Emulating Human Play in a Leading Mobile Card Game

Hendrik Baier, Adam Sattaur, Edward J. Powley, Sam Devlin, Jeff Rollason, and Peter I. Cowling, *Member, IEEE*

***Abstract*—Monte Carlo Tree Search (MCTS) has become a popular solution for game AI, capable of creating strong game playing opponents. However, the emergent playstyle of agents using MCTS is not necessarily human-like, believable or enjoyable. AI Factory Spades, currently the top rated Spades game in the Google Play store, uses a variant of MCTS to control AI allies and opponents. In collaboration with the developers, we showed in a previous study that the playstyle of human players significantly differed from that of the AI players [1]. This article presents a method for player modelling using gameplay data and neural networks that does not require domain knowledge, and a method of biasing MCTS with such a player model to create Spades playing agents that emulate human play whilst maintaining strong, competitive performance. The methods of player modelling and biasing MCTS presented in this study are applied to the commercial codebase of AI Factory Spades, and are transferable to MCTS implementations for discrete-action games where relevant gameplay data is available.**

***Index Terms*—Artificial Intelligence (AI), Neural Networks, Monte Carlo Tree Search, Digital Games**

I. Introduction

It is now feasible to collect vast quantities of highly detailed data from digital games [2]. This ease of access to gameplay data has given rise to an increased use of data mining approaches during various stages of game development [3]. Simultaneously, Monte Carlo Tree Search (MCTS) has become a popular solution for Artificial Intelligence (AI) in digital games, capable of creating strong game playing opponents [4]. However, the emergent playstyle of agents using MCTS is not necessarily human-like, believable or enjoyable. Creating agents that emulate human behavior can create more immersive experiences [5], inform game design [6], and would also be beneficial for various applications of MCTS in non-game contexts, e.g. simulations, decision support systems or operations research (see [7], [8] for examples). Most methods used to create enjoyable, human-like play to date require substantial game expertise. This article presents a method of biasing MCTS using human gameplay data and machine learning to create agents that emulate human play whilst maintaining strong, competitive performance. While our approach needs some machine learning expertise, it does not require extensive knowledge of the game, which simplifies application to other games.

To demonstrate the method, we apply it to a digital implementation of the card game Spades by AI Factory Ltd., the leading commercial implementation available for Android devices, with more than 6.5 million downloads. In previous work with the game's developers, we analysed human gameplay data from $27,592$ games and showed significant differences in the playstyles of the AI controlled by their current implementation of MCTS [9], compared to human players [1]. We then demonstrated first steps towards emulating human play whilst still maintaining equivalent playing strength to an unbiased agent by integrating a human gameplay model into MCTS [10]. This article extends our previous work in the following ways:

1) We are moving from a simplified simulation of the game to the actual commercial codebase of AI Factory Spades and improving the MCTS agent shipped with the game.
2) We are moving from a linear combination of complex hand-crafted features to neural networks that use only the raw game state as input—reducing programming effort and making our method more readily generalizable to other games.
3) We compare *direct imitation* (simply playing what the gameplay model predicts) to *indirect imitation* (incorporating the gameplay model into MCTS) [11], both in terms of human-likeness and playing strength.
4) We compare the previously used *knowledge bias* technique for indirect imitation [12] with *equivalent experience bias* [13], simplifying parameter tuning.
5) We explore the effects of *delaying* the MCTS bias to reduce its computational cost [14], creating an indirect imitation technique that provides strong human-like play without using increased computational resources or time.

This article is structured as follows. Section II covers the necessary background and related work. Section III

Hendrik Baier, Adam Sattaur, Sam Devlin, and Peter I. Cowling are with the Digital Creativity Labs and Intelligent Games and Game Intelligence Centre for Doctoral Training, University of York, UK. (Email: hendrik.baier@york.ac.uk, as1129@york.ac.uk, sam.devlin@york.ac.uk, peter.cowling@york.ac.uk)

Edward J. Powley is with the Digital Creativity Labs and The MetaMakers Institute, Games Academy, Falmouth University, UK (Email: edward.powley@falmouth.ac.uk)

Jeff Rollason is with AI Factory Ltd., Pinner, Middlesex, UK. (Email: jeff.rollason@aifactory.co.uk)

Manuscript received May 4, 2018. This work was supported by the U.K. Engineering and Physical Sciences Research Council (EPSRC), Arts and Humanities Research Council (AHRC) and InnovateUK under Grants EP/M023265/1 and EP/L015846/1.

then describes our neural network models of human play in Spades, and Section IV their integration into an MCTS agent. The paper ends with conclusions and suggestions for further work in Section V.

II. Background

This section begins with a brief background on the application of MCTS (Subsection II-A) and neural networks (Subsection II-B) in games. In Subsection II-C, we then discuss previous related work on player modelling and imitation learning, to emphasise the place of our contribution within the existing literature. The section ends with a discussion of our prior work and ongoing collaboration with AI Factory in Subsection II-D.

A. Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is the underlying decision-making algorithm used by AI Factory Spades. Since its invention in 2006 [15], [16], MCTS has shown state of the art performance in a number of games, most notably Go [17]. It does not require prior knowledge of how to play a game, although such heuristic knowledge is often used to improve performance [4].

MCTS creates an asymmetrical decision tree, with specific focus on exploring the more promising decisions. The root node of the tree represents the current state of the game. MCTS works by repeating the following four-phase loop until computation time runs out, each iteration of the loop representing one simulated game.

- Selection: Starting from the root, the tree is traversed using a *tree policy* or *selection policy*. The selection balances the exploitation of moves with high value estimates with the exploration of moves with uncertain value estimates.
- Expansion: When the selection policy leaves the tree by sampling an unseen move, its successor state is added as a new node to the tree.
- Playout: A *playout policy*, often random, completes the game from the expanded node to the game's end.
- Backpropagation: The result of the playout is back-propagated through the tree until it reaches the root, improving value estimates at the nodes visited during selection and expansion.

One of the most commonly used MCTS variants is *Upper Confidence Bound for Trees* (UCT) [16]. UCT treats each node in the selection phase as a multi-arm bandit problem. Each node is assigned a value that describes its average expected reward, calculated from previous simulations. The tree policy of the UCT algorithm in each visited node is to select a child node i that maximises:

$$\mathrm{UCT}(i) = \overline{X}_i + C\sqrt{\frac{\ln n}{n_i}} \tag{1}$$

where $\overline{X}_i \in [0, 1]$ is the average expected reward of the child node i, C is an exploration constant, n is the number of times the parent node has been visited, and n_i is the number of times the child node i has been visited. This formula allows control of the balance between exploitation and exploration by setting the constant C. We use a value of $C = 0.7$ throughout this paper.

