Monte Carlo Tree Search in a Modern Board Game Framework

G.J.B. Roelofs

Januari 25, 2012

Abstract

This article describes the abstraction required for a framework capable of playing multiple complex modern board games, and provides a proof-of-concept implementation of Settlers of Catan within said framework. Monte Carlo Tree Search (MCTS) is chosen as the basis for our framework, because of its lack of reliance on domain knowledge. The domain implementation is validated, and used to experiment on two proposed structure changes to MCTS for non-deterministic games with a high branching factor. The first technique proposed is a simplification of the Chance Node model as seen in Expectimax. The second technique outlines the introduction of move groups within the tree structure of MCTS to significantly reduce its branching factor. We conclude that both techniques are equivalent to MCTS in terms of playing strength when a sufficient amount of simulations can be guaranteed.

1 Introduction

Traditionally, the focus of games research has been on *deterministic* games with *perfect information*. However, as the field progressed and games have become increasingly complex, this focus has started to shift towards games with different features, such as *non-determinism* and *imperfect information*.

Another avenue of research that has seen recent activity is General Game Playing (GGP). The goal of GGP is to construct an algorithm which is capable of expertly playing previously unknown games of a wide variety, without being given any domain knowledge in advance. This is in contrast with traditional game play algorithms, which are designed to play one particular game well.

To further research in GGP, the Game Description Language (GDL) was created, a language in which to describe different kinds of deterministic, perfect information games. [12]
Effort was recently made to extend the GDL to cope with imperfect, non-deterministic games. [16] However, so far there have been no successful implementations of complex modern board games within GDL.

By analysing the set of actions available in many board games, abstract actions can be defined which are powerful enough to completely describe each domain. As such we propose a framework which implements these abstract actions and the underlying logic to support them. By adhering the domain implementation to this abstraction layer, experiments are no longer restricted to a singular domain, further bridging the gap between the current GDL specification and custom, domain-specific implementations. One of the challenges faced will be to provide an implementation of this abstraction layer that is extensive enough to describe multiple games, but efficient enough to still use as a basis for experimentation.

Monte-Carlo Tree Search (MCTS) [5, 10] has been shown to be a strong search algorithm for cases where little to no domain knowledge is known [1]. As the framework will eventually be used to implement different game domains, MCTS is chosen as the primary search technique to be explored.

A prime example of a modern board game, is the popular Settlers of Catan. Due to its complexity, it is ideally suited to prove the validity of the framework and as such, was chosen as the domain for this article.

Research has already been done in comparing MCTS against traditional search techniques and rule-based heuristic play for Settlers of Catan [15]. The goal of this article is to investigate the application of MCTS in our framework and propose two novel general-purpose techniques to augment MCTS.

One of the problems introduced by the selection strategies in MCTS is that a high branching factor can hide obvious replies [1]. One aspect that will be explored therefore is how this branching factor could be reduced through the use of move groups [13, 18].

A new model for chance nodes is outlined, we call Grouped Chance Nodes. In Grouped Chance Nodes the strength of MCTS, the repeated playouts, is used to converge to the equivalent of chance nodes.

2 Framework

In modern board games abstract tasks can be identified that alter the game state, independent of the game domain. These abstract tasks will be termed *Actions*. By separating them from the context of the game, a list of actions can be provided to an algorithm, independent of the game domain.

As a detailed description of the framework is beyond the scope of this article, an overview and description of the crucial aspects is given.

The basis of the framework is a state machine supported by an event-driven graph structure to represent the board upon which pieces (called *Placeables*) can be placed.

The state machine has multiple *State Cycles*, each containing one or more *Game States* representing the phases of the turn of a player. Each *State Cycle* represents a different state of the game.

State Cycles have an Activation and Deactivation Predicate associated with them which will be checked on the move to a new *Game State*. Whenever both the Deactivation Predicate of the current and Activation of the

next *State Cycle* are met, the state machine will move to the next *State Cycle*.

All possible *Actions* defined by the framework should provide two variants; one defined as to be called by the user, requiring input, and one defined as an automated function. The framework will provide all legal options for an action, each option accompanied by the chance of occurrence, and a unique identifier.

A distinction must be made between the abstract actions a player is allowed to perform, termed *Actions*, and specialized functions of game logic specific to a game domain, termed *Game Logic*. The distinction being that *Actions* could specify that a *Game Logic* may be played, while the *Game Logic* specifies how the gamestate is altered.

An example of specifying a game domain using these concepts with regards to the domain of Settlers of Catan can be found in Appendix A.

2.1 State Cycle

The *State Cycle* possesses one or more *Game States*, and describes a phase in the game. It keeps track of the current player, round and current *Game State*. If a player tries to move to a new *Game State*, and none are available, the next player is given a turn. If each player has had their turn, the round number is increased. A *State Cycle* has multiple triggers to which further logic may be bound:

On Activation **On Deactivation**

2.2 Game State

The *Game State* describes the phase of a players turn, and has multiple triggers to which further logic may be bound:

On Activation **On Deactivation**

2.3 Placeable

A *Placeable* is an object that can be placed on a given type of element (Vertex, Edge or Area) of the Board according to a Placement Predicate, and bought if a Cost Predicate is satisfied. A *Placeable* has multiple triggers, to which further logic may be bound:

On Place	**On Place Neighbour**
On Remove	**On Remove Neighbour**
On Activation	**On Deactivation**
	On Ownership Change

2.4 Game Logic

An *Game Logic* is a specialised function of game logic, normally restricted to a specific game domain, acquirable and playable by either player. Furthermore, an *Game Logic* has an Activation Predicate associated with it, which is initialized upon acquirement of the *Game Logic* by the player, and must be met before the *Game Logic* can be played. An *Action* has multiple triggers, to which further logic may be bound:

On Activation **On Ownership Change**

2.5 Action

Actions are defined as the abstract actions a player is allowed to do. The list of *Actions* which may be exposed to the algorithm as possible actions, is as follows:

Place Placeable
Places a Placeable on the graph given a target location, after checking whether the Placeable can be placed on the given target. If no target element is given, the player is asked for a selection out of a list of targets, generated according to the Placement Predicate.

