A Tablut (Viking Chess) Game Playing Agent

Oliver E Clark

April 2018

Contents

1 Abstract

The following report outlines both the approach taken and the reasoning behind various choices made in designing a Tablut (Viking Chess) game playing agent. The primary goal throughout the process was to develop an increasingly skilled tree search algorithm to drive the decisions behind the goal-oriented agent. In order to handle the large branching factor of Tablut (Tablut has a branching factor of over 100, very large compared to around twenty in Chess or Checkers.[1]) In order to handle this complexity, a Minimax search algorithm using alpha-beta pruning was used, limiting the exploration of moves. The search works by selecting the move yielding the best resulting board state. These board states were evaluated by further exploring the tree recursively assuming the opposite players turn at each successive layer. Upon reaching the recursive base case, the search statically analyzes the current board state before returning. The alpha-beta pruning occurs when the search detects what it deems to be unneeded, it does this when it detects moves that regardless of further exploration cannot yield a move better suited to the player at some higher tiered board state. As the relative maximum for that set of children is now known, there becomes no need for further exploration. The aim of this report is to further examine the approach used in the game playing agent and how it could be improved, as well as considering alternate approaches to choosing a move.

2 Pertinent Vocabulary

- **Muscovites** - Team 0 / Black Team / Attacking Team
- **Swedes** - Team 1 / White Team / Defending Team
- **Safe Zone** - Center tile and directly adjacent tiles
- **Candidate Move** - Only moves from the current board state (moves from root) are considered candidate moves.

3 Approach Used

3.1 Top-Level Overview

The following figure (Figure 1) describes the underlying flow in evaluating a SINGLE candidate move (moves available from the current/root board state)

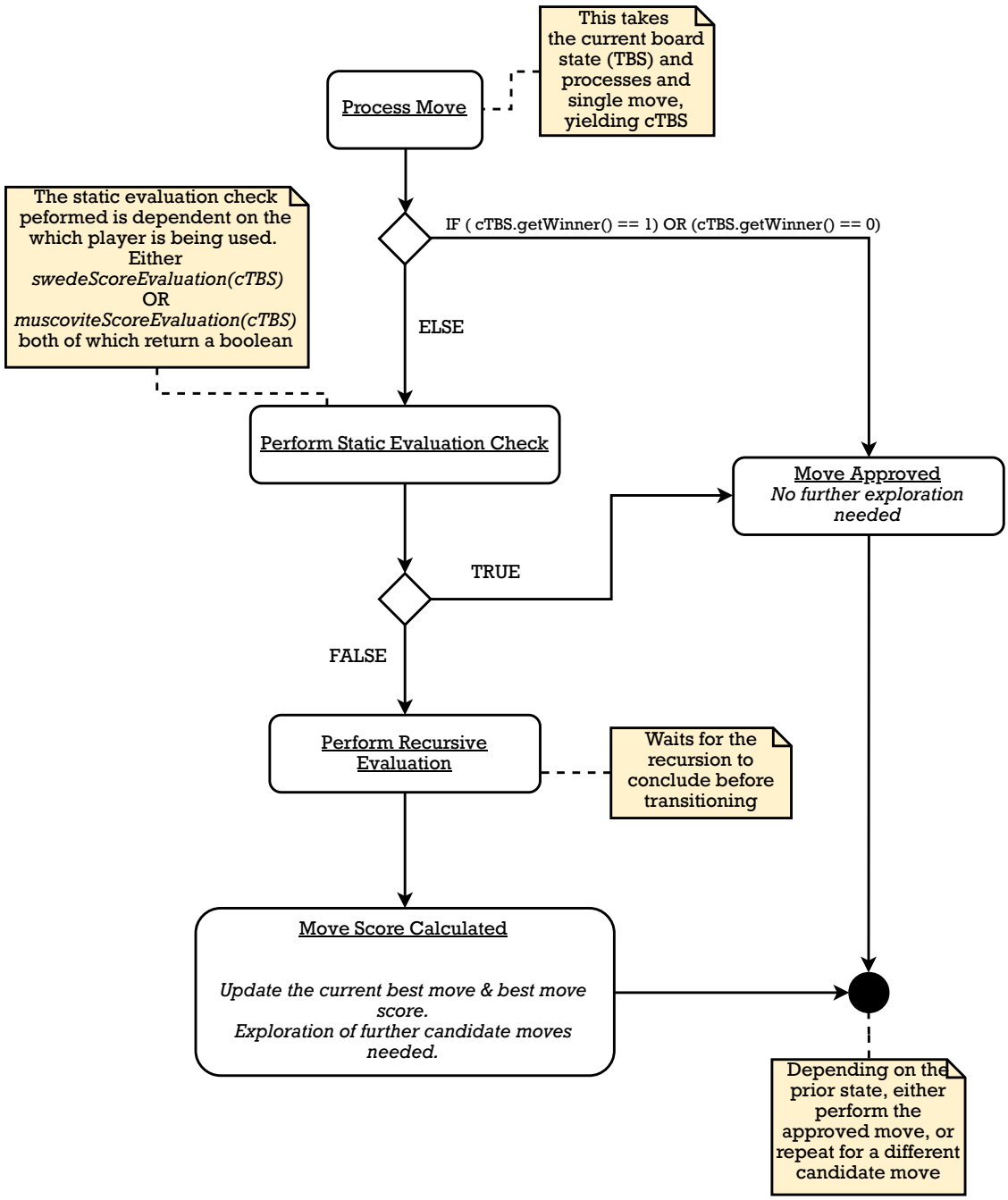

Figure 1: Top-level flow for candidate move evaluation

3.2 Game Search Tree Exploration

Figure 1 (above) shows 'Perform Recursive Evaluation'. This is the point the program that the agent explores moves recursively in order to evaluate a candidate move based on board states that appear deeper into the tree. As mentioned previously, a Minimax search algorithm was used. In order to respect the time limit, this was altered into a depth-limited Minimax search, meaning recursion stops at a certain layer in the tree. Using Minimax search allows the agent to evaluate candidate moves based on their opponents reaction, and then their reaction to their opponents reaction, etc.

