Monte-Carlo Tree Search in Settlers of Catan

István Szita[1], Guillaume Chaslot[1], and Pieter Spronck[2]

[1] Maastricht University, Department of Knowledge Engineering
[2] Tilburg University, Tilburg centre for Creative Computing
szityu@gmail.com, g.chaslot@micc.unimaas.nl, p.spronck@uvt.nl

Abstract. Games are considered important benchmark opportunities for artificial intelligence research. Modern strategic board games can typically be played by three or more people, which makes them suitable test beds for investigating multi-player strategic decision making. Monte-Carlo Tree Search (MCTS) is a recently published family of algorithms that achieved successful results with classical, two-player, perfect-information games such as Go. In this paper we apply MCTS to the multi-player, non-deterministic board game Settlers of Catan. We implemented an agent that is able to play against computer-controlled and human players. We show that MCTS can be adapted successfully to multi-agent environments, and present two approaches of providing the agent with a limited amount of domain knowledge. Our results show that the agent has a considerable playing strength when compared to game implementation with existing heuristics. So, we may conclude that MCTS is a suitable tool for achieving a strong Settlers of Catan player.

1 Motivation

General consensus states that a learning agent must be situated in an experience-rich, complex environment for the emergence of intelligence [1, 2]. In this respect, games (including the diverse set of board games, card games, and modern computer games) are considered to be ideal test environments for AI research [3–6]. This is especially true when we take into account the role of games in human societies: it is generally believed that games are tools both for children and adults for understanding the world and for developing their intelligence [7, 8].

Most games are abstract environments, intended to be interesting and challenging for human intelligence. Abstraction makes games easier to analyze than real-life environments, and usually provides a well-defined measure of performance. Nevertheless, in most cases, the complexity is high enough to make them appealing to human intelligence. Games are good indicators and often-used benchmarks of AI performance: Chess, Checkers, Backgammon, Poker, and Go all define important cornerstones of the development of artificial intelligence [9, 10].

Modern strategic board games (sometimes called "eurogames") are increasing in popularity since their (re)birth in the 1990s. The game Settlers of Catan can be considered an archetypical member of the genre. Strategic board games are of

particular interest to AI researchers because they provide a direct link between classic (two-player, perfect information) board games and video games. On the one hand, state variables of most modern board games are discrete, and decision making is turn-based. On the other hand, the gameplay in modern board games often incorporates elements of randomness, hidden information, multiple players, and a variable initial setup, which make it hard to use classical techniques such as alpha-beta pruning [11] or opening books.

Several computer implementations of Settlers of Catan exist, which typically feature a hand-designed, rule-based AI. The strength of these AIs varies, but an experienced player can defeat them easily. Few research papers are available on autonomous learning in Settlers of Catan [12], and according to the results reported therein, they are far from reaching human-level play yet. In this paper, we investigate whether it is possible to use one of the AI tools of classical board games, namely Monte-Carlo Tree Search, effectively for implementing game-playing agents for games such as Settlers of Catan

2 The game: Settlers of Catan

Settlers of Catan, designed by Klaus Teuber, was first published in 1995. The game achieved a huge success: it received the "Game of the Year" award of the German game critics, and it was the first "eurogame" to become widely popular outside Germany, selling more than 11 million copies, inspiring numerous extensions, successors, and computer game versions.

2.1 Game rules

In Settlers of Catan, the players take the role of settlers inhabiting an island. They collect resources by suitable positioning of their settlements. They use these resources to build new settlements, cities, roads, and developments. The goal of the game is to be the first player who gains at least 10 *victory points*. Detailed descriptions of the rules can be easily found on the internet. Below, we summarize the rules (omitting many details).

Game board and resources. The game board representing the island is randomly assembled from 19 hexagonal *tiles* forming a large hexagon (Figure 1 displays an example game board in our SmartSettlers program). Each tile represents a field that provides one of the five types of *resources*: *wood*, *clay*, *sheep*, *wheat* or *ore*. There is also one *desert* which does not produce anything. Each non-desert tile has a *production number*. Several of the sea tiles surrounding the island contain a *port*. There is a *robber* on the island, initially residing in the desert. During the game, the players can build *settlements* and *cities* on the vertices of the hexagons, and *roads* on the edges. Settlements can be placed on any free vertex that respects the *distance rule*: no settlement may be located on a vertex that is connected to another settlement or city by exactly one edge. Players start the game with two settlements and two roads already built.