Remove Placeable
Removes a *Placeable* from the graph and the game.

Acquire Placeable
Gives a *Placeable* to a target player, and then calls Place Placeable.

Purchase Placeable
Checks whether a *Placeable* can be bought according to its Cost Predicate, deducts the costs if so, and calls Acquire Placeable, followed by Place Placeable.

Acquire Action
Gives an *Action* to the target player.

Purchase Action
Checks whether an *Action* can be bought according to its Cost Predicate, deducts the costs if so, and calls Acquire Action.

Remove Action
Removes an *Action* from player possession.

Play Action
Executes the specialised game logic associated with the *Action* if the Activation Predicate of the *Action* is met.

Offer Trade Player
Sets the current trade offer of the Player for trading between Players.

Accept Trade Player
Accepts a offer from another player, if both players accept the trade, the trade is executed.

Accept Trade Bank
Accepts and initiates a trade with the bank, cancels any active trade offers.

Cancel Trade
Cancels a trade offer from the player.

Next Game State
Attempts to move the game to the next state. Depending on a given variable, the game will also attempt to move to the next state cycle.

Modify Resource
Modifies the resource of a player.

3 Settlers of Catan

Settlers of Catan is a 2-6 player board game designed by Klaus Teuber and first published in 1995, after which several extensions were released. This paper will focus on the core ruleset for 4 players. The goal of this game is to be the first to achieve at least 10 *victory points*.

The game board consists of 16 hexagonal tiles, each representing either:

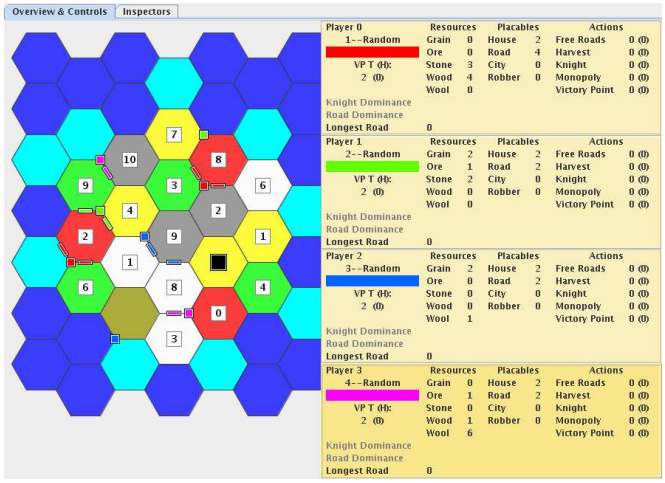

Figure 1: *An example board setup of Settlers of Catan*

- One of five *resources: wood, stone, sheep, wheat* and *ore*;
- A non-producing type *desert*, or *sea*;
- Or a *port*, a tile which gives a bonus to the trade ratio.

Each resource based tile has a number ranging from 2 to 12 associated with it.

The turn of a player consists of two phases. In the initial phase, called the production phases, a player must roll two dice, the sum of which determines which resource tiles are activated. Upon activation, any player owning either a Settlement or City adjacent to the tile is given one or two of the resource according to the type of the tile. On a dice roll of 7, any player in possession of more than 7 resources must discard half of them (rounded down). The current player then moves the robber, a piece which blocks the field it is placed on from activation in the production phase. The current player is then allowed to steal a random resource of any player in possession of a construction adjacent to the blocked tile.

In the second phase a player may, but is not required to, trade resources; build a construction; buy a Development Card or play a single Development Card.

The player is allowed to trade resources with his opponents, or the bank according to a given trade ratio. The default trade ratio is 4 similar resources to a resource of choice. By building a settlement at one of the port tiles, these trade ratios are adjusted according to the rules of the port.

A player is allowed to purchase the following constructions: *Road, Settlement* and *City*. The Road costs *1 Stone and Wood*, and can be placed on any Edge adjacent to any Construction in possession of the Player. The Settlement costs *1 Stone, Wood, Wool and Grain* and can be placed on any Vertex which is adjacent to a Road owned by the Player, and respecting the distance rule: No other settlement may be placed within a 2 Edge distance of an existing settlement or city. The City costs *3 Ore and 2 Wheat* and replaces any existing settlement owned by the player. No construction may be built on an already occupied location.

The player is also allowed to draw a random Development Card for *1 Ore, Grain and Wool*. The Development Cards are: *Victory Point Card*; which gives the player a hidden victory point, *Knight Card*; which activates the Robber, *2 Free Roads*, *2 Resources of Choice* or *Monopoly* which steals all resources of a type from all other players. A Development Card, with exception of the VP Card, may not be used in the round it was purchased, and are discarded after use.

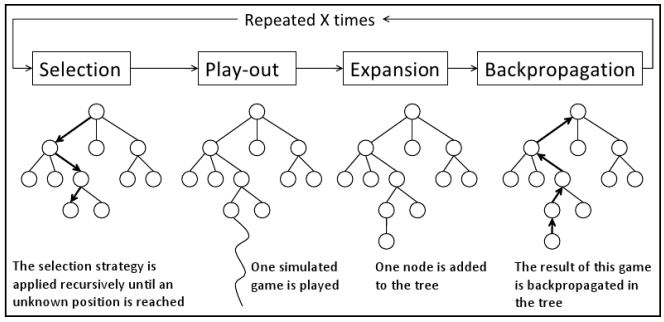

Figure 2: *Outline of the Monte-Carlo Tree Search [19].*

Players gain 1 *victory point* per Settlement, and 2 *victory points* per City owned. The Victory Point Development Card gives the player 1 hidden victory point. The player who has played the Knight Development card the most, with a minimum of 3 times, receives 2 victory points. The player with the longest unbroken link of roads, with a minimum of 5, also receives 2 victory points.

Two pure strategies exist for Settlers of Catan, the *Ore-Grain* strategy, and the *Stone-Wood* strategy. As indicated by its name, the *Ore-Grain* strategy focusses on acquiring the *Ore* and *Grain* resource, building *Cities* and purchasing *Development Cards*. The *Stone-Wood* strategy in contrast focuses on building *Roads* and *Settlements* [15].

