Monte-Carlo Tree Search for the Game of "7 Wonders"

D. Robilliard, C. Fonlupt, and F. Teytaud

LISIC, ULCO, Univ Lille–Nord de France, FRANCE

Abstract. Monte-Carlo Tree Search algorithm, and in particular with the Upper Confidence Bounds formula, provided huge improvements for AI in numerous games, particularly in Go, Hex, Havannah, Amazons and Breakthrough. In this work we study this algorithm on a more complex game, the game of "7 Wonders". This card game gathers together several known difficult features, such as hidden information, N-player and stochasticity. It also includes an inter-player trading system that induces a combinatorial search to decide which decisions are legal. Moreover it is difficult to hand-craft an efficient evaluation function since the card values are heavily dependent upon the stage of the game and upon the other player decisions. We show that, in spite of the fact that "7 Wonders" is apparently not so related to abstract games, a lot of known results still hold.

1 Introduction

Games are a typical AI research subject, with well-known successful results on games like chess, checkers, or backgammon. However, complex boardgames (sometimes nick-named Euro-games) still constitute a challenge, which has been initiated by such works as [17] or [18] on the game "Settlers of Catan", or [23] on the game "Dominion". Most often these games combine several characteristics among multiplayer, hidden information and random elements, together with little formalized expert knowledge on the subject. Monte-Carlo Tree Search (MCTS), which gained much notoriety from the game of Go, seems a method of interest in this context.

In order to simplify the obtaining of an AI, many published works use only a limited subset of the game rules: e.g. no trade interactions between players [18], or only a subset of the possible cards in [23]. In this paper we focus on the creation of a MCTS-based AI player for the recent game of "7 Wonders" (see also [11]). One of our goal is to tackle the full rules of the game, including the trading mechanism. The way we deal with the trading mechanism is quite a novelty as in the usual case more than one hundred new branches can be created just for the different ways of buying/selling goods. The simulation cannot cope with such a fact and we use a smart stochastic approach to deal with this problem introduced in Section 5.

In the first section we introduce the "7 Wonders" game and its rules, before presenting the MCTS heuristic. Then we focus on specific issues that arose during implementation, before presenting a set of experiments and their results.

2 "7 Wonders" Game Description

Boardgames are increasingly numerous, with more than 500 games presented each year at the international Essen game fair. Among these, the game "7 Wonders" (7W), issued

in 2011, obtains a fair amount of success, with about 100.000 copies sold per year. It is basically a card game, whose theme is similar to the many existing "civilization" games, where players develop a virtual country using production of resources, trade, military and cultural improvements.

Before heading onto the game mechanisms, let introduce a game classification based on their characteristics. A game can be:

– Fully or partially observable, depending on whether there is hidden information or not.
– Solitaire, two-player or multiplayer (standing for N-player with $N > 2$).
– Competitive or cooperative: in competitive games, players have their own goal, while they share the same objective in cooperative games.
– Deterministic or stochastic.

For instance, chess would belong to the family of fully observable, 2-player, competitive, deterministic games. The game of 7W is almost in opposite categories, being in the family of partially observable, multiplayer, stochastic, and also competitive games. While this game is competitive under this classification, note that in an N-player game with $N > 2$, several players may share cooperative sub-goals, such as hindering the progress of the current leading player. All these characteristics suggest that 7W is a difficult challenge for AI.

In a 7W game, from 3 to 7 players[1] are first given a random personal board among the 7 available, before playing the so-called 3 ages (phases) of the game. At the beginning of each game age, each player gets a hidden hand of 7 cards. Then there are 6 playing card rounds, where every player simultaneously selects a card from his hand and either:

– puts it on the table in his personal space;
– or puts it under his personal board to unlock some specific power;
– or can discard it for 3 units of the game money.

The last decision (or move) is always possible, while the first two possible moves depend on the player ability to gather enough resources from his board or from the production cards he already played in his personal space. He can also buy, with game money, resources from cards previously played by his left and right neighbors. This trading decision cannot be opposed by the opponent player(s) and the price is determined by the cards already played.

After playing their card, there is a so-called drafting phase, where all players give their hand of remaining cards to their left (age 1 and 3) or to their right (age 2) neighbor. Thus the cards circulate from player to player, reducing the hidden information. When there are less than 6 players, some cards from his original hand will eventually come back to every player . On the 6th turn, when the players receive only two cards in hand, they play one of the two and discard the other.

