A knowledge-based approach to Domineering

Department of Knowledge Engineering, Maastricht University, Bachelor Year 3

Florian van Daalen

June 20, 2012

Abstract

Domineering is a combinatoriall two-player game for which a large number of boards has been solved. This knowledge will be used in a number of approaches to create AI's capable of playing the game. In order to compare the resulting AI's two base cases, a random player and a player based on a very simple evaluation function, are used. All the AI's then perform in a tournament in order to determine which is the best. The results show that the resulting AI's with the current, limited, knowledge available are not good enough.

Keywords: Domineering, Knowledge-based approach, alpha-beta search, Subgames

1 Introduction

Domineering is a combinatoriall two-player game played on a board consisting of small squares. The shape of the board can vary from squares to an entirely irregular polygon or any combination thereof. However in this paper only rectangular boards will be considered. Both players have a selection of dominoes which they can place on the grid in turn, one player placing his dominoes in a horizontal orientation, the other in a vertical orientation. In this way the grid is slowly filled until one of the players becomes incapable of making a move during his turn. Whichever player is the first to find he has no valid moves left loses.

Domineering has, in the past, been thoroughly looked at. The game has been solved for a large number of boards [4][7]. The goal of this paper is to attempt to apply this knowledge about solved games on the subgames that naturally appear while playing. Doing this might result in an algorithm capable of playing Domineering.

1.1 Structure of the paper

In Section 2 a number of different approaches to the use of the knowledge will be presented together with their resulting AI's. Section 3 will discuss some improvements. Section 4 will contain a comparison between the different AI's based on tournaments held on a number of different board sizes. Finally in Section 5 a conclusion will be made as to how useful the applied knowledge is when playing Domineering, and if perhaps other methods would be more promising.

2 AI's

There are a multitude of approaches possible for applying the knowledge about solved games to the subgames in a larger game. To start with, one must think of a way to compare different moves to each other. A good approach to determine the strength of a move in any two-player game is the alpha-beta algorithm [6] which all the following AI's use as their base. The knowledge about solved games is then applied to the subgames one can observe in a larger game and the resulting derivations are then combined into something applicable to the larger game as a whole.

2.1 Subgames

In order to be able to apply the knowledge about solved games to subgames one must first define what exactly a subgame entails as the definition can put restrictions on the performance of the algorithm. For example, any single empty square on the board could be defined to be a "subgame", however looking at only one square at a time surely is not the best option. Furthermore, the absence of knowledge about non-rectangular subgames restricts the algorithm to only use rectangular " subgames". This also means that non-rectangular subgames need to be expressed in rectangular ones. "Subgames" are thus defined according to the following two rules:

1. A subgame is any possible rectangle that can be made around an empty square on the board without including occupied squares.
2. Any rectangle that is completely included in another subgame is not a subgame.

Figure 1 shows an example configuration on a 6×6 board, Figures 2 to 6 show all subgames of the configuration.