3.1 Rule Changes

Several simplifications suggested by Szita, 2010 [15] deal with aspects of the game which are non-trivial in implementation, but do not change the core game-play itself. No trading between players is allowed, and all imperfect information (which development cards are bought, which resources are stolen) is removed from the game. The removal of these aspects of imperfect information do not alter the game play, as they are usually quickly revealed or inferred to be victory points.

4 Monte Carlo Tree Search

Monte-Carlo Tree Search (MCTS) [5, 10], is a search technique in which a tree is iteratively built consisting of nodes representing board states. These nodes are evaluated using the outcome of playouts, the winning ratio. The end result is a search technique that relies on long-term effect of moves rather than a value determined by a heuristic state-based evaluation function.

Because of this feature, MCTS is qualified to handle modern board games for which long-term planning is crucial, and it is difficult to find a good performing evaluation function [6, 15].

MCTS consists of four phases which are repeated in succession until a computational constraint is reached, commonly being the computation time [4]. For an outline of these phases, see Figure 2

Selection In the selection phase, starting from the root node, a child node is recursively selected according to selection a strategy S_s until a leaf node s is reached.

Playouts Starting from leaf node s, the game is simulated in self-play according to some playout strategy S_p until the cut-off strategy S_c decides to stop the simulation, traditionally until a player has won the game. A completed game is called a *playout*.

Expansion In the expansion phase, the expansion strategy S_e is applied to the selected node s.

Backpropagation In this phase the results from the simulation game are propagated backwards from leaf node p to the nodes that were traversed to reach p according to a back-propagation strategy S_b.

After the cut-off point is reached, one of the child nodes of the root is selected according to a final node selection strategy S_f.

4.1 Selection Strategy

The main aspect of MCTS is the balance between **exploration** and **exploitation**. Exploration governs the aspect of the search in which unpromising nodes must be expanded or revisited due to uncertainty in the evaluation. Exploitation, governs the aspect of search where promising nodes are visited again.

Out of several selection strategies (*OMC, PBBM, UCT, UCB1-Tuned, MOSS, HOO*) [10, 8], UCT is by far the most widely applied [14], and will be used as a base for the experiments.

UCT: In MCTS each node i has a value V_i, and visit count N_i. In UCT the child of a node p with children i is chosen that maximizes the following formula:

$$V_i + C \times \sqrt{\frac{lnN_p}{N_i}}$$

C is a coefficient that will have to be tuned experimentally. An additional parameter T proposed by 2007, Coulom [5], introduces a minimum threshold before UCT takes effect. If node p has not been visited a minimum of T times, the playout strategy is used to select the node.

4.2 Expansion Strategy

Traditionally the expansion strategy of MCTS consists of expanding node s and selecting a node randomly from among children of s. This node is then returned as node p, which is used for the playout phase.

4.3 Playout Strategy

A playout strategy is subject to two tradeoffs [4], the first being the tradeoff between search and knowledge. In normal circumstances, knowledge increases playing strength, but decreases playout speed. The second tradeoff is the ratio between exploration and exploitation. If the strategy is too explorative, the playing strength can decrease, while if it is too exploitative, the search will become too selective.

Random play is the most basic playout strategy available, and can be augmented with domain knowledge by adjusting the distribution by which moves are chosen appropriately. While the use of an adequate heuristic playout strategy has shown to improve level of play significantly [2], the aim of this article is to analyse the effects of changes to MCTS for the sake of general gameplay. Therefore, no domain knowledge was added and random play was chosen as the primary playout strategy.

4.4 Backpropagation Strategy

Out of the several backpropagation strategies (*Max, Average, Informed Average, Mix*) existing for MCTS, the general conclusion is that Average performs best [5] and as such, will be used as a base for the experiments.

Traditionally, the winrate of a node is the only metric used to evaluate the effectiveness of a node. To augment this, we added a single metric which takes into account the relative performance of player with regards to the best performing player:

$$V_i = X \times O_p + Y \times \frac{VP_p}{VP_{max}}$$

where $X = 1, Y = 1$. O_p designates the outcome of a playout for player p, 1 for a win, 0 for a draw or loss, VP_p the amount of victory points achieved by the player, and VP_{max} the maximum score achieved by any player.

4.5 Final Node Selection Strategy

Previous research has shown that little difference in quality exists in the Final Node Selection strategies, given that a sufficient number of playouts per move was played [4].

However, given the computational complexity and therefore low expected number of simulations [15], we investigate the performance of the following 4 strategies:

Max Child The child which maximizes $\frac{V_i}{N_i}$ is chosen.

Robust Child The child which maximizes N_i is chosen.

Robust Max Child The child which maximizes both $\frac{V_i}{N_i}$ and N_i is chosen.

Secure Child The child which maximizes a lower confidence bound, $\frac{V_i}{N_i} + \frac{A}{\sqrt{N_i}}$ in which is A is a parameter set to 1, is chosen. [3]

5 MCTS in Non-Deterministic Games

Traditionally, implementation of MCTS in non-deterministic games is handled in a straight forward way by integrating chance nodes as outlined in Expectimax into MCTS whenever actions governed by a stochastic process occur. [9, 17, 15] An example structure of chance nodes is shown in Figure 3.

The backpropagation function of the Chance Node is then replaced by the function:

$$V_c = \sum_{i=1}^{n} C_i V_i$$

where V_c is the value for the Chance Node, n the number of children, C_i the chance of the associated action of the child node, and V_i the value of the child node.

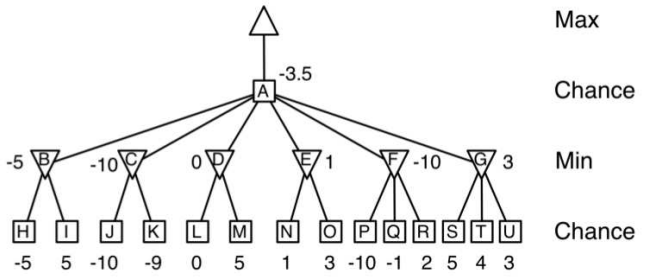

Figure 3: *Structure of Chance Node Model [9].*

5.1 Grouped Chance Model

An alternative chance model, Grouped Chance Model (GCM), is proposed in which the Chance Node and its children are combined into a single node. The resulting change in structure can be compared in Figure 4 and 5. The advantage of this model is that by combining the results of similar moves, the node converges faster to the expected average of a move. The disadvantage of this model is that by combining the moves across the different probability events, situations can occur where moves which give a non-expected high reward for a specific chance event may be masked and not explored by UCT. This phenomena can be seen in the comparison of Figure 4 and 5. In these figures, the paths chosen by UCT are represented by the continuous black edges. From the figures it is clear that action B would be chosen in the occurrence of the 0.10 event with Chance Nodes, while action B would not be considered with the Grouped Chance Model.

