Comparison of Different Selection Strategies in Monte-Carlo Tree Search for the Game of Tron

Pierre Perick, David L. St-Pierre, Francis Maes and Damien Ernst

***Abstract*—Monte-Carlo Tree Search (MCTS) techniques are essentially known for their performance on turn-based games, such as Go, for which players have considerable time for choosing their moves. In this paper, we apply MCTS to the game of Tron, a simultaneous real-time two-player game. The fact that players have to react fast and that moves occur simultaneously creates an unusual setting for MCTS, in which classical selection policies such as *UCB1* may be suboptimal. In this paper, we perform an empirical comparison of a wide range of selection policies for MCTS applied to Tron, with both deterministic policies (*UCB1*, *UCB1-Tuned*, UCB-V, *UCB-Minimal*, *OMC-Deterministic*, *MOSS*) and stochastic policies (ϵ_n-*greedy*, *EXP3*, *Thompson Sampling*, *OMC-Stochastic*, *PBBM*). From the experiments, we observe that *UCB1-Tuned* has the best behavior shortly followed by *UCB1*. Even if *UCB-Minimal* is ranked fourth, this is a remarkable result for this recently introduced selection policy found through automatic discovery of good policies on generic multi-armed bandit problems. We also show that deterministic policies perform better than stochastic ones for this problem.**

I. Introduction

Games provide a popular and challenging platform for research in Artificial Intelligence (AI). Traditionally, the wide majority of work in this field focuses on turn-based deterministic games such as Checkers [1], Chess [2] and Go [3]. These games are characterized by the availability of a long thinking time (e.g. several minutes), making it possible to develop large game trees before deciding which move to execute. Among the techniques to develop such game trees, Monte-Carlo tree search (MCTS) is probably the most important breakthrough of the last decade. This approach, which combines the precision of tree-search with the generality of random simulations, has shown spectacular successes in computer Go [4] and is now a method of choice for General Game Playing (GGP) [5].

In recent years, the field has seen a growing interest for real-time games such as Tron [6] and Miss Pac-Man [7], which typically involve short thinking times (e.g. 100 ms per turn). Due to the real-time constraint, MCTS algorithms can only make a limited number of game simulations, which is typically several orders of magnitude less than the number of simulations used in Go. In addition to the real-time constraint, real-time video games are usually characterized by uncertainty, massive branching factors, simultaneous moves and open-endedness. In this paper, we focus on the game Tron, for which simultaneous moves play a crucial role.

Pierre Perick, David L. St-Pierre, Francis Maes and Damien Ernst are in the Department of Electrical Engineering and Computer Science of Liège University, Montefiore Institute, B28, B-4000 Liège, Belgium (e-mail:pierre.perick@student.ulg.ac.be, {dlspierre, francis.maes, dernst}@ulg.ac.be).

Applying MCTS to Tron was first proposed in [6], where the authors apply the generic Upper Confidence bounds applied to Trees (UCT) algorithm to play this game. In [8], several heuristics specifically designed for Tron are proposed to improve upon the generic UCT algorithm. In both cases, the authors rely on the original UCT algorithm that was designed for turn-based games. The simultaneous property of the game is simply ignored. They use the algorithm as if players would take turn to play. It is shown in [8] that this approximation generates artefacts, especially during the last turns of a game. To reduce these artefacts, the authors propose a different way of computing the set of valid moves, while still relying on the turn-based UCT algorithm.

In this paper, we focus on variants of MCTS that explicitly take simultaneous moves into account by only considering joint moves of both players. Adapting *UCT* in this way has first been proposed by [9], with an illustration of the approach on Rock-paper-scissors, a simple one-step simultaneous two-player game. Recently, the authors of [10] proposed to use a stochastic selection policy specifically designed for simultaneous two-player games: *EXP3*. They show that this stochastic selection policy enables to outperform *UCT* on Urban Rivals, a partially observable internet card game.

The combination of simultaneous moves and short thinking time creates a unusual setting for MCTS algorithms and has received little attention so far. On one side, treating moves as simultaneous increases the branching factor and, on the other side, the short thinking time limits the number of simulations that can be performed during one turn. Algorithms such as *UCT* rely on a multi-armed bandit policy to select which simulations to draw next. Traditional policies (e.g. *UCB1*) have been designed to reach good asymptotic behavior [11]. In our case, since the ratio between the number of simulations and the number of arms is relatively low, we may be far from reaching this asymptotic regime, which makes it legitimate to wonder how other selection policies would behave in this particular setting.

This paper provides an extensive comparison of selection policies for MCTS applied to the simultaneous two-player real-time game Tron. We consider six deterministic selection policies (*UCB1*, *UCB1-Tuned*, UCB-V, *UCB-Minimal*, *OMC-Deterministic* and *MOSS*) and six stochastic selection policies (ϵ_n-*greedy*, *EXP3*, *Thompson Sampling*, *OMC-Stochastic*, *PBBM* and *Random*). While some of these policies have already been proposed for Tron (*UCB1*, *UCB1-Tuned*), for MCTS (*OMC-Deterministic*, *OMC-Stochastic*, *PBBM*) or for simultaneous two-player games (ϵ_n-*greedy*, *EXP3*), we also introduce four policies that, to the knowledge

978-1-4673-1194-6/12/$31.00 ©2012 IEEE

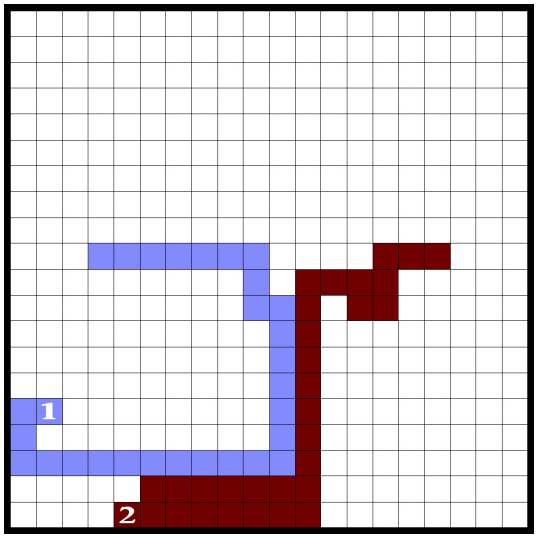

Figure 1. Illustration of the game of Tron on a 20×20 board.

of the authors, have not been tried yet in combination with MCTS: *UCB-Minimal* is a recently introduced policy that was found through automatic discovery of good policies on multi-armed bandit problems [12], UCB-V is a policy that uses the estimated variance to obtain tighter upper bounds [13], *Thompson Sampling* is a stochastic policy that has recently been shown to behave very well on multi-armed bandit problems [14] and *MOSS* is a deterministic policy that modifies the upper confidence bound of the *UCB1* policy.