A wide variety of MCTS variants and enhancements have been investigated [4]. Of particular relevance to this work is *Information Set Monte Carlo Tree Search* (ISMCTS) [18], a variant designed to handle games of imperfect information. The main idea of ISMCTS is that a new determinization of the root game state is created for each simulation. This means simulations are split between different determinizations of the game with a distribution approximating the likelihood of that determinization. The work of this paper and the search algorithm in AI Factory Spades, based on ISMCTS, were developed jointly by some of the authors and AI Factory [9], [19].

B. Neural Networks

Neural networks have been used for various forms of game playing in recent years, often with the goal of creating the strongest agents possible within their domains. For example, Silver et al. [17] used reinforcement learning to train neural networks to play Go. These networks were used to guide Monte Carlo Tree Search, resulting in an artificial player of superhuman strength—improving on their previous program that had beaten some of the best human players in the world [20].

Moravčík et al. [21] used neural networks as an evaluation function for the search performed in heads-up no-limit Texas hold'em poker. The neural networks were trained on random games to provide estimates of the value of holding each card combination in a given game state. Search was performed to limited depth, and leaf states were evaluated by the neural networks as an approximation of the remainder of the search.

Hartford et al. [22] used neural networks to predict the behavior of human participants in two-player, general-sum games with a single simultaneous move, as studied in much of behavioral game theory. Their network architecture is designed to incorporate assumptions of human behavior specific to the application, and to express state of the art models of human bounded rationality.

C. Emulating Human Play

There is a growing body of work which uses gameplay traces to learn a model for predicting the behaviour of a human player, including a well established trend of opponent modeling to exploit weaknesses in their behaviour [23], [24]. Synnaeve and Bessiere [25] predicted the openings of opponents in the real-time strategy game StarCraft by learning the parameters of a Bayesian model from game logs with labeled openings. Dereszynski et al. [26] used replays from StarCraft to learn a Hidden Markov Model for predicting high-level strategic behaviour. These models were integrated into the existing agents in order to improve their ability to predict the strategy of a human player, but the models were not used to bias the agents towards a

human playstyle. In contrast, our focus in this work was the emulation of human play. Maintaining the playing strength of the agent was a secondary objective.

Another area of interest, more closely related to our goal, is developing an agent that imitates a human player. Togelius et al. [11] divide behaviour imitation into two categories: direct and indirect. *Direct imitation* involves using a supervised learning method on a data set that contains a list of states in a game, together with the action that a human player has taken in that state. The resulting predictor for human moves is then used *as the agent*. This can fail in situations that the predictor was not trained on, because the agent is unable to plan or learn during play; for example, an AI agent for a racing game would crash whenever the controller was in a state too unlike those seen before. Despite these problems, there have been a number of attempts to create human-like agents using direct imitation. Thurau et al. [27] used Bayesian imitation learning to teach an agent to navigate through a maze in Quake II using recordings of human players. The results of their experiments showed that the agents appear human-like, but their functionality is limited to imitating movement and would not be able to play against a human player. In this paper we develop an agent that, whilst human-like, can act rationally in state unlike those previously observed, and can safely be deployed within the game.

Alternatively, *indirect imitation* means that a predictor for human moves is used *by the agent*, for biasing an existing decision-making algorithm towards human-like behaviour instead of fully specifying it. This approach can overcome the problem of the agent being unable to act sensibly when in an unfamiliar state, and is the closest to the method we propose. Ortega et al. [5] for example trained agents to play Super Mario Bros using both direct and indirect imitation methods. Their experiments show that the agents using indirect imitation were perceived as significantly more human-like than the agents that used direct imitation, and also performed better. However, they still showed strongly inferior performance compared to humans. Therefore, there is a difficult choice in previous work between maximising the performance of an AI, and making it appear more human-like. As we will show in the remainder of this article, the approach we propose is capable both of playing in a more human-like way and of maintaining an acceptable playing strength comparable to an unbiased MCTS agent.

D. AI Factory Spades

Spades is a four-player trick-taking card game, popular in the USA and played worldwide [28]. It is a partnership game, with North and South in coalition against East and West, and has some similarities with, but slightly simpler rules than, the game of Bridge. A game is played across multiple *rounds* where each partnership receives a score at the end of each round. After being dealt a 13 card hand from a 52 card deck, the players begin the round by providing a *bid* on how many *tricks* they expect to take from their hand that round. Each trick consists of each player in turn playing a card out onto the table, matching the suit of the first (*leading*) player if they can, and otherwise playing any card they choose. The winning card is the highest one matching the leading suit, unless any ♠ card was played, in which case the highest ♠ wins. The winner of the trick becomes the leader of the next trick. The round ends when all cards have been played. Scoring depends on matching the total bid of a partnership as precisely as possible. The full rules are given in [9].

AI Factory[1] is a UK-based independent game developer with over 100 million downloads to date, currently specialising in implementations of classic board and card games for Android mobile devices. AI Factory's implementation of Spades has been downloaded more than 6.5 million times, with an average review score of 4.4/5 from almost 200,000 ratings on the Google Play store[2]. The game is a single-player implementation, in which the user plays as South with an AI partner as North, against two AI opponents as East and West.

The results reported in this paper are based on data collected over a time period of approximately seven months, from a random sample of 690 unique players, consisting of 27,592 full games and 1,356,082 individual tricks in total. Each trick contains three AI plays and one human play.

In previous work, we collaborated with AI Factory to implement ISMCTS-based AI players [9] and then analysed the data collected to gain insights into the playstyles of both the AI and human players [1]. One conclusion was that in certain contexts the AI players were acting significantly different to human players, as evidenced by differences in the distributions of *abstract moves* chosen given the game state [1]. Abstract moves are a hand-coded categorization of the specific cards played in a given trick into more abstract types of moves, using extensive game knowledge and depending on the current state of the game. Examples of abstract moves are "follow suit, and play a card of lower rank than the current highest card in the trick", or "fail to follow suit, instead playing a ♠ card", or "begin a trick with a non-♠ card". Abstract moves are further refined into the different suits, and incorporate game-relevant information such as whether the played card is the highest or lowest in its category. There are 68 possible abstract moves in total.

Assuming that a more human-like AI would be more enjoyable [29], [5], [30], we proposed an indirect imitation solution with the dual aims of:

1) Reducing the differences between the move choices of human and AI players;
2) Maintaining a comparable playing strength to the existing AI players, which was considered appropriate given the previous analysis of player win rates [1] and reviews of the game on the Google Play store.