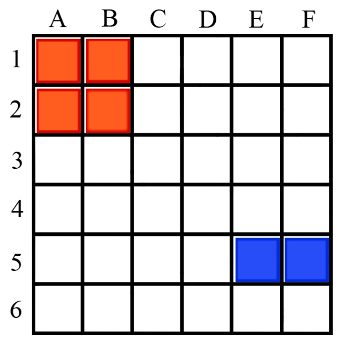

Figure 1: A configuration on a 6 × 6 board

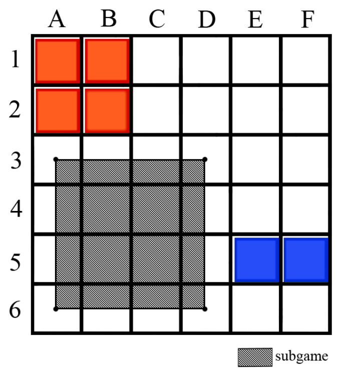

Figure 4: Subgame 3, 4 × 4 with value: 1

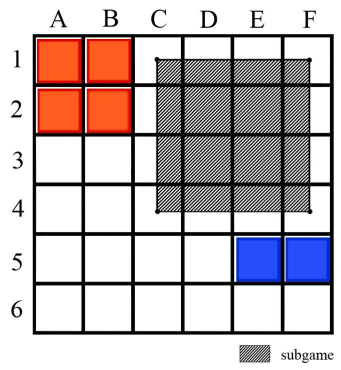

Figure 2: Subgame 1, 4 × 4 with value: 1

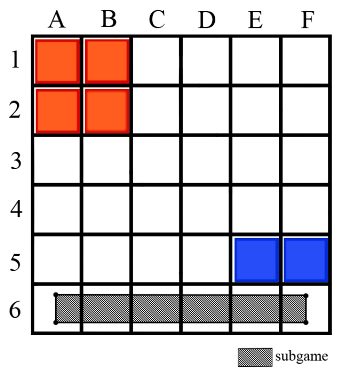

Figure 5: Subgame 4, 1 × 6 with value: H

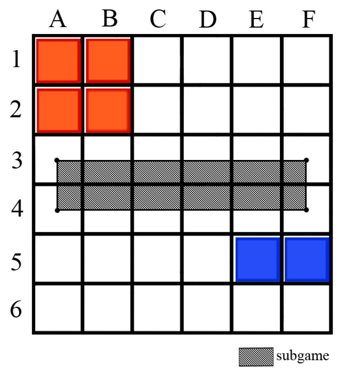

Figure 3: Subgame 2, 2 × 6 with value: 1

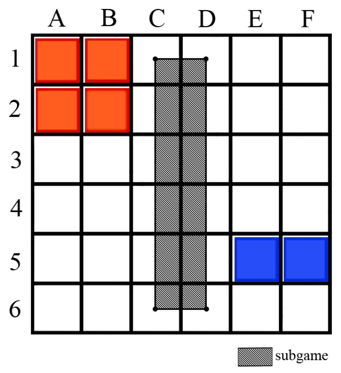

Figure 6: Subgame 5, 6 × 2 with value: 1

These subgames can then be valued according to the following scheme [7] [3]:

1. First player win: 1
2. Second player win: 2
3. Horizontal player win irrespective of who starts: H
4. Vertical player win irrespective of who starts: V

2.2 Control-AI's

In order to compare the AI's to each other one first needs a base case. After all, just comparing the AI's amongst themselves might give a distorted image of how effective they really are given that the material they are compared with might be significantly worse or better than a normal player. Because of this two simple base cases were made. The first base case is a random AI, which as the name implies plays randomly. The second AI is an alpha-beta based AI called the orderingbot.

Orderingbot

The evaluation function of OrderingBot is given by the following formula[3]:

value = *(real moves player)* − *(real moves opponent)* + *(safe moves player)* − *(safe moves opponent)*

The safe moves are the moves one player is able to make regardless of what the opponent will do. The real moves are the total number of moves that are actually possible for the player, assuming the opponent does nothing, given the current board. The higher the resulting value is, the better the move.

It is considered as a base case AI because this value used for move ordering has a large effect on pruning in an alpha-beta search [3], implying that it orders strong moves relatively high. It also owes its name to the use of the evaluation function as a move ordering.

2.3 AI's

All of the following AI's use the knowledge about solved subgames [7] [3] in order to determine if a subgame is a win or a loss. To give indications of the difference between the bots the values for the horizontal and vertical player for the configuration shown in Figure 1 will be given for the deterministic bots. This is assuming that that player is the first player. These scores give an indication for which player this board would be more favorable, whichever player gets the highest score finds this board the most favorable assuming he is the first player.

TotalVoteBot

TotalVoteBot uses the following evaluation function:

1. For all empty squares on the board give them a score according to the following model:
   (a) +1 for every subgame they are in that is a win for the evaluating player.
   (b) −1 for every subgame they are in that is a loss for the evaluating player.
2. Sum the scores of all empty squares to get the score of the complete board.

Vertical value: 50, Horizontal value: 62

MajorityBot

1. For all empty squares on the board give them a score according to the following model:
   (a) Give it *one win vote* for every subgame they are in that is a win for the evaluating player.
   (b) Give it *one loss vote* for every subgame they are in that is a loss for the evaluating player.
   (c) The final score is equal to 1 if there are more *win votes* than *loss votes*, −1 in case of the opposite.
   (d) In case of a draw, we arbitrarily assume it is a win[1].
2. Sum the scores of all empty squares to get the score of the complete board.

