Using Domain-Dependent Knowledge in Stratego

J. Mohnen

June 16, 2009

Abstract

The search space of the game of Stratego is complex, because it is a game of imperfect information. In order to deal more efficiently with this problem, domain-dependent knowledge can be used. One possibility is to construct an evaluation function which determines the value of a position in the game tree. This evaluation function uses features of Stratego positions. A second possibility is to use domain knowledge as a guide for forward pruning, which removes undesirable branches early during the search. These branches are detected using characteristics of the move. While both methods will improve the performance of a Stratego program, using an evaluation function with domain knowledge seems the better method to use domain knowledge.

1 Introduction

The game of Stratego[1] is a game of imperfect information, since the rank of a piece is hidden. Both players do not know the rank of their opponent's pieces at the start of a game. Even though more information becomes available during the course of the game, determining the next best move remains difficult. The search space in which this move can be found is vast, due to the large playing field, the number of pieces and the lack of information. Therefore it is desirable to find a way to reduce this space and to explore it more optimal. This paper discusses the use of domain knowledge from the field of Stratego to achieve these goals.

Firstly, domain knowledge can be used to construct a good evaluation function, which determines how good a board position is. Secondly, domain knowledge can be used to improve the search in the game tree. The search applied to the game tree will use this domain knowledge to explore the game tree, applying the correct amount of pruning where possible. In this paper one form of pruning, namely forward pruning will be discussed. The search depth of an algorithm can be varied, exploring some branches deeper than others. Branches which look more promising are explored further, while others are pruned. This last case is known as forward pruning [5]. This paper will focus on searching and pruning game trees by using domain knowledge. In order to do this, the search algorithm needs domain specific knowledge to determine which branches of the search tree are less likely to backpropagate to the root and can thus be pruned.

The paper will start with a short introduction to the rules of the game and an analysis of why domain knowledge is needed. In Section 3 the evaluation function will be discussed. Section 4 deals with domain knowledge in Stratego and how it is applied to forward pruning. Section 5 shows the experiments based on these theories and in Section 6 the results of these experiments will be discussed.

2 Stratego

This section gives a short introduction to the rules of Stratego.[2] This section ends with a short analysis of the game and several options of related work.

2.1 Rules

The original board game Stratego is a two-player game and has a playing field of 10×10 squares. There are two lakes in the center of the board, each with a size of 2×2 squares. Pieces can not be placed on or moved onto these lakes. Each player starts with 40 pieces with varying ranks as stated in Table 1. The higher the rank in the table, the stronger the corresponding piece is. Both players have the same ranks available.

A player cannot see the ranks of the opponent's pieces. The players take turns moving one piece across the board. The goal of the game is to capture the opponent's Flag or to make the opponent unable to move his pieces. Pieces can only move one square at a time, with certain exceptions (listed below). When two

---

[2]For a more detailed explanation of the rules, see http://www.boardgamecapital.com/stratego-rules.htm

Rank	Name	Quantity on each side
1	Spy	1
2	Scout	8
3	Miner	5
4	Sergeant	4
5	Lieutenant	4
6	Captain	4
7	Major	3
8	Colonel	2
9	General	1
10	Marshall	1
F	Flag	1
B	Bomb	6

Table 1: List of Stratego pieces

pieces move onto the same square, the ranks of both pieces are revealed. The piece with the lowest rank is removed from the game. If the two pieces are of the same rank, both are removed.

Several pieces have special properties. These are:

- Bomb: this piece cannot move, but it can only be destroyed by the Miner.

- Flag: this piece cannot move and the game is lost if this piece is captured by the opponent.

- Miner: this is the only piece which can capture Bombs.

- Scout: this piece can move as a rook in chess.

- Spy: this piece has the lowest rank, however it can capture the Marshall (the highest rank) if the Spy attacks.

