lambda_n}$$

where

$$w_k(s) = \frac{1/\Delta_k(s)}{\sum_{i=0}^{K-1}(1/\Delta_i(s))}$$

and in case there are no notes on both sides of s in column k, we regard $1/\Delta_k(s)$ as zero.

Here λ_n is an extremely important global parameter that represents a norm. It will appear repeatedly. In the 2019 algorithm, $\lambda_n = 4$. However we have now changed it to 5. A higher value pulls the rating towards the peak.

2.4 Cross-Column Pressure: $\bar{X}(s)$

If you have ever played 6K or 7K brackets, you know how painful it is. That's because we need to make alternating moves across fingers, which is highly stressful. Even 4K single-handed trills are more stressful than mashing the keyboard.

First, we need a value that signifies the intensity of such alternating movement between columns $k-1$ and k (columns -1 and K are considered empty). It is denoted $X_k(s)$ where $0 \leq k \leq K$ and $0 \leq s \leq T$.

To do this, just like in section 2.3, aside from the special cases at beginning or end, we can find two adjacent notes (within the two columns) $(k_l, h_l, t_l), (k_r, h_r, t_r)$ such that $h_l \leq s < h_r$. Write $\Delta_{k-1,k}(s) = 0.001(h_r - h_l)$.

$$X_k(s) = 0.16 \left[\max\{x, \Delta_{k,k-1}(s)\}\right]^{-2}.$$

Then we obtain $X(s)$ using a linear combination of $X(k,s)$ for $0 \le k \le K$. The coefficients are hardcoded and described below:

1K: 0.075, 0.075 (Mean = 0.075)
2K: 0.125, 0.05, 0.125 (Mean = 0.1)
3K: 0.125, 0.125, 0.125, 0.125 (Mean = 0.125)
4K: 0.175, 0.25, 0.05, 0.25, 0.175 (Mean = 0.18)
5K: 0.175, 0.25, 0.175, 0.175, 0.25, 0.175 (Mean = 0.2)
6K: 0.225, 0.35, 0.25, 0.05, 0.25, 0.35, 0.225 (Mean = 0.243)
7K: 0.225, 0.35, 0.25, 0.225, 0.225, 0.25, 0.35, 0.225 (Mean = 0.262)
8K: 0.275, 0.45, 0.35, 0.25, 0.05, 0.25, 0.35, 0.45, 0.275 (Mean = 0.3)
9K: 0.275, 0.45, 0.35, 0.25, 0.275, 0.275, 0.25, 0.35, 0.45, 0.275 (Mean = 0.32)
10K: 0.325, 0.55, 0.45, 0.35, 0.25, 0.05, 0.25, 0.35, 0.45, 0.55, 0.325 (Mean = 0.355)

As an example, in 4K we have:

$$X(s) = 0.175X_0(s) + 0.25X_1(s) + 0.05X_2(s) + 0.25X_3(s) + 0.175X_4(s).$$

Definition 2.9 (Cross-Column Pressure)**.**

$$\bar{X}(s) = 0.001 \sum_{t=\max(0,s-500)}^{\min(T,s+499)} X(s).$$

2.5 Pressing Intensity: $\bar{P}(s)$

We need something that is representative of the overall pressure. This is a bit close to the density used in Malody and osu!mania. However, we apply formulas that are a bit more nuanced.

First, we assign a value $P(s)$ where $0 \le s \le T$ which is the pre-smoothed version of pressing intensity. To do this, just like in section 2.3, aside from the special cases at beginning or end, we can find two adjacent notes (across all columns) $(k_l, h_l, t_l), (k_r, h_r, t_r)$ such that $h_l \le s < h_r$. Write $\Delta(s) = 0.001(h_r - h_l)$.

Definition 2.10 (Pressing Note Value)**.** *For each single note n_i where $t_i \neq -1$, define its intersection with the interval around time s:*

$$U_i(s) = 0.0013\Big[\min(h_r,\, h_i + 120,\, t_i) - \max\{h_l,\, \min(h_i + 60,\, t_i)\}\Big] + 0.001\Big[\min(h_r,\, t_i) - \max\{h_l,\, \min(h_i + 120,\, t_i)\}\Big].$$

Then the pressing note value is defined as:

$$v(s) = 1 + \lambda_2 \sum_{i=0}^{N-1} U_i(s).$$

In other words, note value roughly represents the total "amount" of taps and LNs on the interval $[h_l, h_r)$.