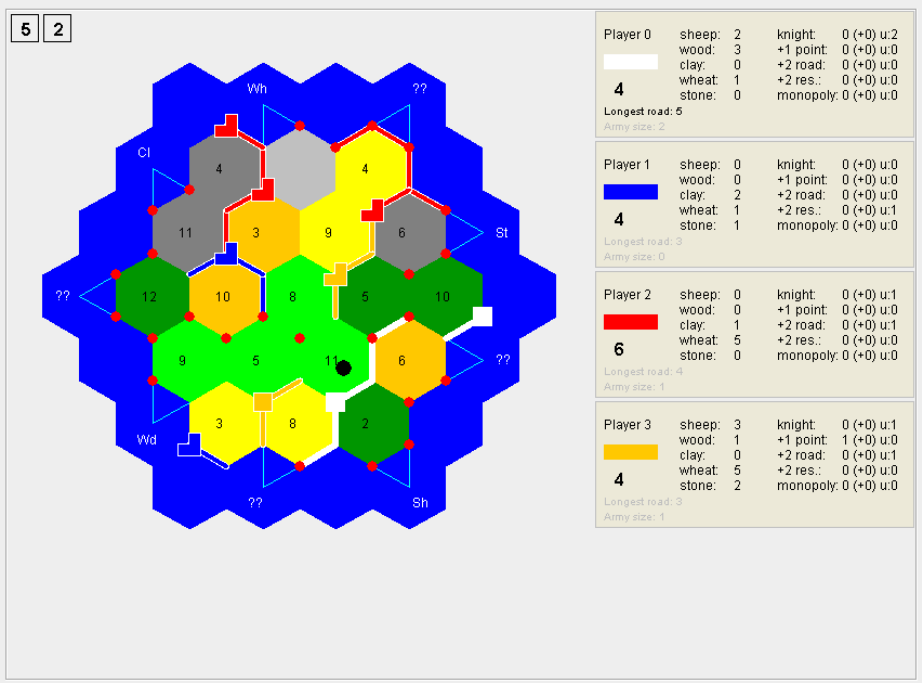


Fig. 1. A game state of Settlers of Catan represented in our program SmartSettlers

Production. Each player's turn starts by the rolling of two dice, which determines production. Any field that bears a production number equal to the sum of the dice, produces resources in that turn. Any player (i.e., not only the player whose turn it is) who has a settlement adjacent to a producing field, receives one of the corresponding resources. For any adjacent city, he receives two. On a dice roll of 7, nothing is produced but the robber is activated. Any player who has more than seven resource cards must discard half of them. After that, the current player moves the robber to a new field. The field with the robber is *blocked*, i.e., it does not produce anything, even if its number is rolled. Furthermore, the current player can draw a random resource from one of his[3] opponents who has a field adjacent to the robber's field.

Buildings, roads, and developments. After the production phase, a player can take zero or more actions. He can use these actions to construct new buildings and roads. A new road costs *1 wood and 1 clay*, and must be connected to the player's existing road network. A new settlement costs *1 wood, 1 clay, 1 sheep, and 1 wheat*, and it must be placed in a way that it is connected to the player's road network, respecting the distance rule. Players may upgrade their settlements to cities for *3 ore and 2 wheat*. Players can also purchase a

[3] For brevity, we use 'he' and 'his' whenever 'he or she' and 'his or her' are meant.

random *development card* for *1 sheep, 1 wheat, and 1 ore*. Cards can give three kinds of bonuses. (1) Some cards give 1 victory point. (2) Some cards are *Knight cards*, which can be used to activate the robber immediately. (3) The third group of cards gives miscellaneous rewards such as free resources or free roads. In his turn, a player may use at most one development card.

Trading. The current player is allowed to trade resources with his opponents, as long as all involved in the trade agree. Players may also "trade with the bank": which means that they exchange four resources of one kind for one of a different kind. If they have built a settlement or city connecting to a port, they can trade with the bank at a better ratio, depending on the kind of port.

Victory points and trophies. Next to the development cards that provide *1 victory point*, a player gains victory points for settlements, cities, and trophies. Each settlement is worth *1 victory point*, and each city *2 victory points*. The first player to use three Knight cards obtains the trophy *"Largest army"*. Subsequently, if another player uses more Knight cards, he becomes the trophy holder. Similarly, the player who is first to build a chain of five or more connected roads, obtains the trophy *"Longest road"*, and owns it until someone builds a longer chain. Both trophies are worth *2 victory points*. The game ends as soon as a player reaches at least 10 victory points. That player wins the game.

