Data Analytics Case Study: An Analysis On Efficiency of a Tetr.IO player

Written by: Lim Guowei

GitHub: https://github.com/limgw1

LinkedIn: https://www.linkedin.com/in/limgw1/

Table of Contents

Phase 1: Ask

In this case study, I used historical data from Tetr.IO's to evaluate historical player statistics, determine trends, and tried to identify a measure of "true efficiency" and "macroplay". I was then able to translate the insights into actionable steps and suggestions for future research based on those findings.

Statement of Business Task

- To attempt to identify metrics that can quantify the skill level of a player's "macroplay".
- To analyze if the community-wide accepted metrics of "efficiency" are true efficiency

Phase specific questions

What topic are you exploring?

A video game, known as Tetr.IO. A guideline modern Tetris game that involves topping the opponent out with attacks.

What are the problems you are trying to solve?

What separates a good Tetris player from a great Tetris player? Is it possible to quantify the difference between the world champion, who has only 1 loss in their entire professional career, and a Tier 1 player, many of which have not been able to beat the world champion despite appearing equally skilled on the surface-level?

What metrics will you use to measure your data to achieve your objectives?

I will be using the metrics that Tetr.IO's API provides such as the 10 game averages of: Attack per minute (APM), Pieces per second (PPS), Versus score (VS) and Glicko-2 Rating.

Who are the stakeholders?

The stakeholders include me, CCRed95#6170 on Discord as they have been very helpful to me by providing guidance on how to work with Tetr.IO's dataset as well as friends and fellow guideline Tetris players who are interested in the insights the project can provide.

Who is your audience for this analysis and how does this affect your analysis process and presentation?

The Guideline Tetris community is my audience for this analysis. The process will remain technical, but the presentation may have to be explained in easier to understand terms as I wish to share my findings with everyone, regardless of skill bracket.

How will this data help your stakeholders make decisions?

Ideally it would bring forth insights that would help everyone understand the game better and understand the techniques that separates a good player from a great player. If not, at the very least start a discussion which may eventually lead to better answers.

Phase 2: Prepare

Data Collection

Two types of data will be used for this analysis.

1. A most recent snapshot of the data from Tetr.IO itself via the API provided by its developer, osk.
2. Historical data of the same data that has been archived by Tenchi#0870 on Discord. Download source: https://archive.org/download/tetrio-stats (8 Apr 2020 to 28 Feb 2022)

Obtaining data for most recent snapshot

The recent snapshot of data from Tetr.IO is done by making a direct API get request call to https://ch.tetr.io/api/users/lists/league/all . There are several methods to do this. Two of which I will show.

Option 1: Using Python

```
```(Python)
import requests
import json
res =
requests.get('https://ch.tetr.io/api/users/lists/league/all').json()
with open('leagueData.json', 'w') as json_file:
    json.dump(res, json_file)
```
```

Option 2: Using Postman

Postman is an API platform for building and using APIs. Postman simplifies each step of the API lifecycle.

1. Make a GET request to https://ch.tetr.io/api/users/lists/league/all via Postman as shown in Figure 1
2. Click the “Save Response” button at the top right of the response section and save directly to the computer as shown in Figure 1.

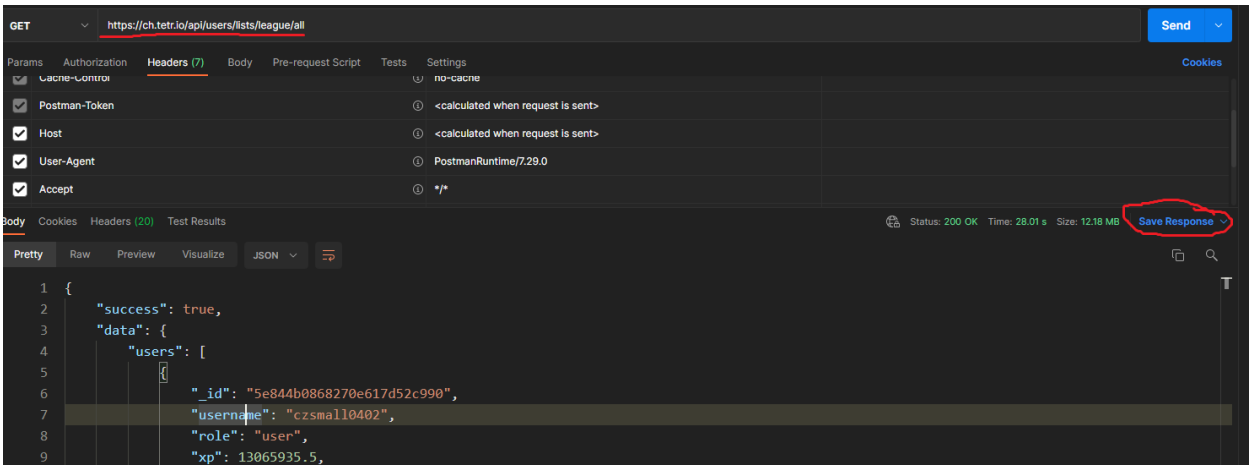

Figure 1: Screenshot of Postman. Enter the link at the red underline and click "Save Response" indicated by a red circle.

Obtaining historical data

https://ch.tetr.io/api/ does not provide an API call for historical data, probably due to the fact that historical data is very large, will grow exponentially as the player base increases and hence uneconomically viable for storage by an indie developer such as osk. Fortunately, the data has been archived by a fellow community member, Tenchi since April of 2020. Upon request, Tenchi has provided the download link to all the data that they have collected which can be accessed and downloaded by anyone in https://archive.org/download/tetrio-stats. The data is stored into folders named after the date and time collected ("2022-02-28T230001", "2022-02-28T220001", "2022-02-28T210001", …). Inside each of these folders is a single JSON source file named "league.json". The JSON data in the file can then be processed into a more workable form such as in a database.

Credibility of the data

The credibility of the most recent snapshot data is as ideal as it gets in terms of ROCCC (Reliability, Original, Current, Comprehensive, Cited). The data was obtained directly from the database of the game itself, making it both reliable, original, cited and as comprehensive as possible. By making the API call when you need the data, it is as recent as possible. However, as this report is static, I will be using the most recent data at time of writing. As the database updates once every hour, the latest player data will be a snapshot at 2:00pm(+8GMT) 26th March 2022.

The credibility of the historical data is slightly less ideal in terms of ROCCC, however the data is deemed to still be reliable. Since the data is historical, it is by no means current. The data is a series of snapshots dating from 7th April 2020 to 28th February 2022. It is as comprehensive as the recent snapshot data, but it is from a third-party source, compiled and archived by Tenchi. However, Tenchi describes obtaining the data by making hourly calls to the Tetr.IO API, similar to the method showed in the "Obtaining Data for most recent snapshot" section. As Tenchi is an active member of the Tetr.IO community and frequently converses with the moderators and the developer of the game himself, osk, it is safe to assume that the data archived by Tenchi is

unbiased, untampered with, and reliable, with errors in the data stemming from code or hardware related issues.

Format of the data

An example of a single row of data in the latest snapshot in Tenchi's archive (28th Feb 2022 11pm) is shown below.


```
{
    "_id": "5e844b0868270e617d52c990",
    "username": "czsmall0402",
    "role": "user",
    "xp": 13065935.5,
    "league": {
        "gamesplayed": 358,
        "gameswon": 333,
        "rating": 24998.617106525395,
        "glicko": 4081.219783349168,
        "rd": 85.49595503571086,
        "rank": "x",
        "apm": 143.06,
        "pps": 2.42,
        "vs": 297.95,
        "decaying": false
     },
    "supporter": true,
    "verified": true,
    "country": null
}
```


The table below describes the meaning of each of the variables in this singular user.

Table 1: Table of variables and their significance

_id	Unique identifier for each account which identifies them even if the name is changed
username	Current name of the account
role	Identifies if the account belongs to a normal user or a moderator
xp	Total experience gained throughout the account's history
league-gamesplayed	Total number of ranked Tetra League matches played
league-gameswon	Total number of ranked Tetra League matches won
rating	Tetr.IO's unique quantifier of skill, ranges from 0-25000, calculated based on glicko. This score is displayed first and foremost compared to TR for readability
glicko	Glicko-2 rating used by Tetr.IO to quantify a player's skill. See: http://www.glicko.net/glicko/glicko2.pdf for formula

rd	Stands for rating deviation. Part of the Glicko-2 formula
rank	Skill bracket in Tetr.IO. See https://cdn.discordapp.com/attachments/696095920248455229/721851657188671578/league-cheatsheet.png for the infographic posted by osk, the developer of Tetr.IO
apm	Attack per minute. Attacks are rows of garbage lines sent to opponents when you clear lines. Full damage table shown in this infographic by osk https://cdn.discordapp.com/attachments/673303546564968566/716080410702118922/2020-05-30_02-07-18.png
pps	Pieces per second. Quantifies the number of pieces placed every second.
vs	Versus score. Calculated as $vs = \frac{attacks\ sent + rows\ cleared}{PPS}$
decaying	Part of Glicko-2, if a player has not played a game in a long time, the RD will begin to increase. When decaying = true, the RD of a player is increasing by 1 a day.
supporter	If supporter = true, it means that the account has bought or been gifted Tetr.IO supporter, which is a way to support the developer and the game.
verified	If verified = true, it means that the player is who people think they are. Reserved for notable players such as popular streamers and top players who are frequently impersonated by others.
country	States the country of residence of the player. If null, means the player set their account settings to hide the country.

Phase 3: Process

Technologies used: MySQL Workbench
Programming languages used: SQL, Python

Data Cleaning

There are some errors in the data which are not apparent when first dealing with the data. The four errors in the data that I have encountered during my process of aggregating the data are:

1. Incorrect file name
2. Missing “league.json” file
3. Present “league.json” file but file is empty
4. Present “league.json” file but string is cut off or incomplete
5. Present “league.json” file but API call failed

The incorrect file name error only occurs in the folder “2020-04-08T174200” which also happens to be the earliest folder in chronological order. In the zip file downloaded from the source listed in Phase 2, the file name for the JSON file was “leaderboard-xp-2020-04-08”. The most likely reason for this is that a naming convention was not established due to it being the first ever snapshot saved. As this was the only occurrence of this issue in the entire set, the file name was manually changed to ”league.json”.