Definition 2.11 (Stream Booster)**.**

$$b(s) = \begin{cases} 1 + 1.7 \cdot 10^{-7} \cdot ((7.5/\Delta(s)) - 160) \cdot ((7.5/\Delta(s)) - 360)^2, & \text{if } 160 < 7.5/\Delta(s) < 360, \\ 1, & \text{otherwise.} \end{cases}$$

Definition 2.12 (Pressing Scaling Factor)**.**

$$f_p(s) = \begin{cases} [0.08(\Delta(s))^{-1}(1 - \lambda_3 x^{-1}(\Delta(s) - x/2)^2)]^{1/4} b(s) v(s), & \text{if } 0 < \Delta(s) \leq 2x/3, \\ [0.08(\Delta(s))^{-1}(1 - \lambda_3 x^{-1}(x/6)^2)]^{1/4} b(s) v(s), & \text{otherwise.} \end{cases}$$

It is clear that

$$\lim_{\Delta(s) \to 0} f_p(s)(\Delta(s)) = [0.02(4/x - \lambda_3)]^{1/4}.$$

For now we use the value $\lambda_3 = 24$. A higher value penalises synchronous notes more heavily.

By combining the previous factors, we define:

$$P(s) = (\Delta(s))^{-1} f_p(s) + 1000 \max(0, |\{i : h_i = s\}| - 1)[0.02(4/x - \lambda_3)]^{1/4}.$$

Remark. *Here 1000 is the inverse of 0.001 representing one millisecond. If there is a timestamp with more than one note, then the additional note is "compressed" into an instant which has an extremely high intensity, much like the Dirac delta function. Since $f_p(s)$ has a limit, there should be no jump in the difficulty rating if we move a note by 1 ms.*

Definition 2.13 (Pressing Intensity)**.**

$$\bar{P}(s) = 0.001 \sum_{t=\max(0,s-500)}^{\min(T,s+499)} P(s).$$

2.6 Unevenness: $\bar{A}(s)$

We want to penalise patterns that are very "even", when notes on adjacent columns are locally similar.

Definition 2.14 (Active Columns)**.** *A column k is said to be active at time t if there exists at least one note, indexed i, such that*

$$k_i = k \quad \text{and} \quad t \in [h_i - 150, \max\{h_i, t_i\} + 149].$$

We write the tuple of active columns at time t as

$$S_k^t = \left\langle k_0^t, k_1^t, \ldots, k_{K(t)-1}^t \right\rangle,$$

where $K(t)$ is the length of S_k^t and $k_0^t, k_1^t, \ldots, k_{K(t)-1}^t$ are the indices of the active columns in an increasing order.

Recall that $\Delta_k(s)$ is the space, in seconds, between two adjacent notes in column k to time s.

For columns $k_{\tilde{k}}^t$ and $k_{\tilde{k}+1}^t$ where $0 \le \tilde{k} < K(t) - 1$, we define their *gap difference* as

$$d_{\tilde{k}}(s) = |\Delta_{\tilde{k}}(s) - \Delta_{\tilde{k}+1}(s)| + 0.4 \max\{0, \max(\Delta_{\tilde{k}+1}(s), \Delta_{\tilde{k}}(s)) - 0.11\}.$$

Then we define the contribution to unevenness by each pair

$$A_{\tilde{k}}(s) = \begin{cases} \min\{0.75 + 0.5 \max(\Delta_{\tilde{k}+1}(s), \Delta_{\tilde{k}}(s)), 1\}, & \text{if } d_{\tilde{k}}(s) < 0.02, \\ \min\{0.65 + 5d_{\tilde{k}}(s) + 0.5 \max(\Delta_{\tilde{k}+1}(s), \Delta_{\tilde{k}}(s)), 1\}, & \text{if } 0.02 \le d_{\tilde{k}}(s) < 0.07, \\ 1, & \text{otherwise.} \end{cases}$$

Definition 2.15 (Unevenness)**.** *Let*

$$A(s) = \prod_{\tilde{k}=0}^{K(t)-2} A_{\tilde{k}}(s).$$

Then the Unevenness is

$$\bar{A}(s) = (\min(T+1, s+500) - \max(0, s-500))^{-1} \sum_{t=\max(0,s-500)}^{\min(T,s+499)} A(t).$$

2.7 Release Factor: $\bar{R}(s)$

The features in the previous subsections mainly deal with rices, or the density of LN bodies. We need something to account for the difficulty to release an LN at an appropriate timing.

First, we realise that if a long note is reasonably short or reasonably close to an inverse, then there is not much need to focus on releasing it.

