Monte Carlo Tree Search for Simultaneous Move Games: A Case Study in the Game of Tron

First author [a] Second author [a] Third author [a]

[a] *Department of Knowledge Engineering, Maastricht University, P.O. Box 616, 6200 MD Maastricht, The Netherlands*

Abstract

This paper investigates the application of Monte Carlo Tree Search (MCTS) to simultaneous move games. MCTS has been successfully applied to many board games, such as Chess, Checkers and Go. In this paper several variants of MCTS are investigated in order to adapt MCTS to simultaneous move games. Through the experiments, which will be conducted in the test domain in the game of Tron on four different boards, it is shown, that deterministic selection strategies such as sequential UCT, UCB1-Tuned and Decoupled UCT are superior to stochastic selection strategies, such as EXP3, Regret Matching and Decoupled UCT(mix).

1 Introduction

An important research topic in Artificial Intelligence (AI) is the development of intelligent agents. A classic benchmark is the ability to play games better than human experts. These games usually include classic board games, such as Chess [10], Checkers [16] and Go [11] because they are simple to learn yet hard to master. In classical game tree search, game-specific knowledge is used to determine the strength of each position using a statuc evaluation function [14]. If the evaluation function is too complex or a large search tree is required, then either the search has to be increased an alternative approach can be chosen.

Monte Carlo Tree Search (MCTS) [3, 5, 8] builds up a search tree without requiring an evaluation function. Instead, it builds the search tree by repeating four phases, as explained below. MCTS was initially applied to the game of Go [5] but has since been applied to many different games and settings [2]. This paper focuses on sampling policies and update rules in MCTS applies to two-player turn-taking simultaneous move games, such as Tron. Some algorithms are investigated in this paper, including sequential UCT [8] are: UCB1-Tuned, Decoupled UCT, Decoupled UCB1-Tuned EXP3 and Regret Matching.

In Tron, two players move at the same time through a discrete grid and at each move create a wall behind them. The first applications of MCTS to Tron, described in [15, 6], applied standard (sequential) UCT while treating the game as a turn-based alternative move game, in the search tree. A comparison of selection and update policies in simultaneous move MCTS are presented in [13]. However, results are only presented for a single map and there is no comparison to previous algorithms, such as sequential UCT. Throughout this paper, we investigated the impact of different selection and update strategies on the playing performance of Monte Carlo Tree Search in the game of Tron.

The paper is organized as follows. It starts with a brief description of the game Tron and Monte Carlo Tree Search in Subsection 2. Section 3 deals with how Monte Carlo Tree Search handles the game specific principles of Tron. In Section 4 the different selection strategies are explained. Afterwards experiments are shown in Section 5 and a conclusion is drawn from the Experiments in Section 6. Furthermore, possible future research is also discussed in Section 6.

2 Background: Tron and Monte Carlo Tree Search

Tron originates from the 1982 movie with the same name. It is a two-player game (See left half of Figure 1) played on discrete grids possibly obstructed by walls. In addition, the maps are mostly symmetric so that none of the players have an advantage. Unlike sequential and turn-taking games where payers play consecutively, at each step in Tron both players move simultaneously. The game is won if opponent crashes into a wall or moves off the board. If both players crash at the same turn into a wall, the game ends in a draw.

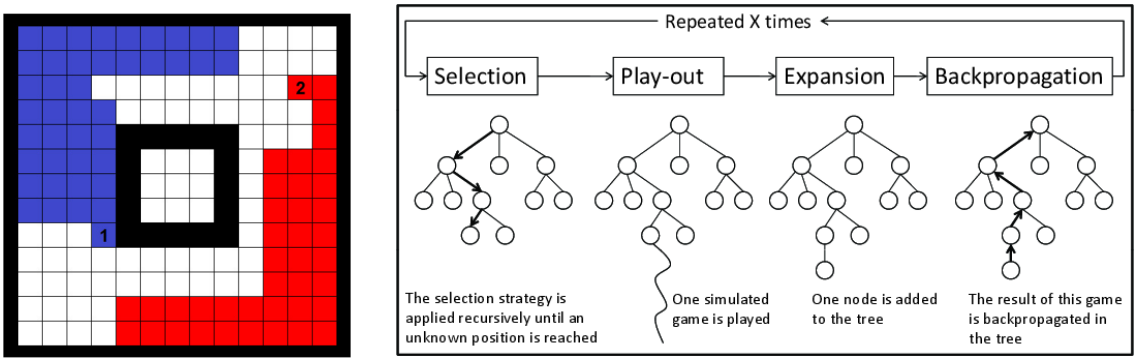

Figure 1: (Left) A game in Tron. 41 moves are already played. Player 1 started in the top left corner and Player 2 started in the bottom right corner. (Right) The four phases of the Monte Carlo Tree Search.

