Designing an Effective Tablut Artificial Intelligence

Sebastian Pilarski
sebastian.pilarski@mail.mcgill.ca

***Abstract*—This paper presents an implementation of a Tablut artificial intelligence player engine model. The model is based on a combination of heuristics and a modified Monte Carlo algorithm widely used for board games. The engine is much stronger than the default greedy and random algorithms, and traditional Monte Carlo implementations. In fact, all games during experimental trials were won by the newly developed player engine.**

***Index Terms*—Tablut, Artificial Intelligence, Game Theory, Monte Carlo, Heuristics**

I. Introduction

Artificial intelligence methods and techniques have been applied to many popular tournament games such as Chess and Go. After extensive research spanning numerous decades, Chess engines are capable of beating top human players. A multitude of techniques have been explored including the use of Deep Learning, Monte Carlo methods, traditional heuristics, and more.

While there has been significant research into the methods and implementations of algorithms of the more popular games, there are many interesting strategy games that remain to be researched. The goal of this paper is to present successful artificial intelligence techniques implemented and evaluated within the board game Tablut.

While the game is simple in rules, there is a high branching factor and strong strategic element, similarly to Chess. This paper will attempt to showcase a model of progressive improvement of an artificial intelligence which runs on a JVM with 500 MB of memory and is given 2 seconds (real-time) to complete a move. After 100 moves have passed, the game is considered a draw. These assumptions are based upon a Tablut Artificial Intelligence competition's rules.

This paper is organized as follows. First, the detailed rules are introduced in Section I-A. Following is an examination of basic strategies and a greedy algorithm in Section II. This is followed by Section III detailing my more advanced AI strategies and heuristics that can be employed for significant improvement onto a greedy algorithm baseline. Next in Section IV, general Monte Carlo technique application are described and compared with a more specific, successful Tablut tailored Monte Carlo variation. Subsequently experimental results are presented in Section V, comparing the various artificial intelligence models together. After presentation of experimental results, a section detailing the advantages and pitfalls of various models is discussed, see Section VI. A brief section detailing optimization techniques (Section VII) is then followed by future improvements Section VIII and by concluding remarks Section IX.

A. Game Rules

Tablut is a game comprised of two players, one playing as the $Swedes$ (white pieces) and the other as the $Muscovites$ (black pieces). The $Swedes$ begin in a cross-like formation at the center of the board, and are surround by $Muscovites$ pieces on the edges as can be seen in Figure 1. To win the game, the $Swedes$ player must lead his or her King piece, labeled K in the center, to any corner. The $Muscovites$ win the game if they capture the king.

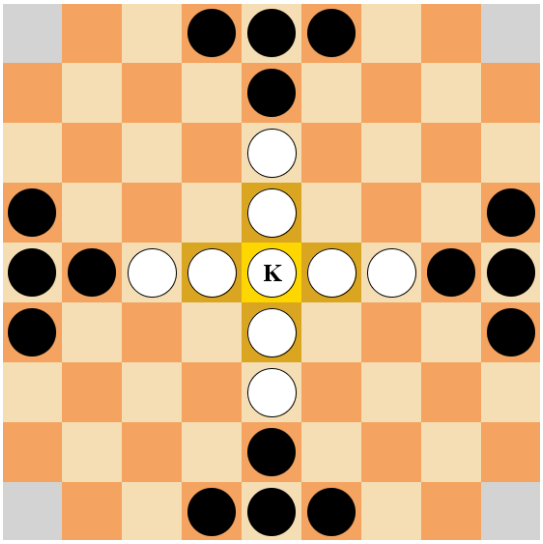

Figure 1. Tablut Starting Position

The game is played in turns, with the black, $Muscovites$, player always starting first. At each player's turn, the player must move a piece. Each piece can move horizontally or vertically until it encounters a wall or another piece, or special square. This behavior is similar to how a rook in Chess moves, but the piece cannot land on another piece. An opponent's piece can be captured by maneuvering two pieces on opposite sides of the opposing piece. The two pieces must be in either the same horizontal or vertical plane. A piece cannot be captured in a diagonal plane. An example capture can be seen in Figure 2. If a player places his piece between two opposing pieces, the player does not lose his or her piece - the capturing must be performed actively by an opponent moving a piece to an adjacent square.