The second case is of the “league.json” file being missing entirely. An instance of this case is in the folder “2020-04-18T143001. In order to find these cases, a Python script is written to check the length of the directory of each folder. If the check returns 0, that means that the folder is empty, missing the “league.json” file, and is then deleted by the script.

In some cases, such as in “2020-07-30T120003”, the “league.json” file is present but the entire file is missing. In order to find these cases, a Python script is written to detect if the “league.json” file has a file size of 0 bytes, indicating that it is empty. If yes, then the file is deleted by the script.

The fourth case, such as in “2021-11-01T070001”, the “league.json” file is incomplete. While unconfirmed, it can be hypothesized that there was an error when saving the JSON file. The way to detect files like this is to open and load the JSON file. If the string is incomplete, it will return an error. The script that was written consists of a try/catch statement which will catch these exceptions. The script was written to append a record of the problem folder in an array containing the names of all problem directories. The array is printed out at the end of running the script. The original idea was that in the interest of salvaging whatever remaining data was in that incomplete string, I would not delete the folder, but manually close up the string. However, this made for reading the manually closed up JSON string difficult as it was no longer in the specific JSONL format. The manually closed up strings were unable to be read by the JSON reading packages in R. After that, it was decided that it was best to delete the folders and perform all future file reading and writing to check if a file size is above 2000 bytes.

The fifth case is failed API calls were due to an invalid authorization token, such as in “2020-04-18T144001. To check for failed calls, simply run a Python script that checks every single file for the key value pair of `{“success”:”false”}`. This is always the first part of the JSON string. However, deleting the folder upon detection proved difficult as the script was in the middle of reading a file inside the folder, which caused `os.remove()` to be unable to remove the “league.json” file. This then caused `os.rmdir(`) to be unable to remove the folder in question. Instead, the solution proposed was to add the folder name to the array of problem directories mentioned in the fourth case, which will be manually deleted. The script to check for such folders was combined with the check for the fourth case, which entails putting the `if data[“success”] is False` check in the try block of the script.

The script to check for all these errors, “checkErrors.py”, can be found in https://github.com/limgw1/google-data-analytics-capstone. Do note that case 1, case 4 and case 5 have to be manually dealt with.

Removing redundant league data

As stated in Phase 2, the historical data of all the players in the Tetra League competitive ladder, is very large, and will only continue to grow. The total file size of the entire archive approaches 98GB. Dealing with data this big is impractical and computationally expensive.

Two techniques have been used to reduce the file size.

1. Filter out data that is irrelevant to the analysis, such as role, xp, league-gameswon, decaying, supporter, verified, country
2. Removing redundant league data over time (Credit to CCRed95 for the idea, explained in the next section)

As explained in “Obtaining historical data” section in Phase 2, the historical data of every user in the Tetra League Competitive Ladder is a collection of folders containing the JSON file of every hourly snapshot of the data of every user in Tetra League. This means that there will be a lot of redundant data. For example, if a user does not play a single game for a whole day, there will be 24 snapshots where the player has the exact same data. In order to overcome this situation. A database is created to store the data of every single player in Tetra League **when it changes.** An extra variable will be added to indicate when the data is recorded. Hence, the earlier described 24 snapshots of data of a player can be reduced to a single snapshot. The variable that will be used to track changes in the stat is “gamesplayed”. All other gameplay relevant stats such as “rating”, “glicko”, “rank”, “apm”, “pps” and “vs” will not change until a game is played. This allows us to recreate a database with only the relevant data, when it changes, as long as we add another variable to track when it changes.

With all of this in mind. We compile the data in about 17000 snapshots of historical Tetra League data into one singular database table. The schema of the table will include these columns.

Table 2: Table of variables and the data they store.

_id	Same name as the previous table
username	Same name as the previous table
verified	Same name as the previous table
apm	Same name as the previous table
pps	Same name as the previous table
vs	Same name as the previous table
tr	Stores the rating variable from the previous table. Name changed to the term more commonly used by the community
tl_rank	Stores the rank variable from the previous table. Name changed to avoid problems down the line with rank() functions from the R programming language
glicko	Same name as the previous table
rd	Same name as the previous table
games_played	Added an underscore to improve readability
games_won	Added an underscore to improve readability
delta_date	Stores the datetime data of when the data is obtained from (Indicates when the data changed to the state in the specific row)

Data Aggregation

As the singular table proposed in the "Removing redundant league data" section is very large (estimated to pass the 1 million rows mark), a database was used. Since the table is relatively simple, with no joins required, MySQL was the choice of RDBMS (Relational Database Management System). MySQL has a visual database design tool, known as MySQL Workbench which allows for writing and executing scripts easily as well as displaying the table.

In the interest of using as little memory as possible for each column, the following data types will be used for each variable.

Table 3: Table of variables and the data types.

_id	CHAR(24). All IDs are 24 characters long
username	CHAR(16). All usernames in Tetr.IO are at most 16 characters long
verified	BOOLEAN
apm	DECIMAL(5,2). Data provided is only accurate to 2 decimal places
pps	DECIMAL(5,2). Data provided is only accurate to 2 decimal places
vs	DECIMAL(5,2). Data provided is only accurate to 2 decimal places
tr	DOUBLE. Used to be an integer but decimal places were added later on
tl_rank	Stores the rank variable from the previous table. Name changed to avoid problems down the line with rank() functions from the R programming language
glicko	DOUBLE. Used to be an integer but decimal places were added later on

rd	TINYINT unsigned. Has many decimal places but since RD does not determine a player's skill, rounded to nearest integer and unsigned as it can only go between 60-100
games_played	SMALLINT unsigned with highest number of 65535. Estimated time of hitting this number is mid to late 2030 by stqrm. Consider changing to INT which accounts for numbers up till 4 billion
games_won	SMALLINT unsigned with highest number of 65535. Estimated time of hitting this number is mid to late 2030 by stqrm. Consider changing to INT which accounts for numbers up till 4 billion
delta_date	DATETIME

The script for creating this database is shown in the code chunk below or can be downloaded by downloading the file "createDeltaTable.sql" from https://github.com/limgw1/google-data-analytics-capstone.

```(SQL)
-- -- Creates Database
-- CREATE DATABASE tl_delta_db;
-- Deletes Table
DROP TABLE tl_delta_table;
-- Creates Table
USE tl_delta_db;
CREATE TABLE tl_delta_table(id CHAR(24), username CHAR(16), verified
BOOLEAN, apm DECIMAL(5,2), pps DECIMAL(5,2), vs DECIMAL(5,2), tr
DOUBLE, tl_rank CHAR(2), glicko DOUBLE, rd TINYINT unsigned,
games_played INT unsigned, games_won INT unsigned, delta_date
DATETIME);
```

In order to populate the table, we must first collect the data from about 17000 separate "league.json" files, each stored in a folder with the name of the timestamp which closely resembles the UTC datetime format. To do this, a Python script is written to automate the process below:

1. Open the timestamp folder (e.g. "2022-02-28T150001")
2. Access the "league.json" file
3. Parse the JSON string into data
4. Iterate over every user (check for either "_id" or "username")
5. Check if redundant data (if games played is same as previous folder's data, stored in a dictionary for O(1) access time.)
6. If not redundant, save each variable into array
7. Write each variable into its respective column in the table by executing SQL code.

The code for this script can be found at https://github.com/limgw1/google-data-analytics-capstone under the name "importFullTable.py". Do read the comments in the code to understand what each line does. Also do note that some changes need to be made to the code in order for it to work, for example changing the directory link to match the location of your files in your local computer. The process will take an estimate of over 2 hours with the current data in Tenchi's archive page. If all steps are followed, the size of the database will be 843MB, which is 0.84% of the original 98GB.

**Disclaimer: The code in importFullTable.py has been updated to include the code necessary for phase 4, which includes extra information. See Phase 4 to read further.

Saving file in CSV format

After the data has been successfully populated, the data is once again saved in the CSV (comma separated values) format. This allows for easy upload to Kaggle and reading in R. Fortunately this can be done in SQL as well. Simply run the below script in MySQL Workbench. Note that there is a specific default folder where MySQL saves the CSV file, this folder will be different from mine.

```(SQL)
SELECT 'id', 'username', 'tlrank', 'apm', 'pps', 'vs', 'tr', 'glicko',
'rd', 'gamesplayed', 'gameswon', 'verified', 'deltadate'
UNION ALL
SELECT id, username, tl_rank, apm, pps, IFNULL(vs, "N/A"), tr, glicko,
rd, games_played, games_won, IFNULL(vs, "N/A"), delta_date
FROM tl_delta_table
INTO OUTFILE 'C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\tl-
delta-table.csv'
FIELDS ENCLOSED BY '"'
TERMINATED BY ','
ESCAPED BY '"'
LINES TERMINATED BY '\n';
```

Alternatively, you can download the code by downloading the file "writeToCSV.sql" from https://github.com/limgw1/google-data-analytics-capstone.

Most recent snapshot data

To populate a table with the most recent snapshot data, the code for creating a table and saving the file in CSV is more or less the exact same. All that is needed is to change the table name and remove delta_date as a column in "createDeltaTable.sql". For "writeToCSV.sql", remove deltadate, delta_date and change the name of the CSV file accordingly.

Download Tetra League Historical Data from Kaggle

If the reader chooses to skip all these steps and download the CSV file directly, I have uploaded the full table on Kaggle under the name "tl-delta-table-v2.0.csv", you can access it by going to this link. https://www.kaggle.com/datasets/limguowei/tetrio-tetra-league-historical-data

Phase 4: Analyze

Technologies used: RStudio, MySQL Workbench, Visual Studio Code
Programming languages used: R, SQL, Python

Now that the data is collected, is it time to evaluate it to answer the business questions. But before that, some preliminary knowledge of the game needs to be presented to those less familiar with the game.

VS Score

In Tetr.IO, the VS score is calculated by summing up the total damage sent to an opponent (Offensive power) and total number of garbage lines (lines received by an opponent's damage) cleared (Defensive power) divided by the total running time of the match and multiplied by 100. To understand it better, refer to the snapshot below.