In our previous work on emulating human play [10], we completed a feasibility study of this approach on a vanilla ISMCTS agent in a basic implementation of the

[1] http://www.aifactory.co.uk

[2] https://play.google.com/store/apps/details?id=uk.co.aifactory.spadesfree&hl=en

core mechanics of the game. This article extends that work by now transferring the findings from the feasibility study to the ISMCTS agent deployed in the commercial game, while also improving on the human play model with the help of neural networks. In the next section we describe our approach to modelling human gameplay, which we then use in Section IV to bias the ISMCTS player.

III. Direct Imitation - Modelling Human Play with Neural Networks

The first part of our work was creating neural network models of the way humans play Spades.

Early results from neural network models seemed promising. The first network we created was a feedforward network with one hidden layer of 200 units and rectified linear activations. The inputs given to the network were:

1) The *abstract moves* that were available for the player, given their cards in hand and the cards on the table, encoded as a binary vector of length 68;
2) The cards of the other three players already on the table, encoded as three binary vectors of length 52. These vectors are "zero- or one-hot", depending on whether the player has played a card yet in this trick.

The network was trained to predict which abstract move would be played by a human given this game state, as the maximal element of an output vector of length 68. We separated the data set into a training/validation set consisting of 90% of the data (1,220,473 individual tricks), and a test set of 10% (135,603 tricks). This network achieved an accuracy of 67.8%, which was already a large improvement on the previous work: the accuracy in [10] was 45.0%, and with decision trees was 57.0% [1]. This improvement comes despite the fact that the previous models were only allowed to select *legal* abstract moves, whereas the neural network had no such restriction. For comparison, always guessing the overall most common abstract move would yield 16.5% accuracy.

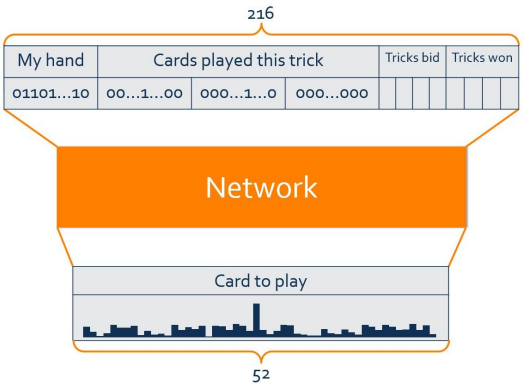

Fig. 1: Inputs and outputs of the networks predicting raw cards, with tricks inputs (`t`).

The success of the network trained to predict abstract moves led us to experiment with networks that would predict exactly which card to play, given only game state information but no game knowledge. These take as input four binary vectors of length 52: the first represents the cards in the player's hand, and the other three represent the cards on the table. This representation does not make use of any hand-coded abstractions of the game state or moves, and does not identify which cards in the player's hand are legal plays. The number of tricks each player bid for the current round, and the number of tricks obtained so far, each represented as a scalar value, were also considered as additional inputs to the networks. Each network outputs a vector of length 52, whose maximum element represents the predicted card. See Figure 1 for a simplified view of the network structure.

Simple feedforward networks do not learn general relationships between sets of cards (for example, that in Spades, consecutive cards in hand of the same suit are essentially identical). For this reason, convolutional network structures were explored as well. As stated above, inputs to the network that represented the cards in the player's hand and the cards played by the other three players were given as binary vectors of size 52. To allow the AI players to spot patterns in their hands—for example, series of consecutive cards—we trained networks that performed one-dimensional convolutions over these inputs, using two kernels of size four, and stride one. We also trained networks that appended the original input to the convolved input, to make sure that crucial information was not being lost by the convolution process.

Various different architectures were trained. We use the following naming scheme:

- The prefix `t` indicates that the input is augmented with the numbers of tricks bid and won by each player. Networks without these inputs use no prefix.
- The following part of the name conveys the internal structure. Numbers, e.g. `200→100`, represent layers of hidden units, listed in depth order, the last being the closest to the output layer. An added letter `c` denotes a convolutional layer. Separate convolutions are applied to the parts of the input vector corresponding to each player. The letter `A` means that the result of the convolution is concatenated with the original, unconvolved input before being passed to the next layer.

The networks were trained using mini-batch gradient descent. Cross-entropy loss was the objective function to minimize [31]. We used TensorFlow version 1.0.1.

We tested a selection of different network architectures for accuracy and playing strength. Accuracy results are based on the test set of 135,603 human moves, and in the rest of this article—unlike in previous work [1], [10]—refer to the accuracy in predicting the specific move made, not just a game-specific abstraction of it.

Table I shows the results. Networks that used cards as inputs and outputs were more accurate those that used abstract moves (67.8%). Using abstract moves requires more domain knowledge and therefore implementation time on the part of a developer, thus is less easily generalizable than the neural network approach. In general, the networks achieved around 70% accuracy on the test set. Those that

TABLE I: Performance of the different networks, direct imitation. Direct imitation uses the network to choose moves.

network	accuracy	win rate vs. baseline	win rate vs. 50 rollouts
`200`	68.3%	11.5%	45.5%
`200→100`	68.2%	11.4%	43.2%
`t-200`	70.5%	17.5%	55.3%
`t-200→100`	71.3%	18.1%	60.0%
`t-200→150→100`	71.1%	22.0%	58.9%
`t-c→200`	70.5%	18.4%	57.9%
`t-c→200→100`	71.0%	14.0%	56.7%
`t-cA→200`	71.1%	20.0%	57.4%
`t-cA→200→100`	70.9%	16.6%	55.3%
ISMCTS baseline	58.2%	(50.0%)	87.5%
random player (legal)	44.0%	0.0%	0.4%

used the additional inputs (tricks bid and tricks won) performed better than those that did not. Deeper networks were able to perform slightly better, but returns quickly diminished after two hidden layers. The convolutional networks did not seem to perform any better than the feedforward networks.

Strength (win rate) results in the middle column of Table I are based on 1,000 games with the tested network playing North and South, against the ISMCTS baseline playing East and West. The ISMCTS baseline is the standard AI player of commercial AI Factory Spades, using 2,500 playouts per move, and a number of game-specific heuristic improvements in the determinizations, playouts, and move selection.

The right column of Table I gives some context on the strength of our networks as standalone players. It shows the results of playing against a weakened version of the baseline agent, using only 50 rollouts per move instead of 2,500, but still using the various hand-coded heuristics of the agents in AI Factory Spades. All networks that include tricks bid and taken in their inputs (`t-`) are significantly stronger than this agent. However, all networks are still clearly inferior to the regular 2,500-rollout baseline (middle column), due to the purely reactive decision making and lack of long-term planning that is typical for direct imitation. We address this in the next section.