The goal of the game is to score the maximum victory points (VP), which are awarded to the players at the end of the game, depending on the cards played on the

---

[1] While the rule allows 2 player games, these are played by simulating a 3rd "dumb" player.

table, under the boards and the respective amounts of game money. The cards are almost all different, but come in families distinguished by color : resources (brown and grey), military (red), administrative (blue), trade (yellow), sciences (green) and guilds (purple). The green family is itself sub-divided between three symbols used for VP count. See Figure 1 for an illustration of a player situation.
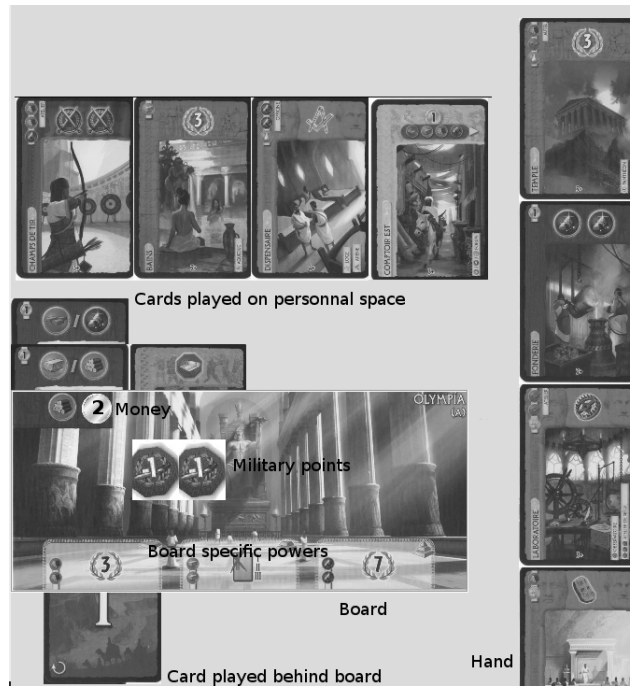

Fig. 1. Illustration of a player board and personal space in the middle of a game. Cards in hand are shown vertically on the right, cards played in the personal space are above the board with only the top of resources cards shown, cards put under the board are shown below the board with hidden face.

This game presents several interesting traits for AI research, that also probably explain its success among gamers:

– it has a complex scoring scheme combining linear and non linear features: blue cards provide directly VPs to their owner, red cards offer points only to the owner of the majority of red cards symbols, yellow ones allow to save or earn game money, green ones give their owner the number of identical symbols to the square, with extra points for each pack of three different symbols.
– resource cards have delayed effect : they mainly allow a player to put VPs awarding cards on later turns; this is also the case of green cards that, apart from the scoring of symbols, allow some other cards to be played for free later on.

- there is hidden information when the players receive their hand of cards at the beginning of each age of the game.
- there is a great interactivity between players as they can buy resources from each others to achieve the playing of their own cards. Some cards also give benefits or VPs depending on which cards have been played by the neighbors. Moreover the drafting phase confronts players with the difficult choice of either playing a card that gives them an advantage, or another card that would give a possibly greater advantage to one of the neighbors that would receive it after the drafting phase.
- the game is strongly asymmetric relatively to the players, since all player boards are different and provide specific powers (such as resources, or military symbols). Most of these benefits are available when playing a card under the personal board at a given cost in resources. Thus some boards are oriented towards specific strategies, such as maximizing the number of military symbols, or increasing the bonuses of collecting green cards symbols, for example.

The number of different cards (68 for 3 players, from those 5 are removed randomly) and the 7 specific boards, together with the delayed effect of many cards and the non linear scoring, make it difficult to handcraft an evaluation function. Notably, the number of VPs gained in the first game age is a bad predictor of the final score, since scoring points at this stage of the game usually precludes playing resource cards that will be needed later on.

We can give an approximation of the state space size for 3 players, by considering that there are 68 possible different cards, from those each player will usually play 18 cards. We thus obtain $\binom{68}{18} \times \binom{50}{18} \times \binom{32}{18} = 1E38$ possible combinations, neglecting the different boards and the partition between on-table and behind-the-board cards (which would increase that number).