Definition 2.16 (Spacing Index)**.** *For a long note n_i, let n_j be the next note in the same column. we define the head spacing index as $I_i^{(h)} = 0.001|t_i - h_i - 80|/x$ and the tail spacing index as $I_i^{(t)} = 0.001|h_j - t_i - 80|/x$. Then the spacing index is:*

$$I_i = 2/[2 + \exp(-5(I_i^{(h)} - 0.75)) + \exp(-5(I_i^{(t)} - 0.75))].$$

Remark. *I_i is between 0 and 1. If a release is easy because of either head spacing or tail spacing, then it dominates the formula and I_i becomes small (close to zero).*

Now we want to define LN Releases in a similar fashion as subsection 2.5. We don't really need a note value here.

Similar to usual, find two adjacent tails (across all columns) $(k_l, h_l, t_l), (k_r, h_r, t_r)$ such that $t_l \le s < t_r$. Write $\Delta^{(r)}(s) = 0.001(t_r - t_l)$.

$$R(s) = 0.08(\Delta^{(r)}(s))^{-1/2} x^{-1} (1 + \lambda_4(I_l + I_r)).$$

Definition 2.17 (Release Factor)**.**

$$\bar{R}(s) = 0.001 \sum_{t=\max(0,s-500)}^{\min(T,s+499)} R(s).$$

3 Overall Model

3.1 The Formula

Definition 3.1 (Local Note-Count)**.**

$$C(s) = |\{i : s - 500 \le h_i < s + 500\}|.$$

Remark. *We now have a vector* $(\bar{J}(s), \bar{X}(s), \bar{P}(s), \bar{A}(s), \bar{R}(s), K(s), C(s))$ *at each time* s*.*

Definition 3.2 (Strain)**.**

$$S(s) = [w_0((\bar{A}(s))^{3/K(s)}\bar{J}(s))^{1.5} + (1 - w_0)((\bar{A}(s))^{2/3}(0.8\bar{P}(s) + \frac{35}{C(s)+8}\bar{R}(s)))^{1.5}]^{(2/3)}.$$

Definition 3.3 (Twist)**.**

$$T(s) = (\bar{A}(s))^{3/K(s)}\bar{X}(s)/(\bar{X}(s) + S(s) + 1).$$

Definition 3.4 (Difficulty Function)**.**

$$D(s) = w_1(S(s))^{1/2}(T(s))^{p_1} + w_2 S(s).$$

Definition 3.5 (Density-Weighted Difficulty Percentiles)**.** *We sort the pairs* $(D(s), C(s))$ *in ascending order by* D *and then by* C *(although the second layer of sorting does not really matter). The sorted pairs are now* $(\tilde{D}(s), \tilde{C}(s))$*.*

Define the percentile of total local note-count function

$$F(s) = \frac{\sum_{u=0}^{s} \tilde{C}(u)}{\sum_{u=0}^{T} \tilde{C}(u)}.$$

Then the percentile of time covered at percentile of total local note-count is given by the following map:

$$s(p) = \inf\{\, t \in [0, T] \mid F(t) \ge p \,\}.$$

Then, the 93rd and 83rd difficulty percentiles are given by

$$D_{93} = \frac{1}{4}\Big(\tilde{D}_{s(0.945)} + \tilde{D}_{s(0.935)} + \tilde{D}_{s(0.925)} + \tilde{D}_{s(0.915)}\Big),$$

$$D_{83} = \frac{1}{4}\Big(\tilde{D}_{s(0.845)} + \tilde{D}_{s(0.835)} + \tilde{D}_{s(0.825)} + \tilde{D}_{s(0.815)}\Big).$$

Definition 3.6 (Weighted Mean)**.**

$$\mu = [\sum_{s=0}^{T}(D(s))^{\lambda_n} W(s)]^{1/\lambda_n}$$

where

$$W(s) = \frac{C(s)}{\sum_{i=0}^{T}(C(i))}.$$

Finally, we are ready to arrive at the star rating value.