It is important to note that due to the nature of the data, achieving 100% prediction accuracy is infeasible. If a player holds 7♡ and 8♡ for example, the choice of which to play is almost entirely arbitrary, yet our accuracy measure will penalize the network for choosing differently from the human. Furthermore there is often no single "correct" or "human-like" move, and two different people may well make different moves given the same in-game situation. Inspection of the data set shows that players do indeed disagree with each other. Less than 10% of the game states in the data set are duplicated (and this decreases further if tricks bid and won are included as part of the state). Most of the examples that share the same game state also lead to the same card being played; however, for any given state, on average 20% of the corresponding examples disagree with the move made by the majority. On the other hand, the fact that our better network architectures achieve above 70% accuracy indicates that there exist strong trends underpinning how humans play the game, and that our models are able to learn these.

Also noteworthy is the fact that the networks very rarely pick illegal moves, such as cards that cannot be played, or cards that the player does not have. In cases where the card valued highest by the network is an illegal move, the choice is replaced by the highest-valued *legal* card (and by a random legal card in cases where all legal cards are valued equally). But only in 0.05% of cases is the highest-valued card actually illegal (as measured for network `t-200→150→100` in more than half a million tricks). This indicates that the networks have learned a highly accurate model of move legality in Spades, despite not being explicitly trained to do so and not being given information on the sets of legal and illegal moves during training. Consequently, enforcing legal outputs does not significantly affect the reported accuracy values.

The lack of increase in accuracy in the larger, more complex networks may be due to the data: there is some level of inconsistency between players. We are only training a single model of human gameplay, but we are basing it on data gathered from a diverse set of players. Another explanation could be that most human players in our dataset might not exhibit significantly more complex strategies than those that can be learnt by a simple feedforward neural network. Training on a sufficient amount of *expert* gameplay data may give different results, and remains as future work.

IV. Indirect Imitation - Biasing MCTS with Neural Networks

Armed with our neural network models of human play, we experimented with two different ways of incorporating them into the ISMCTS player, as described in Subsection IV-A. We then demonstrate a tradeoff between prediction accuracy on the human data set and playing strength for both techniques in Subsection IV-B, and identify promising parameter settings to balance the two. Subsection IV-C introduces *delayed bias* to enable a tradeoff between performance—both in accuracy and in playing strength—and computation time, resulting in a variant of indirect imitation that is competitive without costing additional time. Finally, in Subsection IV-D, direct imitation and indirect imitation are compared directly, demonstrating the effectiveness of delayed neural network bias for creating more human-like play than our baseline, at comparable playing strength and without sacrificing speed.

A. Biasing Techniques

A range of techniques have been proposed for using prior knowledge in MCTS [32]. The first way of integrating the neural network model used here was applied in our previous work [10] and is based on *knowledge bias*, a technique used previously by Ikeda and Viennot [12] for the game of Go.

For this technique, the formula for calculating the UCT score in the baseline ISMCTS agent is modified to include a bias term based on the Bradley–Terry value. The modified UCT score of a node i is calculated as:

$$\mathrm{UCT}(i) = \overline{X}_i + C\sqrt{\frac{\ln n}{n_i}} + C_{\mathrm{BT}}\sqrt{\frac{K}{n+K}}P(m_i) \qquad (2)$$

where $P(m_i)$ is the probability of the move leading to the node i being chosen by a human player (estimated by normalizing the output of the neural network such that it sums up to 1 over all legal moves), C_{BT} is a parameter controlling the influence of the bias on the UCT score, and K is a parameter controlling the rate at which the influence of the bias decreases with the number of parent node visits. These two parameters are tuned experimentally.

The second way of integrating the model of human play into ISMCTS is based on a technique by Gelly and Silver [13]. It biases the selection of nodes in the tree by integrating prior knowledge in the form of *equivalent experience*, meaning that we do not initialize a new tree node i with a visit count of $n_i = 0$ and an average reward of $\overline{X}_i = 0$. Instead, heuristic knowledge about the represented state is added to the node before sampling any rollouts from it, in a similar way as results from a given number of such rollouts would be added—*equivalent* to a given amount of *experience*. For a *subset-armed bandit* as used in ISMCTS, where visit counts n_i and availability counts n_i^{avail} are maintained separately [18], this means

$$\begin{aligned} \overline{X}_i &= P(m_i) \times w_{\mathrm{prior}} \\ n_i &= w_{\mathrm{prior}} \\ n_i^{\mathrm{avail}} &= m \times w_{\mathrm{prior}} \end{aligned} \qquad (3)$$

where $P(m_i)$ as above is the probability of the move leading to node i according to the network—our heuristic knowledge—, w_{prior} is the number of rollouts this knowledge is equivalent to and controls the influence of the bias on the UCT score, and m is the number of legal moves available in the parent node of node i.

B. Accuracy/Strength Tradeoff

In our first set of experiments, we explored the parameter landscape of the knowledge bias and equivalent experience bias techniques. For knowledge bias, we tested $C_{\mathrm{BT}} \in \{0, 0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1, 2\}$ and $K \in \{10, 20, 50, 100, 200, 500, 1000, 2000\}$. For equivalent experience bias, we tested $w_{\mathrm{prior}} \in \{1, 2, 5, 10, 20, 50, 100, 200, 500, 1000\}$. All players were allowed 2,500 rollouts per move, as in the baseline commercial player. The neural networks were called in every node at the time of expansion of its first child node, in order to have their prediction of human play available for biasing every selection step of ISMCTS. As one of our strongest networks according to Table I, `t-200→150→100` was used in all experiments.

For visualization, we use 10, 100, and 1,000 as representative values of K. For these settings and for all tested values of C_{BT} and w_{prior}, Figures 2 and 3 show for both biasing techniques the results of predicting human play on the 135,603 tricks in the test set (accuracy), as well as the results of playing North and South against the ISMCTS baseline playing East and West in 1,000 games per condition (win rate). The figures demonstrate a tradeoff between accuracy on the one hand, and playing strength on the other hand. Lower values of C_{BT} (for knowledge bias) or w_{prior} (for equivalent experience bias) give less influence to the neural network model on tree selection, leading to lower human-likeness, but undiminished playing strength compared to the baseline. Higher values of C_{BT} or w_{prior} give a higher weight to the neural networks and therefore predict human moves increasingly better, but also lead to a drop in playing performance. This is because as C_{BT} or w_{prior} increase, indirect imitation gradually turns into direct imitation, with the network outputs eventually overruling ISMCTS search results. As a result of the lack of planning, we approach the low direct imitation win rate shown in Table I.

