

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

Christopher Wittlinger<sup>a</sup>      Marc Lanctot<sup>a</sup>      Mark H.M. Winands<sup>a</sup>  
Niek G.P. Den Teuling<sup>a</sup>

<sup>a</sup> *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 the simultaneous move game Tron. MCTS has been successfully applied to many board games, such as Chess, Checkers and Go. In this paper several variants of MCTS are proposed to adapt MCTS to simultaneous move games. Through the experiments 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 perform better than 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. Classic examples include Chess, Checkers and Go, each of which 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 static evaluation function. If the evaluation function is too complex or a large search tree is required, then either the search has to be increased or an alternative approach can be chosen.

Monte Carlo Tree Search (MCTS) [4, 6, 10] builds up a search tree without requiring an evaluation function. Instead, it builds a search tree iteratively by repeating four phases, as explained below. MCTS was initially applied to the game of Go [6] but has since been applied to many different games and settings [3]. This paper focuses on sampling policies and update rules in MCTS applied to two-player turn-taking simultaneous move games, such as Tron. Some algorithms are investigated in this paper, including sequential UCT [10] 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 [14, 7] 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 sequential UCT previously used in this domain. 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 part 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 turn-taking games where players 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.

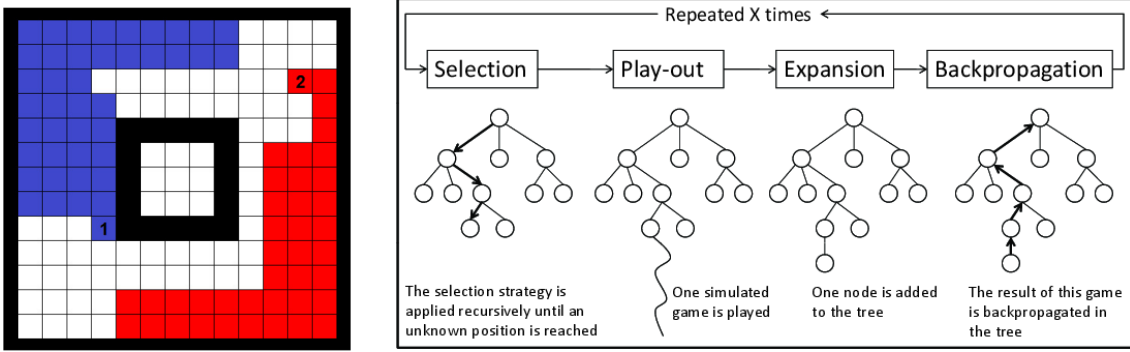


Figure 1: (Left) A game in Tron. 41 moves are already played. Player 1 started in the top left corner and Player 2 started in the bottom right corner. (Right) The four phases of the Monte Carlo Tree Search.

Monte Carlo Tree Search (MCTS) [6, 10] 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 the actions selected during the simulation (the so-called *selection policy*). 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 result of the simulation is then *back-propagated* to every node visited in the simulation up to the root where node statistics are updated accordingly. The right part of Figure 1 illustrates the four different phases of a simulation [5].

## 3 Tron-Specific MCTS

As described in Section 2, MCTS applies to sequential turned-based games. However, in Tron actions are chosen simultaneously. This section describes two models to handle the simultaneous moves that occur in Tron.

The first model, used to implement Sequential UCT, ignores the presence of simultaneous moves and treats the game as a sequential turn-based game inside the search tree. We call this the *sequential tree model*. In practice, this worked well [14, 7], except it clearly favours one player which is especially problematic when players are close to each other. 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 [7]. In this paper, the player running the search always moves first. In essence, in the sequential model, the searching player learns to play safely and chooses the move that will lead to the opponent's least best counter-move. In fact, in the classic minimax setting the value computed in this model is a lower bound of the true optimal value [2, Lemma 4.1].

The 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, 11].

A novel 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. 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 [7].

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) [7]. 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 update 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) [10]. The UCT strategy uses the Upper Confidence Bound (UCB1 [12]) algorithm. After each child has been at least selected once, UCB1 is used to select a child. 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:

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

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 parameter  $C$  is usually tuned to increase performance.