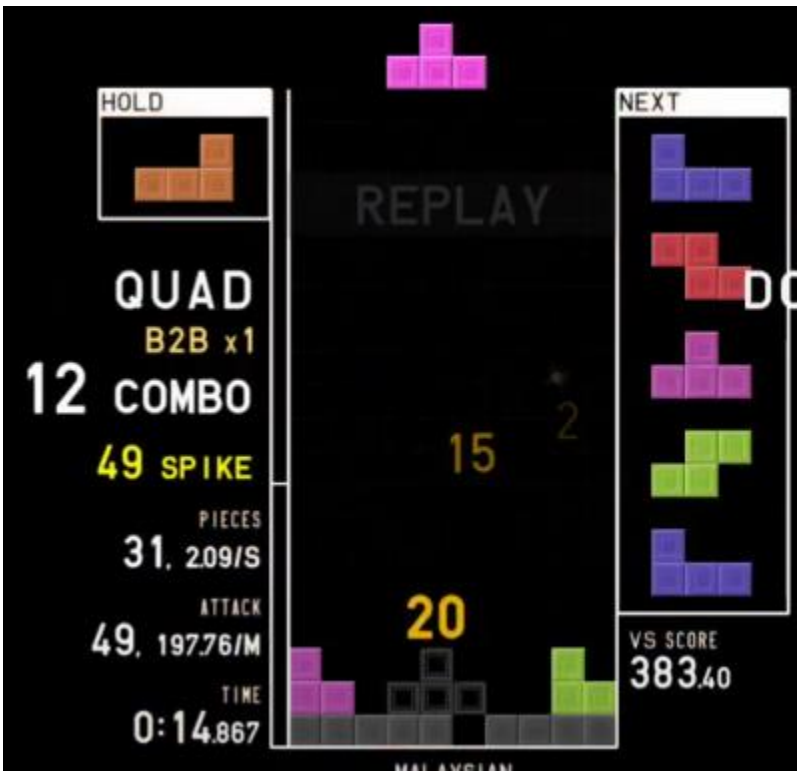

Figure 2: Screenshot of a Tetr.IO board

At the current screenshot's point in time. The player has sent a total of 49 attacks and cleared a total of 8 garbage rows (the rows with only grey blocks and one singular empty space). The VS score calculation would be:

$$VS\ Score = \frac{Attacks + Downstacks}{Game\ Time} * 100 = \frac{49 + 8}{14.867} * 100 = 383.40$$

Efficiency

Generally, the Tetr.IO community have two commonly accepted definitions of Efficiency, one based on the VS score and another based on Attacks per minute (APM).

$$Efficiency = \frac{VS\ Score}{PPS} \quad \boldsymbol{OR} \quad Efficiency = \frac{APM}{PPS * 60 sec/min}$$

The more widely used definition of efficiency is the VS score version, which accounts for downstacking as well. High efficiency players generally have an efficiency of more than 100, with the more notable players to frequently go over this score including names such as CZSMALL0402, QMK, DIAO, THOMASTIK, and the mysterious MAJIMEDESU. One common recurring trait of these players during the period of time that they surpass the 100VS/PPS "golden standard" of efficiency is that they are generally playing slower. Even the players in the lower ranks who have achieved this "golden standard" are almost certainly playing slower than the average in their player for their rank. This begs the question. What is the cause-effect relationship? Are these players playing slower because they are more efficient? Or are these players more efficient because they are playing slower?

From my perspective, I believe the latter to be a more accurate statement. This is because the slower person has a natural efficiency advantage. Take for example a match between two players, let us call them P1 and P2, where P1 is the normal player and P2 is the slow but efficient player. At the start of the game, P1 sends a TKI while P2 waits. P1 took 1 full bag (7 pieces) to complete the TKI, sending 4 lines over to P2. P2 then downstacks the damage received in the same 1 full bag. After this exchange, the sum of P1's attacks and downstacks are 4 (4+0) while the sum of P2's attacks and downstacks are 8 (4+4). The result is that the sum of P2's attacks and downstacks is higher than P1, and the PPS of P1 is higher than P2, both contributing to P2 having the higher efficiency.

With the example provided, I hope that the reader better understands my opinion that the current definition of Efficiency is not a good measure of how efficient a player is. The slower player in the exchange will naturally have the advantage, which means anyone can become efficient by just playing slower. Personally, I have very frequently went over the 100VS/PPS mark myself, by dropping my average PPS from 2.1 to 1.7 in a skill bracket where the average player is around 2.3 PPS.

With all that information in mind, a question arose in my mind:

"What if you compare a player's PPS against their skill level, and penalize those who play slower than the average player at their skill level and vice versa?"

If the equation of efficiency is modified to factor in the relationship between a player's skill level and their PPS, then we would have a more accurate measure of efficiency.

Glicko

The skill level of a player in Tetr.IO is based on the GLIXARE formula used in the Smogon ladder. It is loosely based on the Glicko-2 formula, which not only determines the odds of a player winning against another player, but also adjusts for rating deviation, which increases the longer a player has not played, adjusting for the diminishing confidence that a player will remain the same skill level as time between matches increases. In general, the larger the difference of Glicko between two players, the more the odds will be stacked in one player's favor. Two players of equal Glicko rating are said to have equal chances of winning.

Regression Analysis

In statistical modelling, regression analysis is a set of machine learning methods used to predict a continuous outcome variable (y) based on the value of one or more predictor variables (x). The general idea of regression analysis is to form an equation with the data already provided, then use the equation to make predictions on unknown data.

In our case, we want to find an equation that can describe the change of PPS with Glicko. The PPS is the outcome variable while the Glicko is a predictor variable. The change of PPS with Glicko can then be used to predict what the expected PPS of a player in that certain Glicko is. For example, while CZSMALL0402 has a PPS of 2.34, by logic we would know that an opponent of their skill level should be playing at 3.5PPS or above. Hence, we can conclude that CZSMALL0402 is playing significantly slower than their opponent, which would put them in an unfair advantage in terms of efficiency.

To gather some information of what kind of regression to use, we first plot the relationship between Glicko and PPS in RStudio using the package ggplot(), the scatterplot shown in the figure below shows the PPS and Glicko of every single player in Tetra League in a single point in time.

The code for getting a single snapshot is written in Python for reusability's sake as there are plans to write a script that makes hourly API calls to the Tetr.IO API and hence allowing for one to expand on the database and no longer require Tenchi to supply the data. The code will dump out a JSON file titled "league.json". The code is shown below and can alternatively be downloaded from the GitHub page under "importSingleSnapshot.py"

```(Python)
import requests
import json
res =
requests.get('https://ch.tetr.io/api/users/lists/league/all').json()
with open('league.json', 'w') as json_file:
    json.dump(res, json_file)
```

After an initial plotting of the graph, one very obvious issue arises. There two types of very obvious outliers which will cause problems with regression modelling. There are several players

with negative Glicko as a result of intentionally losing games repeatedly in order to achieve as low of a rating as possible. On top of that, there are also several players who have unrealistic PPS. For example, there are several data points in the C and D ranks with over 2 PPS. Upon manual inspection of these players, it can be inferred that these players are not playing seriously, rather they are placing blocks with little regard of winning the game and are more concerned about appearing as an outlier. A snapshot of the data obtained from April 4th 2022 at 3pm (+8GMT) is shown below, along with the "curve of best fit" plotted.

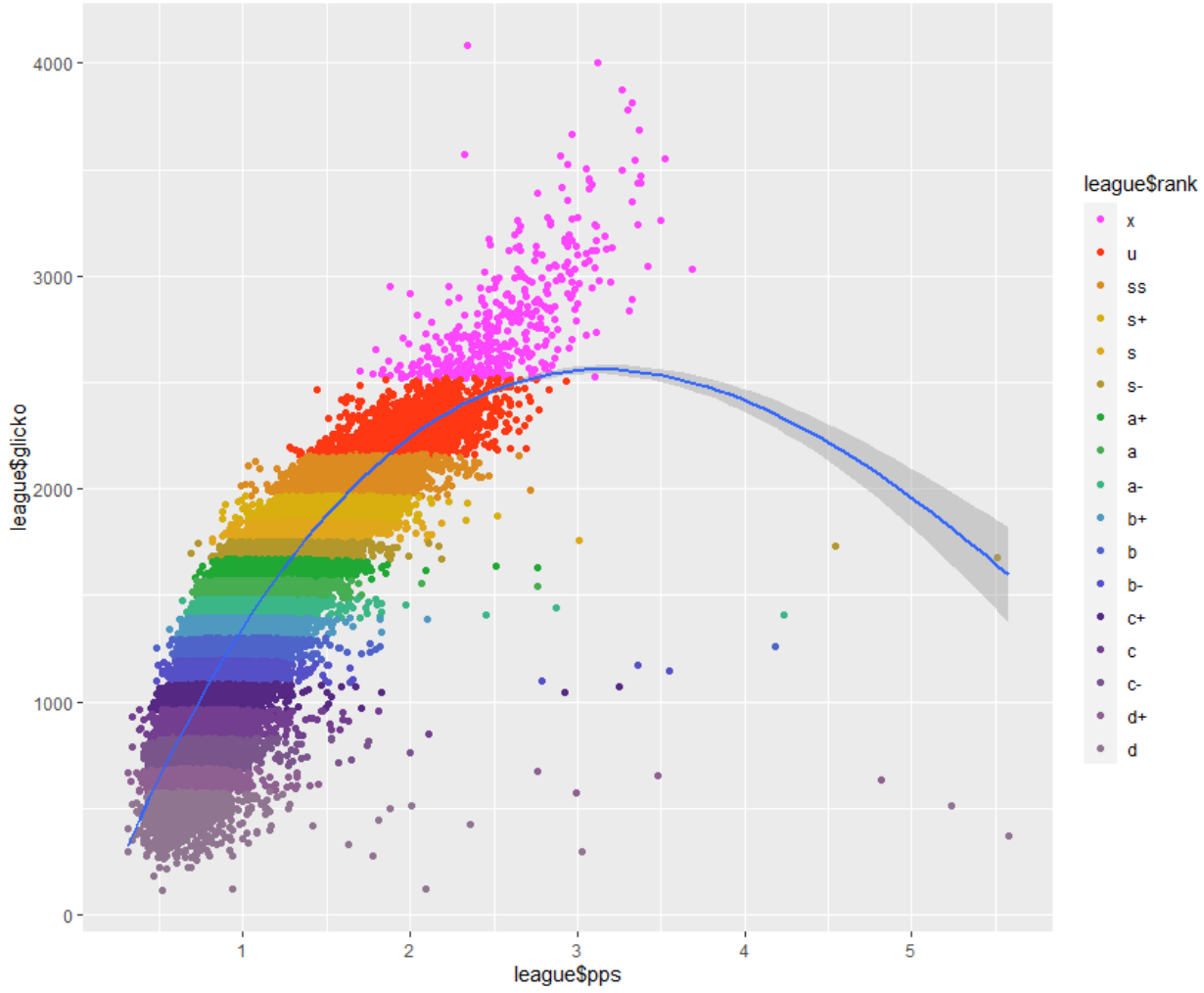

Figure 3: Scatterplot showing relationship between Glicko and PPS (outliers not removed)

