Beyond Skill Rating: Advanced Matchmaking in Ghost Recon Online

Olivier Delalleau, Emile Contal, Eric Thibodeau-Laufer, Raul Chandias Ferrari, Yoshua Bengio, and Frank Zhang

***Abstract*—Player satisfaction is particularly difficult to ensure in online games, due to interactions with other players. In adversarial multiplayer games, matchmaking typically consists in trying to match together players of similar skill level. However, this is usually based on a single skill value, and assumes the only factor of "fun" is the game balance. We present a more advanced matchmaking strategy developed for *Ghost Recon Online*, an upcoming team-focused First Person Shooter from Ubisoft. We first show how incorporating more information about players than their raw skill can lead to more balanced matches. We also argue that balance is not the only factor that matters, and present a strategy to explicitly maximize the players' fun, taking advantage of a rich player profile that includes information about player behavior and personal preferences. Ultimately, our goal is to ask players to provide direct feedback on match quality through an in-game survey. However, because such data was not available for this study, we rely here on heuristics tailored to this specific game. Experiments on data collected during *Ghost Recon Online*'s beta tests show that neural networks can effectively be used to predict both balance and player enjoyment.**

***Index Terms*—Matchmaking, first person shooters, neural networks, player satisfaction, game balance**

Fig. 1. In *Ghost Recon Online*, players have access to various character classes, with unique powers, weapons and high-tech gear. This opens up a wide array of potential playstyles, that basic skill rating algorithms are unable to fully capture.

I. Introduction

MAKING games appealing to a wide audience is a core objective of modern video games [1]. This objective has been driving a significant amount on research on "player-centered" game design [2]. Most of this research has been focused on adapting games to players individually, e.g. by dynamically generating quests in an online Role-Playing-Game [3], adapting tracks in a racing game [4], or dynamically adjusting the difficulty of an arcade game [5]. However, making the game enjoyable for all players in a multiplayer games requires taking into account player interactions. Those are difficult to control, but a good matchmaking process can increase the chance of players having fun with each other, thus improving player retention [6].

The algorithms described in this paper have been designed for the matchmaking system of *Ghost Recon Online*, an online First Person Shooter (FPS) currently being developed by Ubisoft. In this game players control elite soldiers with modern weapons and high-tech equipment (Fig 1), and teams fight against each other in various game modes. These modes include for instance "Capture" (teams fight to capture and control a given number of points on the map) and "Assault" (one team is defending a position which the other team is attacking). The matchmaking task consists in building teams from a pool of players willing to join an adversarial multiplayer match, in a way that maximizes players' enjoyment[1]. This challenge is traditionally solved by assigning a skill rating to each player (inferred from his previous match results), deriving team ratings from individual skills of all players in a team, then having teams of similar strengths fight each other. This is for instance the basic idea behind the TrueSkill matchmaking system developed by Microsoft for their Xbox Live online gaming service [7]. The motivation is that the game is not fun if a match is unbalanced, as weaker players get frustrated while experienced players get bored (even though getting easy kills may initially be fun).

The research we present here aims to address two limitations of such skill-based matchmaking systems:

- Because skill ratings are often used for player ranking (e.g. online leaderboards) in addition to matchmaking, such ratings are usually uni-dimensional (they result in a single number representing a player's overall proficiency in the game). However, complex games like modern FPS require skills in multiple areas like reflex, planning, tactical analysis or teamwork. The relative importance of these skills depends on the map, game mode (e.g.

Olivier Delalleau, Eric Thibodeau-Laufer, Raul Chandias Ferrari and Yoshua Bengio are with the Department of Computer Science and Operations Research, University of Montreal, Canada.

Emile Contal is with the Department of Computer Science of the Ecole Normale Supérieure de Cachan, France.

Frank Zhang is with Ubisoft Montreal, Canada.

[1]Note that here we focus on situations where only two teams face each other and the game is balanced for teams of equal size, but our approach could be generalized to more generic settings as well.

Deathmatch vs. Capture the Flag), player roles, team compositions, etc. Since in this work our goal is to match players together rather than rank them, we can take advantage of a richer player profile and additional contextual information to predict the game balance, instead of relying on a single skill number.

- Although it seems safe to assume that an unbalanced match is not fun (at least for the weaker team), skill-based matchmaking systems implicitly assume the reverse is also true ("a balanced match is fun"), which is not as obvious. For instance in an FPS, having two teams of campers[2] will most likely lead to a boring match where no action ever happens, even if the match is perfectly balanced.

Our methodology to tackle these challenges consists in using machine learning algorithms (more specifically neural networks) to predict the match winner and a measure of individual player enjoyment. These predictions are based on information about players involved in the match as well as on the match's specific settings. The information on players is derived from historical data, taking into account both previous match results and player attributes collected by tracking player behavior over time. Defining what makes a game fun is an interesting but challenging task that has been a research topic for a long time [8], [9], and we do not intend to solve it here. Instead, we plan to rely on user input, by asking players to regularly provide feedback on their online gaming experience through in-game surveys. However, such survey data was not available yet for this study, so instead we handcrafted a "fun formula" that we used to validate our approach. This formula is based on events tracked during each match (like kills and deaths in an FPS). We will show in our experimental results that our neural network model for fun prediction outperforms skill-based systems like TrueSkill and our own match balance predictor, on the task of finding the matches most likely to be fun for all players involved.

II. Neural Network Models

In this section we describe the neural network architectures we have been using. The two opposing teams are denoted by team A and team B respectively. Note that team order is not random: it is arbitrarily fixed by the map settings, for instance on a given map the team starting from the South area would always be team A, while the team starting from the North area would always be team B. This allows the network to take into account the fact that maps may not be symmetric.

In the following we assume that each team can have up to eight players to keep notations simple, but in general the maximum number of players per team depends on the game mode. Note also that although the matchmaking system (described in Section III) attempts to find matches where teams are balanced and at full size, it may sometimes be forced to start a match with fewer players when not enough players are available.

[2]*Campers* are players who tend to stay still, waiting to ambush enemies.

A. Predicting Match Balance

In order to estimate match balance, we train a neural network whose output is the probability that team A wins over team B (the idea is that a match is balanced when this probability is close to 50%). The network's architecture is shown in Fig. 2.

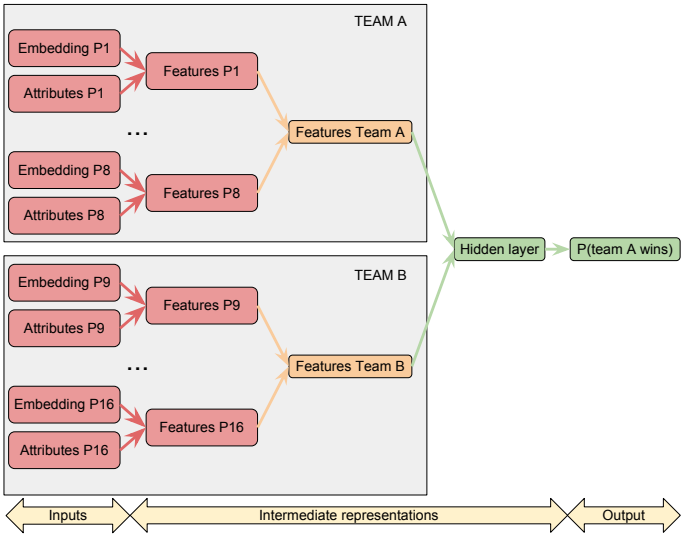

Fig. 2. Neural network computing the probability that team A wins, based on the profiles (embeddings and attributes) of all players involved in the match (players P1 to P8 in team A, and players P9 to P16 in team B).