Definition 3.7 (Star Rating)**.** *The star rating is defined as*

$$\mathrm{SR} = Q(0.97 \cdot [0.25\,(0.88\,D_{93}) + 0.20\,(0.94\,D_{83}) + 0.55\,\mu]^{p_0}\,\frac{8}{8^{p_0}}\,\frac{N_*}{N_* + 60}),$$

where N_* is given by

$$N_* = N + \frac{1}{2}|\{i : t_i \neq -1\}|$$

and

$$Q(x) = \begin{cases} x, & 0 \leq x < 9, \\ 9 + \dfrac{5}{6}(x - 9), & 9 \leq x < 10.2, \\ 10 + \dfrac{10}{13}(x - 10.2), & 10.2 \leq x < 11.5, \\ 11 + \dfrac{2}{3}(x - 11.5), & x \geq 11.5. \end{cases}$$

Remark. *We can in fact apply any monotone function to SR to change scaling.*

3.2 The Parameters

Let's recap all the free variables we have mentioned before.

- $\lambda_n = 5$: The overall norm. A higher value skews the rating towards the peak.
- $\lambda_1 = 0.11$: A scaling intercept for same-column difficulty.
- $\lambda_2 = 6.0$: The weight of LN bodies.
- $\lambda_3 = 24$: A slight penalty in $P(s)$ for (almost) synchronous notes.
- $\lambda_4 = 0.8$: The weight of Spacing Index.
- w_0, w_1, p_1, w_2, p_0: The late-stage coefficients that define SR.

We realise that $\lambda_1 = 0.11$ and $\lambda_3 = 24$ are rather hardcoded in the design process, so we are not interested in changing them.

$\lambda_n = 5$ is inevitably subjective, limited by the varied perception of ground set. If we pick different methods for the ground set construction (accuracy, score, etc.), the value will be different. For now we will just use this value and leave it there.

Of the remaining variables, λ_2 and λ_4 are considered hyperparameters as they are slightly deeper. They are set to be 6.0 and 0.8. On the other hand, w_0, w_1, p_1, w_2, p_0 are more superficial, so they are considered model parameters.

Originally we attempted to use a ground-truth model of ranked beatmaps (see Evening and Keytoix's websites for reference) to learn the parameters by minimising a loss function. However, I was always a bit unhappy with the scaling and some other nuances. Thus, we finally decided to abandon the systematic approach and set the values by hand. We then used the huismet site to check the new ranking and performance lists. This back-and-forth procedure was performed iteratively, with strong reliance on

my prior beliefs. This manual approach avoids the perceived unfairness introduced by any ML.[1]

The final parameters are:

$$(w_0, w_1, p_1, w_2, p_0) = (0.4, 2.7, 1.5, 0.27, 1.0).$$

4 Implementation

4.1 Data Structures

We already have an available parser by Imperial Wolf which converts any .osu file in mania mode into a reasonable format. From there it is crucial that we create the following lists:

- `note_seq`: $[(k_i, h_i, t_i) \mid i$ in range, sorted in (h_i) and then $(k_i)]$;
- `note_seq_by_column`: $[\ [(k_i, h_i, t_i) \mid k_i = k$, sorted in $(h_i)] \mid k = 1, 2, ..., K]$;
- `tail_seq`: $[(k_i, h_i, t_i) \mid i$ in range, sorted in (t_i) and then $(k_i)]$;
- `LN_seq_by_column`: $[\ [(k_i, h_i, t_i) \mid k_i = k$, sorted in $(h_i)] \mid k = 1, 2, ..., K]$;
- `LN_bodies`: the ith element stores $|\{(k_x, h_x, t_x) : h_x \leq i < t_x\}|$.

The implementation details are not very difficult. A time-based naive procedure is of space complexity and time complexity $O(K(T + N \log N))$. We know that for a reasonable map, $T < 10^6$ and $N < 10^4$. It takes only several seconds to calculate a long map in Python 3 (a very slow language).

4.2 Further Optimisation

We now introduce an important observation that further reduces both the space complexity and the time complexity to $O(N(K + \log N))$.

We observe that the variables $(\bar{J}, \bar{X}, \bar{P}, \bar{R})$ are piecewise-linear in time. This behaviour arises because their unsmoothed counterparts are step functions that can only change at note boundaries (i.e., at a note head or tail), or a boundary plus 1 ms when there is a Dirac-delta spike. These step functions are then smoothed by averaging over a window of ± 500 ms, which yields a piecewise-linear function over a set of *corners*.

For the variable $\bar{A}$, the unsmoothed quantities change at ± 500 ms of a note boundary. Consequently, $\bar{A}$ is piecewise-linear with *corners* occurring at the note boundaries and also at ± 1000 ms of the boundaries.

The variables C and K are themselves step functions that change at ± 500 ms of the boundaries.

