Integrating Monte Carlo Tree Search with Knowledge-Based Methods to Create Engaging Play in a Commercial Mobile Game

Daniel Whitehouse and **Peter I. Cowling** and **Edward J. Powley**
Department of Computer Science, University of York, UK
dw830@york.ac.uk, peter.cowling@york.ac.uk, edward.powley@york.ac.uk

Jeff Rollason
AI Factory Ltd., Pinner, Middlesex, UK
jeff.rollason@ntlworld.com

Abstract

Monte Carlo Tree Search (MCTS) has produced many recent breakthroughs in game AI research, particularly in computer Go. In this paper we consider how MCTS can be applied to create engaging AI for a popular commercial mobile phone game: Spades by AI Factory, which has been downloaded more than 2.5 million times. In particular, we show how MCTS can be integrated with knowledge-based methods to create an interesting, fun and strong player which makes far fewer plays that could be perceived by human observers as blunders than MCTS without the injection of knowledge. These blunders are particularly noticeable for Spades, where a human player must co-operate with an AI partner. MCTS gives objectively stronger play than the knowledge-based approach used in previous versions of the game and offers the flexibility to customise behaviour whilst maintaining a reusable core, with a reduced development cycle compared to purely knowledge-based techniques.

Monte Carlo Tree Search (MCTS) is a family of game tree search algorithms that have advanced the state-of-the-art in AI for a variety of challenging games, as surveyed in (Browne et al. 2012). Of particular note is the success of MCTS in the Chinese board game Go (Lee, Müller, and Teytaud 2010). MCTS has many appealing properties for decision making in games. It is an anytime algorithm that can effectively use whatever computation time is available. It also often performs well without any special knowledge or tuning for a particular game, although knowledge can be injected if desired to improve the AI's strength or modify its playing style. These properties are attractive to a developer of a commercial game, where an AI that is perceived as high quality by players can be developed with significantly less effort than using purely knowledge-based AI methods. This paper presents findings from a collaboration between academic researchers and an independent game development company to integrate MCTS into a highly successful commercial version of the card game Spades for mobile devices running the Android operating system.

Most previous work on MCTS uses win rate against a fixed AI opponent as the key metric of success. This is appropriate when the aim is to win tournaments or to demonstrate MCTS's ability to approximate optimal play. However for a commercial game, actual win rate is less important than how engaging the AI is for the players. For example if the AI is generally strong but occasionally makes moves that appear weak to a competent player, then the player's enjoyment of the game is diminished. This is particularly important for games such as Spades where the player must cooperate with an AI partner whose apparent errors result in losses for the human player. In this paper we combine MCTS with knowledge-based approaches with the goal of creating an AI player that is not only strong in objective terms but is also perceived as strong by players.

AI Factory[1] is an independent UK-based company, incorporated in April 2003. AI Factory has developed a successful implementation of the popular card game Spades, which to date has been downloaded more than 2.5 million times and has an average review score of $4.5/5$ from more than 78 000 reviews on the Google Play store. The knowledge-based AI used in previous versions plays competitively and has been well reviewed by users. This AI was developed using expert knowledge of the game and contains a large number of heuristics developed and tested over a period of 10 years. Much of the decision making is governed by these heuristics which are used to decide bids, infer what cards other players may hold, predict what cards other players may be likely to play and to decide what card to play.

In AI Factory Spades, players interact with two AI opponents and one AI partner. Players can select their partners and opponents from a number of AI characters, each with a strength rating from 1 to 5 stars. Gameplay data shows that relatively few players choose intermediate level opponents: occasional or beginning players tend to choose 1-star opponents, whereas those players who play the game most frequently play almost exclusively against 5-star opponents. Presumably these are experienced card game players seeking a challenge. However some have expressed disappointment with the 5-star AI: although strong overall, it occasionally makes apparently bad moves. Our work provides strong evidence for a belief commonly held amongst game developers: the objective measures of strength (such as win rate) often used in the academic study of AI do not nec-

Copyright © 2013, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

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

essarily provide a good metric for quality from a commercial AI perspective. The moves chosen by the AI may or may not be suboptimal in a game theoretic sense, but it is clear from player feedback that humans apply some intuition about which moves are good or bad. It is an unsatisfying experience when the AI makes moves which violate this intuition, except possibly where violating this intuition is a correct play, but *even then* this appears to lead to player dissatisfaction. The primary motivation for this work is to improve the strongest levels of AI play to satisfy experienced players, both in terms of the objective strength of the AI and in how convincing the chosen moves appear.