2.2 An Analysis and Related Work

Stratego is, as the name suggests, a game of strategy. A player will need a strategy which tries to capture as many of the opponent's pieces, while revealing and losing as little as possible of his[3] own pieces. The player will need to plan his moves carefully, in order to achieve this goal. An important issue is the knowledge which the player has. He will not know the rank of all of the pieces of the opponent, meaning that trying to capture a piece may potentially be dangerous: the target piece may in fact be of a higher rank than the player's piece, resulting in that piece being captured. But if the player wins and captures a piece, he will give away information about the rank of his own piece as well. Even by moving a piece he will give away information which might prove to be valuable for his opponent. Thus, a key issue is to find a balance between capturing and finding

---

[3]For brevity, we use 'he' and 'his' when 'he or she' and 'his or her' are meant

information about the pieces of the opponent and not losing or revealing own pieces.

There are several strategic ideas which human players use to improve their performance. These ideas have been incorporated into Stratego programs. One of these ideas concerns the starting setup. A good starting setup has proven to be an important factor to achieve victory. There are some positions which have proven to be generally good, but it is equally important to keep the starting setup unpredictable [8]. Another strategy is to bluff. By bluffing the opponent can become misinformed. This is a subject of opponent-modeling approaches on the game of Stratego. Another idea is to model each piece as a computational agent, which acts based on rules and on its environment [13].

3 Evaluation Function

Evaluation functions were already used in chess programs as early as 1953, by Alan Turing [14]. The evaluation function is used to determine how desirable a certain position is by the algorithm. One of the most basic evaluation functions for chess involves counting the pieces on the board. The evaluation function will 'reward' good positions by increasing their value. Thus, if a position is desirable, it will lead to a higher value for the corresponding branch in the search tree. Improvements of the evaluation function will therefore also lead to improvements in other parts of the program, because the value of each node in the search tree will be more accurate. Optimizing the evaluation function leads to better pruning in general and safer forward pruning due to better estimations of the final values of a branch. Forward pruning will be discussed in section 4. The domain knowledge used to construct the evaluation function relies mostly on human-player experience [8]. Table 2 shows several situations and their respective change to the value of that position.

Situation	Effect
Counting Pieces	Bonus
Miners captured	Penalty
Spy captured	Penalty
Enemy pieces near the Flag	Penalty
Own pieces near the enemy Flag	Bonus
Scouts near unknown enemy	Bonus
Blocking positions	Bonus
Pieces near lower enemy ranks	Bonus

Table 2: Several situation which can occur during a game and their respective influence on the position value

The first factor is material based and commonly used in evaluation functions in, for instance, chess

and checkers [14]. The basic evaluation function used in this paper makes use of this first factor and it is mainly used to make sure the game progresses. For each of the own pieces, the value of the evaluation function is increased with an amount based on the position and rank of the piece. This value is multiplied by a variable: the unknown bonus. This variable is based on the opponent's knowledge about that piece. The unknown bonus is only applied if the rank of the piece is still unknown to the opponent. A second variable consists of a counter which keeps track of the number of turns a player has played without capturing a piece of the opponent. The evaluation function receives a penalty based on this variable. The counter is reset when a piece of the opponent is captured.

The amount with which each of these situations influences the value of a position changes during the course of a game. For instance, losing a Spy will result in a large penalty, because without the Spy a player can not put any pressure on the opponent's Marshall. However, losing a Spy when the opponent's Marshall is already captured will result in a much lower penalty (perhaps even no penalty at all). This leads to the following theory: the usefulness can be expressed as a variable value. The value of a piece changes over the course of a game and is depedent on several factors [8]. The most obvious factor is the rank of the piece. Higher ranks have a higher value than lower ranks, because higher rank are able to capture more pieces. Another factor to take into account is the special ability of certain ranks. Ranks with special abilities will have their values increased. For instance, Miners can have a higher value due to their ability to remove Bombs from the game. Yet another factor is the information a player has about the pieces of the opponent. Suppose for instance, player A has lost his Marshall. The value of the Spy of player B will decrease, because he can not capture any of the opponent's pieces. The General of player B on the other hand will have its value increased, because it can now move more safely around the field. The challenge is to find well performing factors and tuning them.

