Universiteit Leiden
Opleiding Informatica

Using Outcome Weights
in Monte-Carlo Tree Search
for Multiplayer 3D Hex

Name: Sarah Haddou

Date: 28/08/2015

1st supervisor: Walter Kosters
2nd supervisor: Jeannette de Graaf

BACHELOR THESIS

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

Abstract

In this paper we will investigate a 3D version of the game of Hex. In this variant the gameboard is in the shape of a cube and each field is also in the shape of a cube. First we will try to solve the game to know which of the players will win the game when they play in an optimal way. We have solved the game of $2 \times 2 \times 2$ 3D Hex. The outcome of this game, given optimal play, is a draw. Solving games with larger playfields takes a long time.

So for playboards larger than $2 \times 2 \times 2$ we will use Monte-Carlo Tree Search to examine the game. We use outcome weights to decide which of the possible moves will yield the best result. These outcome weights are based on the nature of the players. There are three different natures that we will study in this paper: competitive – selfish, misère and cooperative. For cooperation there are three levels: level 1, level 2 and level 3 cooperative.

We run the experiments on different types of gameboards: a $3 \times 3 \times 3$ gameboard, a $4 \times 4 \times 4$ gameboard and a $3 \times 3 \times 3$ gameboard with the center field excluded. We use the outcome weights belonging to the different types of natures to investigate how the game ends if the players play in that way.

Contents

1 Introduction **4**

2 Related work and rules **6**
- 2.1 Rules of 2D Hex . . . 6
- 2.2 Variants of Hex . . . 7
- 2.3 Multiplayer 3D Hex . . . 7
- 2.4 Rules of Multiplayer 3D Hex . . . 8

3 Brute-force solutions **8**
- 3.1 Sides of size 2 . . . 9
  - 3.1.1 Proof: $2 \times 2 \times 2$ 3D Hex always ends in a draw . . . 9
- 3.2 Sides of size 3 . . . 11

4 Applying Monte-Carlo Tree Search **12**
- 4.1 Method . . . 12
- 4.2 Union find . . . 12
- 4.3 Monte-Carlo Tree Search . . . 13
- 4.4 Evaluation of Monte-Carlo moves . . . 13
- 4.5 Nature of the players . . . 14

5 Experiments **16**
- 5.1 Number of games . . . 16
- 5.2 Number of playouts . . . 17
- 5.3 Results . . . 18
  - 5.3.1 Study of a whole game: sides of size 3 . . . 18
  - 5.3.2 Study of a whole game: sides of size 4 . . . 19
  - 5.3.3 Study of a whole game: exclude fields . . . 19
  - 5.3.4 Study of the best moves: sides of size 3 . . . 19
  - 5.3.5 Study of the best moves: sides of size 4 . . . 21
  - 5.3.6 Study of the best moves: exclude fields . . . 22
- 5.4 Discussion . . . 23
  - 5.4.1 Competitive – selfish. . . . 23
  - 5.4.2 Misère. . . . 24
  - 5.4.3 Cooperative. . . . 25
  - 5.4.4 The best moves . . . 29
  - 5.4.5 Runtime . . . 31

6 Conclusion and future work **31**

References **33**

1 Introduction

Hex is a two-player board game played on a hexagonal grid in the shape of a rhombus. In this game there are two players. Both have their own pieces, marked with their color and have their own sides of the rhombus marked with their color. Two opposite sides belong to one player, the other two sides to the other. The two players place in turn pieces of their color, usually black and white (or red and blue), on empty cells of the hexagonal grid. Each player tries to build a chain of pieces of their own color, from one of their sides to the other. The playboards can vary in size, but usually the game is played on a 11x11 rhombus, see [18]. A 11x11 playboard is shown in Figure 1, Figure 2 shows a playboard in which blue wins.