Vertical value: 26, Horizontal value: 30

ProportionalBot

1. For all empty squares on the board give them a score according to the following model:
   (a) $+1/n$ for every subgame they are in that is a win for the evaluating player, where n is the number of subgames this square is a part of.
   (b) $-1/n$ for every subgame they are in that is a loss for the evaluating player, where n is the number of subgames this square is a part of.
2. Sum the scores of all empty squares to get the score of the complete board.

Vertical value: $23\frac{1}{3}$, Horizontal value: 30

ProportionalOverlapBot

This bot applies a penalty based on the amount of overlap there is between subgames in an attempt to minimize the amount of overlap. This is based on the notion that the more overlap there is the more uncertain it becomes that the values of the rectangular subgames are representing the true value of the game.

1. For all empty squares on the board give them a score according to the following model:
   (a) $+1/n$ for every subgame they are in that is a win for the evaluating player, where n is the number of subgames this square is a part of.

[1] In hindsight it would've been better to give this a score of 0 as this would most likely improve the AI. However the expectation is that this improvement would not be large.

   (b) $-1/n$ for every subgame they are in that is a loss for the evaluating player, where n is the number of subgames this square is a part of.

2. Sum the scores of all empty squares to get the total score.
3. For each square in more than one subgame add $+1$ for each subgame more than one to the penalty.
4. Substract the penalty from the total score to get the final score.

Vertical value: $-8\frac{2}{3}$, Horizontal value: -2

ProportionalSubgameOnlyBot
Similar to ProportionalOverlBot this bot gives a penalty based on the amount of overlap between subgames, However it only looks at subgames and not at individual squares on the board.

1. For all subgames that are present on the board increase the total score as follows:
   (a) $+1/n$ for every subgame that is a win for the evaluating player, where n is the number of subgames.
   (b) $-1/n$ for every subgame they are in that is a loss for the evaluating player, where n is the number of subgames.
2. For each square in more than one subgame add $+1$ to the penalty for each subgame more than one.
3. Substract the penalty from the total score to get the final score.

Vertical value: -31.2, Horizontal value: -31

ProportionalAbsoluteBot

1. For all empty squares on the board give them a score according to the following model:
   (a) For every subgame that has a length and width of more than one:
      i. $+1/n$ for every subgame they are in that is a win for the evaluating player, where n is the number of subgames this square is a part of.
      ii. $-1/n$ for every subgame they are in that is a loss for the evaluating player, where n is the number of subgames this square is a part of.
   (b) For every subgame that has a length or width of one:
      i. $+1/n \times \gamma$ for every subgame they are in that is a win for the evaluating player, where n is the number of subgames this square is a part of. γ is a preset weight.
      ii. $-1/n \times \gamma$ for every subgame they are in that is a loss for the evaluating player, where n is the number of subgames this square is a part of. γ is a preset weight.
2. Sum the scores of all empty squares to get the score of the complete board.

Vertical value: $26\frac{1}{3} - 3\frac{2}{3}\gamma$, Horizontal value: $26\frac{1}{3} + 3\frac{2}{3}\gamma$

ProportionalAbsoluteSwapBot
This AI uses the same evaluation function as ProportionalAbsoluteBot. However, normally in alpha-beta search if one encounters two possible moves with the same value, one picks the move that occurs first in the move list. This AI has a small random chance of picking the second move if it encounters a move that is equally good as the current best move.

ProportionalAbsoluteRandomBot
This AI is a combination of the RandomBot and the ProportionalAbsoluteBot. Each time it has to decide on making a move it will randomly pick one of the two AI's to decide for it. This option was considered because RandomBot in preliminary testing often got long lucky streaks where it would make multiple decent moves in a row. This led to the conclusion that a small chance of doing something random would improve the AI as a whole and make it less predictable [1].

OrderedAbsoluteSwapBot
This AI uses the same evaluation function as ProportionalAbsoluteSwapBot, however it uses the move ordering that OrderingBot uses as its evaluation function. This is done with the expectation that the move ordering would improve the pruning in the alpha-beta search, possibly leading to a better end result.