4   Forward Pruning

As can be concluded from the short introduction into the rules of Stratego, players need to think ahead and come up with strategies to ultimately capture the opponent's Flag. Stratego is a game of imperfect information, meaning that obtaining information about the opponent and not revealing one's own information is an important part of these strategies. This makes the search tree of a game of Stratego large and complex. The widely used $\alpha\beta$-algorithm is not sufficient to provide a best move

in a reasonable amount of time, because the number of nodes visited by this algorithm increases exponentially with increasing search depth [9, 10].

It is useful to use heuristics, much like human players do. Promising branches of the game tree will be explored further, while others will be cut off. The removal of these unpromising branches is called forward pruning [5, 16]. There are several factors which are important to effective forward pruning. One of these factors is domain-independency. Ideally a pruning method should be applicable to several domains. But most existing forward-pruning methods take advantage of characteristics of the search space and are therefore domain specific [5]. Some examples of forward pruning methods are null-move [3], ProbCut [7] and multi-cut [5, 6]. Due to Stratego's complex game tree, we investigate whether domain-dependent forward pruning can improve the search as well.

Forward pruning will decrease the search depth of moves and decisions which look unpromising. These branches in the game tree will not be expanded further, saving time which can be used to look deeper into better moves. The most important question is: how to decide if a move is potentially bad? There are moves which would seem obviously bad to human players. But a Stratego AI will need to find out which branches in the tree will provide a better position by searching. To find out if a node has a better or worse value than other nodes, a shallow search has to be performed. Several plies will be needed to determine the value of the move. If it becomes clear that the value is lower than other, already explored, moves, the normal search does not have to be performed. Otherwise, the branch can be explored further. The amount of plies the algorithm should perform before deciding whether or not to prune is subject to tuning.

Besides forward pruning, there is also another method to improve the tree search. This method is basically the opposite of forward pruning and is called singular extensions. Singular extentions seek to explore promising branches further, essentially providing more time to search through these branches [2]. In this paper the focus will be on pruning undesirable branches from the game tree, so the algorithm does not waste time on bad moves. The idea is that this results in more time to further explore potentially desirable branches using iterative deepening.

4.1   Domain Knowledge Applied to Moves

The domain-specific knowledge of Stratego consists of several aspects, such as knowing what a good starting

position is, determining if a move will be obviously bad or good and determining whether certain positions are desired positions. This paper will not go further into determining a good starting position. It is assumed that both players have an equally good starting position during the experiments discussed later. The other aspects are explored further and will be discussed.

Certain moves will immediately be recognised as being undesirable moves by human players, for example: moving a Scout next to a known higher-ranked piece. The Scout will be captured and no new information will be revealed. A computer player however, will have to go through the search tree to find that this is a bad move. This takes time and since the search algorithm only has a limited amount of time available, it is desirable to not include this move into the search. By using this and other domain knowledge it is possible to apply forward pruning to the tree in order to remove these undesirable branches. Now follows a list of moves which can be used as domain-specific knowledge or heuristics in the pruning process:

Moves to be pruned or investigated with a reduced depth:

- Moves which lead to losing a piece: it would seem rational to protect your own pieces.

- Moves which reveal (important) information: moves which give away the position of the Spy or Marshall are, for instance, highly undesirable.

Moves to be investigated further or investigated with more plies:

- Moves which lead to the capture of an opponent's piece: almost any capture of an opponent's piece is a good move.

- Moves which reveal information about an opponent's piece: any gain of information is beneficial and makes searching the game tree later on easier.

- Moves which trap enemy pieces or block paths: if it becomes possible to block the opponent's way with a high-ranked piece (or by bluffing) this is mostly useful.

These are some basic examples, but already some interesting issues arise. Some of these situations may become more valuable or less valuable during the course of the game. For instance: gaining information is important at the start of a game. Most of the opponent's pieces are unknown and thus it is important to gain information as soon as possible. This may conflict with the undesirable situation of losing pieces. A Scout used to scout may be lost during its quest for information.