Monte Carlo Tree Search (MCTS) [5, 8] is a technique used for decision-making in the turn-based, sequential problems. To make a decision, MCTS makes use of simulations combined with an incrementally-built search tree. In the search tree, each node represents a state in the game. To evaluate a state, a game is simulated in self play from the current (root) state of the game until the game is finished. The first part of each simulation will encounter states that are part of the search tree. Which states are encountered in this first part depends on how future states are selected. When MCTS encounters a state that is not in the tree, it is expanded (added to the tree) and then a play-out policy takes over choosing successor states until the end of the game. The results of many simulations are back propagated to every node/state until the root node. MCTS is divided into four phases [4]: selection, play-out, expansion and backpropagation. These different phases are illustrated in the right half of Figure 1.

3 Tron-Specific MCTS

In the following subsections, it is explained, how MCTS is applied to the game of Tron [6]. In particular how the simultaneous moves of the game are handled and which heuristic knowledge of the game of Tron can be used to increase the performance of the MCTS.

Modeling Simultaneous Move Games

As described in Section 2, MCTS applies to sequential turned-based games. However, in Tron actions are chosen simultaneously. Our first model, used to implement Sequential UCT, ignores the presence of simultaneous moves and treat the game as a sequential turn-based game inside the search tree. We call this the *sequential tree model*. In practice, this worked well [15, 6], except it clearly favors one player which is especially problematic when players are close to each other. Therefore, in our version of Tron we add an exception to the expansion phase. If the first player moves to a position that the second player can also move to, the opponent is still able to move there to achieve a draw rather than a loss [6]. In this model, the game is sequential inside the search tree, until a leaf node is reached. The play-outs are then simulated as a simultaneous game [6].

Our second model, used to implement simultaneous move MCTS, stores a matrix at each node. We call this the *stacked matrix model*. Each cell of the matrix corresponds to a joint action, *i.e.,* an action chosen by both players simultaneously, and a corresponding child node (successor state). This is a more accurate representation of the underlying game since players are not able to base their current decision after having seen the other player's current move choice. This is the model used in [13, 9].

One contribution of this paper is the comparison of MCTS applied in these two different models.

Space Estimation, Predictive Expansion Strategy, and Play-out Cut-Offs

Tron is played in a grid-like environment and so often the two players become separated from each other. When this happens, each agent is essentially playing their own single-player game and the goal of the game becomes to outlast the opponent. Therefore the result can be computed by counting the number of squares captured by each player assigning a win to whoever has claimed the most space. The problem is that some positions might not offer a way back and therefore become suicide moves. For that reason a greedy wall-following algorithm can be used, which tries to fill out the remaining space by following a wall. When both players have filled their space, the moves which were made are counted and the player with the higher move count wins. This approach was proposed by Teuling [6].

Also, when players are separated, there is no need to let a play-out decide which player would win. Instead, it can be predicted by the Predictive Expansion Strategy (PES) [6]. PES is used to avoid play-outs when they are not necessary. Each time the non-root player tries to expand a node, the PES checks whether the two players are separated from each other. If this is the case, space estimation is used to predict which player would win. Finally, the expanded node becomes a leaf node and no more play-outs have to be done when reaching this node again.

4 Selection and Update Strategies

The default selection strategy is to choose a child node uniformly at random. This can be obviously improved by using heuristics and collected statistics. In the following Subsections, different selection and uopdate strategies are introduced including deterministic strategies such as Sequential UCT, UCB1-Tuned, DUCT(max) and DUCB1-Tuned(max), as well as stochastic strategies, which include DUCT(mix), DUCB1-Tuned(mix), Exp3 and Regret Matching.

4.1 Sequential UCT

The most common selection strategy is the Upper Confidence Bounds for Trees (UCT) [8]. The UCT strategy uses the Upper Confidence Bound (UCB1 [12]) algorithm. After a sufficient number of play-outs (parameter T), UCB1 is used to select a child, otherwise a child is selected randomly. This algorithm maintains a good balance between exploration and exploitation. UCB1 selects a child node k from a set of nodes K, from parent node j by using Equation 1:

$$k = \underset{i \in K}{\operatorname{argmax}} \left\{ \bar{X}_i + C\sqrt{\frac{\ln(n_j)}{n_i}} \right\}, \tag{1}$$

where n_i is the number of visits of child node i and $\bar{X}_i$ is the sample mean of the rewards of child node i. The parameters T and C are usually tuned to increase performance. For the game of Tron these are initially set [6] to $T = 30$ and $C = 10$.