Previous work has adapted MCTS to games which, like Spades, involve hidden information. This has led to the development of the *Information Set Monte Carlo Tree Search* (ISMCTS) family of algorithms (Cowling, Powley, and Whitehouse 2012). ISMCTS achieves a higher win rate than a knowledge-based AI developed by AI Factory for the Chinese card game Dou Di Zhu, and also performs well in other domains. ISMCTS uses *determinizations*, randomisations of the current game state which correspond to guessing hidden information. Each determinization is a game state that could conceivably be the actual current state, given the AI player's observations so far. In Spades, a determinization is generated by randomly distributing the unseen cards amongst the other players. Each ISMCTS iteration is restricted to a newly generated determinization, resulting in a single tree that collects statistics from many determinizations.

We demonstrate that the ISMCTS algorithm provides strong levels of play for Spades. However, previous work on ISMCTS has not dealt with the requirements for a commercially viable AI. Consequently, further research and development was needed in order to ensure the AI is perceived to be high quality by users. However, the effort required to inject knowledge into MCTS was small compared to the work needed to develop a heuristic-based AI from scratch. MCTS therefore shows great promise as a reusable basis for AI in commercial games. The ISMCTS player described in this paper is used in the currently available version of AI Factory Spades for the 4- and 5-star AI levels, and AI Factory have already begun using the same code and techniques in products under development.

This paper is structured as follows. We begin by outlining the rules of Spades and describing the knowledge-based approach used in AI Factory Spades. We then discuss some of the issues encountered in integrating MCTS with an existing mature codebase, and in running MCTS on mobile platforms with limited processor power and memory. We assess our MCTS player in terms of both raw playing strength and player engagement. We conclude with some thoughts on the promise of MCTS for future commercial games.

Spades

Spades is a 4-player trick taking card game which originated in the United States in the 1930s but has since spread worldwide (Pagat 2013). Spades shares many similarities with the game of Bridge, with equally deep strategy yet slightly simpler rules. The players are named after the compass points, with North and South forming a coalition against East and West. A game is played across multiple *rounds* where each partnership receives a score at the end of each round. The winning partnership has the highest score when one or both partnerships exceed 500 total points at the end of a round.

At the start of a round each player is dealt a 13 card hand from a standard 52 card deck. In turn, players provide a single *bid* which is an estimate of 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. One of the players (rotating between rounds) is designated the *leader* and may play any card (with the exception that ♠ cards cannot be led until *broken*, i.e. until a ♠ is played as a non-leading card). Subsequent players must match the suit of that card if they can. If a player cannot follow suit, they may play any card. The winning card is the one with the highest rank matching the suit played by the leader, unless a ♠ card was played in which case the ♠ *trumps* the other suit (and the highest ranked ♠ wins the trick instead). The winner of the trick becomes the leader of the next trick. The round ends when all players have played all their cards, so a round consists of 13 tricks.

At the end of the round, scoring is performed. Each partnership aims to take enough tricks between them to match their total bid. It does not matter whether or not the individuals in a partnership make their bids, only that the total is met. If a partnership makes their bid, they earn 10 times their bid in points but if they fail they lose that many points instead. Any extra tricks taken are called *bags* and are worth 1 point each. If throughout the course of a game a partnership collects 10 or more bags, they lose 100 points and 10 bags. The scoring rules change when a player bids *nil* (0 tricks). In this case scoring is performed as usual but the player bidding nil wins 100 points if they individually took no tricks, or loses 100 points if they took any tricks.

Knowledge-based AI for Spades

The existing AI player for AI Factory Spades is based on a heuristic-based program written in 2003 that was later evolved with substantial heuristic knowledge and in 2012 enhanced to use basic variations of Monte Carlo methods.