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  $j$ :

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

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

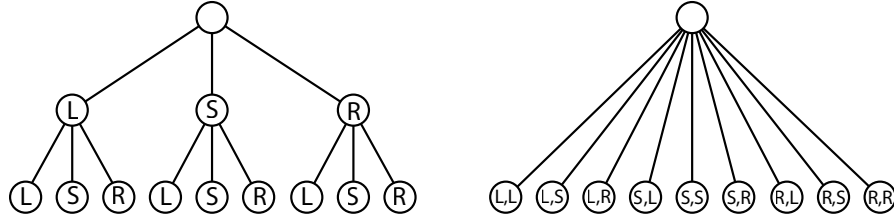


Figure 2: (Left) The sequential tree model. Each node represent the child node of the corresponding move. The first level represents player 1 moves and the second level represents player 2 moves. L,S and R represents the different moves a player can make. (Right) The stacked matrix model. Each node corresponds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

## 4.2 Decoupled UCT

Unlike standard sequential UCT and UCB1-Tuned used in 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 is nine instead of three.

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 [11]. To the best of our knowledge, DUCT(max) and DUCT(mix) were first used in general game-playing programs [8, 15].

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, except for the final move selection in DUCT(mix), policies for selecting actions have been deterministic. Exp3 [1] belongs 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,

$$P_{a_k^p} = \frac{e^{\eta\omega(a_k^p)}}{\sum_{i \in A_p} e^{\eta\omega(a_i^p)}}, \quad \hat{X}_{a_k^p} = \hat{X}_{a_k^p} + \frac{r_{a_{k_1}^1, a_{k_2}^2}^p}{\sigma_{a_k^p}^p}, \quad (3)$$

where  $K_p$  is the set of actions from player  $p$ ,  $\omega$  can be scaled by some constant  $\eta$ ,  $r_{a_{k_1}^1, a_{k_2}^2}^p$  is the reward of the play-out when player 1 chose move  $k_1$  and player 2 chose move  $k_2$  and is give in respect to player  $p$ . In standard Exp3,  $\omega(a_k^p) = \hat{X}_{a_k^p}$ , but in practice we use  $\omega(a_k^p) = \hat{X}_{a_k^p} - \arg\max_{i \in K_p} \hat{X}_{a_i^p}$  since it

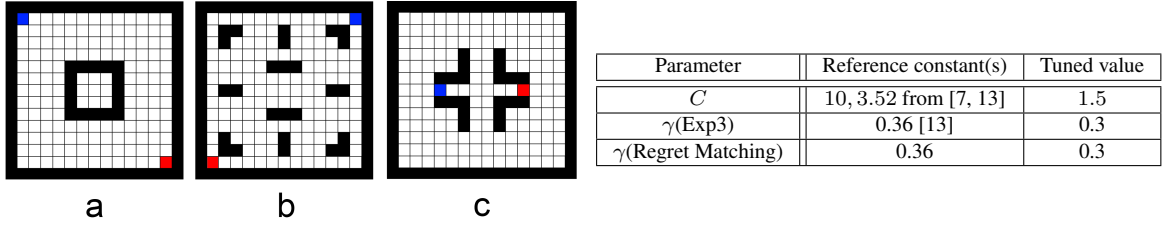


Figure 3: Left: Different boards are used for the experiments in the round-robin tournament. Right: tuned parameter constants.

is equivalent and more numerically stable [11]. The action selected is then sampled from the mixed strategy where action  $a_k^p$  is selected with probability  $\sigma_{a_k^p} = (1 - \gamma)P_{a_k^p} + \frac{\gamma}{|K_p|}$ . As in [11], we set  $\eta = \gamma/K$ .