Therefore it is needed to use the information available as a weight in the decision to prune or not.

4.2 Domain-Dependent Forward Pruning

In order to apply forward pruning on a game tree, it is needed to know what undesirable moves actually are. This is where domain knowledge should be used. By using heuristics provided by the domain knowledge, the search is able to prune undesirable moves. The knowledge makes it possible to recognize a bad move early on in the branch and stop the search there. If the algorithm would not use domain knowledge, this branch would be classified as undesirable later in the search. This results in lost time, which could be used to search further through more promising branches.

Human players of Stratego use a great deal of game specific knowledge when they play. This knowledge should be used for forward pruning. Table 3 shows several move types which are potentially bad. These moves should be pruned or at the least explored with fewer plies. The second column shows the reduction depth.

Move	Reduction depth
losing a piece	0.5
revealing a piece to the opponent	0.5
losing the last Miner	1.0
losing the Spy	1.0
losing the Marshall or General	1.0

Table 3: Undesirable move types and their reduction depths

5 Experiments

In this section the experiments regarding the evaluation function and the forward pruning will be discussed. The program which is used for testing uses several search methods to improve the search through the game tree. These methods are: transposition tables [15], the history heuristic [12], the killer heuristic [1], aspiration search [11] and null-move [3]. Finally, a random factor is used to detect mobility [4]. These methods will be used during the tests by both the side using domain knowledge and the side without domain knowledge.

5.1 Evaluation Function

The evaluation function consists of several factors. The goal is to decide which factors provide a better algorithm. To test if a certain factor improves the algorithm and to globally see how large its effects are, self-play is performed. Both sides will use the same

basic evaluation function, as discussed in Section 3. One side will use an added evaluation factor. A simulation is performed, in which a large number of games is played. The number of games won by each player and their respective win percentage is recorded to determine the performance of each player.

Now follow the factors which are different for both sides. The side which uses the evaluation function with domain knowledge, uses these factors, the other side does not. The factors which will be tested are:

- Miners captured: the value of the position will be lowered for each Miner lost during the game, making it less favorable. The penalty will be larger if there are no Miners left and the opponent still has Bombs on the field, regardless of whether their position is known.

- Spy captured: the value of the position is decreased if the Spy is lost and the opponent still has his Marshall. If the own Marshall is still on the field the penalty will be smaller, because the opponent's Marshall can still be captured, although at the cost of the player's own Marshall.

- Near enemy Flag: the value is increased if the pieces are near the Flag of the opponent. The enemy Flag will always remain unknown, because technically the Flags position is only revealed when it is captured, thus ending the game. Therefore, the distance from friendly pieces to any unknown piece on the last row is calculated. The shortest distance from a friendly piece to any unknown enemy piece is taken as the distance of that piece to the enemy Flag. The closer a piece is, the larger the bonus for the evaluation function.

- Enemy near own Flag: the value is descreased if there are enemy pieces near the own Flag. The distance of any enemy piece to the Flag is calculated, whether the piece is known or not. If the distance is shorter than a certain threshold the position receives a penalty. The threshold is used because not all pieces are an immediate threat to the Flag. For instance, a piece on the other side of the board is not dangerous.

- Moving towards the enemy: this is mainly done to enable long-term planning. The idea is to move pieces which are one rank higher than a known piece of the enemy towards this enemy piece. This will ultimately lead to the capture of this piece. Figure 1 shows an example of the red General (white circle) moving closer to a blue Colonel (black circle) in order to capture him. The closer the General is to the Colonel, the higher the bonus for the evaluation.
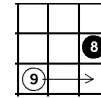