The outline of this paper is as follows. Section II first presents a brief description of the game of Tron. Section III describes MCTS and details how we adapted MCTS to treat simultaneous move. Section IV describes the twelve selection policies that we considered in our comparison. Section V shows obtained results and, finally, the conclusion and an outlook of future search are covered in Section VI.

II. The game of *Tron*

This section introduces the game Tron, discusses its complexity and reviews previous AI work for this game.

A. Game description

The Steven Lisberger's film *Tron* was released in 1982 and features a Snake-like game. This game, illustrated in Figure 1, occurs in a virtual world where two motorcycles move at constant speed making only right angle turns. The two motorcycles leave solid wall trails behind them that progressively fill the arena, until one player or both crashes into one of them.

Tron is played on a $N \times M$ grid of cells in which each cell can either be *empty* or *occupied*. Commonly, this grid is a square, i.e. $N = M$. At each time step, both players move simultaneously and can only *(a)* continue straight ahead, *(b)* turn right or *(c)* turn left. A player cannot stop moving, each move is typically very fast (e.g. 100 ms per step) and the game is usually short. The goal is to survive his opponent until he crashes into a wall. The game can finish in a draw if the two players move at the same position or if they both crash at the same time. The main strategy consists in attempting to reduce the movement space of the opponent. For example, in the situation depicted in Figure 1, player 1 has a bigger share of the board and will probably win.

Tron is a finite-length game: the number of steps is upper bounded by $\frac{N \times M}{2}$. In practice, the number of moves in a game is often much lower since one of the player can usually quickly confine his opponent within a small area, leading to a quick end of the game.

Tron became a popular game implemented in a lot of variants. A well-known variant is the game *"Achtung, die kurve!"*, that includes bonuses (lower and faster speed, passing through the wall, *etc.*) and curve movements.

B. Game complexity

Several ways of measuring game complexity have been proposed and studied in game theory, among which game tree size, game-tree complexity and computational complexity [15]. We discuss here the game-tree complexity of Tron. Since moves occur simultaneously, each possible pair of moves must be considered when developing a node in the game tree. Given that agents have three possible moves (go straight, turn right and turn left), there exists 3^2 pairs of moves for each state, hence the branching factor of the game tree is 9.

We can estimate the mean game-tree complexity by raising the branching factor to the power of the mean length of games. It is shown in [16], that the following formula is a reasonable approximation of the average length of the game:

$$a = \frac{N^2}{1 + \log_2 N}$$

for a symmetric game $N = M$. In this paper, we consider 20×20 boards and have $a \simeq 75$. Using this formula, we obtain that the average tree-complexity for Tron on a 20×20 board is $\mathcal{O}(10^{71})$. If we compare 20×20 and 32×32 Tron to some well-known games, we obtain the following ranking:

$$Draughts(10^{54}) < Tron_{20\times20}(10^{71}) < Chess(10^{123}) < Tron_{32\times32}(10^{162}) < Go_{19\times19}(10^{360})$$

Tron has been studied in graph and game complexity theory and has been proven to be *PSPACE*-complete, *i.e.* to be a decision problem which can be solved by a Turing machine using a polynomial amount of space [17], [18], [19].

C. Previous work

Different techniques have been investigated to build agents for Tron. The authors of [20], [21] introduced a framework based on evolutionary algorithms and interaction with human players. At the core of their approach is an Internet server that enables to perform agent vs. human games to construct the fitness function used in the evolutionary algorithm. In the same spirit, [22] proposed to train a neural-network based

agent by using human data. Turn-based MCTS has been introduced in the context of Tron in [6] and [16] and further developed with domain-specific heuristics in [8].

Tron was used in the *2010 Google AI Challenge*, organised by the University of Waterloo Computer Science Club. The aim of this challenge was to develop the best agent to play the game using any techniques in a wide range of possible programming languages. The winner of this challenge was Andy Sloane who implemented an Alpha-Beta algorithm with an evaluation function based on the tree of chambers heuristic[1].

III. Simultaneous Monte-Carlo Tree Search

This section introduces the variant of MCTS that we use to treat simultaneous moves. We start with a brief description of the classical MCTS algorithm.

A. Monte-Carlo Tree Search

Monte-Carlo Tree Search is a best-first search algorithm that relies on random simulations to estimate position values. MCTS collects the results of these random simulations in a game tree that is incrementally grown in an asymmetric way that favors exploration of the most promising sequences of moves. This algorithm appeared in scientific literature in 2006 in three different variants [23], [24], [4] and led to breakthrough results in computer Go. Thanks to the generality of random simulations, MCTS can be applied to a wide range of problems without requiring any prior knowledge or domain-specific heuristics. Hence, it became a method of choice in General Game Playing.

The central data structure in MCTS is the game tree in which nodes correspond to game states and edges correspond to possible moves. The role of this tree is two-fold: it stores the outcomes of random simulations and it is used to bias random simulations towards promising sequences of moves.

MCTS is divided in four main steps that are repeated until the time is up [25]:

1) Selection: This step aims at selecting a node in the tree from which a new random simulation will be performed.

2) Expansion: If the selected node does not end the game, this steps adds a new leaf node to the selected one and selects this new node.

3) Simulation: This step starts from the state associated to the selected leaf node, executes random moves in self-play until the end of the game and returns the following reward: 1 for a victory, 0 for a defeat or 0.5 for a draw. The use of an adequate simulation strategy can improve the level of play [26].

4) Backpropagation: The *backpropagation* step consists in propagating the result of the simulation backwards from the leaf node to the root.