Parameter  $\gamma$  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  $r_{a_{k_1}, a_{k_2}}^p$  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 (RM) [9], 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).

$$M = \begin{bmatrix} \bar{X}_{1,1} & \bar{X}_{2,1} & \bar{X}_{3,1} \\ \bar{X}_{1,2} & \bar{X}_{2,2} & \bar{X}_{3,2} \\ \bar{X}_{1,3} & \bar{X}_{2,3} & \bar{X}_{3,3} \end{bmatrix} \quad \begin{aligned} \forall a_i^1 \in K_1, R_{a_i^1} &= R_{a_i^1} + (\text{reward}_{i,n}^1 - r_{a_m, a_n}^1) \\ \forall a_i^2 \in K_2, R_{a_i^2} &= R_{a_i^2} + (\text{reward}_{m,i}^2 - r_{a_m, a_n}^2) \end{aligned} \quad (4)$$

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  $\gamma$  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  $(r_{a_m, a_n}^p)$  for player  $p$  gets back propagated, the cumulative reward values for each cell are updated using  $X_{m,n} = X_{m,n} + r_{a_m, a_n}^p$  and the regret values are updated using the right side of Equation 4, where  $\text{reward}_{m', n'}^p = r_{a_m, a_n}^p$  if  $(m', n') = (m, n)$  or  $\bar{X}_{m,n}$  otherwise. The final move is selected using the average of all the mixed strategies used over all simulations as described in [11].

## 5 Experiments

In this section the different selection and update strategies are evaluated. In order to make it a fair comparison between the two search tree models using a common implementation, each agent is allowed to simulate a fixed number of simulations (100,000).

The experiments are run on four different boards, three with obstacles (see Figure 3) and an empty board (d)), all with dimensions of  $13 \times 13$ . On each board 200 games are played with swapped starting positions. The play-out strategy, which is used in all experiments, is the random strategy with play-out cut-offs enhancement mentioned in Section 3. The expansion strategy uses the predictive expansion strategy.

Before running performance experiments, some parameters ( $C$  in UCT,  $\gamma$  in Exp3 and Regret Matching) are tuned. As reference constants, values are used which were taken from different sources [7, 13]. Parameter  $C$ , which is used in Equation 1) was tuned by letting a UCT player playing games against a MCTS

Board a	UCT	UCBIT	DUCT(max)	DUCT(mix)	DUCBIT(max)	DUCBIT(mix)	Exp3	RM
UCT	-	44%	51%	58%	40%	45%	53%	41%
UCBIT	56%	-	67%	68%	48%	55%	60%	51%
DUCT(max)	49%	33%	-	53%	28%	32%	73%	35%
DUCT(mix)	42%	32%	47%	-	28%	33%	58%	35%
DUCBIT(max)	60%	52%	72%	72%	-	59%	78%	63%
DUCBIT(mix)	55%	45%	68%	67%	41%	-	67%	48%
Exp3	47%	40%	27%	42%	22%	33%	-	33%
RM	59%	49%	65%	65%	37%	52%	67%	-
Board b	UCT	UCBIT	DUCT(max)	DUCT(mix)	DUCBIT(max)	DUCBIT(mix)	Exp3	RM
UCT	-	50%	51%	57%	49%	51%	72%	50%
UCBIT	50%	-	53%	65%	50%	52%	70%	56%
DUCT(max)	49%	47%	-	63%	41%	46%	79%	52%
DUCT(mix)	43%	35%	37%	-	24%	32%	70%	38%
DUCBIT(max)	51%	50%	59%	76%	-	55%	83%	62%
DUCBIT(mix)	49%	48%	54%	68%	45%	-	80%	54%
Exp3	28%	30%	21%	30%	17%	20%	-	23%
RM	50%	44%	48%	62%	38%	46%	77%	-
Board c	UCT	UCBIT	DUCT(max)	DUCT(mix)	DUCBIT(max)	DUCBIT(mix)	Exp3	RM
UCT	-	64%	57%	56%	46%	54%	56%	50%
UCBIT	36%	-	44%	60%	42%	47%	62%	52%
DUCT(max)	43%	56%	-	55%	42%	49%	57%	41%
DUCT(mix)	44%	40%	45%	-	28%	36%	57%	49%
DUCBIT(max)	54%	58%	58%	72%	-	60%	75%	61%
DUCBIT(mix)	46%	53%	51%	64%	40%	-	64%	55%
Exp3	44%	38%	43%	43%	25%	36%	-	36%
RM	50%	48%	59%	51%	39%	45%	64%	-
Board d	UCT	UCBIT	DUCT(max)	DUCT(mix)	DUCBIT(max)	DUCBIT(mix)	Exp3	RM
UCT	-	46%	47%	57%	44%	48%	52%	48%
UCBIT	54%	-	51%	60%	50%	52%	57%	53%
DUCT(max)	53%	49%	-	64%	35%	45%	58%	48%
DUCT(mix)	43%	40%	36%	-	29%	35%	39%	30%
DUCBIT(max)	56%	50%	65%	71%	-	55%	65%	56%
DUCBIT(mix)	52%	48%	55%	65%	45%	-	56%	50%
Exp3	48%	43%	42%	61%	35%	44%	-	42%
RM	52%	47%	52%	70%	44%	50%	58%	-