Capturing is performed the same way for both players. The king, however, is a special piece and can be more difficult to capture. When the king is in the center or in the adjacent squares, highlighted in gold in Figure 1, he must be surrounded

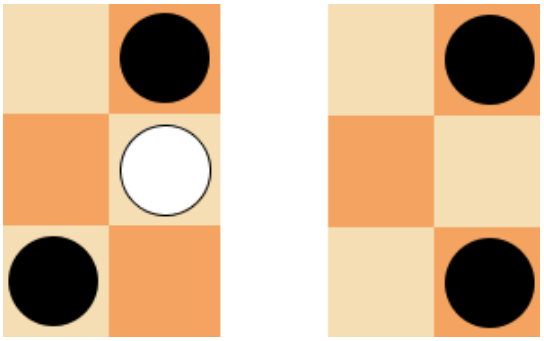

Figure 2. Basic capture in vertical plane. The $Muscovites$ player moves his piece to sandwich the $Swedes$ piece, and the $Swedes$ piece is removed from the board.

by enemy pieces on all four sides in order to be captured, as can be seen in Figure 3. The king can be captured by just two $Muscovites$ pieces, like any other piece, outside of these central squares. The king can also be used to capture opposing pieces, just like any other regular piece.

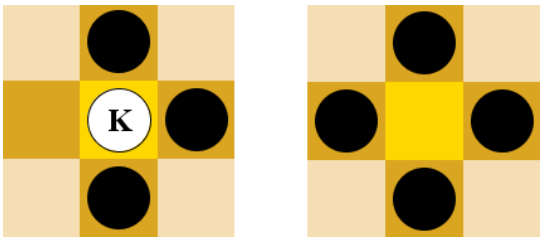

Figure 3. King capture in center

The Tablut board has five special squares: The four corners, and the very center. Only the king may land on these squares. All other pieces may move through the center, but they may not end their turn on it. Each of these squares can be used to capture an opponent. If an opposing player sandwiches a piece between a special square and his or her piece, the sandwiched piece is captured. This can while using the center regardless if whether the king stands in the center or not. When the king is positioned on a gold central square other than the center, only three opposing pieces are needed, as the special center piece is adjacent to the king. A couple of examples of capturing using the special squares can be seen in Figure 4. These are the complete rules for the game.

II. Basic Strategies and a Greedy Attempt

The $Muscovites$ begin with a significant advantage in number of pieces - starting with 16 while the $Swedes$ start with 9. As such, as a basic strategy, it is beneficial for them to capture opponent pieces in an effort to expose the king and lead to his eventual capture.

The $Swedes$ must find a way to create an opening for the king to escape. Less pieces, both white and black, enable the king to have more room to maneuver. Therefore, as a simple strategy, the $Swedes$ can seek to capture the $Muscovite$ pieces. As the king's final goal is to reach the corner, the strategy

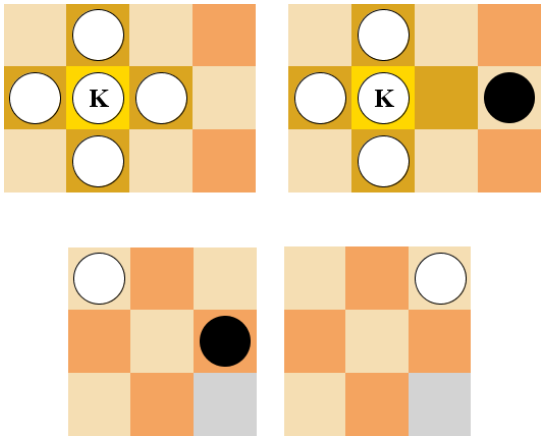

Figure 4. Example captures using a special square

can involve moving towards the nearest corner, when a piece cannot be captured.