Two filters will be added. One will filter out any data point where the Glicko rating is below 100. This will be used to weed out players who are intentionally losing to get as low as possible of a Glicko score. It is understood that filtering out players under 100 Glicko might exclude some legitimate players in that skill bracket, but it can be assumed with minimal loss that players at that skill level are not concerned with their efficiency. Another filter will filter out any data point where the PPS is six standard deviations away from the mean of the rank in question. The R code

for filtering and plotting the graph is shown below. The reader may also choose to download the code under the name “plotRecentSnapshot.R” on the GitHub page.

```(R)
library(tidyverse)#Comes with all the basic data analytics packages
library(jsonlite) #This one's fromJSON has flatten=TRUE which allows
to read nested JSON
directory = DIRECTORY ADDRESS HERE
tlData = fromJSON(directory)$data$users
tlData = tlData %>%
  group_by(league$rank) %>%
  filter(!(abs(league$pps - median(league$pps)) >
6*sd(league$pps))) %>%
  filter(!league$glicko < 100)
#Plot the pps~glicko graph with a very loose regression eqn
implemented
ggplot(requiredData, aes(x=pps, y=glicko)) +
geom_point(mapping=aes(x=pps, y=glicko,
color=tl_rank))+scale_color_manual(values=c('x'= '#FF45FF','u'=
'#FF3813','ss'= '#DB8B1F','s+'= '#D8AF0E','s'= '#E0A71B', 's-'=
'#B2972B', 'a+'= '#1FA834','a'= '#46AD51','a-'= '#3BB687','b+'=
'#4F99C0','b'= '#4F64C9','b-'= '#5650C7','c+'= '#552883','c'=
'#733E8F','c-'= '#79558C','d+'= '#8E6091','d'= '#907591'))+
stat_smooth(method="lm", formula= y ~ poly(x, degree=3, raw=TRUE))
```

The reason a 3rd degree polynomial was selected is from an estimation of what the trend curve would look like. It was then decided that the curve of a 3rd order polynomial fits the visualized trend curve best. If there are any suggestions for different formulas and improvements, please do not hesitate to post a GitHub issue or contact Lim Guowei#2465 on Discord.

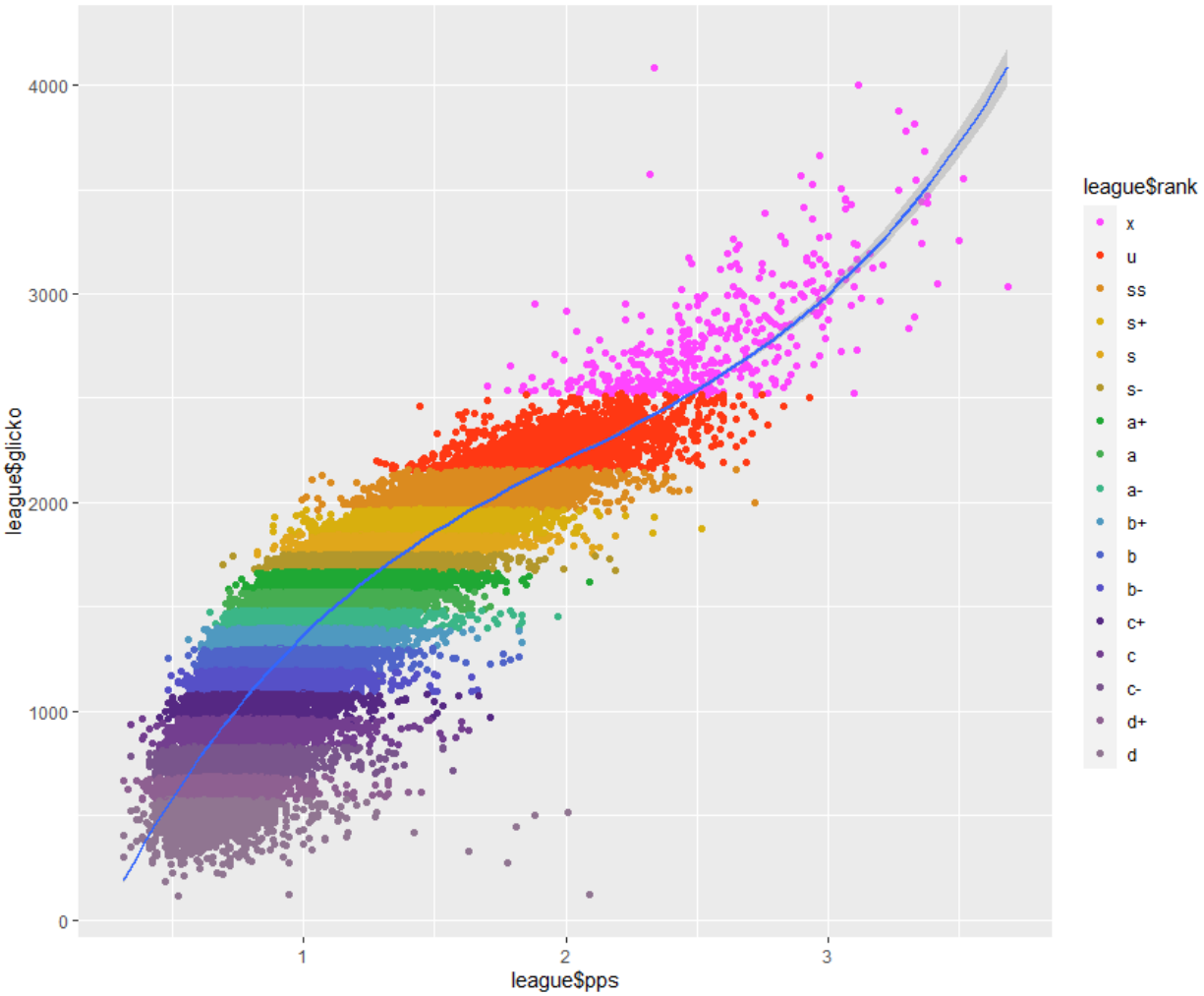

Figure 4: Scatterplot showing the relationship between Glicko and PPS (outliers removed)

In order to get a regression model for every single snapshot in time, a cost benefit analysis is done. Is it better to maximize the accuracy of the regression models by performing regression analysis on every single snapshot, or should compromises be made and use one regression model for every 10 snapshots?

To get an estimate on how long it would take, the last 10 folders are separated, which are also the 10 biggest snapshots, into a separate folder and run the code below. The code is also available to download from GitHub under “writeRegression.R”

```(R)
library(RMySQL)#Required for accessing your MySQL DB as reading the created 1.2GB CSV file is too hard on my laptop
library(tidyverse)#Comes with all the basic data analytics packages
library(jsonlite) #This one's fromJSON has flatten=TRUE which allows to read nested JSON
library(lubridate)
```

```
library(dplyr)
#Connect to DB

con = dbConnect(RMySQL::MySQL(),
                dbname="tl_delta_db",
                host="localhost",
                port=3306,
                user="root",
                password=INSERT PASSWORD HERE)
#Identify files to read in
rootDir = "D:\\tetrio-stats"
globalStartTime = Sys.time()
foldersToProcess = list.files(path=rootDir) #For entire folder
readFileInside = function(folderName){

if(file.info(paste(rootDir,folderName,"league.json",sep="\\"))$size>20
00){
    startTime = Sys.time()
    tlData = fromJSON(txt =
paste(rootDir,folderName,"league.json",sep="\\"))$users
    tlData = tlData %>%
      group_by(league$rank) %>%
      filter(!(abs(league$pps - median(league$pps)) >
6*sd(league$pps))) %>%
      filter(!league$glicko < 100)
    dataRegression = lm(league$pps ~ poly(league$glicko, degree=3,
raw=TRUE), data=tlData)
    coefficients = unname(coef(dataRegression))
variables=paste(coefficients[1],coefficients[2],coefficients[3],coeffi
cients[4], sep = ",")
    query=paste("INSERT INTO tl_time_series_regression VALUES (\"",
ymd_hms(folderName),"\",", variables, ")")
    a = dbSendQuery(con,query)
    endTime = Sys.time()
    print(endTime-startTime)
  }else{
    print(folderName)
    print("File size too small")
  }
}
lapply(X = foldersToProcess, FUN=readFileInside)
globalEndTime = Sys.time()
print(globalEndTime-globalStartTime)
```

The last 10 folders took approximately 1 second each. Which would mean that an upper limit of the entire process would take less than 4.5 hours. The accuracy of having a unique regression model for every single snapshot is worth the wait. The entire process ended up taking 2.56 hours.

Data Aggregation Part 2

As it has been established that new calculated data is required to draw conclusions about efficiency, the previous delta table that was created and populated in Phase 3 is no longer sufficient. Hence, a new table with the calculated values is created. The columns and their respective information are listed below. New columns are highlighted in yellow.

Table 4: New Table of data and their respective data types.