A promising parameter setting for further experiments with knowledge bias is $C_{\mathrm{BT}} = 0.2$ and $K = 1000$, which achieves nearly maximal accuracy while not yet playing significantly weaker than the baseline. Note that previous work observed the same for $C_{\mathrm{BT}} = 0.03$ and $K = 1000$ [10], but for prior knowledge of a different nature. It is likely that our neural networks are effective at higher C_{BT} weights because they are more accurate models of human play. A promising parameter setting for equivalent experience bias, based on similar reasoning, is $w_{\mathrm{prior}} = 10$.

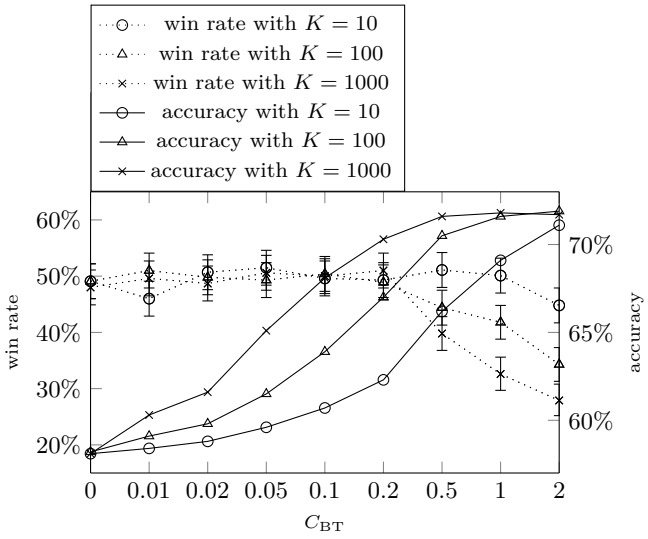

Fig. 2: Tradeoff between accuracy and win rate for knowledge bias, network `t-200→150→100`. Network is used to bias every expanded node.

C. Performance/Speed Tradeoff

All experiments in the previous subsection used 2,500 rollouts per move for all players, just like the baseline ISMCTS agent. This however does not take the additional time into account that the indirect imitation players need to call their neural network models. On a laptop computer with an Intel Core i7-3667U CPU at 2 GHz, for example,

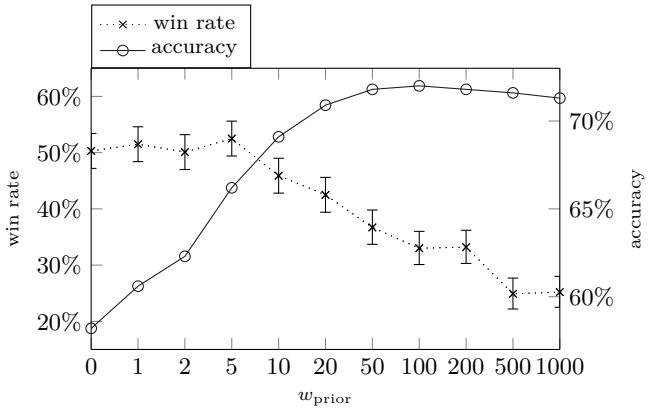

Fig. 3: Tradeoff between accuracy and win rate for equivalent experience bias, network `t-200→150→100`. Network is used to bias every expanded node.

the baseline needs about 34 ms on average to make a move, while the indirect imitation players need about 85 ms. This small difference for powerful hardware becomes highly significant when we consider Spades running on a mobile phone, and would result in unacceptable delays. A 2.5-fold increase in computation time would also restrict the applicability of our methods to other games.

In order to tackle this issue, we implemented *delayed bias* for both biasing techniques. Delayed bias means that the network is not called in every node at the time its first child node is expanded, but only in nodes that have exceeded a certain minimum number d of visits. Until this number of visits is reached, selection in any given node uses unbiased UCT scores. $d = 1$ for example means that the network is called at the second visit to a node, when its first child is expanded—as in the previous subsections; $d = 2$ means it is called at the third visit, etc. This restricts the influence of the human gameplay model to a subset of more relevant nodes in the tree, ignoring those nodes that are revisited rarely due to their less promising value estimates. The technique is similar to one proposed for the game of Go in [14].

Table II shows the effects of the delay parameter d on both knowledge bias and equivalent experience bias. The table demonstrates that the more the neural network calls are delayed, the faster an indirect imitation player can search with a fixed number of rollouts ("Time" columns), because fewer network calls are executed ("Calls" columns). Bias delayed by 1 visit ($d = 1$) leads to the type of player we tested in the last subsection; bias delayed by more than the number of playouts used ($d = \infty$) leads to the baseline ISMCTS agent. The "Rollouts" columns show how many rollouts a player with the given d can execute in order to play at roughly the same speed as the baseline ISMCTS player with 2,500 rollouts.

These data enabled us to do experiments on the tradeoff between performance and playing speed for our indirect imitation agents with delayed bias. For $d \in \{1, 2, 5, 10, 20, 50, 100, 200, 500, 1000, 2000, \infty\}$, we tested the accuracy on predicting human play and the win rate against the baseline of delayed knowledge bias and equivalent experience bias. Using the parameter settings identified as promising in the last subsection, we compared two different conditions: one where both the indirect imitation player (playing North and South) and the baseline (playing East and West) use 2,500 rollouts per move as in the previous subsection, and one where the indirect imitation player uses the number of rollouts indicated in Table II in order to play with comparable speed to the baseline.

Figures 4 and 5 show the results for the two biasing techniques. Two observations can be made. First, delayed bias with $d > 1$ leads to the expected improvement in playing strength at equal time, and the technique is not very sensitive to its parameter d in Spades. As d is increased, strength at equal time quickly catches up to strength at equal rollouts. Second, the accuracy and therefore human-likeness of the delayed bias agents is relatively independent of how often the neural network model is used per move search, as long as it is used at all (it is not used only in the $d = \infty$ conditions).