Table 1: Results of the different selection strategies playing on Board a, b, c and d. Each percentages refers to the win rate of the row player.

Total	UCT	UCBIT	DUCT(max)	DUCT(mix)	DUCBIT(max)	DUCBIT(mix)	Exp3	RM
UCT	-	50%	51%	55%	46%	49%	57%	48%
UCBIT	50%	-	53%	61%	48%	51%	60%	52%
DUCT(max)	49%	47%	-	58%	39%	45%	63%	45%
DUCT(mix)	45%	39%	42%	-	31%	36%	55%	40%
DUCBIT(max)	54%	52%	61%	69%	-	56%	70%	59%
DUCBIT(mix)	51%	49%	55%	64%	44%	-	64%	51%
Exp3	43%	40%	37%	45%	30%	36%	-	34%
RM	52%	48%	55%	60%	41%	49%	66%	-

Table 2: Overall results of the different selection strategies playing against each other over all boards.

6×6 Board	UCBIT	UCT	DUCT(max)	DUCBIT(max)	DUCT(mix)	DUCBIT(mix)	Exp3	RM
UCT	-	43%	50%	49%	50%	48%	50%	49%
UCBIT	57%	-	50%	51%	50%	50%	51%	50%
DUCT(max)	50%	50%	-	54%	50%	52%	51%	50%
DUCT(mix)	51%	49%	46%	-	46%	47%	49%	48%
DUCBIT(max)	50%	50%	50%	54%	-	51%	51%	50%
DUCBIT(mix)	52%	50%	48%	53%	49%	-	51%	49%
Exp3	50%	49%	49%	51%	49%	49%	-	38%
RM	51%	50%	50%	52%	50%	51%	62%	-

Table 3: Results of the different selection strategies playing against each other on a 6 × 6 board.

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. Parameter  $T$  and  $\gamma \in [0, 1]$  were tuned similarly. The tuned values can be seen in Figure 3.

## 5.1 Round-Robin Tournaments

In this subsection, the performance of several players using different selection and update strategies are compared. This is done by playing matchups (of 500 games) of each player type against every other player type. Table 1 presents the results of the games on all four maps and Table 1 presents the average performance of all players. Figure 4 presents the average performance of each agent over all maps.

From this, we see that the algorithms that use UCB1-Tuned perform best overall, with the decoupled version winning significantly (8%) more often than its next three competitors. Given that the top three players use UCB1-Tuned, including DUCB1T(mix), it appears that a smarter way of performing exploration has a bigger impact on performance than the choice of the game model (sequential versus stacked matrix). The relative rank of DUCB1T(max), DUCT(max), and Exp3 presented here on board (d) are consistent with previous results on the open map [13].

As in Goofspiel [11], Regret Matching outperforms Exp3 and DUCT(mix). However unlike previous results in Goofspiel, Exp3 does not outperform DUCT(max). This may be because in Tron mistakes are not forgiving in the sense that it is easy to recognize and exploit suboptimal moves made by the opponent. Also, results of the algorithms vary from board to board, which is consistent with previous experiments in Tron [7]. Board (b), for instance, can lead to many situations where a mistake made is particular difficult to recover from. In this situation, the deterministic strategies tend to perform better since they avoid mistakes.