_id	CHAR(24). All IDs are 24 characters long
username	CHAR(16). All usernames in Tetr.IO are at most 16 characters long
verified	BOOLEAN
apm	DECIMAL(5,2). Data provided is only accurate to 2 decimal places
pps	DECIMAL(5,2). Data provided is only accurate to 2 decimal places
vs	DECIMAL(5,2). Data provided is only accurate to 2 decimal places
tr	DOUBLE. Used to be an integer but decimal places were added later on
tl_rank	Stores the rank variable from the previous table. Name changed to avoid problems down the line with rank() functions from the R programming language
glicko	DOUBLE. Used to be an integer but decimal places were added later on
rd	TINYINT unsigned. Has many decimal places but since RD does not determine a player's skill, rounded to nearest integer and unsigned as it can only go between 60-100
games_played	SMALLINT unsigned with highest number of 65535. Estimated time of hitting this number is mid to late 2030 by stqrm. Consider changing to INT which accounts for numbers up till 4 billion
games_won	SMALLINT unsigned with highest number of 65535. Estimated time of hitting this number is mid to late 2030 by stqrm. Consider changing to INT which accounts for numbers up till 4 billion
delta_date	DATETIME
raw_eff	FLOAT to allow for more decimal places than DECIMAL
adjusted_eff	FLOAT to allow for more decimal places than DECIMAL
raw_app	FLOAT to allow for more decimal places than DECIMAL
adjusted_app	FLOAT to allow for more decimal places than DECIMAL

The new data will use the formulas listed in the table below

Table 5: Table of formulas

raw_eff	$eff_{raw} = \frac{VS}{PPS}$

adjusted_eff	$eff_{adjusted} = \frac{2}{3} * \frac{VS}{PPS} + \frac{1}{3} * \frac{VS}{PPS} * \frac{PPS}{Regression\ Model}$
raw_app	$app_{raw} = \frac{APM}{PPS * 60}$
adjusted_app	$app_{adjusted} = \frac{2}{3} * \frac{APM}{PPS * 60} + \frac{1}{3} * \frac{APM}{PPS * 60} * \frac{PPS}{Regression\ Model}$

The code for creation of the delta table and population of the table can be found on https://github.com/limgw1/google-data-analytics-capstone under the names "createDeltaTable.sql" and "importFullTable.py".

Phase 5: Share

To visualize the results, we will take the historical data of the current best player in the world, CZSMALL0402. As a player, he is known as the king of macroplay, which describes all the non-mechanical skills of the game such as timing, understanding when to accept garbage and when to cancel garbage, board health and momentum, which indirectly translates to higher efficiency. In recent history, he has remained undefeated in any tournament he participates in, except for one tournament organized on the platform Jstris where he lost to FIRESTORM, who is known to be the best Jstris player as well as one of the best players in the world. In Tetr.IO however, CZSMALL0402 remains the undisputed champion.

Another reason for selecting CZSMALL0402 is because he has not changed his username. Which makes it easy to get the data by username rather than by userID.

Raw Efficiency vs Adjusted Efficiency

In order compare raw efficiency and adjusted efficiency, we make two plots. On one plot, raw efficiency and adjusted efficiency are plotted on the same graph over time. On the other plot, the PPS over time is plotted, along with expected PPS. The plot is created in R using the ggplot() command. The resulting R (and a little bit of SQL) code is shown below. Alternatively, it can be downloaded from the GitHub repository under the name “historicalPlayerData.R”

```(R)
library(RMySQL)#Required for accessing your MySQL DB as reading the
created 1.2GB CSV file is too hard on my laptop
library(tidyverse)#Comes with all the basic data analytics packages
library(gridExtra)
#Connect to DB
con = dbConnect(RMySQL::MySQL(),
                dbname="tl_delta_db",
                host="localhost",
                port=3306,
                user="root",
                password="Insert your password here")
#Example: Getting historical TL data of player
playerHistorical = dbSendQuery(con, "SELECT * FROM tl_delta_table
INNER JOIN tl_time_series_regression
     ON tl_delta_table.delta_date = tl_time_series_regression.currTime
WHERE username='czsmall0402'")
playerTable = dbFetch(playerHistorical, n=100000)
relevantData =
playerTable[,c("delta_date","raw_eff","adjusted_eff","pps","glicko")]
relevantData["expected_pps"] = playerTable['zero_order'] +
playerTable['first_order'] * playerTable['glicko'] +
playerTable['second_order'] * playerTable['glicko']^2 +
playerTable['third_order'] * playerTable['glicko']^3
```

```
#Plot the graphs
plot1 = ggplot(data=relevantData, aes(x=delta_date, group = 1))+
        geom_line(aes(y=pps, color="PPS"), size=1)+
        geom_line(aes(y=expected_pps, color="Expected PPS"), size=1)+
        scale_color_manual(values=c("PPS"="Orange", "Expected
PPS"="Purple"))+
theme(axis.text.x=element_blank(),axis.title.y=element_blank(),legend.
title=element_blank(),legend.position="top")

plot2 = ggplot(data=relevantData, aes(x=delta_date, group = 2),
color=variable)+
        geom_line(aes(y=adjusted_eff, color="Adjusted Efficiency"),
size=1)+
        geom_line(aes(y=raw_eff, color="Raw Efficiency"), size=1)+
        scale_color_manual(values=c("Adjusted Efficiency"="Red", "Raw
Efficiency"="Green"))+
theme(axis.title.x=element_blank(),axis.text.x=element_blank(),axis.ti
tle.y=element_blank(),legend.title=element_blank(),legend.position="to
p")

grid.arrange(plot1, plot2, layout_matrix=matrix(c(2,2,1), nrow=3))
```

The resulting plot for CZSMALL0402 will look like this when the code is run. Note that the VS Score plots do not start at the very beginning as VS Score was implemented several months after the release of Tetra League.

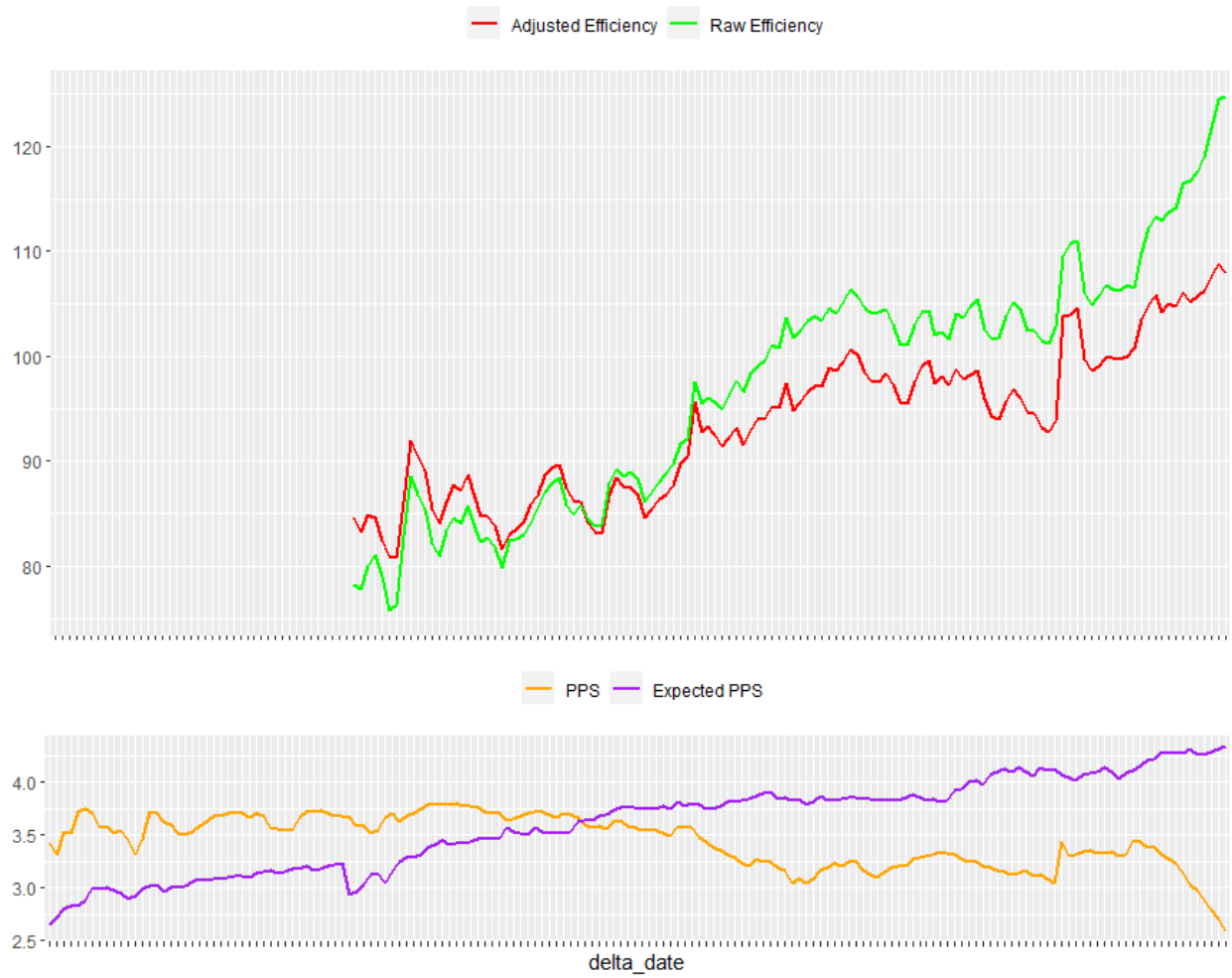

Figure 5: Two plots comparing the Adjusted Efficiency and Raw Efficiency, as well as the PPS and expected PPS of CZSMALL0402

From the data, it can be seen that difference between the raw efficiency and the adjusted efficiency changes according to the difference in PPS and expected PPS. This is in line with the equation that dictates if a player plays faster than an opponent of similar skill level, they are at a raw efficiency disadvantage. The crossover point where the adjusted efficiency and raw efficiency meet is also in line with the point where the player's PPS and the expected PPS meet. A small caveat is that the earlier data might be less accurate than desired, considering that the player in question might still be climbing the ranks and not really playing with people of equal skill level. The accuracy of this chart increases with total number of games played, as a player with more games played is more likely to have already reached their actual ranking in the leaderboard.

As the regression model changes day by day, getting the most efficient player is not as simple as sorting by adjusted efficiency in descending order. In reality, the number itself bears no real meaning without being compared to other players. There are two methods to do this type of comparison. The first method is to get the latest snapshot, perform the data cleaning and populate a new smaller SQL table just for the specific snapshot. The second method will be to

plot the historical adjusted efficiency of at least two players and compare them that way. We will be displaying both methods.