The network's inputs are the player *profiles*. A profile is the combination of an *embedding* and an *attributes* vectors:

- The n_e-dimensional *embedding* vector $\mathbf{e}_i$ of player i is automatically learned during the training phase, and can be seen as a set of numbers that summarize previous matches in which a player participated. These numbers cannot be easily interpreted by a human, but the neural network can use them to tweak its predictions so that they better match the players' individual playstyles. For instance, in this architecture, the embedding vector is expected to contain information about the player skill in various aspects of the game ("good sniper", "poor assault", etc.).
- The n_a-dimensional *attributes* vector $\mathbf{a}_i$ of player i is a set of normalized statistics that are extracted from the game logs. It contains for instance the average kill / death ratio of the player, the number of matches he played, his firing accuracy, etc.

The input profiles are successively transformed as follows:

1) The profile information (embedding and attributes) are linearly combined into a single vector of *player features*

$$\mathbf{p}_i = \mathbf{e}_i + \mathbf{W}\mathbf{a}_i \tag{1}$$

with $\mathbf{W}$ an $(n_e \times n_a)$ matrix. One may think of these features as a summary of the profile, containing an estimate of a player's skills in various areas of the game, given his history.

2) For each team $j \in \{A, B\}$, *team features* $\mathbf{t}_j$ are computed as the sum of all player features in the team:

$$\mathbf{t}_j = \sum_{i \in \text{team } j} \mathbf{p}_i. \quad (2)$$

3) Team features are compared and summarized by a non-linear transformation into the *hidden layer* $\mathbf{h}$ defined as

$$\mathbf{h} = \tanh\left(\mathbf{b} + \sum_{j \in \{A,B\}} \mathbf{V}_j \mathbf{t}_j\right) \quad (3)$$

with $\mathbf{b}$ an n_h-dimensional vector and $\mathbf{V}_A$ and $\mathbf{V}_B$ two $(n_h \times n_e)$ matrices. Note that here, the $\tanh$ function is applied on a vector: this is a shortcut notation to represent an element-wise $\tanh$ operation on each element of this vector.

4) The last step of the computation is a single sigmoid unit computing the probability α that team A wins by

$$\alpha = \sigma(\mathbf{u} \cdot \mathbf{h} + c) \quad (4)$$

where $\mathbf{u}$ is an n_h-dimensional vector, c is a scalar, and σ is the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$.

The model described above is one of the most basic that fits our approach, but we will see in experiments that it can already yield a significant improvement when compared to a rating-based system like TrueSkill [7]. It is likely that more complex architectures will be able to reach even higher accuracy. Potential improvements include:

- Performing feature extraction on player attributes to extract high-level information on playstyle, e.g. through unsupervised pre-training of deep neural networks [10], [11]. This may be especially useful as more player attributes are added to the player profile.
- Trying more pooling operations than the sum performed in eq. 2, e.g. also concatenating the mean, standard deviation, (soft)min, (soft)max, ...
- Adding additional hidden layers to learn a decision function more complex than eqs. 3-4. Recent work on discriminative deep networks may be useful in this regard [12].

Note also that in order to take map and game mode into account, we propose to learn different matrices $\mathbf{V}_A$ and $\mathbf{V}_B$ (see eq. 3) for each map and game mode. This will allow different maps / modes to favor specific skills, as well as to weigh differently the contributions of the two teams (which may be important in unbalanced maps, or in an asymmetric game mode like "Assault"). However, this strategy could not be evaluated yet because our current dataset is limited to a single map and game mode (see experiments in Section IV-A).

B. Predicting Player Enjoyment

For the purpose of fun prediction, we assume that each match in the training set is labeled with a target vector $\mathbf{f}$ such that f_i is 1 if player i had fun during the match, and 0 otherwise. This label may come from an in-game player survey, or could be computed from prior knowledge on what makes the game fun. Note that some elements of $\mathbf{f}$ may be missing, either because some players skipped the survey in the first case, or because we did not have enough confidence in our "fun estimator" in the second. The neural network architecture we use, depicted in Fig. 3, differs from the one used for balance (Fig. 2) in that it predicts a player-dependent output (the probability that a specific player has fun in the match), instead of a single global value.

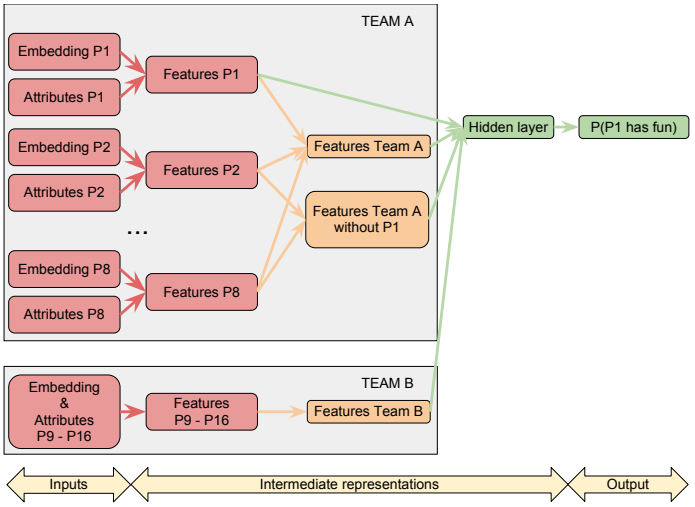

Fig. 3. Neural network computing the probability that player P1 has fun, based on the profiles (embeddings and attributes) of all players involved in the match (players P1 to P8 in team A, and players P9 to P16 in team B). Team B computations are identical to those in Fig. 2 and are not shown in details here. Note that similar networks are defined to compute the probabilities that players P2 to P16 have fun (and all these networks share the same weight parameters).

In particular, in this architecture the hidden layer takes as input the feature vector of the player whose fun is being estimated and the feature vector of the rest of his team, in addition to the feature vectors of both (full) teams. The motivation behind this specific connectivity pattern is that in order to compute in the hidden layer useful information about how likely a player is to have fun, we would like to take into account (i) the player's individual profile, (ii) the profiles of his teammates, and (iii) the global profiles of the two teams.

The inputs (profiles $\mathbf{e}_i$ and attributes $\mathbf{a}_i$) are the same as in Section II-A, and player features are also computed by eq. 1. However, if for instance we want to estimate the fun of player i in team A, the hidden layer $\mathbf{h}$ is now computed by

$$\mathbf{h} = \tanh\left(\mathbf{b} + \mathbf{Y}\mathbf{p}_i + \mathbf{V}_1\mathbf{t}_A + \mathbf{V}_2\mathbf{t}_B + U \sum_{k \in \text{team j}, k \neq i} p_k\right).$$

Note that if player i was in team B, this formula would instead use $\mathbf{V}_1\mathbf{t}_B + \mathbf{V}_2\mathbf{t}_A$. Finally, the output probability $P(\text{Player } i \text{ has fun}) = \alpha$ is given again by eq. 4. The extensions of the balance predictor mentioned at the end of Section II-A can also be considered for this model, in particular matrices $\mathbf{V}_1$, $\mathbf{V}_2$, $\mathbf{Y}$, $\mathbf{U}$ may be learned independently for each unique map and game mode.

