# Tablut -
# A Minimax Approach

**Isaac Sultan**
McGill University
260680295

### Abstract

Tablut is a two player board-game, with a distinct defensive and offensive objective. The *minimax algorithm* was implemented, with *alpha-Beta pruning*, which allowed success against random and greedy players.

## Introduction

Tablut is a variant of the ancient Scandinavian games known as Tafl. It is a two player game, played on a nine-by-nine board. The aim of the first player, *the Muscovites*, is to capture the king piece of the second player, *the Swedes*. The aim of the Swedes is evade capture by moving the king to a corner. Rules for capturing a piece are described by (Linn and Smith 1811).

Tablut presents various challenges in implementing an AI agent. Tablut is a game with a high branching factor of around 100 (chess has a branching factor of 35) (gam 2018). Moreover, to follow the specifications of the project, moves must be decided by the agent within five seconds.

On the other hand, certain features of Tablut are conducive to implementing a powerful AI agent. There is perfect information - both players can see the exact state of the game as the board and the pieces that lie on it are always visible. The game is deterministic, if a move is legal, it can be taken by a player without chance playing a role in the change of state. Tablut is a zero-sum game as the utility value at the end of the game is equal and opposite for either player. These features, make the minimax algorithm an appropriate game strategy. The theoretical basis of minimax, with alpha-beta pruning, will be introduced in methodology.

## Methodology

**Minimax algorithm** Minimax is an adversarial search algorithm. A max player (who wants to maximize its utility) and a min player (who wants to minimize its utility) are defined. Optimal play for max assumes that min also plays optimally. The algorithm expands a search tree, until terminal states have been reached and their utilities computed. The *Student Player* implementation defines a terminal state as either the leaves of the tree (depth two), or if the game has been won by either of the players. In the latter case, if the winner is *Student Player*, the max value is returned. When the leaves are reached, a heuristic function calculates the number of pieces taken from the enemy. After this, the algorithm backs up the worst value at each min node, and the best value at each max node. This backup occurs from leaves towards the current state of the game.



### Alpha-beta pruning

The use of alpha-beta pruning adds two arguments to the function call that track of best leaf value for *Student Player* ($\beta$) and best value ($\alpha$) of the opponent. Whenever the maximum score that the minimizing player is less than the minimum score that the maximizing player ($\alpha$) there is no need for the maximizing player to consider the descendants of this node. Since these descendants will never be reached, the move is returned immediately. Pseudo-code for minimax with alpha-beta pruning is provided below in the appendix (Russell and Norvig 2009).

A known drawback of the minimax algorithm, even with alpha-beta pruning implemented, is its poor time complexity. The run-time is dependent on the ordering of moves. In cases where no moves are pruned, there is a run-time of $O\left(b^m\right)$. With a perfect ordering, the run-time is $O\left(b^{\frac{m}{2}}\right)$. Thus, alpha-beta pruning is a useful addition as it allows double the search depth with the same computational resources.

**Greedy Strategy** The Greedy strategy implemented in the project is defined in the appendix.

## Results

The implemented algorithm was tested against a random player (that picks a random move at each turn), and a greedy player. The student player consistently won against the random player. Strong results were also against recorded against the greedy player, as demonstrated by Table 1.

Table 1: The wins of *Student Player* against *Greedy Player*

| Model Type | Muscovite | Swede |
|---|---|---|
| alpha-beta | 42 | 50 |
| vanilla minimax | 40 | 49 |

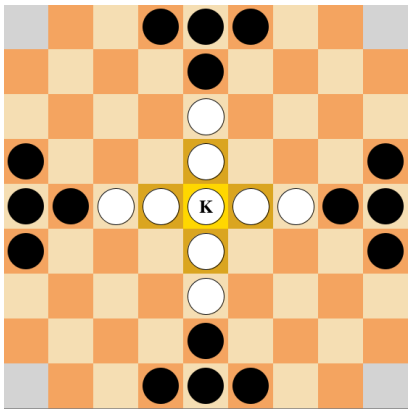


Figure 1: Start of Game

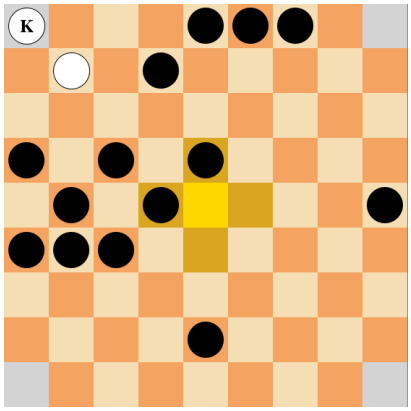


Figure 2: The Swedes win

Hypothesis testing with conducted to evaluate whether there was a difference in performance against the greedy player, when alpha-beta pruning is included. Let $p_1$ and $p_2$ be the proportion of wins for the vanilla minimax and minimax with alpha-beta pruning respectively. A two proportion z-test was conducted.

$$H_0 : \ p_1 = p_1 \tag{1}$$
$$H_1 : \ p_1 \neq p_1 \tag{2}$$

The test resulted in a p-value of $0.4795$. Thus, we have insufficient evidence to reject the null hypothesis i.e. we cannot conclude that the addition of alpha-beta pruning improved performance. However, given the *Student Player* will play against more sophisticated players in the Tablut tournament, it is expected that alpha-beta pruning will be useful in preventing "move time-outs".

