Selection and Play-out Enhancements in MCTS for Tron

C.J.R.H. Laschet

June 19, 2012

Abstract

Monte-Carlo Tree Search (MCTS) has been successfully applied in many games, such as Go and Hex. Previous work by den Teuling tested the performance of MCTS and various MCTS enhancements in the game of Tron. Tron is a two-player simultaneous-move game. Results showed that there was a lot of room for improvement. In this paper, enhancements for the selection and play-out phase of MCTS are suggested to increase the playing strength of MCTS in Tron.

Results of the experiments show that both N-Grams and LGR applied in the play-out phase, are improvements for the playing strength of MCTS in Tron. These techniques seem to increase the playing strength of MCTS equally. A combination of N-Grams and LGR does not perform well, because the overhead of these techniques becomes too large. The UCT-ULM and Progressive Bias (PB) selection strategies have little to no influence on the performance of MCTS, due to a low branching factor of the MCTS tree. The N-Gram and LGR simulation strategies can also be combined with MCTS-Solver, which provides a slight improvement in playing strength if there is no cut-off involved during the expansion and play-out phase.

1 Introduction

Board games such as Go, Chess, Hex, Checkers, and Havannah have been popular research topics in the field of Artificial Intelligence over the past fifty years. Their simple environment and rules, combined with a large state space provide an excellent test domain for intelligent search techniques. The challenge of these search techniques is to find good moves in the large state space.

In the past, $\alpha\beta$-search [10] has proven to be a successful technique for Chess and Checkers. It is a depth-first search technique that uses an evaluation function which assigns a score to a leaf node. This technique tries to improve its performance by pruning branches that cannot influence the final decision, and therefore reduces the size of the search tree. However, the evaluation function is dependent on the context of the game, and therefore a different evaluation function is required for each game. Moreover, in games that contain dynamic and fluid positions, an evaluation function can be quite complex and hard to develop.

An alternative, called Monte-Carlo Tree Search (MCTS) [7] [11], has been used for Go and Hex with promising results. MCTS relies on stochastic simulations to evaluate the effectiveness of moves. The advantage of MCTS is that it does not rely on domain-dependent knowledge to evaluate states, and therefore can be applied to a broad range of games without any structural changes aside from the game rules.

MCTS consists of four phases [3] that are repeated until the time given to compute a move has passed. These four phases are selection, expansion, play-out and backpropagation. In the selection phase, the tree is traversed, starting from the root node until a leaf node is reached. Selection of a node is dependent on the selection strategy. In the expansion phase, the children of this leaf node are added to the tree. In the play-out phase, starting from the state of one of these child nodes, a game is simulated, in which both players perform moves until the end of the game is reached, resulting in either a win, a draw or a loss. During backpropagation, the result of this simulated game is propagated back through the previously traversed nodes.

In this paper, enhancements in the selection and play-out phase of MCTS in the game of Tron are investigated, continuing on the previous work of Den Teuling [8] and Samothrakis *et al.* [16], in which MCTS was used for Tron with positive results. The focus of this paper is to enhance the MCTS performance in Tron with domain-independent heuristic additions to both the selection and play-out phase of MCTS. The methods investigated in this paper are *N-Grams* [19] and the *Last-Good-Reply Policy* [9], because both of these methods have previously shown to improve the performance of MCTS in other domains. The N-Gram technique has been successful in General Game Playing (GGP) [19] and Havannah [17]. The Last Good Reply Policy has been successfully applied in GGP [19], Go [1] and Havannah [17].

The research questions of this paper are:

1. How can the N-Gram and LGR simulation strategies improve the playing strength of MCTS in the game of Tron?

2. How can the N-Gram technique improve the selection strategy of MCTS in the game of Tron?

The outline of this paper is as follows. Section 2 provides a brief introduction to the game of Tron and MCTS. Section 3 contains the proposed play-out enhancements, followed by the selection enhancements discussed in Section 4. In Section 5, experiments and results are given. Finally, in Section 6 conclusions are drawn from the results and future research is suggested.

2 Background

In this section, the rules and important properties of the game of Tron will be explained. Furthermore, a brief explanation of Monte-Carlo Tree Search [7] is given.

2.1 The Game of Tron

The game of Tron was first seen in the movie Tron, made by Walt Disney Studios in 1982. In this game, two players move in discrete time steps on a $m \times n$ grid of cells. Figure 1 depicts an example of a typical Tron game. The outer edges of the grid are enclosed by a wall, which is inaccessible for the players. At each time step, each player has the choice to go either up, down, left or right. However, both players leave a solid wall behind them as they are moving forward, filling up the previous cell they were located in, and reducing the size of the available area to move to as the game progresses. Should one of the players choose a direction that causes them to crash into a wall, or the edge, then this player loses and the other player wins. If both players crash at the same time, a draw occurs. If both players choose moves such that they both end up in the same position on the board, a draw occurs as well.

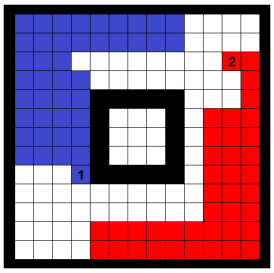

Figure 1: An example of a typical Tron game

Another important property of Tron is that moves are chosen simultaneously. At each time step, both players declare their chosen move at the same time. This increases the difficulty of the game, as the effectiveness of a move depends on the move made by the other player.

The general strategy of playing a Tron game, is to outlast your opponent such that his options run out and crashes into a wall. To increase the speed of this process, it is a good approach to try to limit the available space of your opponent, by separating the board into multiple parts, without endangering yourself into a vulnerable position on the board. Because boards can contain obstacles, these can be used as well to block the path of your opponent. However, these obstacles also make it more difficult to fill up the available space surrounding the player. At the start of the game, it is difficult to decide upon which move to make, since the state space is quite large and predicting the opponent is troublesome.

2.2 Monte-Carlo Tree Search

Monte-Carlo Tree Search (MCTS) [7] is a best-first search strategy that builds up a tree in memory by iterating through four phases, until it runs out of time to calculate the next move [5]. Each node in the tree holds a state of the game, which in this case is the configuration of the board. Edges between nodes represent the move which, if applied to the board configuration in the first node, results in the board configuration of the second node. Furthermore, each node i holds a value v_i and a visit count n_i. Nodes that have a higher value are preferred over nodes with a lower value. These values are estimates based on the average return of simulations, also called play-outs.

MCTS starts from the root node, which is the current state of the game. First, the tree is explored at random, but as more simulations have been performed, a shift occurs from random exploration to exploitation of the most promising nodes.

