Ranking and Matchmaking

How to rate players' skills for fun and competitive gaming

By Thore Graepel and Ralf Herbrich

Introduction

With the widespread availability of broadband connections and the emergence of unified online gaming services, more and more games provide online multiplayer modes in which gamers can play together, either cooperatively or competitively – often even in teams. These online game modes are so popular because playing with or against other human players can be much more interesting, challenging and fun than playing with a typical game AI.
The popularity of these online multiplayer scenarios makes it necessary to address two questions:

1. How can we match players online for the best possible experience?
2. How can we provide players with incentives to continue playing?

It turns out that a key consideration to answering both these questions is the measurement of players' skills. Besides aspects of social matchmaking such as language, reputation, age and gender, it is mostly the skills of the gamers involved that decide if a game is balanced and fun. Also, for the more competitive gamers it is crucial to obtain fair feedback on their skills, be it as an incentive to improve their game or simply for bragging rights towards others.
In this article we consider the question how a multiplayer online game together with a suitable skill rating system (ranking system) should be designed so as to enable a great and sustainable experience for the participating players. We will consider three aspects of designing such an experience

1. The underlying ranking and matchmaking system. How do ranking systems such as Elo and TrueSkill work? What do the numbers mean and how can they be used for matchmaking?
2. The design of ranked game modes. How can ranked games be organized so as to give players an appropriate choice, make the skill rating meaningful and ensure that enough players are available for matchmaking?
3. The design of skill displays. When and how should the gamers receive feedback about their playing strength? How can the skill rating be used to improve the game experience?

Skill-Based Ranking and Matchmaking

The key idea behind skill-based ranking and matchmaking is that a game is fun for the participating players if the outcome is uncertain in the sense that each of the participating teams of players has a fair chance of winning. In typical sports disciplines like tennis or Formula One racing, players are awarded points according to their performance in certain competitive events. Typically, these points are purely cumulative: participating in more events is always better for collecting more points.

The resulting performance numbers typically only partially reflect a player's or team's skill. In particular, they depend on the number of events the player participated in but disregard the quality of the opposition the player was facing.

In contrast, the Elo rating system, introduced by Prof. Árpád Élő in 1959 for Chess and adopted by the World Chess Federation (FIDE) in 1970, takes a different approach. In the Elo system (and in related systems such as TrueSkill™) rating players is viewed as an estimation problem with the ultimate goal of estimating players' skills so as to be able to predict the probability of game outcomes between players purely based on their skill ratings.

For a two player game the probabilities of win and loss are modeled in terms of Gaussian probability densities, sometimes called Bell curves. Figures 1 and 2 illustrate how to obtain the winning probabilities from the scores in a two-player game. Suppose Player Blue has a skill of 25 and Player Red has a skill of 33 (peak of the Bell curves indicated by the dashed line). The performances of both players will vary from game to game around their skills as shown by the red and blue performance distribution curves on the walls in Figure 1. If Player Blue's performance exceeds player Red's performance it is predicted that Player Blue wins and vice versa. The probability of these two events is proportional to the red and blue volume under the two-dimensional bell curve in Figure 1. This probability is most efficiently computed using the code in Listing 1 – an implementation of the red Φ function in Figure 2. The Elo system aims to adjust the ratings such that the observed game outcome becomes more likely under the assumed model. This results in the two-player Elo update function given in Listing 3.

The TrueSkill system available in the Xbox 360 Live online gaming service is a proprietary generalization of the Elo system to games in which players compete in teams and to games in which there are more than two parties involved. The TrueSkill system also explicitly models draws and tracks the uncertainty associated with a player's rating thereby automatically adjusting the step-size in the update equations. Because currently only Xbox 360 game titles can benefit from the TrueSkill system, we will describe ways to achieve similar results based on the standard Elo update in the next Section.

Two interesting quantities to check are skill developments of individual players and skill distributions over the whole population. In Figure 3 and 4 we compare the TrueSkill and Elo systems on the Halo 2 Beta game data of the Free-for-All game mode. For the entire player population (Figure 3) the distribution of Elo ratings is still very peaked around the starting level 25 while the TrueSkill skill distribution is already nicely spread out due to its adaptive learning rate. However, when considering only players that have played at least 50 games (Figure 4) the distributions start to match up. In Figure 5 – 7 we show individual players' skill developments as a function of the numbers of games played. TrueSkill converges extremely fast. The speed of convergence of the Elo system can be controlled by a parameter α which trades off convergence speed with stability – large values lead to fast convergence but single game outcomes still have a large impact on the rating. For small values of α the Elo system reaches the true skill more slowly (> 100 games) leading to a more stable estimate. Note that reducing α as a function of the number of games played could be used to further mitigate this behavior.

It is interesting to note that Elo's update equation depends only on the win/loss outcome. Similarly, TrueSkill's update equations take into account only the finishing order of the players/teams involved. None of these systems incorporates the actual final score, say, the number of kills in a shooting game or the finishing time in a racing game. This is a deliberate choice.

First, taking into account only the finishing order makes these rating systems universal because in almost any game a finishing order can be computed from the detailed game outcomes.

