УДК 004.421.4

TEAM BASED PLAYER RANKING ALGORITHMS

DOI 10 36994 / 2788- 5518- 2022-02-04-22

Sergii Suglobov. *Open International University of Human Development "Ukraine",Kyiv, Ukraine. s.suglobov@gmail.com*

Anatolii Tymoshenko, *Ph.D., Ass. Prof. Open International University of Human Development "Ukraine",Kyiv,Ukraine. timoshag@i.ua*

Abstract: *The article considers algorithms for assessing the personal capabilities of players that are suitable for multiplayer online games with teams. Analyzed popular algorithms and proposed changes to adapt them work with teams of multiple players.*

Key words*: rating system, matchmaking*

АЛГОРИТМ ПІДРАХУНКУ РЕЙТИНГА ГРАВЦІВ В КОМАНДАХ

Суглобов С.В*. Відкритий міжнародний університет розвитку людини «Україна», Київ, Україна. s.suglobov@gmail.com*

Тимошенко А.Г., *к.т.н.., доц. Відкритий міжнародний університет розвитку людини «Україна», Київ, Україна.*

Анотація*: У статті розглянуто алгоритми оцінки особистих можливостей / рейтингу гравців, які підходять для багатокористувацьких онлайн-ігор з командами. Проаналізовані популярні алгоритми та запропоновані зміни (для алгоритмів що не підтримують команди), щоб адаптувати їх для роботи з командами з кількох гравців. Зроблено акцент саме на використанні алгоритмів для командних ігор. Алгоритми розглядаються по часу їх створення, так як кожен наступний алгоритм це покращена версія попереднього. Але при цьому кожен із цих алгоритмів використовується в сучасних іграх так як всі вони мають свої переваги. Рейтингові алгоритми в своєму загалі використовують два показники щоб оцінити гравця, рейтинг – це реальний рейтинг гравця котрий змінюється після кожної гри, та сігма – це відхилення від рейтингу яке показує наскільки ми можемо вірити в точність рейтингу, чим менше значення сігма тим точніше рейтинг гравця. Рейтинг змінюється після кожного матча в залежності від результата, якщо виграш то збільшується, в разі програшу то знижується. Сігма ж зменшуватися з кількістю ігор котрі гравець зіграв, тим самим підтверджувати реальний рейтинг гравця, або збільшуватися але з часом, тобто з часом зростає неточність рейтингу гравця. Для урахування рейтингу системою підбору зазвичай зручно мати представлення про рейтинг одним числом, в випадку якщо рейтинг представлено двома числами, тоді його обчислюють як рейтинг – 3* сігма. Таким чином беруть сильно песимістичне представлення про рейтинг гравця. Представлена порівняльна таблиця розглянутих алгоритмів.*

Ключові слова: *рейтингова система, пошук партнерів*

Introduction

Why do we need player ranking in the game?

- To make game fair enough for players which are added to one game session. Teams should have close summary ranking to each other. Close - is not mean that they should be identical, no, but to some extend so we can calculate that they have at least some chances of winning.
- To ensure teams balance in game session all chosen players for the game session splitting +/- equally on teams, that's also should be done based on each player ranking.

Ranking of a player should be corresponding to player performance in game or his real skills. Skill is tricky especially in shooter games when it depends on player reaction, practice, play time and

even some bit of luck. But still, skill can be measure as rough approximation of player abilities. What we really interesting is who's better than whom in our particular game. If a player has a higher rating than we expert him to win against player with lower rating. That give us a scale where every player has a rough approximate number which we can use for comparison.

Player ranking and they intersection can be visualized using Gaussian curve when every player can be described by two values: mean (rating), that representing skill difference and standard deviation, that representing probability of difference. If players intersect within borders of their respective deviations than we can say that they are compatible to play together (fig.1).