III. Architecture

Although the main focus of this paper is on the new machine learning models described above, it is also important

to understand how they are integrated into the game. In this section we briefly answer the three following questions:

- How are players matched together?
- Where does the training data come from?
- How are the model parameters optimized?

A. Matchmaking

Once the models described above have been trained, how should they be used in the matchmaking process? Fig. 4 gives a simplified view of the global architecture (with only 1v1 matches for the sake of clarity). Players who want to join an online match are placed in a queue from which the matchmaking algorithm randomly samples to try various team combinations. Match candidates are scored by one of the neural network models described in Section II to obtain an estimate of match quality. The matchmaking server then launches those with highest scores.

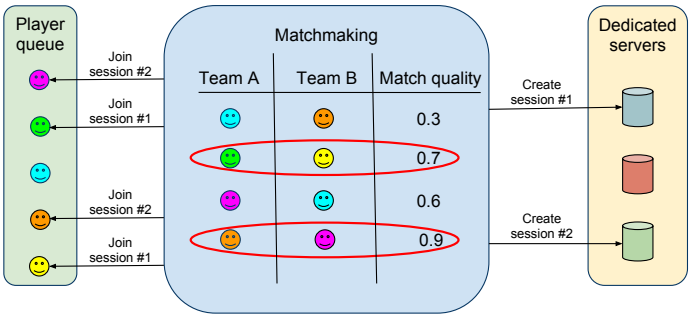

Fig. 4. Sketch of the matchmaking process: various match compositions are evaluated by random sampling from the player queue. The best matches are launched on dedicated servers that chosen players are instructed to join.

Without going too deep into the details, the following points are worth pointing out:

- The sampling strategy has an important role: in particular it can ensure all players are in a compatible skill range and have a good connection with the same dedicated server. It should also favor players who have been waiting for a longer time, to minimize the wait.
- There is a trade-off between sampling time and match quality: we sample as many matches as possible while maintaining matchmaking wait below a given threshold.
- This sampling scheme is well suited to distributed computations, so the number of match candidates that can be evaluated mostly depends on the amount of processing power available.
- A similar approach is used for "hot-join" situations, i.e. when selecting players best fit to fill open slots in ongoing matches.

B. Data Collection

After each match, the game saves into a database all relevant information like which team won, which objectives were completed, who were the players in each team, when they joined and left the game, how many kills they got, how many deaths, etc. These statistics are accompanied by a "snapshot" of the players' state (for all players involved in the game that just ended), which includes additional data like current gear, level, special abilities, etc. Then, a parser reads these logs from the database and generates the corresponding match results and player attributes, which form the basis of the training data.

Another source of data collection is the in-game player survey (which had not yet been activated at the time of writing this paper). This survey pops up after every match (or less frequently if needed), and asks in particular whether (i) the player had fun, and (ii) he thought the match was balanced. The first answer will be used to train and evaluate our model for player enjoyment, while the second one will provide us with another way to compare game balance models. Additional survey questions may also be used for the purpose of player modeling (see Section V-C).

C. Model Optimization

Training is split in two phases, which we call respectively *offline training* and *online update*. Offline training may be slow, and is meant to periodically provide a "fresh" model optimized on a large amount of data, to be deployed for instance during a weekly maintenance window. On the contrary, the online update needs to be fast enough to update the model in real time from the results of matches being played online.

The offline training phase consists in learning two kinds of parameters:

- The parameters governing the network transformations. For instance for the winner prediction model described in Section II-A, this set of parameters is $\{\mathbf{W}, \mathbf{V}_A, \mathbf{V}_B, \mathbf{b}, \mathbf{u}, c\}$.
- The players' embeddings.

We optimize our models by stochastic gradient descent, minimizing the Negative Log-Likelihood (NLL) of the model's prediction [13] (in the fun prediction task, missing targets are ignored). The evolution of player embeddings through time (modeling the fact that we expect players to evolve as they play more matches) is currently considered linear in the number of matches that have been played, i.e.

$$\mathbf{e}_{ik} = \mathbf{e}_i^0 + k\mathbf{e}_i^1$$

where $\mathbf{e}_{ik}$ is the embedding of player i after he has played k matches, and the embedding parameters $\mathbf{e}_i^0$ and $\mathbf{e}_i^1$ are optimized by the gradient descent algorithm. Note that a linear evolution is most likely sub-optimal, and we plan to experiment with other variants in future work.

The online update phase takes place once the model is deployed and new matches are being played. At this point, the network's transformation parameters are kept fixed, but we use the information available from new matches to update the players' embeddings. Whenever a match ends, we recover from the database the composition of the last few[3] matches of all players involved in the match that just ended. The prediction error is then minimized on this small subset of the data, by a fast *conjugate gradient descent* optimization

[3]The number of matches to recover needs to be validated to obtain good performance, both in terms of prediction accuracy and speed.

algorithm [14] optimizing only the players' embeddings. This ensures that embeddings always reflect the recent matches of the players (since if they were kept fixed, or optimized with a slow optimization method like stochastic gradient descent, they would become outdated after a while).

IV. Experiments

A. Dataset

The data at our disposal for this study consists in matches played during an early *Ghost Recon Online* beta-test. All matches were played on the same map and the same multiplayer mode ("Capture"), with up to eight players in each team. After filtering out uninteresting matches (those that are too short or involve less than two players per team), the dataset contains 3937 matches involving 3444 unique players. The histogram of the number of matches per player is shown in Fig. 5. For each player we use the following attributes:

- number of matches played
- sum and mean of kills and deaths
- average kill / death ratio
- sum and mean of number of captures
- TrueSkill skill estimate
- mean and standard deviation of firing accuracy and headshot percentage

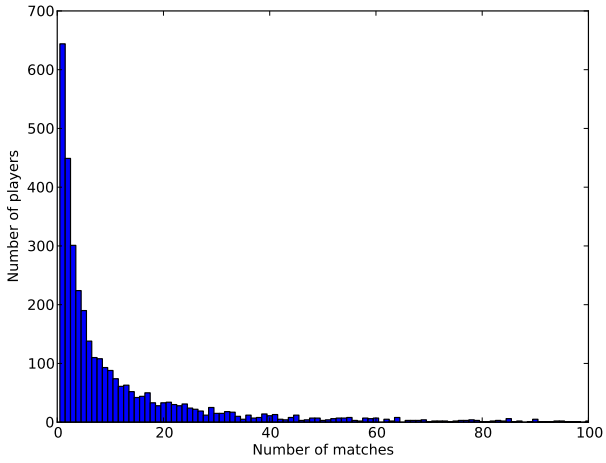

Fig. 5. Distribution of the number of matches per player (truncated to a maximum of 100 matches to keep the figure easy to read – very few players played above 100 matches).

B. Algorithms

In the experimental results that follow, we call *BalanceNet* the neural network model that is trained to predict the probability that team A wins (Section II-A) and *FunNet* the one that predicts the probabilities that players have fun in the match (Section II-B). We compare them to two variants of the TrueSkill algorithm [7]:

- *TrueSkill-Team* only takes match results into account (i.e. "vanilla" TrueSkill).
- *TrueSkill-Player* actually ignores the winning team, focusing instead of individual player performance by ranking players according to their in-game score (a function of their achievements during the match, i.e. for instance kills and captures). This amounts to pretending that an 8v8 match is actually a free-for-all (each player is a team by himself)[4].