The AI chooses its bid based on a heuristic assessment of its hand. Each card has a certain probability of taking a trick (which may depend on what other cards are in hand). Summing these probabilities gives an expected number of tricks. This type of hand evaluation is commonly used by experienced human players. The bidding heuristics also take into account whether either partnership is at risk of exceeding the 10 bag limit and thus incurring a penalty: if the player partnership is close to the bag limit, it is often better to bid high than risk taking more bags. The bid is also adjusted by a personality factor (allowing for "timid" and "aggressive" AI characters) and a small random factor.

In Spades, players can make inferences on what cards other players hold in their hands, based on their bids and the cards they choose to play. The AI maintains a table for each player other than itself, recording the probabilities of that player holding each individual card in the deck. These probabilities are updated as the game progresses according to a set of hand-designed rules. There are also *hard constraints* which follow from the rules of the game: if a player fails to follow suit then they cannot possibly hold any cards in that suit, so the probabilities are set to zero.

For card play, the AI uses *flat Monte Carlo* evaluation with heuristic simulations. Each move from the current game state is assessed by playing out a set number of simulated rounds. The simulations use a combination of random moves and moves chosen according to a heuristic evaluation with automatically tuned weights (Rollason 2012). The AI then chooses the move with the best average simulation result. The number of simulations depends on the stage in the current round and on the AI player level: at the beginning of the round a 5-star player uses 128 simulations per available move, and a 4-star player uses half as many. To avoid occasional mistakes or "oddball" plays, a bonus is added to the Monte Carlo evaluations for the best and second best moves according to the knowledge-based heuristic evaluation. Thus the final choice of move is skewed towards that recommended by a heuristic evaluation (Rollason 2011).

To model the hidden information in the game, each simulation is played out on a randomised version of the current game state (a *determinization*). The AI uses the inferred probabilities described above to randomise the other players' hands, i.e. to redistribute the unseen cards amongst the players. The algorithm first deals high cards (J, Q, K, A) and $\spadesuit$ cards to each player according to their bids, using a similar hand assessment to the one used in bidding. It then deals the remainder of the cards in descending suit order, dealing each card according to its inferred probabilities and ensuring that each player is dealt the correct number of cards. It is important to note that the AI never "cheats": knowledge of other players' cards, other than that gained by inference, is never used in the AI's decision process. There are similarities between this approach and that used by GIB (Ginsberg 2001), which achieved world champion level play in Bridge. GIB uses fewer determinizations (around 50), but solves each one to find the exact game theoretic value rather than performing Monte Carlo simulations.

It is difficult to estimate the strength of state of the art AI for Spades, since we are unaware of any competitions or benchmark programs to test against. Based on player feedback and reviews, the AI is perceived as reasonably strong overall, but some of its individual decisions can appear weak. For example if the human player has bid nil, the AI partner will sometimes lead with a low card which forces the human player to take a trick, where leading with a high card in the same suit would have been safer. Also, the AI players sometimes waste high-value cards or trumps, playing a card of much higher value than needed to win the trick or stealing a trick from their partner. These occurrences are rare, but the size of the player base for AI Factory Spades means they are encountered by hundreds of players and potentially result in commercially damaging negative reviews.

MCTS implementation

MCTS for mobile devices

Many of the successes of MCTS, particularly for Go and General Game Playing, have seen highly parallelised implementations of the algorithm running on powerful workstations or clusters (Enzenberger et al. 2010; Björnsson and Finnsson 2009). Mobile devices are rapidly increasing in power but have yet to achieve this level of performance, and a significant proportion of AI Factory's user base still play on older devices. Even on high-end devices, a well-behaved mobile game should moderate its processor usage in order to preserve multitasking performance and extend battery life.

Our MCTS implementation is written in C++ and interacts directly with the AI Factory Spades game engine. The number of MCTS iterations for the highest AI difficulty level is set to 2500, which gives a reasonable tradeoff between playing strength and decision time. Executing 2500 iterations takes less than a quarter of a second on a Samsung Galaxy S II. On less capable handsets the number of iterations is automatically scaled down so that moves do not take longer than 250ms; this is not a hard technical limit, but many players complain of slow performance if it is exceeded. The bottlenecks for fast performance in MCTS implementations tend to be cloning the game state, generating lists of legal moves, and playing moves during simulation. and these parts of the code were extensively optimised.