OrderingAbsoluteSwapBot
This is a combination of ProportionalAbsoluteSwapBot and OrderingBot. ProportionalAbsoluteSwapBot occasionally makes bad moves, such as placing a domino against the side of the playing field instead of one row or column away from it, whereas OrderingBot would never do that. On the other hand OrderingBot has trouble distinguishing between certain moves because in its eyes they are equivalent. By combining the two it is hoped that these problems are solved. They are combined in the following way:

1. Calculate the score Naccording to OrderingBot
2. Calculate the score Maccording to ProportionalAbsoluteSwapBot
3. The final score is: $\alpha \times N + \beta \times M$, where α and β are preset weigths

3 Improvements

After the initial tests some improvements were made in hopes of increasing the performance of the eleven AI's.

3.1 Nondeterminism

After the initial tests it became clear that OrderingBot, although potent in comparison to the others, was overall one of the easier bots to play against. Due to its determinism and simple evaluation function it is relatively easy for a human to use a strategy to counter it. Furthermore, given its simplicity it has a tendency of giving the same score to a multitude of moves. For example: when playing on an 8×8board, for its opening move as horizontal, any move that is in the second or seventh row and does not leave an odd number of squares empty above or below it has an equivalent score. All of these moves would be equivalently good opening moves. This leads to the following change in the evaluation function:

$value = (real\ moves\ opponent) - (real\ moves\ player) + (safe\ moves\ opponent) - (safe\ moves\ player) + \lambda$

where λ is a random value from zero to one. This would allow the bot to become nondeterministic making it a more capable opponent against humans. Furthermore, although moves that used to be equivalent will no longer be, moves that were deemed better than others will not suddenly become worse due to this addition. This is due to the fact that if move A is better than B its score will be at least one higher and the random factor is unable to increase the score of B by enough to outrank A.

For similar reasons this random factor was also applied to TotalVoteBot and MajorityBot. However it was not applied to the variations that used a proportional-vote approach as these do not have integer scores, and thus lack the clear classes in their scoring model of equivalent configurations.

Furthermore, this nondeterminism meant all AI's, with the exception of ProportionalBot and the bots using overlap, have a random element in them. This makes the tournaments more representable of a real situation.

4 Experiments and Results

In the following section the experiments and their results will be discussed.

4.1 Initial experiments

In order to test the different AI's a number of tournaments were held, not including the improvements mentioned in Section 3. In these tournaments all bots played 10 matches against each other, both in the role as first and as second player. Each bot has slight variations with respect to search depth, and any other possible variable. Furthermore, the tournaments were held on 5×5, 6×6 and 8×8 boards. For both the 5×5 and 6×6 board 45 different bots were tested. For the 8×8 board the amount of bots was limited to 11. Summaries of the results of the tournament on the 5×5 board and the 8×8 board can be found in tables 1 and 2, respectively. The 6×6 results have been excluded from this paper due to the similairity with the 5×5 results. Further results can be requested from the author. In all cases the vertical player is the first player. Furthermore, it should be noted that no AI can ever win all matches. This is due to the fact that it cannot win the matches against itself as both the first and second player. Because of this any AI will lose at least 10 matches in each tournament. The following parameters where varied for each AI:

1. Search depth
2. Absolute: the weight given to the subgames with a width or height of one by ProportionalAbsoluteBot, ProportionalAbsoluteSwapBot, OrderedAbsoluteSwapBot, OrderingAbsoluteSwapBot and ProportionalAbsoluteRandomBot.
3. Threshold: The chance of doing a random move or choosing an equivalent move by ProportionalAbsoluteRandomBot, ProportionalAbsoluteSwapBot, OrderedAbsoluteSwapBot and OrderingAbsoluteSwapBot.
4. Absolute weight, Ordering weight: The weights used by OrderingAbsoluteSwapBot.

Discussion

Rapidly it became clear that TotalVoteBot, MajorityBot, ProportionalSubgameOnlyBot and ProportionalOverlapBot were outclassed by the other AI's. All of them were barely capable of standing up against even the random AI. Furthermore, OrderedAbsoluteSwapBot had no significant difference in the results compared to ProportionalAbsoluteSwapBot. This indicates that the mere use of a better ordering did not result in better pruning as hoped. ProportionalAbsoluteBot, ProportionalAbsoluteSwapBot, ProportionalAbsoluteRandomBot and OrderingAbsoluteSwapBot consistently occupied the upper half of the rankings, with OrderingAbsoluteSwapBot tending to outperform the others in most cases. However, depending on the board size, the internal ranking could differ. Furthermore, one should note that OrderingBot, although overall rarely ranked first, had a tendency of outperforming the best AI when only compared to that one.

