Internal Report 2010–14 September 2010

Universiteit Leiden
Opleiding Informatica

Calculating and Predicting

the Game

Five or More

Oswald A. de Bruin

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Calculating and Predicting the Game Five or More

Oswald de Bruin, #0301450

September 2, 2010

1 Introduction

For this paper we will examine a puzzle game called Five or More (a.k.a. *color lines*) in the field of artificial intelligence. The target we set here is to try to understand the way the game behaves, what the result is from certain actions taken in the game and ultimately finding a strategy that is based on knowledge of the game and can achieve a high total score.

The rules of the game will be explained in Section 2. We will look at the behaviour of Five or More in Section 3 while using the Monte Carlo algorithm and a simple strategy function on it. The Monte Carlo algorithm has shown to give nice results with other games like, for example, Same Game [6]. With the results from Section 3 we try to develop some more sophisticated strategies in Section 4 and see how those work out. Finally in Section 5 we will combine these results and strategies to develop an algorithm that is specialised on Five or More. Section 6 concludes our research.

The research will be quite difficult for a couple of reasons:

- **No previous research:** As far as we know there is no scientific research done on this game, so we have to come up with practically every strategy and technique ourselves, apart from the conventional ways to attack a game in the field of artificial intelligence.
- **No available benchmarks:** There are a lot of versions of the game Five or More with different sizes of the playfield, number of colours and ways of counting the score. None of the versions had available high-scores from players. For target scores we have to rely on people in the

author's direct environment and the scores achieved by the strategies described in this paper.

- **The random nature of the game:** FIVE OR MORE is a one player game with randomly generated situations through the game, making it unpredictable. This makes certain strategies, such as a full search tree with all possible states, practically impossible. For more on full search trees and other algorithms used on games, see [2] and [5].

When comparing calculation times for research like this, one should use only one kind of machine. The main calculating machine we used was a EEE laptop (netbook) with a 1.6GHz single core Intel processor and 1GB RAM. However, this was not sufficient for the more complicated strategies and for those we were committed to distribute the calculations over multiple, more powerful desktop computers with 3.16GHz dual core processors and 2GB RAM.

This paper is the Bachelor thesis of the author written at the Leiden Institute of Advanced Computer Science (LIACS), part of Leiden University. The writing of this paper is supervised by Walter Kosters and Frank Takes.

2 Five or More

In this section wel will explain the rules and hardness of the game.

2.1 Rules

FIVE OR MORE is a single player game. The player has a playfield, also called field or board, with $M \times N$ empty places, see Figure 1. The game starts with X randomly placed stones of a random colour. The game uses at most Y different colours. The player can make a move by moving a stone along a path of empty places. The stone is not allowed to jump over another stone and also can not make a diagonal path if the diagonal is not accessible by going horizontal and vertical. After every move, X new random stones will be put on random empty places on the playfield. The colours of these stones are known beforehand, but the places where they will be put are not.

If the player succeeds in making a horizontal, vertical or diagonal line of 5 or more stones of the same colour, then these stones are removed from the field and a score is calculated for the number of stones removed, hence the

Figure 1: Screenshot of FIVE OR MORE, courtesy of the Free Software Foundation

name FIVE OR MORE. That score is then added to the total score achieved so far. After scoring a line the player also gets the opportunity of what we will call a *free-play move*. In a free-play move the random stones that would be dropped are not put in the field. The player can make another move and the random stones will be dropped the next non-free-play move.

It is possible that the randomly dropped stones finish a line if a stone of the right colour drops in the right line. In this case the line will be removed and a score will be banked before the next move of the player begins, but the next move after that will still include new randomly dropped stones.

The reader is referred to Table 1 for the scores per stones. The version of the game we use comes from a Linux distribution and has a help-file [3], but this help-file does not name the type of scores made, like in TETRIS naming a 4 row score *Tetris* [7]. (For more on TETRIS, see [1].) For better readability we will name a line of 5 and 9 a *Minimal* and a *Maximal* respectively. The other possibilities will be called a *Sixer*, *Sevener* and *Eighter* for 6, 7 and 8 stones respectively.

The game ends when all places in the playfield are filled with stones and therefore no more moves are possible. The *goal* of the game is to achieve the highest total score in the end-state.

Our version of FIVE OR MORE uses a field of $M \times N = 9 \times 9$ places, $Y = 7$ different colours and $X = 3$ random stones dropped per non-free-play

#in line	score	name
5	10	*Minimal*
6	12	*Sixer*
7	18	*Sevener*
8	28	*Eighter*
9	42	*Maximal*

Table 1: Scores in Five or More

move. Our version can be found in every version of Linux with the Gnome environment with difficulty setting on medium board size.

2.2 Hardness

With the rules in Section 2.1 come a set of problems for the player. The first problem is that, because a stone needs to be moved along a path, paths can be blocked and certain moves can become impossible in one state while they were possible in the previous state.

The second problem is that the game does not really show winning or losing. There is only one state which shows the player has lost: the full field. Every move the player makes eventually just postpones the player's loss. There is no winning state to work to.