It can be shown that the vector

$$(\bar{J}, \bar{X}, \bar{P}, \bar{A}, \bar{R}, K, C)$$

[1] I feel unsafe using any non-linear machine learning models on a noisy and scarce dataset. Does anyone else feel the same way?

is, component-wise, either piecewise-linear or a step function on the set of corner points, which is defined as

$$\{0,\, T\} \cup \bigcup_{s \in S} \{\, s - 1000,\; s - 499,\; s,\; s + 501,\; s + 1000 \,\},$$

where S denotes the set of note boundaries (not including -1).

Finally, the only non-linear layer in the overall difficulty computation arises when these variables are collected and combined into the final quantities S, T and ultimately D. Additional mathematical techniques or optimisations may be applied at this stage, although even a linear estimate is sufficiently accurate for gaming purposes.

Figure 1 shows the difficulty calculation over a ranked beatmap: Echoes of Memoria [Solace of Oblivion] by -mint-. The computation is done in the Jupyter Notebook environment. The timeseries plot shows $0.97D(s)$, which clearly indicates the difficulty over time. The dotted red line shows the star rating, which is 7.24 (rounded down).

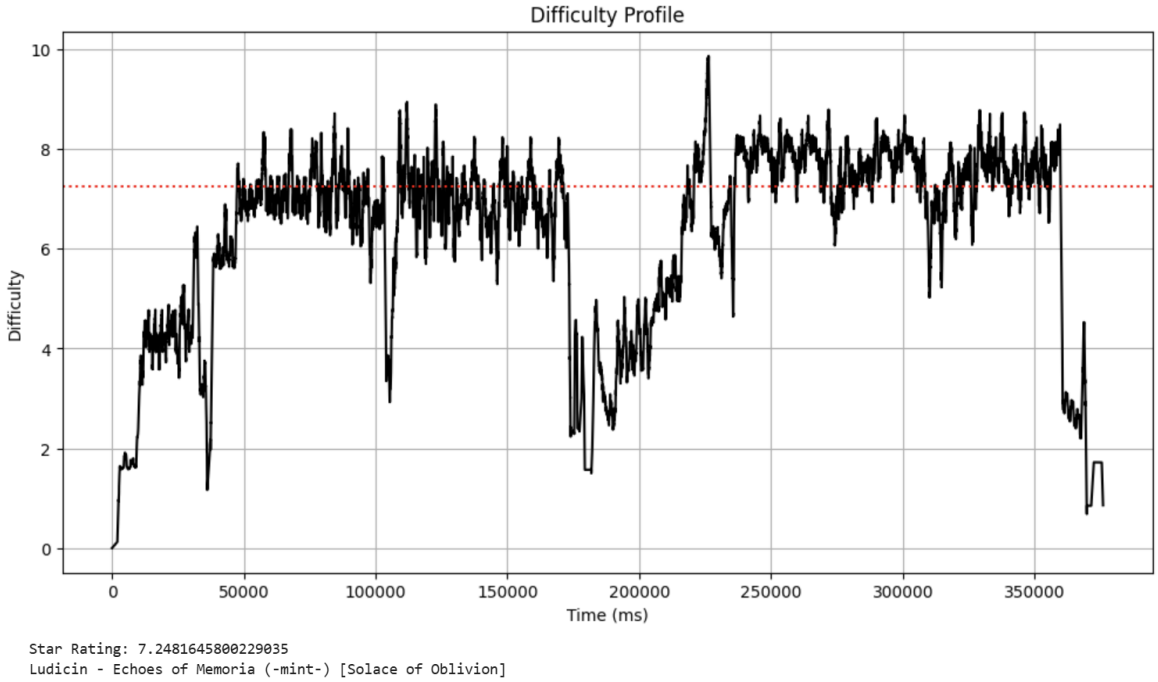

Figure 1: A visualisation of Echoes of Memoria.

5 Conclusion

We have proposed a new and accurate model for mania difficulty calculation based on first principles. The model is generalisable to all games that are VSRGs.

6 Credits

I am, of course, the designer of this mathematical algorithm. However, the success of this project, especially on the more practical side, would not have been possible without four other significant contributors whose work kept the project alive:

- **Natelytle** – provided the huismet integration for large-scale experimentation.
- **vernonlim** – same as above. He has been very patient when I was constantly adjusting things.
- **ChlorieHCl** – provided a prototype program to test my initial ideas. He was also heavily involved in the 2019 solution write-up.
- **Imperial Wolf** – provided the very first version of the Python code in 2019.

I also need to give credits to ChatGPT for a substantial amount of coding assistance and a small amount of LaTeX/writing assistance.