An enhancement to the UCT selection strategy can be made by replacing the parameter C by a smart upper bound of the variance of the rewards [13]. This is either $\frac{1}{4}$, which is an upper bound of the variance of a *Bernoulli* random variable, or an upper confidence bound computed using Equation 2 which has the parameters the parent node j and some child node i. This variant is referred to as UCB1-Tuned [12]. Then, a child node k is selected from parent node j:

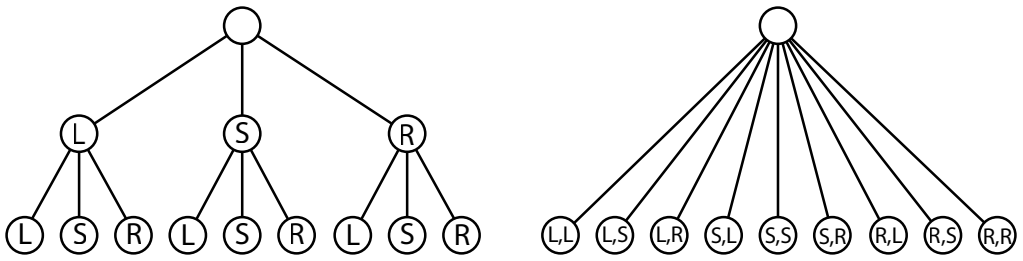

Figure 2: (Left) The sequential tree model. Each node represent the child node of the corresponding move. The first level represents Player 1 moves and the second level represents Player 2 moves. L,S and R represents the different moves a player can make. (Right) The stacked matrix model. Each node corresponds to a child node from a combination of actions for Player 1 and Player 2.

$$k = \underset{i \in K}{\operatorname{argmax}} \left\{ \bar{X}_i + \sqrt{\frac{\min(\frac{1}{4}, \text{Var}_{UCB1}(j,i)) \ln(n_j)}{n_i}} \right\}, \quad \text{Var}_{UCB1}(j,i) = \bar{s}_k^2 + \sqrt{\frac{2\ln(n_j)}{n_i}}, \quad (2)$$

where $\bar{s}_k^2$ is the sample variance of the observed rewards for child node k.

4.2 Decoupled UCT

Unlike standard sequential UCT and UCB1-Tuned apply to the sequential tree model, Decoupled UCT (DUCT) applies UCB1 selection in the stacked matrix model for each player separately. In DUCT a node stores the both moves, the one from player 1 and from player 2. UCB1 selects a move twice, once for each player, and independently of the other player's choice. To better illustrate the difference between these two concepts, see Figure 2. In the left figure, the sequential tree model is shown. In the right figure figure, each node contains two moves, one belonging to player 1 and one to player 2. These combinations are called *joint actions*. Because each level in the search tree represents now one step in the game, the branching factor increases from three to nine.

When selecting a child node, DUCT applies the default UCB1 algorithm which was described in Equation 1 with the statistics from each player's perspective independently. After a move is selected player 1, the selection process is repeated for player 2 (without knowledge of the choice made by player 1) using the statistics from player 2's perspective. These two actions are combined to form a joint action. The final move, after many simulations, can be selected in two different ways. The first, DUCT(max), selects the action, with the most visits. DUCT(mix) normalizes the visit counts and samples an action according to this distribution, *i.e.,* using a mixed strategy [9]. To the best of our knowledge, DUCT(max) and DUCT(mix) where first used in general game-playing programs [7, 17].

Just as an enhancement can be made by replacing the parameter C by a smart upper bound of the variance of the rewards in UCT, it can also be made to DUCT. Each time a node is selected and a joint action is chosen, Equations 2 are used. We refer to this variant as Decoupled UCB1-Tuned.