Since each MCTS iteration adds a new node to the tree, the memory required by MCTS is the maximum number of iterations multiplied by the size of the node structure. In our implementation this gives a modest memory footprint of 2500×56 bytes $= 140$ kilobytes.

ISMCTS for Spades

We use ISMCTS for card play only; initial testing showed that ISMCTS is poor at making bidding decisions. This is not surprising, as assessing the bid purely by tree search potentially requires the search to look ahead to the end of the round. Thus our player uses the knowledge-based bidding heuristics described in the previous section.

Each ISMCTS iteration begins by generating a determinization of the current information set at random. To let the ISMCTS player use the knowledge-based inference engine, we use the hand randomisation algorithm described in the previous section to generate determinizations.

Usually MCTS algorithms play out a game to completion on each iteration. This performs poorly in Spades since there may be many rounds left to play, and playing rounds randomly usually results in both teams repeatedly failing their bid and accumulating negative points so that the score threshold of 500 points is never reached. Instead we terminate the playouts at the end of the current round and backpropagate the *score difference* value

$$\frac{(s_p - 10b_p) - (s_o - 10b_o)}{c} . \qquad (1)$$

Here s_p and s_o are the numbers of points accumulated in this round by the player and opponent partnerships respectively, and b_p and b_o are the numbers of bags accumulated in this round. The normalising constant c is chosen to ensure the evaluation produces values typically within the interval $\left[-\frac{1}{2}, \frac{1}{2}\right]$; the size of this interval is 1, so standard values for the UCB1 exploration constant can be used (Audibert, Munos, and Szepesvári 2009). Each bag is equated with a penalty of 10 points, meaning that the AI sees the penalty of bags upfront rather than when 10 bags are accumulated. This simple evaluation captures several key tactical ideas in Spades such as making your bid, preventing your opponents' bid, avoiding bags and giving bags to your opponent. This evaluation is applied only to compare outcomes of the current round during card play, so it does not need to capture tactical ideas for bidding.

Integrating heuristic knowledge

We shall see that the ISMCTS player described here is strong in terms of empirical win rate, but occasionally makes plays which appear weak to a human observer. This issue was addressed in the knowledge-based player by skewing the final choice of move away from that suggested by the flat Monte Carlo assessment and towards that recommended by the heuristic move evaluation. However, overruling the MCTS player in this way would remove the flexibility of MCTS to find stronger card plays when the play recommended by the knowledge-based heuristic, while plausible, is weak.

Instead we use an approach similar to progressive bias (Chaslot et al. 2008), in which heuristic knowledge is used to skew the initial exploration but the influence of this knowledge decays to zero as the search progresses. Specifically, each available move is assigned a skew value in the range $\left[\frac{1}{2}, \frac{3}{2}\right]$. These skew values are derived from a modification of the existing knowledge-based heuristics. During search, the backpropagated reward is multiplied by the corresponding skew value, so that moves with skew value greater than 1 have their rewards inflated and moves with skew value less than 1 have their rewards reduced. The skew values decay towards 1 as more MCTS iterations are executed: the values are scaled by $\frac{t}{k+t}$, where t is the iteration number and k is a tuned constant. This allows MCTS to overrule the heuristic knowledge as search progresses.

Quality and experience of play

Playing strength

To test the objective strength of the MCTS AI, we played a large number of AI-versus-AI games. In each of these games, both members of one partnership use the MCTS AI (with or without integrated heuristic knowledge) and both members of the other partnership use the knowledge-based AI. We test $2 \times 4 \times 2 = 16$ combinations in total: two settings for knowledge embedding (enabled or disabled), four settings for the number of MCTS iterations (1200, 1700, 2600 or 5000), and two settings for the difficulty level of the knowledge-based AI ("default" or "hardest"). In the current version of the game, 4- and 5-star players use MCTS and the lower levels use the knowledge-based AI. The 4- and 5-star MCTS players use 1000 and 2500 MCTS iterations respectively. The default knowledge-based AI tested here corresponds to a 3-star player. Extrapolating the star rating system, the hardest knowledge-based AI is effectively a 6-star player. The hardest AI would consume more than 2 seconds per move on a Samsung Galaxy S II, and represents a more challenging level of play than is available in the published game. The MCTS player with 5000 iterations would need only 0.5 seconds per move.