This is the concept of a baseline greedy algorithm. If a piece can be captured, capture it. If no piece can be captured, and playing as the $Swedes$ move the king towards the nearest corner. Else, perform a random move.

This sort of greedy algorithm can generally beat a random assortment of moves, but it breaks down when playing against more 'intelligent' opponents, as it fails to prioritize capturing the king while playing as the $Muscovites$ and fails to ensure the king is safe in a given position when playing as the $Swedes$. The following section will provide significant expansion of how this baseline model can be greatly improved to beat more sophisticated models and even human players.

III. Developing New Advanced Strategies and Heuristics

To develop a more effective player, it is imperative to develop a method of searching for a victory. In the case of the $Muscovites$, this means searching move possibilities to whether the enemy king can be captured, or preventing the $Swedes$ king from having a path to the corner. The $Swedes$ must search for a path to the corner.

A. Details of Reasoning and Motivation for New Strategies

The most important skill for an artificial intelligence to possess in the game of Tablut is the ability to take a clear path to a corner as the $Swedes$. This is a relatively easy task - and was accomplished in the greedy algorithm, by searching for a shortest distance to a corner and then striving to minimize the distance to the corner.

In the greedy baseline algorithm, however, this was an after thought, with the algorithm prioritizing a piece capture to moving directly to a corner if presented with an open path. The baseline algorithm is quickly improved by checking for a possible victory state, and if it can be accomplished, choosing the corresponding move. Now, what if there were a sequence of moves that could guarantee a victory within a certain number of moves - such as 2 or 3 or more? This would provide a

significant advantage to the artificial intelligence - however, calculation through all possible moves would be very resource intensive.

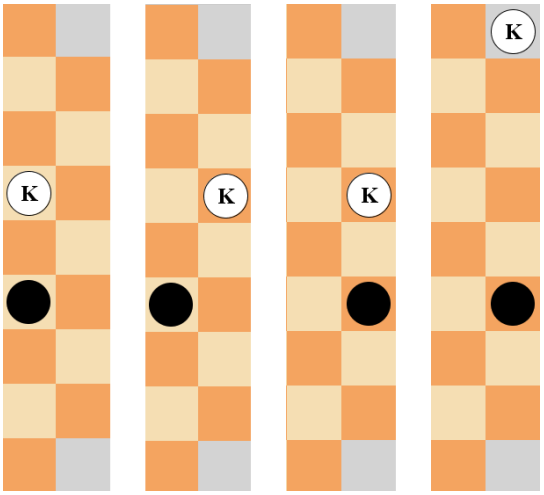

Figure 5. A clear edge can guarantee a victory. The opponent can at best, block a single corner with one move, allowing the king to move to the other in the following move.

Through deep analysis of the game, however, looking only at a current move, a sequence of moves can be determined. First, if an edge is free of any piece, whether opposing or own, then it is an automatic win in two moves as is demonstrated in Figure 5. This is one circumstance where looking from the perspective of one move, is the equivalent of finding a victory two moves ahead. With more careful analysis, another such guaranteed victory path can be found.

Another situation, which will henceforth be referred to as a safe edge, is as follows. If an edge is clear of pieces from the edge square onto which the king can move, and all of the perpendicular rows (assuming edge on vertical axis) are free of opposing pieces (meaning no $Muscovite$ piece can then block the path to the corner from the edge), then this is a guaranteed path to victory, as demonstrated in Figure 6.

With careful analysis, this concept can be expanded further. If there is only a single opposing piece in the final row before the corner, the king can then capture the piece, by wedging it between the special corner square and himself. This creates an open path for the next turn, and the king cannot be captured - thus also guaranteeing a win. An example of such a sequence is diagrammed in Figure 7.