Method 1: Single Snapshot (scatterplot)

For method 1, we will be using the latest snapshot saved at the time of writing this, which is 3rd April 2022 8PM (+8GMT). To obtain the JSON file, either use the methods in Phase 2. The JSON file is loaded in R, the additional required columns are added, and the saved into a comma separated values (CSV) file, which is then loaded to Tableau as Tableau has the ability to show more details of every single point in the scatterplot. The code for the entire process can be found on the GitHub repo under the name "plotRecentSnapshot.R". The image below is a screenshot of the Tableau sheet, which can be accessed from Tableau Public at the website https://public.tableau.com/app/profile/lim.guowei/viz/GlickovsAdjustedEfficiencyonasinglesnapshot/Sheet1. I highly recommend going to the website instead of looking at the snapshot below, as you are able to mouse over each data point to see more information like username and value of adjusted efficiency.

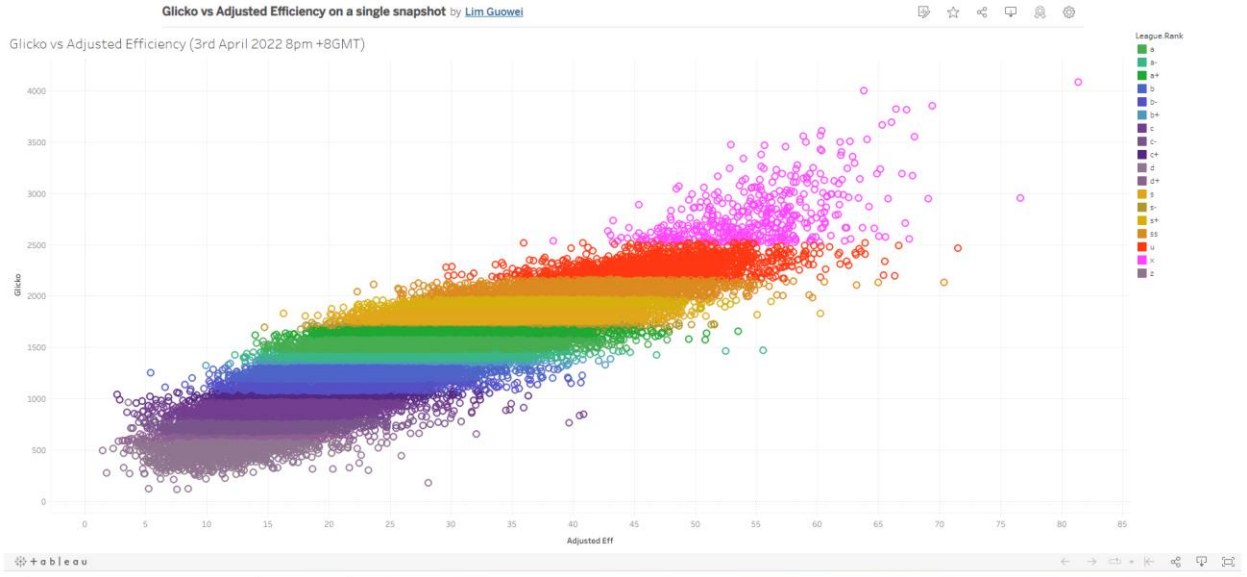

Figure 6: Tableau Visualization of the relationship between Glicko and Adjusted Efficiency

Key takeaways from this data visualization:

- CZSMALL0402 remains the most efficient player, despite playing at a significantly slower speed.
- The higher the Glicko, the higher the Adjusted Efficiency. This indicates the importance of working on efficiency on top of just improving speed to climb the ladder.
- At X rank, the slope of Glicko to Adjusted Efficiency increases. Indicating less importance on improving efficiency to increase from a lower X rank to a higher X rank compared to all ranks below X.

- From the graph shown, I will personally recommend watching CZSMALL0402, as their efficiency, even when speed adjusted, appears to be well beyond even the historically efficient players like QMK, FIRESTORM, KAZU, VINCEHD.
- There is a notable player who is not on my radar as an efficient high skilled player, QUARPI01, I intend to observe their gameplay to learn their techniques.

A similar scatterplot visualizing the relationship between Glicko and Adjusted APP can be seen in the snapshot below or accessed from https://public.tableau.com/app/profile/lim.guowei/viz/GlickovsAdjustedEfficiencyonasinglesnapshot/APP?publish=yes

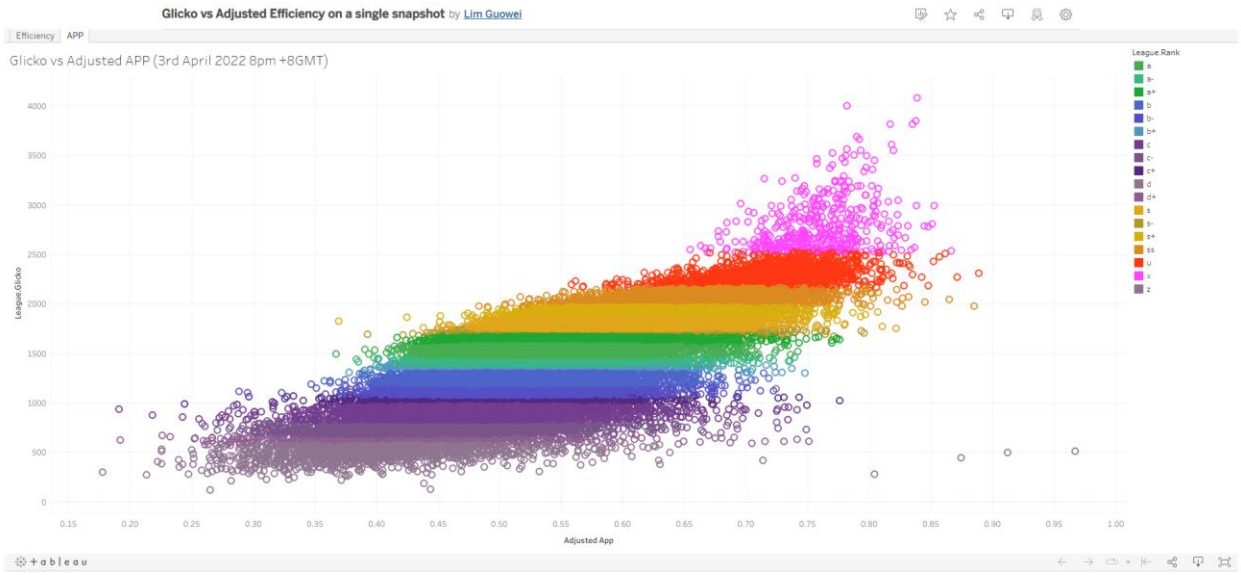

Figure 7: Tableau visualization of the relationship between Glicko and Adjusted APP

Key takeaways from this data visualization:

- Adjusted APP heavily favors the opener mains, defined by the community as players who are extremely fast for the first few seconds and slow down once the memorized opener sequence is over.
- The opener mains do not have a very high adjusted efficiency to adjusted APM ratio, indicating that they are less likely to downstack, either by virtue of the game ending at the very start or from lower tendency to downstack and instead trying to out-pressure the opponent.
- Positive trend is still observed. Higher adjusted APP players have higher ranks and Glicko rating in general.

- There appears to be a more prominent “wall” at around 0.8 adjusted APP, suggesting that players need not focus on increasing adjusted APP once that threshold is met and instead should focus on other aspects of skill.
- Due to the low average PPS of the D and D+ ranks, some players appear to have extremely high adjusted APP by virtue of having higher PPS. These players should be ignored in the context of finding a truly “offensively efficient” player.
- From the graph shown, I will personally recommend watching the following players to learn about being “offensively efficient” as they are of extremely high Glicko and do not rely on openers and short games to maintain their adjusted APP: FIRESTORM, KAZU, CZSMALL0402.

Method 2: Historical Data

For Method 2, we will be comparing FIRESTORM and QMK as they have appeared as rivals for many years in the community, with QMK being the only person to have defeated FIRESTORM in a Jstris tournament during their peak era. FIRESTORM is very well known for his ability to maintain offensive pressure while QMK is very well known for his ability to maintain strong defenses. Comparing them would be analogous to asking: “What is more powerful, the spear that can pierce through anything, or the shield that can block anything?” The code to generate the plot below can be downloaded from the GitHub repo under “comparePlayers.R”