The main focus of this paper is on the *selection* step. The way this step is performed is essential since it determines in which way the tree is grown and how the computational budget is allocated to random simulations. It has to deal with the exploration/exploitation dilemma: *exploration* consists in trying new sequences of moves to increase knowledge and *exploitation* consists in using current knowledge to bias computational efforts towards *promising* sequences of moves.

When the computational budget is exhausted, one of the moves is selected based on the information collected from simulations and contained in the game tree. In this paper, we use the strategy called *robust child*, which consists in choosing the move that has been most simulated.

[1]http://a1k0n.net/2010/03/04/google-ai-postmortem.html

Algorithm 1 Simultaneous two-players selection procedure

Require: The root node $\mathbf{n}_0 \in \mathcal{N}$
Require: The selection policy $\pi(\cdot) \in \mathcal{M}$
  $\mathbf{n} \leftarrow \mathbf{n}_0$
  while $\mathbf{n}$ is not a leaf **do**
    $\alpha \leftarrow \pi(P^{agent}(\mathbf{n}))$
    $\beta \leftarrow \pi(P^{opponent}(\mathbf{n}))$
    $\mathbf{n} \leftarrow child(\mathbf{n}, (\alpha, \beta))$
  end while
  return $\mathbf{n}$

B. Simultaneous moves

In order to properly account for simultaneous moves, we follow a strategy similar to the one proposed in [9], [10]: instead of selecting a move for the agent, updating the game state and then selecting an action for its opponent, we select both actions simultaneously and independently and then update the state of the game. Since we treat both moves simultaneously, edges in the game tree are associated to pairs of moves (α, β) where α denotes the move selected by the agent and β denotes the move selected by its opponent.

Let $\mathcal{N}$ be the set of nodes in the game tree and $\mathbf{n}_0 \in \mathcal{N}$ be the root node of the game tree. Our selection step is detailed in Algorithm 1. It works by traversing the tree recursively from the root node $\mathbf{n}_0$ to a leaf node $\mathbf{n} \in \mathcal{N}$. We denote by $\mathcal{M}$ the set of possible moves. Each step of this traversal involves selecting moves $\alpha \in \mathcal{M}$ and $\beta \in \mathcal{M}$ and moving into the corresponding child node, denoted $child(\mathbf{n}, (\alpha, \beta))$. The selection of a move is done in two steps: first, a set of statistics $P^{player}(\mathbf{n})$ is extracted from the game tree to describe the selection problem and then, a *selection policy* π is invoked to choose the move given this information. The remainder of this section details these two steps.

For each node $\mathbf{n} \in \mathcal{N}$, we store the following quantities:

- $t(\mathbf{n})$ is the number of simulations involving node $\mathbf{n}$, which is known as the *visit count* of node $\mathbf{n}$.
- $\overline{r}(\mathbf{n})$ is the empirical mean of the rewards the agent obtained from these simulations. Note that because it is a one-sum game, the average reward for the opponent is $1 - \overline{r}(\mathbf{n})$.
- $\overline{\sigma}(\mathbf{n})$ is the empirical standard deviation of the rewards (which is the same for both players).

Let $L^{agent}(\mathbf{n}) \subset \mathcal{M}$ (resp. $L^{opponent}(\mathbf{n}) \subset \mathcal{M}$) be the set of legal moves for the agent (resp. the opponent) in the game state represented by node $\mathbf{n}$. In the case of Tron, legal

moves are those that do not lead to an immediate crash: e.g. turning into an already existing wall is not a legal move[2].

Let $player \in \{agent, opponent\}$. The function $P^{player}(\cdot)$ computes a vector of statistics $S = (m_1, \overline{r}_1, \overline{\sigma}_1, t_1, \ldots, m_K, \overline{r}_K, \overline{\sigma}_K, t_K)$ describing the selection problem from the point of view of $player$. In this vector, $\{m_1, \ldots, m_K\} = L^{player}(\mathbf{n})$ is the set of valid moves for the player and $\forall k \in [1, K]$, $\overline{r}_k$, $\overline{\sigma}_k$ and t_k are statistics relative to the move m_k. We here describe the statistics computation in the case of $P^{agent}(\cdot)$. Let $C(\mathbf{n}, \alpha)$ be the set of child nodes whose first action is α, i.e. $C(\mathbf{n}, \alpha) = \{child(\mathbf{n}, \alpha, \beta) | \beta \in L^{opponent}(\mathbf{n})\}$. For each legal move $m_k \in L^{agent}(\mathbf{n})$, we compute:

$$t_k = \sum_{\mathbf{n}' \in C(\mathbf{n}, m_k)} t(\mathbf{n}'),$$
$$\overline{r}_k = \frac{\sum_{\mathbf{n}' \in C(\mathbf{n}, m_k)} t(\mathbf{n}') \overline{r}(\mathbf{n}')}{t_k},$$
$$\overline{\sigma}_k = \frac{\sum_{\mathbf{n}' \in C(\mathbf{n}, m_k)} t(\mathbf{n}') \overline{\sigma}(\mathbf{n}')}{t_k}.$$

$P^{opponent}(\cdot)$ is simply obtained by taking the symmetric definition of C: i.e. $C(\mathbf{n}, \beta) = \{child(\mathbf{n}, \alpha, \beta) | \alpha \in L^{player}(\mathbf{n})\}$.

The selection policy $\pi(\cdot) \in \mathcal{M}$ is an algorithm that selects a move $m_k \in \{m_1, \ldots, m_K\}$ given the vector of statistics $S = (m_1, \overline{r}_1, \overline{\sigma}_1, t_1, \ldots, m_K, \overline{r}_K, \overline{\sigma}_K, t_K)$. Selection policies are the topic of the next section.

IV. Selection policies

This section describes the twelve selection policies that we use in our comparison. We first describe *deterministic* selection policies and then move on *stochastic* selection policies.

A. Deterministic selection policies

We consider deterministic selection policies that belong to the class of index-based multi-armed bandit policies. These policies work by assigning an index to each candidate move and by selecting the move with maximal index: $\pi^{deterministic}(S) = m_{k^*}$ with

$$k^* = \underset{k \in [1,K]}{\operatorname{argmax}}\, index(t_k, \overline{r}_k, \overline{\sigma}_k, t)$$

where $t = \sum_{k=1}^{K} t_k$ and $index$ is called the *index function*. Index functions typically combine an exploration term to favor moves that we already know perform well with an exploitation term that aims at selecting less-played moves that may potentially reveal interesting. Several index-policies have been proposed and they vary in the way they define these two terms.

