CTL Dashboard v2 Documentation

Tetris at Berkeley CTL Team

22 March 2024

Contents

1 Win Probability

The rest of the prediction tools rely on accurate *win probabilities*, your estimates of the probability that your player wins in each of the $5^2 = 25$ possible 1v1 matchups between the teams. These probabilities are first estimated with Glicko-2 but can then be adjusted at your discretion.

1.1 Glicko-2 Probabilities

For the default `Win Probability Table (Glicko-2)`, we use the E function from the Glicko-2 system based on players' most recent TETRA LEAGUE (TL) Glicko rating (often with small manual adjustments to smooth fluctuations.)

s_i represents outcomes of 1v1's with the following encodings:

1. P1 wins = 1
2. Draw = 0.5
3. P1 loses = 0.

We make one simplification: Instead of also importing both players' rating deviation (RD) values, we just use a constant RD of $RD_0 \coloneqq 61$. We justify this by computing that the relative error ϵ in $g(RD_i)$ between the RD_i values of 61 and 100 is only $\epsilon \approx 3\%$.

This now lets us use a constant

$$\begin{aligned} g_0 &\coloneqq g(RD_0) \\ &= \frac{1}{\sqrt{1 + \frac{3\left(\frac{\ln 10}{400} \cdot 61\right)^2}{\pi^2}}} \\ &\approx 0.9818. \end{aligned}$$

This is implemented in the spreadsheet's named function `G_0()`.

Substituting into the original E function, we simplify it to a function of only the exact Glicko ratings r_0 and r_1:

$$E(s|r_0, r_1) = \frac{1}{1 + 10^{-\frac{g_0(r_0 - r_1)}{400}}}. \tag{1}$$

This is implemented in the named function `GLICKOS_TO_S(rating1, rating2)`.

1.2 Final Probabilities

For actual matches, to decide on the `Win Probablity Table (Final)`, we also make some manual adjustments. By conducting replay analysis of players' TL history for specific 1v1 matchups, we make decisions about whose playstyles are likely to be better/worse against their opponent than their Glicko alone predicts.

2 Engine

2.1 Main Algorithm

We'll use the terms N and P from game theory to refer to the next team and previous (or next next) team to pick, or "move" in a normal 2-player game like chess. We'll also use the term "pool" to refer to the players available for a team to use later, which excludes players already sent in previous sets, and possibly a player already sent for this set.

The engine treats the counterpick system much like a chess engine would treat a chess game, in the sense that it assumes one team makes a pick first, and then the other team makes a counterpick knowing this information. This is needed to keep the math precise. However, there is a difference in that because either team can have the counterpick, sometimes the "turn" order will be more like 1, 2, 2, 1, instead of 1, 2, 1, 2.

The algorithm considers at each game state, for all players in team N's pool, the expected value of the **set score** (number of sets won by Team 1) if that player were to be picked next. The distribution of possible outcomes for each set is quantified using a win probability table passed as an input to the engine. For each game state, a nonnegative real number is computed, quantifying the state's score. Team 1 will want to maximize the score, while Team 2 will want to minimize this. From here, we can build a full recursion tree.

If, at the start of a set, one team's pool is exhausted, but the other team still has k players, the other team is considered to win the k remaining sets.

2.2 Assumptions

There are several important assumptions this model makes in order to have a mathematically precise quantification of the game. If your team decides to deviate from the model, it'll likely be because you believe flaws in these assumptions are too big.