Second, it is crucial that the purpose of the game and the behavior of the rating system be aligned: people striving for high ratings should be forced to play in accordance with the spirit of the game. Taking the margin into account by which a game was won can be very misleading. As an example, consider a player who is trying to catch up with the opponent before the time is up, thereby taking into account the risk of falling further behind due to counter chances of the opponent. Should the rating system be designed to discourage this fighting spirit?

For the updates of the individual players' scores in team games it is crucial to consider the finishing order of the teams only. If individual performance metrics are used in the rating system players will aim at maximizing those metrics instead of teaming up with their team members and try to win the game. For example, if you choose to reward the number of flags scored in a capture the flag game, do not be surprised if you find everyone rushing for the flag at once, but also some dead bodies who died with the flag in their hands betrayed by an overly ambitious team mate …

Skill-Based Matchmaking

The main application of skill rating is matchmaking. Ideally if every player has played enough matches to estimate their skill and if enough players of each skill are available for matchmaking it would be possible to compose matches such that every team on average has the same chance of winning.

The TrueSkill matchmaking system uses the winning probability based on skill ratings of the players or teams involved to find the most balanced matches possible. A simple heuristic to achieve similar results is to use the rating differences of players directly to match them. One simple criterion for a multi-player match is to ensure that the difference between the highest rating and the lowest rating does not exceed a pre-defined rating level gap. This gap should be chosen depending on the expected player population as discussed further below. If the gap is small, tight matches result but it is harder to find players of appropriate skill. If the gap is large, more unbalanced matches may result, but it is easier to find players of appropriate skill.

For the team configuration the following method for assigning players to teams can be used. Order the players by their skill ratings and start by assigning the strongest player to team A, the next strongest to team B, etc. Once, each team has exactly one player, always assign the next player to the team with the lowest total skill rating as given by the sum of the individual team members' ratings.

Multi-Team Rating Made Easy: Generalized Elo

The classical Elo system is designed to handle games between two individual players. However, in online games it is quite common to have games with more than two players and even games with more than two teams. While the TrueSkill system addresses these issues in a principled way, we will describe here the so-called dueling heuristic that allows you to use the Elo update equation in these more general cases. The key idea of the dueling heuristic is simple: After a game is finished, consider the game outcome as a collection of two player game outcomes ("duels").

In a game with K individual players you have $K * (K - 1)$ pairs of players. You calculate all the Elo updates for each player using their team's relative standing in the finishing order (without adjusting the ratings of the participating players) and average those updates to obtain that player's update.

In a game with k teams essentially the same heuristic is used, but now each player has duels only with all the players in the other teams, again calculating all the individual Elo updates based on the relative standing of the respective teams and averaging

You can find the data-structures for representing team game outcomes in Listing 2 and the code for the dueling heuristic in Listing 4.

How much rating information from a single game?

Suppose you have two players A and B, and the only possible outcomes are "A wins" or "B wins". Clearly, a game outcome contains at most 1 bit of information. "At most" because if we knew from the ratings that player A was much stronger than player B, the game outcome would be no surprise and hence there would be almost no information conveyed.

So, how many bits of information are contained in a ranking of N players? Suppose you wanted to store an integer with each of the players reflecting their position in the ranking. You would need N integers each of which with a range of at least 0 to N-1, requiring $\log_2(N)$ bits per integer for a total of $N * \log_2(N)$ bits.

Suppose you have a popular game mode with 64,000 players. Then you need to learn about $64{,}000 * \log_2(64{,}000)$ bits $\approx$ 1,000,000 bits of information for a complete ranking. Incidentally, $K * \log_2(K)$ is also an upper bound on the information you can learn from a game with K participating parties (single players or teams). So, suppose you have 8 players individually competing in a game, then you can learn at most $8 * \log_2(8)$ bits = 24 bits from that game. As a consequence, we need about 1,000,000 bits / 24 (bits/game) $\approx$ 40,000 games to learn the overall ranking. Since we assumed that each game requires 8 participants, each player must play about 40,000 games * 8 players / 64,000 players = 5 games to achieve an accurate ranking. Note, that in the case of team games, the learned information is shared among the players, so team games with big teams tend to convey very little information about individual players' skills.

The TrueSkill system almost attains the information-theoretic limits explained above. For the (generalized) Elo system, they still give a rough lower bound on how many games are required for convergence.

"Gaming the System"

A question often asked in relation to rating systems is if the system can be "gamed." For combined rating and matchmaking systems there are essentially two things people might want to do:

1. Have their ratings appear higher than they actually are to show off on the leader boards ("stats boosting")
2. Have their ratings appear lower than they actually are to manipulate the matchmaking so they get easy-to-win games ("de-leveling")

Stats boosting in Elo type systems as described here is essentially only possible if the boosters are able to manipulate the game outcome in their favor.

1. They can have a higher-skilled gamer play under their account.
2. They can try to get matched with friends who are willing to help manipulate the game outcome. Possibilities here are network "bridging" and entering matchmaking at exactly the same time when a low population of players can be expected.
3. They can manipulate the game software or the network connection.

De-leveling may be achieved as follows.

1. They can obtain a new account effectively resetting their ratings.
2. They can lose matches on purpose to get negative Elo updates. In order to de-level as quickly as possible de-levelers frequently leave the game as soon as it has started.