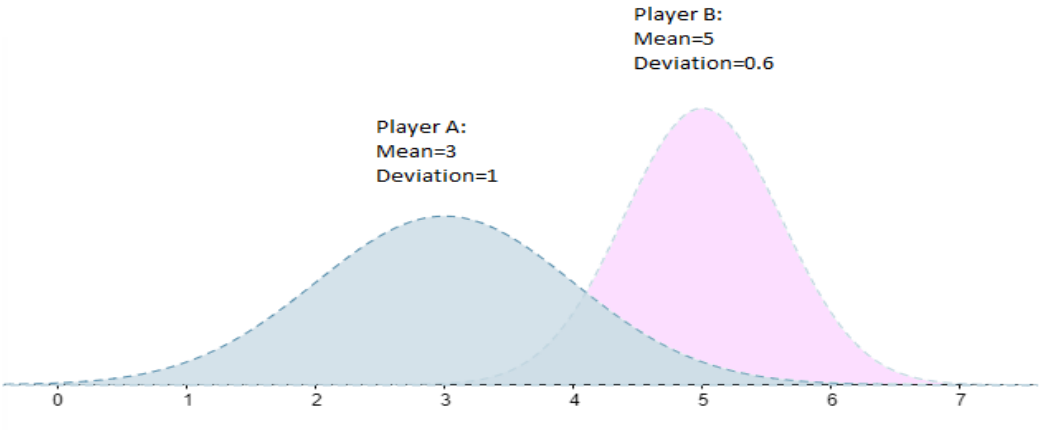


Fig. 1. Illustration of two player ratings

Goal of ranking algorithms is to give players a fun and enjoyable game when he have a challenge and some probability of winning. Knowing player skills, we can predict rough probability of winning in the match. Good metric here can be used a 50% win rate in total number of matches.

Simple solution that also can be used for some purpose.

Research

Win – Lose. Every of presented algorithms use game outcome as main parameter for creating player rating. Simplest way to differentiate player ranking based on win/lose outcome of game session can be calculated:

$$R_{new} = R_A + S_T \ , \tag{1}$$

S_T – it is a match result of player team, win counts as 1, a loss as -1 and draw as 0**.**

Matching player among each other can be done by determining allowed distance between they ranks as 200 (value should be based on statistic) when probability of winning is not less than 60%(fig.2).

Pros:

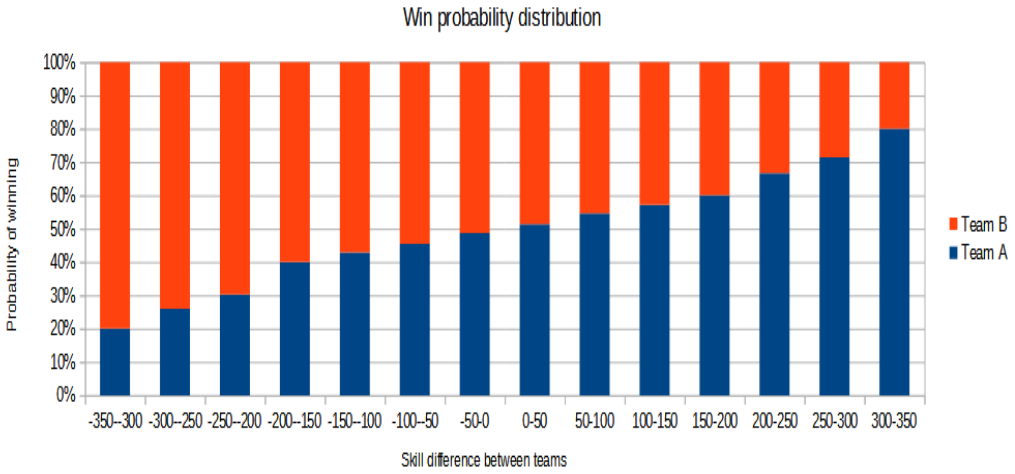
- Easy to implement that can be great for game development time.
- It is enough to disperse players on groups with some distance from each other that allows matchmaking to do it's job. Players that play better will go higher and less skilled player go down. In result you should
- ave probability of winning as visualized on the next picture:

* negative skill difference mean that advantage to Team B (positive for Team A)

Cons:

- Big spreading with time.
- Slow adaptation for player ranking. Rank always change monotonically by 1 point.

- Has no control on rank scaling.
- Does not take into account player personal performance. It will be worse in situation when players play as part of group and other players of the group play much better. Even if one of the players always with the lowest score, his overall skill can be high.
- Does not take into account time of playing in the match. Not every player play from the start of the match.
- Does not change with time when player don't played the Game for long period of time. In this case player rating become unreliable.