Safe edge variations such as these can be quickly calculated, well under 1ms speeds, with certain optimization techniques as discussed in the later optimizations section. Further variations of such strategies, enabling searching for victory several moves ahead, can be discovered. For the sake of brevity of this paper, the topic of discussion will now shift to strategic king positions.

The king is generally safer and more difficult to capture in the central squares, as more opposing pieces are required for capture. It is also more difficult, especially for artificial models, to coordinate an effective attack to trap the king from escaping from all four sides, thus making the center a stronger

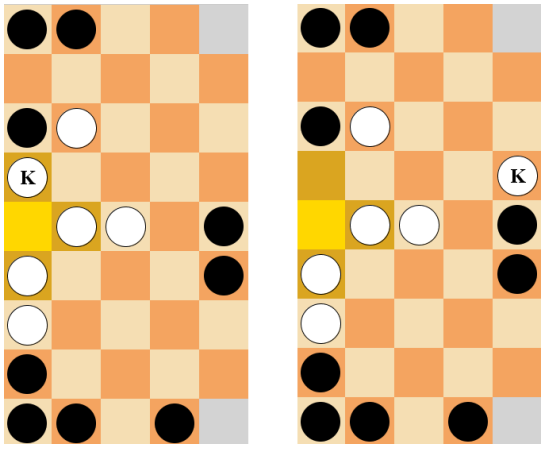

Figure 6. An example of a safe edge. Assuming the second row is free of black pieces to the full left of the board. Black cannot maneuver any pieces to block the $Swedes$ king from being able to move to the top-right corner on the following $Swedes$ turn.

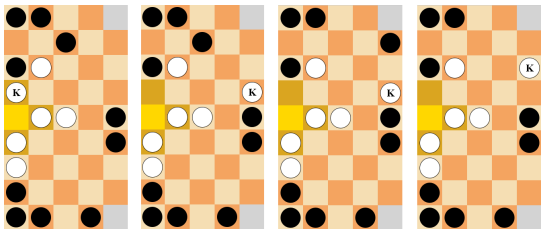

Figure 7. An example of a more complicated safe edge escape. The $Muscovites$ are capable of moving a piece between the king and the exit. The king, however, is able to move forward and capture the piece by sandwiching it between the corner. The $Muscovites$ now have no way of blocking the king from the exit or capturing him, and thus the $Swedes$ are guaranteed a victory in the next move.

strategic position than most elsewhere on the board. This is another pitfall of the greedy algorithm's approach to playing as $Swedes$, as it moves the king from safer central pieces to more dangerous positions in the corner.

It is important to provide the king maneuvering room, to create openings for clear edges and safe edges, as well as to allow him to escape. Allowing $Swedes$ pieces to be captured, however, is not an effective strategy. Pieces are necessary to capture opposing $Muscovite$ pieces, and are especially important to prevent from being captured for the already smaller $Swedes$ force. Thus, a proposed method is to calculate the relative danger to each of a player's pieces (how many could be captured, etc.), and compare the relative gain in capturing opponent pieces. If two or three pieces can be captured in exchange for a single piece, this is generally an effective move.

A method must be determined to keep the king safe when he is placed inside a dangerous situation. If there is an escape that guarantees victory, meaning no danger, it should be taken. If no such victory path exists, what move should be performed to take the king out of danger? Combining the previous discussed strategic necessities into a heuristic and giving value to certain strategic elements can help determine what is an ideal move in

a situation. If a move creates the potential for a victory path and removes the king from danger, it is a good path.

Now to briefly describe the black $Muscovites$ more advanced strategies. Essentially, the strategy is the reverse of the other player's, as Tablut is a zero-sum game. If at any given time, the king is exposed and can be captured, this is the optimal path and should be taken. The $Muscovites$ player can also use the same knowledge of clear edges and safe edges to his or her advantage. It can be used to calculate and ensure that king victory paths are not available following a move. If no path to victory exists for the king, the king cannot win. Using the same capturing strategy as the $Swedes$ is effective. It ensures the $Muscovites$ force has enough pieces to block king victory paths, and strives to capture opposing pieces, slowly. This leaves limited maneuvering room for the king, and allows the $Muscovites$ forces to slowly attempt to capture the king.