3 Monte-Carlo Tree Search

The Monte-Carlo Tree Search algorithm (MCTS) has been recently proposed for decision-making problems [15, 9, 5]. Applications are numerous and varied, and encompass notably games [13, 16, 4, 1, 19]. In games MCTS outperforms alpha-beta techniques when evaluation functions are hard to design. The most known implementation of MCTS is Upper Confidence Bound (UCT), that is presented below. Enhancements have also been proposed, such as progressive widening [8, 7, 20], that is described at the end of this section.

3.1 UCT description

Let us first define two functions: `mc(s)` which plays a uniform random decision (move) from the situation s and returns the new position, and `result(s)` which returns the score of the final situation s. The idea is to build an imbalanced partial subtree by performing many random simulations from the current state, and simulation after simulation biasing these simulations toward those that give good results. The construction is then done incrementally and consists in three different parts : *descent*, *evaluation* and *growth*, illustrated in Fig. 2.

The *descent* is done by using a bandit formula, i.e. by choosing the node j among all possible nodes C^s which gives the best reward according to the formula :

$$s' \leftarrow \underset{j \in C^s}{\arg\max} \left[\bar{x}_j + K_{UCT} \sqrt{\frac{ln(n_s)}{n_j}} \right]$$

with $\bar{x}_j$ the average reward for the node j (it is the ratio of the number of victories over the number of simulations, thus belonging to interval $[0, 1]$), n_j the number of simulations for the node j, n_s is the number of simulation in s, and $n_s = \sum_j n_j$. K_{UCT} is the exploration parameter and is used to tune the trade-off between exploitation and exploration. At the end of the *descent* part, a node which is outside the subtree has been reached. In order to evaluate this new node, a so-called *playout* is done: random decisions are taken until the end of the game, when the winner is known. The last part is the *Growth* step which consists in simply adding the new node to the subtree, and updating all the nodes which have been crossed by the simulation. The algorithm is presented in Alg.1.

Algorithm 1 MCTS

---

argument node s, MCTS subtree $\hat{T}$
while there is some time left **do**
    $s' \leftarrow s$
    Initialization: $game \leftarrow \emptyset$
    // DESCENT
    while s' in $\hat{T}$ and s' not terminal **do**
        $s' \leftarrow \underset{j \in C^{s'}}{\arg\max} [\bar{x}_j + K_{UCT} \sqrt{\frac{ln(n_{s'})}{n_j}}]$
        $game \leftarrow game + s'$
    $S \leftarrow s'$
    // EVALUATION
    while s' is not terminal **do**
        $s' \leftarrow mc(s')$
    $r = \text{result}(s')$
    // GROWTH
    $\hat{T} \leftarrow \hat{T} + S$
    for each s in $game$ **do**
        $n_s \leftarrow n_s + 1$
        $\bar{x}_s \leftarrow \frac{(\bar{x}_s * (n_s - 1) + r)}{n_s}$

In our implementation, a single N-player game turn (corresponding to the N player simultaneous decisions), is represented in the MCTS subtree by N successive levels, thus for a typical 3-player game with 3 ages and 6 cards to play per age, we get $3 \times 6 = 18$ decisions per player and the depth of the subtree is $18 \times 3 = 54$. Of course we keep the "simultaneous decisions" property, that is the state of the game is updated only when reaching a subtree level whose depth is a multiple of N, thus successive players (either real or simulated) make their decision without knowing their opponent
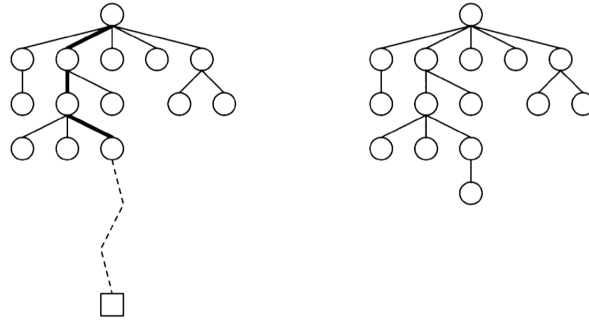
Fig. 2. Illustration of the MCTS algorithm from [3]. Circles represent nodes and the square represents the final node. On the left subtree, the descent part (guided by the bandit formula in the UCT implementation) is illustrated in bold. The evaluation part is illustrated with the dotted line. The subtree after the growth stage is illustrated on the right: the new node has been added.

choices. The average arity of a node can be estimated: a player has 4 cards in hand on average that can possibly be played in 3 different ways: discard, on the table, and behind the board. These two last options can usually be accomplished by different trading options, 2 trading options being common. This leads to an estimated average arity of $4 * (1 + 2 \times 2) = 20$ children per node.