For each combination we play 1000 full games, and record the number of games won by the MCTS partnership. To assess statistical significance we compute 95% confidence intervals (Clopper and Pearson 1934). To reduce the variance of the results, each combination is tested with the same 1000 random seeds, i.e. the same 1000 sets of deals.

Figure 1 (a) shows the results of this experiment. For all tested numbers of iterations the MCTS AI is significantly stronger than the default level of the knowledge-based AI. For 2600 iterations and above MCTS is on a par with the hardest level. The benefit of performing more MCTS iterations is small but statistically significant: increasing from 1200 to 5000 iterations increases the win rate by 7–8%. Integrating heuristic knowledge has no statistically significant impact, but appears slightly beneficial for small numbers of iterations against the hardest opponent.

Strong play in the presence of nil bids requires different tactics from normal play, both for the nil bidder and for their partner. However nil bids are rare in normal play, so the influence of weak nil bid play cannot be assessed from overall win rates. Instead we test on an artificial variant of Spades designed to ensure that every round has at least one nil bid. At the start of the round, players bid as normal. If no player bids nil, the hands are redealt. Once a nil bid has been made, the round proceeds. This process continues until the score limit is reached as normal. We repeated the previous experiment on this variant of the game, with results shown in Figure 1 (b). Here the benefit of MCTS over the default knowledge-based AI is much larger than for the standard game, and MCTS is significantly better than the hardest opponent level. Note that the knowledge-based AI incorporates many heuristics for the special case of nil bids, whereas MCTS uses exactly the same evaluation as in the non-nil game (Equation 1), yet still the latter outperforms the former. Again there is no statistically significant difference between the ISMCTS players with and without heuristic knowledge, although the overall trend suggests that the knowledge may be slightly detrimental against the hardest opponent.

Playing style

In creating enjoyable AI for a commercial game, playing strength is secondary to player *perception* of intelligence. This is particularly true for Spades, where the majority of complaints regarding playing style are to do with perceived weak plays. In its aheuristic form, MCTS is more prone to occasional "oddball" plays than knowledge-based AI, particularly if the expected reward from the oddball play is close to that from the more sensible play. For example if a player is short of tricks and can choose to discard $8\diamondsuit$ or $10\diamondsuit$ (with $9\diamondsuit$ unseen), then the discard of $8\diamondsuit$ is the obvious choice as $10\diamondsuit$ has marginally more chance of taking a trick later in the round. In reality the advantage delivered in this choice is actually very small, too small for MCTS to make a guaranteed distinction between these two choices. Even in situations where the choice of card has no influence on the game whatsoever, for example if neither $8\diamondsuit$ nor $10\diamondsuit$ can possibly take a trick, a human observer still has certain expectations regarding what constitutes a plausible play.

