ppshift machine learning

In this document, we will be discussing methods of obtaining a credible way of classifying difficulty in VSRG maps. We will first establish what makes a map difficult, then we build from there!

Part I
Define Difficulty

1 A Difficult Map

What makes a map difficult, what is a difficult map? Could it be the following? The map was difficult because of ...

1. Failing
2. Combo Breaks
3. High Stamina Requirement
4. Low Accuracy

We discuss all of these scenarios and we will choose one to tackle, possibly integrate the other options into our calculations in the future.

Failing The most significant way that we can readily control if players fail is via **Health Drain** in which most VSRGs will implement. However, this value is inconsistent and will not provide useful information on higher **Health Drain** values due to lack of players passing certain maps.

Combo Breaks Combo Breaks analysis is another method that isn't consistent, whereby chokes can be random, creating too much noise on higher skill plays. Combo breaks mainly can only determine the **hardest** points on the map, it doesn't depict a difficulty cure.

High Stamina Requirements While stamina is a good way to look at difficulty, it can readily be derived from accuracy, which is conveniently what we'll be looking at next

Low Accuracy This is the best way to look at difficulty, because not only it gives us a figure, it tells us the story and correlation between **accuracy** and **patterning**. This will be the main focus of the document.

2 Difficulty from Accuracy

It is possible to measure difficulty, by just looking at accuracy, in-fact, it's quite straight-forward to do so.

Consider this...

1. $Player_1$ plays $Beatmap_A$ and $Beatmap_B$
2. $Player_1$ achieves $Accuracy_A > Accuracy_B$
3. Considering $Accuracy_A$ and $Accuracy_B$ are independent events
4. We deduce $Difficulty_A < Difficulty_B$

Now, on a larger scale...

1. $Player_{all}$ play $Beatmap_A$ and $Beatmap_B$
2. $\prod(Accuracy_A/Accuracy_B) > 1$
3. Considering $Accuracy_A$ and $Accuracy_B$ are independent events
4. We deduce $Difficulty_A < Difficulty_B$

We can further develop this algorithm to estimate what the difficulties would be by just looking at the ratios, I won't elaborate this further as this won't be the main aim of this document because **this only works if you have scores to pivot from**. This will not work with newly developed maps.

2.1 Replay data to accuracies

In order to compare A (Accuracy) and P (Patterning) we need to change our perspective, instead of looking at A as a number, we will look at it as a vector. Whereby we extract player data from provided replays.

2.1.1 GETting Replay data from osu!API

osu!API provides us with essential data, including the replay file itself. Not going into detail, we are able to extract all key taps (including releases) of the player during the play.

This is the Python code I used to extract replay from the osu!API

```python
def get_replay(beatmap_id: int, mode: int, user: int):
    param = {
            "k": str(api_key),
            "b": beatmap_id,
            "m": mode,
            "u": user
            }
    api_url = base_api_url + "get_replay"
    response = requests.get(api_url, params = param)
    json_data = json.loads(response.text)

    # We will decode the content here
    # But we check the status code before proceeding
    time.sleep(6) # Pause

    # If response is not 200
    if (response.status_code != 200):
        return "bad_request", response.status_code

    decoded = base64.b64decode(json_data["content"])
    decompressed = str(lzma.decompress(decoded))[:-2]

    # We split it into a nested list
    output = [x.split("|") for x in decompressed.split(",")]

    # Drop the last 2 useless elements
    [x.pop(2) for x in output]
    [x.pop(2) for x in output]
```

```
return output, response.status_code
```