3.3 Static Board Evaluation

Static board evaluation occurs when the recursion has reached it's maximum depth and the agent would like to evaluate a game board without simulating moves. This is done in two stages:

First, the agent performs one of two 'checks' depending on the current player, either swedeScoreEvaluation() or muscoviteScoreEvaluation. These checks return true when a move is identified that leads to a definite win. These two checks were used in other locations throughout the code in order to limt resources expended on simulating further moves (see 4.1). If the check performed returns true there is no need for further static evaluation, and the board can be scored as either the max Swede score (3500) or the max Muscovite score (-3500).

Secondly, if the check performed returns false, a generic evaluation must be performed (independent of current player). This evaluation is referred to as sharedScoreEvaluation. In order to keep the complexity of this method relatively simple, it evaluates board states based on a linear combination of several heuristics relevant to the board. The final list of heuristics used is as follows:

- *Number of Swede pieces*
- *Number of Muscovite pieces*
- *Manhattan distance from king to closest corner*
- *Number of Swede pieces neighbouring the king*
- *Number of Muscovite pieces neighbouring the king*
- *Number of Muscovites 'blocking' the corners*
- *Number of Swedes on the edges*

4 Analysis of Approach Taken

4.1 Advantages

Having a discrete evaluation of a board that statically checks if there are potential 'great' moves (see 3.3) for the player who's turn it is currently proved a great advantage of this approach. One of the main benefits of this approach was the functionality of the discrete 'great move checker' throughout the entire program. For example, rather than only performing this discrete evaluation at the depth limit, the agent performs these 'checks' immediately before cloning and state and processing a move. This limits the amount of boards cloned and moves processed (both relatively time/memory intensive).

Finally, by using Alpha-Beta pruning, a huge advantage is the limited exploration that comes from pruning. By limiting specific exploration, the program is simultaneously 'promoting' specific different exploration. Depending on the order of moves (as they come from TablutBoardState) and the quality of the static board evaluation, is the amount of pruning that occurs.

4.2 Disadvantages

One of the major disadvantages of this approach is the lack of depth explored. Due to the inefficiency of the alpa-beta pruning and the large branching factor, consistent game playing results are only found for depth = 2 (respecting the alloted time to select a move). Lacking exploration depth in a game tree-search algorithm can lead to missed good moves and missed risks further into the game.

A final disadvantage unique to Minimax is that the agent assumes the opposing player is playing the game with a similar set of heuristics (values comprising the static board evaluation method). Otherwise, the agent makes decisions based on future opponent moves that are increasingly less likely to happen.

5 Alternative Approaches Taken

Throughout the duration of the project there were brief but valuable alternate approaches attempted.

Initially, a Minimax search was used without Alpha-beta pruning. This approach yielded mediocre results, however this could potentially be due to the relative lack of complexity in the static evaluation method.

Upon adding Alpha-beta pruning, the top-level approach did not change significantly for the remainder of the project. The majority of resoursces were dedicated to small tweaks and varying the heuristics comprising the static evaluation method.

Some of the tweaks attempted:

- Stopping exploration if the king has less than a certain amount of possible moves
- Having a static evaluation method that is based on the quality of possible moves available. This proved difficult due to inconsistencies between Swedes and Muscovites.

Some of the various heuristics used during development:

- Number of unique rows / columns occupied (out of 81)
- Possible king moves
- Total possible moves
- Portion of pawns that do not have an adjacent opponent (out of 9 or 16)
- Prioritization of moves that take the King outside the safe zone (only AFTER turn 5)

6 Possible Further Improvements

6.1 Minor Changes

The following list outlines some potential minor changes that could be done to improve game-playing performance. The changes suggested here are ones that could be made without significant overhaul of the approach already in place.

- Perform structured testing varying the weighting of different heuristic components of the static board evaluation. These results can be used to further enhance performance by perfecting the weighting of the static board evaluation heuristics.
- Create an optimized method to get all possible moves from a board state such that the outputted set is filtered of moves that, for one reason or another, can be detected as not worth exploring.
- Before move exploration, sort the set of moves in a way that optimizes the amount of pruning that occurs in order to increase possible exploration depth.
- Increase the amount of different heuristics that comprise the static evaluation method, potentially improving both pruning and moves selected if the new heuristics provide a more 'accurate' static evaluation.

6.2 Major Changes

The following list outlines some potential major changes that could be done to improve game-playing performance. The changes suggested here would require a signifcant overhaul of the current program or represent a notable change in approach used.

- Implement some type of learning using a large dataset of historical games. Learning could be used to determine optimal moves for any board state. For example : always choose move by ranking possible moves according their probability of being involved in a winning set of moves.
- Due to the large branching factor, implementing Simulated Annealing to determine global optimimum (best move) among possible moves could yield superior results.
- Replacing the search used by using a Monte Carlo Search Tree could provide significant benefit by providing what could be a much needed element of randomness to move selection.

References

[1] Hingston, Phillip. 2007. *Evolving Players for an Ancient Game: Hnefatafl*. Edith Cowan University.