Simply by implementing this easily observed domain knowledge, the greedy baseline can become a strong artificial intelligence player.

B. Proposed Heuristic Scoring Scheme

The following proposed heuristic was discovered to be effective after some tuning. For each legal move the value of the move is calculated as follows:

$Swedes$ (White):

If king can be captured:	-1000
For each piece opponent can capture:	- 5
For each opponent piece captured:	+ 10
If king has open victory path:	+ 5
If king has moved to central square:	+ 1

$Muscovites$ (Black):

For each piece opponent can capture:	- 5
For each opponent piece captured:	+ 10
If king does not have open victory path:	+1000
If king is captured:	+1000

IV. Strengthening Monte Carlo

Monte Carlo simulations are a traditional model used to create artificial intelligence for games such as Chess and Tablut. By simulating random moves in a game over a period of time, the strengths of certain moves can be approximated through a measure of the number of wins or losses that occurred in a sequence involving a given move.

A. Traditional Monte Carlo

A very traditional Monte Carlo approach was implemented and compared to the baseline-greedy player. This Monte Carlo method implemented UCT for selecting states to explore and implemented an objective function of Wins/Total. After some tests, a depth of 8 explored steps per simulation was found to be appropriate. The method performed well against a greedy $Swedes$ king trying to move ever closer to a corner and disregarding any danger - it was capable of seeing when the king can be captured. In the reverse, playing as the $Swedes$ against the $Muscovites$, the Monte Carlo method performed better than a greedy $Swedes$ does, generally keeping the king out of danger, and sometimes being able to determine safe and clear edges through its simulations. It was is not however, as effective as the heuristic model.

B. Improving Monte Carlo

To improve the traditional Monte Carlo method, several hyperparameters can be experimented with. First, the objective function, can be changed to (Wins-Losses)/Total, to penalize losses more than draws. This maintains a penalty on the number of draws still penalized, as with the same number of wins, but an increased number of draws, the objective function decreases. Second, the UCT formula can be changed to a non-traditional variant which changes the logarithmic term of UCT to the natural logarithm of the sum of the number each next state was visited. Third, to aid the Monte Carlo method in seeing a victory or a loss, the checks for safe edges, clear edges, etc, can be implemented as a victory or loss rather than searching for a more unlikely solution where the king makes his way all the way to a corner. For the other side, we can determine whether the king can be killed in a given state, in order to prevent a random non-capture in the simulation while the king is in a dangerous situation.

$$\frac{\text{Wins - Losses}}{\text{Times Current State Seen}} + c\sqrt{\frac{\ln(\sum_i \text{Times State } i \text{ Seen})}{\text{Times Current State Seen}}}$$

Figure 8. UCT formula variant used in improved Monte Carlo Method

This new variation is now a stronger player than the traditional Monte Carlo model, and performs significantly better, but it still loses to the traditional heuristic-based model. What if the two were combined? The heuristic model supplies a very strong set of moves - the heuristic model can be used as a filter for a subset of starting moves to be considered inside Monte Carlo. The benefit here is twofold. One - fewer states to simulate in Monte Carlo, which provides better statistical insight. Two - Monte Carlo may now determine which moves returned from the heuristic provide greater advantage when exploring several moves into the future. In theory, and when tested with longer time constraints, there was significant evidence which showed this model to be effective.

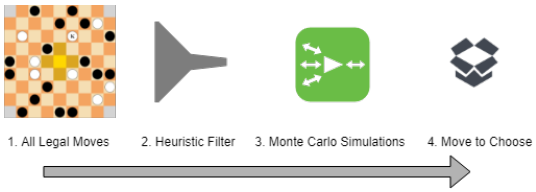

Figure 9. Flow diagram of the heuristic-based Monte Carlo model process

V. Results

First, tests were performed compare the greedy algorithm to the random move generator. The greedy model defeated the random move generator while playing as both the $Swedes$ and

as the $Muscovites$. The experimental results can be found in Table I.