Fig, 2. Win probability distribution

Elo. Key ideas of Elo rating system:

- Developed mainly as rating system for two players (like a chess, but can be used for many match-ups with some modifications.
- Operated by only one storable parameter known as rating (R) that should describe player skill. Initial rating recommended to set = 1500, but also it can be any other value even 0.
- Rating calculated in two steps:
- Calculating expected result (E) of the Game based on known player ranks. It get us a normalized value from 0 to 1 of a chance to win.
- Calculating update for player ranks based on the expected result and game outcome or final score (S).

- Let's take a closer look at the math behind Elo. Imagine that we have two players: A with rating 120 and B with rating 80. Expected result of this match can be calculated by the formula:

$$E = \frac{1}{1+10^{(R_B - R_A)/D}} \ , \qquad (e.1)$$

R_A – rating of the player for which expected result is calculating; R_B – rating of the opponent:

$$E_A = \frac{1}{1+10^{(80-120)/400}} \approx 0.56 \ .$$

It is mean that player *A* probability to win the game is 0.56. If we calculate the same for Player *B* we will have probability = 0.44 of winning.

D – ratio of difference, that known in Elo as mysterious 400 number. It should be chosen as difference between players with which one player will have 10 times more chances to win the game. A player with double *D* rating (800 points) over another player have 100 times better chances to win.

D number is arbitrary, if you want a wider range of possible rankings, increase the value, if you want a narrower scale, decrease it.

Let's calculate new rating for player *A* using formula (also assume that player is a winner):

$$R_{new} = R_A + K(S_A - E_A)\ , \qquad \text{(e.2)}$$

$R_A = 120 + 10(1 - 0.56) \approx 124$;

S_A – final score, it is an actual result of the game when win=1, loss=0, draw=0.5.;

K – is known as the k-factor, or development coefficient.

It usually take values started from 2 to 40. It determines how a result affects the rating of players after the game. The higher the value the more change to the rating it brings. In our example we use k-factor equal to 10 and it give player A increase in 4 point to rating, in case of K=20 it will be 8 points. It can be a good idea to use higher coefficient for new players as they tend to improve faster and lower coefficient for all other players. If your game has many game sessions than it's better to use lower K to prevent measures from changing by large amount.

At the end of the game Player A rating rises by 4 point to 124 while Player B rating falls by the same number of point to 76.

The Elo rating system measures the relative strength between players when calculating players rating. If Player A had lost that game than his rating would have fallen by 6 points and Player B will increase his skill by 6 point. It does make sense, as Player B should be rewarded with more points for winning to more skilled player even when he is expected to lose.

Elo rating system work well for estimation player rating and we can use it to combine each players ratings to create a team rating. Unfortunately, Elo only knows how to compare two players at a time. So, first of all let's adapt it to multiplayer game with two teams by changing logic of estimation player skill to match every player of one team to every player of opponent team ($Team_{A1}$ vs $Team_{B1-BN}$). A game with teams of the same size has N(N-1)/2 distinct pairwise matching of player against team opponents.

Expected score in this case can be calculated using the formula:

$$E_A = \frac{\sum_{n=1}^{N}\left(1+10^{(R_n - R_A)/D}\right)^{-1}}{N(N-1)/2}\ , \qquad \text{(e.3)}$$

N – it is a number of players in team,

R_N – every player in opposite team.

Actual score can be calculated in the same way as an original Elo proposed, winner team get 1 and losing team get 0. If we do it like this than every player of team get the same amount of points. Just like simple Win-Lose algorithm except number of points can be different because of K value. It is not fair for players which performed the most.

Let's think about conditions to calculate actual score that include personal performance of a player:

- All players of winning team should increase their score as well as all score of the losing team should be decreased.
- Scores of players in each team should be decreasing monotonically. 1st place has a higher score than 2st place.
- Personal score representing the actual rank in team at the end of the match, is normalized so that the ranks of players in team sum up to 1.