Another thing worth noting is that the search depth of the alpha-beta search did not seem to have a consistently positive, or even negative, effect on the performance of the algorithm. The winner of the 6×6 tournament, for example, was a variation of OrderingAbsoluteSwapBot with a search depth of 1. However, the same variation but with a search depth of 2 resulted in the second worst AI in this tournament, and a depth of 3 and 4 ended somewhere in the middle of the rankings. A similar lack of consistency can be observed with the

AI	Vertical win/total	Horizontal win/total	total win/total
ProportionalOverlapBot, depth: 1	40/450	30/450	70/900
ProportionalSubGameOnlyBot, depth: 1	43/450	41/450	84/900
MajorityBot depth: 1	50/450	87/450	137/900
OrderingAbsoluteSwapBot, depth: 2, absolute: 3, treshold: 0.05, weigth absolute: 1, weigth order 5	29/450	111/450	140/900
TotalVoteBot, depth: 1	61/450	90/450	151/450
TotalVoteBot, depth: 2	33/450	132/450	165/900
MajorityBot, depth: 2	28/450	153/450	181/900
ProportionalBot, depth: 2	30/450	163/450	193/900
OrderingBot, depth: 2	62/450	140/450	202/900
ProportionalAbsoluteSwapBot, depth: 2, absolute: 2, treshold: 0.05	58/450	146/450	204/900
ProportionalAbsoluteRandomBot, depth: 2, treshold: 0.05, absolute: 2	60/450	162/450	212/900
OrderedAbsoluteSwapBot, depth: 2, absolute: 2, treshold: 0.05	56/450	157/450	213/900
ProportionalOverlapBot, depth: 2	43/450	203/450	246/900
ProportionalSubGameOnlyBot, depth: 2	132/450	121/450	253/900
ProportionalAbsoluteBot, depth: 2, absolute: 3	75/450	180/450	255/900
RandomBot	127/450	180/450	307/900
ProportionalBot, depth: 4	96/450	280/450	376/900
ProportionalOverlapBot, depth: 4	120/450	286/450	406/900
MajorityBot, depth: 3	218/450	205/450	423/900
ProportionalOverlapBot, depth: 3	167/450	258/450	425/900
OrderingAbsoluteSwapBot, depth: 4, absolute: 3, treshold: 0.05, weigth absolute: 1, weigth order 5	202/450	231/450	433/900
MajorityBot, depth: 4	206/450	228/450	434/900
OrderingBot, depth: 4	214/450	222/450	436/900
TotalVoteBot, depth: 3	223/450	231/450	454/900
OrderedAbsoluteSwapBot, depth: 4, absolute: 2, treshold: 0.05	208/450	257/450	465/900
TotalVoteBot, depth: 4	245/450	237/450	482/900
ProportionalAbsoluteSwapBot, depth: 4, absolute: 2, treshold: 0.05	220/450	262/450	482/900
ProportionalAbsoluteRandomBot, depth: 4, treshold: 0.05, absolute: 2	201/450	285/450	486/900
ProportionalSubGameOnlyBot, depth: 3	271/450	147/450	518/900
ProportionalAbsoluteBot, depth: 4, absolute: 3	249/450	213/450	564/900
ProportionalSubGameOnlyBot, depth: 4	246/450	319/450	565/900
OrderedAbsoluteSwap, depth: 3, absolute: 2, treshold: 0.05	362/450	254/450	616/900
ProportionalAbsoluteSwapBot, depth: 3, absolute: 2, treshold: 0.05	277/450	340/450	617/900
OrderingAbsoluteSwapBot, depth: 3, absolute: 3, treshold: 0.05, weigth absolute: 1, weigth order 5	308/450	345/450	653/900
ProportionalAbsoluteRandomBotBot, depth: 1, treshold: 0.05, absolute: 2	339/450	351/450	690/900
ProportionalAbsoluteSwapBot, depth: 1, absolute: 2, treshold: 0.05	344/450	365/450	709/900
OrderedAbsoluteSwapBot, depth: 1, absolute: 2, treshold: 0.05	333/450	380/450	713/900
OrderingAbsoluteSwapBot, depth: 1, absolute: 3, treshold: 0.05, weigth absolute: 1, weigth order 5	358/450	399/450	757/900
ProportionalAbsoluteBot, depth: 1, absolute: 3	377/450	386/450	763/900
OrderingBot, depth: 1	373/450	398/450	771/900
ProportionalBot, depth: 1	384/450	389/450	773/900
ProportionalAbsoluteBot, depth: 3, absolute: 3	326/450	338/450	774/900
ProportionalAbsoluteRandomBot, depth: 3, treshold: 0.05, absolute: 2	376/450	429/450	805/900
OrderingBot, depth: 3	358/450	450/450	808/900
ProportionalBot, depth: 3	389/450	450/450	839/900