The third problem is an estimated average upper bound of the possible score. Imagine making a line of stones. If a couple of stones of the same colour are scattered around an empty field and the player wants to align those, the player needs 4 moves to align those stones and score a *Minimal*. In the first 3 moves no line is formed, thus $3 \times 3 = 9$ random stones are dropped. Then the line is formed, so 5 places in the field are cleared, making the total winnings of empty places $5 - (3 \times 3) = -4$ or: the player loses 4 empty places to score a *Minimal*. These losses will increase for longer lines. If the player takes the opportunity of 2, 3 or the unlikely event of 4 stones that are already aligned, the winnings of empty places are -1, $+2$ and $+5$ respectively. A state with 3 already aligned stones is very rare, so we have to be aware of a loss per *Minimal*, *Sixer* or more of 1 empty place. Since the field has 81 places, a first estimate of the maximum score is $81 \times 10 = 810$ points or less. We can calculate the losses and scores for other situations, but as we will see in Section 3.3 these calculations do not add much to this research.

Theoretically it is possible that only stones of one colour will drop down, making aligning and scoring always possible. This way we could say that the estimated average upper bound can be broken and that an infinitely long played game could also be possible. This is highly unlikely and therefore we try to break the estimated average upper bound by using better strategies and not by manipulating the game. The infinitely played game will not be discussed in this paper.

3 Analysing the game

In this section we approach the game by first applying the Monte Carlo algorithm to it, then trying to enhance the algorithm a little and study how the game behaves.

3.1 The Monte Carlo algorithm

The Monte Carlo algorithm plays a game by calculating a number Z of random simulations in memory for all possible moves in one state. From these simulations a rating is calculated and the move for which the best rating was calculated from the simulations is the move that will be done in the played game. Which rating is best (like highest maximum score, calculation on eventual structures, etc.) is decided by the designer of the implementation. The played game comes in a new state after a chosen move and the algorithm repeats itself until a desired state is reached [4, 5].

3.1.1 Implementation

To gain full control of the game, we coded the environment and rules of the game in C++. The first step after that was trying the Monte Carlo algorithm on the game. Our implementation did every possible move in memory and for every different state created by those moves the algorithm simulated $Z = 100$ random games which were played until they reached an end-state, a full field. From these simulated moves the move with the highest average score was chosen and done in the game. These steps were repeated until the played game also reached the end-state of a full field. Depending on the game the calculations for a whole game took roughly 45 minutes to an hour on the laptop.

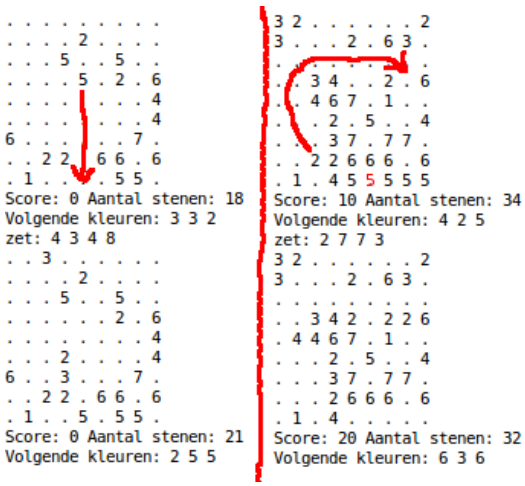

Figure 2: (left) The algorithm lines up a set of stones blocking a path and (right) accidentally frees it up again.

3.1.2 Results

The Monte Carlo algorithm reached an average score of 78 in 64 games, which is quite good, since a fully random played game practically never reaches above 22 and has an average total score of just above 0.5 points, this in playing 1000 games. Yet, Monte Carlo is not as good compared to human players, which average at least 150, see Table 2.

3.1.3 Discussion

The Monte Carlo algorithm aligns stones in lines, mostly horizontal and vertical, not diagonal, but quite often fails to finish a line sometimes leaving an unremovable wall in the field.

The algorithm usually makes poor choices. For example: In a calculated game that is partly shown in Figure 2 it aligned a stone with stones of a similar colour. The problem was that it also blocked the path to complete the line at the bottom of Figure 2 (left). To give the reader an idea of strategy: the 5 blocks the path to finishing its own line while it is also blocked by a line of 6's which is also blocked by a 7. To get the 5's in a *Minimal*, the 7 must be (re)moved, then the 6's should be removed and then a 5 must be inserted. Eventually a stone of that colour fell on the empty place in the

line, see Figure 2 (right). One must not count on this kind of luck. The game eventually scored 120 points, which is not that high.

Another kind of move worth noting which we did not understand was that the algorithm sometimes placed similar stones in a "horse jump" from each other as seen in chess, one step vertical and two steps horizontal or vice versa from each other. The algorithm probably tried to maximize the possibilities to score. This behaviour vanished, however, when we got to the next step of the research.

3.2 The Simple Bias Function

Since the Monte Carlo algorithm had trouble finishing lines, the next step in our research was to create a simple function which could finish lines. Because the function gives the Monte Carlo algorithm a bias to score when possible instead of calculating simulations, this function will be called the *Simple Bias function*. This function could detect a line of a given length and, when it found the first line that had this length, it could finish it with stones from elsewhere on the field. If it could not find a line, it would do nothing. The function is called "simple", because it had to find and finish a line with not too many calculations, since it was going to be used in the simulations of a Monte-Carlo-like algorithm.