Table I
GREEDY ALGORITHM AGAINST RANDOM MOVE GENERATOR

	Wins	Losses	Draws	Avg. Moves (Non Draw)
$Muscovites$	10	0	0	26
$Swedes$	10	0	0	16

After completion of the purely heuristic-based model, a series of tests were performed to see how it compares to the greedy baseline. It managed to defeat the greedy algorithm quickly, requiring less than 15 moves on average, regardless of whether playing as the $Muscovites$ or as the $Swedes$. The results are shown in Table II

Table II
HEURISTIC MODEL AGAINST BASELINE GREEDY ALGORITHM

	Wins	Losses	Draws	Avg. Moves (Non Draw)
$Muscovites$	10	0	0	9
$Swedes$	10	0	0	14

The next model evaluated against the greedy algorithm is the pure traditional Monte Carlo algorithm with various depth searches (depth of 8 was best) and the c constant set at the commonly used value of $\sqrt{2}$. The model appears to play at a level similar to greedy as is presented in Table III.

Table III
PURE TRADITIONAL MONTE CARLO VS GREEDY BASELINE

	Wins	Losses	Draws	Avg. Moves (Non Draw)
$Muscovites$	6	4	0	15
$Swedes$	6	4	0	20

After adding encodings of *king can be captured* and a *victory path exists* to the definition of a win or loss in the Pure Monte Carlo model, the performance as measured in relation to the greedy significantly improves for the $Swedes$. The model, however, does perform worse for the $Muscovites$ as it does not attribute any greater win value for killing the king - determining that being able to kill the king is a winning state, before actually killing the king. Thus, these expanded definitions should be used for $Swedes$, but not for $Muscovites$ when using purely traditional Monte Carlo Methods. See Table IV for statistics.

Table IV
MONTE CARLO WITH EXPANDED DEFINITIONS AGAINST GREEDY ALGORITHM

	Wins	Losses	Draws	Avg. Moves (Non Draw)
$Muscovites$	0	10	0	18
$Swedes$	10	0	0	18

The next test is a performance comparison test between the heuristic values and Monte Carlo. In these tests, the Monte Carlo model disregards the expanded definitions for $Muscovites$, but uses them while playing as $Swedes$. This heuristic helps determine the effectiveness of the defined heuristics. The heuristic model for $Swedes$ is significantly stronger than Monte Carlo's $Muscovite$ playing ability. The heuristic's level of $Muscovite$ play appears comparable to the Monte Carlo's $Swedes$. Please refer to Table V.

Table V
HEURISTIC MODEL AGAINST MONTE CARLO

	Wins	Losses	Draws	Avg. Moves (Non Draw)
$Muscovites$	1	1	8	24
$Swedes$	10	0	0	20

Having seen that the heuristics developed in the preliminary model are effective, a comparison of the final model combining the heuristic model with Monte Carlo is constructed and evaluated. It is capable of defeating all previous models and performs better than the previous best heuristic model. Results are summarized in Table VI.

Table VI
HEURISTIC MODEL COMBINED WITH MONTE CARLO AGAINST OTHER MODELS

	Model Plays As		
	$Muscovites$	$Swedes$	**Opponent Model**
Wins	20	20	Random
Losses	0	0	
Draws	0	0	
Avg. Moves (Non Draw)	21	16	
Wins	20	20	Greedy
Losses	0	0	
Draws	0	0	
Avg. Moves (Non Draw)	9	17	
Wins	13	18	Heuristic Model
Losses	6	2	
Draws	21	10	
Avg. Moves (Non Draw)	67	43	
Wins	10	10	Pure Traditional Monte Carlo
Losses	0	0	
Draws	0	0	
Avg. Moves (Non Draw)	19	20	
Win	5	10	Expanded Definition Monte Carlo
Losses	0	0	
Draws	5	0	
Avg. Moves (Non Draw)	34	20	