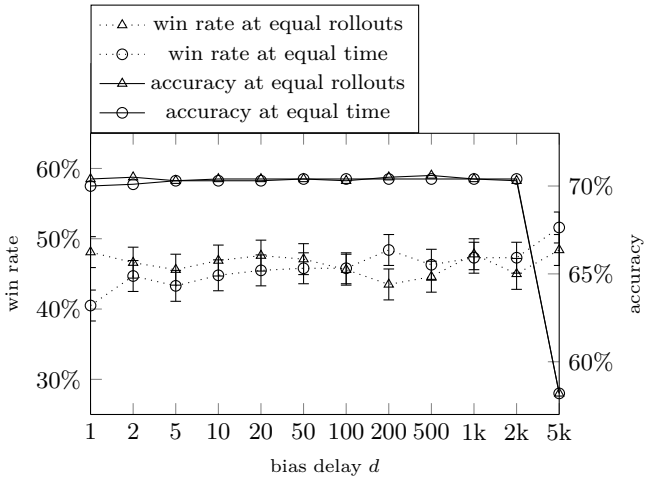

Fig. 4: Tradeoff between performance (accuracy and win rate) and speed for delayed knowledge bias. Network `t-200→150→100`, $K = 1000$, $C_{\text{BT}} = 0.2$. "Equal rollouts" conditions all use 2,500 rollouts per move. "Equal time" conditions all use the same time the baseline needs for 2,500 rollouts per move.

This last observation led us to decide on the final design of our indirect imitation agents. By calling the neural networks only once for each move search—in the root node—we may gain both accurate emulation of human play, and good playing speed. Figures 6 and 7 show the results of repeating the parameter exploration of Figures 2 and 3 when using the network models only in the root node instead of every expanded node. Playing strength and accuracy are very similar—but now based on using the same computation time as the baseline agent at 2,500 rollouts per move.

As a final remark on these data, it appears that equivalent experience bias and knowledge bias achieve similar performance to each other if tuned correctly. Equivalent experience bias at $w_{\text{prior}} = 10$ wins 46.5% of games against the baseline and has an accuracy of 69.6%; knowledge bias

TABLE II: Speed of delayed knowledge bias and equivalent experience bias, network `t-200→150→100`. Knowledge bias uses $C_{\mathrm{BT}} = 0.2$ and $K = 1000$; equivalent experience bias uses $w_{\mathrm{prior}} = 10$. "Time" is the average time taken per 2,500-rollout move search; "Calls" is the average number of calls to the network per 2,500-rollout move search; "Rollouts" is the number of rollouts per move that achieves equal playing speed to the baseline at 2,500 rollouts per move.

bias delay in visits	knowledge bias			equivalent experience bias		
	Time	Calls	Rollouts	Time	Calls	Rollouts
1	85.56 ms	343.17	989	83.71 ms	339.69	1031
5	63.66 ms	173.70	1329	79.44 ms	305.27	1086
20	45.88 ms	59.25	1844	49.97 ms	88.86	1727
100	38.71 ms	13.53	2185	38.81 ms	13.64	2223
500	35.68 ms	3.36	2371	36.53 ms	3.17	2362
2000	35.46 ms	1.64	2386	35.37 ms	1.52	2439
∞ (no bias)	33.84 ms	0	2500	33.84 ms	0	2500

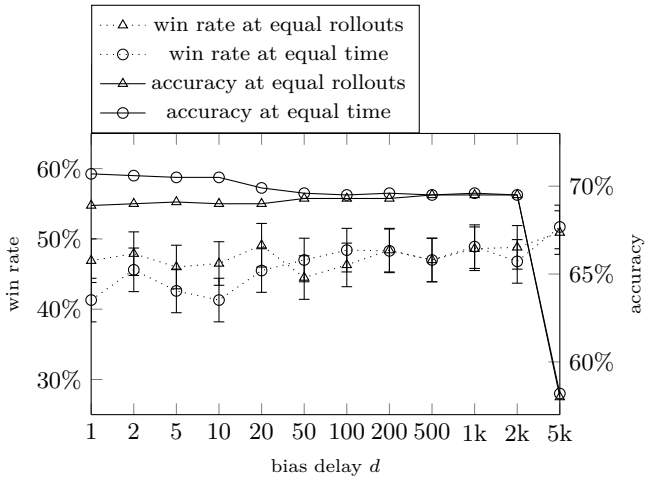

Fig. 5: Tradeoff between performance (accuracy and win rate) and speed for delayed equivalent experience bias. Network `t-200→150→100`, $w_{\mathrm{prior}} = 10$. "Equal rollouts" conditions all use 2,500 rollouts per move. "Equal time" conditions all use the same time the baseline needs for 2,500 rollouts per move.

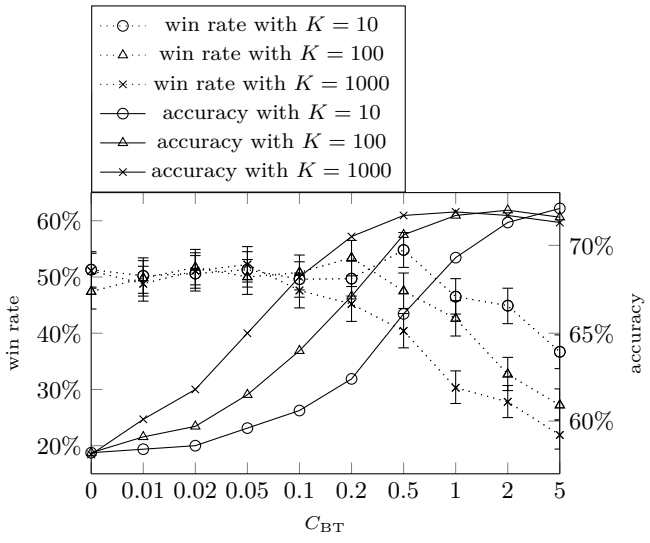

Fig. 6: Tradeoff between accuracy and win rate for knowledge bias, network `t-200→150→100`, at equal time. Network called in the root node only.

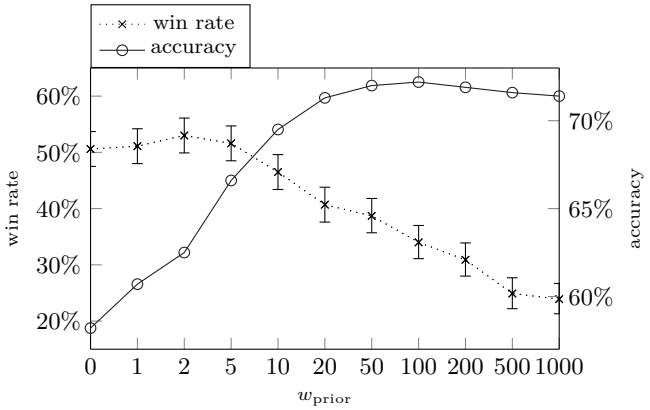

Fig. 7: Tradeoff between accuracy and win rate for equivalent experience bias, network `t-200→150→100`, at equal time. Network called in the root node only.

at $C_{\mathrm{BT}} = 0.2$ and $K = 1000$ wins 45.2% and is 70.3% accurate, at $C_{\mathrm{BT}} = 0.5$ and $K = 100$ wins 47.5% and is 70.5% accurate, and at $C_{\mathrm{BT}} = 1$ and $K = 10$ wins 46.5% and is 69.6% accurate. Since equivalent experience bias has only one tunable parameter compared to the two of knowledge bias, it may be the preferable method for biasing MCTS with heuristic knowledge. However, we are using knowledge bias in the following subsection because of its slightly higher accuracy at the specific parameter settings we tested.