3.2.1 Implementation

The Simple Bias function rates an empty place with a score for every colour. It looks in all 4 directions, being horizontal, vertical and the both diagonals, then counts the adjacent aligned stones for a direction, takes the square of that number and adds all the squares from the 4 directions. This outcome is the rating for an empty place. When an empty place is above a desired rating, the Simple Bias function will look for a stone to finish this line and try to score a *Minimal*. Note that this function is considered "simple", because it takes the empty places in order of the first that is rated above the desired rating, rather than in order of places with highest rating first. It also does not check if a formed line is not blocked by other colours making a *Minimal* or better impossible.

For example: see Figure 3. The simple rating algoritm gives every place that is adjacent to a stone a rating of how many stones from a single color are in a line. Next, the Simple Bias function searches the first place with a

Figure 3: Rating of empty places in the Simple Bias function

rating higher or equal to "3 stones in a row", in our case 9. This will be the 4th place in the second row which the algorithm will try to find a matching stone to which is not in that line. In this case that is impossible, because the algorithm needs an extra blue stone which is not on this example's board.

We have used the Simple Bias function in three different ways. In every implementation the Simple Bias function finished lines of 3 or more. The different implementations are:

1. For every turn the algorithm would first try the bias function and on no result, it would try the Monte Carlo algorithm with random simulations. In short: *Biased Monte Carlo*, hereafter abbreviated to *BMC*.

2. For every turn the algorithm would try a move with the Monte Carlo algorithm, without searching for a line first. The simulations in the Monte Carlo algorithm, however, first tried the Simple Bias function for every turn before doing a random move. In short: *Monte Carlo with Random simulation with Bias*, abbreviated to *MCRB*.

3. The last way combined the first two, first trying to finish a line, then trying the Monte Carlo algorithm with Biased simulations. In short: *Biased Monte Carlo with Random simulation with Bias*, abbreviated to *BMCRB*.

3.2.2 Results

In Table 2 we compare the results of our Monte Carlo based strategies against those of human players. Note that the average of human players are estimates of the recalled total scores of the players. As can be seen in Table 2, BMC did not get a total score as high as a normal Monte Carlo algorithm, which was odd, because the Simple Bias function should be an extension to the Monte Carlo algorithm to help it with moves to align stones to a line and score. The Simple Bias function was intended to help, but in BMC it eventually made the Monte Carlo algorithm worse.

MCRB did it a lot better, which is less odd because the simulations gave higher results, since in random games potential lines were made and thus moves that led to a higher score got a higher average. The "horse jump" behavior disappeared with the biased random games. Again, stones were aligned, but not always finished to a *Minimal*.

Because the simulation now had an extra calculation, the simulations took much longer in computing time. For 1000 games on the laptop the unguided random games took ± 1.5 seconds, the random games with bias took ± 11.3 seconds.

BMCRB scored highest by far, but on average not as high as a good human. Stones got aligned and those lines were finished. One flaw in the Simple Bias function emerged, however. We noticed that in some cases, when a line of 3 stones is found in the field before a line of 4 stones of the same colour is found in the field, the Simple Bias function will take stones from the line of 4 and use those to finish the line of 3. This is not a good move, because if the line of 4 is finished with a stone of the line of 3, less moves are necessary for a line to score and there is the risk that the line of 3 (by then 4) is blocked by a randomly dropped stone. Note that with a free-play move the line of 3 (by then 2) is easier to make into a *Minimal* or better.

3.3 Discussion

When a strategy in a simulation is used to determine the average score in the Monte Carlo algorithm, a similar strategy must be used around the Monte Carlo algorithm. BMC (and actually MCRB too) failed, because the simulation anticipated different moves than those that were eventually done.

In Table 3 we have listed the moves sorted by the increase of score after that move. On the left are the number of moves made by the algorithms sorted

strategy	max	average
Monte Carlo	190	78.38
BMC	132	59.71
MCRB	334	206.15
BMCRB	750	375.03
Human 1	202	±150
Human 2	398	±380

Table 2: Different strategies compared to eachother and human players

	Total moves				Percentage of moves			
Score	BMCRB	MCRB	MC	902	BMCRB	MCRB	MC	902
None	3562	2386	2416	174	70.12%	74.87%	83.20%	66.16%
Minimal	1463	742	467	86	28.80%	23.28%	16.08%	32.70%
Sixer	51	47	20	2	1.00%	1.47%	0.69%	0.76%
Sevener	3	9	0	1	0.06%	0.28%	0.00%	0.38%
Eighter	1	2	1	0	0.02%	0.06%	0.03%	0.00%
Maximal	0	1	0	0	0.00%	0.03%	0.00%	0.00%

Table 3: Number of certain kinds of moves and their percentages of the total per different strategy.

by the score that is banked with that move. On the right are the percentages of these moves of the total number of moves per algorithm. (Ignore the 902 column for now.) This way we can see how often a score is banked and how many stones are in a banked score. Most of the total score is made with lines of 5 and 6 stones. It seems that greediness in an algorithm pays off for this game. BMCRB is the greediest of the strategies Monte Carlo, MCRB and BMCRB, since it makes the shortest lines, and gets the highest scores, see Table 2.

Greediness is probably not the best strategy throughout the game. In the graph in Figure 4 we have counted the states of the 3 strategies of all the games by how many stones were in the field. These numbers are devided by the amount of games played by a strategy creating an average amount of states with a given amount of stones per game. Putting the amount of stones on the horizontal axis and the average amount of states with an amount of stones on the vertical axis resulted in the graph in Figure 4. This way we can see the average flow of a game by how many stones are on the board.