As was mentioned before, MCTS is divided into four phases: selection, expansion, play-out and backpropagation. The phases are explained in detail below. MCTS iterates through these four phases until the time to compute the next move is up, as is depicted in Figure 2 [5].

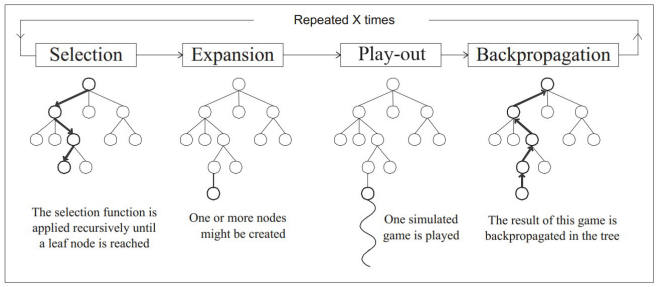

Figure 2: The four phases of MCTS [5]

Selection

Starting from the root node of the tree, a child node of the current node is selected until a leaf node is reached. These child nodes are selected by a selection strategy, which can be as simple as selecting a random child, but can also be more sophisticated strategies such as UCT. The selection phase is important, since it determines which node will be evaluated in the play-out phase.

Promising moves should be played more often than unpromising moves, which is called exploitation. However, to know which moves are promising, unexplored moves should be taken into consideration as well, for the sake of discovering new promising moves. This is called exploration. Therefore, a balance has to be found between exploring new moves, for which the value is unknown, and exploitation of moves which are known to be effective moves in the current state. This is the task of the selection strategy.

One such selection strategy, called UCT (Upper Confidence bound applied to Trees) [11], takes this balance into account. UCT selects from node p its child k as follows:

$$\arg\max_i \left(v_i + C \times \sqrt{\frac{\ln n_p}{n_i}} \right)$$

UCT selects the child of the current node which maximizes this value. The first term supports the exploitation of known successful moves, whilst the second term supports the exploration of new moves. The C constant can be tuned to change this balance between exploitation and exploration. As C gets higher, more exploration takes place, and therefore less exploitation. Furthermore, if a node has been visited fewer than T times, the random selection strategy is applied, because in that case the values of the nodes being evaluated are not accurate enough.

Expansion
During the expansion phase, all children of the leaf node k are added to the tree. Since the number of children of each node is at most 3 in the case of Tron, these children are all added at once. Then, the selection strategy is once more applied to the leaf node k, selecting one of the new children i as the starting point for the next phase.

Play-out
During the play-out phase, starting from the node retrieved in the expansion phase, the game is simulated in self-play, until the game has ended or when the resulting end can be reliably predicted. During this play-out phase for Tron, both players move simultaneously [8]. The moves chosen during the play-out by both players can be chosen randomly, but moves can also be chosen quasi-randomly by using a knowledgeable simulation strategy. It has been shown that the use of such a heuristic simulation strategy can improve the play-out significantly [2], [6]. After a simulation has been finished, it returns 1, 0 or -1 to the play-out starting node i, according to a win, draw or loss, respectively.

Backpropagation
The result of the play-out that was returned to node i is backpropagated along the path of nodes traversed in the selection phase, until the root node is reached, updating v_i of each node with the result of the play-out along that path, by calculating the new average value.

When the time to search has passed, MCTS stops the iteration and selects the child of the root node determined by the final selection strategy. In this paper, the most secure child is chosen [5], which is defined as:

$$v_i + \frac{A}{\sqrt{n_i}}$$

where A is a constant. Based on results of trial-and-error testing, a value of 1 was used [8].

3 Play-out Enhancements

In this section, two play-out enhancements are discussed which are used as simulation strategies for the game of Tron. A good simulation strategy can improve play-outs significantly [19]. A simulation strategy based on N-Grams is explained in the first subsection. In the following subsection, a simulation strategy based on the Last-Good-Reply policy is explained.

3.1 N-Grams

N-Grams originate from the field of statistical natural language processing [13] and are also used for predicting the next moves of the players in video games [14]. N-Grams contain large sets of move sequences. Each move sequence S holds an average reward, denoted by *R(S)*. The higher this reward is, the better the sequence of moves is. N-Grams offer a simple way of adding context to board configurations, enabling the player to detect critical positions on the board. A correct usage of N-Grams can also predict player behaviour by using context [12]. The context is provided by the previously made moves of the players. Although N-Grams can provide a fast and simple way of predicting player behaviour, the execution of long-term plans is difficult, since N-Grams only take local relationships into account.

The length of the move sequences held by a N-Gram can vary. N denotes the length of the move sequences a N-Gram contains. For example, a 3-Gram holds move sequences of length 3. The moves in such a sequence are alternating moves from the current player and the opponent. A sequence of length 1 would consist of just one move of the current player. A sequence of length 2 would contain a move of the opponent, followed by a move of the current player. Finally, a move sequence of length 3 consists of first a move of the current player, followed by a move of the opponent, and finally another move of the current player. In all these cases, the current player is the player for which the N-Grams are consulted. In the case of Tron, only 2-Grams and 3-Grams are used, because the 1-Gram provides too little context to be of any use. N-Grams containing longer move sequences are

not considered, since the memory requirements would be too demanding.

In the case of a turn-taking game, a move sequence consists of moves that were performed consecutively in alternating fashion between the players. However, Tron is a simultaneous-move game. A player cannot know which move the opponent will perform at the current time step, when choosing a move for itself. Therefore, move sequences should ignore moves of the opponent that were made simultaneously. Therefore, the move made by the other player at the same time step is ignored when move sequences are added to the N-Grams. Instead, the moves made by both players in previous time steps are used.

N-Grams can be used for a simulation strategy in the play-out phase of MCTS as follows [19]. After each play-out, all move sequences of length 2 and 3 are extracted from the moves performed in the simulated game. If a move sequence is extracted that does not yet exist in the corresponding N-Gram, then this move sequence is added to the N-Gram with an initial *R(S)* value equal to the reward of the play-out for the current player. If an extracted move sequence already exists in the corresponding N-Gram, then the average reward *R(S)* is updated by adding the reward of the play-out for the corresponding player. The process of extracting move sequences is identical to the method described in [19].

Figure 3 shows an example of extracting move sequences from a play-out. For example, the move sequence F-G-J consists of move F made by player 2, followed by move G of player 1, and finally move J of player 2. Move I is not taken into consideration, because this move was performed at the same time move J was performed. Therefore, move I cannot be a reaction of player 1 to move J of player 2. Similarly, move J cannot be a reaction to move I. Move H is not taken into consideration because the presence of moves F and J in the move sequence imply that move H was performed as well between those two moves. Because a player in the game of Tron cannot skip positions, move H becomes obsolete when moves F and J are included in the move sequence. This method of move sequence extraction allows the N-Grams to hold move sequences of length 3 which describe a longer move sequence than just three moves.