Based on this condition let's write the formula for calculating final score:

$$S_A = S_T + \frac{N - Pl_A}{N(N-1)/2} \quad , \tag{e.4}$$

S_T – it is a match result of player team: win=1, loss=0, draw=0.5,
Pl_A – it is a place of a player in his team in final score board (1 for 1st, 2 for 2nd, …, *N* for last).

Played time it is one more parameter that can be included in skill calculation. There is many possibilities in online games and one of them that team players in game session can be added in the middle of a game as well as for the end of the game session. In this case it is incorrect to calculate player score that can be 0. To avoid this situation we can add an additional variable for calculation:
$\hat{t}$ - normalized version of minutes or seconds played by the player.

$$\hat{t} = \frac{t}{S_t} \quad , \tag{e.5}$$

t – time played in the game session by player,
S_t – total time of the game session.

Final formula for calculating new player ranking should be look like:

$$R_{new} = R_A + K\hat{t}(S_A - E_A). \tag{e.6}$$

Seems everything is fair:

- Player 24 was expected to have the best performance that's why got the biggest rating decline because of loss and bad scoring.
- Player 25 was expected to have lowest performance that's why got the lowest rating decline because of loss

Glicko. The Glicko rating system extends Elo system by adding to it - rating deviation (RD). Therefore rating system operated by three storable parameters player rating, rating deviation which measures the uncertainty in a player rating and latest rating update to find out time period.

Main difference from Elo is that player rating changes not only because of number of matches played but rating changing with time.

How rating deviation works:

- Than lower RD than more certain that rating of a player is accurate. The RD increase slowly over time of inactivity.
- Than lower RD than slower changing rating of a player. That allows for fast adaptation for player rating from the start and after that it not change so dramatically.
- RD takes meaning from 30 (recommended to not use the RD lower than this value to ensure rating can change appreciably even in a relatively short time) to 350 (assumed to be the RD of an unrated player).

Where **c** is a constant that based on the uncertainty of a player skill over a certain amount of time. It assumes how many rating periods would take for a player rating deviation to return to an initial uncertainty of 350. For example, let's suppose that one rating period it is a one month of a playing and assume that after 5 years (or 60 rating periods) we will be uncertain about player skill. Also suppose that typical player has RD = 50.:

$$c = \sqrt{(350^2 - 50^2)/60} \approx 44.72 \quad . \qquad \text{(g.1)}$$

Where **t** is the amount of rating periods since player was in the Game. For online game we can decide that if player played at least once in the latest month than it is 0, and with every month that he doesn't play it is +1. Let's calculate RD=50 for different t (t=1 mean player doesn't played more than a.

Table1.
Elo rating calculation for 2 teams (K=10)

Player	R	E	Game Score	R_{new}
Team A (win – ratings go up)				
Player 11	1800	0.68	3500	1802.6 (+2.6)
Player 12	1700	0.57	2200	1702.8 (+2.8)
Player 13	1650	0.52	2100	1652.7 (+2.7)
Player 14	1850	0.73	1200	1851.6 (+1.6)
Player 15	1140	0.13	1100	1143.7 (+3.7)
Player 16	1500	0.37	720	1502.5 (+2.5)
Player	R	E	Game Score	R_{new}
Team B (loss – ratings go down)				
Player 21	1800	0.68	2000	1798.6 (-1.4)
Player 22	1750	0.62	1800	1748.5 (-1.5)
Player 23	1600	0.466	1800	1598.9 (-1.1)
Player 24	1850	0.73	1200	1847.6 (-2.4)
Player 25	1100	0.106	1100	1099.8 (-0.2)
Player 26	1500	0.37	750	1498.5 (-1.5)

The Player A current RD should be determined :

$$RD = min\left(\sqrt{RD_A^2 + c^2 t}, 350\right) \qquad \textit{(g.2)}$$

month) using formula g.2:

Table 2.
Calculating RD with different t

t=0	t=1	t=2	t=3
50.0	67.08	80.62	92.2

```
from collections import namedtuple

P = namedtuple('PlayerInfo', ['rating', 'score'])
D = 400
K = 10

winning_team = [P(1800, 3500), P(1700, 2200), P(1650, 2100),
            P(1850, 1200), P(1140, 1100), P(1500, 720)]
losing_team = [P(1800, 2000), P(1750, 1800), P(1600, 1800),
             P(1850, 1200), P(1100, 1100), P(1500, 750)]


def calc_for_team(team: list[P], opponent_team: list[P], win=True):
    team_size = len(team)

    new_ratings = {}
    for player in team:
        place_in_team = sorted(team, key=lambda p: p.score, reverse=True).index(player) + 1
        personal_score = (team_size - place_in_team)/(team_size*(team_size - 1)/2)
        expected_scores = [(1 + 10**((op.rating - player.rating) / D))**-1
                           for op in opponent_team]
        actual_scores = [K * ((1 if win else 0) + personal_score - e)
                         for e in expected_scores]

        new_rating = player.rating + round(sum(actual_scores) / (team_size * (team_size -
1)/2), 1)
        new_ratings[player] = new_rating

    return new_ratings


if __name__ == '__main__':
    print("Winning team:")
    for player_, rating in calc_for_team(winning_team, losing_team, win=True).items():
        print(f" {player_} rating become {rating}")

    print("Losing team:")
    for player_, rating in calc_for_team(losing_team, winning_team, win=False).items():
        print(f" {player_} rating become {rating}")
```

Table above shows how RD increasing over a time.
Next step is to calculate expected score between two players:

$$E_A = \left(1 + 10^{-g(R_A - R_N)/D}\right)^{-1} \quad , \qquad \text{(g.3)}$$

D=400 – ratio of difference, the same as in Elo,
R_A – is the player's rating,
R_N – is the opponent's rating.
g – is a function that calculated by the formula:

$$g = \frac{1}{\sqrt{1+3q^2RD^2/\pi^2}} \quad , \tag{g.4}$$

q – is a constant which is equal to log(10)/D = 0.0058565.

Before new rating is calculated, the new RD is needed to calculate for the player:

$$RD_{Anew} = \frac{1}{\sqrt{1/RD^2+1/d^2}} \quad , \tag{g.5}$$

where d^2 calculated as:

$$d^2 = \frac{1}{q^2g^2E(1-E)} \quad . \tag{g.6}$$

Having expected score, game session outcome and new RD we can find out updated rating using formula:

$$R_{Anew} = R_A + qRD_{Anew}^2 g(S_A - E_A) \quad , \tag{g.7}$$

S_A – is the outcome of the Game session: win=1, loss=0, draw=0.5.
Glicko as well as Elo designed to work only with two players, so we need to adapt it to work with teams, just like we do for Elo. In the simplest way we just need to sum up rating and rating deviation for every player of the Team i with every player in Team j and than divide on number of player in teams. For two teams of the same size calculation of rating and rating deviation can be wrote like:

$$RD_{Anew} = \frac{\sum_{1\le j\le N}\left(\sqrt{1/RD^2+1/d^2}\right)^{-1}}{N_T} \quad , \tag{g.8}$$

$$R_{Anew} = \frac{\sum_{1\le j\le N} R_A + qRD_{Anew}^2 g(S_A - E_A)}{N_T} \quad , \tag{g.9}$$

N_T – is the number of player in one team.
Let's get two teams and calculate ratings for them:

TrueSkill

The TrueSkill rating system is also generalization of the Elo system, developed by Microsoft Research and has been patented (no free license) and used for Xbox games. System additionally to rating tracks the uncertainty about player rating, can deal with any number of teams (even with asynchronous team sizes) and can infer individual skills from team results.

The TrueSkill more adapted to multiplayer online games with teams of many players including:

- Individual player skill for every team member. Team wins but player own skill needed for further matchmaking.
- Partial play that allows to handle situation where a player wasn't present for the entire game ssesion. Implemented as weighted sum factor where the weights are the percentage of the time player was present.