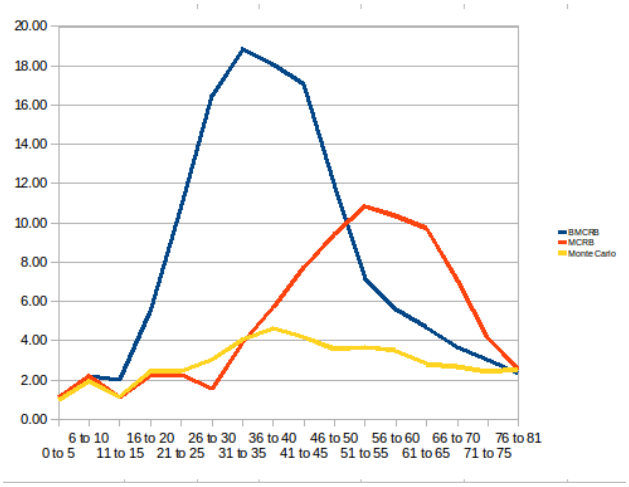

Figure 4: Flow of the game per strategy. The horizontal axis denotes the number of stones in the field per state discretised to steps of 5. The vertical axis denotes the average number of states with a number of stones defined on the horizontal axis

Incidentally the field under a line in the graph shows the average number of states in a game.

BMCRB (the blue line) has its most states with around 30 to 35 stones in the field, while Monte Carlo (the yellow line) has its maximum around 36 to 40 stones. MCRB (the red line) has more states nearing the endgame while BMCRB has more states nearing the beginning of the game. Somehow MCRB knows better what to do in a different state than BMCRB. Note that it does not mean that MCRB is better with more stones in the field, but it might be a consideration.

From the graph in Figure 4 we can deduct that an algorithm while playing a game of Five or More is circling around a certain number of stones. It is clear that the game has a certain flow from an empty beginning with not enough stones to align, to an optimal number of stones where one can score a lot, to getting near the end game where paths are blocked and the field is filled rather quickly.

The estimated average upper bound of the score, suggested in Section 2.2, could influence the way our algorithm is working. The score can be used as a sensor to predict in what kind of state the game is. The results showed until

now do not score over 810 points, so this could be a good way to approach the game. However, we have tried different implementations with simulations that did less random games and/or did not play the simulation to the end-game. These all scored slightly lower than the implementation mentioned in the first paragraph of this section. Using more games than 100 simulations per anticipated move resulted in a longer computing time, but not better scores. Also we tried a Monte Carlo algorithm that chose the highest maximum score, instead of the highest average. Choosing the move with the maximum score led to an algorithm that could only score one or two *Minimals* or *Sixers* in the whole game.

On average, all these attempts scored lower than the original Monte Carlo, but one game scored 902 points in total. Its moves are listed in Table 3. The playing of the algorithm was not optimal, but this shows that there might not be an average upper bound to the total score as we estimated.

To tackle most of our problems and questions from Section 1 and 2.2, we can use our results as benchmarks for new strategies and we now know that we are not limited to a maximum possible score.

4 Second generation algorithms

Now we know more about the game, we can design our second generation algorithms.

4.1 Dynamic BMCRB and Inverted

4.1.1 Implementation

Since BMCRB and MCRB both work best at different numbers of stones in the field, it is logical to try a setting in which the algorithm works with BMCRB in states with less than 45 stones on the field (Figure 4) and with MCRB for states with more than 45 stones in the field. This strategy will be called *Dynamic BMCRB*. For good measures we also want to see what happens when we invert the two, using MCRB before 45 stones and BMCRB after. This strategy will be called *Dynamic BMCRB inverted*.

Break on	normal	inverted
30 stones	313	257
40 stones	333	265
50 stones	345	229

Table 4: Average scores of the Dynamic BMCRB

4.1.2 Results

The dynamic BMCRB strategies did not perform as expected. The normal Dynamic BMCRB with a break on 50 stones scored best, but not as high as the original BMCRB.

4.2 Advanced and Legacy Bias Function

The simple bias function was not very complicated, but gave good results. To acheive better results we try to enhance this function.

4.2.1 Implementation

Our new bias functions will order the empty places in rating for a certain colour and start with the place with the highest rating, the place will be filled with a stone that is in the shortest row of that colour, if it is in a row at all. This will be called the *Advanced Bias function*.

Figure 5: Rating of empty places in the Advanced Bias function

Where the Simple Bias function accumulated the square of the number of stones directly aligned to a place, the Advanced Bias function will look in 4 directions, horizontal, vertical and both diagonals, count the number of stones and empty places until the edge or another colour than the one counted is found and will rate the place according to the formula

$$(\textit{stones left} + 1)^2 \times (\textit{stones right} + 1)^2 - \textit{distance}$$

where *stones left* and *stones right* are each the number of stones of a side in the horizontal, vertical or diagonal direction (if it is counting vertical, left and right become up and down etc.) and *distance* is the number of empty places to the first stone of the colour that is counted. With this formula the algorithm will first try to fill a line between two stones and then add stones at both ends of the line. A good property of this rating is that is can also be applied to stones which gives a rating for the importance of a stone in a line.

