Michael Turnell

Problem Statement and Background:

Slippi is a free software package developed to record replays for *Super Smash Bros. Melee*, a platform fighting game for the Nintendo GameCube. It was released early 2019. It runs on software-modded *Wii* consoles to collect pre-frame data (namely, controller input) and post-frame-rendered game states. With this, a GameCube emulator (*Dolphin*) can reconstruct the replay and we can store videos of tournament games in much smaller file formats.

The problem, however, is that the game doesn't know who has sat down to play it. It knows, for instance, that the two players have chosen Fox and Marth, two of the most common (and considered 'best') characters. And it knows what stage the match was played on, and who won, and how long the game took, and what every frame looked like in between.

Generally, most major tournaments that use Slippi have only been recording replays on the livestream setup. The sets played on the livestream are generally edited, labelled, and uploaded to YouTube playlists. These video titles contain both the player's names and the characters they used. Because of this, in theory, it should be possible to match a video to a slippi replay, and in so doing gain the ability to label a slippi replay with the players' names.

I wanted to do this because it is far more interesting to be able to generate per-player statistics than per-character. (e.g. How efficiently does Team Liquid's Hungrybox kill his opponent vs how efficiently do Jigglypuffs kill their opponent), but there were a few hurdles.

Hurdles:

-YouTube video titles: If the video title doesn't contain all the set information I need (both players' names, both characters they used in the set) in a consistent way throughout the playlist, it'd be painstaking to extract that information. As such, I was reduced to using only tournaments run by BTSSmash.

-Slippi set collation: Slippi records games. Tournaments are run in sets, either best of 5 or best of 3. So to connect slippi files to sets, I needed to 'bucket' slippi replays into sets

-Insufficient unique identifiers: *Melee* has 26 characters. However, in tournament play, 8 are extremely common, 4 more are kind of rare, and the remaining 14 practically unseen. Because of this, specific character matchups (Fox v Fox, Falco v Marth) are incredibly common, so confidently stating 'these replays are this set and thus these players' is difficult.

Extraction:

The data sources I set out to use were the .slp replay files, hosted on slippi.gg, and the uploaded YouTube playlists hosted on BTSSmash's channel on youtube.com. Originally, I had intended to use the smash.gg API in order to collect additional set information, but smash.gg tournament data is uploaded and input to the bracket during the tournament, is sometimes rushed, and is often wrong and never corrected. The data that -could- be relied upon (such as who won a given set) I did not consider as useful enough to use.

The .slp files for the tournaments I chose to use were scraped from slippi.gg (Slippi_Scraper.py). I stored these locally. The Youtube titles were scraped into a local mongoDB database (Youtube_Scraper.py).

Transformation:

As stated before, I needed to bucket the slippi files into sets, and sets do not have a consistent length, since they are either best of 3 or best of 5, they can range from 2 to 5 games. Because of this I chose to use a mongoDB to store the records.

Slippi has a python module (py.slippi) that allowed me to parse each replay file as a Game object. From this I gained:

-The timestamp of the game's start
-The game's duration, and therefore when the game ended
-The ports* in use
-What characters were played
-What stage was played on
-The winner of the game

For each Slippi file that I had scraped, I walked through the directories and formed a dictionary containing this information, and uploaded each dictionary as a record to mongoDB. (Slippi_Parser.py)

---

*A few notes about *Melee*'s tournament ruleset before continuing: The GameCube has 4 controller ports, and players can choose any of them, 1 through 4. These do not change mid-set. Though players pick a character at the start of a set, players can change characters between games. Generally, though, players will only play one character. Finally, due to *Melee*'s intense technical dexterity requirements, most players will opt for a 'handwarmer' game before their set.

Bucketing:

Given what I know about tournament structure, I came up with the following logic for collating the games into sets:

1) If there's more than 4 minutes between the end of a game and the start of a game, the start of the new game is a new set. Even the most well-run tournaments struggle to rotate players out on the same setup that quickly.
2) If one of the ports that were in use changes (e.g., port 1 vs port 4 changes to port 1 vs port 2), that game is a new set. After attending tournaments for 12 years, I have seen ports change mid-set twice. This ended up being a safe assumption.
3) If both characters change (e.g., Fox v Marth becomes Sheik vs Falco), that game is a new set. This is the assumption I'm the least confident about, but double counter-picks are very, very rare, to the point that them happening is "newsworthy".
4) If, after applying these rules, we have a 6 game set, drop the first game by assuming it was a warmup. This is because if a set goes all 5 games, a handwarmer beforehand would make it look like a 6 game set.
5) If, after applying these rules, we have a 1 game set, drop the set. This can happen if a non-tournament match was played on a slippi recording setup.