Each algorithm has a number of hyper-parameters that need to be set carefully. For instance, in TrueSkill the β parameter (that gives the expected variability in a player's performance) and the dynamic factor τ (that ensures skills can evolve over time) can make a significant difference in terms of performance. Our algorithms' most important parameters are the learning rate in stochastic gradient optimization, and capacity-related quantities that help fight overfitting: sizes of the embeddings (n_e) and of the hidden layer (n_h), and weight decay coefficients (we use ℓ_2 regularization on the network matrices and on embeddings, with a separate regularization coefficient for the online update phase). We use a "brute-force" approach to model selection that consists in training a large amount of model variants with randomly chosen hyper-parameters (after running preliminary experiments to define sensible ranges). We ensure we are not overfitting on these hyper-parameters by a rigorous sequential validation setup described below. To give a rough idea, the optimal network sizes in these experiments are on the order of 10 for the embedding size, and on the order of 100 for the hidden layer size. The β parameter in TrueSkill had to be set around 10-20, and τ around 5.

Our neural network models are implemented in Python, using the Theano library [15] for efficient computations and gradient-based optimization. For the TrueSkill models, we use pure Python code based on a publicly available C# implementation [16].

C. Experimental Setup

Most previous work on matchmaking and skill rating systems usually evaluate algorithms in either an "online" [7] or a "batch" [17] setting:

- In an online setting, the algorithm starts from scratch and is immediately evaluated in the prediction task (also updating its parameters at the same time after each match result).
- In a batch setting, parameters are first optimized on a training set, then performance is evaluated on a disjoint test dataset, keeping parameters fixed.

These settings do not reflect the way our algorithm is meant to be used, so we rely on a different setup that generalizes them. The algorithm is first trained on a training set (this is the "offline training" phase described in Section III-C), then its performance is evaluated on a disjoint test dataset while also updating parameters after each test match result ("online update" phase). Note that other algorithms can also be evaluated in this setup, for instance the "offline training"

[4]Note that although we will show it helps getting better performance for the purpose of matchmaking, we also argue in Section V that this approach should be avoided for public player rankings, as it would promote selfish play.

step of a purely online learning algorithm like TrueSkill simply consists in updating player skills by going through all matches in the training set[5].

In order to obtain an unbiased estimate of the generalization ability of the algorithms being compared, we use sequential validation with model selection. The first 25% of the dataset (ordered chronologically) is isolated as a base training set which is used to "seed" all algorithms. The remainder of the data is split into five folds, and generalization error is estimated by averaging the test error over folds 2 to 5. The test error on fold $k \in \{2, 3, 4, 5\}$ is computed as follows (Fig. 6 illustrates the process for $k = 3$):

1) Training: train multiple variants of the model ("offline training") on the concatenation of the base training set and folds $1, \ldots, k-2$. Each variant corresponds to a different random choice of hyper-parameters (e.g. learning rate, number of hidden units, embedding size).
2) Validation: evaluate each variant by "online update" on fold $k-1$.
3) Re-training: re-train the best variant ("offline training") after adding fold $k-1$ to the train set.
4) Test: evaluate the re-trained model by "online update" on fold k.

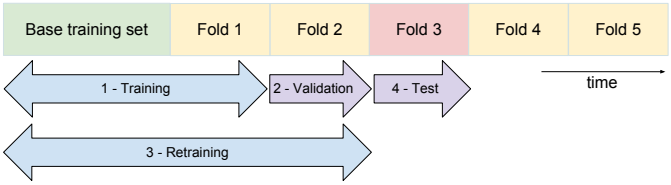

Fig. 6. Five-fold sequential validation with model selection: illustration of the computation of the third fold's test error. Model variants are compared by evaluating their validation error on the previous fold, and the one performing best is evaluated on the test fold after re-training on all data available before this point. Double arrows indicate "offline training" (which may perform optimization based on all matches – past and future), while single arrows represent "online update" (optimization is performed sequentially, one match after the other).

D. Game Balance

We first evaluate the model presented in Section II-A, that predicts the probability that team A wins. In our matchmaking framework, this model is used by preferring matches for which this probability is close to 50%. It is thus important to predict an accurate probability, not just to predict which team will win the match. For this reason, our main criterion for comparison is the Negative Log-Likelihood (NLL). Following [17], we truncate the model outputs so that they remain within the $(0.01, 0.99)$ range: this prevents a model's NLL from growing arbitrarily large when it is too confident in its prediction (which often happened for TrueSkill in our experiments). We also compute the classification error on the winner prediction task since it is easier to interpret, and provides insight on the overall balance of matches in our dataset (note in particular that if all matches were perfectly balanced, then all algorithms would have 50% error).

TABLE I
WINNER PREDICTION TASK (WILL TEAM A WIN THIS MATCH?)

	NLL	Class. Error (%)
TrueSkill-Team	0.547 ± 0.019	26.6 ± 1.9
TrueSkill-Player	0.4785 ± 0.022	21.5 ± 1.8
BalanceNet	$\mathbf{0.457 \pm 0.020}$	$\mathbf{19.2 \pm 1.7}$

Table I presents the results, with 95% confidence intervals (two standard errors). The best results are shown in bold (we performed a paired t-test to verify that *BalanceNet* is significantly better than *TrueSkill-Player* in terms of both NLL and classification error – with p-value of respectively 0.004 and 0.003). It is obvious that *TrueSkill-Team* is much worse than the other two models. This is because it takes longer for skills to converge when they are based only on the match results, compared to *TrueSkill-Player* that has access to direct player rankings through in-game scores. This supports our argument that for the purpose of matchmaking, it is advised to take advantage of individual statistics on players beyond the global result of their teams.

BalanceNet outperforms *TrueSkill-Player*, but the difference is not as striking. We believe the main reasons are related to the current dataset we are experimenting with:

- As can be seen from the low error rate that can be achieved (under 20%), the matches in the dataset are not properly balanced. This is because the matchmaking algorithm in this beta-test was not trying to match players according to their skill. Thus the majority of the matches are significantly unbalanced, making the task easy to solve with simple algorithms[6].
- Data comes from an early beta-test, where most players are still learning the game on their own, without caring much about teamwork. Consequently, there is not much benefit to gain from a model that can deal with team interactions.
- All the data comes from a single map and game mode, and the map is symmetric: although our model is meant to be able to take into account map and game mode specificities, it is not needed here.
- Obviously, 3937 matches is quite small: it is not possible to take full advantage of a high capacity model with this amount of data.

As a result, we expect that as we collect more data (with more variety), our neural network model's advantage over *TrueSkill-Player* will become even more significant.

E. Player Fun

Ultimately, we intend to make player enjoyment the main criterion used in *Ghost Recon Online*'s matchmaking. Note however that we believe a match balance predictor like the one we discussed in the previous experiments would still remain

[5]For the sake of fairness, we added a new hyper-parameter to TrueSkill which is the number of times it iterates on the "offline training" set, in order to potentially let it refine its skill estimates. This did not appear to help much.