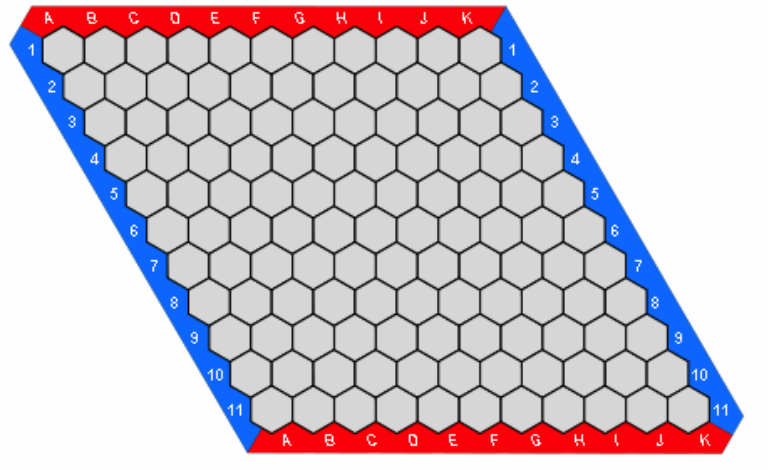

Figure 1: A Hex gameboard with sides of size 11. From: MrLsMath.com, uploaded by Bill Lombard.

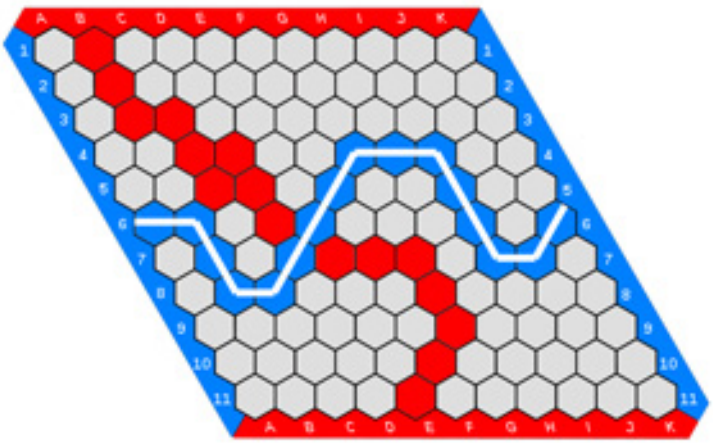

Figure 2: A Hex gameboard with sides of size 11. With this configuration blue wins. From: wikipedia, uploaded by Jean-Luc W.

In this paper we will study Multiplayer 3D Hex played with three players. First we will try brute-force solving and then we will use Monte-Carlo Tree Search to study the game. The Monte-Carlo Tree Search algorithm that we used, uses outcome weights to evaluate a possible move. For every configuration of the playboard, when Monte-Carlo Tree Search is called, four outcome weights are needed. The points that are assigned to the value of a possible move when a random playout is played from a certain configuration are based on these four weights. For example, take a look at the following outcome weights: 1 2 3 4. When evaluating a possible move from a certain configuration these values are needed. The first value denotes the number of points that will be assigned to the value of a move if player 1 wins the random playout. The second value denotes the number of points that will be assigned to the value of a move if player 2 wins the playout. The third for player 3 and the fourth value for if the game ends in a draw. From now on we will use that order for the outcome weights and we will denote player 1 with 'P1', player 2 with 'P2', player 3 with 'P3', and a draw with 'draw'.

We use Monte-Carlo Tree Search with outcome weights, for which there are several values possible. For example if all of the players want P1 to win, then the outcome weights for all of the players will be 3 0 0 1. Now all the possible moves from the present configuration are one by one checked. First one of the possible moves is done and then the game will be played out randomly. If the game is won from a certain configuration by P1, then the value of the move that was checked will be increased by 3 points. The value of the move will be increased by 0 points if one of the other players won and with 1 if the game ends in a draw. The outcome weights that will be used in this paper can be found in Table 1.