During backpropagation the value of the GCM node is determined by the following formula:

$$V_i = C \times V_b$$

, where C is the probability of the action that occurred when the GCM Node was applied during initial traversal. V_b is the value given by the backpropagation function. If a node is guaranteed to be traversed a sufficient number of times, this model can even be discarded as the value of a normal node trivially converges to the value of a GCM node.

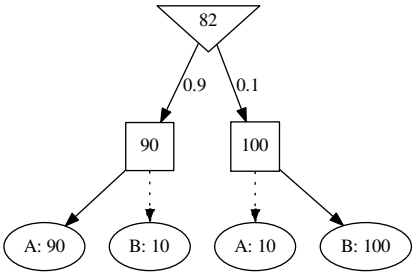

Figure 4: *Chance Nodes*

6 Move Groups

In MCTS, all moves are added as direct descendants upon expansion of a node. There exist domains, like Settler of Catan, where both the branching factor of the tree, and the time needed to compute new valid moves are high. In these domains it could be of interest to use a technique which reduces the branching factor of the tree, thereby reducing the amount of time spent in the selection and expansion phase of MCTS [11]. The grouping of nodes has shown to lead to an increase in playing strength for games such as Amazons and Go [13, 18]. A model is proposed which alters the structure of the tree by defining groups for moves. This structure is shown in Figure 6 [18].

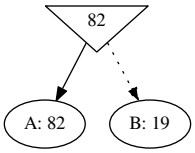

Figure 5: *Grouped Chance Model*

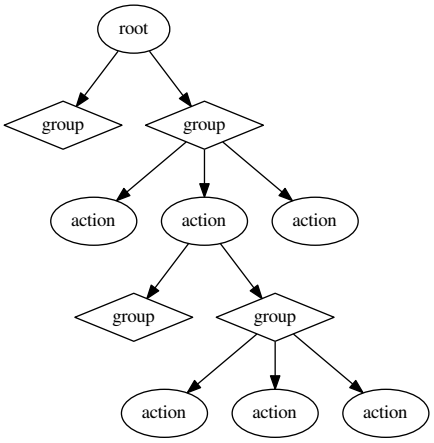

Figure 6: *Move Group Tree Structure*

The framework provides an ideal abstraction for categories; namely the action type layer provided for the algorithm. *(Buy Placable, Do Next Gamestate, ...)* Because a group node is not guaranteed to have any legal action node as children, the selection phase may never end in a group node. Upon applying a group node, it must be expanded to check the game is not in a terminal state. If no legal move exists, the algorithm must return to the parent and remove the group node from its children. Another selection may then take place from among the remaining children. Preliminary results indicate that this model significantly speeds up the selection and expansion phase of MCTS.

This model also provides a natural entry point for online learning techniques such as RAVE [7] and Progressive Bias [3]. As actions are already divided into categories, the move space required could be reduced to these groups.

7 Experiments

All experiments, where applicable, were run with no cutoff in the playouts and 2,000 ms computation time per move, using the core ruleset of Settler of Catan with the added changes as suggested by Szita, 2010 [15].

Through experimentation it was found that the average number of rounds per game varies greatly between algorithms, and is a good indicator of playing strength.

On average, a single game takes between 20 to 30 rounds, each player taking approximately 2.4 moves per turn. This results in a running time of approximately 10 minutes per game. To overcome this computational hurdle, all experiments were divided among a computing grid of 38 relatively homogeneous clients with Core2Duo E6750 (2.66Ghz Dual Core) CPU's.

The time restraint of 2,000 ms per move resulted in approximately 1,300 playouts per second. On equivalent hardware, other implementations note simulation speed of around 300 playouts per second. [15] The primary source of performance increase can be contributed to caching techniques with regards to the game logic used for placing and removing objects from the graph, which account for the gross of the computation time.

The seating arrangement of algorithms has a significant impact on their overall performance [15]. To exclude this factor from the experiments, all experiments are run such that all permutations of seating arrangements are simulated and uniformly distributed within the experiment. To further analyse the effect of the seating arrangement, their results are displayed where appropriate.

The main acronyms used in these experiments will be: Monte Carlo Tree Search (MCTS), Chance Nodes (CN), Move Groups (G-MCTS) and Grouped Chance Model (GCM), Average Win Rate (Avg. WR), Average Victory Points (Avg. VP) and Average Rounds per Game (Avg. R/G).

7.1 Game Analysis

Model Validation

In this experiment we validate the underlying framework by comparing the results of random play (800.000 games) versus the one found in 2007, Szita (400 games) [15].

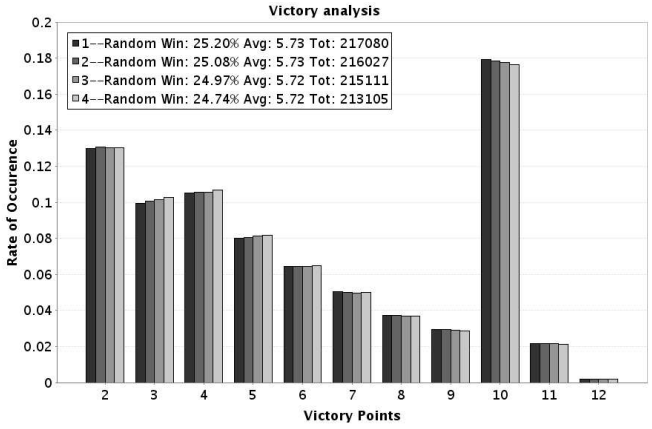

Figure 7: *Self Play, Random Play, 800.000 games.*

The overall structure of the graph, Figure 7, is approximately the same as the one found in [2007, Szita et al.] [15], with any discrepancies explained by the difference of population size from which the results were drawn.
The similarity in overall victory division and rate of victory point acquirement indicate that, for random play, the models used are approximately the same. As no mention is made on the definitive configuration [15] of the MCTS player, no comparison can be made on its results.