3.2 Progressive widening

UCT algorithms are efficient methods for balancing exploration and exploitation, however, only few information are provided for decisions loosely explored. Several enhancements have been provided to tackle this problem. The most famous are Progressive Widening [8, 7, 20] and Rapid Action Value Estimate [13].

The principle of progressive widening is to rank possible decisions according to some heuristics and to discard certain decisions while the number of simulations is not large enough. More precisely, let us rename decisions $y_1 \ldots y_n$, with $i < j$ when decision y_i is better than y_j according to the heuristic. At the mth simulation, all decisions with an index larger than $f(m)$ are discarded. The following formula: $f(m) = \lfloor Qm^P \rfloor$ performs well in the literature [8]. We let the parameter $Q = 1.0$ and tune only P, as reported in the experiment section.

We have chosen a simple ordering heuristic based on human play. This ranking consists in having first alternative resource type cards, then single resource type cards, then military cards, followed by science cards. Other cards are left unordered since for trade cards it is difficult to assess an a priori order, and for civil cards we can expect that they are easier to evaluate by MCTS, since their reward is mostly independant of other cards.

4 MCTS and 7 Wonders

In this section we present how we dealt with partial information and weak moves in playouts.

4.1 Handling partial information

In order to handle partial information, we used the determinization paradigm, see [14, 2]. This consists in choosing decisions via several simulations of perfect information games that are consistent with what is known of the real, imperfect information, game state. We keep a single MCTS subtree, and the MCTS AI records the whole set of possible cards in its opponents hands. Any card revealed during play implies a reduction of the opponents possible cards in hand. When a simulation is done, we perform a determinization by an equiprobable random draw of a real size hand for each opponent, from their set of possible cards. The MCTS subtree is then descended, ignoring children nodes that are not playable with the current determinization, and adding newly available children nodes if required. This way of doing seems similar in principle to what is called information set UCT in [21].

Coping with weak decisions A 7W player has always the choice of putting a card on the discard pile to earn 3 coins of game money. This is the sole option available when the resources needed to play any card are not available, but it can also be a tactical choice, e.g. in order to have enough money to buy resources from neighbors on a later turn. However, most of the time discarding is a worse than average move: notably 3 coins amount to 1 VP, while played cards should bring an average reward of about 3 VPs per card. As any one card can be discarded for money, while not all cards can usually be played, depending on available resources, thus discards are the most common moves and would be the dominant moves explored by a fully random playout procedure. This results in non-informative playouts and a weak MCTS player. The presence of such a class of weak moves is not uncommon in other games too, for instance in Go, all programs discard the "empty triangle" move from their playouts. We act similarly in the playouts, allowing discards only when no other move is possible. On the contrary we keep all moves in the MCTS subtree construction, in order to allow for tactical discards.

5 Managing trading decisions

In order to play a card in 7W, a player may choose to buy resources either from the right, left or from both of his two neighbors, at possibly different costs, depending on the resource type. Moreover some of the resource producing cards provide alternate choices: e.g. the "Caravanserai" card provides one unit of either ore, wood, brick or stone. These two game rules induce a combinatorial tree of possible resource trade choices, and it is not uncommon that this tree has several hundred branches, especially during the 3rd game age. Some of the branches are dead end, that is after setting some trade choices, it appears that some required resources cannot be gathered or are now

unaffordable to the player. Other branches may offer valid and affordable trade choices, that constitute as many possible game decisions. Note that there is such a tree of possible trade decisions for every card in hand.

While the exploration of the whole tree for every card is recommended for building the MCTS subtree, since one does not want to forget a possible card playing decision, this would greatly impact the speed of play when it is done in playouts. Nonetheless it is not possible to ignore the trade decision tree, since truly random decisions would make nonsense in most cases (such as trying to buy resources from a player that do not own them, or that own them in insufficient number). Thus some sort of exploration of the trade tree must be done.

We dealt with that issue in playouts by imposing a random order on the branches of the tree of possible trade choices, and fixing a hard limit L on the number of branches explored, depending on the game age: 3rd age cards generally award more VPs and require more resources, so it is sensible to spend more exploration time before deciding if they are playable or not in playouts. If we cannot find a suitable branch in the first L branches explored, then we consider that the card cannot be played (either on the table or under the board), thus it must be discarded.