I ran this logic (Set_Collation.py) and ended up with many "flagged" (too large/long) sets, so I added some additional steps. For those sets flagged after the "naive" above logic, go through each individual game using py.slippi and see if there was a warmup game. I did this by calculating some basic common factors of warmup games:

-The players generally do not hit each other with moves, so the damage dealt totals are low

-The players generally will run-off stage to end the game once they are done warming up, so there are many "self-destructs" at 0% damage

-In a non-tournament game, pause is enabled, allowing the players to reset out of the match. If they do so, this is flagged in the slippi file.

If any of these conditions were detected, the game was marked as a 'warmup' game and discarded. Then, the sets were run back through the "naive" logic for set-length/time.

Still, there were a few flagged sets that remained. Sixteen sets were still too long, so I unfortunately needed to look through them manually to determine what was happening. Six of the sets were very long blocks of "casual" (non-tournament matches), which I ascertained by comparing the timestamps of the replays to the schedules of the tournaments, as well as what stages were played on. (Generally, tournament set stage choices follow a pattern depending on the character matchup, while casual match stages are chosen randomly). The other 8 needed to be split into 2 sets but had not been by my logic. All of them had the following:

-Under 4 minutes between game end and start (about 3:50 on average)

-No port changes

-Only 1 character change (e.g. Pikachu vs Yoshi to Pikachu vs Marth)

I tried rerunning the bucketing logic to have a lower 'time limit' but that created far too many "false sets" where a player took too long between games.

These manual drops/splits were done in Visual_Inspection.py

Title Formatting:

With my slippi replays bucketed into sets, I now needed to standardize the information I had scraped from the YouTube playlists. The py.slippi module has a list of values for each character that it outputs, which sometimes conflicts with the “common” usage. This required transforming the YouTube data via aliasing. The easiest example here is Donkey Kong -- slippi uses “DONKEY_KONG”, while YouTube titles might use “Donkey Kong”, “DK”, or even “D. Kong”. Thus I had to hard-code an alias list for each character to match the YouTube character names and transform them in the mongoDB. (Youtube_Titles.py)

Set-Title Matching:

Finally, the last transformation step was to attach a set of slippi replays to a given set title. A set title contains:

-player names
-character names
-what tournament it was played at

While my sets of slippi replays contained:

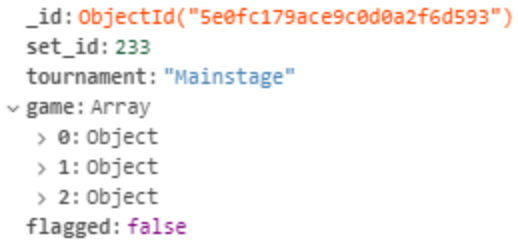
-characters played
-what tournament the replays were from
-a unique set id

From this, I compared the two lists and found matches based on characters played. I prioritized “exact” matches but accepted “loose” matches. Sometimes, despite a player using more than one character, the YouTube title only lists the character used the most. But, if a title lists multiple characters, it will list all the characters used.
Using Youtube-Slippi_Matcher.py, I stepped through both lists and generated possible matches for each set. This is where I “failed” due to one of the hurdles I mentioned earlier. Certain character matchups are absurdly common, and thus if a set title is for Fox v Fox, there are a dozen possible replay sets for Fox v Fox in a given tournament. Without more information, they aren’t differentiable. Because of this, I chose to give each set-title a list of “possible” replay sets, and marked it as a “confident” match if there was only 1 replay set found.

This left a handful (7) videos without a replay set match, which I assume is either due to mislabelling of the YouTube video, the nonexistence of the replay files due to slippi failure, or an undiscovered oversight in my set collation logic.

Loading:

So at the end result, I have the following structures:



A “set object”, containing its ID, the tournament the replays are from, a list of each slippi replay file with its filepath and some basic game information, and a flag marking if it the set is “too long”.

And a “video object”, containing its ID, the tournament it is from, the players, what characters they used, a list of set_ids it matches to, and if it is a “confident” (unique) match.

These are loaded in separate collections in mongoDB. I used noSQL due to my fields having variable length (the game array, the char1/char2 arrays, and the matched_sets array).

Final Thoughts:

This approach is ultimately non-scalable -- there’s too much need for manual inspection of the output that with a larger number of tournaments/videos/sets it’d be untenable. The correct solution for matching sets to replays is an upstream approach as opposed to a post-hoc pairing. Upstream approaches might include:

-marking replay files as “these belong to the same set”
-attaching replay files to a given set in smash.gg
-attaching player names to replay files mid-tournament
-attaching replay files in the description of the youtube videos

All of these would require additional labor by tournament organizers, so an ‘automated’ approach should still be searched for.

I am disappointed that I failed to achieve perfect matching, but I think my approach would work for smaller groupings -- perhaps only the Top 16 of a tournament. Top 16s usually do not run concurrent sets, so I would gain additional information via replay timestamping for matching a set of replays to a specific tournament set.

However, overall I was glad to be able to practice web-scraping and learn about the py.slippi module more in-depth, and to have produced an end result to show off.