Designing the Experience I: Game Modes and Rules

In multiplayer online games the participating players can be viewed as demanding customers as well as a resource to help serve other customers, because it is the players who provide the intelligent opposition for other players. In an ideal world there would be an unlimited supply of players online at every time, for every game mode, and at any skill level. However, in practice the amount of players available online is limited by factors such as the number of game copies sold, the amount of time gamers spend online and the number of titles competing for the gamers' online time. In addition, those gamers who do play the title may have varying skills and preferences. As a consequence, it is essential to design the game such that the matchmaking algorithm will be able to find appropriate matches for a player in an appropriate amount of time (see box "You Can't Have It All: Match Quality and Waiting Time.").

You Can't Have It All: Match Quality and Waiting Time

Suppose your multiplayer online game can be described by the following numbers:

- N_{Modes} is the number of different game modes in your game, where gamers can only ever get matched if they choose the same game mode.
- T_{Match} is the average duration of a match in minutes.
- $N_{\text{Participants}}$ is the average number of players in a single match.
- N_{Skills} is the number of buckets required to divide the gamer population into buckets of roughly equal skill.
- T_{Waiting} is the average time in minutes a gamer has to wait until the match starts.
- N_{Online} is the average number of players online in the title at any given time.

Then we have the following formula:

$$T_{\text{Waiting}} = N_{\text{Modes}} * T_{\text{Match}} * N_{\text{Participants}} * N_{\text{Skills}} / N_{\text{Online}}$$

As a consequence we know that

- The waiting time *decreases* the more people are online playing your game.
- The waiting time *increases* the longer each match takes, the more game modes there are, the more participants are required for each match and the more skill buckets there are to your game.

The take-away message of this calculation is that the more players are playing your game online the more you can afford to have any of the following while maintaining an acceptable average waiting time per player:

- Long matches that take away players from matchmaking for a long period of time.
- Big matches that take away many players from matchmaking.
- More game modes that reduce the matching pool of players to the same game mode.
- Tight matches that reduce the matching pool of players to buckets of similar skill.

As an example, suppose you have $N_{\text{Modes}} = 10$ game modes, your average match takes $T_{\text{Match}} = 10$ minutes and requires $N_{\text{Participants}} = 8$ participants all of which should belong to the same of $N_{\text{Skills}} = 10$ skill buckets. Then, in order to have a waiting time of not more than 1 minute you should have at least 8,000 players online at any moment in time playing your game. Note that this calculation does not even take into account that the very low and very high skill buckets are probably under-populated, that some game modes may be much less popular than others, and that player populations vary greatly across time of day and different time zones.

While the previous calculation indicates that there should be less rather than more game modes for a typical title, it is also important to consider the actual skills necessary to succeed in a given game mode. The general rule of thumb is that a game mode should comprise games that require similar skills. Not only will the gamers appreciate the grouping of games according to the skills required, but the skill estimation process will benefit from homogeneous game modes and allow for more consistent matchmaking. For example, in an FPS it may be good to have separate game modes for team games and for Free-For-All games because these require different skill sets. For the same reason, it may be good in a racing game to have separate game modes for low-powered and high-powered cars.

Finally, an important aspect of ranked games is fairness, both in an objective sense to give everyone a level playing field as well as in a subjective sense for the perception of the gamers. Nothing is more frustrating than seeing other gamers achieve an unfair advantage! As a consequence, ranked play works best if it is organized much like a sports tournament with certain rules enforced.

1. No individual player (“host”) should enjoy any privileges over the other participating players. This includes privileges such as decisions about the game mode (including maps, tracks, rules etc.), booting unwanted players, setting the start time for the match, or deciding about team assignments. Ideally, even network-based advantages should be leveled, possibly by appropriate matchmaking.
2. Matchmaking should be based primarily on skill and be otherwise not under the influence of the gamers. Ranked re-matches should be disallowed or limited to one to avoid the risk of collusion.

3. Players should not be allowed to drop from the pre-game lobby once they have gained knowledge about the upcoming match such as map/track, other participating players or game type. Otherwise, they would be able to drop out selectively, gain an advantage and prolong waiting times for the other participants. Drop-outs should be treated as forfeits.
4. Termination criteria for games should be designed such that it is impossible for a player or team to force others into giving up by delaying the end of the game. For example, in a racing game a forced finished timer can be set off once the first player crosses the finishing line.
5. It should be avoided that the same player can participate under different accounts or identities because this would facilitate manipulation of the rating system. Ideally, there should be a cost associated with creating new accounts that discourages multiple identities.
6. In order to discourage de-leveling by dropping, we recommend to either prevent a player from joining a new session when the session they dropped from is still running or to make their Elo update proportional to the fraction of time they spent in the match.

Designing the Experience II: Feedback and Display

Through good matchmaking a good rating system provides great benefits to the players no matter if their skill rating is displayed to them or not. Displaying skill rating information to players can create healthy competition and can provide an incentive for players to continue playing and practicing. It gives great opportunities for bragging in the online community and many players even identify with their skill rating ("I *am* a level 40!"). If skill-based matchmaking is used, displaying the skill ratings leads to a predictable and transparent system.

On the other hand, displaying skill ratings can also be an incentive for cheating and unsporting conduct. Some players are willing to do anything to see a high rating associated with their online identity. As a consequence, fierce competition can destroy the relaxed and fun atmosphere of a game. Also, some people may get frustrated if they see their skill ratings stagnate or even go down. Finally, as discussed above, a fair ranked game mode requires certain restrictions on the configurability of the game that may be undesired for fun play.