## Discussion

To increase the performance of the Muscovite *Student Player* against the Greedy Player, two more heuristics were combined with existing one. The first measured the distance of the king to any of the corners. This value should be maximised by the Muscovites, and minimised by the Swedes. The second additional heuristic measured the number of turns that have passed in order to influence the AI to prefer a faster strategy, with all other factors equal. Unfortunately, these heuristics did not improve the performance of the AI. The use of multiple hyper-parameters required a careful tuning beyond the simple grid-search implemented for the project. More crucially, the distance heuristic increased time-complexity such that the player too often "timed-out", leading to a sub-optimal random move to be chosen.

The *Monte Carlo Tree Search* (MCTS) algorithm was also considered as an alternative approach. MCTS differs from minimax in that its performance is less biased by the heuristic function. This is advantageous given difficulty experienced in harnessed limited domain knowledge to tune its heuristic. However, the limitations on time and RAM enforced by the project would constrain MCTS significantly - it requires many simulations for convergence. AlphaGo is the most prominent example of MCTS being used in game playing (Silver et al. 2016). This model mastered the Go game, another ancient two-player board game. However, unlike Tablut, Go has an extremely large branching factor of 250.

AlphaGo Zero builds upon the power of the aforementioned model (Bib 2018). A deep reinforcement learning strategy (RL), such as using deep Q-networks, was considered. Deep RL models are very difficult to train, as the algorithms are notoriously unstable and sample inefficient. The model could be trained on a cloud platform, and deployed locally. However, this would require utilising a deep learning framework such as PyTorch or TensorFlow, which are prohibited in the project.

Other than heuristic tuning, there is scope for improvement with the project. Use of a transposition table is a promising area for further exploration. A transposition is defined as a a sequence of moves that results in a position which may also be reached by another sequence of moves (gam 2018). A hash-table could be implemented that stores information about positions previously searched, their depth of search, and their utility value. In this way, the transposition table prevents the minimax algorithm from searching again when it encounters a transposition (Greenblatt, Eastlake, and Crocker 1967).

## Appendix A

**Algorithm 1** function alpha-beta(node, depth, $\alpha$, $\beta$, maximizingPlayer)

**if** $depth = 0 \vee nodeisaterminalnode$ **then**
  **return** heuristic value of node
  **if** maximizingPlayer **then**
    $v \leftarrow$ negative infinity
    **for all** child of node **do**
      $v \leftarrow max(v, alphabeta(child, \text{depth - } 1, \alpha, \beta, F))$
      $\alpha \leftarrow max(alpha, v)$
      **if** $\beta \leq \alpha$ **then**
        $break$
      **end if**
    **end for**
    **return** v
  **else**
    $v \leftarrow$ positive infinity
    **for all** child of node **do**
      $v \leftarrow min(v, alphabeta(child, \text{depth } 1, \alpha, \beta, T))$
      $\beta \leftarrow min(\beta, v)$
      **if** $\beta \leq \alpha$ **then**
        $break$
      **end if**
    **end for**
    **return** v
  **end if**
**end if**

## Appendix B

**Algorithm 2** function greedy strategy(node)

$v \leftarrow numberOfOpponentPieces$
**for all** child of node **do**
  $numberOfOpponentPieces \leftarrow processMove(node)$
  **if** numberOfOpponentPieces $\leq$ v **then**
    $bestMove \leftarrow node$
  **end if**
**end for**
**return** node

Documentation for all methods implemented can be found on Github.

## References


[Bib 2018] 2018. AlphaGo Zero: Learning from scratch | DeepMind. [Online; accessed 8. Apr. 2018].

[gam 2018] 2018. GameDev.net. [Online; accessed 8. Apr. 2018].

[Greenblatt, Eastlake, and Crocker 1967] Greenblatt, R. D.; Eastlake, III, D. E.; and Crocker, S. D. 1967. The greenblatt chess program. In *Proceedings of the November 14-16, 1967, Fall Joint Computer Conference*, AFIPS '67 (Fall), 801–810. New York, NY, USA: ACM.

[Linn and Smith 1811] Linn, C. v., and Smith, J. E. 1811. Lachesis lapponica; or, a tour in lapland.

[Russell and Norvig 2009] Russell, S., and Norvig, P. 2009. *Artificial Intelligence: A Modern Approach*. Upper Saddle River, NJ, USA: Prentice Hall Press, 3rd edition.

[Silver et al. 2016] Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; van den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; Dieleman, S.; Grewe, D.; Nham, J.; Kalchbrenner, N.; Sutskever, I.; Lillicrap, T.; Leach, M.; Kavukcuoglu, K.; Graepel, T.; and Hassabis, D. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529(7587):484–489.