4.3 Exp3

To this point, all selection strategies introduced, except DUCT(mix), are deterministic strategies. DUCT(mix) and Exp3 [1] belong to the group of stochastic selection strategies, which means that there is a random factor involved and instead actions are sampled according to some probability distribution. Exp3, as DUCT, always uses the stacked matrix model and hence selects joint actions. Exp3 stores a list of estimated sums

of payoffs $\hat{X}_{a_k^p}$, where a_k^p refers to player p's action k. From the list of payoffs, a policy P is created. The probability of choosing action a_k^p of policy P is shown in Equation 3,

$$P_{a_k^p} = \frac{e^{\eta\omega(a_k^p)}}{\sum\limits_{i \in A_p} e^{\eta\omega(a_i^p)}}, \qquad \hat{X}_{a_k^p} = \hat{X}_{a_k^p} + \frac{r^p_{a_{k_1}, a_{k_2}}}{\sigma_{a_k^p}}, \tag{3}$$

where K_p is the set of actions from player p, ω can be scaled by some constant η, $r^p_{a_{k_1}, a_{k_2}}$ is the reward of the play-out when player 1 player chose move k_1 and player 2 chose move k_2 and is give in respect to player p. For simplicity, $\eta = 1$ is chosen. In standard Exp3, $\omega(a_k^p) = \hat{X}_{a_k^p}$, but in practice we use $\omega(a_k^p) = \hat{X}_{a_k^p} - \operatorname{argmax}_{i \in K_p} \hat{X}_{a_i^p}$ since it is equivalent and more numerically stable [9]. The action selected is then sampled from the mixed strategy where action a_k^p is selected with probability $\sigma_{a_k^p} = (1-\gamma)P_{a_k^p} + \frac{\gamma}{|K_p|}$.

Parameter γ can be optimized by tuning it. The update of $\hat{X}_p(a_n)$ after selecting a joint action (a_1, a_2), by using the probability $\sigma_i(a_j)$, which returned some simulation result of a play-out $r^p_{a_{k_1}, a_{k_2}}$ is given in Equation 3. As in DUCT(mix), the final move is sampled from the normalized visit count distribution.

4.4 Regret Matching

Regret Matching, as EXP3 and DUCT, selects always joint actions. Opposed to the other strategies, Regret Matching stores a matrix M with the estimated mean of the rewards (See Equation 4, where $\bar{X}_{m,n}$ is the mean of the rewards for player 1 when the joint action $(a_1 = m, a_2 = n)$ was selected).

$$M = \begin{bmatrix} \bar{X}_{1,1} & \bar{X}_{2,1} & \bar{X}_{3,1} \\ \bar{X}_{1,2} & \bar{X}_{2,2} & \bar{X}_{3,2} \\ \bar{X}_{1,3} & \bar{X}_{2,3} & \bar{X}_{3,3} \end{bmatrix} \qquad \begin{array}{l} \forall a_i^1 \in K_1, R_{a_i^1} = R_{a_i^1} + (\bar{X}_{i,n} - r^1_{a_m, a_n}) \\ \forall a_i^2 \in K_2, R_{a_i^2} = R_{a_i^2} + (\bar{X}_{m,i} - r^2_{a_m, a_n}) \end{array} \tag{4}$$

Additional to matrix M, two lists are stores which keep track of the cumulative regret for not taking move a_k^p, denoted $R_{a_k^p}$. The regret is a value, which indicates how much the player regrets not having played this action. A policy P is then constructed created by normalizing over the positive cumulative regrets (*e.g.,* if the regrets are $R_{a_1^1} = 8.0$, $R_{a_2^1} = 5.0$ and $R_{a_3^1} = -4.0$, then the policy is the probability distribution $(\frac{8}{13}, \frac{5}{13}, 0)$. As in Exp3, the selected action is sampled from $\sigma_{a_k^p} = (1-\gamma)P_{a_k^p} + \frac{\gamma}{|K_p|}$. where the variable γ can be tuned to increase performance as in Exp3.

Initially, all values in matrix M and all values in the regret lists are set to zero. After the play-out is finished and the result ($r^p_{a_m, a_n}$) of it gets back propagated, the cumulative reward values for each cell are updated using $X_{m,n} = X_{m,n} + r^p_{a_m^1, a_n^2}$ and the regret values are updated using the right side of Equation 4. The final move is selected as in DUCT(Mix) and EXP3 by using a mixed strategy by normalizing over the visit counts.