During the play-out phase, the N-Grams can be used to select which move to perform, depending on the board configuration. Given a board configuration, the player determines which sequences S of length 2 and 3 would occur, considering the previously made moves and the current legal moves. Since only move sequences of length two and three are considered, at most two sequences can occur per legal move. A score for each legal move k is then calculated, by taking the average of the *R(S)* values of the move sequences that contain k as the final

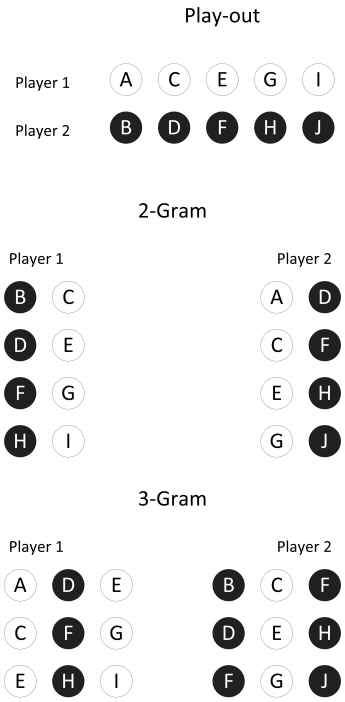

Figure 3: An example of the move sequences generated from a play-out

move. If none of the move sequences are contained in the N-Grams, then the default policy is used. The default policy selects a random move from the legal moves.

The move with the highest score is selected to be played in the play-out, according to an ϵ-greedy strategy [18], playing a random move with probability ϵ, and the N-Gram based move with probability $1 - \epsilon$. In such a way, the play-out is prevented from becoming too deterministic.

3.2 Last-Good-Reply Policy

The Last-Good-Reply Policy (LGR) [1], [9], [17] is somewhat similar to the idea of N-Grams. LGR keeps track of good replies to the preceding moves of both players. Instead of keeping track of the effectiveness of multiple moves, LGR keeps track of only the best reply to the previous moves. If such a move is not legal, then the default simulation strategy is used, which in this case performs a random legal move.

LGR has the advantage of having a smaller memory footprint than N-Grams, since only one move per move sequence is stored. Furthermore, LGR does not have to calculate and keep track of average reward values as the N-Gram technique does.

LGR holds for each player two separate tables, called LGR-1 and LGR-2. Similar to the move sequence extraction process of N-Grams, these tables are updated after each play-out. In the LGR-1 table, the best reply to the previous move of the opponent is stored. Because Tron

is a simultaneous-move game, the move of the opponent in the previous time step is used, instead of the opponent move performed during the current time step. In the LGR-2 table the last good reply to the two previous moves is stored. These two previous moves consist of a move made by the player himself, followed by an opponent move. Similar to the N-Gram technique, the previous moves are the moves made in the previous time step, due to Tron being a simultaneous-move game.

If the reward obtained by the player is at least as high as the reward of the opponent, it is stored in the corresponding LGR table. If there already is a reply known for the given previous moves, then this reply is overwritten with the new move if it received a higher reward.

Move sequences can be deleted from the LGR tables as well. If the reward obtained by the player after a play-out is lower than the reward of the opponent, then this move is deleted from the tables as a good reply to the previous moves. This technique is called forgetting. A LGR policy with forgetting is commonly abbreviated as LGRF [1]. By using this technique, the play-out is prevented from becoming too deterministic. Tests showed that LGRF outperformed LGR in the game of Go [1]. This process of adding and deleting move sequences is explained in more detail in Figure 4.

Similar to the N-Gram enhancement, the LGR-1 and LGR-2 tables can be used during the play-out phase to determine which move to perform, given the previous moves. First, the LGR-2 table is checked for which move to choose, since this provides the most context. If the LGR-2 does not hold a best reply for the previous moves, then the LGR-1 table is consulted. When the LGR-1 table has no reply as well, the default simulation strategy is used. This strategy chooses a random move from the legal moves.

3.3 N-Gram LGR Combination

These two strategies can also be combined as one simulation strategy [17]. In this case, the LGR tables are consulted first to decide which move to perform. If the tables do not hold a best reply, then the N-Grams are consulted for a move. The LGR tables therefore have a higher preference for choosing the next move than the N-Grams. The process of choosing a move with either the LGR tables or the N-Grams is identical to the technique described in the previous subsections. If the N-Grams hold no suitable reply as well, then the default simulation strategy is used. In this case, a random move is chosen from the legal moves.

4 Selection Enhancements

Similar to the play-out phase, the selection phase can be enhanced with more sophisticated selection strategies as well. Section 2.2 already mentioned random selection and UCT selection. In this section, two additional selection strategies are discussed: UCT-ULM and Progressive Bias.

4.1 UCT-ULM

Because MCTS has a limited search time, it is preferred to spend this time searching in parts of the tree which contain good move sequences, than searching in parts of the tree with a neutral or bad move sequence history. The UCT-ULM selection strategy takes this preference into account. UCT-ULM applies regular UCT selection to a node in the MCTS tree when all the children of that node have been visited before. If this is not the case and the current node has unvisited legal moves (ULM), then from the children which are ULM, the child with the highest heuristic value is chosen. The heuristic value of a child node is determined by the 2-Gram and 3-Gram value for the corresponding move sequence, similar to the way moves are chosen in the play-out phase when using the N-Gram play-out enhancement.

In the case of a node with ULM, UCT-ULM directs the tree search to the subtree which is most likely the best option, according to the average of the move sequence evaluations held by the 2-Gram and 3-Gram. Therefore, good move sequences are preferred over worse move sequences. Unvisited legal moves that have a lower heuristic value therefore have a lower priority, because they might never be selected by UCT-ULM.

However, UCT-ULM has a notable disadvantage. The preference of certain parts of the MCTS tree over other parts of the tree depends solely on the judgment of the N-Grams. There is no smooth transition from a good move to a bad move. Only one of the three children of a node can have the highest heuristic value. Moreover, no additional parameters such as the number of visits and win count are taken into consideration. Therefore, UCT-ULM may become too deterministic.

4.2 Progressive Bias

The Progressive Bias (PB) selection technique [5] does not have the disadvantage of UCT-ULM. Instead of using UCT and the heuristic value technique separately, PB combines UCT selection with heuristic knowledge. The heuristic knowledge is provided by the 2-Gram and 3-Gram in the same way as with the UCT-ULM selection strategy.