In Table 1 we show the number of discards in playouts. This number of discards is listed per each game age and for 4 different values of the hard limit L on branch exploration (this is done for 3 players and 15000 simulations per game turn and there is some variance due to the game randomness). Some hands of cards really do not contain any playable card, thus a discard is compulsory by the game rules, but in other cases there are playable cards that are not seen if L is too low: the exploration of trading choices is ended too early. Indeed we see in Table 1 that the number of discards increases when putting too strict limits on trade exploration, meaning that we drop cards that could have been played by allowing a longer search.

In the Table 1 we see that discards are more common in the first age and are not affected by our limit thresholds. This is explained by the fact that not many resources have usually been played by random playouts in the first age, thus discards are often the sole possible decisions. On the contrary, the L threshold impacts the successful discovery of playable moves in age 2 and 3. Discards remain more common in age 3 than in age 2, since age 3 cards require more resources.

Table 1. Number of discards in playouts depending on game age and on the hard limit on the number of trading branches explored.

Game age	Limit L on branches explored			
	16 br.	24 br.	32 br.	48 br.
1	35908	40304	38080	36340
2	14540	6727	6009	6191
3	67171	15578	11370	11314

By setting a limit of 8, 24 and 32 branches on respectively age 1, 2 and 3, we gained a more than 5 times speedup against the exploration of the whole trading tree. This

allows for about 1000 simulations per second in age 1. The loss of precision in playouts (discards that could have been avoided with a longer search) is more than compensated for by the greater number of simulations allowed in the same time. Note that humans are also confronted to the same problem, and it is not uncommon that a player thinks a card is not playable, while it really is.

6 Experiments and results

The objective of the experiments is to study the level of efficiency of MCTS to play successfully at 7W. This is done in comparison with a simple handcrafted rule-based AI presented below, and also by studying several values of MCTS parameters and enhancements, such as progressive widening.

Note that our MCTS AI was also successfully opposed to some experienced human players, even if the number of plays (and players) cannot yet be considered significant and reported here.

6.1 Rule-based AI implementation

The rule-based AI (rbAI) is deterministic and is managed, in age 1 and 2, along the principles listed below by priority order (when a card is "always played", it means of course if it is affordable):

- a card providing 2 or more resource types is always played;
- a card providing a single resource type that is lacking to the rbAI is always played;
- a military card is always played if rbAI is not the only leader in military, and the card allows rbAI to become the (or one of the) leading military player(s);
- the civil card with the greatest VP award is always played;
- a science card is always played;
- a random remaining card is played if possible, else a random card is discarded.

In the third game age, the set of rules is superseded by choosing the decision with best immediate VP reward.

6.2 Experimental setting

All experiments are composed of 1000 runs, and the number of MCTS simulations per game turn is given for the 1st of the 3 ages of the game. This number is multiplied respectively by 1.5 and 2 in the 2nd and 3rd ages, since the shorter playouts allow more simulations in the same time (so a "1000 simulations" means respectively 1000, 1500 and 2000 simulations per turn in ages respectively 1, 2 and 3).

In the MCTS vs rbAI test, we use one instance of MCTS versus 2 instances of the rbAI. Iteratively, 20 sets of personal boards are drawn, and 50 independent random cards distributions are played on every board set.

In the MCTS parameter tuning experiments, we use one instance of the rbAI versus 2 instances of MCTS with different parameters/enhancements, called MCTS-1 and MCTS-2. We draw iteratively 5 board sets where the same board is duplicated for the

two MCTS (this duplication of boards is not allowed in the original game rules but is handy to ensure that the MCTS comparison is not biased by the strength of the different boards). For each board setting, 100 independent random card distributions are played, then the two different MCTS instances swap position and the same 100 card distributions are played again. This is to remove any bias that could be generated by the position of each MCTS relatively to the rbAI player.

6.3 MCTS against simple AI

In this section we compare the success rate of one MCTS player against two instances of the simple rule-based AI. Table 2 shows that MCTS is clearly superior. Adding more simulations improves the MCTS success rate, although with diminishing returns as the number of simulations increases, as expected by the theory.