Table 3.
Glicko rating calculation for 2 teams

Player	R	RD initial	t	RD	g	E	d^2	RD_{new}	R_{new}
Team A (win – ratings go up)									
Player 11	1662	290	0	290	0.736	~0.43	499.04^2	249.74	1812.0
Player 12	1746	252	0	252	0.781	~0.51	469.47^2	220.98	1851.66
Player 13	1794	100	1	109.46	0.945	~0.57	402.07^2	105.35	1819.4
Player 14	1823	50	2	66.95	0.969	~0.61	399.15^2	78.68	1836.26
Player 15	1842	30	0	30	0.995	~0.63	403.81^2	30	1843.84
Player 16	1500	350	0	350	0.669	~0.30	585.90^2	299.81	1743.31
Team B (loss – ratings go down)									
Player 21	1662	290	0	290	0.736	~0.43	499.04^2	249.74	1547.83
Player 22	1746	252	0	252	0.781	~0.51	469.47^2	220.98	1632.1
Player 23	1794	100	2	118.17	0.936	~0.57	402.07^2	113.12	1754.08
Player 24	1823	50	3	91.9	0.960	~0.61	399.15^2	89.41	1795.82
Player 25	1842	30	0	30	0.995	~0.63	403.81^2	30	1838.72
Player 26	1500	350	0	350	0.669	~0.30	585.90^2	299.81	1396.98

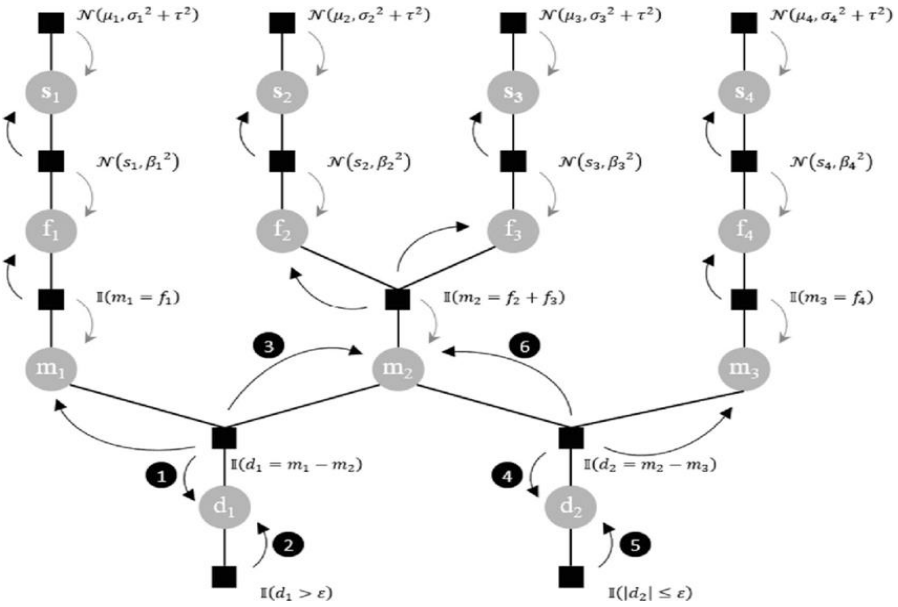
From the first look on factor graph it is hard to understand how skills should be calculated. That is why we will try to explain TrueSkill without it but in straight forward way.

As well as Glicko rating system TrueSkill system assumes that the skill represented a Gaussian distribution when each player satisfies a normal distribution of **mean μ** and **variance σ**, which are fixed to be the same for all players at the initiate step: $\boldsymbol{\mu}_{\mathbf{initial}} = 25$, $\boldsymbol{\sigma} = 25/3$ and $\boldsymbol{\beta} = \sigma/2$. The parameter β controls the updating effect, larger β leads to a more dramatic changes.

```
import math
from collections import namedtuple

P = namedtuple('PlayerInfo', ['skill', 'rd', 'score', 't'])

D = 400
RD_MAX, RD_MIN, RD_TYPICAL = 350, 30, 60
RT_AWAY = 60

c = math.sqrt((RD_MAX**2 - RD_TYPICAL**2)/RT_AWAY)
q = round(math.log(10)/D, 7)

winning_team = [P(1662, 290, 3500, 0), P(1746, 252, 2200, 0), P(1794, 100, 2100, 1),
            P(1823, 50, 1200, 2), P(1842, 30, 1100, 0), P(1500, 350, 720, 0)]
losing_team = [P(1662, 290, 3500, 0), P(1746, 252, 2200, 0), P(1794, 100, 2100, 2),
            P(1823, 50, 1200, 3), P(1842, 30, 1100, 0), P(1500, 350, 720, 0)]

def calc_rd(rd_initial, time_periods_away=0):
   return min(math.sqrt(rd_initial ** 2 + c ** 2 * time_periods_away), RD_MAX)

def calc_g(rd):
   return 1/math.sqrt(1 + (3 * q**2 * rd**2)/math.pi**2)

def calc_expected_score(g, player: P, opponent: P):
   return 1/(1 + 10 ** ((-g * (player.skill - opponent.skill)) / D))
```

```
def calc_new_rating(r, rd_new, g, s, e):
   return r + q * rd_new**2 * g * (s - e)

def calc_for_player(player: P, opponent_team: list[P], win=True):
   rd = calc_rd(player.rd, time_periods_away=player.t)
   g = calc_g(rd)
   expected_scores = [calc_expected_score(g, player, p)
                      for p in opponent_team]
   rd_x = [calc_rd_new(rd, calc_d2(g, e)) for e in expected_scores]
   ranking = [calc_new_rating(player.skill, rd, g, (1 if win else 0), e)
              for rd, e in zip(rd_x, expected_scores)]
   rd_new = sum(rd_x)/len(opponent_team)
   r_new = max(sum(ranking)/len(opponent_team), RD_MIN)
   return r_new, rd_new
```