By using a mixture of UCT selection and heuristic knowledge, node selection can be guided in a better fashion than with plain UCT selection. However, because MCTS has a limited amount of time to determine the next move, there is a trade-off to be made, between having more play-outs and spending more time on computing heuristic values. This results in having

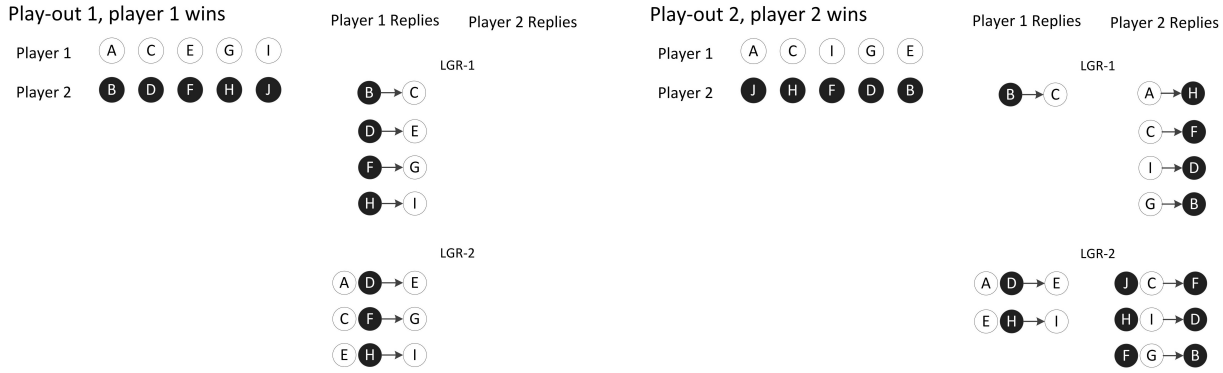

Figure 4: An example of the updating of the LGR tables during 2 consecutive play-outs

more play-outs with possibly less relevance, and having fewer play-outs with more relevance. The formula of the PB selection strategy is shown below.

$$\arg\max_i \left(v_i + C \times \sqrt{\frac{\ln n_p}{n_i}} + \frac{W \times H_i}{l_i + 1} \right)$$

In this formula, the left term is identical to the original UCT selection strategy. The right term represents the progressive bias enhancement. W is a constant set to 10 [8]. H_i is the N-Gram based heuristic value for child i, based on the move sequence consisting of the child node i, its parent p and the parent of node p. The heuristic value of node i is divided by the number of losses of that node, l_i. Dividing the heuristic value by the number of losses assures that nodes that do not turn out well are not biased for too long [15]. When a node has been visited only a few times, or not at all, the heuristic knowledge has a major influence on the final decision of which node to select. Therefore, this heuristic knowledge stays a major influence as long as the node has proven to be a good move, and becomes less significant when the move does not perform too well.

The advantage of PB over UCT-ULM is that it has the best of both worlds. UCT handles the selection of moves based on win and loss parameters when these nodes have been visited sufficiently, and the heuristic evaluation handles the selection of nodes with heuristic knowledge if the UCT parameters have not settled yet. Moreover, PB provides a smooth transition between these two techniques.

5 Experiments & Results

In this section, experiments are conducted to test the performance of the previously discussed simulation and selection strategies. In the first subsection the ϵ parameter used by the N-Gram simulation strategy is tuned. Next, the previously discussed updating process of the N-Gram tables is compared with a sequential update method. In the third subsection, the the N-Gram, LGR and N-Gram LGR Combo strategies are tested. In the fourth subsection, the UCT, UCT-ULM and PB selection strategies are tested. The last subsection deals with the performance of the N-Gram simulation strategy when combined with the MCTS-Solver player, as described in [20].

The experiments are conducted on a 2.4 GHz AMD Opteron CPU with 4GB of RAM. Unless noted otherwise, experiments were conducted on 10 different maps, shown in Figure 5. Although all the boards are symmetric, each player played each color exactly half of the number of matches played for that map.

Unless stated otherwise, the following settings were used for each experiment. Each experiment consisted of 200 games played per map. Every player is given 1 second to decide upon its next move. This process is approached as if the moves are made simultaneously, and therefore a player has no knowledge of which move the opponent will perform. All experiments were executed in the framework created by den Teuling [8]. The simulation and selection enhancements were added to the MCTS player of this framework, resulting in the enhanced player variations used for the experiments. This original MCTS player of den Teuling uses a C constant with value 10 and a T constant with value 30 for the UCT selection strategy. The MCTS player uses a random simulation strategy. The final selection strategy for all the players is the previously discussed secure child technique.

5.1 Tuning the N-Gram ϵ parameter

Before any other experiments are conducted, the optimal value for the ϵ parameter used by the N-Gram simulation strategy has to be determined. This value was determined by an experiment in which a MCTS player with UCT selection and the N-Gram simulation strategy plays against a MCTS player with UCT selection and a random simulation strategy. The experiment was conducted on 5 different maps, resulting in a total of 1000