Nature of the player	Outcome weights	Meaning
Competitive – selfish	P1: 3 0 0 1 P2: 0 3 0 1 P3: 0 0 3 1	Every player wants to win himself.
Cooperative level 1	P1: 3 1 0 1 P2: 1 3 0 1 P3: 0 0 3 1	Every player wants most to win himself, but now there exist 'teams'. So a player that is in a team also kind of wants his teammate to win.
Cooperative level 2	P1: 3 2 0 1 P2: 2 3 0 1 P3: 0 0 3 1	Every player wants most to win himself, but now there exist 'teams'. So a player that is in a team also kind of wants his teammate to win.
Cooperative level 3	P1: 3 3 0 1 P2: 3 3 0 1 P3: 0 0 3 1	Every player wants most to win himself, but now there exist 'teams'. A player that is in a team is equally satisfied if his teammate wins.
Misère	P1: 0 3 3 1 P2: 3 0 3 1 P3: 3 3 0 1	Every player does not want to win himself.

Table 1: The outcome weights for the different natures of the players.

In this paper Section 2 introduces the rules of Multiplayer 3D Hex and contains some background information on 2D Hex. In Section 3 we will give a brute-force solution for $2 \times 2 \times 2$ Multiplayer 3D Hex and proof that, given optimal play, the game ends in a draw. In Section 4 we will apply Monte-Carlo Tree Search using outcome weights to study the game in Section 5. Furthermore we will study the three first most done moves in Section 5. Section 6 contains the conclusion of this study and future work.

This research bachelor thesis is written under supervision of Walter Kosters and Jeannette de Graaf in pursuit of a Bachelor of Science degree in Computer Science at Leiden University.

2 Related work and rules

Hex was introduced in 1942 by Piet Hein at Niels Bohr's Institute for Theoretical Physics, see [15, p. 73–83]. In 1947 it was also invented independently by the mathematician John Nash at Princeton University, see [18]. More on Hex can be found at [8].

2.1 Rules of 2D Hex

The game starts with an empty playboard. Board sizes vary, but usually the game is played on a board with sides of size 11. Two players in turn place pieces of their color on a hexagonal grid in the shape of a rhombus, the player with black pieces makes the first move. They can only place pieces on the grid where there is no other piece. One player plays black and the other plays white pieces, see [2].
A player wins the game if he/she manages to create a chain of pieces of his/her own color from one of his/her sides to the other. The other player tries to stop player 1 from creating such a chain.
When there are no empty places left on the board then one of the players has won the game, because it was proven that the game never ends in a draw. This can easily be seen, see [14] and [16].
Furthermore, given perfect play in Hex the first player has a theoretical win in any size because of the *strategy-stealing argument*, see [17]. This argument proves that the second player cannot have a winning strategy, so the first player always wins the game. This is because if there was a winning strategy for the second player, then the first player could *steal* this strategy and be one turn ahead, see [16]. The strategy-stealing argument works as follows: the first player does a random move on the playboard. Now the other player does a move. The first player now can pretend to be the second player, by ignoring the first move he made. He steals the winning strategy from the second player by pretending to be the second player. He plays the winning strategy that belonged to the second player and when for this strategy he has to put a piece on the place where he already had put a piece in the first turn, he just does another random move instead. This argument can only be used for games where the players have the same set of possible moves with the same results. Because then an extra piece of yours on the playboard cannot be disadvantageous.

For this reason in the game of Hex the pie rule is more often than not used. The *pie rule* is a rule that gives the second player the opportunity to take over the move of the first player or to make his own move. So the second player has the possibility to become the first player or to stay the second. So if the first player does a move that leads to winning, then the second player can take over the move.

2.2 Variants of Hex

The game of Hex has many variants that were proposed in the years after its invention. Some of the variants of Hex are Blockbusters, the game of Y, Mind Ninja, Chameleon, the Shannon switching game, Gale, Pex, Hecks, Nex and Havannah. These games are strategy board games played on a hexagonal grid of any shape and any size. Monte-Carlo Tree Search was applied to many variants of the game of Hex, see [12], [4] and [9], more on Monte-Carlo Tree Search can be found at [10] and [5]. Also other algorithms are used to solve Hex, see [1], [7], [3] and [6].