[2]If the set of legal moves is empty for one of the players, this player looses the game.

1) UCB1: The index function of *UCB1* [11] is:

$$index(t_k, \overline{r}_k, \overline{\sigma}_k, t) = \overline{r}_k + \sqrt{C \frac{\ln t}{t_k}},$$

where $C > 0$ is a parameter that enables to control the exploration/exploitation trade-off. Although the theory suggest a default value of $C = 2$, this parameter is usually experimentally tuned to increase performance.

UCB1 has appeared the first time in the literature in 2002 and is probably the best known index-based policy for multi-armed bandit problem [11]. It has been popularized in the context of MCTS with the Upper confidence bounds applied to Trees (UCT) algorithm [23], which is the instance of MCTS using *UCB1* as selection policy.

2) UCB1-Tuned: In their seminal paper, the authors of [11] introduced another index-based policy called *UCB1-Tuned*, which has the following index function:

$$index(t_k, \overline{r}_k, \overline{\sigma}_k, t) = \overline{r}_k + \sqrt{\frac{min\{\frac{1}{4}, V(t_k, \overline{\sigma}_k, t)\} \ln t}{t_k}},$$

where

$$V(t_k, \overline{\sigma}_k, t) = \overline{\sigma}_k^2 + \sqrt{\frac{2 \ln t}{t_k}}.$$

UCB1-Tuned relies on the idea to take empirical standard deviations of the rewards into account to obtain a refined upper bound on rewards expectation. It is analog to *UCB1* where the parameter C has been replaced by a smart upper bound on the variance of the rewards, which is either $\frac{1}{4}$ (an upper bound of the variance of *Bernouilli* random variable) or $V(t_k, \overline{\sigma}_k, t)$ (an upper confidence bound computed from samples observed so far).

Using *UCB1-Tuned* in the context of MCTS for Tron has already been proposed by [6]. This policy was shown to behave better than *UCB1* on multi-armed bandit problems with *Bernouilli* reward distributions, a setting close to ours.

3) UCB-V*:* The index-based policy UCB-V [13] uses the following index formula:

$$index(t_k, \overline{r}_k, \overline{\sigma}_k, t) = \overline{r}_k + \sqrt{2 \frac{\overline{\sigma}_k^2 \zeta \ln t}{t_k}} + c \frac{3 \zeta \ln t}{t_k}.$$

UCB-V has two parameters $\zeta > 0$ and $c > 0$. We refer the reader to [13] for detailed explanations of these parameters.

UCB-V is a less tried multi-armed bandit policy in the context of MCTS. As *UCB1-Tuned*, this policy relies on the variance of observed rewards to compute tight upper bound on rewards expectation.

4) UCB-Minimal: Starting from the observation that many different similar index formulas have been proposed in the multi-armed bandit literature, it was recently proposed in [12], [27] to explore the space of possible index formulas in a systematic way to discover new high-performance bandit policies. The proposed approach first defines a grammar made of basic elements (mathematical operators, constants and variables such as $\overline{r}_k$ and t_k) and generates a large set of candidate formulas from this grammar. The systematic search

for good candidate formulas is then carried out by a built-on-purpose optimization algorithm used to navigate inside this large set of candidate formulas towards those that give high performance on generic multi-armed bandit problems. As a result of this automatic discovery approach, it was found that the following simple policy behaved very well on several different generic multi-armed bandit problems:

$$index(t_k, \overline{r}_k, \overline{\sigma}_k, t) = \overline{r}_k + \frac{C}{t_k},$$

where $C > 0$ is a parameter to control the exploration/exploitation tradeoff. This policy corresponds to the simplest form of UCB-style policies. In this paper, we consider a slightly more general formula that we call *UCB-Minimal*:

$$index(t_k, \overline{r}_k, \overline{\sigma}_k, t) = \overline{r}_k + \frac{C_1}{t_k^{C_2}},$$

where the new parameter C_2 enables to fine-tune the decrease rate of the exploration term.

5) OMC-Deterministic: The Objective Monte-Carlo (*OMC*) selection policy exists in two variants: stochastic (*OMC-Stochastic*) [24] and deterministic (*OMC-Deterministic*) [28]. The index-based policy for *OMC-Deterministic* is computed in two steps. First, a value U_k is computed for each move k:

$$U_k = \frac{2}{\sqrt{\pi}} \int_{\alpha}^{\infty} e^{-u^2} du,$$

where α is given by:

$$\alpha = \frac{v_0 - (\overline{r}_k t_k)}{\sqrt{2}\overline{\sigma}_k},$$

and where

$$v_0 = \max(\overline{r}_i t_i) \ \forall i \in [1, K].$$

After that, the following index formula is used:

$$index(t_k, \overline{r}_k, \overline{\sigma}_k, t) = \frac{tU_k}{t_k \sum_{i=1}^{K} U_i}.$$

6) MOSS: Minimax Optimal Strategy in the Stochastic Case (*MOSS*) is an index-based policy proposed in [29] where the following index formula is introduced:

$$index(t_k, \overline{r}_k, \overline{\sigma}_k, t) = \overline{r}_k + \sqrt{\frac{\max\left(\log\left(\frac{t_k}{Kt}\right), 0\right)}{t}}.$$

This policy is inspired from the *UCB1* policy. The index of a move is the mean of rewards obtained from simulations if the move has been selected more than $\frac{t_k}{K}$. Otherwise, the index value is an upper confidence bound on the mean reward. This bound holds with a high probability according the Hoeffding's inequality. Similarly to *UCB1-Tuned*, this selection policy has no parameters to tune thus facilitating its use.