VI. STRENGTHS AND WEAKNESSES OF PROPOSED ARTIFICIAL INTELLIGENCE MODELS

A. Pure Heuristic-Based Model

The strengths of the heuristic-based model is that it encodes a significant level of discovered domain knowledge. This enables finding clear paths to victory up to five total moves ahead as the $Swedes$ and enables the $Muscovites$ to use this knowledge to prevent these victory paths as they emerge. It is also very quick and decides a move between sub-millisecond times to 6ms in real-time. By selecting a random move from its list of strongest moves, it tends to explore new options when faced with the same situation, resulting in new strategy attempts. Existing weaknesses still include lack of significant foresight when protecting the king - the model selects the best heuristic value, preventing capture in one turn if is possible, while another option may still have been better for several moves into the future.

B. Heuristic Model Combined with Monte Carlo

Through use of the heuristic model's evaluation function, it filters out the first ineffective moves, enabling greater insight into the statistics of the most-effective moves. This yields insight into which move is the best move several moves ahead, and not just for the basic heuristic's end state evaluation. This method does have some weaknesses however. First, it is more likely to oscillate and repeat the same pattern of moves continuously than the purely heuristic-based model, which, given a strict limit of 100 moves, moves the player closer to a draw. It is also more prone to timeouts than other models. Given the greater average calculation time and variability in determining whether a given state contains a sequence of victory paths, the simulation is more likely to overextend the allotted time. Especially given the competition's rules of real-time, rather than calculation-time, timeouts can occur unexpectedly. To remedy this issue, one must select a short allotted simulation time, preventing highly detailed statistical learning. Thus, there must be significant optimizations to enable greater amounts of simulations to be performed.

VII. Optimizations

Optimizations are important to develop an effective artificial intelligence. With slow methods of computing, successful algorithms may not be able to fully complete their calculations given a limited amount of time.

The first optimization implemented is a new custom board object, called `My_board` which is much more lightweight than the tournament provided `TablutBoardState`. The `My_board` object contains an array representing a board state, as well as the king's position coordinates, which player is in the current state $Swedes$ or $Muscovites$, and a storage for the start and end positions for a move. This provides significant speed advantage (up to 30x faster) over using `TablutBoardState` as it does not involve copying large data structures such as HashMaps. A `My_move` object is also created to perform moves on the optimized board.

Certain optimizations can greatly improve the Monte Carlo speed. Through the use of a unique hashing function, independent of memory location, for the `My_board` object, board states can be discovered quickly and stored between simulation runs. This enables a Monte Carlo tree to be built, without forming an entire traditional tree structure.

Various other small optimizations exist. For example, ArrayLists are allocated with sufficient capacity to avoid time expensive reallocation of memory. Reallocation of memory was likely the cause of some timeouts occurring. Some ArrayLists are reused and cleared rather than reallocated each time in a loop.

VIII. Future Improvements and Works

The artificial intelligence models proposed in this paper can be significantly improved by moving implementation onto a C++ backend. This would enable many more simulation runs to occur and would prevent costly and unpredictable garbage-collections during simulation times. Due to competition rules this was not an option, but for standard play this is highly recommended.

Additional works and experiments can be performed by training neural networks. Likely, a combination of Convolutional Neural Networks and Recurrant Neural Networks could be used to make a strong Tablut player, especially with encoding the domain knowledge described in this paper.

IX. Conclusions

This paper presents that my heuristics combined with a modified Monte Carlo algorithm produced a much stronger algorithm than the greedy algorithm baseline. Experimental results show that neither the greedy algorithm nor the random move generator could win a single game against my implemented player engine model. Moreover, my model defeats the traditional Monte Carlo algorithm every time.

The results demonstrate that a heuristic model connected to a Monte Carlo simulator generated a strong Tablut player. This work shows how discovering and encoding domain-specific knowledge can greatly strengthen traditional artificial intelligence techniques.