5 Experiments

In this section the different selection strategies, which were proposed in Section 4 are tested. Because the framework, which was implemented by Teuling [6], used for this experiments is based on sequential UCT, DUCT, EXP3 and Regret Matching were not optimally implemented. Therefore it would be an unfair comparison, if all agents would have the same thinking time. In order to make it a fair comparison, each agent is allowed to simulate a fixed number of simulations. In the case of this experiment the number is set to 100,000. The experiments are run on four different boards (Three boards with obstacles on it (See Fig. 3) and an empty board (d)), all with dimensions of 16×16. On each board 200 games are played and, even though all the boards are symmetric, the starting positions are switched after half of the games, to assure that none of the agents are benefiting from starting in a certain position. The play-out strategy, which is used in all experiments is the random strategy with play-out cut-off as an enhancement. Also, the expansion strategy uses the predictive expansion strategy.

Some parameters are tuned in Subsection 5.1 and the several selection strategies introduced in this paper,

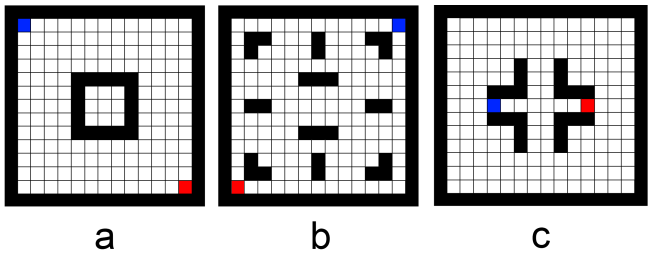

Figure 3: Four different boards are used for the experiments in the round-robin tournament. a,b and c, as well as the empty board, d, of the same size.

are tested against each other in a round-robin tournament in Subsection 5.2. In addition another round-robin tournament is conducted on a smaller 6×6 board to see how the results differ from the ones conducted in the round-robin tournament on the large boards.

5.1 Parameter Tuning

In the beginning, some parameters (C and T in UCT and γ in EXP3 and Regret Matching) are tuned. This is done, to assure that the different strategies perform optimally. As reference constants, values are used, which were taken from different sources [6, 13]. Parameter C, which is used in UCB1 (See Equation 1) was tuned, by letting a UCT player playing games against a MCTS player with a random selection and a random play-out strategy. After 10 games the value of the parameter C was slightly increased or decreased, depending on how strong the UCT player played. Starting with the reference constant, 120 games were played in order to find a value for C which did not change significantly anymore.
The parameter T used in UCT and UCB1-Tuned, which is the threshold before either UCT or UCB1-Tuned is applied as a selection strategy was tuned in the same way as the parameter C. This parameter tuning showed the same result as shown by Teuling [6], namely that the parameter T does not influence the performance of UCT and UCB1-Tuned.
The parameter γ of EXP3 and Regret Matching was also tuned in the same way as the parameter C with the only difference, that γ must be in the interval of $[0, 1]$.
All the tuned values can be seen in Table 1.

5.2 Round-Robin Tournament

In this subsection, several players using different selection strategies are tested. In Table 2 the results of the games on all four maps are seen and Table 3 shows the average performance of all players.

From Table 4 we can see, that the sequential UCT and UCB1-Tuned selection strategies perform best. The Max versions of the decoupled UCT and UCB1-Tuned strategy still performs reasonably well ($> 50\%$). All the other strategies, which include the Mix versions of the decoupled UCT and UCB1-Tuned, as well as EXP3 and Regret Matching, perform not that well ($< 50\%$).

Table 1: Results of the parameter-tuning of parameters C and T from the UCT Algorithm and from the parameter γ of the EXP3 algorithm

Parameter	Reference constant	Tuned value
C	10 [6]	13.2
T	30 [6]	30
γ(EXP3)	0.36 [13]	0.39
γ(Regret Matching)	0.36	0.31

Table 2: Results of the different selection strategies playing on Board a,b,c and d.