[6]As a baseline, a model that simply predicts the winning team as the team with most players achieves 35% classification error.

useful, to (i) act as a safeguard against the fun predictor's mistakes, and (ii) possibly speed up computations (the sampling phase described in Section III-A) by pre-filtering matches in order to fully evaluate only reasonable candidates. Thus we envision a matchmaking system that would take advantage of the unique benefits brought by both approaches.

As mentioned in the introduction, we are setting up an in-game survey to gather player feedback about their gaming experience. While waiting for this data to be collected, we ran experiments by "simulating" a survey, based on our assumptions on what makes the game fun. We did not put a lot of efforts in designing the ultimate "fun formula" since our goal is to replace it eventually, so we used the following process:

- We started by defining 11 features such that we expect a player to have more fun when these features increase. Some of these features are local to the player (e.g. average life span, number of bullets fired, whether he finished the match or disconnected while in progress), some are team-based (e.g. the ratio of our teammates kill/death ratio compared to our own kill/death ratio, capped to 1 in order to mostly catch frustrating situations where we are matched with less skilled players), and finally some are global to a match (e.g. the match duration, and the total number of kills in the match).
- These features were normalized between 0 and 1 so as to obtain a uniform distribution in the (0, 1) range (this basically amounts to using the rank of the value in the sorted list of all observed values for the same feature).
- For each unique player in the dataset, we randomly picked 4 out of these 11 features as those he actually cares about. Then we uniformly sampled 4 weights (rescaled so that they sum to 1) to weigh these features differently. This way, each player has his own individual criteria to evaluate fun in a match (although many of these criteria are correlated).
- The fun factor of each player in each match was computed as this weighted sum of features, then all these fun values were normalized into (0, 1) like we did for individual features.
- Finally, we assumed the player answered "Yes" to the question "Did you have fun in the match" when his fun factor was above 0.7, "No" when it was below 0.3, and skipped the survey otherwise.

Note that although our *FunNet* model predicts individual probabilities for each player to have fun, in the end we need a global match quality score. Based on the assumption that we want everyone to have fun in the match, we interpret our normalized "fun factor" score as the ground truth probability that a player has fun, assume independence among players, and define the match quality by

$$\Pi_{\text{Player } i\in \text{ match}} P(\text{Player } i \text{ has fun}).$$

We actually take the logarithm of this score for convenience, and average the resulting sum to make it independent of the number of players involved, yielding the final formula

$$Score(\text{match}) = \frac{1}{n} \sum_{\text{Player } i\in \text{ match}} \log P(\text{Player } i \text{ has fun}). \tag{5}$$

where n is the number of players in the match.

TABLE II
SURVEY PREDICTION TASK (WILL PLAYER i HAVE FUN IN THIS MATCH?)

	NLL	**Class. Error (%)**
FunNet	0.571 ± 0.008	29.2 ± 0.7

TABLE III
MATCH RANKING TASK (WHICH MATCHES WILL BE MOST FUN?)

	τ **(Kendall's tau)**
TrueSkill-Team	0.11
TrueSkill-Player	0.17
BalanceNet	0.20
FunNet	**0.23**

We first validate that our model is able to predict the survey answers by looking at the NLL and classification error on this target. Table II shows that our model can reach under 30% error, which tells us that it was able to capture at least some of the underlying fun patterns we made up.

Then, we turn to the main question this research is concerned about, which is: can such a model select fun matches better than balance-based models like TrueSkill and *BalanceNet*? To answer it, we use a ranking measure, which has the advantage of being independent of the scale of the models' scores, and of reflecting our real-world application where the goal is to rank candidate matches to find the best ones. Specifically, we compute the Kendall's tau rank correlation coefficient [18], denoted by τ, between the ground truth match score (eq. 5) and the models' rankings on test matches. For the balance-based models (*TrueSkill-Team*, *TrueSkill-Player* and *BalanceNet*) the matches are ranked by increasing value of $|P(\text{team A wins}) - 0.5|$. The *FunNet* model uses eq. 5 with its own estimated probabilities of each player having fun. A perfect ranking would achieve $\tau = 1$, while random ranking corresponds to $\tau = 0$ (and $\tau = -1$ corresponds to ranking in the exact opposite order of the ground truth).

We see from Table III that all models achieve a significantly positive τ, which is not surprising since many of our features that define fun correlate with match balance. We also recover the same ordering w.r.t. performance as in the balance task (Table I) i.e. *TrueSkill-Team* < *TrueSkill-Player* < *BalanceNet*. However, as we expected, there is a benefit in designing a specific "fun predictor" like the *FunNet* neural network, since it is the one that achieves best performance. This is a promising result, but of course it remains to be validated on "true" player feedback, which will be the topic of our future research.

V. RELATED WORK

Our work puts together ideas originating from several fields of research: matchmaking, skill rating and player modeling. We describe below previous work in those areas that is most relevant in the context of the proposed methodology.

A. Matchmaking

The primary concern for matchmaking in an action game is often the network connection quality. This is especially true in an FPS where accurate aiming is key: being able to reliably estimate latency between players is thus very important, and is a topic of ongoing research [19]. This challenge is made easier in our situation because games are run on dedicated servers, so all we need to do is ensure that we only match together players who have a good connection to the same dedicated server.

From a high-level point of view, our matchmaking architecture is in the same spirit as the one described in [20], but with a more complex match selection process. In that work, it is suggested to divide players among "bins" (based on their skill) in order to ensure match balance, and it is not said how to pick players from a bin to obtain the final team composition. This kind of skill-based strategy is used by many games, that do not attempt to globally optimize team compositions. Instead, they match together players of similar skill, then distribute players in teams either randomly or so as to achieve teams of equal strength [6], [21].

The idea that players should be matched based on their gaming profile is not new: [22] showed empirically that different types of players do not share the same preferences with respect to who they enjoy playing with. However, they did not actually propose a specific matchmaking algorithm based on these considerations. [23] describes such a matchmaking system, where they call "role" a player's individual type. In this system, examples of "good" matches are first memorized (where good matches are found for instance by asking feedback from players like we intend to, or by human experts who observe matches). These good matches are analyzed in terms of the roles played by the players involved in them, where roles are manually defined in a subjective manner and may correspond to various traits of players that are considered important for matchmaking purpose (e.g. "sniper", "power gamer", "socializer"). A specific algorithm to infer player role from tracked player behavior is not detailed, but several player modeling techniques have been developed in the past years and may be used for this purpose [24], [25], [26]. When a match needs to be created from a pool of players waiting in the matchmaking queue, candidate matches are then evaluated by being compared to the set of good matches (in terms of similarity in their role compositions). This approach is thus similar to ours, but replacing our neural network evaluation system with a memory-based algorithm and using only roles as input. Although we believe this is a sensible idea worth experimenting with, it has not been actually implemented yet. One difficulty is that it is not obvious which roles are to be defined (one can think of our algorithm as a way to learn roles automatically within the player embeddings). Also, their proposed algorithm only keeps "good" examples, while also taking into account examples of "bad" situations is probably important as well. Finally, they mention the problem of the combinatorial cost of trying all player combinations to find the best match, but do not propose a solution to this issue: we suggest here to solve it by random sampling.