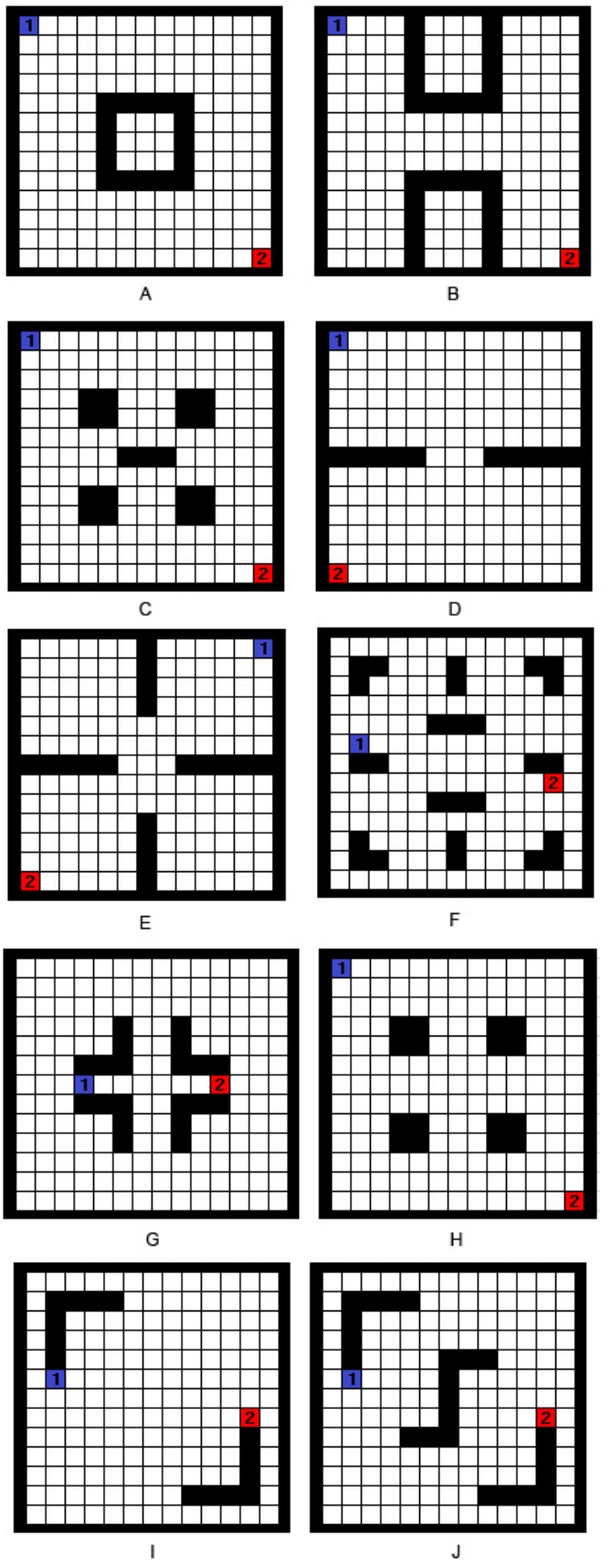

Figure 5: The maps used for the experiments

ϵ/map	A	D	F	G	I	Avg
0.0	58	59	52	48	80	59% ± 2.2%
0.1	64	61	60	41	75	60% ± 2.1%
0.2	73	57	55	46	73	60% ± 2.1%
0.3	70	60	53	64	74	64% ± 2.1%
0.4	73	55	51	70	65	63% ± 2.1%
0.5	72	56	51	81	57	63% ± 2.1%
0.6	80	54	47	82	58	64% ± 2.1%
0.7	78	50	48	75	59	62% ± 2.1%
0.8	76	44	46	73	53	58% ± 2.2%
0.9	77	43	39	64	43	53% ± 2.2%
1.0	56	51	49	51	52	52% ± 2.2%

Table 1: Win rates of MCTS with UCT & N-Gram play-out with different ϵ values vs. MCTS with UCT & random play-out

games. The win rates for the MCTS N-Gram player per map and ϵ value are shown in Table 1. The last column shows the average win rate per epsilon value over all maps.

Each map shows an initial rise and latter decline of the win rates depending on the ϵ value, indicating that there is an optimal value. However, the results also show that the optimal ϵ of the MCTS N-Gram player varies per map. Therefore, it seems that optimal ϵ is map dependent. The last column shows the average win rate over all maps for each ϵ. This column clearly shows this rise and decline in win rates as well, with an ϵ value of 0.3 as the most probable optimal value. Therefore, for each following experiment that used the N-Gram strategy, an ϵ value of 0.3 was used.

However, a side remark should be made here. At ϵ value 1.0, the simulation strategy does not use the N-Grams during the play-out phase. The simulation strategy is in that case a random simulation strategy. Therefore, the N-Grams were no longer updated after a simulation when the ϵ value was set to 1.0. This explains the sudden rise and decline in win rates on some maps at this value, because there no longer is a N-Gram updating overhead.

5.2 Simultaneous Moves vs. Sequential Moves

Tron is a simultaneous-move game and therefore the moves made by the players are chosen simultaneously. The N-Gram updating process should take this property into account. In this subsection, two techniques of extracting move sequences are tested. The simultaneous-moves-based updating process was discussed in Subsection 3.1 and Figure 3. This simultaneous updating process has the possible advantage that move sequences contain more information than just the three moves it describes. The second technique updates the N-Grams

as if the moves made during the play-out phase were performed sequentially. In other words, the simultaneous moves property of Tron is not taken into account by this sequential-move-based updating process. For example, from the play-out shown in Figure 3, move sequences G-H-I and H-I-J would be extracted by this sequential method. The sequential-moves-based updating process does not have the possible advantage of the simultaneous-moves-based updating process. Instead, it extracts more compact move sequences which describe a more recent move history.

Two experiments were conducted that show the difference in performance between these two methods of updating the N-Grams. In the first experiment, a MCTS player with UCT selection and the N-Gram simulation strategy that uses the *sequential* update method played against a MCTS player with UCT selection and a random simulation strategy. The second experiment consisted of a MCTS player with UCT selection and the N-Gram simulation strategy that uses the *simultaneous* update method playing against a MCTS player with UCT selection and a random simulation strategy. Both of the experiments were conducted on the maps A, D, F, G and I. On each board 200 games were played, resulting in a total of 1000 games. The results of the first experiment (sequential) are shown in Table 2. The results of the second experiment (simultaneous) are given in Table 3.

Map	A	D	F	G	I	Avg
%	50	56	48	49	68	54.0% ± 3.1%

Table 2: Win rates of MCTS with UCT & N-Gram play-out (sequential based) vs. MCTS with UCT & Random play-out

Map	A	D	F	G	I	Avg
%	70	60	49	64	74	63.4% ± 3.0%

Table 3: Win rates of MCTS with UCT & N-Gram play-out (simultaneous based) vs. MCTS with UCT & Random play-out

The results clearly indicate that the simultaneous method achieves higher win rates than the sequential method. The sequential method has an average win rate of 54.0%, while the simultaneous method has an average win rate of 63.4%. Moreover, the simultaneous method performs better than the sequential method on almost every map. Therefore, the simultaneous method was used for all the following experiments that used the N-Gram simulation strategy.

5.3 Play-out Enhancement Experiments

In this subsection, the N-Gram, LGR and the N-Gram LGR Combo simulation strategies are tested.

MCTS-UCT-N-Gram vs. MCTS-UCT-Random
To proper assess the increase in performance caused by the N-Gram simulation strategy, an experiment was conducted in which a MCTS player with UCT selection and the N-Gram simulation strategy played against a MCTS player with UCT selection and a random simulation strategy. The results are given in Table 4.