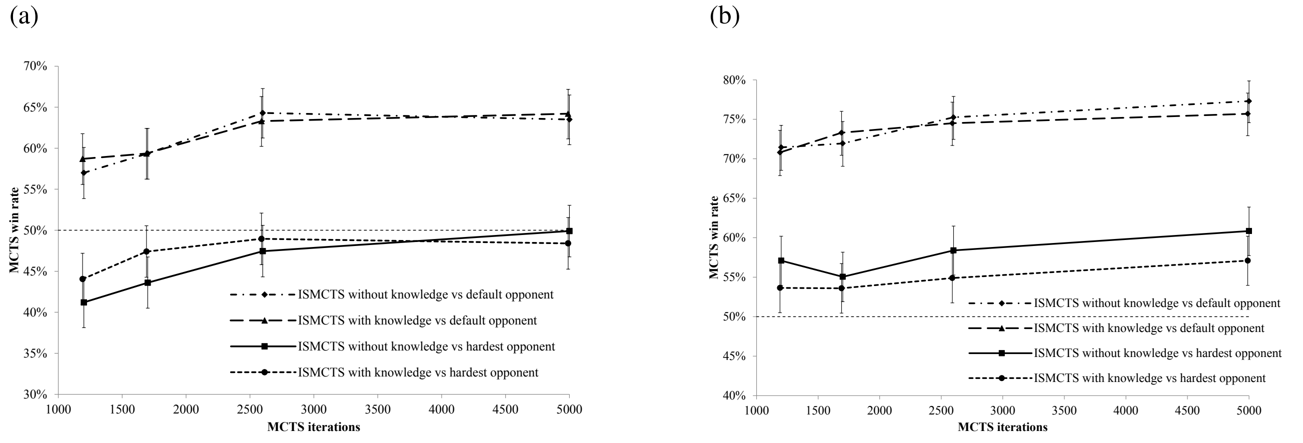

Figure 1: Playing strength of MCTS AI versus knowledge-based AI, for (a) the standard game of Spades; (b) the modified game where every round features a nil bid. Error bars show 95% confidence intervals.

A beta version of AI Factory Spades with the MCTS AI, but without the integrated heuristic knowledge, was distributed to a number of testers. An in-game menu option allowed the tester to send bug reports, comprising user comments and information about the AI settings, current game state and move history. We received a total of 18 beta test reports which highlighted perceived instances of poor play. It is worth emphasising that all of these reports were highlighting specific instances of perceived incorrect play, and not for example making more general comments regarding the AI's overall playing style. Of these, three were situations in which the AI player was forced to play an apparently poor move: for example, a tester complains that leading a ♡ was a bad play, but the AI player had only ♡ cards in their hand at that point. The remaining 15 reports highlight poor decisions by the AI player. These fall into two main categories, with roughly half of the reports in each:

1. The AI partner fails to cover the human player's nil bid. For example, North leads with 3♡, East plays 4♡, and South (the human player) is forced to follow suit with 5♡. West plays 2♡, so South wins the trick and loses the nil bid. North held higher cards in ♡, any of which would have been a safer leading play.
2. An AI player (partner or opponent) wastes a high value card. For example, North leads with $K\diamondsuit$ when $A\diamondsuit$ has not yet been seen, or North plays $K\clubsuit$ in a trick where $A\clubsuit$ is already the highest card. (This is not always a bad play, but is often perceived as such.)

The majority of the reports in both categories related to the player's partner rather than an opponent. This does not necessarily mean that the partner AI is weaker or less plausible than the opponent AI, just that any unusual moves are more likely to be noticed (especially if, from the human player's point of view, a mistake by the AI partner results in a loss). Mistakes by the AI opponents generally lead to wins for the human player, and thus are less frustrating.

A plausible explanation for the first category of problem is that random simulations tend to fail nil bids. Thus the AI player will effectively assume the nil bid will fail and instead aim for other objectives (making its own bid, causing the opponents to fail theirs) to minimise the loss in score difference. Preliminary experiments with a simple rule-based simulation policy (instead of playing randomly, the nil bidder plays the highest card that avoids taking the current trick) show promise for mitigating this, but are not employed in the currently released version of the game as knowledge embedding is also effective in these cases.

For the second category of problem, the AI player may be unable to see the value of the high card, as the trick that could be won by that card is beyond the horizon of the tree. The ISMCTS tree for a decision near the start of a round usually has a maximum node depth of 4 or 5, so the tree generally includes only the current trick and part of the next trick. We observed a similar effect in the board game Lord of the Rings: The Confrontation, where an MCTS player tends to waste high-value combat cards early in the game (Cowling, Powley, and Whitehouse 2012).

With any method based on random simulation, it is inevitable that poor quality moves will be chosen with nonzero probability, due to a particularly lucky run of simulations making the move appear better than it is. For each poor decision, we ran MCTS 1000 times and counted how many times each move was chosen. In six of the reports, we found that MCTS chose the incorrect move in fewer than 5% of trials. Performing more MCTS iterations, or performing several independent searches and combining the results in the style of ensemble MCTS (Fern and Lewis 2011), would mitigate these problems, but would be costly in terms of computation time and thus are not currently implemented.