Seat Analysis

In this experiment the advantage of seats for games of different length using random play is analysed. Each case is run for 250.000 games, ensuring that all results found are statistically significant, giving a margin of error of 0.196% with 95% confidence. The game is cut-off after x rounds have been played, and the player having the most victory points is declared the winner.

Turn	60	40	20	10
Seat 1	25.44%	24.99%	24.57%	24.36%
Seat 2	25.00%	25.07%	24.90%	24.77%
Seat 3	24.88%	25.04%	25.19%	25.20%
Seat 4	24.67%	24.91%	25.34%	25.64%

Table 1: *The effects of preset game length on random play.*

Table 1 shows that as the length of the game is shortened, the advantage of the 4th seat becomes apparent. This effect can be explained by the fact that the 4th seat, on average, has the most resources available in his first turn as there have been 3 preceding dice rolls and the income generated by the placement of the initial settlement. This statement is confirmed by the fact that upon inspection the 4th seat tends to be the first player to place a new construction in random play.

Preliminary experiments run on cutting off the playouts after X rounds after the start of the playout showed no improvement of playing strength, although a slight increase in playout speed was gained. Playing strength declined minimally for both Move Groups and traditional MCTS, for $X = 20, 30, 40$. This could be because the metric used to declare the winner (highest victory points) is not a good indicator of the actual winner if the game were to continue.

7.2 UCT Tuning

In this experiment the C-parameter is tuned by first testing a broad range of suggested variables [10]. The local maxima found are then fine tuned by inspecting the range around them. This experiment is run for Move Groups, and traditional MCTS, both with Chance Nodes.

It has been shown that Move Groups in conjunction with UCT could benefit from tuning of the T parameter. [18] Preliminary experiments have however shown no increase for any algorithm, which could be caused by the relatively low branching factor of the domain Settlers of Catan, compared to the domains explored in the article. The only gain that could be found was when T was used as requirement for the minimum number of visits per child node, in contrast to minimum number of visits per parent node. Simply ensuring a minimum visits of 1 per child node seemed to increase performance the most, but only barely (1.47%).

In the initial experiment each test case consists of three different UCT C-parameter algorithms compared against MCTS without UCT. In this experiment all 16 permutations of the seating order are uniformly distributed in the experiment. For each case 900 games are played. In the fine-tuning of the C parameter, the tuned parameter is compared against MCTS without UCT, the win rate shown is the rate of victory of the tuned UCT agent. For each case 3,600 games are played, giving a margin of error of 1.6% with 95% confidence.

C-parameter Tuning

C	MCTS	G-MCTS
0.2	0.00%	0.00%
0.5	25.63%	19.84%
0.75	31.62%	23.65%
1.0	24.70%	22.12%
1.25	27.07%	24.32%
1.5	35.61%	26.30%
2	21.82%	23.70%
4	29.65%	27.31%
8	32.45%	29.31%
16	28.42%	25.21%
32	29.35%	26.59%
64	31.62%	27.51%

Table 2: *The effect of coarse tuning the C parameter on MCTS and G-MCTS.*

Table 2 shows the local maxima found in the initial tuning stage, namely 0.75, 1.5 and 8. Upon inspection of the low valued C-parameter, it is observed that during the setup phase of the game the algorithm chooses an extremely poor starting location, which results in extremely poor overall play. This choice can be explained by the fact that in this phase, the initial playouts base results on almost pure random play. Due to the "greediness" of the low valued C-parameter, this leads to premature selection and therefore exclusion of vital moves. Which in turn also confirms the strategic importance of the starting location. While MCTS seems to receive a larger performance gain from UCT, these values primarily result from play against other UCT algorithms; therefore we cannot directly compare the performance of MCTS against G-MCTS.

C-parameter Fine Tuning

C	MCTS	G-MCTS
0.6	20.82%	23.81%
0.7	23.88%	26.90%
0.8	21.73%	25.24%
0.9	22.15%	27.16%
1.35	21.74%	29.29%
1.45	27.31%	30.00%
1.55	25.95%	31.80%
1.65	25.23%	29.86%
1.75	28.31%	33.42%
6	25.97%	35.41%
7	28.86%	35.95%
9	28.73%	34.00%
10	28.57%	33.42%

Table 3: *Fine tuning of C for the earlier found local maxima.*

As Table 3 shows, a performance gain is shown for both Move Groups and traditional MCTS. While the net gain of performance for the Grouped MCTS algorithm is much higher than that of the MCTS algorithm, both algorithms seem to react approximately the same to the tuning of the C-parameter, with $C = 7$ being the optimum found.

Upon inspection of the play style of both the MCTS and G-MCTS algorithms, two differing strategies seem to occur. The G-MCTS algorithm seems to prefer the Grain-Ore strategy, in which emphasis is placed on the acquirement of Development Cards and the building of Cities. The MCTS algorithm however focusses on a mixed strategy, with a slight tendency towards the Stone-Wood strategy, in which the acquirement of Stone and Wood are central and the focus lies on the construction of settlements and roads.

7.3 Final Node Selection

In this experiment the effect of differing the Final Node Selection strategy on both Move Groups and traditional MCTS combined with UCT ($C = 7, T = 30$), and Chance Nodes is analysed. In each experiment the new strategy is compared to the baseline strategy, *Secure Child (SC)*. For all cases, 900 games were played, giving a margin of error of 3.26% with 95% confidence.