Board a	UCT	UCB1T	DUCT(Max)	DUCT(Mix)	DUCB1T(Max)	DUCB1T(Mix)	EXP3	RM
UCT	-	61%	51%	54%	53%	67%	65%	62%
UCB1T	39%	-	53%	61%	52%	65%	66%	61%
DUCT(Max)	49%	47%	-	67%	43%	62%	63%	67%
DUCT(Mix)	46%	39%	33%	-	37%	48%	53%	51%
DUCB1T(Max)	47%	48%	57%	63%	-	61%	64%	69%
DUCB1T(Mix)	33%	35%	38%	52%	39%	-	54%	51%
EXP3	35%	34%	37%	47%	36%	46%	-	51%
RM	38%	39%	33%	49%	31%	49%	49%	-
Board b	UCT	UCB1T	DUCT(Max)	DUCT(Mix)	DUCB1T(Max)	DUCB1T(Mix)	EXP3	RM
UCT	-	47%	49%	57%	51%	61%	61%	59%
UCB1T	53%	-	51%	54%	49%	65%	64%	57%
DUCT(Max)	51%	49%	-	54%	51%	58%	54%	52%
DUCT(Mix)	43%	46%	46%	-	41%	52%	49%	46%
DUCB1T(Max)	49%	51%	49%	59%	-	68%	62%	65%
DUCB1T(Mix)	39%	35%	42%	48%	32%	-	57%	52%
EXP3	39%	36%	46%	51%	38%	43%	-	57%
RM	41%	43%	48%	54%	35%	48%	43%	-
Board c	UCT	UCB1T	DUCT(Max)	DUCT(Mix)	DUCB1T(Max)	DUCB1T(Mix)	EXP3	RM
UCT	-	37%	81%	89%	84%	96%	71%	72%
UCB1T	63%	-	79%	71%	88%	89%	73%	70%
DUCT(Max)	19%	21%	-	58%	41%	52%	69%	61%
DUCT(Mix)	11%	29%	42%	-	39%	49%	51%	51%
DUCB1T(Max)	16%	12%	59%	61%	-	59%	68%	64%
DUCB1T(Mix)	4%	11%	48%	51%	41%	-	50%	58%
EXP3	29%	27%	31%	49%	32%	50%	-	49%
RM	28%	30%	39%	49%	36%	42%	51%	-
Board d	UCT	UCB1T	DUCT(Max)	DUCT(Mix)	DUCB1T(Max)	DUCB1T(Mix)	EXP3	RM
UCT	-	52%	48%	51%	50%	64%	67%	65%
UCB1T	48%	-	45%	57%	53%	67%	59%	67%
DUCT(Max)	52%	55%	-	60%	49%	61%	59%	62%
DUCT(Mix)	49%	43%	40%	-	41%	55%	50%	49%
DUCB1T(Max)	50%	47%	51%	59%	-	57%	64%	68%
DUCB1T(Mix)	36%	33%	39%	45%	43%	-	51%	49%
EXP3	33%	41%	41%	50%	36%	49%	-	52%
RM	35%	23%	38%	51%	32%	51%	48%	-

UCB1T=UCB1-Tuned, DUCB1T(Max)=DUCB1-Tuned(Max), DUCB1T(Mix)=DUCB1-Tuned(Mix), RM=Regret Matching

Table 3: Average results of the different selection strategies playing against each other.

Total	UCT	UCB1T	DUCT(Max)	DUCT(Mix)	DUCB1T(Max)	DUCB1T(Mix)	EXP3	RM
UCT	-	49%	57%	63%	60%	72%	66%	64%
UCB1T	51%	-	57%	61%	61%	71%	65%	63%
DUCT(Max)	43%	43%	-	60%	46%	58%	61%	60%
DUCT(Mix)	37%	39%	40%	-	40%	51%	51%	49%
DUCB1T(Max)	40%	39%	54%	60%	-	61%	65%	67%
DUCB1T(Mix)	18%	19%	42%	49%	39%	-	53%	52%
EXP3	34%	35%	39%	49%	35%	47%	-	52%
RM	36%	37%	40%	51%	33%	48%	48%	-

UCB1T=UCB1-Tuned, DUCB1T(Max)=DUCB1-Tuned(Max), DUCB1T(Mix)=DUCB1-Tuned(Mix), RM=Regret Matching

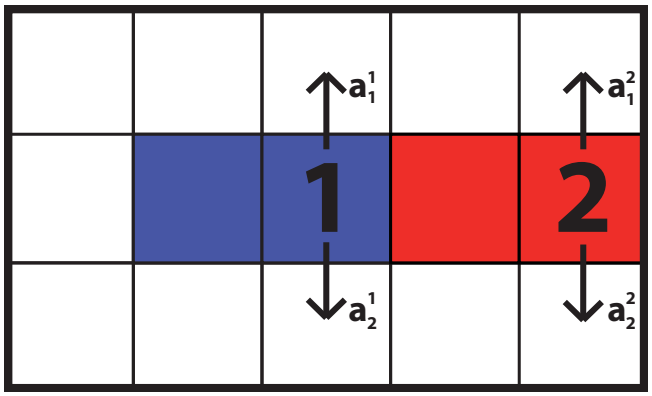

Figure 4: A Position, which indicates that mixing strategies is important.