Figure 1: Example of moving towards a piece which is one rank lower

- Moving Scouts to unknown pieces: the Scouts are used to discover the ranks of unknown enemy pieces. By using the low-ranked Scout to determine the ranks of the enemy pieces, casualties of higher pieces can be avoided. After an enemy piece is revealed, higher-ranked pieces can possibly capture it. Figure 2 shows an example of using the Scout to reveal an unknown enemy piece. By moving close to the piece, the opponent is forced to capture the Scout or the Scout can eventually be used to attack the enemy piece to reveal its rank. The closer the Scout is to the unknown piece, the higher the bonus for the evaluation.
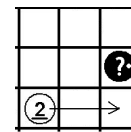

Figure 2: Example of scouting

- Blocking paths: the goal of this strategy is to make sure that enemy pieces can not cross the area next to the lakes. The passages on either side of the lake are small. By placing a high-ranked piece on one of these fields, it is possible to prevent the enemy from passing with lower ranks. With high ranks, all pieces with rank Lieutenant or higher are meant. However, the bonus awarded for placing such a rank on one of these positions should not be too high. Otherwise, all the higher ranks might get stuck in the center, not moving towards the enemy side anymore. Figure 3 shows an example of a blocked path. The blue General (black circle) prevents the blue Sergeant (white circle) from moving upwards, otherwise the Sergeant will be captured.
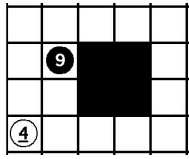
Figure 3: Example of a blocked path

Table 4 shows the results of implementing these factors. The results are given in percentages won by using only the basic evaluation function versus the modified evaluation function. Each factor has been tested separately. The last row covers the results from combining all factors which had a positive influence on the win percentages. The percentages are calculated by playing 1,000 games and counting the wins and losses.

Eval. Factor	Basic eval.	Factor added	Draw
Miners captured	46.2	53.8	0.0
Spy captured	35.4	64.6	0.0
Near enemy Flag	54.3	45.7	0.0
Near own Flag	42.3	57.5	0.2
Near enemy	41.2	58.7	0.1
Scouts	42.1	57.8	0.1
Blocking	42.2	57.6	0.2
All factors	61.4	38.6	0.0

Table 4: Experimental results for different evaluation factors

From the experiments it can be concluded that using domain knowledge to construct an evaluation function provides a considerable benefit. Almost each individual evaluation factor tested yielded a higher win percentage versus an evaluation function not using these factors. An analysis of the performance of each factor will now be given.

As shown by the experiments, keeping Miners alive will give a small increase in the number of victories. The relatively low change can be attributed to a too large emphasis on keeping these pieces alive. If the penalty is too high, using them may become undesirable. It is also important to realize that Miners are not necessary for achieving victory. They will often be needed to reach the enemy Flag, but the game can also be won by making sure the enemy can not make any more moves. In the last case Miners are not as important as in the first case. The number of victories can thus be improved by tuning the evaluation factor.

The same basically holds for the Spy, the amount of emphasis placed on keeping it alive is an important part of the performance of this factor. The better performance of keeping the Spy alive can be attributed to the 'back-up plan' of the factor. This back-up plan consists of using the Marshall to capture the enemy Marshall. So even if the Spy is lost, the penalty will not be too great, which does not restrict the option the player has during the game.

The next two factors, own pieces near the supposed location of the enemy Flag and enemy units near the own Flag, are fairly similar. The distance to the location of the Flags is important in both factors. But here also lies the significant difference between the two factors. While the location of the own Flag can be easily determined, it is harder and perhaps impossible at points in the game to determine the position of the enemy Flag. This crucial uncertainty explains the difference in performance between these factors. A good method to determine the location of the enemy Flag is needed in order to use the second factor successfully.

The near-enemy factor has shown to work as well. By moving pieces to an enemy piece with one rank lower, minimal information is given to the opponent, while still capturing one of his pieces. This factor could be modified to reward pieces close to enemies with a rank difference of more than one. The main goal of this factor, to enable some sort of planning by giving a reward based on the distance, works well. This factor can be applied to other strategies as well, to move Miners to known Bombs by rewarding them based on their distance to the Bomb.