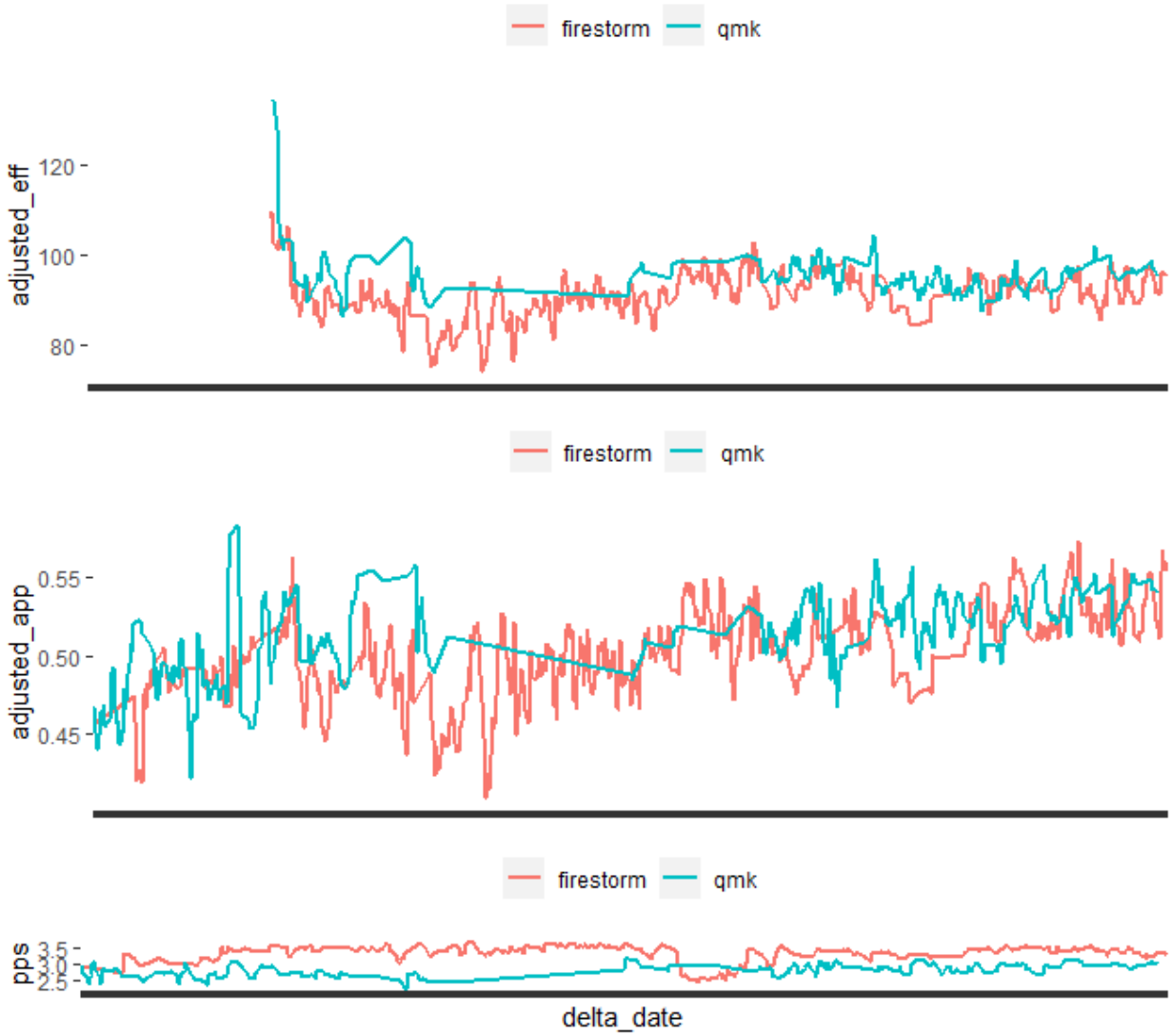

Figure 8: Top plot – Comparison of Adjusted Efficiency between FIRESTORM and QMK, Middle plot – Comparison of Adjusted APP between FIRESTORM and QMK, Bottom plot – Comparison of PPS between FIRESTORM and QMK

Before concluding key takeaways, there is an abnormally high adjusted efficiency score in the early days of the implementation of VS score. To investigate further, I went on to Tableau to check and see if VS scores are indeed higher during this period of time, or is it just a coincidence that both FIRESTORM and QMK were absurdly efficient at the time. The snapshot below shows the change of VS score of every single player in Tetra League over time. Do not attempt to identify each individual player but try to see the overall trend.

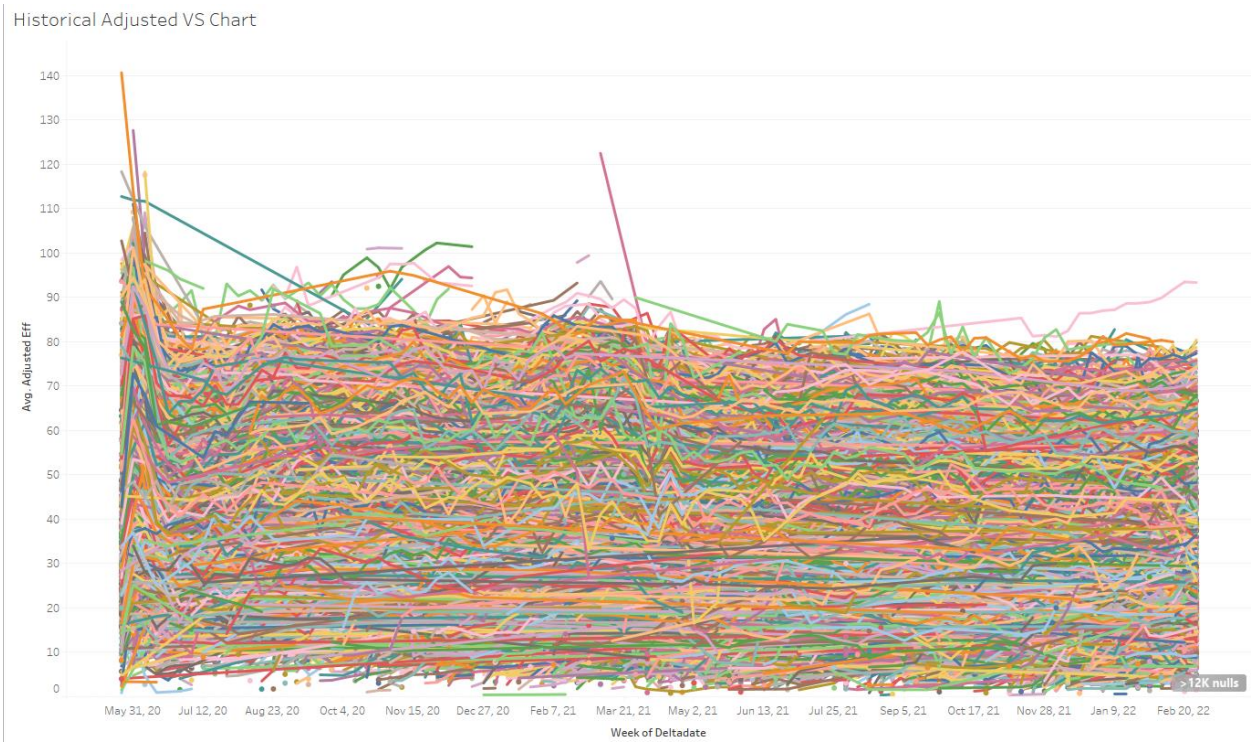

Figure 9: Tableau Visualization of the historical plot of weekly Adjusted Efficiency of every single player in the history of Tetra League.

From the above snapshot, we can see that there is indeed an unusual spike in adjusted VS Score in the beginning. However, when taking the monthly average, the data paints a different picture as can be seen in the figure below. This indicates that only a small fraction of the player base experienced a spike in efficiency. Upon asking the developers in the Tetr.IO discord and analyzing some cases manually via SQL, it is concluded that the high VS/PPS for certain people is due to them only having played one or a few matches at the time, and netting a high VS score for that particular match. For instance, QMK's first game with VS score being tracked was 408.8 which boosted their average score, until they played more games and the it eventually even-ed out.

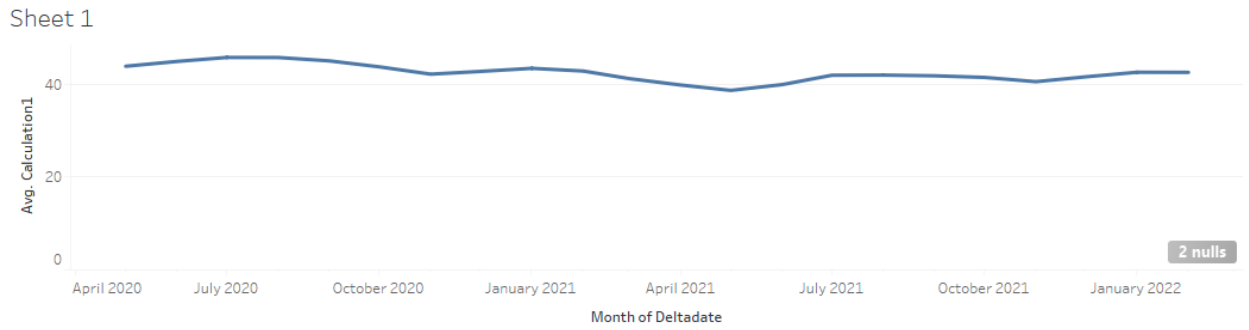

Figure 10: Tableau Visualization of the Average Raw Efficiency of every single player in Tetra League combined over time.

Back to the historical data, of FIRESTORM and QMK, now that we know the source of the massive spike in efficiency for both players, I feel safe in my decision to add another filter to the SQL query,

to only plot the statistics after 15th June 2020, giving them half a month to even out their stats. The resulting plot looks like this.