Map	A	B	C	D	E	F	G	H	I	J	Avg
%	70	61	56	60	39	49	64	77	74	53	60.3% ± 2.1%

Table 4: Win rates of MCTS with UCT & N-Gram play-out vs. MCTS with UCT & Random play-out

The results show a significant increase in performance of the MCTS player that used the N-Gram simulation strategy. An average win rate of 60.3% is achieved by using the N-Gram simulation strategy. Although most maps show close to a 60% or even higher win rate, maps E, F and J have a lower win rate. As was mentioned before, the performance of the N-Gram simulation strategy can vary per map because the optimal value of the ϵ parameter can differ greatly for each map.

MCTS-UCT-LGR vs. MCTS-UCT-Random
The next experiment was conducted to show the increase in performance caused by the LGR simulation strategy. It consisted of a MCTS player using UCT selection and the LGR simulation strategy playing against a MCTS player with UCT selection and a random simulation strategy. The results are shown in Table 5.

Map	A	B	C	D	E	F	G	H	I	J	Avg
%	84	78	66	60	41	47	50	69	73	59	62.6% ± 2.1%

Table 5: Win rates of MCTS with UCT & LGR play-out vs. MCTS with UCT & Random play-out

The results show that MCTS with LGR has an average win rate of 62.6%. On most maps the LGR player has a win rate far higher than this average. However, on maps E and F the LGR player has a below average win rate. It is interesting to see that these are the same maps on which the N-Gram simulation strategy also had trouble to outplay its opponent. This could possibly indicate that both the LGR and N-Gram simulation strategies have difficulties providing good move sequences due to the structure of these maps. The MCTS with LGR achieves slightly higher win rates on some maps than the MCTS with the N-Gram strategy. This is most likely caused by the fact that the N-Gram strategy has a lower number of simulations per second (about 55,000) when compared to the LGR strategy (about 75,000) and therefore has gathered less information on which it has to base its final decision.

MCTS-UCT-N-Gram vs. MCTS-UCT-LGR
The next experiment shows the playing strength of both the N-Gram and LGR simulation strategy when these strategies play against each other. In this experiment a MCTS player with UCT selection and N-Gram simulations played against a MCTS player with UCT selection and LGR simulations. The results are shown in Table 6.

Map	A	B	C	D	E	F	G	H	I	J	Avg
%	59	50	47	52	53	47	51	45	45	53	50.2% ± 2.2%

Table 6: Win rates of MCTS with UCT & LGR play-out vs. MCTS with UCT & N-Gram play-out

The results show that the N-Gram and LGR simulation strategies are evenly matched. On almost every map the win rate of the two players are almost equal, indicating that neither of them can convincingly outplay the other player. Averages show how close the two players actually are, indicating a 50.2% win rate for the LGR player and therefore a 49.8% win rate for the N-Gram player.

It is interesting that the N-Gram player, which achieves 55,000 simulations per second, can perform as well as the LGR player which has 75,000 simulations per second. This suggests that the N-Gram player is better in retrieving more useful information with fewer play-outs than the LGR player. To see if this was actually the case, another experiment was conducted with the same players, but twice the time (2 seconds) to decide upon which move to perform. The results are shown in Table 7.

Map	A	B	C	D	E	F	G	H	I	J	Avg
%	61	53	48	52	49	43	83	52	39	86	56.3% ± 2.2%

Table 7: Win rates of MCTS with UCT & LGR play-out vs. MCTS with UCT & N-Gram play-out - 2 sec

The results indicate that it is not the case that N-Gram outperforms LGR. The LGR player starts to dominate heavily on maps A (61%), G (83%) and J (86%). However, the N-Gram technique achieves a 57% win rate on map F and a 61% win rate on map I. On the other maps the equal win rate is retained. The LGR strategy achieves an average win rate of 56.3%, therefore resulting in a 43.7% win rate for the N-Gram strategy.

MCTS N-Gram LGR Combo
The next experiment tested the performance of the previously discussed LGR N-Gram Combo simulation technique, by letting a MCTS player with this simulation strategy and UCT selection play against a MCTS player with UCT selection and various simulation strategies. The results are shown in Table 8.

Map	A	B	C	D	E	F	G	H	I	J	Avg
Random	57	45	64	47	40	45	41	70	74	57	53.8% ± 2.2%
N-Gram	49	50	48	58	47	46	47	51	55	30	47.9% ± 2.2%
LGR	33	50	46	43	45	48	42	57	60	32	45.5% ± 2.2%

Table 8: Win rates of MCTS with UCT & LGR N-Gram Combo play-out vs. MCTS with UCT & various play-outs

The results clearly indicate that the LGR N-Gram Combo does not improve the performance of the MCTS player at all, but actually underperforms when compared to the LGR and N-Gram simulation techniques. This is caused by the large overhead of using both LGR tables and N-Gram tables. Because all 5 of these tables need to be updated after each simulation, the number of simulation per second drop significantly (40,000), resulting in inferior playing performance. Since the performance of the LGR N-Gram Combo player was disappointing, an additional test was conducted with exactly the same players but with double the time (2 seconds) to decide upon which next move to perform. This did not reveal an increase of performance for the LGR N-Gram Combo player. Results did not differ too much from those shown in Table 8.

5.4 Selection Enhancement Experiments

In this subsection the previously discussed UCT-ULM and PB selection strategies are tested.

UCT-ULM
In this experiment a MCTS player with UCT-ULM selection and the N-Gram simulation strategy played against a MCTS player with UCT selection and the N-Gram simulation strategy. In this way, the performance of UCT-ULM can be tested. The results are given in Table 9.

Map	A	B	C	D	E	F	G	H	I	J	Avg
%	55	50	49	53	50	49	50	52	54	47	50.9% ± 2.2%

Table 9: Win rates of MCTS with UCT-ULM selection & N-Gram play-out vs. MCTS with UCT selection & N-Gram play-out

Results show that the UCT-ULM selection strategy does not have a significant influence on the performance of the MCTS-N-Gram player. The most likely cause is that the nodes in the MCTS tree have a low branching factor, in the case of Tron. Each node has at most three children. Because only the ULM are considered by UCT-ULM, all children of a node will most likely

be visited soon. Once most of the nodes have been visited, UCT replaces the ULM selection strategy and there is no difference in selection strategies left between the two players, resulting in the 50.9% win rate.

Progressive Bias
Another experiment was conducted to show the improvement in performance by using the previously discussed Progressive Bias selection strategy. In this experiment, a MCTS player with the PB selection strategy and the N-Gram simulation strategy played against a MCTS player with UCT selection and the N-Gram simulation strategy. The results are shown in Table 10.