2.2 Rule changes

From the rules it follows that Settlers of Catan has a variable initial setup, non-deterministic elements (the dice rolls), elements of imperfect information (buying of development cards, and a lack of knowledge of cards stolen), and more than two players. For an easier implementation of the game, we changed the rules to remove the elements of imperfect information. In our opinion, this change does not alter gameplay in a significant way: knowing the opponents' development cards does not significantly alter the strategy to be followed, and information on the few cards stolen is usually quickly revealed anyway. We also chose to not let our game-playing agent initiate trades with or accept trades from other players (although it may trade with the bank). Note that the other players may trade between themselves, which handicaps our agent slightly. While we believe that these changes do not alter the game significantly, it is our intention to upgrade our implementation to conform completely to the rules of Settlers of Catan in future work.

2.3 Previous computer implementations

There are about ten computer implementations of Settlers of Catan available. We mention two of the strongest ones. The first is Castle Hill Studios' version of the game, currently part of Microsoft's MSN Games. The game features strong AI players who use trading extensively. The second is Robert S. Thomas' JSettlers, which is an open-source Java version of the game, also having AI players. The latter is the basis of many Settlers of Catan game servers on the Internet. The JSettlers architecture is described in detail by Thomas [13]. Pfeiffer [12]

also used the JSettlers environment to implement a learning agent. His agent uses hand-coded high-level heuristics with low-level model trees constructed by reinforcement learning.

3 Implementation

We implemented the Settlers of Catan game in a Java software module named *SmartSettlers*. JSettlers was used for providing a graphical user interface (GUI) and the baseline AI. Because of the Internet-based architecture of JSettlers, gameplay is fairly slow: a single four-player game takes about 10 minutes on an average PC. The learning algorithm that is investigated in our project requires thousands of simulated games before making a single step, so the JSettlers environment in itself is clearly inadequate for that purpose. Therefore, we implemented SmartSettlers as a *standalone* Java software module. It was designed for fast gameplay, move generation, and evaluation. For optimum speed, game data was divided into two parts: information that is constant throughout a game was stored in a static object, while game-state dependent information was stored in unstructured arrays of integers. The latter structure enabled quick accessing and copying of information. On an average PC, SmartSettlers is able to play around 300 games per second with randomly generated moves. JSettlers handles the SmartSettlers AI as a separate thread, querying it for decisions through a translation interface. In addition, SmartSettlers is able to run as a stand-alone program for investigating the self-play of the SmartSettlers AI. The goal of the current research is to create a strong AI agent for Settlers of Catan.

4 Monte-Carlo Tree Search

In recent years, several Monte-Carlo based techniques emerged in the field of computer games. They have already been applied successfully to many games, including Poker [14] and Scrabble [15]. Monte-Carlo Tree Search (MCTS) [16], a Monte-Carlo simulation based technique that was first established in 2006, is implemented in top-rated Go programs [17–19]. The algorithm is a best-first search technique which uses stochastic simulations. MCTS can be applied to any two-player, deterministic, perfect information game of finite length (but its extension to multiple players and non-deterministic games is straightforward). Its basis is the simulation of games where both the AI-controlled player and its opponents play random moves. From a single random game (where every player selects his actions randomly), very little can be learned. But from simulating a multitude of random games, a suitable strategy can be inferred. In subsection 4.1 we discuss the effect of a starting position, or otherwise stated the effect of the seating order Studies of MCTS in Go have shown that inclusion of domain knowledge can improve the performance significantly [17, 20]. There are at least two possible ways to include domain knowledge: (1) using non-uniform sampling in the Monte-Carlo simulation phase and (2) modifying the statistics stored in the game tree. The approaches are discussed in Subsection 4.2 and 4.3,