Table 2. Comparisons of MCTS success rate (SR) versus rule-based AI (rbAI) for several number of simulations per game turn (mean value $\pm$ 95% confidence interval).

# sim.	SR MCTS	SR rbAI
125	$67.63\% \pm 2.89$	$32.37\% \pm 2.89$
250	$81.87\% \pm 2.38$	$18.13\% \pm 2.38$
500	$87.25\% \pm 2.06$	$12.75\% \pm 2.06$
1000	$92.63\% \pm 1.62$	$7.37\% \pm 1.62$

6.4 Comparison of MCTS with different parameters

Tuning of the Bandit formula First Table 3 shows a comparison of success rate for several values of the exploration constant K_{UCT}, on 1000 games, with 1000 simulations per game turn for each MCTS player. The success rates of the rbAI player are very low when using this number of simulations for MCTS and are not reported here (thus the success rates displayed do not sum up to 100%). The experiment show that good K_{UCT} values can be obtained in the range $[0.3, 1.0]$, although values strictly greater than 0.3 do not yield no much significant improvement. This good 0.3 value is slightly superior to the standard 0.2 found in the literature. This might be explained by the fact that some moves may appear very attractive in a few simulations (e.g. playing a military card) while some other good moves need more simulations to show their robustness (e.g. playing a science card that also allows some good later cards for free).

Scaling of UCT A second set of experiments, in Table 4 explores the impact of the number of MCTS simulations per game turn. The K_{UCT} value is set to 0.3 for all experiments in this table. For each experiment we compare a given number of simulations against twice as many simulations. The expected gain decreases when the number of simulations rises, which is consistent with the literature (see [12]).

Table 3. Comparisons of MCTS success rates (SR) for different values of K_{UCT} and 1000 simulations per game turn (mean value $\pm$ 95% confidence interval, rule-based AI is not reported).

K_{UCT} MCTS-1	K_{UCT} MCTS-2	SR MCTS-1	SR MCTS-2
0.1	0.2	$34.20\% \pm 2.94$	$64.2\% \pm 2.97$
0.2	0.3	$42.60\% \pm 3.06$	$55.90\% \pm 3.08$
0.3	0.4	$48.00\% \pm 3.10$	$51.30\% \pm 3.10$
0.3	0.5	$48.70\% \pm 3.10$	$51.20\% \pm 3.10$
0.3	0.7	$49.10\% \pm 2.68$	$50.52\% \pm 2.68$
0.3	1.0	$48.30\% \pm 3.10$	$51.20\% \pm 3.10$

Table 4. Comparisons of MCTS success rates (SR) for different number of simulations per game turn (mean value $\pm$ 95% confidence interval, rule-based AI is not reported).

# sim. MCTS-1	# sim. MCTS-2	SR MCTS-1	SR MCTS-2
125	250	$34.50\% \pm 2.95$	$62.00\% \pm 3.01$
250	500	$39.00\% \pm 3.02$	$59.40\% \pm 3.04$
500	1000	$43.90\% \pm 3.08$	$55.50\% \pm 3.08$
1000	2000	$44.80\% \pm 3.08$	$54.60\% \pm 3.09$
2000	4000	$46.20\% \pm 3.09$	$53.50\% \pm 3.09$
4000	8000	$44\% \pm 3.08$	$\% 55.90 \pm 3.08$

Progressive widening We experiment the progressive widening enhancement with 1000 simulations per move and $K_{UCT} = 0.3$. Several values for the P parameter are experimented and results against a standard MCTS are presented in Table 5. Except when P is too small and reduces too much the MCTS exploration, we obtain an improvement for several values of P. While the improvement is small, it appears quite robust in front of P. One point is that we have the same sorting for all moves for all ages. Maybe we should have an independent progressive widening for each age, as the importance of a family of moves could be different in different stages of the game.

Using the real score of the game It has been shown that using an evaluation function instead of a Monte-Carlo policy can improve the global strength of the MCTS algorithm [16, 22]. However, building such a function is not always possible and sometimes Monte-Carlo evaluations is the only choice. Sometimes, an intermediate solution consists in taking the real score of the game in order to bias the bandit formula. This is only possible when such a score exists and results are moderate. For instance, in the game of Go, there is only a very small (but significant) improvement [10]. One emphasis reason is that it becomes too greedy to win by more points and takes risks. We try to use the real score to bias the bandit formula as this score exists in the original game. Following

Table 5. Comparisons of MCTS success rates (SR) without or with progressive opening of sub-trees for different values of the P parameter (mean value $\pm$ 95% confidence interval, rule-based AI is not reported).