B. Skill Rating

The problem of assigning skills to players or teams has a long history in both games and sports, mostly for the tasks of ranking, matchmaking and outcome prediction. Although all these tasks may be tackled independently, a skill rating system is very appealing as it can provide a statistically motivated answer to all of them. The ranking task, however, imposes some specific constraints that may hurt performance for matchmaking and outcome prediction. Besides the fact that a uni-dimensional skill is needed to easily make comparisons, the competitive nature of rankings also makes them a favorite target of players trying to "exploit" the system [6]. This is one important reason why most skill rating systems only consider match results to compute the skill: accounting for extra information like the attributes we feed to our neural network might be abused by players. This could be very detrimental to team-based games, where players would try to maximize statistics that boost their skill (e.g. their own number of kills or captures in an FPS) instead of doing what is best for their team to win.

A skill rating algorithm meant to be used for matchmaking in a multiplayer game like *Ghost Recon Online* needs to be able to assign individual ratings to players, then to derive ratings for arbitrary teams from these player ratings. This rules out most algorithms used in sports, where typically either only global team ratings are considered [27], or, if individual ratings are sought, players are assumed to play in the same team for a long enough period of time to estimate meaningful correlations [28].

The large majority of skill rating systems developed for games take their root from the Bradley-Terry model [29], that in its basic formulation models the probability that team A wins over team B by

$$\alpha = P(\text{team A wins}) = \frac{s_A}{s_A + s_B}$$

with s_j the skill of team j. If we write $s_j = e^{t_j}$ this becomes

$$\alpha = \frac{e^{t_A}}{e^{t_A} + e^{t_B}} = \frac{1}{1 + e^{t_B - t_A}} = \sigma(t_A - t_B). \qquad (6)$$

Note that if we assume

$$t_j = \sum_{i \in \text{team } j} p_i \qquad (7)$$

with p_i the individual skill of player i, then this is a variant of our neural network model described in Section II-A. This can be seen when:

- A player's embedding is made of a single scalar (his skill) and there are no player attributes, so that eq. 1 becomes $p_i = e_i$, and thus team features (eq. 2) are scalars computed as in eq. 7.
- The hidden layer $\mathbf{h}$ (eq. 3) is simplified to be the concatenation of the team features, i.e. $\mathbf{h} = (t_A, t_B)^T$.
- The parameters of the output probability α (eq. 4) are $\mathbf{u} = (1, -1)^T$ and $c = 0$, making it equivalent to eq. 6.

The Bradley-Terry model has already been presented in such a neural network form [30], but it was generalized in a different way than in our model. The application the authors

were interested in was in a game where teams were expected to have a significant imbalance (in terms of the number of players facing each other), which led them to model differently the combined player strengths to better account for such large differences. In particular, they incorporate this difference into a so-called "home field advantage" that can also model imbalances resulting from asymmetric maps, and can have a stronger influence on the predicted result when there is a high uncertainty on the player skills (i.e. when many players are new to the game). They also take time into account by weighting a player's contribution with the time he spent in the match, and using the elapsed time as input so that the winning probability evolves as time elapses. Such an extension would be interesting to incorporate in our model to better evaluate "hot-join" situations. Compared to their formulation, the novelty of our approach lies in using a multi-dimensional embedding rather than a single skill value, adding additional player attributes as input, and having more parameters to the neural network transformations in order to potentially learn more complex functions.

As discussed in Section III-C, the ability to update player ratings after each match efficiently is important for online rating systems, that need to update ratings in real time. The Elo rating system [31], adopted by the World Chess Federation, is very close to the Bradley-Terry model described above and is based on an efficient online update algorithm. The Elo rating was later extended, in particular to model uncertainty [32], eventually leading to the fully Bayesian TrueSkill system that is also able to infer individual skills from team results [7]. Various improvements and variants of TrueSkill have been proposed since then (see e.g. [33], [34], [35]). Such methods differ significantly from ours: they are probabilistic algorithms that model a player's skill as a scalar random variable, and perform inference based only on match results (in particular they ignore player attributes). One research direction that bears resemblance to our work is the idea of computing skill ratings in a "batch" setting, i.e. instead of only updating current ratings incrementally after each match, both past and future ratings are optimized to globally fit all match results available [36], [37][7]. This is also what our offline training phase (described in Section III-C) is meant to achieve: it can "revisit the past", while our online update phase is currently a more myopic (but faster) incremental procedure.

Although using a single scalar to represent player skill is convenient, it has been recently noted that increased performance on the outcome prediction task can be obtained when using additional factors. A first idea, explored by [38] and [39], consists in adding the concept of "contexts" associated to vectors $\boldsymbol{\theta}_k$, such that the skill of player i in context k is given by the dot product $\mathbf{p}_i \cdot \boldsymbol{\theta}_k$. In our FPS application, a context would be for instance a specific map and game mode: we proposed a similar idea in Section II-A by learning context-dependent weight matrices $\mathbf{V}_A$ and $\mathbf{V}_B$ (used in eq. 3). Note that in our model we use a context matrix rather than a vector because we want to extract multiple features rather than a single skill value. Another way to use a multi-dimensional skill vector in a Bayesian setting was presented in [17], whose idea consists in modeling the fact that a player may have strengths and weaknesses in different areas. Our neural network approach is also be able to model such strengths and weaknesses in the embedding vector $\mathbf{p}_i$, which the hidden layer transformation (eq. 3) can combine optimally for outcome prediction. This is all done implicitly here, while in a Bayesian setting the relations between elements of $\mathbf{p}_i$ are explicitly defined by the graphical model architecture.

To conclude the comparison with Bayesian skill rating models, we should emphasize that such models naturally handle uncertainty, since they are fully probabilistic. For instance, evaluating balance with TrueSkill is not usually done by simply looking at $|P(\text{team A wins}) - 0.5|$. Instead, the balance is computed from the asymptotic probability that the two teams perform equally well (i.e. a draw), which depends on the uncertainty on players' skill and performance [7]. On another hand, our current model ignores uncertainty: player embeddings are fixed and the network transformations are deterministic. However, we expect the addition of player attributes (that contain for instance the number of matches already played) to help by indirectly taking into account uncertainty about new players' embeddings.

C. Player Modeling

The basic idea of *player modeling* [2] is to extract information about players, to eventually provide them with an improved gaming experience (either directly – e.g. tuning the game to better suit the player's playstyle – or indirectly – e.g. collecting data to help later improve the game or its sequel). Note that here we only consider models based on players' actions within the game: more intrusive systems based for instance on heart rate monitoring may also bring useful insight into the way players experience video games [40], but are out of the scope of our present research.

Our matchmaking application can be seen as a kind of "game adaptation" mechanism in the context of matchmaking, where the game parameters being tuned are those of the matchmaking decision function. A special case of game adaptation consists in dynamically adjusting the game difficulty to better suit the player's individual skill level [41]. In single player games, dynamic difficulty adjustment is usually based on the analysis of relevant statistics (e.g. number of successes / failures, rate of damage) to adjust game settings during gameplay [42], [43], [44], [45]. Other game adaptation techniques to maximize player enjoyment have been proposed before in other contexts like game content generation and adaptation: [46] provides a good overview of previous work in this area. Although many of these methods share goals similar to our work, (they aim at making the game more balanced and more fun), they cannot be readily applied to matchmaking. The main reason is they are designed for single-player games and thus are not meant to simultaneously optimize the game experience of many players at the same time (especially because of player interactions).