B. Stochastic selection policies

In the case of simultaneous two-player games, the opponent's moves are not immediately observable, and following the analysis of [10], it may be beneficial to also consider stochastic selection policies. Stochastic selection policies π are defined through a condition distribution $p_\pi(k|S)$ of moves given the vector of statistics S:

$$\pi^{stochastic}(S) = m_k, \quad k \sim p_\pi(\cdot|S).$$

We consider six stochastic policies:

1) Random: This baseline policy simply selects moves with uniform probabilities:

$$p_\pi(k|S) = \frac{1}{K}, \quad \forall k \in [1, K].$$

2) ϵ_n-greedy: The second baseline is ϵ_n-greedy [30]. This policy consists in selecting a random move with low probability ϵ_t or the empirical best move according to $\overline{r}_k$:

$$p_\pi(k|S) = \begin{cases} 1 - \epsilon_t & \text{if } k = \text{argmax}_{k \in [1,K]} \overline{r}_k \\ \epsilon_t / K & \text{otherwise.} \end{cases}$$

The amount of exploration ϵ_t is chosen to decrease with time. We adopt the scheme proposed in [11]:

$$\epsilon_t = \frac{c\ K}{d^2\ t},$$

where $c > 0$ and $d > 0$ are tunable parameters.

3) Thompson Sampling: Thompson Sampling adopts a Bayesian perspective by incrementally updating a belief state for the unknown reward expectations and by randomly selecting actions according to their probability of being optimal according to this belief state.

We consider here the variant of *Thompson Sampling* proposed in [14] in which the reward expectations are modeled using a beta distribution. The sampling procedure works as follows: it first draw a stochastic score

$$s(k) \sim beta\big(C_1 + \overline{r}_k t, C_2 + (1 - \overline{r}_k) t_k\big)$$

for each candidate move $k \in [1, K]$ and then selects the move maximizing this score:

$$p_\pi(k|S) = \begin{cases} 1 & \text{if } k = \text{argmax}_{k \in [1,K]}\, s(k) \\ 0 & \text{otherwise.} \end{cases}$$

$C_1 > 0$ and $C_2 > 0$ are two tunable parameters that reflect prior knowledge on reward expectations.

Thompson Sampling has recently been shown to perform very well on Bernouilli multi-armed bandit problems, in both context-free and contextual bandit settings [14]. The reason why *Thompson Sampling* is not very popular yet may be due to his lack of theoretical analysis. At this point, only the convergence has been proved [31].

4) EXP3: This stochastic policy is commonly used in simultaneous two-player games [32], [10], [33] and is proved to converge towards the Nash equilibrium asymptotically. *EXP3* works slightly differently from our other policies since it requires storing two additional vectors in each node $\mathbf{n} \in \mathcal{N}$ denoted $w^{agent}(\mathbf{n})$ and $w^{opponent}(\mathbf{n})$. These vectors contain one entry per possible move $m \in L^{player}$, are initialized to $w_k^{player}(\cdot) = 0, \forall k \in [1, K]$ and are updated each time a reward r is observed, according to the following formulas:

$$w_k^{agent}(\mathbf{n}) \leftarrow w_k^{agent}(\mathbf{n}) + \frac{r}{p_\pi(k|P^{agent}(\mathbf{n}))},$$
$$w_k^{opponent}(\mathbf{n}) \leftarrow w_k^{opponent}(\mathbf{n}) + \frac{1-r}{p_\pi(k|P^{opponent}(\mathbf{n}))}.$$

At any given time step, the probabilities to select a move are defined as:

$$p_\pi(k|S) = (1-\gamma)\frac{e^{\eta w_k^{player}(\mathbf{n})}}{\sum_{k' \in [1,K]} e^{\eta w_{k'}^{player}}} + \frac{\gamma}{K},$$

where $\eta > 0$ and $\gamma \in]0; 1]$ are two parameters to tune.

5) OMC-Stochastic: The *OMC-Stochastic* selection policy [24] uses the same U_k quantities than *OMC-Deterministic*. The stochastic version of this policy is defined as following:

$$p_\pi(k|S) = \frac{U_k}{\sum_{i=1}^{K} U_i} \ \forall k \in [1, K].$$

The design of this policy is based on the Central Limit Theorem and *EXP3*.

6) PBBM: Probability to be Better than Best Move (*PBBM*) is a selection policy [4] with a probability proportional to

$$p_\pi(k|S) = e^{-2.4\alpha} \ \forall k \in [1, K].$$

The α is computed as:

$$\alpha = \frac{v_0 - (\overline{r}_k t_k)}{\sqrt{2(\overline{\sigma}_0^2 + \overline{\sigma}_k^2)}},$$

where

$$v_0 = \max(\overline{r}_i t_i) \ \forall i \in [1, K],$$

and where $\overline{\sigma}_0^2$ is the variance of the reward for the move selected to compute v_0.

This selection policy was successfully used in Crazy Stone, a computer Go program [4]. The concept is to select the move according to its probability of being better than the current best move.

V. Experiments

In this section we compare the selection policies $\pi(\cdot)$ presented in Section IV on the game of Tron introduced previously in this paper. We start this section by first describing the strategy used for simulating the rest of the game when going beyond a terminal leaf of the tree (Section V-A). Afterwards, we will detail the procedure we adopted for tuning the parameters of the selection policies (Section V-B). And, finally, we will present the metric used for comparing the different policies and discuss the results that have been obtained (Section V-C).

A. Simulation heuristic

It has already been recognized for a long time that using pure random strategies for simulating the game beyond a terminal leaf node of the tree built by MCTS techniques is a suboptimal choice. Indeed, such a random strategy may lead to a game outcome that poorly reflects the quality of the selection procedure defined by the tree. This in turn requires to build large trees in order to compute high-performing moves. To define our simulation heuristic we have therefore decided to use prior knowledge on the problem. Here, we use a simple heuristic developed in [16] for the game on Tron that, even if still far from an optimal strategy, lead the two players to adopt a more rationale behaviour. This heuristic is based on a distribution probability $P_{move}(\cdot)$ over the moves that associates a probability of 0.68 to the "go straight ahead" move and a probability of 0.16 to each of the two other moves (turn left or right). Afterwards, moves are sequentially drawn from $P_{move}(\cdot)$ until a move that is legal and that does not lead to self-entrapment at the next time step is found. This move is the one selected by our simulation strategy.