2.3 Multiplayer 3D Hex

One version of 3D Hex was studied before, see [8]. In that version the gameboard was folded so that you could play on a 3D-shaped gameboard. But you could only place pieces on the surface of the shape and there were only two players. A new variant of 3D Hex that we will study was to the best of our knowledge never studied before. In this variant the game is played on a cube, but here not only the surface of the cube is used to play on and in this variant we have three players. The most interesting part about playing Hex on a 3D board is that the fields now are cubes that also adjoin six other fields like the 2D version, but in a different way.

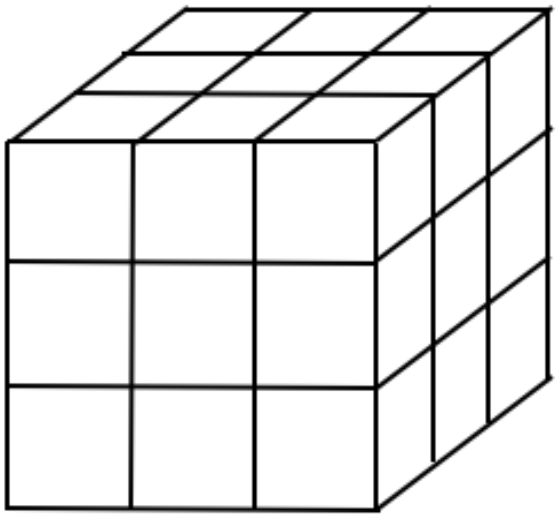

Figure 3: 3D Hex is played on a cube.

2.4 Rules of Multiplayer 3D Hex

The game of Hex is played on a rhombus, where we have four sides. Each field is at most connected to six adjacent fields. 3D Hex is played in a cube with six sides consisting of fields in the form of a cube. Each of these fields now is also at most connected to six adjacent fields, see Figure 3.
There are two more sides to the playboard in 3D Hex, therefore we have decided to add another player to the game. Thus 3D Hex is played with three players. One player needs to connect the top and bottom of the cube with pieces of his color. The second player needs to connect the left and the right and the goal of the third player is to connect the front and the back.
In turn all three players place a piece of their color on one of the fields in the cube. One of the players wins if he/she manages to connect his two sides of the playboard. The game ends in a draw if all fields in the cube contain a piece belonging to one of the players and no one managed to make an unbroken chain from one of his/her sides to the other. This game may end in a draw, because there are now two players who can block the third. So if one player is blocked, there is not necessarily a winner, in contrast to 2D Hex. To keep up with the different chains made by the players, we will use Union Find, more on Union Find can be found in [13]. It was proven that three-player games are hard, see [11] and [19].

3 Brute-force solutions

By solving the game of 3D Hex, we can for sure say who will win the game given perfect play. The problem statement of this section is therefore:

"How can we solve 3D Hex, to find out who will win the game given perfect play?"

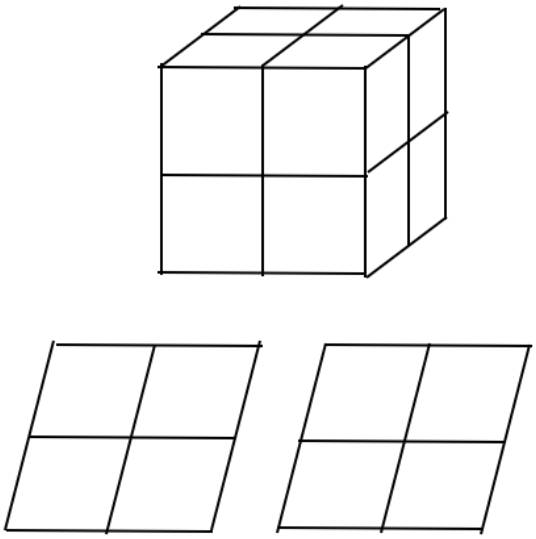

Figure 4: A 3D Hex playboard with sides of size 2, the real representation and the representation that is going to be used in this paper.