P	SR without progressive MCTS-1	SR with progressive MCTS-2
0.15	58.60% $\pm$ 3.05	40.70% $\pm$ 3.04
0.25	46.40% $\pm$ 3.09	53.20% $\pm$ 3.09
0.35	46.40% $\pm$ 3.09	53.30% $\pm$ 3.09
0.45	46.90% $\pm$ 3.09	52.70% $\pm$ 3.09

[6], the bandit formula becomes then

$$score_j \leftarrow \bar{x}_j + K_{UCT} * \sqrt{\frac{ln(n_s)}{n_j}} + \frac{K_{score}}{\log(n_j)} * RealScore$$

With this formula, the impact of the real score decreases with the number of simulations.

Results with numerous K_{score} are presented in Table 6. We can note two things : first using only the real score, which is approximated by using a large K_{score} does not work. The problem with this tuning is that the MCTS algorithm tries to win as many points as possible, even if it is a risky strategy. This is not reasonable, since it is better to be sure to win by only one point than to take risks to win by more points. Second, with only a small help of the real score ($K_{score} = 0.1$) it seems possible to obtain a small improvement, although not very significant. Both these results are consistent with the previous literature; in [10] the help of the real score gives a gain of only 0.02 for the game of Go.

Table 6. Comparisons of MCTS success rates (SR) without or with of the use of the real score game for biasing the tree policy (mean value $\pm$ 95% confidence interval, rule-based AI is not reported).

K_{score}	SR without the real score MCTS-1	SR with the real score MCTS-2
0.01	50.00% $\pm$ 3.10	49.70% $\pm$ 3.10
0.05	50.30% $\pm$ 3.10	49.20% $\pm$ 3.10
0.10	47.40% $\pm$ 3.10	52.00% $\pm$ 3.10
0.15	51.30% $\pm$ 3.10	48.00% $\pm$ 3.10
0.20	48.50% $\pm$ 3.10	50.60% $\pm$ 3.10
0.50	60.00% $\pm$ 3.04	39.60% $\pm$ 3.03
1.00	69.40% $\pm$ 2.86	29.60% $\pm$ 2.83
10.00	76.50% $\pm$ 2.63	22.20% $\pm$ 2.58

7 Conclusion and future works

In this paper we have shown the interest of the Monte Carlo Tree Search algorithm in the case of the 7W complex boardgame, using full rules. The game of 7W presents several hard features: multiplayer, hidden information, random elements and a complex scoring mechanism, that renders difficult to handcraft an evaluation function. In this context, the MCTS method obtains convincing results, both against a human designed rule-based AI and against experienced human players.

However, the implementation is not straightforward: we use determinization to handle the hidden information element, and we refine the playouts by suppressing a class of weak moves. Moreover the computing cost of exploiting all possible trading decisions in the playouts is too large to allow enough simulations for real-time play against humans. We solve this problem by approximating the set of allowed trading decisions. Thus the gap between MCTS theory and practice is not negligible.

We notice that the various parameter effects (K_{UCT}, scalability, progressive widening, and adding a score information) are quite similar to what is observed in classic abstract games, despite the fact that this game seems substantially different.

Future works consist in implementing the use of the Rapid Action Value Estimate (RAVE) enhancement [13], which is one the most powerful improvement for several games [13, 19]. Another interesting work should be to analyze and improve the enhancements tried in this paper. In particular, having one progressive widening per age seems to be a good idea. The real score could also be incorporated in a similar way, as the relevance of its impact is probably bigger in the last stages of the game. Parallelization of the playouts could have both advantages of increasing the level of play by using more simulations, and allowing enough time to explore all trading moves in playouts. Last but not least, we plan to interface our AI with a gaming website in order to obtain a better assessment of its game level through the confrontation with more human players.