To prove the efficiency of this heuristic, we performed a short experiment. We confronted two identical UCT opponents on $10\ 000$ rounds: one using the heuristic and the other making purely random simulations. The result of this experiment is that the agent with the heuristic has a winning percentage of $93.42 \pm 0.5\%$ in a 95% confidence interval.

Note that the performance of the selection policy depends on the simulation strategy used. Therefore, we cannot exclude that if a selection policy is found to behave better than another one for a given simulation strategy, it may actually behave worse for another one.

B. Tuning parameter

The selection policies have one or several parameters to tune. Our protocol to tune these parameters is rather simple and is the same for every selection policy.

First, we choose for the selection policy to tune reference parameters that are used to define our reference opponent. These reference parameters are chosen based on default values suggested in the literature. Afterwards, we discretize the parameter space of the selection policy and test for every element of this set the performance of the corresponding agent against the reference opponent. The element of the discretized space that leads to the highest performance is then used to define the constants.

To test the performance of an agent against our reference opponent, we used the following experimental protocol. First, we set the game map to 20×20 and the time between two recommendations to 100 ms on a $2.5Ghz$ processor. Afterwards we perform a sequence of rounds until we have 10,000 rounds that do not end by a draw. Finally, we set the performance of the agent to its percentage of winnings.

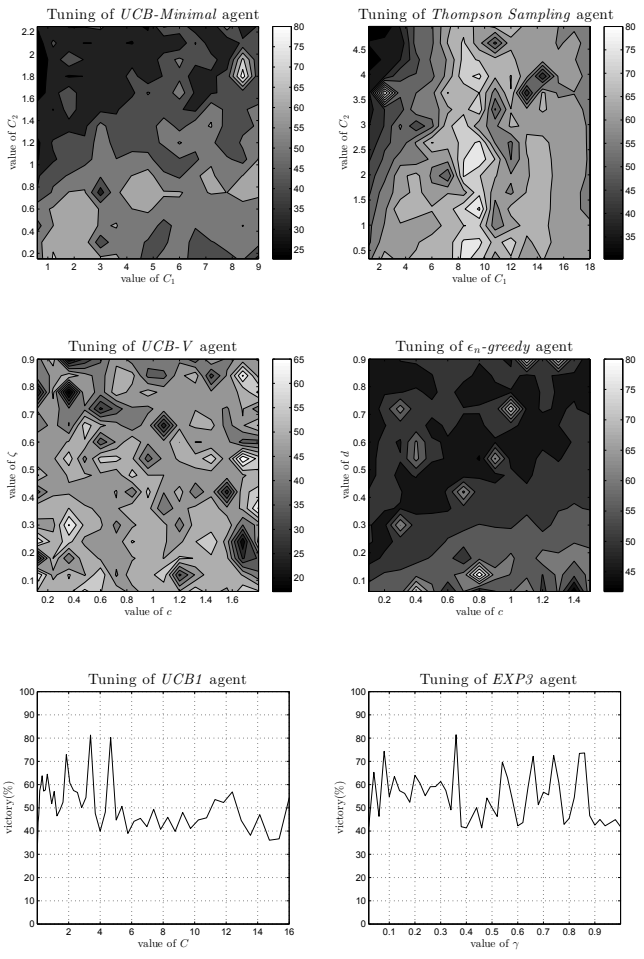

Figure 2. Tuning of constant for selection policies over 10 000 rounds. Clearer areas represent a higher winning percentage.

Figure 2 reports the performances obtained by the selection policies on rather large discretized parameter spaces. The best parameters found for every selection policy as well as the reference parameters are given in Table I. Some side simulations have shown that even by using a finer disretization of the parameter space, significantly better performing agents cannot not be found.

It should be stressed that the tuned parameters reported in this table point towards higher exploration rates than those usually suggested for other games, such as for example the game of Go. This is probably due to the low branching factor of the game of Tron.

C. Results

To compare the selection policies, we perform a round-robin to determine which one gives the best results. Table II presents the outcome of the experiments. In this double entry table, each data represents the victory ratio of the row selection policy against the column one. Results are expressed in percent $\pm$ a 95% confidence interval. The last column shows the average performance of the selection policies.

Table I
REFERENCE AND TUNED PARAMETERS FOR SELECTION POLICIES

Agent	*Reference constant*	*Tuned*
UCB1	$C = 2$	$C = 3.52$
UCB1-Tuned	–	–
UCB-V	$c = 1.0,\ \zeta = 1.0$	$c = 1.68,\ \zeta = 0.54$
UCB-Minimal	$C_1 = 2.5,\ C_2 = 1.0$	$C_1 = 8.40,\ C_2 = 1.80$
OMC-Deterministic	–	–
MOSS	–	–
Random	–	–
ϵ_n*-greedy*	$c = 1.0,\ d = 1.0$	$c = 0.8,\ d = 0.12$
Thompson Sampling	$C_1 = 1.0,\ C_2 = 1.0$	$C_1 = 9.6,\ C_2 = 1.32$
EXP3	$\gamma = 0.5$	$\gamma = 0.36$
OMC-Stochastic	–	–
PBBM	–	–

The main observations can be drawn from this table:

***UCB1-Tuned* is the winner**. The only policy that wins against all other policies is *UCB1-Tuned*. This is in line with what was reported in the literature, except perhaps with the result reported in [6] where the authors conclude that *UCB1-Tuned* performs slightly worse than *UCB1*. However, it should be stressed that in their experiments, they only perform 20 rounds to compare both algorithms, which is not enough to make a statistically significant comparison. Additionally, their comparison was not fair since they used for the *UCB1* policy a thinking time that was greater than for the *UCB1-Tuned* policy.

Stochastic policies are weaker than deterministic ones. Although using stochastic policies have some strong theoretical justifications in the context of simultaneous two-player games, we observe that our three best policies are deterministic. Whichever selection policy, we are probably far from reaching asymptotic conditions due to the real-time constraint. So, it may be the case that stochastic policies are preferable when a long thinking-time is available, but disadvantageous in the context of real-time games. Moreover, for the two variants of *OMC* selection policy, we show that the deterministic one outperforms the stochastic.

UCB-V performs worse. Surprisingly, UCB-V is the only deterministic policy that performs bad against stochastic policies. Since UCB-V is a variant of *UCB1-Tuned* and the latter performs well, we expected UCB-V to behave similarly yet it is not the case. From our experiments, we conclude that UCB-V is not an interesting selection policy for the game of Tron.