The TrueSkill express the model as a factor graph and use approximate message passing to infer the marginal belief distribution over the skill of each player. An example of it:



Fig, 3. TrueSkill factor graph example

The Player current sigma (**σ**) should be determined by using formula:

$$\sigma = \sqrt{\sigma^2 + \tau^2} \quad . \tag{t.1}$$

Where **τ** is the dynamic factor which restrains a fixation of rating. The recommended value is percent of sigma: σ/100.

Updating mean value calculation after game session shown below:

$$\mu = \mu + S_T \left(\frac{\sigma^2}{c}\right) \upsilon\left(\frac{t}{c}\right) \ , \tag{t.2}$$

S_T - is the outcome of the Game session for your team: win=1, loss=-1:

$$t = \mu_{winner} - \mu_{loser} \quad , \tag{t.3}$$

$$c = \sqrt{\left(N_S \beta^2 + \sigma_{winner}^2 + \sigma_{loser}^2\right)} \ , \tag{t.4}$$

N_S – it is a total number of players in game session:.

$$\upsilon(t/c) = \frac{N(t/c)}{\Phi(t/c)} \tag{t.5}$$

N and Φ represent the PDF and CDF of a standard normal distribution, respectively.

Updating sigma new value calculated by formula:

$$\sigma = \sqrt{\left(\sigma^2 \left(1 - \frac{\sigma^2}{c^2} \omega\left(\frac{t}{c}\right)\right)\right)} \quad , \tag{t.6}$$

where:

$$\omega\left(\frac{t}{c}\right) = \upsilon\left(\frac{t}{c}\right)\left(\upsilon\left(\frac{t}{c}\right) + t\right). \tag{t.7}$$

TrueSkill was the first algorithm documentally applicable to team games. TrueSkill sums the ratings of team members to calculate the rating of their team. Most of the ensuing rating systems leveraged a similar approach in their calculations. Calculation result for two teams:

Table 4.
TrueSkill rating calculation for 2 teams

Player	μ	σ	μ_new	σ_new
Team A (win – ratings go up)				
Player 11	40	2.10	40.158	2.09
Player 12	55	3.30	55.389	3.29
Player 13	35	1.25	35.056	1.25
Player 14	25	6.10	26.33	5.96
Player 15	33	5.92	34.25	5.79
Player 16	25	8.333	27.48	7.96
Team B (loss – ratings go down)				
Player 21	45	1.50	44.92	1.50
Player 22	60	1.30	59.94	1.30
Player 23	30	3.25	29.62	3.23
Player 24	20	5.10	19.07	5.02
Player 25	33	5.92	31.748	5.79
Player 26	25	8.333	22.52	7.96

- Player 16 & Player 26 are newbies and we can saw that they have more rating change than everyone else. This means the algorithm try to quickly adapt player to reasonable approximations of skill.
- Player 13 got the lesser change to rating because of the lowest value of sigma.