Table 1: Summary of initial results for the 5×5 board

AI	Vertical win/total	Horizontal win/total	total win/total
TotalVoteBot, depth: 1	20/110	0/110	20/220
MajorityBot, depth: 1	30/110	0/110	30/220
ProportionalSubGameOnlyBot, depth: 1	30/110	10/110	40/220
ProportionalOverlapBot, depth: 1	40/110	30/110	70/220
ProportionalBot, depth: 1	73/110	42/110	115/220
ProportionalAbsoluteBot, depth: 1, Absolute : 3	87/110	42/110	129/220
ProportionalAbsoluteRandomBot, depth: 1, treshold: 0.05, absolute: 2	84/110	50/110	134/220
OrderedAbsoluteSwapBot, depth: 1, absolute: 2, treshold: 0.05	84/110	55/110	139/220
ProportionalAbsoluteSwapBot, depth: 1, absolute: 2, treshold: 0.05	92/110	58/110	150/220
OrderingBot, depth: 1	100/110	91/110	191/220
OrderingAbsoluteSwapBot, depth:1, absolute: 3, treshold: 0.05, weigth absolute: 1, weigth order: 5	99/110	93/110	192/220

Table 2: Summary of initial results for the 8×8 board

other AI's. A possible explanation for this phenomenon would be that many configurations have similar, if not the same score. Furthermore, making move A first, then move B has the same result as when one would first make move B followed by A. This may lead the different AI's to evaluate two moves as equivalent when in reality they are not. The only exception to this rule was when the search depth allowed the AI to solve the remainder of the current game, in which case it gave a significant improvement. Although there is a lack of consistency, odd depths do seem to outperform even depths. This could be caused by the odd-even effect which occurs frequently in alpha-beta and mini-max searches for two-player games.

4.2 Final Experiments

For the final experiment the best values for the parameters as found in earlier tests were used. Due to the fact that odd depths seemed to outperform even depths, this test only had depth 1 and depth 3. Furthermore, the improvements discussed in Section 3 have been applied to the AI's. It was only run on an 8×8board. A summary of the results can be found in Table 3.

Discussion

As can be seen very clearly the OrderingBot with its improvements has become significantly better than the other AI's. Where earlier the difference between the top bots was only a few matches it now has won roughly ten percent more matches than its competitors. Other than that there has not been much change.

5 Conclusion

In the end using the knowledge about the results of subgames proved to be inefficient, as a simple evaluation function is capable of outperforming any of the proposed methods. A possible explanation for this is that many configurations of subgames have equal values. Furthermore overlap is not covered by this knowledge and attempts to manage this lack of information by using various voting schemes failed. This caused the AI's to make mistakes when there was a lot of overlap between different subgames. However, when the overlap was limited, some AI's such as the ProportionalBot and the ProportionalAbsoluteBot were fairly capable. This overlap is solely caused by the lack of knowledge about non-rectangular subgames, commonly refered to as snakes [8]. Since the value of these non-rectangular subgames is not known one has to make due with rectangular ones inevitably creating overlap and thus uncertainty. This indicates that if one could add knowledge about non-rectangular subgames, thus removing overlap between subgames, the AI's would be significantly improved. However, adequate knowledge about these subgames is currently unavailable as far as known. Further improvements could be made by using combinatorial game theory [5]. By using combinatorial game theory one could get more accurate values for the subgames than simply a win or a loss.

A final improvement to the knowledge could be made by taking into account the number of moves a subgame would take to win [2], as there could be a significant difference between a configuration where, for example, one has two subgames which both can be won in four moves, or a configuration with two subgames one of which can be won in five moves, the other in three. However, currently these two configurations could be given the same value by the evaluation functions used, as this is not taken into account.