***UCB-Minimal* performs quite well**. Even if ranked fourth, *UCB-Minimal* gives average performances which are very close to those *UCB1* and *MOSS* ranked second and third, respectively. This is remarkable for a formula found automatically in the context of generic bandit problems. This suggests that an automatic discovery algorithm formula adapted to our specific problem may actually identify very good selection policies.

VI. CONCLUSION

We studied twelve different selection policies for MCTS applied to the game of Tron. Such a game is an unusual setting compared to more traditional testbeds because it is

Table II
PERCENTAGE OF VICTORY ON 10 000 ROUNDS BETWEEN SELECTION POLICIES

Selection policies \ Selection policies	*UCB1-Tuned*	*UCB1*	*MOSS*	*UCB-Minimal*	*EXP3*	*Thompson Sampling*	ϵ_n*-greedy*	*OMC-Deterministic*	UCB-V	*OMC-Stochastic*	*PBBM*	*Random*	*Average*
UCB1-Tuned	–	60.11 ± 0.98%	59.14 ± 0.98%	53.14 ± 1.00%	91.07 ± 0.57%	80.79 ± 0.80%	86.66 ± 0.68%	90.20 ± 0.60%	79.82 ± 0.80%	84.48 ± 0.72%	87.08 ± 0.67%	98.90 ± 0.21%	85.11 ± 0.71%
UCB1	39.89 ± 0.98%	–	55.66 ± 0.99%	35.76 ± 0.96%	84.36 ± 0.73%	85.57 ± 0.70%	81.18 ± 0.78%	94.02 ± 0.47%	81.02 ± 0.75%	91.24 ± 0.57%	87.38 ± 0.67%	99.47 ± 0.15%	75.69 ± 0.86%
MOSS	40.86 ± 0.98%	44.34 ± 0.99%	–	–	63.34 ± 0.96%	34.10 ± 0.95%	83.08 ± 0.75%	82.24 ± 0.76%	93.38 ± 0.50%	91.02 ± 0.57%	89.00 ± 0.63%	87.88 ± 0.65%	98.98 ± 0.21%	72.24 ± 0.90%
UCB-Minimal	46.86 ± 1.00%	64.24 ± 0.96%	36.66 ± 0.96%	–	80.79 ± 0.79%	85.27 ± 0.71%	82.15 ± 0.77%	88.12 ± 0.65%	87.71 ± 0.66%	32.64 ± 0.94%	89.82 ± 0.61%	99.37 ± 0.16%	70.40 ± 0.91%
EXP3	8.93 ± 0.57%	15.64 ± 0.73%	65.90 ± 0.95%	19.21 ± 0.79%	–	59.01 ± 0.98%	84.19 ± 0.73%	68.28 ± 0.93%	39.89 ± 0.98%	77.72 ± 0.83%	72.30 ± 0.90%	54.18 ± 0.99%	53.24 ± 0.99%
Thompson Sampling	19.21 ± 0.79%	14.43 ± 0.70%	16.92 ± 0.75%	24.73 ± 0.86%	40.99 ± 0.98%	–	62.40 ± 0.97%	69.08 ± 0.92%	49.68 ± 1.00%	84.62 ± 0.72%	83.42 ± 0.74%	95.80 ± 0.40%	50.80 ± 1.00%
ϵ_n*-greedy*	13.34 ± 0.68%	18.82 ± 0.79%	17.76 ± 0.76%	17.85 ± 0.77%	15.81 ± 0.73%	37.60 ± 0.97%	–	68.24 ± 0.93%	66.62 ± 0.94%	80.16 ± 0.80%	83.12 ± 0.75%	91.45 ± 0.56%	46.16 ± 1.00%
OMC-Deterministic	9.80 ± 0.60%	5.98 ± 0.47%	6.62 ± 0.50%	11.88 ± 0.65%	11.72 ± 0.93%	30.92 ± 0.92%	31.76 ± 0.73%	–	87.60 ± 0.66%	69.12 ± 0.92%	83.18 ± 0.75%	64.14 ± 0.96%	35.07 ± 0.95%
UCB-V	20.18 ± 0.80%	18.99 ± 0.78%	8.98 ± 0.57%	12.29 ± 0.66%	60.11 ± 0.98%	50.32 ± 1.00%	33.38 ± 0.94%	12.40 ± 0.66%	–	39.16 ± 0.98%	46.02 ± 0.99%	65.60 ± 0.95%	34.43 ± 0.95%
OMC-Stochastic	15.52 ± 0.72%	8.76 ± 0.57%	11.00 ± 0.63%	67.36 ± 0.94%	22.28 ± 0.83%	15.38 ± 0.72%	19.84 ± 0.80%	30.88 ± 0.92%	60.84 ± 0.98%	–	60.04 ± 0.98%	52.50 ± 1.00%	31.72 ± 0.93%
PBBM	12.92 ± 0.67%	12.62 ± 0.66%	12.12 ± 0.65%	10.18 ± 0.61%	27.70 ± 0.90%	16.58 ± 0.74%	16.88 ± 0.75%	16.82 ± 0.75%	53.98 ± 0.99%	39.96 ± 0.98%	–	52.76 ± 1.00%	23.98 ± 0.85%
Random	1.10 ± 0.21%	0.53 ± 0.15%	1.02 ± 0.21%	0.63 ± 0.16%	45.82 ± 0.99%	4.20 ± 0.40%	8.55 ± 0.56%	35.86 ± 0.96%	34.40 ± 0.95%	47.50 ± 1.00%	47.24 ± 1.00%	–	19.09 ± 0.79%

a fast-paced real-time simultaneous two-player game. There is no possibility of long thinking-time or to develop large game trees before choosing a move and the total number of simulations is typically small.

We performed an extensive comparison of selection policies for this unusual setting. Overall the results showed a stronger performance for the deterministic policies (*UCB1*, *UCB1-Tuned*, UCB-V, *UCB-Minimal*, *OMC-Deterministic* and *MOSS*) than for the stochastic ones (ϵ_n*-greedy*, *EXP3*, *Thompson Sampling*, *OMC-Stochastic* and *PBBM*). More specifically, from the results we conclude that *UCB1-Tuned* is the strongest selection policy, which is in line with the current literature. It was closely followed by the recently introduced *MOSS* and *UCB-Minimal* policies.