As a compromise, a title ideally provides both:

- A competitive set of game modes with strict rules and skill rating displayed.
- A set of fun game modes that can be freely configured and do not display any skill ratings.

Note that skill-estimation and matchmaking can still be used under the hood for such a fun mode. In any case it is important to provide the players with incentives and achievements that complement the skill rating and continue to reward players even when their skill ratings have converged.

If the decision is made to display the skill rating a decision has to be made where, when and how to display them. The obvious starting point is leader boards ordered by skill rating. We found that two kinds of leader boards are most important:

- The top players from the global leader board and
- The player himself in a leader board with their friends or buddies.

Furthermore, statistics about rank in the global leader board, number of players in the global leader board and the percentage of players above the player in the global leader board are interesting quantities to display.

One important concern with skill based leader boards is that players who occupy the top ranks may retire from active play in order to defend their leader board positions. This behavior can be discouraged, e.g., by not displaying players in the leader board who have not played within a given period of time.

Other places to display the skill rating are the pre- and post-game lobbies. In the pre-game lobby the displayed skills help players understand the matchmaking process, gauge the strength of the opposition, and set goals for the match to come ("I am ranked 3rd out of 8, so I had better come out at least 3rd in the match!"). A recent study has shown that in the post-game lobby players are primarily interested in how they performed as compared to their pre-game rank and in how the past game has affected their skill rating. Finally, one can image interesting ways of incorporating the skill ratings into the game itself, e.g., by automatically setting a goal position in a racing game.

Conclusion

Skill based ranking and matchmaking are crucial elements for designing competitive yet enjoyable online gaming experiences. They are essential tools of game design helping shape the experience of online gamers similar to the careful design process necessary to tune offline game AI. We hope that this article not only provides the technical tools for implementing skill based rating but also helps the game designer make the right choices in this relatively uncharted territory of game development. After all, unleashing the competitive element can bring a lot of fun and excitement to online gaming – but with it comes a responsibility to carefully design the experience.

Listings

Listing 1: Numerics Code

```
// Computes the complemetary error function.
public static double erfc (double x)
{
    // check for boundary cases
    if (double.IsNegativeInfinity (x)) return 2.0;
    if (double.IsPositiveInfinity (x)) return 0.0;

    // ... otherwise do the hard work
    double z = Math.Abs (x);
    double t = 1.0 / (1.0 + 0.5 * z);
    double Result = t * Math.Exp (-z * z
        - 1.26551223 +
        t * (1.00002368 +
        t * (0.37409196 +
        t * (0.09678418 +
        t * (-0.18628806 +
        t * (0.27886807 +
        t * (-1.13520398 +
        t * (1.48851587 +
        t * (-0.82215223 +
        t * 0.17087277)))))))));

    return (Result = (x >= 0.0) ? Result : 2.0 - Result);
}

// Computes the cummulative Gaussian distribution.
public static double Phi (double t)
{
    return erfc (-t / 1.4142135623730951) / 2.0;
}
```

Listing 2: Data Structures

```
public struct Team
{
    public int []  PlayerIDs;       // List of player IDs.
    public double  Points;          // The victory points.
}

public class Game
{
    public Team [] Teams;           // The list of teams for a game.
}

public struct ELOParameters
{
    public double Alpha;        // Weight of current game outcome.
    public double Beta;         // Performance variation.
}

public enum GameOutcome
{
    Player1Wins, Player2Wins, Draw;  // Three different game outcomes.
}
```

Listing 3: ELO Core Code

```
public static void ELOUpdate (ref double skill1, ref double skill2,
GameOutcome outcome, ELOParameters param)
{
    double K = param.Alpha * param.Beta * Math.Sqrt (Math.PI);
    double C = Phi ((skill1–skill2) / (Math.Sqrt (2.0) * param.Beta));
    double Delta = 0.0;

    switch (outcome)
    {
        case GameOutcome.Player1Wins:
            Delta = K * (1.0 - C); break;
        case GameOutcome.Draw:
            Delta = K * (0.5 - C); break;
        case GameOutcome.Player2Wins:
            Delta = -K * C; break;
    }

    skill1 += Delta;
    skill2 -= Delta;
}
```

Listing 4: General ELO Code (with Dueling heuristic)

```
private static GameOutcome Compare (double points1, double points2)
{
    if (points1 > points2)
        return (GameOutcome.Player1Wins);
    if (points1 < points2)
        return (GameOutcome.Player2Wins);
    return (GameOutcome.Draw);
}


public static double [] ELOUpdate (Game game, double [] skills,
ELOParameters param)
{
    double [] newSkills = new double [skills.Length];

    // loop over all players in all teams
    for (int cTIdx = 0; cTIdx < game.Teams.Length; cTIdx++)
    {
        Team cT = game.Teams [cTIdx];
        for (int cPIdx = 0; cPIdx < cT.PlayerIDs.Length; cPIdx++)
        {
            int noUpdates = 0;
            double delta = 0.0;
            int idx = cT.PlayerIDs [cPIdx];

            // loop over all other players in all other teams
            for (int oTIdx = 0; oTIdx < game.Teams.Length; oTIdx++)
            {
                if (oTIdx == cTIdx) continue;

                Team oT = game.Teams [oTIdx];
                for (int oPIdx = 0;oPIdx < oT.PlayerIDs.Length;oPIdx++)
                {
                    double tmp1 = skills [idx];
                    double tmp2 = skills [oT.PlayerIDs [oPIdx]];
                    GameOutcome outcome = Compare(cT.Points,oT.Points);
                    ELOUpdate (ref tmp1, ref tmp2, outcome, param);
                    delta += (tmp1 - skills [idx]);
                    noUpdates++;
                }
            }

            if (noUpdates > 0) delta /= (double) noUpdates;
            newSkills [idx] = skills [idx] + delta;
        }
    }

    return newSkills;
}
```