Another link with player modeling consists in the addition of player attributes. In the present research we use simple

[7]Note that the batch approach from [37] can actually be made fast enough for real-time use (with some approximations to speed up computations).

statistics extracted from the game logs, but in the future we intend to add more high-level information about the players' profiles. Such information could for instance be whether the player is more interested in the competitive aspects of the game, in its social interactions, in having casual fun shooting random people, etc. If such "classes" of players can be defined beforehand from a priori knowledge, a survey could be sent to players asking them to identify which class they belong to, or human experts could watch some players and manually label their playstyle. Once a number of such "prototypical" players are available, supervised learning methods can be applied to profile the whole playerbase [25], [47], [48]. Alternatively, unsupervised clustering methods can also be used to discover typical classes of player behavior without much prior knowledge [26], [49]. We expect that adding such high-level profiling of players into their attributes vector will help our predictive models achieve better accuracy.

VI. Conclusion and Future Directions

Our main contributions are as follows:

- We demonstrated that in order to evaluate match balance in a multiplayer game, using a skill value is not enough. There is much to be gained from a richer player profile, in particular by adding player statistics collected within the game.
- We argued that fun is more important than balance, and showed it is possible to use fun as the main criterion in a matchmaking system (to the best of our knowledge, this is the first implementation of this idea).
- We proposed an implementation based on neural networks, which makes it easy to include additional parameters and to design architecture variants able to better suit a game's specific needs.
- We showed how to integrate these neural networks within an online game's matchmaking system, providing solutions to the problems of (i) finding the best team combinations from a pool of players waiting for a match, and (ii) continuously updating the model in real time as new data is being collected.

Our experimental results, although promising, remain preliminary: as more data is being collected during *Ghost Recon Online*'s beta tests, we will be able to better evaluate the proposed models, and experiment with more variants. The main directions we plan to investigate are the following:

- Once enough data from the in-game player survey has been collected, it will be interesting to compare our handcrafted formula of fun with actual player feedback. One question is also how often the survey should be presented to players after launch: we may not even need it if it proves possible to learn a reliable enough predictive model of fun.
- The current set of attributes we are using is very limited. We will augment it with more statistics, as well as with more high level information derived from player modeling.
- With more data, we may be able to take advantage of more elaborate neural network architectures so as to better learn complex statistical dependencies.

Acknowledgment

We would like to thank the *Ghost Recon Online* development team for their support throughout this project. We are also thankful to Frédéric Bernard, Myriam Côté, Aaron Courville and Steven Pigeon for the many fruitful discussions on various aspects of this research. In addition, this work was made possible thanks to the research funding and computing support from the following agencies: NSERC, FQRNT, Calcul Québec and CIFAR.

References

[1] C. Bateman and R. Boon, *21st Century Game Design (Game Development Series)*. Rockland, MA, USA: Charles River Media, Inc., 2005.

[2] D. Charles, M. McNeill, M. McAlister, M. Black, A. Moore, K. Stringer, J. Kcklich, and A. Kerr, *Player-Centred Game Design: Player Modelling and Adaptive Digital Games*, 2005.

[3] L. Shi and W. Huang, "Apply social network analysis and data mining to dynamic task synthesis for persistent MMORPG virtual world," in *Proceedings of the Third International Conference on Entertainment Computing, Eindhoven, The Netherlands, 2004*, ser. Lecture Notes in Computer Science, M. Rauterberg, Ed., vol. 3166. Springer, 2004, pp. 204–215.

[4] J. Togelius, R. De Nardi, and S. M. Lucas, "Towards automatic personalised content creation for racing games," in *IEEE Symposium on Computational Intelligence and Games*, 2007.

[5] O. Missura and T. Gärtner, "Player modeling for intelligent difficulty adjustment," in *Discovery Science*, ser. Lecture Notes in Computer Science, J. Gama, V. Costa, A. Jorge, and P. Brazdil, Eds. Springer Berlin / Heidelberg, 2009, vol. 5808, pp. 197–211.

[6] C. Butcher, "E pluribus unum: Matchmaking in Halo 3," in *Game Developers Conference (GDC 2008)*, 2008.

[7] R. Herbrich, T. Minka, and T. Graepel, "TrueSkill™: A Bayesian skill rating system," in *Advances in Neural Information Processing Systems 19 (NIPS'06)*, B. Schölkopf, J. Platt, and T. Hoffman, Eds. MIT Press, 2007, pp. 569–576.

[8] T. Malone, "What makes computer games fun?" *SIGSOC Bull.*, vol. 13, pp. 143–, May 1981.

[9] G. Yannakakis and J. Hallam, "Towards optimizing entertainment in computer games," *Applied Artificial Intelligence*, vol. 21, no. 10, pp. 933–971, 2007.

[10] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Advances in Neural Information Processing Systems 19 (NIPS'06)*, B. Schölkopf, J. Platt, and T. Hoffman, Eds. MIT Press, 2007, pp. 153–160.

[11] G. E. Hinton and R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, Jul. 2006.

[12] S. Rifai, Y. Dauphin, P. Vincent, Y. Bengio, and X. Muller, "The manifold tangent classifier," in *NIPS'2011*, 2011, student paper award.

[13] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[14] J. R. Shewchuk, "An introduction to the conjugate gradient method without the agonizing pain," Pittsburgh, PA, USA, Tech. Rep., 1994.

[15] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, "Theano: a CPU and GPU math expression compiler," in *Proceedings of the Python for Scientific Computing Conference (SciPy)*, 2010.

[16] J. Moser, "Computing your skill," 2010. [Online]. Available: http://www.moserware.com/2010/03/computing-your-skill.html

[17] M. Stănescu, "Rating systems with multiple factors," Master's thesis, 2011.

[18] Wikipedia, "Kendall tau rank correlation coefficient," 2011. [Online]. Available: http://en.wikipedia.org/wiki/Kendall_tau_rank_correlation_coefficient

[19] S. Agarwal and J. R. Lorch, "Matchmaking for online games and other latency-sensitive P2P systems," in *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, ser. SIGCOMM '09. New York, NY, USA: ACM, 2009, pp. 315–326.

[20] J. S. Tobias Fritsch, Benjamin Voigt, "The next generation of competitive online game organization," in *Netgames 2008*, 2008.

[21] League of Legends, "League of Legends matchmaking," 2010. [Online]. Available: http://na.leagueoflegends.com/learn/gameplay/matchmaking