The next step in this research is to broaden the scope of the comparison by adding other real-time testbeds that possess a higher branching factor to further increase our understanding of the behavior of these selection policies.

REFERENCES

[1] A. Samuel, "Some Studies in Machine Learning using the Game of Checkers," *IBM Journal of research and development*, vol. 44, no. 1.2, pp. 206–226, 2000.

[2] M. Campbell, A. Hoane, and F. Hsu, "Deep Blue," *Artificial intelligence*, vol. 134, no. 1, pp. 57–83, 2002.

[3] M. Müller, "Computer Go," *Artificial Intelligence*, vol. 134, no. 1, pp. 145–179, 2002.

[4] R. Coulom, "Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search," *Computers and Games*, pp. 72–83, 2007.

[5] Y. Björnsson and H. Finnsson, "CadiaPlayer: A Simulation-Based General Game Player," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 1, no. 1, pp. 4–15, 2009.

[6] S. Samothrakis, D. Robles, and S. Lucas, "A UCT Agent for Tron: Initial Investigations," in *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, 2010, pp. 365–371.

[7] S. Lucas, "Evolving a Neural Network Location Evaluator to Play Ms. Pac-Man," in *Proceedings of the IEEE Symposium on Computational Intelligence and Games*. Citeseer, 2005, pp. 203–210.

[8] N. Den Teuling, "Monte-Carlo Tree Search for the Simultaneous Move Game Tron," *Univ. Maastricht, Netherlands, Tech. Rep*, 2011.

[9] M. Shafiei, N. Sturtevant, and J. Schaeffer, "Comparing UCT versus CFR in Simultaneous Games," in *Proceedings of the General Game Playing workshop at IJCAI'09*, 2009.

[10] O. Teytaud and S. Flory, "Upper Confidence Trees with Short Term Partial Information," in *Proceedings of the 2011 international conference on Applications of evolutionary computation - Volume Part I*, ser. EvoApplications'11, 2011, pp. 153–162.

[11] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time Analysis of the Multiarmed Bandit Problem," *Machine learning*, vol. 47, no. 2, pp. 235–256, 2002.

[12] F. Maes, L. Wehenkel, and D. Ernst, "Automatic discovery of ranking formulas for playing with multi-armed bandits," in *9th European workshop on reinforcement learning (EWRL'11)*, Athens, Greece, September 2011.

[13] J. Audibert, R. Munos, and C. Szepesvári, "Tuning bandit algorithms in stochastic environments," in *Algorithmic Learning Theory*. Springer, 2007, pp. 150–165.

[14] O. Chapelle and L. Li, "An Empirical Evaluation of Thompson Sampling," in *Neural Information Processing Systems (NIPS)*, 2011.

[15] L. Allis *et al.*, *Searching for Solutions in Games and Artificial Intelligence*. Ponsen & Looijen, 1994.

[16] B. Saverino, "A Monte-Carlo Tree Search for playing Tron," Master's thesis, Montefiore, Department of Electrical Engineering and Computer Science, Université de Liège, 2011.

[17] H. Bodlaender, "Complexity of Path-Forming Games," *RUU-CS*, no. 89-29, 1989.

[18] H. Bodlaender and A. Kloks, "Fast Algorithms for the Tron Game on Trees," *RUU-CS*, no. 90-11, 1990.

[19] T. Miltzow, "Tron, a combinatorial Game on abstract Graphs," *Arxiv preprint arXiv:1110.3211*, 2011.

[20] P. Funes, E. Sklar, H. Juillé, and J. Pollack, "The Internet as a Virtual Ecology: Coevolutionary Arms Races Between Human and Artificial Populations," *Computer Science Technical Report CS-97-197, Brandeis University*, 1997.

[21] ——, "Animal-Animat Coevolution: Using the Animal Population as Fitness Function," *From Animals to Animats 5: Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior*, vol. 5, pp. 525–533, 1998.

[22] A. Blair, E. Sklar, and P. Funes, "Co-evolution, Determinism and Robustness," *Simulated Evolution and Learning*, pp. 389–396, 1999.

[23] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo Planning," *Machine Learning: ECML 2006*, pp. 282–293, 2006.

[24] G. Chaslot, J. Saito, B. Bouzy, J. Uiterwijk, and H. Van Den Herik, "Monte-Carlo Strategies for Computer Go," in *Proceedings of the 18th BeNeLux Conference on Artificial Intelligence, Namur, Belgium*, 2006, pp. 83–91.

[25] G. Chaslot, "Monte-Carlo Tree Search," Ph.D. dissertation, Ph. D. thesis, Department of Knowledge Engineering, Maastricht University, Maastricht, The Netherlands.[19, 20, 22, 31], 2010.

[26] S. Gelly, Y. Wang, R. Munos, and O. Teytaud, "Modification of UCT with Patterns in Monte-Carlo Go," INRIA, Research Report, 2006.

[27] F. Maes, L. Wehenkel, and D. Ernst, "Learning to play K-armed bandit problems," in *International Conference on Agents and Artificial Intelligence*, Vilamoura, Algarve, Portugal, February 2012.

[28] G. Chaslot, S. De Jong, J. Saito, and J. Uiterwijk, "Monte-Carlo Tree Search in Production Management Problems," in *Proceedings of the 18th BeNeLux Conference on Artificial Intelligence*, 2006, pp. 91–98.

[29] J. Audibert and S. Bubeck, "Minimax policies for adversarial and stochastic bandits," 2009.

[30] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. Cambridge Univ Press, 1998, vol. 1, no. 1.

[31] B. May and D. Leslie, "Simulation Studies in Optimistic Bayesian Sampling in Contextual-Bandit Problems," Technical Report 11: 02, Statistics Group, Department of Mathematics, University of Bristol, Tech. Rep., 2011.

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

[33] D. St-Pierre, Q. Louveaux, and O. Teytaud, "Online Sparse bandit for Card Game," in *Proceedings of Advanced in Computer Games 2011 (ACG'11)*, 2011.