Figures

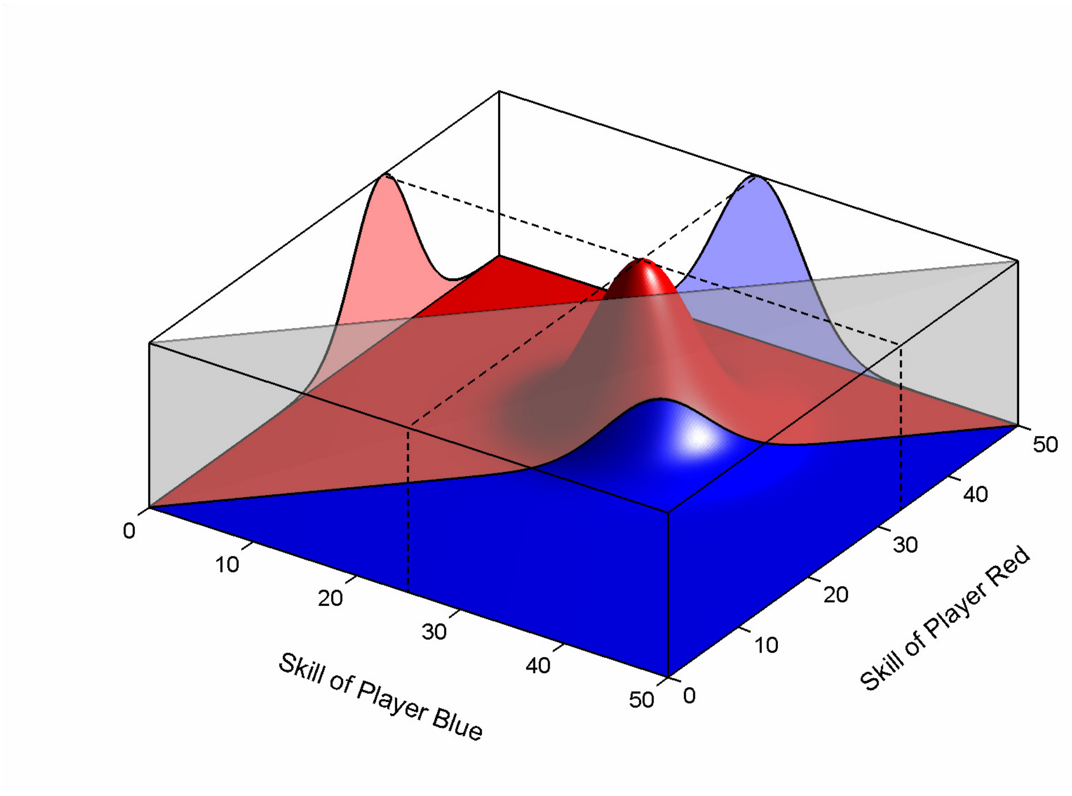


Figure 1

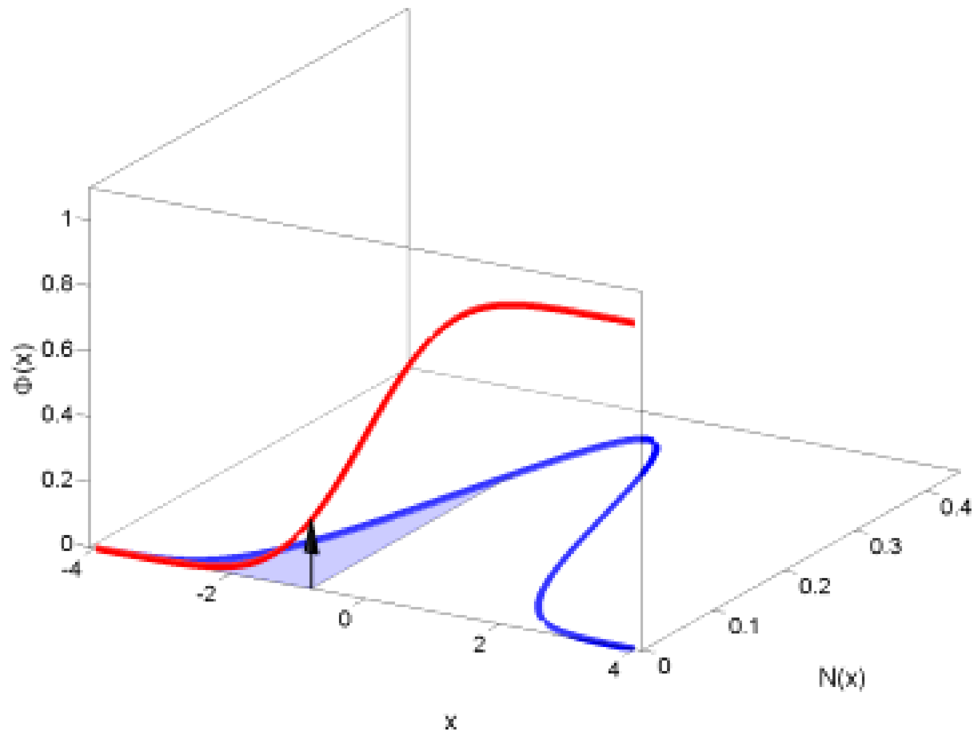


Figure 2

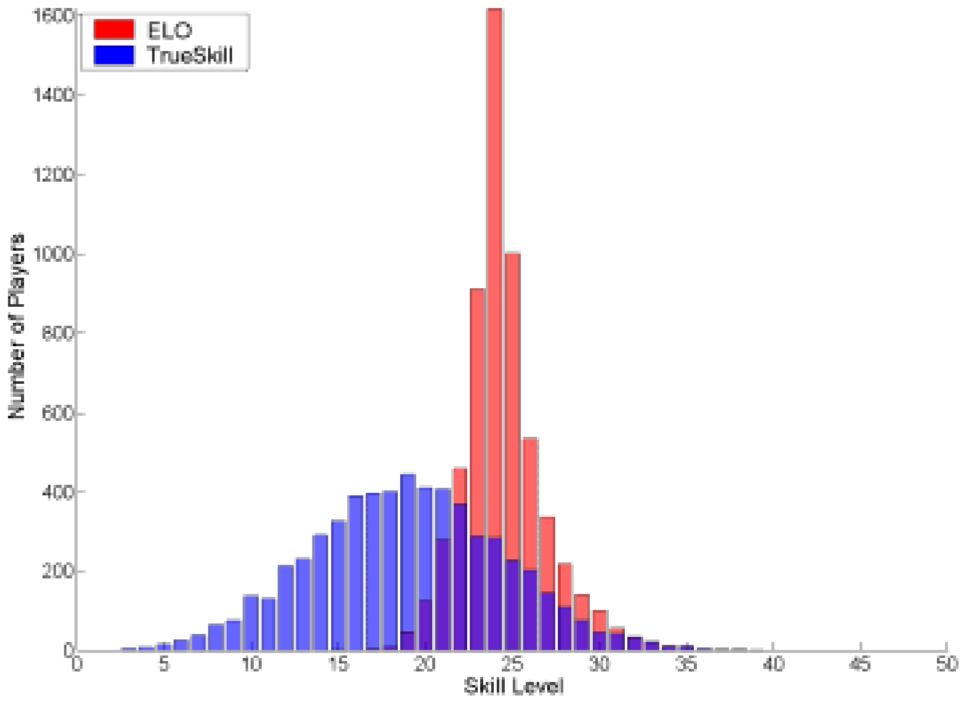

ELO
TrueSkill
Number of Players
Skill Level
0
200
400
600
800