	MCTS		**G-MCTS**	
	MC	SC	MC	SC
Seat 1	19.00%	15.89%	23.81%	16.19%
Seat 2	16.20%	24.75%	23.33%	14.29%
Seat 3	25.24%	31.73%	29.05%	26.19%
Seat 4	30.58%	37.02%	30.48%	36.67%
Avg. VP.	6.54	6.74	7.28	7.03
Avg. WR.	22.72%	27.29%	26.67%	23.33%
Avg. R./G.	**21.79**		**24.75**	

Table 4: *Max Child (MC) vs. Secure Child (SC)*

MC is suggested to be the worst possible S_f [3], which seems confirmed by the observation that the best performing algorithm, MCTS, performs worse by using *MC*, as seen in Table 4. However, G-MCTS seems to improve its performance when combined with *MC*. This could be due to the fact that the abstraction layer provided by the move groups already provides a measure of the security *Secure Child* normally introduces. The overhead of the "security" variable introduced by *SC* could mask better options, which explains the reduced performance when used by G-MCTS.

While the difference in seating outcomes could arguably be contributed by noise, it must be noted that the percentages shown are for a 4 player setup. A difference of 5% would indicate a 10% difference in a 2 player setup.

Table 5 indicates that *RC* is detrimental to performance for MCTS, but indicates an indifference in quality for G-MCTS which could be explained by the fact that *SC* already seems to weaken the search. The grouping of the nodes also ensures that fewer children exist among which the number of visits can be divided. Of note is that a sharp drop in performance is seen for *SC* for the third seat, suggesting the strategy played is inferior for this seat against *RC*. This performance drop is not shown by G-MCTS however.

Table 6 shows that the overall performance of MCTS is relatively unaffected by *RMC*, however a clear difference is seen in per-seat play performance. However performance with RMC drops for G-MCTS, which seems to concur with the findings for RC. The earlier weak play of the third seat by MCTS reoccurs for *RMC*.

7.4 Computation Time Analysis

In this experiment we compare the performance impact of calculation time on the variations of the proposed MCTS augmentations: Move Groups and Grouped Chance Model. These experiments are done in self-play. As data points, 2,000 ms, 5,000 ms and 15,000 ms were chosen, as these represent both a slight and large increase in simulation time.

In this setup, UCT with $C = 7$ was used for all variations of the algorithm. For all cases, 600 games were played, giving a margin of error of 4% with 95% confidence.

Each table compares the relative winning percentage of each seat to visualise changes in overall strategy; average victory points and average rounds per game are shown to visualize the change in play strength. A decrease in rounds per game generally correlates with stronger play.

The expected correlation between computation time and playing strength is clearly shown for all variations of the algorithm. Of interest however is that all algorithms seem to indicate a clear advantage of seat 2 and 4 over the rest of the players. This *could* indicate that a similar strategy is used by the different algorithms, or a general weakness for seat 1 and 3 is found. The MCTS algorithm seems to benefit the most from both the small and large increase in computation time.

	MCTS		**G-MCTS**	
	RC	SC	RC	SC
Seat 1	16.32%	17.82%	21.43%	15.71%
Seat 2	17.89%	28.22%	26.19%	29.52%
Seat 3	23.36%	19.66%	22.86%	22.38%
Seat 4	35.26%	40.59%	28.57%	33.33%
Avg. VP.	6.65	6.69	6.98	7.06
Avg. WR.	23.21%	26.79%	24.76%	25.24%
Avg. R./G.	**22.42**		**23.88**	

Table 5: *Robust Child (RC) vs. Secure Child (SC)*

	MCTS		**G-MCTS**	
	RMC	SC	RMC	SC
Seat 1	20.76%	12.28%	15.71%	22.12%
Seat 2	24.77%	26.03%	12.98%	26.19%
Seat 3	24.17%	19.64%	32.69%	32.86%
Seat 4	31.74%	40.17%	28.10%	29.33%
Avg. VP.	6.61	6.63	6.99	7.17
Avg. WR.	25.32%	24.68%	22.37%	27.63%
Avg. R./G.	**22.10**		**24.33**	

Table 6: *Robust Max Child (RMC) vs. Secure Child (SC)*

Time	2 s	5 s	15 s
Seat 1	16.67%	19.07%	10.81%
Seat 2	17.24%	24.74%	33.78%
Seat 3	33.91%	30.93%	21.62%
Seat 4	32.18%	25.26%	33.78%
Avg. VP.	6.99	6.93	6.82
Avg. R./G.	23.97	21.47	20.11

Table 7: *Effect of computation time for* **MCTS**.

Time	2 s	5 s	15 s
Seat 1	21.51%	16.90%	12.51%
Seat 2	21.51%	19.37%	34.13%
Seat 3	30.23%	28.52%	21.15%
Seat 4	26.74%	35.21%	32.21%
Avg. VP.	6.92	6.95	6.62
Avg. R/.G.	24.55	23.01	20.05

Table 8: *Effect of computation time for* **MCTS,GCM**.

Time	2 s	5 s	15 s
Seat 1	20.39%	24.39%	16.67%
Seat 2	21.71%	18.29%	31.94%
Seat 3	25.66%	25.00%	16.67%
Seat 4	32.24%	32.32%	34.72%
Avg. VP.	6.91	7.22	6.81
Avg. R./G.	24.82	24.26	20.39

Table 9: *Effect of computation time for* **G-MCTS**.

Time	2 s	5 s	15 s
Seat 1	18.94%	22.60%	17.57%
Seat 2	27.27%	31.25%	39.19%
Seat 3	24.24%	25.48%	13.51%
Seat 4	29.55%	20.67%	29.73%
Avg. VP.	7.07	7.00	7.03
Avg. R./G.	25.83	22.91	21.5

Table 10: *Effect of computation time for* **G-MCTS,GCM**.

However, it seems that the strength of play of GCM becomes equivalent to CN when enough simulations are run, as both algorithms seem to need an equivalent number of rounds to finish their game.

7.5 Algorithm Analysis

In this experiment we analyse the playing strength of all proposed augmentations on MCTS, with *Secure Child* and *Max Child* as Final Node Selection strategy. For every case, 1,200 games were played to further reduce the amount of noise in the end result, giving a margin of error of 2.83% with 95% confidence.
The label on the arrow indicates the win rate of the origin algorithm over the target.