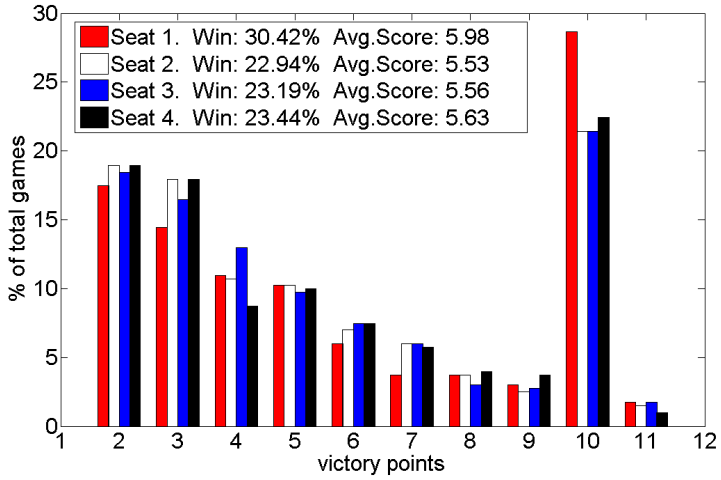


Fig. 2. The effect of seating order on the score distribution of random agents. Agents select a random legal move each turn.

respectively. For a detailed description of MCTS we refer the reader to one of the previously mentioned sources.

4.1 Effect of starting position

Reviews of Settlers of Catan suggest that the seating order of players has a significant effect on their final rankings (although opinions differ which seat gives the best strategic position). We conducted two preliminary sets of experiments to confirm this effect. We recorded the games of four identical agents playing random moves. In our first set of experiments, all agents made random (but legal) moves, while for the second set, the agents used MCTS with 1000 simulated games per move. The score distributions for different seating orders are shown in the Figures 2 and 3. In both cases, 400 games were played.

The results show that the effect of the seating order is present and is statistically significant. It is surprising, however, that the actual effect can differ for different strategies, despite the fact that both tested strategies are rather weak. For random agents, the starting player has a distinct advantage, while for MCTS agents player 1 is worst off, and players 2 and 3 have an advantage. A possible explanation is that, for purely random players, the first player has an advantage because in general he can play one extra move in comparison to the other players. Conversely, when players follow some kind of strategy, it seems that being second or third in the seating order gives a strategic advantage. This effect is probably related to the placement of initial settlements, and it would be interesting to study the seating-order effect for stronger MCTS players that play 10,000 or more simulated games per move.

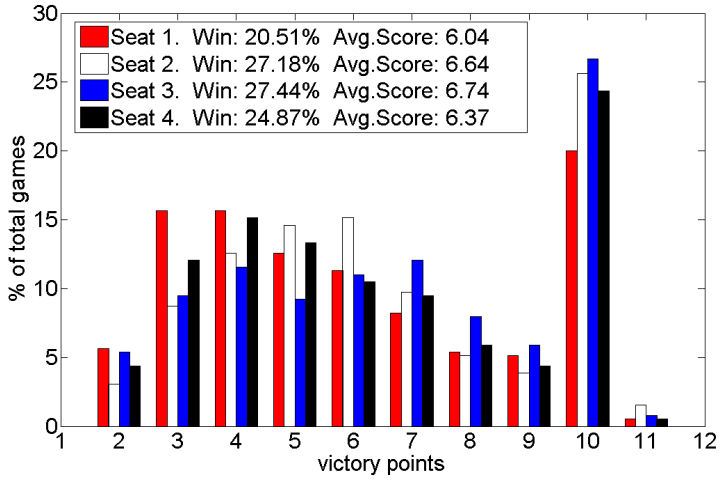


Fig. 3. The effect of seating order on the score distribution of SmartSettlers agents. Agents play 1000 simulated games per each real move.

The two preliminary experiments showed that the seating order effect can introduce an unknown bias to the performances of agents. In order to eliminate this bias, the seating order was randomized for all subsequent experiments where different agents were compared.

4.2 Domain knowledge in Monte-Carlo simulations

If all legal actions are selected with equal probability, then the strategy played is often weak, and the quality of a Monte-Carlo simulation suffers as a result. We can use heuristic knowledge to give larger weights to actions that look promising. However, we must be careful if the selection strategy is too deterministic, then the exploration of the search space becomes too selective, and the quality of the Monte-Carlo simulation suffers. Hence, a balance between exploration and exploitation must be found in the simulation strategy.