1000
1200
1400
1600
0
5
10
15
20
25
30
35
40
45
50

Figure 3

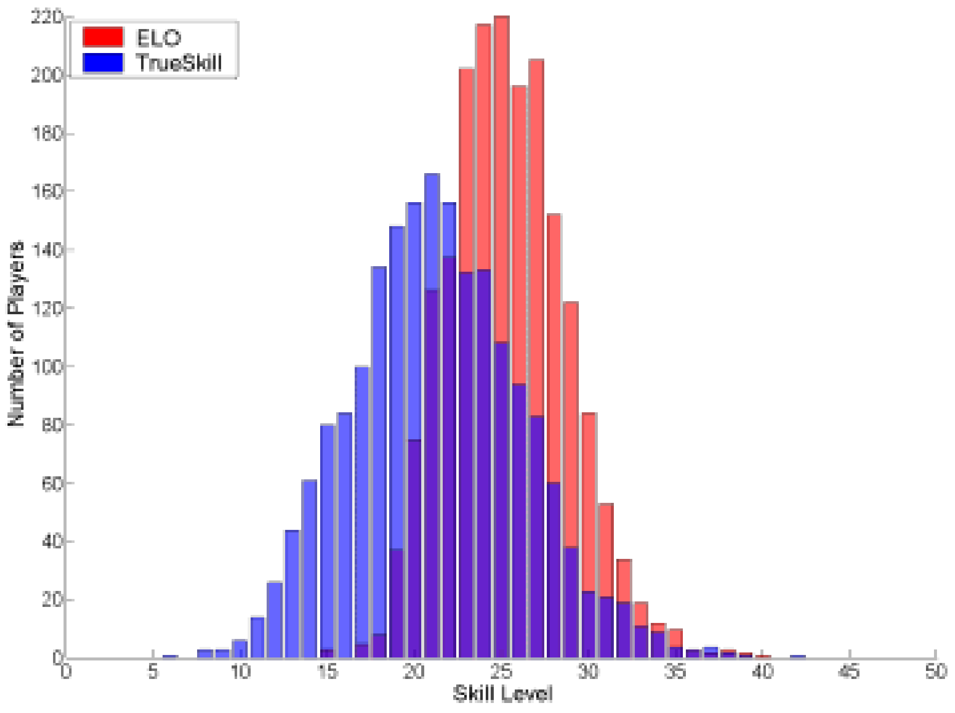

ELO
TrueSkill
Number of Players
Skill Level
0
20
40
60
80
100
120
140
160
180
200
220
0
5
10
15
20
25
30
35
40
45
50