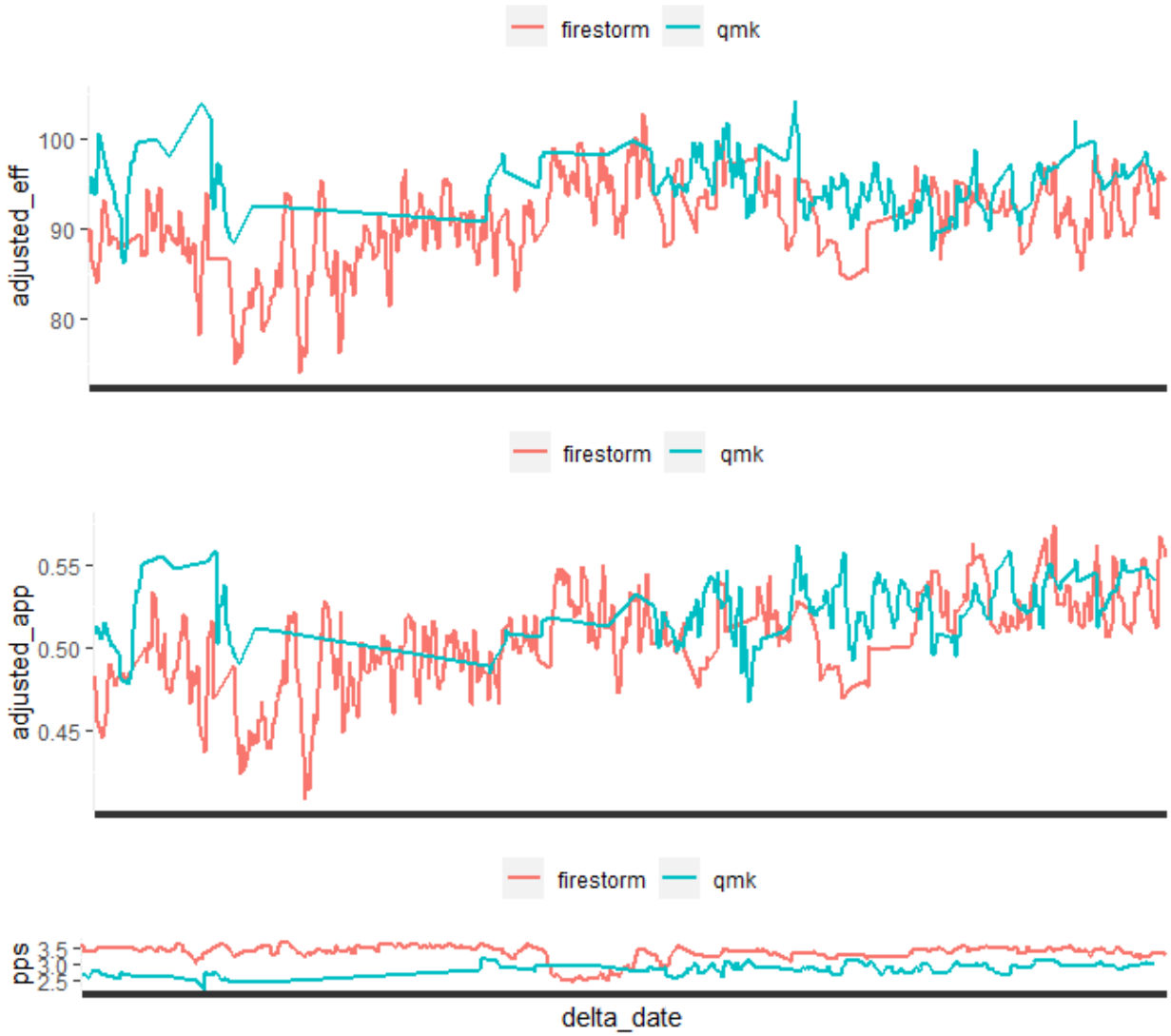

Figure 11: Top plot – Comparison of Adjusted Efficiency between FIRESTORM and QMK, Middle plot – Comparison of Adjusted APP between FIRESTORM and QMK, Bottom plot – Comparison of PPS between FIRESTORM and QMK (removed all stats before 15th June 2020)

Key takeaways from the above plots:

- It appears that in terms of adjusted APP, FIRESTORM and QMK are equally matched.
- However, QMK has a tiny advantage over FIRESTORM in terms of adjusted efficiency.
- A defensively efficient player would not be able to keep up with FIRESTORM at this speed deficiency. Hence it would be incorrect to refer to QMK as a defensive player. A counterspiker would be a more accurate description of QMK's playstyle.

Is Efficiency truly a measure of Efficiency?

If we go by the definition of efficiency established in Phase 4, the short answer is “No” and the long answer is “It depends”.

The short answer is “No” because raw efficiency can be easily manipulated by forcing a speed disadvantage. Personally, I have crossed the 100VS/PPS threshold many times and with ease by simply playing at below 1.7PPS in a skill bracket where the average player plays at 2.3PPS.

The long answer is “It depends” because at the end of the day, not every single player is deliberately slowing down to chase an arbitrary VS/PPS ratio. Comparing the raw efficiency of the entire Tetra League in general is still able to give you a general idea of who is more efficient, and the fact that X ranks are more efficient than U ranks, U ranks are more efficient than SS ranks, and so on. However, CZSMALL0402 with a raw efficiency of around 125 is actually only running at a speed adjusted efficiency of about 81.37 as they are deliberately slowing down. Hence, inferences made from raw efficiency are still usable, but need to be taken with a grain of salt.

One limitation with evaluating efficiency based is the fact that we are assuming that efficiency is determined by the player and the player only. In reality efficiency is also determined by the opponent. Do they send messy or clean garbage? Do they prefer counterspiking over cancelling? Without detailed statistics of the match, it is impossible to account for these factors.

Macroplay

Macroplay, as defined by NOOMOI, the person who coined the term in this context, is the non-mechanical ability of a player – meaning anything that does not have to do with speed, offensive power, defensive power, combo vision, perfect clear vision or T-spin vision. A similar term used which is more commonly heard in other games is “Game Sense”. The most well-known expert of macroplay is CZSMALL0402, who has trained intensively under NOOMOI. As CZSMALL0402 is the current best player in Tetr.IO, and is also deemed by several to be the best guideline Tetris player in history across all platforms, the idea of practicing macroplay has become a frequently discussed topic for the community, regardless of skill level. However, the fact that there is no known metric that can be used to quantify “macro”, there has been no properly defined method to train such a skill. The more commonly discussed aspect of macroplay is something coined by CZSMALL0402 as “Accept Clean Cancel Cheese”. Which means to willingly receive clean garbage, which is easier to downstack and can be turned into offensive power, and to cancel cheese, meaning to send your own attacks to negate incoming cheese garbage or messy garbage, where the holes do not appear in the same column consecutively.

Macroplay vs Efficiency

From the analysis of the data points in https://public.tableau.com/app/profile/lim.guowei/viz/GlickovsAdjustedEfficiencyonasinglesnapshot/Efficiency , it can be seen that CZSMALL0402 is indeed the most efficient player. Other players known for practicing the idea of “Accept Clean Cancel Cheese” to a certain degree like TING704, QMK, GABETHEMAN and to an extent FIRESTORM are also observed to have high

efficiency. This would indicate that high efficiency players tend to be better at macroplay. However, it is impossible to tell by statistic alone the cause-effect relationship of macroplay and efficiency. Does high efficiency result in good macroplay? Or does good macroplay result in high efficiency?

How to quantify macroplay?

From my research and the data that is tracked by Tetr.IO, the best possible metric we have to quantifying macroplay is a speed adjusted efficiency. Until there is a good way to quantify macroplay, the best method we have is to look at players who we determine is “good at macroplay” and observe their stats. In my opinion, speed adjusted efficiency is a more accurate measure of macroplay than raw efficiency, but it is still imperfect.

Phase 6: Act

High-level insights

1. Raw efficiency is not a good measure of efficiency, as it is too susceptible to players abusing the efficiency advantage associated with speed disadvantage
2. Proposed a better measure of efficiency: "Speed Adjusted Efficiency", calculated by $eff_{adjusted} = \frac{2}{3} * \frac{VS}{PPS} + \frac{1}{3} * \frac{VS}{PPS} * \frac{PPS}{Regression\ Model}$
3. Raw APP is not a good measure of offensive efficiency for the same reasons as raw efficiency is not a good measure of efficiency.
4. Adjusted APP can be a decent measure of offensive efficiency, but has the problem of favoring opener mains.
5. Opener mains in general suffer from a downstack deficiency, indicated by the % of their adjusted efficiency being attributed to their adjusted APP
6. There is an observable "skill cap" at 0.8APP
7. CZSMALL0402 is still the most efficient player in Tetr.IO, despite adjusting for speed difference.
8. The higher the Glicko, the higher the Adjusted Efficiency.
9. At X rank, the slope of Glicko to Adjusted Efficiency increases.
10. Offensive players and defensive players of equal skill level generally have the same speed adjusted efficiency.
11. Known defensive players like QMK are more of a counterspike player than a defensive player.
12. The best measure of macroplay in Tetr.IO at the moment is speed adjusted efficiency.
13. It is still unknown if good macroplay causes high efficiency or high efficiency causes good macroplay

Actionable steps

1. For pure efficiency, I recommend watching CZSMALL0402. Notable alternatives include QMK, FIRESTORM, VINCEHD, QUARPI01
2. For offensive efficiency, I recommend watching CZSMALL0402, KAZU, FIRESTORM, VINCEHD
3. If you are able to achieve approximately 0.75APP in normal gameplay, it is recommended to start focusing on improving raw mechanical skills like speed and downstacking as further increasing APP has diminishing returns.
4. Do not chase raw efficiency and raw APP numbers, it may distract you from improving your efficiency and APP properly.
5. Opener mains are recommended to start practicing downstacking as it is the most "bang for your buck" increase to your skill level. My personal recommendation is to deliberately think about all the ways you can stack your opener of choice different to facilitate for good downstacks instead of chasing a PC finish. Alternatively, you can play into the midgame and focus on practicing downstacking defensively and offensively.

6. Consider adjusting play speed, slow players with high raw efficiency might be able to achieve higher adjusted efficiency by increasing their speed slightly and vice versa.
7. To practice one aspect of macroplay, one should deliberately think about their attacks and their opponent's attacks. Consider if you should take the clean incoming damage or use your own attacks to cancel cheese.

Limitations of this case study

- Tetr.IO does not provide in depth statistics of each individual match. If it is viable, I hope that osk will implement statistics that could possibly measure macroplay such as 2 second APM moving average, 2 second PPS moving average, 2 second VS score moving average and the "relative attacking power" over time. I understand that this data is very large, hence I do not think it is feasible for Tetr.IO to save this data.
- I am personally new and still learning in the field of Data Analytics, Data Science and Machine Learning. The ML model I have used is based on the assumption that the graph follows a 3rd order polynomial. Currently I lack the experience to determine if this is the best way to do things. Hence, I will make the necessary changes as I learn more over time.
- On top of that, the Tableau visualizations that I have made are a work in progress as I intend to refine the visualizations as I learn more about the BI tool.
- The idea of having the ratio of player speed over expected speed at that Glicko to only affect 1/3 of the final value is based on my loose idea that the Speed, Opponent's attack and Player's own attack are the 3 main contributors to efficiency. In an ideal reality, I would be able to identify all the factors that contribute to efficiency and quantify their weights accurately.
- Macroplay is still a concept with unknown possibilities. Even experts like NOOMOI and CZSMALL0402 are unsure about the true extent of macroplay. More research needs to be done on the concept of Macroplay itself.

Final words

I sincerely thank everyone who has taken the time to read even a small portion of this report. I would also like to extend my gratitude to CCRed95#6170 for their guidance in conducting this case study, Tenchi for providing the archives of Tetr.IO's historical data, my real-life friends who have helped me with some programming problems as well as allowing me to bounce ideas off of them, and ZaptorZap for answering my questions about the inner workings of Tetr.IO. I hope that the findings of this case study will spark new discussion in the Tetris community in the right direction, the direction of improvement. If you have any further inquiries or would like to suggest changes, please do not hesitate to contact me at Lim Guowei#2465.