We set the following weights for actions:

- the basic weight of each action is +1
- building a city or a settlement: +10,000; building a city or a settlement is always a good move: it gives +1 point, and also increases the resource income; all possible city/settlement-building possibilities are given the same weight; it is left to the MC simulation to differentiate between them;
- building a road: define the road-to-settlement ratio as

$$R := \frac{\text{No. of player's roads}}{\text{No. of player's settlements + cities}}$$

and the weight of a road-building move as $10/10^R$. If there are relatively few roads, then road building should have a high weight; conversely, if there are many roads, then road building is less favourable than ending the turn;
- playing a knight card: +100 if the robber is blocking one of the player's own fields, otherwise +1;
- playing a development card: +10.

Probabilities of choosing an action in the Monte-Carlo simulation were proportional to their weights. The settings seem reasonable according to our experience with the game and with expert advice on Monte-Carlo simulations. However, the actual performance of our agent dropped significantly when using the modified probabilities instead of uniform sampling. A possible explanation is that with the given weights, the agents are too eager to build settlements and cities, while there may be a strategic advantage in giving preference to extending the roads network to reach a better strategic position. Identifying the precise reasons for the observed performance drop and working out a better set of weights (possibly in an automated process) is part of our future work.

4.3 Domain knowledge in MCTS

Recent work [21] showed that domain knowledge can be easily injected into the tree-search aspects of MCTS, too. We can give "virtual wins" to preferred actions (and we could also give "virtual losses" to non-preferred ones). We added a quite limited amount of domain knowledge: all settlement-building actions receive 20 virtual wins, and all city-building actions receive 10. Other actions do not receive any virtual wins. This means that for each time a settlement-building action is added to the search tree, its counters for the number of visits and the number of wins are both initialized to 20.

Note that virtual wins are not propagated back to parent nodes, as that would seriously distort selection. For example, consider a situation where the player has two options: he can either build a settlement in one of 5 possible places (giving +20 virtual wins) or buy a development card first (giving no virtual wins), and build a settlement afterwards (giving +20 virtual wins for any of the 5 possible placements). If virtual wins are backpropagated, then the card-buying action obtains a huge +100 bonus. As an effect, the *potential to build* a settlement will be rated higher than the *actual building* of that settlement. This is an effect that should be avoided.

The small addition of virtual wins increased the playing strength of our agent considerably, so subsequent tests were run with this modification. Further optimization of distributing virtual wins should be possible, and is part of our plans for future work.

5 Playing strength

We tested the playing strength of SmartSettlers against the JSettlers AIs (5.1) and against humans (5.2).

5.1 Testing MCTS against JSettlers

Against the baseline heuristics of JSettlers we tested three different AIs: the random player, MCTS with $N = 1000$ simulated games per move, and MCTS with $N = 10000$. In all experiments, three JSettlers agents were playing against one other AI. For each experiment, 100 games were played. Following our preliminary investigations, we assume that no biasing terms are present, so results for the three JSettlers agents are interchangeable. Therefore we obtain three times as many data points for the JSettlers agents than for the other AIs. The winning percentages presented below correspond to a *single* JSettlers agent. This means that a player that is equal in strength to JSettlers wins 25% of the time. The results are shown in Figure 4.

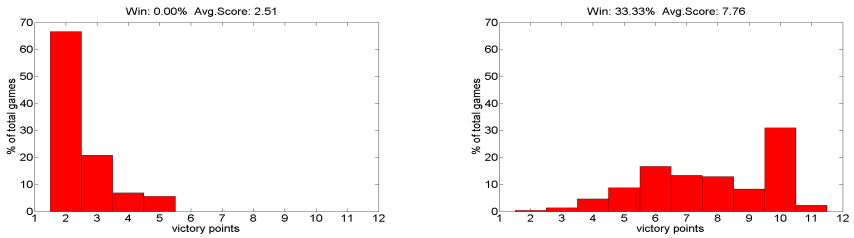
From Figure 4 (a) and (b) we may conclude that, as expected, the random player is very weak: it does not win a single game, and in fact, in most games it does not even score a point (besides the points it receives for its initial settlements). From Figure 4 (c) and (d) we may conclude that MCTS with 1000 simulated games is roughly as strong as JSettlers: it wins 27% of the games, and the score distribution is also similar. Finally, from Figure 4 (e) and (f) we may conclude that MCTS with 10000 simulated games is convincingly better than JSettlers: it wins 49% of all games, and reaches good scores even when it does not win.