Map	A	B	C	D	E	F	G	H	I	J	Avg
%	46	50	47	41	47	52	50	45	48	51	47.7% ± 2.2%

Table 10: Win rates of MCTS with PB selection & N-Gram play-out vs. MCTS with UCT selection & N-Gram play-out

Similar to the results of the UCT-ULM selection strategy, the PB selection strategy does not improve the performance of the MCTS player. The reasons stated for UCT-ULM hold here as well. Due to the low branching factor of the MCTS tree in Tron, selection strategies do not seem to be a critical part of MCTS. Therefore, the selection strategies have not shown to be a performance increasing enhancement for the MCTS player in the case of Tron.

5.5 MCTS-Solver Experiments

In this subsection the N-Gram simulation strategy was combined with the MCTS-Solver player provided by the framework of den Teuling. The workings of this MCTS-Solver player are not explained here. A complete explanation of this MCTS enhancement can be found in the paper of Winands *et al.* [20]. To adapt this strategy in such a way that it can handle draws, a modification is necessary as described in [4, 8]. MCTS-Solver is an enhancement over MCTS that is able to prove the game-theoretic value of a position. Therefore, proven good positions do not have to be evaluated again and more search time can be spent on unexplored positions. This addition requires almost no additional computation time and has shown that it can outperform a standard MCTS-UCT player in the case of Tron [8].

MCTS-Solver N-Gram vs. MCTS-Solver
The first experiment was conducted to show the improvement in playing strength of combining the N-Gram simulation technique with the MCTS-Solver player. A MCTS-Solver player with the N-Gram simulation strategy played against a MCTS-Solver player without the N-Gram enhancement. The win rates for the N-Gram enhanced MCTS-Solver are shown in Table 11.

Map	A	B	C	D	E	F	G	H	I	J	Avg
%	50	48	40	54	50	54	34	47	52	40	46.8% ± 2.2%

Table 11: Win rates of MCTS-Solver & N-Gram play-out vs. MCTS-Solver

Results indicate that the N-Gram enhanced MCTS-Solver did not outperform the normal MCTS-Solver. This could be caused by the simulation and expansion cut-off applied in the MCTS-Solver. The expansion or play-out phase of MCTS-Solver is aborted, when the player already knows, before completing the simulation or expansion phase, that the game will inevitably end in either a win or a loss. In this case the value of a node can be set to either the best possible value (in the case of winning), or the worst possible value (in the case of losing). In that way, only moves that are not certain to lead to either an immediate win or loss are considered during the search. This cut-off causes the N-Grams either to be not filled with enough move sequences, or that the move sequences held by the N-Grams are not updated regularly, resulting in out-of-date statistics of those sequences. The same experiment was conducted for the LGR enhanced MCTS-Solver, but results did not differ too much from those of the N-Gram enhanced MCTS-Solver.

MCTS-Solver N-Gram with no cut-off vs. MCTS-Solver
To actually see whether the previously discussed problem was the cause of the weak performance of the N-Gram enhanced MCTS-Solver player, two additional experiments were conducted. The first experiment consisted of a normal MCTS-Solver with no cut-off which played against another MCTS-Solver with a cut-off. The results are shown in Table 12, indicating the win rates of the MCTS-Solver without a cut-off. In the second experiment, a N-Gram enhanced MCTS-Solver player which did not perform the cut-off in the simulation and expansion phases played against a normal MCTS-Solver player. This could show an improvement in performance of the N-Gram enhanced MCTS-Solver player, since there is now more information available to use for updating the N-Grams. The results, indicating the win rates of the N-Gram enhanced player are shown in Table 13.

The results of Table 13 show that the N-Gram enhanced MCTS-Solver player does perform better than the original MCTS-Solver player. An average win rate of 55.9% is achieved by the N-Gram enhanced player. Win rates vary per map, but the N-Gram MCTS-Solver

Map	A	B	C	D	E	F	G	H	I	J	Avg
%	50	48	53	51	53	31	51	37	42	14	42.8 ± 2.2%

Table 12: Win rates of MCTS-Solver with no cut-off vs. MCTS-Solver with cut-off

Map	A	B	C	D	E	F	G	H	I	J	Avg
%	55	49	53	57	45	73	46	59	58	65	55.9% ± 2.2%

Table 13: Win rates of MCTS-Solver & N-Gram play-out with no cut-off vs. MCTS-Solver

does not underperform against the original MCTS-Solver on any map.

Furthermore, the results of Table 12 indicate that a MCTS-Solver without a cut-off performs worse than a MCTS-Solver with a cut-off. An average win rate of 42.8% is achieved by the no cut-off MCTS-Solver. There is clearly an increase in performance when the N-Gram enhancement is added to this no cut-off player. This increase in performance is therefore solely caused by the N-Gram enhancement, because the MCTS-Solver without the N-Gram enhancement and without a cut-off performs worse than the MCTS-Solver with a cut-off. This indicates that the cut-off was actually the cause of the previously disappointing win rates. Disabling this cut-off is necessary for assuring that the N-Grams can be updated frequently and sufficient new move sequences are added.

It should however be noted that the average win rates are not completely representative of the individual win rates per map. On some maps the win rate increases when the N-Gram enhancement is added, while on other maps the win rate does not change much or even decreases slightly. This could be caused by a sub-optimal ϵ value for that map. Because the ϵ value was tuned for a normal MCTS player, the optimal ϵ value may be different for the MCTS-Solver player. Therefore, retuning the ϵ value for the MCTS-Solver player could improve the performance of this player. Another possible cause is that by not performing a cut-off, simulations take longer to complete. This results in having fewer simulations per second.

6 Conclusions & Future Research

In this paper the N-Gram and LGR simulation strategies were discussed as enhancements for MCTS for Tron. Furthermore, the ULM and PB selection strategies were suggested as enhancements as well. Multiple combinations of selection and simulation strategies were compared to each other, resulting in mostly conclusive results.

First of all, the N-Gram simulation strategy showed a clear improvement in performance over the MCTS player with a random simulation strategy. This indicates that the N-Gram simulation strategy is in fact a genuine improvement for the MCTS player for the game of Tron, achieving a 60.3% win rate. It has been shown that the optimal ϵ value varied heavily per map, making it difficult to pick one optimal value such that the N-Gram enhancement performed well on all maps.

Second, the LGR enhancement showed an even greater average win rate of 62.6% against a MCTS player with a random simulation strategy. The LGR enhancement is an improvement for MCTS for the game of Tron. The LGR strategy had slightly higher win rates than the N-Gram strategy, mainly due the fact that it achieved a higher number of simulation per second, resulting in more usable statistical information.