Figure 4

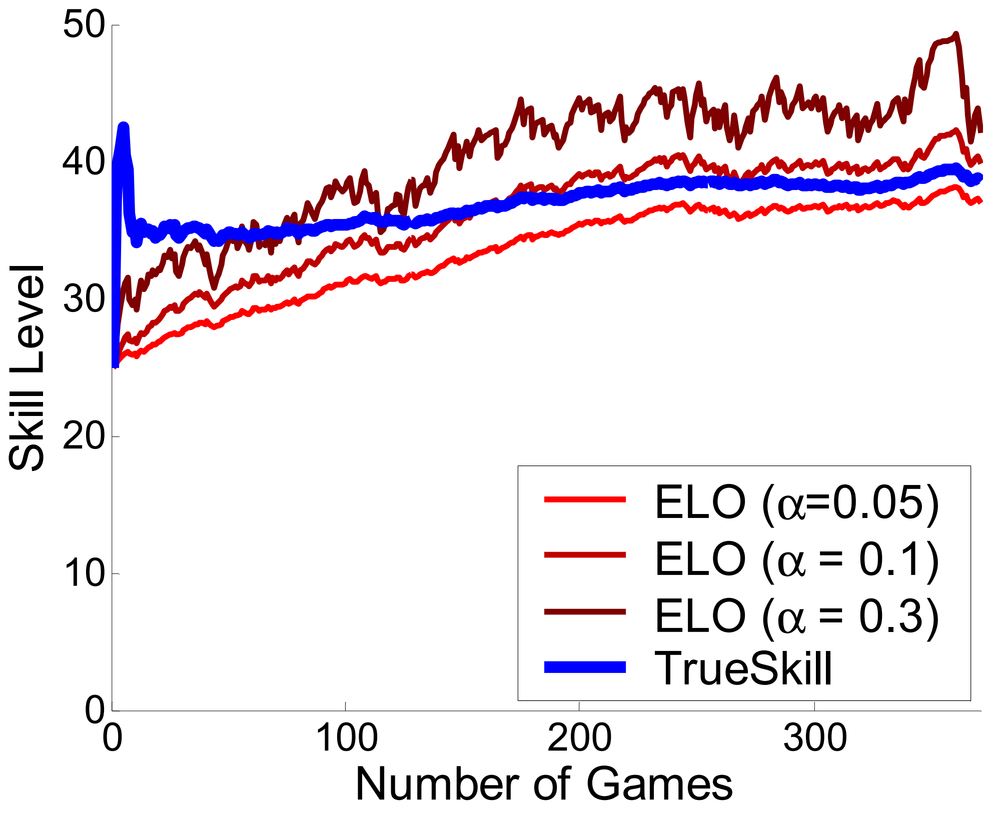

50
40
30
20
10
0
Skill Level
0
100
200
300
Number of Games
ELO (α=0.05)
ELO (α = 0.1)
ELO (α = 0.3)
TrueSkill