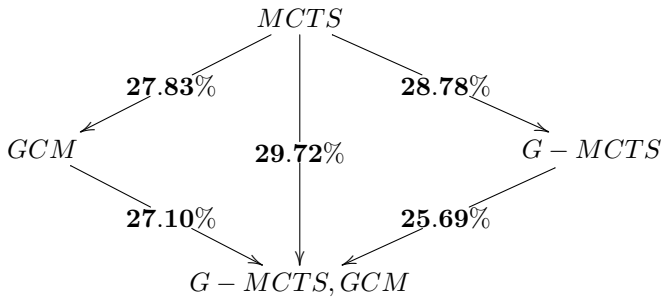

Figure 8: *Algorithm comparison with Secure Child*

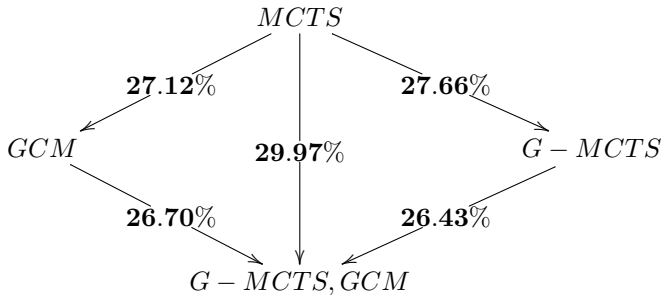

Figure 9: *Algorithm comparison with Max Child*

Figure 8 and 9 both show that the main observation that can be made is that G-MCTS as well as GCM are slightly weaker to performance for both *Secure Child* and *Max Child*. The previous observation of stronger play by G-MCTS under *Max Child* is proven yet again by the decrease of win rate of MCTS over G-MCTS in Figure 9. However this does not seem to result in a stronger play by (G-MCTS,GCM) versus MCTS, which in turn is probably caused by the inherent weakness of GCM given a lack of sufficient simulations.

7.6 Backpropagation Strategy

In this experiment the effect of the two different metrics used in the evaluation of the nodes will be analysed by setting the factor of one metric to 0. The effect is compared for traditional MCTS and Move Groups, both with Chance Nodes. The altered evaluation function is compared to the non-altered version. For each case, 600 games are played, giving a margin of error of 4% with 95% confidence.

	G-MCTS		**MCTS**	
	Disabled	Normal	Disabled	Normal
Seat 1	31.33%	18.33%	25.17%	17.67%
Seat 2	28.67%	20.33%	27.67%	20.81%
Seat 3	30.33%	17.00%	32.89%	22.33%
Seat 4	28.33%	25.67%	29.33%	24.16%
Avg. WR.	29.67%	20.33%	28.76%	21.24%
Avg. VP.	6.97	6.91	6.96	6.90

Table 11: *Disabling Victory Point Ratio*
Avg. Rounds / Game: 22.69 vs. 21.83

	G-MCTS		**MCTS**	
	Disabled	Normal	Disabled	Normal
Seat 1	19.93%	18.67%	18.12%	14.67%
Seat 2	26.33%	22.97%	24.33%	25.50%
Seat 3	38.26%	23.83%	34.56%	23.00%
Seat 4	27.85%	22.15%	27.33%	32.55%
Avg. WR.	28.10%	21.90%	26.09%	23.91%
Avg. VP.	7.33	7.00	7.25	6.92

Table 12: *Disabling Win Ratio*
Avg. Rounds / Game: 24.24 vs. 23.07

Comparing Figures 11 and 12, disabling the victory point ratio seems to give the greatest boost in performance, and benefits G-MCTS the most. However, looking at the average number of rounds required to win the game, the increase in performance is not enough to match MCTS. Peculiar though is the fact that disabling either ratio gives a boost in performance. This could be explained by the occurrence where states occur in which the algorithm is presented with two moves, one in which the chance to win is lower, but a high victory point ratio is guaranteed; and one for which the chance to win is higher, but a lower victory point ratio exists, e.g. risk must be taken to achieve the win. In these circumstance, the overall value of the first node could be higher.

8 Conclusion

In this article a implementation for the game Settlers of Catan is presented and validated, using an abstract framework capable of describing multiple non-deterministic imperfect information board games. As MCTS will serve as an AI framework for future implementations within this framework, several general enhancements were suggested and tested. These enhancements were tested using multiple variations for the Final Node Selection Strategy, Node Selection Strategy and computation time to evaluate their performance and more importantly, robustness.

The proposed Grouped Chance Model enhancement displayed slightly weaker performance for all tested MCTS variations when a sufficient amount of simulations could not be guaranteed.

The Move Groups Model as implemented in this article displayed a slight decrease in performance in all test cases compared to traditional MCTS. The main benefit of the Move Groups, a decrease in move generation time, is apparent but insignificant when compared to the computation time required by the playouts. Better results will probably be achieved in a domain where the computation time of move generation is more apparent.

The choice of groups could be a factor which contributed to the outcome, and should be investigated. Of note however is the fact that the model performed better with the Max Child selection strategy over Secure Child, while traditional MCTS performed better using Secure Child.

Tuning of the C parameter in the UCT-model revealed a local maxima at 7 for both MCTS and Move Groups. It should be noted that this tuning should probably be performed again for computation times other than 2,000

ms.

The weak playing strength of the low value C parameter, coupled with the observance of a weak play in the starting phase, could indicate that different values for the C parameter could increase performance in varying phases of the game.

While independent tuning of the T parameter in UCT had no effect, a different model could be of interest; making T dependent on the number of children in a node, instead of a fixed constant. This could enhance performance, as many complex board games have varying degrees of branching throughout the tree.

Increasing the computation time increased the playing strength of all models, and showed a supremacy over the other players by the second and fourth seat. However, different models showed varying playing strengths on the different seats before converging; this could indicate that the different seats require differing strategies for optimum play.

Experiments in random play on seating order have shown that the seating arrangement only has a small impact on the outcome of a game, caused by a advantage for the last seats on initial setup. Playing by strategy however, fully negates this effect. Further experiments showed a advantage for seat 3 and 4, when a little computation time was given, and shifted to a superiority of seat 2 and 4 as computation time increased. The author suspects this to be linked to the tournament board used in the experiments.

Adjusting the parameters of the evaluation function greatly affected performance, and the win ratio seemed to be the most sensitive metric with regards to performance.