References

1. Arneson, B., Hayward, R.B., Henderson, P.: Monte-Carlo tree search in Hex. Computational Intelligence and AI in Games, IEEE Transactions on 2(4), 251–258 (2010)
2. Bjarnason, R., Fern, A., Tadepalli, P.: Lower bounding Klondike Solitaire with Monte-Carlo planning. In: ICAPS (2009)
3. Browne, C.B., Powley, E., Whitehouse, D., Lucas, S.M., Cowling, P.I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S.: A survey of Monte-Carlo tree search methods. Computational Intelligence and AI in Games, IEEE Transactions on 4(1), 1–43 (2012)
4. Cazenave, T.: Monte-Carlo Kakuro. In: van den Herik, H.J., Spronck, P. (eds.) ACG. Lecture Notes in Computer Science, vol. 6048, pp. 45–54. Springer (2009), http://dblp.uni-trier.de/db/conf/acg/acg2009.html#Cazenave09
5. Chaslot, G., Saito, J.T., Bouzy, B., Uiterwijk, J., Van Den Herik, H.J.: Monte-Carlo strategies for computer Go. In: Proceedings of the 18th BeNeLux Conference on Artificial Intelligence, Namur, Belgium. pp. 83–91 (2006)
6. Chaslot, G., Fiter, C., Hoock, J.B., Rimmel, A., Teytaud, O.: Adding expert knowledge and exploration in Monte-Carlo tree search. In: Advances in Computer Games, pp. 1–13. Springer (2010)

7. Chaslot, G.M.J., Winands, M.H., HERIK, H.J.V.D., Uiterwijk, J.W., Bouzy, B.: Progressive strategies for Monte-Carlo tree search. New Mathematics and Natural Computation 4(03), 343–357 (2008)
8. Coulom, R.: Computing elo ratings of move patterns in the game of Go. In: Computer games workshop (2007)
9. Coulom, R.: Efficient selectivity and backup operators in Monte-Carlo tree search. In: Computers and games, pp. 72–83. Springer (2007)
10. Enzenberger, M., Muller, M., Arneson, B., Segal, R.: Fuegoan open-source framework for board games and Go engine based on Monte-Carlo tree search. Computational Intelligence and AI in Games, IEEE Transactions on 2(4), 259–270 (2010)
11. Gardiner: A proposal for an agent that plays 7 Wonders. Tech. rep., Willamette University (2012), `http://www.willamette.edu/~bgardine/Thesis_Files/ben_gardiner_proposal_final.pdf`
12. Gelly, S., Hoock, J.B., Rimmel, A., Teytaud, O., Kalemkarian, Y., et al.: On the parallelization of Monte-Carlo planning. In: ICINCO (2008)
13. Gelly, S., Silver, D.: Combining online and offline knowledge in uct. In: Proceedings of the 24th international conference on Machine learning. pp. 273–280. ACM (2007)
14. Ginsberg, M.L.: Gib: Imperfect information in a computationally challenging game. J. Artif. Intell. Res.(JAIR) 14, 303–358 (2001)
15. Kocsis, L., Szepesvári, C.: Bandit based Monte-Carlo planning. In: Machine Learning: ECML 2006, pp. 282–293. Springer (2006)
16. Lorentz, R.J.: Amazons discover Monte-Carlo. In: Computers and games, pp. 13–24. Springer (2008)
17. Pfeiffer, M.: Reinforcement learning of strategies for Settlers of Catan. In: Q, M., N, G., S, N., D, A.D. (eds.) 5th international conference on computer games: artificial intelligence, design and education. pp. 384–388 (2004)
18. Szita, I., Chaslot, G., Spronck, P.: Monte-Carlo tree search in Settlers of Catan. In: Advances in Computer Games, pp. 21–32. Springer Berlin Heidelberg (2010)
19. Teytaud, F., Teytaud, O.: Creating an upper-confidence-tree program for Havannah. Advances in Computer Games pp. 65–74 (2010), `http://hal.inria.fr/inria-00380539/en/`
20. Wang, Y., Audibert, J.Y., Munos, R., et al.: Infinitely many-armed bandits. Advances in Neural Information Processing Systems (2008)
21. Whitehouse, D., Powley, E.J., Cowling, P.I.: Determinization and information set Monte-Carlo tree search for the card game Dou Di Zhu. In: Computational Intelligence and Games (CIG), 2011 IEEE Conference on. pp. 87–94. IEEE (2011)
22. Winands, M.H., Björnsson, Y.: Evaluation function based Monte-Carlo LOA. In: Advances in Computer Games, pp. 33–44. Springer (2010)
23. Winder, R.K.: Methods for approximating value functions for the Dominion card game. Evolutionary Intelligence (2013)