Furthermore it stands out that on Board c, the sequential MCTS out performs the decoupled versions of MCTS.

The same selection strategies were tested again against each other in another round-robin tournament, which will be played on a 6×6 board. From the results in Table 5 can be seen, that deterministic selection strategies are still doing better that stochastic ones. But, stochastic strategies perform better on the 6×6 board than on the larger 16×16 board.

5.3 Discussion

The experiments revealed that the parameter C of the UCT selection strategy was optimal at 13.2. Usually, the parameter C is fairly low, but in Tron, there are only 3 possible moves, therefore it is beneficial to know where all three moves would lead us. A high C parameter ensures that the exploration of all nodes is given, which explains the result of the high C parameter.

The parameter T of the UCT and UCB1-Tuned selection strategy was tuned. After some experiments, the tuning was stopped, because it did not seem to have much impact on the result of the selection strategies. Therefore, 30 is still used for the parameter T. The parameter γ from the EXP3 and Regret Matching selection strategy was tuned to 0.39 and 0.31, respectively, to increase performance of those strategies.

The round-robin tournament, in which every selection strategy played against each other, showed that the sequential UCB1-Tuned algorithm wins on average 63% of its games and is therefore the winner of the tournament. Further it may be concluded, that the sequential strategies perform better than the decoupled strategies, which is especially visible on Board c (See Table 2), where the UCT and UCB1-Tuned only lost 4%-19% of its games against all DUCT strategies and only 28%-29% against EXP3 and Regret Matching, respectively. Additionally, it can be concluded that deterministic selection strategies perform superior than stochastic selection strategies on 16×16 boards (All deterministic strategies win $> 50\%$ and all stochastic win $< 50\%$ of their matches). Already the selection strategies, which are deterministic, but only use a stochastic process to determine their final move, such as DUCT(Mix) and DUCB1-Tuned(Mix), perform way weaker than their counterparts DUCT(Max) and DUCB1-Tuned(Max).

Furthermore, the round-robin tournament on the smaller 6×6 board showed, that if a smaller board is used, the performance of the deterministic strategies decrease and the performance of the stochastic strategies increase. This indicates, that as soon as both players come closer together, using a stochastic strategy might be more suitable than a deterministic one. This phenomenon is illustrated in Figure 4, where both players have two possible moves. If both players choose a_1^p ("Left"), Player 1 wins and if both players choose a_2^p ("Right"), Player 1 also wins. Therefore it might be beneficial for Player 2 to play with a mixed strategy, where the probabilities for choosing actionfor $a_1^2 = 0.5$ and for $a_2^2 = 0.5$.

6 Conclusion and Future Research

Table 4: Results of the different selection strategies playing against each other. a,b,c and d stand for the four different boards, which are used in the round-robin tournament. $\pm$ refers to 95% confidence intervals.

Selection Strategy	a	b	c	d	Total
UCB1-Tuned	57%	58%	78%	58%	63$\pm$1.26%
UCT	59%	55%	76%	57%	61$\pm$1.28%
DUCB1-Tuned(Max)	58%	58%	48%	57%	55$\pm$1.30%
DUCT(Max)	56%	53%	37%	57%	51$\pm$1.31%
DUCT(Mix)	44%	46%	39%	47%	45$\pm$1.30%
DUCB1-Tuned(Mix)	43%	44%	38%	42%	42$\pm$1.29%
EXP3	41%	44%	38%	43%	42$\pm$1.29%
Regret Matching	41%	44%	39%	40%	41$\pm$1.28%

In this paper, several selection strategies for the selection phase were introduced, including the deterministic strategies UCT, UCB1-Tuned, DUCT(Max) and DUCB1-Tuned(Max) and the stochastic strategies DUCT(Mix), DUCB1-Tuned(Mix), EXP3 and Regret Matching. All these enhancements were tested in the game of Tron on different boards to see which strategies perform better than others.
Overall, the round-robin tournament showed, that UCB1-Tuned performs the best in the game of Tron. Furthermore the experiments showed, that deterministic strategies are superior to stochastic ones, but also that the performance of stochastic strategies increases as the board gets smaller. It also showed, that the layout of the board has great influences on the outcome of a game.
For future research, more experiments with different boards are required, in order to determine why some boards (as Board c) create more difficulties for the decoupled strategies.
In addition, the selection strategy EXP3 can be enhanced by tuning parameter η, which was set to $\eta = 1$ for simplicity.
Moreover, a hybrid selection strategy should be tested, which uses a deterministic strategy if both players are far away from each other and a stochastic one as soon as both players come fairly close to each other. This hybrid selection strategy would indicate, if the assumption made in this paper, that stochastic strategies perform better when both players are close to each other, is correct.