The Advanced Bias function also counts the amount of empty places and will only rate a place if $\textit{places} + \textit{stones in line} \geq 5$. The strategy will not try to finish a line if there is not enough place to finish it. Also the strategy will not close off a corner of the field by making a diagonal of 4 stones 3 places from the corner.

For example: see Figure 5. There are many redundant rated places, but the important places are the 4th place in the 3rd row with the blue rating of 36 and two red ratings of 0 on the 5th column, 5th row and 8th column, 8th row. The red ratings of 0 show that there are not enough empty places to create a *Minimal* of red stones, so the algorithm abandons that line until enough places are freed up. To try and score a blue line, the algorithm has 3 places to put a blue stone. With the advanced formula the place in the middle of the line gets a rating of 36 and 16 on the outside, which gives the Advanced Bias function a preference to put stones in the middle. This is preferable, because the middle places are mostly the places that get blocked first. Also it will not leave 5 disjunct stones in a line this way. For example: 2 red stones, an empty place and another 3 stones in one line are 5 stones in a line, but do not score a *minimal*.

For good measures we make another strategy based on the Advanced Bias function with the rating formula of the Simple Bias function for empty places, but still using the rating for stones of the Advanced Bias function to find a stone to put on an empty place. This function will be called *Legacy Bias function*.

Strategy	max	average	no score
Advanced Bias aligning	402	135	1
Advanced Bias finish from 3	356	108	5
Legacy Bias aligning	262	62	45
Legacy Bias finish from 3	110	12	365
Simple Bias finish from 3	92	12	347

Table 5: Scores of the Bias functions after 1000 games. The column *no score* shows the number of games in which the total score was 0

4.2.2 Results

We have run 1000 games with the Advanced Bias function, the Legacy Bias function and the Simple Bias function on finishing lines of 3 stones.

The Advanced Bias Function is clearly better than the Simple Bias function when working on its own. The Advanced Bias function scores higher than the original Monte Carlo algorithm and as a score comes near MCRB, see Table 5 and 2.

As can be seen in Table 5 the Legacy Bias function is not very good at aligning and when finishing from 3 it does not outperform the Simple Bias function much.

4.3 The new bias functions and Monte Carlo

4.3.1 Implementation

With the Advanced and Legacy Bias function tested, we now implement them in a Monte Carlo structure in the same way as BMCRB, making the strategies Advanced Bias Monte Carlo with Advanced Biased simulations or *ABMCRAB* and Legacy Bias Monte Carlo with Legacy Biased simulations or *LBMCRLB*.

4.3.2 Results

While the Advanced Bias function does well on its own, somehow linking it to the Monte Carlo algorithm does not give better scores than BMCRB, of Table 2. Yet, when we compare LBMCRLB with BMCRB, the results are very good. ABMCRAB had difficulties scoring the average of BMCRB, but

LBMCRLB got a higher average than BMCRB and it also scores higher than estimated average upper bound in Section 2.2. The maximum game listed in Table 6 is not a fluke, in fact 5 of the 110 played games were above the 810 score.

Strategy	max	average
ABMCRAB	440	197
LBMCRLB	1104	472

Table 6: Scores of the Advanced Monte Carlo functions

4.4 Discussion

The idea with the dynamic BMCRB strategies was that if the strategies could cope with a different number of stones, switching between them would make the game take longer, have more states and thus have more states to score in. Apparently the endstates of BMCRB and MCRB are different. Where MCRB will have a lot of unfinished lines, BMCRB will have a lot of "white noise", stones of all different colours with a slim or no chance of aligning and removing. We achieved an average score between MCRB and BMCRB, compare Table 4 with Table 2. From the dynamic strategies we can conclude that it does matter how many stones one has on the field, but that the number of stones does not correspond with a state a strategy can cope with. The best strategy still is to keep the number of stones in the field to a minimum, so the possible number of moves is maximal.

Because the Advanced Bias Function looks at all the possible moves and does not take the first that has the rating it is looking for, the Advanced Bias function can align stones from across the board and can therefore also serve as an align function. The Simple Bias function is not capable of this action, since for aligning the rating of 1 is asked. The Simple Bias function will then take any place and put in a stone of the desired colour, which will result in fields of stones of the same colour instead of lines.

What happens in BMCRB is that the Simple Bias function aligns stones no matter if a *Minimal* score is possible and the nature of the Monte Carlo algorithm makes sure that when a line of stones is blocked, it will be freed up, so a *Minimal* score can be made eventually. When the Advanced Bias function decides there are not enough places and stones to make a *Minimal*,

a line is given up on when it is blocked and will never be attempted to finish. Note that freeing up a spot to score is a very expensive move in Five or More, but apparently not too expensive to give up on an unfinished line. Thus ABMCRAB scores lower than BMCRB. Since the rating formula of the Legacy Bias function in LBMCRLB has been copied from the Simple Bias function in BMCRB, lines are not given up on and the total score in with LBMCRLB is higher.

The game Five or More is actually counterintuitive. Speculation is out of the question, the player/strategy must be as greedy as possible aligning stones next to each other with the slight exception of sometimes freeing up a line when it has advanced to a certain length. We can distinguish at least 3 different actions in the game Five or More: Aligning (start building a line), Scoring (finishing a line) and Freeing (remove a stone that is blocking an almost finished line). More actions are discussed in Sections 5 and 6. A strategy not using Monte Carlo could replace the Monte Carlo part with the speculative actions (all actions except the scoring action).