At least three of the incorrect decisions in the beta reports are a result of incorrect biases introduced by the determinization sampling algorithm. If a player has bid high, the algorithm will tend to fill that player's hand with ♠ cards and high cards, so that when the low cards are dealt that player's hand is already full. Conversely a player who has bid low will be dealt disproportionately many low cards. In one situation for example, the inferred probability of South holding

$4\diamondsuit$ is $\frac{1}{3}$, and South does not actually hold this card. However South has bid nil whilst the other players have bid high, so 87% of determinizations have $4\diamondsuit$ in South's hand. This leads North to assume that $5\diamondsuit$ is a safe leading play to cover South's nil bid, when in fact it is not. We tested an alternative determinization algorithm which respects hard constraints (i.e. unfollowed suits) but otherwise distributes cards uniformly at random. This leads to correct decisions in the situations where this type of bias occurs, but leads to many more unusual or incorrect plays in other situations. A combination of the two algorithms could work, as could a more probabilistically sound sampling method which does not introduce these erroneous biases. This is a subject for future work; currently the game uses only biased determinizations.

The remainder of the test reports highlight poor decisions which are not attributable to bad luck or bad determinizations. The knowledge injection method described in the previous section was introduced and tuned to avoid these problems. This also produces more plausible plays in situations where ensemble methods or unbiased determinizations proved effective, and any problem situations identified by players in future can be addressed easily by modifying the heuristic. The heuristic knowledge embedding was tuned carefully to address these problem decisions without negatively affecting playing strength, but it is worth pointing out that this tuning took days, rather than the weeks or months often required to tune a purely heuristic move selection method. Although the heuristic knowledge has very little effect on objective playing strength, and may even be detrimental in some cases, it has a significant positive impact on the player's subjective impressions of the AI's strength.

Conclusion

This paper demonstrates MCTS as an AI technique that is general purpose and easy to integrate into an existing game product, while also being efficient enough to run on current mobile devices. Developing a strong AI based on MCTS requires less effort and time than developing one based purely on knowledge-based approaches. However if such a knowledge-based AI has already been developed, its heuristics can be repurposed within the MCTS framework to produce an even stronger AI, which retains the personality and plausibility of the knowledge-based AI. Constructing a plausibility measure to skew the choices made by MCTS is much easier than creating an evaluation to choose the actual move, since is it not necessary to ensure a plausibility measure is always accurate. Designing a heuristic to recommend several moves to avoid is often easier than designing one to recommend a single move to play. Knowledge-based approaches alone are vulnerable in situations that were not anticipated by the AI designer. However, MCTS can "fill in the gaps" of included knowledge whatever the situation, either by evaluating situations that the knowledge does not cover or by overruling the knowledge when it is flawed.

The existing knowledge-based AI for Spades was already a market leader, and generally recognised as strong. The MCTS-based player performs better by a statistically significant margin, playing at or above a level for the previous AI which consumed over 8 times as much computation time. MCTS play is particularly strong in the challenging case of nil bid play. The Spades-specific parts of the MCTS implementation are the determinization method (linked with the inference engine), the end-of-round evaluation function, and the initial biasing of the search by heuristic knowledge. None of these components is essential to the basic operation of MCTS, and without these the implementation is highly generic. The MCTS implementation now constitutes a ready-made AI module that can simply be dropped into future projects and produce a strong level of play with almost no modification. Indeed, AI Factory are already using the same code for new games currently in development. The MCTS framework also provides extensive hooks for game-specific knowledge injection to improve both objective playing strength and subjective playing style.