The Scout factor works basically in the same way. The success of the Scout can be explained by the need to gather information. By identifying enemy pieces, it becomes possible to capture them. Since the Scout is a low rank, the loss will generally not be large, but the information gained by scouting may be significant.

The blocking factor also performs well. By blocking a player can prevent the opponent from reaching certain areas of the board, effectively protecting that area. The reward for keeping high pieces on the fields next to the lakes should not be too high. Otherwise, these pieces will not be moved from these positions, excluding them from further play.

Finally, the combination of all positive factors will be discussed. This combination performs poorly. This is because some factors contradict each other. For instance, the blocking factor tries to keep high-ranked pieces near the lakes, while the near-enemy factor moves them towards the enemy. In order to combine all factors tuning is again needed. The factors have to be adjusted to complement each other, instead of being contradicting.

One advantage of using domain knowledge as a guideline for the evaluation function is that many strategies can be implemented fairly straightforward. A disadvantage however is that some strategies may work against each other, for instance: keeping the own pieces hidden, while still trying to capture enemy pieces. Another disadvantage is that it may not be obvious which values

and weights should be assigned to the separate factors of the evaluation function. For instance, putting too much emphasis on blocking paths may result in pieces being forced to stay on certain positions, while this is not the best possible option overall.

To improve the performance of the evaluation function, one should find the optimal values for each factor in this function. By placing the emphasis on the right factors, while still making sure that other factors are taken into account as well can prove to be a challenge. Thus, further tuning and balancing is highly recommended in order to increase the performance of the evaluation function. Secondly, there is much more domain knowledge to be used as possible factors for an evaluation function. This paper has only used a small amount of the knowledge available. By cooperating with top-level human players, more domain knowledge and thus more factors can be gathered.

5.2 Forward Pruning

The goal of the second series of experiments is to test the use of domain knowledge as a guide for the search. In order to perform this test, the move types listed in Table 3 will be searched with the corresponding depth reduction. The performance will be measured by the percentage of wins of the search method with forward pruning versus the search method without pruning. The measurements will again be performed over 1,000 games. The results are shown in Table 5. The last row shows the combination of all positive prunings.

Pruned move	Pruning	No pruning
Losing a piece	36.3	63.7
Revealing a piece	34.5	65.5
Losing last Miner	50.7	49.3
Losing Spy	51.8	48.2
Losing Marsh./Gen.	52.4	47.6
All moves	43.8	56.2

Table 5: Experimental results for different pruned moves

The success rate of the forward pruning is considerably lower than that of the evaluation function. The cause of this problem with forward pruning is that it is not immediately possible to determine the amount with which the search depth should be reduced. If the depth is reduced too much, potentially good branches will be removed. If the depth is reduced with an amount that is too small, the time gained by pruning will be too low, reducing the effectiveness of the search. Furthermore, most of the experiments are based on pruning moves that lead to the loss of certain pieces. It may sometimes be beneficial to sacrifice pieces, for instance in order to obtain information about the opponent. By reducing the

search depth for moves which lose a piece, the search may be too shallow to ultimately find the benefit of the move. This applies even more to moves which reveal a piece to the enemy. Pruning moves which reveal pieces proves to be too restricting. In order words: strategies which require some losses at the beginning, but pay off in the end will not be correctly found by the search. This is especially the case when pruning all moves which lead to the loss of a piece or when pruning all moves which reveal pieces.

As can be seen in Table 5 the difficulties described above lead to a low win-percentage. If the focus lies on only pruning moves which leads to the loss of special pieces, the results are better, but the gain is still not as large as with the use of a modified evaluation function. Therefore the conclusion is, that domain-dependent forward pruning can potentially lead to better plays, but a good tuning is required in order to achieve this result.

6 Conclusions

It can be concluded that using domain knowledge to construct an evaluation function can increase the performance of a Stratego program. The results depend on the factors being evaluated and on how these factors are tuned. Factors which depend on distances between pieces need a reliable way to determine these distances, especially when the location of one of the pieces being evaluated is unknown. One way to solve this problem could be to use pattern recognition to more precisely determine the location of the missing piece. When using several factors at once, it is important to find a balance between conflicting factors. Further research concerning domain-dependent evaluation functions could focus on tuning the factors investigated in this paper or finding other evaluation factors.