References

[1] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire. Gambling in a rigged casino: The adversarial multi-armed bandit problem. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, pages 322–331, 1995.

Table 5: Results of the different selection strategies playing against each other on a 6×6 board. $\pm$ refers to 95% confidence intervals.

6×6 Board	UCB1T	UCT	DUCT(Max)	DUCB1T(Max)	DUCT(Mix)	DUCB1T(Mix)	RM	EXP3	Total
UCB1T	-	52%	59%	62%	59%	62%	57%	61%	59$\pm$2.57%
UCT	48%	-	55%	61%	57%	56%	60%	59%	57$\pm$2.56%
DUCT(Max)	41%	45%	-	48%	56%	56%	54%	60%	52$\pm$2.62%
DUCB1T(Max)	38%	39%	52%	-	56%	57%	63%	61%	52$\pm$2.62%
DUCT(Mix)	41%	43%	44%	44%	-	53%	50%	49%	47$\pm$2.61%
DUCB1T(Mix)	38%	44%	44%	43%	43%	-	51%	54%	46$\pm$2.61%
RM	43%	40%	46%	37%	50%	46%	-	46%	46$\pm$2.61%
EXP3	39%	41%	40%	39%	51%	41%	54%	-	44$\pm$2.60%

UCB1T=UCB1-Tuned, DUCB1T(Max)=DUCB1-Tuned(Max), DUCB1T(Mix)=DUCB1-Tuned(Mix), RM=Regret Matching

[2] C.B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, March 2012.

[3] G.M.J-B. Chaslot. *Monte-Carlo Tree Search*. PhD thesis, Department of Knowledge Engineering, Maastricht University, Netherlands, 2010. Ph.D. dissertation.

[4] G.M.J-B. Chaslot, M.H.M. Winands, H.J. van den Herik, J.W.H.M. Uiterwijk, and B. Bouzy. Progressive strategies for Monte-Carlo tree search. *New Mathematics and Natural Computation*, 4(3):343–357, 2008.

[5] R. Coulom. Efficient selectivity and backup operators in Monte Carlo Tree Search. In *CG 2008*, volume 4630 of *LNCS*, pages 72–83, 2007.

[6] N.G.P. Den Teuling and M.H.M. Winands. Monte-Carlo Tree Search for the simultaneous move game Tron. In *Proceedings of Computer Games Workshop (ECAI)*, pages 126–141, June 2012.

[7] H. Finnsson. Cadia-player: A general game playing agent. Master thesis, Reykjavik University, Iceland - School of Computer Science, December 2007.

[8] L. Kocsis and C. Szepesvári. Bandit based Monte Carlo planning. volume 5131 of *LNCS*, pages 282–293, 2006.

[9] M. Lanctot, V. Lisý, and M.H.M. Winands. Monte carlo tree search in simultaneous move games with applications to Goofspiel. In *Proceedings of IJCAI 2013 Workshop on Computer Games*, 2013.

[10] A. Hoane Jr. M. Campbell and F. Hsu. Deep Blue. *Artificial Intelligence*, 134(1):57–83, 2002.

[11] M. Müller. Computer Go. *Artificial Intelligence*, 134(1):145–179, 2002.

[12] N. Cesa-Bianchi P. Auer and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(3):235–256, 2002.

[13] F. Maes P. Perick, D. L. St-Pierre and D. Ernst. Comparison of different selection strategies in Monte-Carlo Tree Search for the game of TRON. pages 242–249, 2012. 2012 IEEE Conference on Computational Intelligence and Games (CIG'12).

[14] S. Russel and P. Norvig. *Artificial Intelligence - A Modern Approach*. Pearson, 3rd edition, 2010.

[15] D. Robles S. Samothrakis and S.Lucas. An UCT agent for TRON: Initial investigations. pages 365–371, 2010. 2010 IEEE Conference on Computational Intelligence and Games (CIG'10).

[16] Jonathan Schaeffer. *One Jump Ahead: Computer Perfection In Checkers*. Springer, 2009.

[17] M. Shafiei, N. R. Sturtevant, and J. Schaeffer. Comparing UCT versus CFR in simultaneous games. In *Proceeding of the IJCAI Workshop on General Game-Playing (GIGA)*, pages 75–82, 2009.