[22] J. Riegelsberger, S. Counts, S. Farnham, and B. C. Philips, "Personality matters: Incorporating detailed user attributes and preferences into the matchmaking process," in *HICSS*. IEEE Computer Society, 2007, p. 87.
[23] J. Jimenez-Rodriguez, G. Jimenez-Diaz, and B. Diaz-Agudo, "Matchmaking and case-based recommendations," in *Workshop on Case-Based Reasoning for Computer Games, 19th International Conference on Case Based Reasoning*, 2011.
[24] H. J. van den Herik, H. H. L. M. Donkers, and P. H. M. Spronck, "Opponent modelling and commercial games," in *Proceedings of IEEE 2005 Symposium on Computational Intelligence and Games CIG'05*, G. Kendall and S. Lucas, Eds., 2005, pp. 15–25.
[25] A. Tychsen and A. Canossa, "Defining personas in games using metrics," in *Proceedings of the 2008 Conference on Future Play: Research, Play, Share*, ser. Future Play '08, New York, NY, USA, 2008, pp. 73–80.
[26] R. Thawonmas and K. Iizuka, "Visualization of online-game players based on their action behaviors." *Int. J. Computer Games Technology*, 2008.
[27] J. Park and M. E. J. Newman, "A network-based ranking system for US college football," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2005, no. 10, 2005.
[28] J. Piette, S. Anand, and L. Pham, "Evaluating basketball player performance via statistical network modeling," in *MIT Sloan Sports Analytics Conference*, 2011.
[29] R. A. Bradley and M. E. Terry, "The rank analysis of incomplete block designs — I. The Method of Paired Comparisons," *Biometrika*, vol. 39, pp. 324–345, 1952.
[30] J. E. Menke and T. R. Martinez, "A Bradley-Terry artificial neural network model for individual ratings in group competitions." *Neural Computing and Applications*, vol. 17, pp. 175–186, 2008.
[31] A. E. Elo, *The rating of chessplayers, past and present*. Batsford, 1978.
[32] M. E. Glickman, "Parameter estimation in large dynamic paired comparison experiments," *Applied Statistics*, vol. 48, no. 3, pp. 377–394, 1999.
[33] R. C. Weng and C.-J. Lin, "A Bayesian approximation method for online ranking," *Journal of Machine Learning Research*, vol. 12, pp. 267–300, 2011.
[34] S. Nikolenko and A. Sirotkin, "A new Bayesian rating system for team competitions," in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, L. Getoor and T. Scheffer, Eds. New York, NY, USA: ACM, June 2011, pp. 601–608.
[35] J. C. Huang and B. J. Frey, "Cumulative distribution networks and the derivative-sum-product algorithm: Models and inference for cumulative distribution functions on graphs," *Journal of Machine Learning Research*, vol. 12, pp. 301–348, 2011.
[36] P. Dangauthier, R. Herbrich, T. Minka, and T. Graepel, "TrueSkill through time: Revisiting the history of chess," in *Advances in Neural Information Processing Systems*, M. Press, Ed., Vancouver, Canada, 2007.
[37] R. Coulom, "Whole-history rating: A Bayesian rating system for players of time-varying strength," in *Proceedings of the 6th International Conference on Computers and Games*, ser. CG '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 113–124.
[38] L. Zhang, J. Wu, Z.-C. Wang, and C.-J. Wang, "A factor-based model for context-sensitive skill rating systems," in *Proceedings of the 2010 22nd IEEE International Conference on Tools with Artificial Intelligence – Volume 02*, ser. ICTAI '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 249–255.
[39] S. Usami, "Individual differences multidimensional Bradley-Terry model using reversible jump Markov chain monte carlo algorithm," *Behaviormetrika*, vol. 37, no. 2, pp. 135–155, 2010.
[40] A. Drachen, L. E. Nacke, G. Yannakakis, and A. L. Pedersen, "Correlation between heart rate, electrodermal activity and player experience in first-person shooter games," in *Proceedings of the 5th ACM SIGGRAPH Symposium on Video Games*. New York, NY, USA: ACM, 2010, pp. 49–54.
[41] E. Jimenez, "The Pure advantage: Advanced racing game AI." [Online]. Available: http://www.gamasutra.com
[42] R. Hunicke and V. Chapman, "AI for dynamic difficulty adjustment in games," in *Proceedings of AIIDE 2004*, 2004.
[43] P. Spronck, S. I. Kuyper, and E. Postma, "Difficulty scaling of game AI," in *Proceedings of the 5th International Conference on Intelligent Games and Simulation (GAME-ON 2004)*, 2004, pp. 33–37.
[44] K. Harward and A. Cole, "Challenging everyone: Dynamic difficulty deconstructed," in *Game Developers Conference (GDC 2007)*, 2007.
[45] O. Missura and T. Gärtner, "Predicting dynamic difficulty," in *Ninth Workshop on Mining and Learning with Graphs*, 2011.
[46] P. C., T. J., and G. N. Yannakakis, "Modeling player experience for content creation," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 1, pp. 54–67, 2010.
[47] R. Thawonmas and J.-Y. Ho, "Classification of online game players using action transition probabilities and Kullback Leibler entropy," *Journal of Advanced Computational Intelligence and Intelligent Informatics, Special issue on Advances in Intelligent Data Processing*, vol. 11, no. 3, pp. 319–326, 2007.
[48] R. Thawonmas and K. Iizuka, "Haar wavelets for online-game player classification with dynamic time warping," *Journal of Advanced Computational Intelligence and Intelligent Informatics, Special issue on Intelligence Techniques in Computer Games and Simulations*, vol. 12, no. 2, pp. 150–155, 2008.
[49] D. Ramirez-Cano, S. Colton, and R. Baumgarten, "Player classification using a meta-clustering approach," in *Proceedings of the International Conference on Computer Games, Multimedia and Allied Technology*, 2010.

Olivier Delalleau graduated from the Ecole Polytechnique de Paris in 2001 and from the Ecole Nationale Supérieure des Télécommunications de Paris in 2003. He then worked as a Machine Learning research assistant in the Department of Computer Science and Operations Research of the University of Montreal until 2006, when he started a Ph.D under the supervision of Prof. Yoshua Bengio. His current research is focused on Machine Learning applications to video games.

Emile Contal is a Master student in Theoretical Computer Science and Machine Learning at the Ecole Normale Supérieure in France. His interest is the understanding of learning procedures in order to make computers able to solve more and more complex tasks.

Eric Thibodeau-Laufer received his B.S. degree in Math and Computer Science at the University of Montreal in 2011. He is currently a Master student under the supervision of Prof. Yoshua Bengio, at the LISA lab. Current research focuses on collaborative filtering and recommender systems.

Raul Chandias Ferrari received his B.S. degree in Computer Science from University of Montreal in 2011. He is now a M.S. candidate in Computer Science from University of Montreal under the supervision of Prof. Yoshua Bengio. His research interests include Machine Learning and computer vision.

Yoshua Bengio received his Ph.D in CS from from McGill University, Canada, 1991, in the areas of HMMs, recurrent and convolutional neural networks, and speech recognition. Post-doc 1991-1992 at MIT with Michael Jordan. Post-doc 1992-1993 at Bell Labs with Larry Jackel, Yann LeCun, Vladimir Vapnik. Professor at U. Montreal (CS and Operations Research) since 1993. Canada Research Chair in Statistical Learning Algorithms. Fellow of the Canadian Institute of Advanced Research. NSERC chair. Co-organizer of the Learning Workshop since 1998. NIPS Program Chair in 2008, NIPS General Chair in 2009. Urgel-Archambault Prize in 2009. Fellow of CIRANO. Current or previous associate/action editor for Journal of Machine Learning Research, IEEE Transactions on Neural Networks, Foundations and Trends in Machine Learning, Computational Intelligence, Machine Learning. Author of two books and over 150 scientific papers, with over 9400 citations according to Google Scholar and an H-Index of 43 at the end of 2011.

Frank Zhang received his B.S degree in Computer Science from Shanghai Jiao Tong University in 1999. He started working at Ubisoft Entertainment Shanghai Studio in 1999 and moved to Ubisoft Entertainment Montreal Studio in 2000. He is an architect specialized in online services. He oversees the engine implementation to ensure it aligns well with team and studio objectives.