To ensure a high quality AI opponent in a commercial game, search based methods require some tweaking in order to ensure the AI behaves plausibly in all situations. It is not enough for the AI to be strong empirically, it must be *perceived* as strong by the player. For example in situations where the outcome of the round is already decided, the choices made by MCTS are essentially random. However a human player would still prefer some moves over others, due to their intuition for which moves are sensible and which are not. There are also situations where the optimal move (in the game theoretic sense) is correctly chosen by MCTS but looks highly counterintuitive to a human player. For example the AI player can count cards perfectly, so potentially has access to information that even an expert human player would not. We have demonstrated that an MCTS based AI can be modified to behave more plausibly without compromising playing strength.

AI Factory Spades uses MCTS only for the highest AI difficulty levels, as the knowledge-based AI is adequate for lower levels. An MCTS based AI allows difficulty to be adjusted by altering the number of iterations used. It is ongoing work to study how the playing style of MCTS changes when tuning difficulty in this way. When using a low number of iterations it is likely that moves which naïvely appear strong without much analysis will be chosen, which potentially makes these mistakes more "human-like" than those which might be obtained through approaches such as randomly perturbing an evaluation function.

Since its invention in 2006, MCTS has been established as a strong general-purpose technique in game AI. We believe that MCTS will rapidly gain traction in AI for commercial games, due to its flexibility, ease of implementation and out-of-the-box playing strength. Coupled with more traditional knowledge-based approaches, MCTS has the potential to deliver AI that provides not only a strong challenge for expert players but also a wide variety of playing styles and behaviours for interesting, immersive and fun player experiences.

Acknowledgements

We thank the anonymous reviewers for their useful comments. This work is funded by grant EP/H049061/1 of the UK Engineering and Physical Sciences Research Council (EPSRC).

References

Audibert, J.-Y.; Munos, R.; and Szepesvári, C. 2009. Explorationexploitation tradeoff using variance estimates in multi-armed bandits. *Theor. Comput. Sci.* 410(19):1876–1902.

Björnsson, Y., and Finnsson, H. 2009. CadiaPlayer: A Simulation-Based General Game Player. *IEEE Trans. Comp. Intell. AI Games* 1(1):4–15.

Browne, C.; Powley, E. J.; Whitehouse, D.; Lucas, S. M.; Cowling, P. I.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; and Colton, S. 2012. A Survey of Monte Carlo Tree Search Methods. *IEEE Trans. Comp. Intell. AI Games* 4(1):1–43.

Chaslot, G. M. J.-B.; Winands, M. H. M.; van den Herik, H. J.; Uiterwijk, J. W. H. M.; and Bouzy, B. 2008. Progressive Strategies for Monte-Carlo Tree Search. *New Math. Nat. Comput.* 4(3):343–357.

Clopper, C. J., and Pearson, E. S. 1934. The use of confidence or fiducial limits illustrated in the case of the binomial. *Biometrika* 26(4):404–413.

Cowling, P. I.; Powley, E. J.; and Whitehouse, D. 2012. Information Set Monte Carlo Tree Search. *IEEE Trans. Comp. Intell. AI Games* 4(2):120–143.

Enzenberger, M.; Müller, M.; Arneson, B.; and Segal, R. B. 2010. Fuego - An Open-Source Framework for Board Games and Go Engine Based on Monte Carlo Tree Search. *IEEE Trans. Comp. Intell. AI Games* 2(4):259–270.

Fern, A., and Lewis, P. 2011. Ensemble Monte-Carlo Planning: An Empirical Study. In *Proc. 21st Int. Conf. Automat. Plan. Sched.*, 58–65.

Ginsberg, M. L. 2001. GIB: Imperfect Information in a Computationally Challenging Game. *J. Artif. Intell. Res.* 14:303–358.

Lee, C.-S.; Müller, M.; and Teytaud, O. 2010. Guest Editorial: Special Issue on Monte Carlo Techniques and Computer Go. *IEEE Trans. Comp. Intell. AI Games* 2(4):225–228.

Pagat. 2013. Spades. `http://www.pagat.com/boston/spades.html`.

Rollason, J. 2011. Mixing MCTS with conventional static evaluation. `http://www.aifactory.co.uk/newsletter/2011_02_mcts_static.htm`.

Rollason, J. 2012. Tuning Spades. `http://www.aifactory.co.uk/newsletter/2012_01_tuning_spades.htm`.