AI	Vertical win/total	Horizontal win/total	total win/total
MajorityBot, depth: 1	40/220	0/220	40/440
ProportionalSubGameOnlyBot, depth: 1	40/220	0/220	40/440
TotalVoteBot, depth: 1	50/220	0/220	50/440
ProportionalOverlapBot, depth: 1	52/220	30/220	82/440
TotalVoteBot, depth: 3	70/220	30/220	100/440
MajorityBot, depth: 3	70/220	40/220	110/440
ProportionalSubGameOnlyBot, depth: 3	70/220	42/220	112/440
ProportionalOverlapBot, depth: 3	75/220	52/220	127/440
OrderedAbsoluteSwapBot, depth: 3, absolute: 2, treshold: 0.05	97/220	87/220	184/440
ProportionalAbsoluteSwapBot, depth: 3, absolute: 2, treshold: 0.05	103/220	97/220	200/440
OrderingAbsoluteSwapBot, depth: 3, absolute: 3, treshold: 0.05, weigth absolute: 1, weigth order 5	130/220	109/220	229/440
ProportionalBot, depth: 1	133/220	129/220	262/440
ProportionalAbsoluteBot, depth: 3, absolute: 3	147/220	134/220	281/440
ProportionalAbsoluteBot, depth: 1, absolute: 3	173/220	110/220	283/440
ProportionalAbsoluteRandomBot, depth: 1, treshold: 0.05, absolute: 2	174/220	124/220	298/440
OrderedAbsoluteSwapBot, depth: 1, absolute: 2, treshold: 0.05	178/220	121/220	299/440
ProportionalAbsoluteSwapBot, depth: 1, absolute: 2, treshold: 0.05	177/220	136/220	313/440
ProportionalBot depth: 3	187/220	132/220	319/440
OrderingAbsoluteSwapBot, depth: 1, absolute: 3, treshold: 0.05, weigth absolute: 1, weigth order 5	189/220	170/220	359/440
ProportionalAbso- luteRandomBot, depth: 3, treshold: 0.05, absolute: 2	189/220	173/220	362/440
OrderingBot, depth: 1	197/220	190/220	387/440
OrderingBot, depth: 3	196/220	207/220	403/440

Table 3: summary of final results for the 8x8 board

Furthermore, the various AI's seem to perform at their best on shallow depths, with the exceptions of depths that allow them to solve the remainder of the game. If this also holds with the addition of non-rectangular subgames this would be a significant advantage given the exponential growth in time needed when searching deeper.

In conclusion, as it stands now methods using the knowledge about subgames are inferior to the simple evaluation function. However with the possible addition of knowledge about differently shaped subgames, more accurate knowledge provided by using combinatorial game theory to elaborate on the current values, and the addition of knowledge about the amount of moves required to win a subgame it is clear that there is room for further improvement.

References

[1] Beal, D. and Smith, M.C. (1994). Random evaluations in chess. *ICCA*, Vol. 17, No. 1, pp. 3–9.

[2] Berlekamp, E.R., Conway, J.H., and Guy, R.K. (2001-2004). *Winning Ways for your Mathematical Plays*, Vol. 1-4. A K Peters, Wellesley, MA.

[3] Breuker, D.M., Uiterwijk, J.W.H.M., and Herik, H.J. van den (2000). Solving 8 x 8 domineering. *Theoretical Computer Science* (Math Games), Vol. 230, pp. 195–206.

[4] Bullock, N. (2002). Domineering: solving large combinatorial search spaces. *ICGA*, Vol. 25, No. 2, pp. 67–84.

[5] Conway, J.H. (2001). *On Numbers and Games.* A K Peters, Natick, MA.

[6] Knuth, D.E. and Moore, R.W. (1975). An analysis of alphabeta pruning. *Artificial Intelligence*, Vol. 6, No. 4, pp. 293–326.

[7] Herik, H.J. van den, Uiterwijk, J.W.H.M., and Rijswijck, J. van (2002). Games solved: Now and in the future. *Artifical Intelligence*, Vol. 134, pp. 277–311.

[8] Wolfe, D. (1993). Snakes in Domineering games. *Theoret. Comput. Sci.* (Math Games), Vol. 119, pp. 323–329.