D. Comparison of Direct Imitation and Indirect Imitation

In our last set of experiments, we compared our indirect imitation agents from the last subsection with the direct imitation agents from Section III. The indirect imitation agents all used knowledge bias with $C_{\mathrm{BT}} = 0.2$ and $K = 1000$. Using all tested networks, we computed their accuracy in predicting human play as well as their win rate playing North and South against the ISMCTS baseline playing East and West.

Table III shows the results, contrasting them with the direct imitation results copied from Table I. The indirect imitation players reach almost the same accuracy as the direct imitation players—over 70%—while avoiding the networks' weakness in actual gameplay thanks to

their integration into ISMCTS. Finding a compromise between emulating humans and searching for the best moves to play increases the win rates of the indirect imitation players against the baseline from 11–22% to 45–47%, despite the mixed quality of the human gameplay data. This successfully demonstrates, for the first time in a commercial digital game, that indirect imitation via combining neural networks with MCTS can emulate human play and maintain competitive play with a state-of-the-art MCTS implementation, without requiring increased computational resources or time.

We also tested teams of the indirect imitation players as South and the baseline player as North. The resultant performance, in terms of win rate against the baseline was closer still to 50%. This demonstrates robustness of our proposed indirect imitation agent to differences in their partner's playing style, as will be needed when playing with human players.

V. Conclusions and Further Work

In this article, we proposed a method for biasing ISMCTS with gameplay data to emulate human play. We studied AI Factory Spades, a commercial mobile game which uses ISMCTS to control non-player characters. Previous work had indicated that adding a bias towards human play can enable ISMCTS agents to play in a style more similar to that of human players whilst maintaining playing strength equivalent to unmodified ISMCTS agents. In this article, we moved this work from a simulation of the game to the commercial codebase; removed the need for domain expertise, using neural networks to train our gameplay models to predict raw cards instead of game-specific abstract moves, which simplifies the transfer of this work to different games; and significantly increased the accuracy of the prediction over previous work. In addition, we empirically demonstrated the superiority of *indirect imitation* (combining the network's prediction with search) over *direct imitation* (directly playing the prediction of the network) in this domain; showed the comparable performance of the easier to tune *equivalent experience bias* technique to the previously used *knowledge bias* technique for implementing indirect imitation; and alleviated the added computational demands of using neural networks by employing *delayed bias*.

The result of this work is a biased ISMCTS agent that can emulate human play and maintain competitive play with a state-of-the-art ISMCTS implementation, without requiring increased computational resources or time. This successfully demonstrates, for the first time in a commercial game, the benefits of indirect imitation for creating believable non-player characters. The methodology is in principle applicable to any game currently played by an MCTS variant, provided that gameplay data from human players is available and that the agents' actions are (or could be modelled as) discrete.

Promising directions for future work include the following. First, we have so far used machine learning only to predict human moves and bias MCTS in their direction. It would also be possible to use it to predict the *cards in hand*, i.e. the hidden part of the game state (similar to the work on Android: Netrunner [33]), and use that to bias ISMCTS determinizations in order to achieve more intelligent play. Such approaches have proved enormously successful for Bridge [34]. AI Factory Spades currently uses hand-designed heuristics for this, so this is another opportunity to use machine learning to supplant the need for game-specific expertise.

Second, the application of this method to more games and different genres would help to analyze its generality and allow us to explore two aspects of this work more deeply: the ability of neural networks to learn the rules of a game, and to model multiple playstyles. Regarding the former, our networks effectively learned the rules of Spades (or at least the definition of a legal move) from simply observing human gameplay. This could be a key ability in scaling planning algorithms to complex, modern video games, where an approximation of the game rules learnt by a neural network could be used as a computationally efficient forward model. In Spades it appears that a single neural network was sufficient to model human playstyle — but in other games multiple models may be needed to capture significantly different clusters of playstyles.

Finally, it would also be desirable to empirically test the somewhat understudied assumption underlying all work on emulating human play: that human-like agents are more *fun* to play with or against [29], [5], [30]. This would require a significant user study in close collaboration with psychologists and human-computer interaction researchers. Given the advances we have made in emulating human play and the potential to get feedback from millions of human players of this commercial game, such a study becomes feasible and could help to further emphasise the importance of both this paper's contribution and the existing work on believable characters done by the community [27], [11], [5].

References

[1] P. I. Cowling, S. Devlin, E. J. Powley, D. Whitehouse, and J. Rollason, "Player preference and style in a leading mobile card game," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 7, no. 3, pp. 233–242, 2015.

[2] J. Bohannon, "Game-miners grapple with massive data," *Science*, vol. 330, no. 6000, pp. 30–31, 2010.

[3] M. S. El-Nasr, A. Drachen, and A. Canossa, *Game analytics: Maximizing the value of player data.* Springer Science & Business Media, 2013.

[4] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of Monte Carlo Tree Search methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, 2012.

[5] J. Ortega, N. Shaker, J. Togelius, and G. N. Yannakakis, "Imitating human playing styles in Super Mario Bros," *Entertainment Computing*, vol. 4, no. 2, pp. 93–104, 2013.

[6] A. Zook, B. Harrison, and M. O. Riedl, "Monte-Carlo Tree Search for simulation-based strategy analysis," in *10th Conference on the Foundations of Digital Games, FDG 2015*, 2015.

[7] T. Cazenave, F. Balbo, and S. Pinson, "Using a Monte-Carlo approach for bus regulation," in *12th International IEEE Conference on Intelligent Transportation Systems, ITSC 2009*, 2009, pp. 340–345.