1. **Set scores dominate**: Glicko ratings are only based on set wins in TL, so we assume there just isn't much additional info to go off to determine the number of individual points won. The engine assumes we just want to optimize team N's score, and that nuances of point values are adequately accounted for by fractional computations from the expected values.
2. **Opponent's wins are negligible**: This model only tries to optimize the score of team N, which does not necessarily make the score of team P worse. So this will not consider any long-term tiebreaker advantage over other teams in the league.
3. **No throwing**: We assume all players try their best to win within each individual set (as opposed to intentionally losing the set near tiebreaker to get counterpick for the next set.) This should be a fairly sound assumption, especially with CTL 2024 rules, where sacrificing the last point, even at 6-6, would mean sacrificing a point difference of 4 after including the extra point ($6 - 8 = -2$, which is 4 less than $8 - 6 = 2$.)
4. **The first set is turn-based**: Because the model has no way of representing both teams picking simultaneously for the first set, it just assumes some team has to go

first. To get around this, there are a few approaches, including but not limited to:

(a) *Optimistic*: Assume you know which player the other team picks first, and fix them as the opponent (as if you were counterpicking against them). In reality, this is probably close to what you would be doing anyways, except you'd consider a few possible starting picks. So one good strategy might be to just test all 5 possible starting picks from the opposing team, comparing the engine recommendations, and then making your own call.

(b) *Pessimistic*: Assume the other team has the advantage because they have the counterpick for set 1. To do this, set your team as moving first with the opponent unknown.

2.3 Performance

The engine should be reasonably performant since it is coded in Google Apps Script (a variant of JavaScript), not normal Google Sheets.

The search tree is maximized when all 5 players are present and the pessimistic strategy is used, forcing the engine to start recursing from the very first pick. But even in this case, the engine reaches a depth of 10 and processes 1014656 nodes in only about 1 second on a fast laptop.

3 Tutorial

Cells with a light yellow background are editable, but you should still follow existing data validation rules (ex. if there's a dropdown, don't delete that.) In general, you shouldn't modify any of the underlying infrastructure (one exception is mentioned below.) But you may want to hover/click on some UI elements, like the bubble chart.

3.1 Pre-Match Setup

1. The first table to edit is the big game state table in the top-left corner. This supplies all the information about the situation before a specific turn that the model needs. (The Glicko values are technically optional; they just enable computing the initial Glicko win probability table.)
   (a) For each player, enter their name and Glicko. The checkbox next to each player indicates whether they're "available", meaning they can still be picked. Note that players already sent, **including any player designated "opponent" for this set**, CANNOT be marked available.
   (b) The turn (1 or 2) describes which team picks the next player.
   (c) The opponent is the name of the player sent by the team that picks first, if team N has counterpick for the set. If team N is picking first, the opponent should be the empty string.
2. The other table to edit is the Win Probability Table (Final). It should store the probability that each player from Team 1 wins the set against each player from Team 2. By default, this just copies over the estimates from Win Probability Table (Glicko-2). But if you think you have more info (ex. you know the playstyles of the players), you can change some or all probabilities to your own values.

3.2 During the Match

After getting some new information, like the other team choosing a pick or a set finishing, you can update the game state:

1. Uncheck the player(s) that just played.
2. Update the turn.
3. Update the opponent.
4. The engine will run an analysis, but **remember that the eval scores may be decreased overall** because fewer sets are left in the match.
   (a) If you're thinking from the perspective of Team 1, you want to **maximize** the eval.
   (b) If you're thinking from the perspective of Team 2, you want to **minimize** the eval.

3.3 Engine Troubleshooting

The engine output calls the function `genAnalysis` in cell `B20`; the function is special because it's a custom function coded with Google Apps Script. Everything else in the dashboard uses regular Google Sheets functionality. So there are a few technical quirks to watch out for:

1. Sometimes, after updating a yellow cell, the function output gets stuck on "Loading..." This is not due to the engine being slow (when it works, the engine consistently runs within about 1 second). Here are two simple options, one of which will usually make it update again:
   (a) Delete the formula, wait for the sheet to save, and then put it back.
   (b) Refresh the page.
2. Each call of a custom function makes a separate request to the Apps Script server, which occasionally causes availability issues, like the sheet not loading at all for a few minutes. To avoid this, you may want to avoid overloading the engine.