3.1 Sides of size 2

When we look at 3D Hex with sides of size 2, we get a playboard like shown in Figure 4. In this figure also the 2D representation of a configuration is shown. Here the first field represents the front and the second field represents the back of the cube. This representation will be used in the following proof that shows that $2 \times 2 \times 2$ 3D Hex, if all players play optimal, always ends in a draw. Also with a simple brute-force program it was confirmed that the game ends in a draw.

We looked at the gametree that can be drawn to check how this game will end. We can calculate the number of states by knowing that in each field there are four possible values: the field can be in possession of P1, P2, P3 or can be empty. For a $2 \times 2 \times 2$ gameboard there are eight fields. The maximum amount of states is therefore: $4^8 = 65536$. Some of these states will never be reached, because for example you cannot have only '1's on the whole gameboard.

About the size of the gametree we can say that in the first turn P1 can do eight different moves. In the next turn there are only seven moves left to choose from for P2 and so on. So the size of the gametree is $8! = 40320$ configurations. A number of these configurations is reachable and the rest can already be won in a previous stage. This tree, with at most 40320 nodes, is really big and hard to draw. For larger gameboards the tree is even larger, see Table 2. So we need another way to examine the game.

Size of the gameboard	Maximum number of configurations
$2 \times 2 \times 2$	$8! = 40320$
$3 \times 3 \times 3$	$27! = 1.0888869 \times 10^{28}$
$4 \times 4 \times 4$	$64! = 1.2688693 \times 10^{89}$

Table 2: The upperbound for the size of the game tree.

3.1.1 Proof: $2 \times 2 \times 2$ 3D Hex always ends in a draw

Let us take a look at Figure 5. We see the 2D representation of the 3D gameboard. The first square is the front side and the second square is the back side. On the right of the 2D representation of the gameboard the direction in which the players play is displayed. In the figure the only possible first move is done by P1 — because of symmetry all possible first moves are the same, they all cover one corner of the cube. This player needs to connect the top and the bottom of the cube. So the only possible move for the first player to win by the next turn is to do the move marked with a gray '1'.

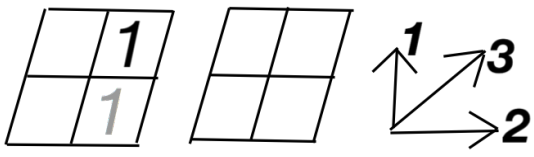

Figure 5: A 2D representation of the only possible first move for P1.

So when a player places a piece on the gameboard, he only has one possible next move to win the game here indicated with the gray '1'. P2 now has two options:

1. Block P1.
2. Do another move and hope to win himself.

Let us first look at the first option. He can block P1 by placing his piece on the gray '1'. This situation is displayed in Figure 6. The only possible move for P2 to win in the next turn is shown with a gray '2'.

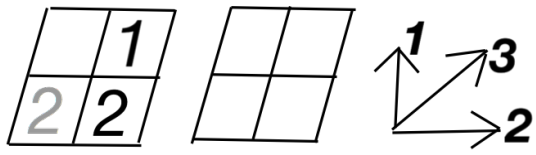

Figure 6: P2 blocks P1.

Now P1 has no chance of winning the game in the next turn, but P2 does. But because there always is one effective counter move, the next player (P3) can block this chance of winning.

Let us take a look at the other option P2 had to not block P1, but do another move displayed in Figure 7.

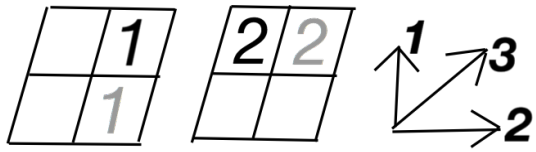

Figure 7: P2 does another move.

Now P1 and P2 both have a chance of winning in their next turn. It is now P3's turn. He can choose between the following options:

1. Block P1.
2. Block P2.
3. Don't block and play another move.

The next player after P3 is P1. So if P3 does not block him, he will win in the next turn. So P3 cannot do other than choose for the first option.