Weng-Lin

Weng-Lin or the OpenSkill rating system is similar to TrueSkill.
They propose to use generalized approach with different distribution models for different purposes. It is developed specially for online games with multiple teams, this algorithm aims to be simpler and faster than TrueSkill while yielding similar accuracy. Also it is distributed under free license.

Can be used with distribution models, like:

- Bradley-Terry - follow a logistic distribution over a player's skill, similar to Glicko.
- Plackett-Luce - generalized Bradley-Terry model for k ≥ 3 (k-number of teams participating in a game).
- Thurstone-Mosteller - models follow a Gaussian distribution, similar to TrueSkill.

Rating system assumes that skill represented by two value of **mu** (=25) and **sigma** (=25/3). We will skip the review of formulas this time, because this rating system includes several models and therefore deserves a separate article with an overview of mathematics and approaches. Limiting ourselves to results that can be obtained using, for example library [10].

Every of observed algorithms can be suitable for modern game. The most advanced are the algorithms of TrueSkill, also the most famous and OpenSkill (Weng-Lin) that looks simpler, designed to be faster and give the same results.

Table 5.
OpenSkill rating calculation for 2 teams

Player	μ	σ	μ_new	σ_new
Team A (win – ratings go up)				
Player 11	40	2.10	40.12	2.09
Player 12	55	3.30	55.30	3.29
Player 13	35	1.25	35.04	1.25
Player 14	25	6.10	26.01	6.04
Player 15	33	5.92	33.95	5.87
Player 16	25	8.333	26.89	8.18
Team B (loss – ratings go down)				
Player 21	45	1.50	44.94	1.50
Player 22	60	1.30	59.96	1.30
Player 23	30	3.25	29.71	3.24
Player 24	20	5.10	19.29	5.06
Player 25	33	5.92	32.05	5.87
Player 26	25	8.333	23.11	8.19

Table.6.
Comparison of some difference about algorithms

	Win-Lose	**Elo**	**Glicko**	**TrueSkill**	**OpenSkill**
Support for teams	-	+	+	++	++
Fast adaptation for newbie	-	-	+	+	+
Take into account personal performance	-	+	+	++	++
Take into account time when player not in the Game	-	-	+	+ (TrueSkill2)	-
Has control of rank scaling	-	+	+	+	+

Conclusion

In this article, we observed the four popular rating systems for online multiplayer games. We dived into the mathematics behind algorithms and compare the results. It's also clear that every new algorithms is an evolution of the pasts one and that's why all of them have many similarity among each other. And even updates of exist algorithms in some way it is a loan of good idea of other similar algorithms

Elo is the most simple to implement and can easily be good used for PvE games and possible for PvP, but not suitable for multiple teams and asynchronous game modes.

We hope this article has clear up the workings of the most popular rating systems. The focus was on core logic and calculation behind algorithms, so it is not cover all details about them.

References

1.TrueSkill official documentation. URL: https://proceedings.neurips.cc/paper/2006/file/f44ee263952e65b3610b8ba51229d1f9-Paper.pdf
2.TrueSkill2 official documentation. URL: https://www.microsoft.com/en-us/research/uploads/prod/2018/03/trueskill2.pdf
3.Math behind TrueSkill. URL: http://www.moserware.com/assets/computing-your-skill/The%20Math%20Behind%20TrueSkill.pdf
4.Weng Lin official documanteation. URL: https://www.csie.ntu.edu.tw/~cjlin/papers/online_ranking/online_journal.pdf
5.Glicko official documentation. URL: http://www.glicko.net/glicko/glicko.pdf
6.“Developing a generalized Elo rating system for multiplayer games“. URL: https://towardsdatascience.com/developing-a-generalized-elo-rating-system-for-multiplayer-games-b9b495e87802
7.“An ELO based approach to model team players and predict the outcome of games“. URL: https://core.ac.uk/download/pdf/187127715.pdf
8.“The Evaluation of Rating Systems in online free-for-all games” URL: https://arxiv.org/pdf/2008.06787.pdf
9.“Gaussian Process Priors for Dynamic paired comparison modelling”. URL: https://arxiv.org/pdf/1902.07378.pdf/
10. OpenSkill source code. URL: https://github.com/OpenDebates/openskill.py