5.2 Testing MCTS against humans

To test how SmartSettlers performs against humans, the first author, who is an accomplished Settlers of Catan player, played a few dozen games against a combination of two JSettlers' agents and one SmartSettlers agent. While the number of games played is not sufficient for drawing statistically relevant conclusions, these games do provide some information about the playing strength and style of our agent. Qualitatively speaking, we assess that the SmartSettlers agent makes justifiable moves that often coincide with moves that a human would play. Still, we found that an expert human player can confidently beat the SmartSettlers agent.

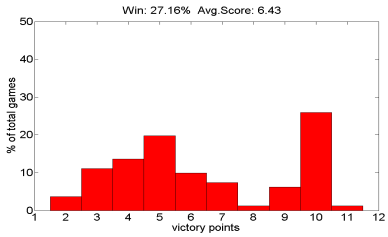
For an analysis of the possible reasons for the human supremacy over SmartSettlers, we examined different strategies for Settlers of Catan. There are two major "pure" winning strategies (in practice, human players often follow a combination of these two): the "ore-and-wheat" strategy and the "wood-and-clay" strategy. The former focuses on building cities and buying development cards, Knight cards (receiving +2 points for the largest army), and 1-point cards. The latter strategy focuses on building settlements and an extensive road network (which may lead to receiving +2 points for the longest road). Our SmartSettlers agent seems to prefer the "ore-and-wheat" strategy, together with building as many roads as possible, but not building as many settlements as an expert human player would.

The source of this behavior is probably that MCTS does not look forward in the game tree to a sufficient depth. Building a settlement requires four different

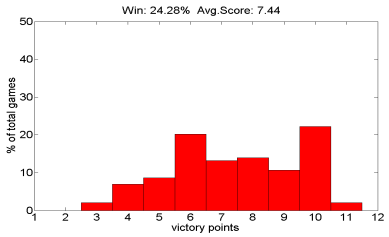


(a) Random player

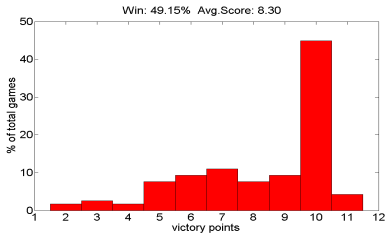
(b) JSettlers player



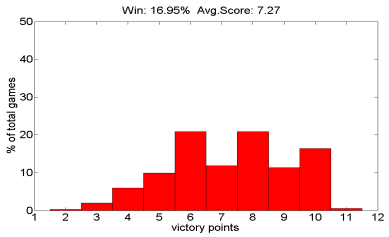
(c) MCTS-1000 player



(d) JSettlers player



(e) MCTS-10000 player



(f) JSettlers player

Fig. 4. Three sets of score distributions. Top: (a) Random player vs. (b) JSettlers. The random player won 0% of the time. Middle: (c) MCTS player ($N = 1000$) vs. (d) JSettlers. The MCTS-1000 player won 27% of the time. Bottom: (e) MCTS player ($N = 10,000$) vs. (f) JSettlers. The MCTS-10,000 player won 49% of the time.

resources. Collecting these often requires 3 to 4 rounds of waiting, trading, and luck. In the meantime, the agent can spend some of the resources to buy a road to obtain a marginal advantage. While this may be much less preferable in the long run (for example, obtaining 2 victory points for the longest road is a huge advantage in the short run, but it does not help increasing the production rates), the agent may not see this, as it does not analyze properly the game tree at that depth. Increasing the number of simulated games per turn would probably alleviate this weakness, but at the cost of a significant speed decrease. An alternative approach we wish to pursue is to improve the move-selection heuristics for the Monte-Carlo simulations.

6 Future work

Our plans for future work can be grouped into two categories. First, we plan to update our software so that it complies fully with the original rules of Settlers of Catan. To this end, we need to implement trading (which should be straightforward) and the handling of hidden information. The latter requires the implementation of a plain inference routine (keeping track of the possible values of the unknown cards) and an extension of MCTS to the case where the current state has uncertainty.

Second, we believe that the playing strength of our agent can be improved considerably by injecting domain knowledge in a proper way. There are at least two opportunities to place domain knowledge: by modifying the heuristic action-selection procedure inside the MCTS tree (by adding virtual wins to encourage favorable actions), and by modifying the move-selection probabilities in Monte-Carlo move selection. In both cases, it is possible to extract the necessary domain knowledge from a large database of played games.