We can now say that in games where there is always one effective counter move the game always ends in a draw, because each player can always block the next player if he is winning or do another move if he is not winning at that moment. So the argument here is in fact: if a player is about to win, he can always be blocked by the previous player, therefore Hex 3D $2 \times 2 \times 2$ always ends in a draw.

3.2 Sides of size 3

When we play 3D Hex on a playboard with sides of size 3, then there is not always one effective counter move, so we cannot use the argument from the previous subsection that the game always ends in a draw.

To use the computer to compute the result of a game given perfect play, we need to know how long this will take. Table 3 gives us an idea on how long it will take to run our program from a certain depth in the gametree. In the first column the depths are given. This means that to that depth the game was played random and from there solved exactly. So if you want to solve the game from depth 21, then you need to do 21 random moves and then brute-force solve it for the remaining six moves and this will take 0.001 seconds according to the table.

From depth	Runtime	Size: maximum number of configurations
21	0.001s	$6! = 720$
20	0.007s	$7! = 5040$
19	0.048s	$8! = 40320$
18	0.384s	$9! = 362880$
17	4.065s	$10! = 3628800$
16	8m 52.849s	$11! = 39916800$
15	6m 35.951s	$12! = 479001600$
14	9u 8m 30.837s	$13! = 6227020800$

Table 3: The runtime to brute-force solve $3 \times 3 \times 3$ 3D Hex from a certain depth.

From this table we can see that it takes a long time to solve $3 \times 3 \times 3$ Hex. To still get an idea of the outcome of the game, Monte-Carlo Tree Search is used in the next chapter.

These experiments were run on the terminal, with version 2.5.3 (343.7) and gcc version 4.2.1, of a MacBook Pro with a 2.6 GHz Intel Core i5 processor and with OS X Yosemite version 10.10.4.

4 Applying Monte-Carlo Tree Search

Solving the game of $3 \times 3 \times 3$ 3D Hex takes a long time, but to get an idea of a good strategy for the game Monte-Carlo Tree Search is going to be used. First several experiments will be done and then a good strategy for the players will be derived by looking at the most played moves. The problem statement of this section is therefore:

> "How can we apply Monte-Carlo Tree Search to the game of $3 \times 3 \times 3$ 3D Hex, to find a good strategy for the players?"

4.1 Method

Our implementation of 3D Hex is used to produce the results from the next section. `C++` is used to implement Hex 3D. The cube that represents the playboard is a three-dimensional array where all fields are initially set to a character that represents an empty field. If one of the players plays a turn, the empty field is set to his corresponding number.

To check if the game is won Union-Find is used. For all chains on the gameboard made by the same player there exists a set. For every new move a set is produced containing only the field where this move was done. Then the program checks if this set is adjacent to one of the elements in another already existing set containing fields that are occupied by the same player. If so this set with the newly occupied field is merged with the already existing set, because it also belongs to that chain if it is occupied by the same player as the elements in the already existing set and if it is adjacent to one of the elements in the set. For each set it is kept up if there is an element in the set that touches one side of the playboard and if there is an elements that touches the other side. This way it can be checked if someone has won, because if he/she managed to connect both sides and all the elements in the set are adjacent fields from the player, then the game is won.

Monte-Carlo Tree Search selects a next move by checking all possible moves from the present configuration and for each of these moves a number of random playouts will be played. It checks how many of these random playouts the player whose turn it is wins and how many times any of the other players wins or the game ends in a draw. The outcomes are each given a value: the outcome weight, later on we will discuss how these values will be chosen. The outcome weight belonging to a certain outcome of a random playout is added to the value of the move. So the value of the move is in fact the sum of all outcome weights corresponding to the outcomes of the random playouts. The function eventually selects the move that is assigned the most points, so the move that seems to be the best at that point. This move is played. To do the random playouts with Monte-Carlo Tree Search we need a function that plays the game randomly. This function checks how many fields are empty and generates a random number to select one of these fields.