5 Strategies without Monte Carlo

We have found that speculation is punished in Five or More but that we need some kind of speculation to create a situation to score in. We now try to create some speculative functions that should replace Monte Carlo.

5.1 Align-Score

First we tried combining the parts we already have: the Legacy Bias function as a score function and the Advanced Bias function as an align function resulting in the Align-Score strategy.

5.1.1 Implementation

We first let the program try to finish a line of v stones and if it could not find any, it would try to align stones. If aligning also was not possible (this is usually in the begin- or endgame) a random move would be made. This means we first let the Legacy Bias function search for a line of v stones to finish, if that failed, the Advanced Bias function would try to align a stone to a line of 1 stone and, if even that failed, a random move would be done.

Figure 6: Rating shown of the old (left) and new (right) bias function rating

To see where the break between aligning and scoring could be set best, we tried different values for v, see Table 7 in the "Score on" column.

We altered the align function's rating formula from the one specified in Section 4.2 to:

$$(\textit{stones left} + 1)^2 \times (\textit{stones right} + 1)^2 - \textit{distance}^2$$

This was done because if the distance was taken linearly, the aptness for an empty square on the other side of the field could be too high for a colour. Note that the Advanced Bias rating function only returns the highest aptness. This results in the situation that a line of 3 can make another line redundant in certain states, see Figure 6. What happens here is that the aptness of the blue line is drowned out by the red line. Both are equally rated, but in this case it will be better to try to align the blue line. By altering the rating function, the blue line becomes more important for the strategy and cases like these are better handled. Note that near the red stones (with rating 17 and 18) it is still handy to keep the same rating, because keeping those paths clear makes moves possible and thus scoring easier.

5.1.2 Results

Comparing Table 7 and 5 it is shown that the Align-Score strategy scores an average of 10 to 15 points higher with $v = 3$ or $v = 4$. With the enhanced rating formula for the Advanced Bias Function it scores even 15 to 20 points higher. With $v = 3$ the maximum is higher, while with $v = 4$ the average is

Align-Score Strategy

	old align function			new align function		
Score on	max	average	no score	max	average	no score
1 stone	242	60	38	286	64	40
2 stones	312	77	17	262	78	13
3 stones	548	144	2	566	150	5
4 stones	446	**149**	0	462	**154**	1
5 stones	374	132	3	412	142	1

Table 7: Scores of the Align-Score for 1000 games per try

higher. Although the unbiased Monte Carlo has been beaten by the Advanced Bias function finishing lines of 1 stone which we used as align function, our enhanced Align-Score strategy does not beat the slightly better strategies as MCRB, see Table 2.

5.2 The clear-up function

When the Align-Score strategy is busy building a line and during that phase a random stone is dropped blocking the line, it can not solve that problem and starts aligning another line. A simple way to free a blocked line is to find a stone with a high aptness of a different colour than itself, its un-aptness, and then move that stone to a place where it is best suitable for its own colour and not blocking another line of a different colour.

5.2.1 Implementation

We took the code of the Advanced Bias function as a template and rewrote the rating of stones and empty places. For every empty place a certain *aptness* was calculated for every colour and saved in one of seven descending ordered lists corresponding with that colour. The aptness was obtained by calculating a rating for a colour for a place with the rating formula of the Advanced Bias function and subtract the rating for every other colour from that. In abstract:

$$\textit{aptness for a colour} = \textit{rating for a colour} - \textit{sum of rating for other colours}$$

The higher the aptness, the better the place can be used to put a stone that is blocking a line. If the aptness is negative, it is better not to move a stone

Align-Clear-Score Strategy

	Clearing on # of stones in blocked line								
	3 stones			4 stones			5 stones		
Score on	max	avg.	no	max	avg.	no	max	avg.	no
3 stones	424	118	3	476	**160**	0	608	152	2
4 stones	202	68	11	416	130	0	428	149	1
5 stones	142	41	33	230	80	3	336	134	2

Table 8: Different configurations of Align-Clear-Score and their scores

of that colour to that place. Using this aptness rating the algorithm will not block a line with a stone that was blocking another line.

For every stone the maximum un-aptness was calculated in the same way as aptness, but now we calulate on a stone and only the highest rating is remembered. The un-aptness is stored in a list in descending order if the number of stones in that row was more than an asked number. Starting with the most un-apt stone the Clear-up function tried to move an un-apt stone to the most apt place for its colour.

We implemented the clear-up function in Align-Score, creating the Align-Clear-Score strategy. There are different possible values for the Score function and the Clear function to move a stone to a certain place. The results are shown in Table 8.

5.2.2 Results

The Clear-up action is a very unfavourable action, because the player wastes an extra action to make a line, but again we have made a bit of progress. With trying to score on a line of 3 and cleaning up blocked lines of 4 the average score is 4 points higher than the Align-Score strategy with score on 4 stones in a line.

The Align-Clear-Score strategy makes a line, tries to finish it, and when a stone blocks it, that stone is removed and the line is finished anyway. It goes wrong when more stones are placed on the board and paths to a line become blocked, see Figure 7. The score function can not score, because no stones can be moved to the line and, since the last place of the line is clearly free, the Clear-up function can not make it possible to clear a path. The field fills up after this situation and the score still is not very high.