7 Conclusions

In this paper we described an agent that learns to play Settlers of Catan. For move selection, the agent applies Monte-Carlo Tree Search, augmented with a limited amount of domain knowledge. The playing strength of our agent is notable: it convincingly defeats the hand-coded AI of JSettlers, and is a reasonably strong opponent for humans.

So far, applications of MCTS have been mainly constrained to Go. The success of our Settlers of Catan agent indicates that MCTS may be a viable approach for other multi-player games with complex rules.

Acknowledgments

The authors are supported by grants from the Dutch Organisation for Scientific Research (NWO grant 612.066.406 for the first and third author, NWO grant 612.066.409 for the second author).

References

1. Grabinger, R., Dunlap, J.: Rich environments for active learning: a definition. Association for Learning Technology Journal **3**(2) (1995) 5–34
2. Singh, S.P., Barto, A.G., Chentanez, N.: Intrinsically motivated reinforcement learning. In: Advances in Neural Information Processing Systems 17. (2005)
3. Dekker, S., van den Herik, H., Herschberg, I.: Perfect knowledge revisited. Artificial Intelligence **43**(1) (1990) 111–123
4. Laird, J., van Lent, M.: Human-level AI's killer application: Interactive computer games. AI Magazine **22**(2) (2001) 15–26
5. Sawyer, B.: Serious games: Improving public policy through game-based learning and simulation. Foresight and Governance Project, Woodrow Wilson International Center for Scholars Publication **1** (2002)
6. Schaeffer, J., van den Herik, H.: Games, computers, and artificial intelligence. Artificial Intelligence **134** (2002) 1–7
7. Caldera, Y., Culp, A., O'Brien, M., Truglio, R., Alvarez, M., Huston, A.: Children's play preferences, construction play with blocks, and visual-spatial skills: Are they related? International Journal of Behavioral Development **23**(4) (1999) 855–872
8. Huitt, W.: Cognitive development: Applications. Educational Psychology Interactive (1997)
9. van den Herik, H., Iida, H., eds.: Games in AI Research. Van Spijk, Venlo, The Netherlands (2000)
10. van den Herik, H.J., Uiterwijk, J.W.H.M., van Rijswijck, J.: Games solved: Now and in the future. Artificial Intelligence **134** (2002) 277–311
11. Marsland, T.A.: Computer chess methods. In Shapiro, S., ed.: Encyclopedia of Artificial Intelligence, J. Wiley & Sons (1987) 157–171
12. Pfeiffer, M.: Reinforcement learning of strategies for settlers of catan. In: Proceedings of the International Conference on Computer Games: Artificial Intelligence, Design and Education. (2004)
13. Thomas, R.: Real-time Decision Making for Adversarial Environments Using a Plan-based Heuristic. PhD thesis, Northwestern University, Evanston, Illinois (2003)
14. Billings, D., Davidson, A., Schaeffer, J., Szafron, D.: The challenge of poker. Artificial Intelligence **134**(1) (2002) 201–240
15. Sheppard, B.: World-championship-caliber scrabble. Artificial Intelligence **134**(1) (2002) 241–275
16. Kocsis, L., Szepesvári, C.: Bandit based monte-carlo planning. In: Proceedings of the 15th European Conference on Machine Learning. (2006) 282–293
17. Chaslot, G., Saito, J., Bouzy, B., Uiterwijk, J., van den Herik, H.: Monte-carlo strategies for computer go. In: Proceedings of the 18th BeNeLux Conference on Artificial Intelligence. (2006) 83–90
18. Chaslot, G., Winands, M., van den Herik, H., Uiterwijk, J., Bouzy, B.: Progressive strategies for monte-carlo tree search. New Mathematics and Natural Computation **4**(3) (2008) 343
19. Gelly, S., Wang, Y.: Exploration exploitation in go: UCT for monte-carlo go. In: NIPS-2006: On-line trading of Exploration and Exploitation Workshop. (2006)
20. Bouzy, B., Chaslot, G.: Monte-Carlo go reinforcement learning experiments. In: IEEE 2006 Symposium on Computational Intelligence in Games. (2006) 187–194
21. Chatriot, L., Gelly, S., Jean-Baptiste, H., Perez, J., Rimmel, A., Teytaud, O.: Including expert knowledge in bandit-based monte-carlo planning, with application to computer-go. In: European Workshop on Reinforcement Learning. (2008)