Third, the results showed that the playing strength of both N-Gram and the LGR strategies are quite evenly matched when playing against each other. The LGR achieved a slightly higher win rate of 50.2%. It was remarkable that the N-Gram simulation strategy, with 20,000 fewer simulations per second was such an even match for the LGR strategy. This suggests that the N-Gram strategy is better at extracting more useful information in fewer simulations. Experiments with more search time for both of the players did not indicate that this was the case.

Fourth, the combination of LGR and N-Grams did not show to be an improvement, because the overhead of updating all 5 of the tables needed by these strategies was too large. Win rates of 53.8%, 47.9% and 45.5% were achieved by the combo player against a random, N-Gram and LGR simulation strategy MCTS player, respectively. Additional experiments that provided the players with more search time showed almost no difference in performance.

Fifth, both of the selection strategies did not show to be an improvement for the MCTS player in the case of Tron. Average win rates of 50.9% and 47.7% were achieved by the ULM-MCTS and PB-MCTS players, respectively. Selection strategies do not seem to be a critical part for Tron, since the branching factor of the MCTS tree is low. Furthermore, the improvement in performance, if there was any, did not outweigh the added computation time of these enhancements.

Sixth, the N-Gram simulation strategy when combined with the MCTS-Solver player performs best when the MCTS-Solver player does not perform any cut-offs during the simulation or expansion phase. Win rates of 46.8% and 55.9% were achieved by the N-Gram enhanced MCTS-Solver player and the N-Gram enhanced MCTS-Solver player with no cut-offs, respectively. Taking the average 42.8% win rate of the no cut-off MCTS-Solver

into account, these results may indicate, that by not performing any cut-offs, more N-Gram relevant information is gathered from the MCTS process and a slightly higher win rate is achieved by the N-Gram MCTS-Solver player.

Future research could be done in dynamically determining what the best ϵ value for a given map is. This could vastly improve the performance of the N-Gram simulation strategy on some of the test maps. Another possible improvement is the development of a more aggressive selection strategy. This could be realized by using the discussed PB strategy with a higher W value, or a variant of ULM that is applied on all nodes instead of only ULM nodes. Another option is the use of progressive unpruning [5]. It would also be interesting to see how these enhancements fare against an $\alpha\beta$-player for Tron. Because this algorithm has proven to lead to the best performance, it would be the ultimate test that shows the true strength of these enhancements.

References

[1] Baier, H. and Drake, P. (2010). The Power of Forgetting: Improving the Last-Good-Reply Policy in Monte-Carlo Go. *IEEE Transactions on Computational Intelligence and AI in Games*, Vol. 2, No. 4, pp. 303–309.

[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] Browne, C., Powley, E.J., Whitehouse, D.l, Lucas, S.M., Cowling, P., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. (2012). A Survey of Monte-Carlo Tree Search Methods. *IEEE Trans. Comput. Intellig. and AI in Games*, Vol. 4, No. 1, pp. 1–43.

[4] Cazenave, T. and Saffidine, A. (2011). Score bounded Monte-Carlo tree search. *Proceedings of the 7th international conference on Computers and games*, CG 2010, pp. 93–104, Springer-Verlag, Berlin, Heidelberg.

[5] Chaslot, G.M.J-B., Winands, M.H.M., Uiterwijk, J.W.H.M., Herik, H.J. van den, and Bouzy, B. (2008). Progressive Strategies for Monte-Carlo Tree Search. *New Mathematics and Natural Computation*, Vol. 4, No. 3, pp. 343–357.

[6] Chen, K-H. and Zhang, P. (2008). Monte-Carlo Go with Knowledge-Guided Simulations. *ICGA Journal*, Vol. 31, No. 2, pp. 67–76.

[7] Coulom, R. (2007). Efficient selectivity and backup operators in Monte-Carlo tree search. *Computers and Games (CG 2006)* (eds. H.J. van den Herik, P. Ciancarini, and H.H.L.M. Donkers), Vol. 4630 of *LNCS*, pp. 72–83, Springer-Verlag, Heidelberg, Germany.

[8] Den Teuling, N.G.P. (2011). Monte-Carlo Tree Search for the Simultaneous Move Game Tron. Bachelor Thesis, Maastricht University.

[9] Drake, P. (2009). The Last-Good-Reply Policy for Monte-Carlo Go. *ICGA Journal*, Vol. 32, No. 4, pp. 221–227.

[10] Knuth, D.E. and Moore, R.W. (1975). An Analysis of Alpha-Beta Pruning. *Artificial Intelligence*, Vol. 6, No. 4, pp. 293–326.

[11] Kocsis, L. and Szepesvári, C. (2006). Bandit Based Monte-Carlo Planning. *Machine Learning: ECML 2006* (eds. J. Fürnkranz, T. Scheffer, and M. Spiliopoulou), Vol. 4212 of *LNCS*, pp. 282–293.

[12] Laramee, F.D. (2002). Using N-Gram statistical models to predict player behavior. *AI Programming Wisdom* (ed. S. Rabin), Vol. 1, Chapter 11, pp. 596–601. Charles River Media, Inc., Rockland, MA, USA.

[13] Manning, C.D. and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, USA.

[14] Millington, I. (2006). *Artificial Intelligence for Games (The Morgan Kaufmann Series in Interactive 3D Technology)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[15] Nijssen, J.A.M. and Winands, M.H.M. (2011). Enhancements for multi-player Monte-Carlo Tree Search. *Computers and Games (CG 2010)* (eds. H.J. van den Herik, H. Iida, and A. Plaat), Vol. 6151 of *LNCS*, pp. 238–249, Springer, Berlin Heidelberg, Germany.

[16] Samothrakis, S., Robles, D., and Lucas, S.M. (2010). A UCT Agent for Tron: Initial Investigations. *IEEE* (eds. Georgios N. Yannakakis and Julian Togelius), Conference on Computational Intelligence and Games (CIG), pp. 365–371, IEEE.

[17] Stankiewicz, J.A., Winands, M.H.M., and Uiterwijk, J.W.H.M. (2012). Monte-Carlo Tree Search Enhancements for Havannah. *Advances in Computer Games (ACG 13)*. Accepted.

[18] Sutton, R. and Barto, A. (1998). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA.

[19] Tak, M.J.W., Winands, M.H.M., and Björnsson, Y. (2012). N-Grams and the Last-Good-Reply Policy applied in General Game Playing. *IEEE Transactions on Computational Intelligence and AI in Games*, Vol. 4, No. 2. Accepted.

[20] Winands, M.H.M., Björnsson, Y., and Saito, J.T. (2010). Monte Carlo Tree Search in Lines of Action. *IEEE Trans. Comput. Intellig. and AI in Games*, Vol. 2, No. 4, pp. 239–250.