5.3 The Shovel function

Where the previous functions generated the coordinates of a source stone and destination place, our new function will need to find a destination place that needs a stone to finish a *Minimal*, a source stone of the colour of that *Minimal* line, another source stone that blocks the path of the aforementioned source stone to the aforementioned destination and finally it needs to find a destination place for the blocking stone to move to, hoping it will not block the path also when moved there. The function that will find these coordinates will be called the *Shovel function*.

5.3.1 Implementation

The Shovel function is called when the Score function can not align a line. This is very important, since the Shovel function deliberately does not try to align a stone, but moves a stone that at best can only be aligned with a stone of its own colour.

First the function analyses the board by creating a list with blocked places in descending order calculated with the Legacy Bias rating to move to and 7 lists for the colours containing places to move to for blocking stones calculated with the Advanced Bias rating. For every blocked place the algorithm marks all the stones on the edge of the closed field and puts these stones in a list sorted ascending by aptness calculated with the Advanced Bias formula. For every blocking stone in the list the algorithm removes the stone temporarily from the board and looks if a path is possible from a stone with the desired colour. If a path is possible with this blocking stone removed, the algorithm tries to move the blocking stone to all the places defined in one of the 7 lists with empty places. The algorithm does this move in memory including 3 new random stones dropping down and if by then a path is possible in this memory state from a desired stone to the target place, then the move of the blocking stone to a place is made in the played game.

In short: see Figure 7. The Shovel function algorithm takes the stone with the cross and moves it away from the red line to an empty place where it can do no harm, so the desired stone can move along the red line. This function is not optimal. It can not really predict random falling stones and also can not break a “wall” thicker than 1 stone.

We implement the Shovel function in two ways: one version where it works on its own with Align-Score, creating Align-Shovel-Score (Table 9)

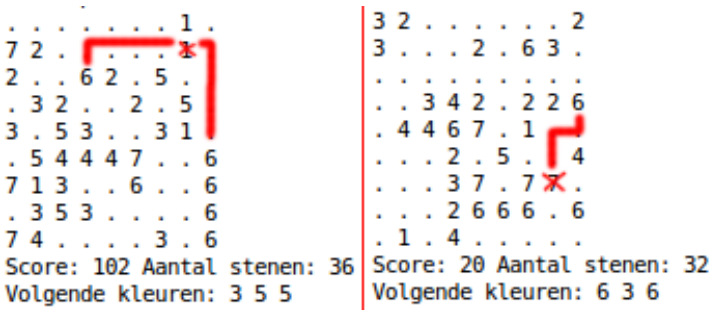

Figure 7: Paths in different games being blocked while not possible to reach a desired line

and one version where it works together with the Clear function, creating Align-Shovel-Clear-Score (Table 10).

Align-Shovel-Score Strategy

	Shoveling on # of stones in blocked line								
	3 stones			4 stones			5 stones		
Score on	max	avg.	no	max	avg.	no	max	avg.	no
3 stones	510	162	0	474	**170**	0	494	149	8
4 stones	132	28	109	462	**172**	0	552	160	2

Table 9: Different configurations of Align-Shovel-Score and their scores. Finishing on 5 or more is left out

5.3.2 Results

From Tables 9 and 10 it is shown that the Shovel function makes a nice improvement and can work well together with the Clear-up function. With Align-Shovel-Score the highest average total scores are 172 and 170 for finishing lines of 4 or more and 3 or more respectively and shoveling paths to blocked lines of 4 stones or more. Together with the Clear-up function the total scores increased even more to an average of 180 and 182 for shoveling for lines of 3 and 4 stones respectively, finishing lines of 3 and clearing lines of 4. Again we have a slight improvement, but not with an average total score as high as MCRB (206, Table 2).

Align-Shovel-Clear-Score Strategy

		Scoring on # of stones in line					
		3 stones			4 stones		
Clr. on	Shvl. on	max	avg.	no	max	avg.	no
3	3	366	122	1	152	40	41
	4	314	120	1	194	66	9
	5	382	118	1	214	64	10
4	3	448	**180**	0	140	34	84
	4	516	**182**	1	392	142	0
	5	450	170	0	422	136	0
5	3	512	166	1	124	28	102
	4	464	176	1	458	178	1
	5	538	162	2	452	160	0

Table 10: Different configurations of Align-Shovel-Clear-Score and their scores

5.4 The Semi-Tree strategy

One aspect of Monte Carlo we have not tried to mimic yet is the ability to predict future states.

5.4.1 Implementation

We know from our Monte Carlo based strategies that it is best to score a *Minimal* or higher first and then start thinking about speculative and predictive moves. The strategy we are going to use now is using a kind of search tree which decides on which strategy it will use for the next move if it can not align stones for a score. We will call this strategy the Semi-Tree strategy. Its structure is depicted in Figure 8.