As neither metric of the evaluation function seemed detrimental to the performance of either algorithm, and adjusting the constants which governed them positively affected performance; it is highly suggested to search for a optimum combination of both parameters.

While the Move Groups model performed slightly weaker than traditional MCTS, one important aspect was not researched; learning, be it offline or online. It is suspected that online learning models like RAVE [7] will enhance the performance of move groups [18], and could even be used to steer the playout strategy of MCTS.

As the per seat performance seemed to variate with different models and strategies, it could be of interest to explore a learning algorithm which tunes the parameters to a specified seat.

The random strategy used as the playout strategy should be constrained or guided in some way as pure random play results in poor performance and unnecessarily long playouts.

References

[1] Bjornsson, Y. and Finnsson, H. (2009). Cadiaplayer: A Simulation-Based General Game Player. *Computational Intelligence and AI in Games, IEEE Transactions on*, Vol. 1, No. 1, pp. 4–15.

[2] Bouzy, B. (2005). Associating Domain-Dependent Knowledge and Monte Carlo approaches within a Go program. *Information Sciences*, Vol. 175, No. 4, pp. 247–257.

[3] Chaslot, G.M.J-B., Winands, M.H.M., Herik, H., Uiterwijk, J., and Bouzy, B. (2008). Progressive strategies for Monte-Carlo tree search. *New Mathematics and Natural Computation*, Vol. 4, No. 3, p. 343.

[4] Chaslot, G.M.J-B. (2010). *Monte-Carlo Tree Search*. Ph.D. thesis, Department of Knowledge Engineering, Maastricht University, Maastricht, The Netherlands.

[5] Coulom, R. (2007). Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. *Computers and Games (CG 2006)*, Vol. 4630 of Lecture Notes in Computer Science (LNCS), pp. 72–83.

[6] Fossel, JD (2010). Monte-Carlo Tree Search Applied to the Game of Havannah. B.Sc. thesis, Maastricht University.

[7] Gelly, S. and Silver, D. (2011). Monte-Carlo Tree Search and Rapid Action Value Estimation in computer Go. *Artificial Intelligence*.

[8] Gelly, S., Wang, Y., Munos, R., Teytaud, O., et al. (2006). Modification of UCT with Patterns in Monte-Carlo Go.

[9] Hauk, T.G. (2004). Search in Trees with Chance Nodes. M.Sc. thesis, Edmonton University.

[10] Kocsis, L. and Szepesvári, C. (2006). Bandit Based Monte-Carlo Planning. *Machine Learning: ECML 2006*, pp. 282–293.

[11] Lorentz, R. (2008). *Amazons discover Monte-Carlo*. Springer.

[12] Love, N., Hinrichs, T., Haley, D., Schkufza, E., and Genesereth, M. (2006). General Game Playing: Game Description Language Specification. Technical report, Technical Report LG-2006-01, Stanford Logic Group.

[13] Saito, J.T., Winands, M.H.M., Uiterwijk, J.W.H.M., and Herik, HJ (2007). Grouping nodes for Monte-Carlo tree search. *BNAIC 2007: The 19th Belgian-Dutch conference on artificial intelligence, Utrecht, 5-6 November, 2007*, Vol. 19, pp. 276–283, Utrecht University.

[14] Schadd, M.P.D. (2011). *Selective Search in Games of Different Complexity*. Maastricht University.

[15] Szita, I., Chaslot, G.M.J-B., and Spronck, P. (2010). Monte-Carlo Tree Search in Settlers of Catan. *Advances in Computer Games* (eds. H.J. van den Herik and P. Spronck), Vol. 6048 of *Lecture Notes in Computer Science*, pp. 21–32. Springer Berlin / Heidelberg.

[16] Thielscher, M. (2010). A General Game Description Language for Incomplete Information Games. *Proceedings of AAAI*, pp. 994–999.

[17] Broeck, G. Van den, Driessens, K., and Ramon, J. (2009). Monte-Carlo Tree Search in Poker using Expected Reward Distributions. *Advances in Machine Learning*, pp. 367–381.

[18] Van Eyck, G. and Müller, M. (2012). Revisiting Move Groups in Monte Carlo Tree Search.

[19] Winands, HM, Bjornsson, Y., and Saito, J. (2010). Monte-Carlo Tree Search in Lines of Action. *Computational Intelligence and AI in Games, IEEE Transactions on*, , No. 99, pp. 1–1.

A Settlers of Catan Implementation

Starting with the *State Cycles*, SETTLERS OF CATAN can be divided into three phases: *Setup*, *Main Game*, *Victory*.

The *Placables* defined in the game are the *City*, *Settlement*, *Road*, *Robber* and *Tiles*. For the *City* and *Settlement*, the *Ownership Change* event adjusts the amount of victory points of the player. The *Activation* event of the Robber checks whether any player is over the resource limit, and asks them to hand in resources. The *On Place* event describes the blocking of a tile, and the stealing of a resource. The *Activation* event of the Tiles define the resource income logic.

The *Game Logic* defined correspond with the *Development Cards*; *Knight*, *Harvest*, *Monopoly*, *Construction* and *Victory Point*. Each with a default Activation Predicate which prevents them from being played in the round they were acquired.

The *Setup* can be divided into a single *Game State*, in which the only two actions allowed are *Acquire Placable*; which gives the player a road or settlement, and *Next Gamestate*; which moves to the next player if both a road and settlement are built. The *Setup* phase ends when all players have built 2 Settlements and Roads.

The *Main Game* can be divided into two states, *Pre Income*, and *Post Income*. The *Pre Income* state defines a single action, *Next Game State*, which triggers the *Deactivation* of *Pre Income*, which in turn handles the dice roll and *Activation* of the appropriate Tiles. The Post Income state defines multiple actions: *Acquire Placable*; either a City, Settlement or Road, *Buy Action*; a randomly chosen Action is given, *Play Action*; which allows the player to play an Action in his possession, *Trade Start Bank*; which allows the player to trade with the bank, and *Next Gamestate*; which ends a players turn and checks whether any player has the appropriate amount of victory points, and if so, moves the game to the *Victory* phase.