TABLE III: Performance of the different networks, comparison of direct and indirect imitation. Direct imitation uses the network to choose moves; indirect imitation uses the network to bias ISMCTS (knowledge bias with $C_{\mathrm{BT}} = 0.2$, $K = 1000$). Indirect imitation results use equal time per move to the ISMCTS baseline with 2,500 playouts.

	direct imitation		indirect imitation	
	accuracy	win rate	accuracy	win rate
`200`	68.3%	11.5%	69.0%	47.3%
`200→100`	68.2%	11.4%	69.1%	46.2%
`t-200`	70.5%	17.5%	70.0%	45.8%
`t-200→100`	71.3%	18.1%	70.4%	46.6%
`t-200→150→100`	71.1%	22.0%	70.3%	46.1%
`t-c→200`	70.5%	18.4%	70.1%	46.0%
`t-c→200→100`	71.0%	14.0%	70.3%	45.9%
`t-cA→200`	71.1%	20.0%	70.3%	44.9%
`t-cA→200→100`	70.9%	16.6%	70.3%	48.8%
ISMCTS baseline			58.2%	50.0%
random player			44.0%	0.0%

[8] J. van Eyck, J. Ramon, F. Güiza, G. Meyfroidt, M. Bruynooghe, and G. V. den Berghe, "Guided Monte Carlo Tree Search for planning in learned environments," in *5th Asian Conference on Machine Learning, ACML 2013*, 2013, pp. 33–47.

[9] D. Whitehouse, P. I. Cowling, E. J. Powley, and J. Rollason, "Integrating Monte Carlo Tree Search with knowledge-based methods to create engaging play in a commercial mobile game." in *Ninth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE-13*, 2013.

[10] S. Devlin, A. Anspoka, N. Sephton, P. I. Cowling, and J. Rollason, "Combining gameplay data with Monte Carlo Tree Search to emulate human play," in *Twelfth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE-16*, 2016, pp. 16–22.

[11] J. Togelius, R. De Nardi, and S. M. Lucas, "Towards automatic personalised content creation for racing games," in *2007 IEEE Symposium on Computational Intelligence and Games*, 2007, pp. 252–259.

[12] K. Ikeda and S. Viennot, "Efficiency of static knowledge bias in Monte-Carlo Tree Search," in *Eighth International Conference on Computers and Games, CG 2013. Revised selected papers*. Springer, 2014, pp. 26–38.

[13] S. Gelly and D. Silver, "Combining online and offline knowledge in UCT," in *24th International Conference on Machine Learning, ICML 2007*, ser. ACM International Conference Proceeding Series, vol. 227, 2007, pp. 273–280.

[14] G. M. J. B. Chaslot, M. H. M. Winands, J. W. H. M. Uiterwijk, H. J. van den Herik, and B. Bouzy, "Progressive strategies for Monte-Carlo Tree Search," *New Mathematics and Natural Computation*, vol. 4, no. 03, pp. 343–357, 2008.

[15] R. Coulom, "Efficient selectivity and backup operators in Monte-Carlo Tree Search," in *5th International Conference on Computers and Games, CG 2006. Revised Papers*, ser. Lecture Notes in Computer Science, vol. 4630, 2007, pp. 72–83.

[16] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo planning," in *17th European Conference on Machine Learning, ECML 2006*, ser. Lecture Notes in Computer Science, vol. 4212, 2006, pp. 282–293.

[17] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[18] P. Cowling, E. J. Powley, and D. Whitehouse, "Information set Monte Carlo Tree Search," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 2, pp. 120–143, 2012.

[19] E. J. Powley, P. I. Cowling, and D. Whitehouse, "Memory bounded Monte Carlo Tree Search," in *Thirteenth Artificial Intelligence and Interactive Digital Entertainment Conference, AIIDE-17*, 2017, to appear.

[20] BBC News. (2017) Google AI defeats human Go champion. http://www.bbc.co.uk/news/technology-40042581. Accessed: July 13th 2017.

[21] M. Moravčík, M. Schmid, N. Burch, V. Lisý, D. Morrill, N. Bard, T. Davis, K. Waugh, M. Johanson, and M. Bowling, "DeepStack: Expert-level artificial intelligence in heads-up no-limit poker," *Science*, vol. 356, no. 6337, pp. 508–513, 2017.

[22] J. S. Hartford, J. R. Wright, and K. Leyton-Brown, "Deep learning for predicting human strategic behavior," in *29nd Conference on Neural Information Processing Systems, NIPS 2016*, D. D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, and R. Garnett, Eds., 2016, pp. 2424–2432.

[23] D. Billings, D. Papp, J. Schaeffer, and D. Szafron, "Opponent modeling in poker," in *AAAI/IAAI*, 1998, pp. 493–499.

[24] F. Schadd, S. Bakkes, and P. Spronck, "Opponent modeling in real-time strategy games." in *GAMEON*, 2007, pp. 61–70.

[25] G. Synnaeve and P. Bessiere, "A Bayesian model for opening prediction in RTS games with application to StarCraft," in *2011 IEEE Conference on Computational Intelligence and Games, CIG 2011*, 2011, pp. 281–288.

[26] E. W. Dereszynski, J. Hostetler, A. Fern, T. G. Dietterich, T.-T. Hoang, and M. Udarbe, "Learning probabilistic behavior models in real-time strategy games," in *Seventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE-11*, 2011.

[27] C. Thurau, T. Paczian, G. Sagerer, and C. Bauckhage, "Bayesian imitation learning in game characters," *International Journal of Intelligent systems Technologies and Applications*, vol. 2, no. 2-3, pp. 284–295, 2007.

[28] Pagat. (2016) Spades. http://www.pagat.com/boston/spades.html. Accessed: May 20th 2016.

[29] P. Hingston, *Believable bots*. Springer, 2012.

[30] A. Khalifa, A. Isaksen, J. Togelius, and A. Nealen, "Modifying MCTS for human-like General Video Game Playing," in *25th International Joint Conference on Artificial Intelligence, IJCAI 2016*, S. Kambhampati, Ed. IJCAI/AAAI Press, 2016, pp. 2514–2520.

[31] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[32] E. J. Powley, P. I. Cowling, and D. Whitehouse, "Information capture and reuse strategies in Monte Carlo Tree Search, with applications to games of hidden information," *Artificial Intelligence*, vol. 217, pp. 92–116, 2014.

[33] N. Sephton, P. I. Cowling, S. Devlin, V. J. Hodge, and N. H. Slaven, "Using association rule mining to predict opponent deck content in Android: Netrunner," in *2016 IEEE Conference on Computational Intelligence and Games, CIG 2016*, 2016, pp. 1–8.

[34] M. L. Ginsberg, "GIB: Imperfect information in a computationally challenging game," *Journal of Artificial Intelligence Research*, vol. 14, pp. 303–358, 2001.