The implementation of the Semi-Tree strategy first tries to call the Score function. When the score function can not finish a line, the strategy tries all three speculative functions described above: the align function (Advanced Bias), the Clear-up function and the Shovel function. When a function can make a move, the Semi-Tree strategy makes a simulation in memory after that function's move, trying to score in after each speculative move and thus creating a tree as seen in Figure 8. The height of the tree is limited to save calculation time and is called the *peek depth*. The results are presented in

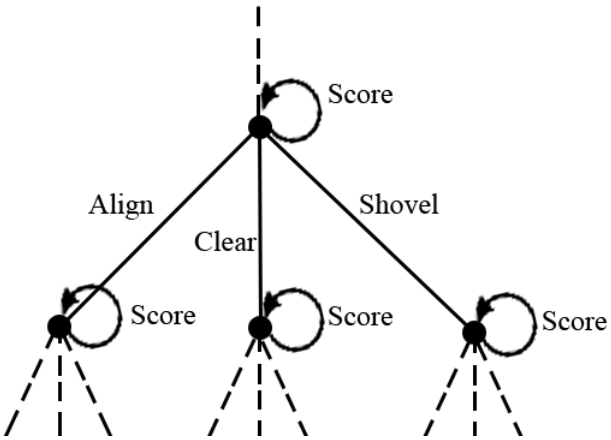

Figure 8: Structure of the Semi-Tree search tree strategy.

Table 11.

5.4.2 Results

For these simulations we took a maximum peek depth of 4. The calculation time per move increased exponentially, so 4 seemed a reasonable maximum. Again we played 1000 games per configuration. Because this took a lot longer to calculate, the tasks were separated over 10 processor cores of our desktop PC's. Every core calculated 100 games per configuration and it took around 1 full day to calculate all the 32 configurations. This brings the complexity of our Semi-Tree strategy near that of our Monte Carlo based strategies where we calculated around 20 games per core in half a day.

The best games have an attempted score on 3 stones, a clearing on 4 and shoveling on 3 or 4 stones. As expected the results become better when the peek depth is higher. Unfortunately the best results, 192 on peek depth 4, still do not match MCRB.

5.5 Discussion

There are still some ways to enhance our Semi-Tree strategy, but the direction that it is going is not the one we want. For a perfect strategy we most likely need to take into account all possible moves, calculate those moves to the

Semi-tree Strategy

Variables			peek depth of tree							
			1		2		3		4	
Scr.	Clr.	Shvl	max	avg.	max	avg.	max	avg.	max	avg.
3	3	3	700	178	526	170	684	180	486	180
		4	532	178	560	176	524	180	564	180
	4	3	588	180	472	180	572	188	566	192
		4	582	178	596	184	620	190	516	192
4	3	3	560	146	426	152	354	144	356	134
		4	528	158	498	162	538	158	444	154
	4	3	438	144	430	148	414	144	402	140
		4	524	154	506	164	480	172	436	172

Table 11: Results of the semi tree algorithm with variables on Finishing a line (Scr.), Clearing a line (Clr.) and opening a path (Shvl.) on number of stones in the line

end state to predict the consequence of a move and thus make multiple simulations after each move. Unfortunately, this means going back to the Monte Carlo structure.

Our speculative functions are not useless, however. They have shown us that the speculative actions aligning, clearing and shoveling are all beneficial to the flow and total score of the game.

6 Conclusions and Future Work

The game Five or More is a quite complicated game, but when researched further might give a nice addition to the field of artificial intelligence. This is because the game with our specifications in Section 2.1 is contradictory in a sense that it punishes speculation, but it needs speculation in order to score. It needs knowledge of the game for high scores, but an algorithm based not on that knowledge has achieved the highest score by far up until now. This algorithm is LBMCRLB, Legacy Bias Monte Carlo with Legacy Biased simulations, which is based on Monte Carlo and therefore does not use (much) knowledge of the game.

We can use our findings to devise a new strategy. Banking a score is

always favourable (Section 3.3). The number of stones in the field should not affect the strategy for scoring (Section 4.1.2 and 4.4). The speculative moves that do work are aligning, clearing a line and clearing a path to a line, as suggested in Section 4.4 and shown in Section 5, yet there might be more. There is a possibility that a new strategy can show that the game can be played infinitely (Sections 3.3 and 4.3), as in: the game can have an infinite amount of states in one game.

The random nature of Five or More might mean that for a high total score the player needs to be a bit lucky. LBMCRLB might have achieved a total score of 1104, but its average was not even half of that. To find out if there is a real strategy possible and the game does not depend on luck, future work on Five or More might consider focussing on the possibility of an "infinitely" played game, a better way for rating than the Advanced and Legacy Bias function's formulae and/or identifying new speculative actions apart from those specified in Section 4.4 and 5.

References

[1] R. Breukelaar, E. Demaine, S. Hohenberger, H. J. Hoogeboom, W. Kosters, and D. L. Nowell. Tetris is hard, even to approximate. *International Journal of Computational Geometry and Applications*, 14:41–68, 2004.

[2] R. A. Hearn and E. D. Demaine. *Games, Puzzles, and Computation.* Peters, 2009.

[3] E. Kovács. *Five or More Manual.* GNOME Documentation Project, 2002.

[4] A. M. Law and W. D. Kelton. *Simulation Modeling and Analysis.* McGraw Hill, 3rd edition, 1991.

[5] S. Russell and P. Norvig. *Artificial Intelligence, A Modern Approach.* Prentice Hall, 3rd edition, 2010.

[6] F. W. Takes and W. A. Kosters. Solving Samegame and its chessboard variant. In *Proceedings of 21st Benelux Conference on Artificial Intelligence*, pages 249–256, 2009.

[7] Tetris Inc. www.tetris.com, accessed june 2010.