Using domain knowledge as a guide for searching and specifically for forward pruning can increase the performance of a Stratego program as well. However, the effects of using forward pruning are small as can be seen in Section 5.2. Reducing the search depth of certain move types too much removes possibly good branches from the tree, which leads to desirable moves being missed in the search. This translates to lower win-percentage. In order to improve the forward pruning, the depth reduction needs to be tuned with a correct number of plies.

Using domain-dependent knowledge has proven to be useful to improve a Stratego program. The evaluation function with domain knowledge outperforms the domain-dependent forward pruning. The domain knowledge can also be converted into an evaluation function

with relative ease. Both improvement methods need tuning, which can prove to be a challenge in the case of forward pruning.

References

[1] Akl, S.G. and Newborn, M.M. (1977). The principal continuation and the killer heuristic. *1977 ACM Annual Conference Proceedings*, pp. 466–473.

[2] Anantharaman, T., Campbell, M.S., and Hsu, F. (1990). Singular extensions: Adding selectivity to brute-force searching. *Artificial Intelligence*, Vol. 43(1), pp. 99–109.

[3] Beal, D.F. (1989). Experiments with the null move. *Advances in Computer Chess 5* (ed. D.F. Beal), pp. 65–89, Elsevier Science Publishers B.V.

[4] Beal, D.F. (1994). Random evaluations in chess. *ICCA Journal*, Vol. 18 (2), pp. 3–9.

[5] Björnsson, Y. and Marsland, T.A. (2000). Risk management in game-tree pruning. *Information Sciences: an International Journal*, Vol. 122(1), pp. 23–41.

[6] Björnsson, Y. and Marsland, T.A. (2001). Multi-cut $\alpha\beta$-pruning in game-tree search. *Theoretical Computer Science*, Vol. 252 (1-2), pp. 177–196.

[7] Buro, M. (1995). Probcut: An effective selective extension of the alpha-beta algorithm. *ICCA Journal*, Vol. 18 (2), pp. 71–76.

[8] Boer, V. de (2007). Invincible: A stratego bot. M.Sc. thesis, Delft University of Technology, the Netherlands.

[9] Knuth, D.E. and Moore, R.W. (1975). An analysis of alpha beta pruning. *Artificial Intelligence*, Vol. 6 (4), pp. 293–326.

[10] Lim, Y.J. and Lee, W.S. (2006). Properties of forward pruning in game-tree search. *Proceedings of the National Conference on Artificial Intelligence*, pp. 1020–1025, AAAI Press.

[11] Lu, H. and Xia, Z. (2008). AWT: Aspiration with timer search algorithm. *Computers and Games, CG2008* (eds. H. J. van den Herik, X. Xu, Z. Ma, and M. H. M.Winands), Vol. 5131 of *Lecture notes in Computer Science*, pp. 264–274, Springer-Verlag, Berlin, Heidelberg.

[12] Schaeffer, J. (1983). The history heuristic. *ICCA Journal*, Vol. 6 (3), pp. 16–19.

[13] Treijtel, C. (2000). Multi-agent stratego. M.Sc. thesis, Delft University of Technology, the Netherlands.

[14] Turing, A.M. (1953). Digital computers applied to games. *Faster than thought* (ed. B.V. Bowden), Pitman Publishing, London.

[15] Veness, J. and Blair, A. (2007). Effective use of transposition tables in stochastic game tree search. *IEEE Symposium on Computational Intelligence and Games, 2007 (CIG 2007)*, pp. 112–116.

[16] Winands, M. H. M., Herik, H. J. van den, Uiterwijk, J. W. H. M., and Werf, E. C. D. van der (2006). Enhanced forward pruning. *Information Sciences*, Vol. 175 (4), pp. 315–329.