The same round-robin tournament was repeated on a smaller ( $6 \times 6$ ) board. From the results in Table 3, the round-robin tournament on the smaller  $6 \times 6$  board the performance of the deterministic strategies decrease and the performance of the stochastic strategies increase. This could be due to the fact that as both players come closer together, using a stochastic strategy might be more suitable than a deterministic one. For example, the in Figure 4 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. The optimal strategy for both players is to play with a mixed strategy, where each action is chosen with probability 0.5.

Selection Strategy	a	b	c	d	Total
DUCB1T(max)	65%	62%	62%	59%	$62.32 \pm 0.56\%$
DUCB1T(mix)	56%	57%	53%	53%	$54.82 \pm 0.61\%$
UCB1T	58%	57%	49%	54%	$54.32 \pm 0.55\%$
RM	56%	52%	51%	53%	$53.13 \pm 0.62\%$
UCT	47%	54%	55%	49%	$51.39 \pm 0.55\%$
DUCT(max)	43%	54%	49%	50%	$49.05 \pm 0.61\%$
DUCT(mix)	39%	40%	43%	36%	$39.51 \pm 0.64\%$
Exp3	35%	24%	38%	45%	$35.47 \pm 0.61\%$

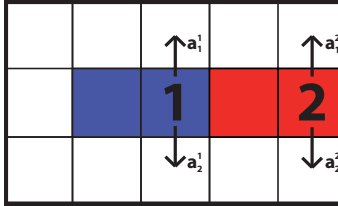


Figure 4: Left: Results of the different selection strategies playing against each other. a,b,c and d stand for the four different boards, which are used in the round-robin tournament.  $\pm$  refers to 95% confidence intervals. Right: A position whose optimal strategy is a mixed distribution.

## 6 Conclusion and Future Research

In this paper, several selection and update strategies were introduced for MCTS in Tron, including sequential UCT, UCB1-Tuned, DUCT(max) and DUCB1-Tuned(max), DUCT(mix), DUCB1-Tuned(mix), Exp3 and Regret Matching. The performance of these variants were tested in the game of Tron on different boards. 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 performed better, but also that the performance of stochastic strategies increases as the board gets smaller. The experiments also suggest that the layout of the board can influence the outcome of the search.

For future research, we aim to do more experiments with different boards. In addition, the selection strategy Exp3 can be enhanced by tuning parameter  $\eta$ . Moreover, a hybrid selection strategy could 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.

## 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.
- [2] Branislav Bošanský, Viliam Lisý, Jiří Čermák, Roman Vitek, and Michal Pěchouček. Using double-oracle method and serialized alpha-beta search for pruning in simultaneous moves. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI)*, 2013.
- [3] 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.
- [4] G.M.J-B. Chaslot. *Monte-Carlo Tree Search*. PhD thesis, Department of Knowledge Engineering, Maastricht University, Netherlands, 2010. Ph.D. dissertation.
- [5] 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.
- [6] R. Coulom. Efficient selectivity and backup operators in Monte Carlo Tree Search. In *CG 2008*, volume 4630 of *LNCS*, pages 72–83, 2007.
- [7] 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.
- [8] H. Finnsson. Cadia-player: A general game playing agent. Master thesis, Reykjavik University, Iceland - School of Computer Science, December 2007.
- [9] S. Hart and A. Mas-Colell. A simple adaptive procedure leading to correlated equilibrium. *Econometrica*, 68(5):1127–1150, 2000.
- [10] L. Kocsis and C. Szepesvári. Bandit based Monte Carlo planning. volume 5131 of *LNCS*, pages 282–293, 2006.
- [11] 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.
- [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] 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).
- [15] M. Shafiei, N. R. Sturtevant, and J. Schaeffer. Comparing UCT versus CFR in simultaneous games. In *Proceedings of the IJCAI Workshop on General Game-Playing (GIGA)*, pages 75–82, 2009.