Figure 5

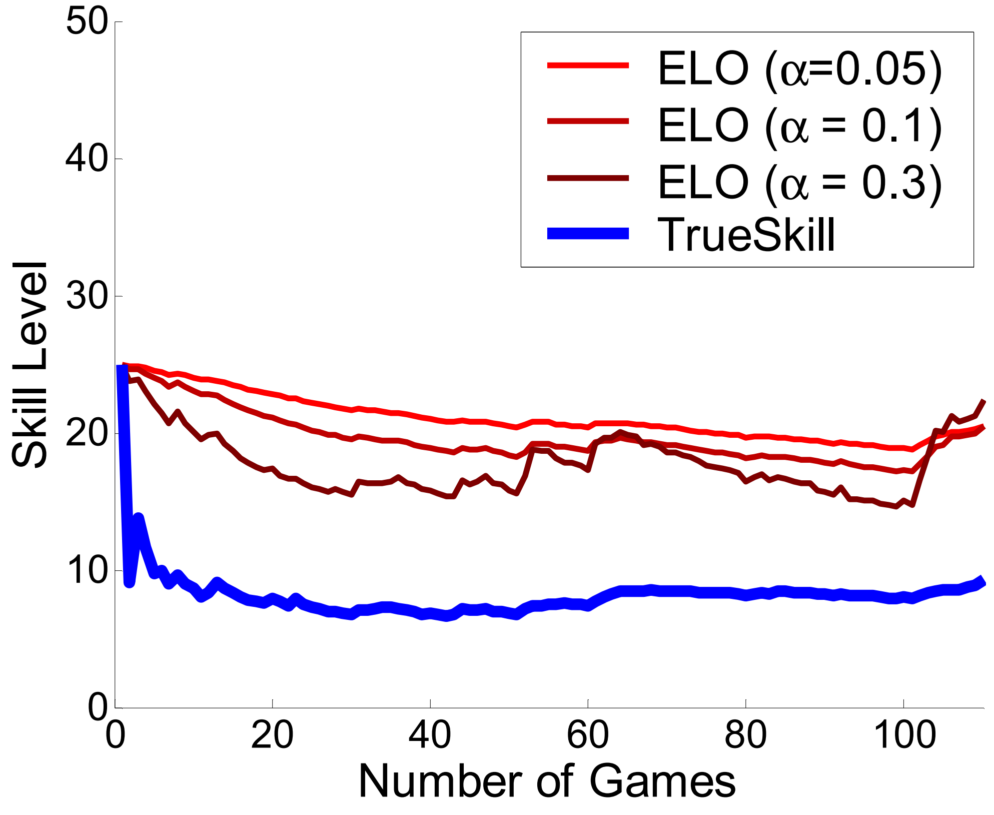
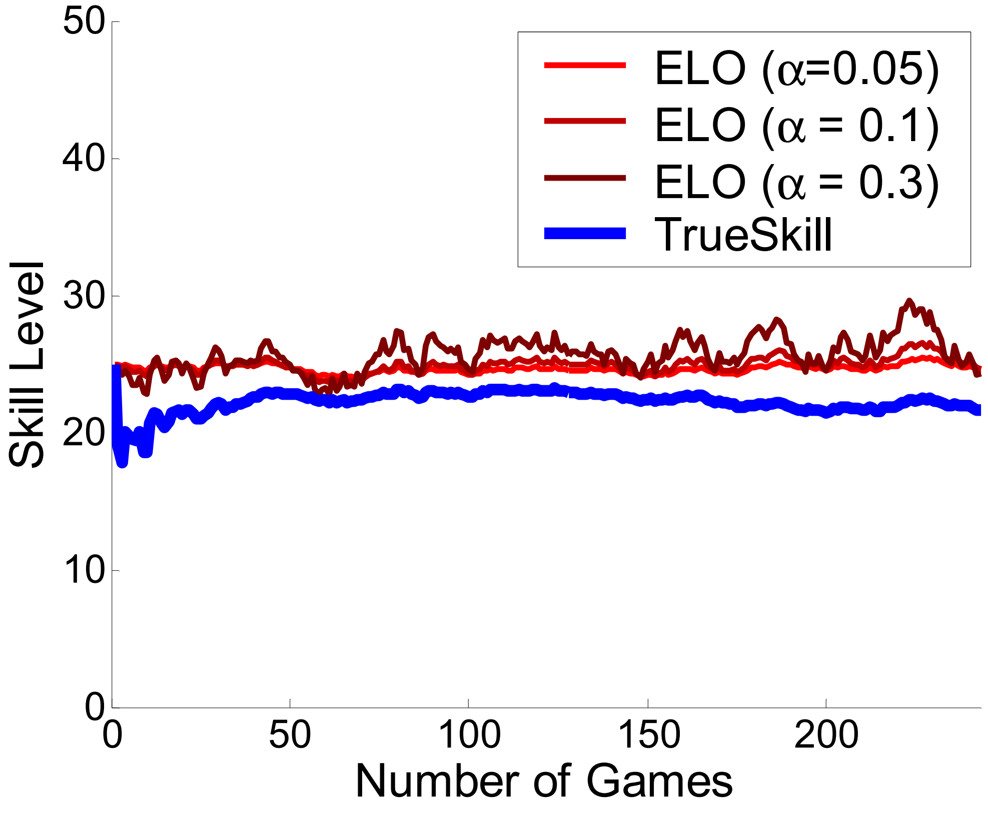

ELO (α=0.05)
ELO (α = 0.1)
ELO (α = 0.3)
TrueSkill
Skill Level
Number of Games
0
10
20
30
40
50
0
20
40
60
80
100

Figure 6



ELO (α=0.05)
ELO (α = 0.1)
ELO (α = 0.3)
TrueSkill
Skill Level
Number of Games
0
10
20
30
40
50
0
50
100
150
200

Figure 7

For Further Reading

- Mark Glickman's chess rating: http://math.bu.edu/people/mg/ratings.html
- Elo dueling heuristic:
  http://www.lifewithalacrity.com/2006/01/ranking_systems.html
- Halo 2
  - Ranking system: http://www.bungie.net/Stats/page.aspx?section=FAQInfo&subsection=stats&page=statoverview
  - Data Set: (http://research.microsoft.com/mlp/apg/downloads.htm)
- TrueSkill
  - Main Page: http://research.microsoft.com/mlp/trueskill/
  - Technical Report: ftp://ftp.research.microsoft.com/pub/tr/TR-2006-80.pdf

The Authors

Ralf Herbrich and Thore Graepel are researchers at Microsoft Research Cambridge, U.K., where they are heading the Applied Games Group (ApG) which is focused on applying machine learning and Bayesian inference to problems in game AI and games of social interaction. Their group is responsible for a number of innovations in game technology such as the **Drivatar AI in Forza Motorsport** and the development of the **TrueSkill™ Ranking and Matchmaking system** now in operation in Xbox Live. The ApG group is working closely with Bizarre Creations (PGR 3), Rare Ltd. (PDZ), Bungie Studios (Halo 3) and the Games Technology Group at Microsoft.