4.2 Union find

The *union find* or *disjoint-set* data structure is a data structure that keeps track of a number of non-overlapping sets of elements, see [13]. Each of these sets is represented by the root, which is called the representative of the set. This data structure uses three operations:

Make(X) Creates a new set with one single element X.

Find(X) Checks in which subset an element X is contained and returns the representative of that subset.

Union(X,Y) Joins two subsets X and Y into a single subset containing all elements of X and Y exactly once.

4.3 Monte-Carlo Tree Search

Monte-Carlo Tree Search (MCTS) is a strategy used to build a program that plays games very well. It constructs a search tree by evaluating possible moves. It has successfully been used to optimize game playing programs, see [12]. Monte-Carlo Tree Search checks for every possible move the outcome of the game by doing random playouts on a playboard from that move and further. From the results of these random playouts a search tree is built. When all possible moves are given a value based on the outcomes of the random playouts and the search tree is built, the best move is selected and played by the player whose turn it is, see [10]. An illustration of MCTS is shown in Figure 8.

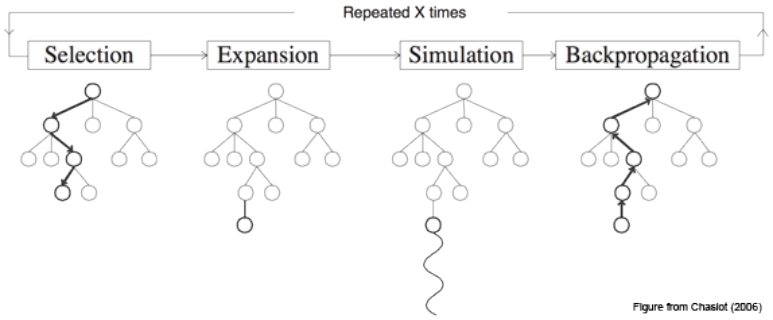

Figure 8: A search tree for the strategy of Monte-Carlo Tree Search. From: [5].

These are several variants of MCTS. For example there is an Upper Confidence Bounds (UCB) formula that is used to choose the node that will be expanded: $v_i + C \times \sqrt{\frac{\ln N}{n_i}}$. In this formula v_i stands for the estimated value of the node, n_i is the number of times this node has been visited during the game and N is the number of times that its parent has been visited during the game. C is a parameter that can be changed. There is also a variant of MCTS using Upper Confidence Bounds for Trees (UCT). In this paper we will use the basic Monte-Carlo Tree Search algorithm, without UCB or UCT.

4.4 Evaluation of Monte-Carlo moves

When our program searches for the best move by using the algorithm of Monte-Carlo Tree Search, for every possible move from that configuration a number of playouts are done to see which move yields on average the best result. To assign points to these moves by random playouts, four outcome weights are needed. The first outcome weight denotes the points added to the value of that move if in the random playout P1 wins, the second weight denotes the points added to the value of that move if P2 wins and the third weight denotes the points added to the value of that move if P3 wins. If the game ends in a draw, then the fourth weight is added to the value of that move.

So when it is P1's turn, the function that plays the Monte-Carlo Tree Search strategy might be called with outcome weights 3, 0, 0, 1 respectively. With Monte-Carlo Tree Search it is checked who wins the random playout game by playing random moves. If P1 himself wins, then the value of that move is increased by 3 points. If P2 or P3 wins the game, nothing happens. And if the game ends in a draw, 1 point is added to the value of that move. If all moves are checked and assigned a value, the move with the highest value will be played. This way P1 is more likely to win with this move.

4.5 Nature of the players

Table 1 in the introduction summed up all the natures of players we will use in this paper. There are five natures that will be studied in the next section.

Competitive – selfish: Every player wants to win himself, there is no cooperation whatsoever. If the player himself wins the random playout, the value of the move is increased by 3 points. If any of the other players wins the value stays the same and if the game ends in a draw the value of the move is increased by 1 point. See Table 4 for the outcome weights.

Player	Outcome Weights
P1	3 0 0 1
P2	0 3 0 1
P3	0 0 3 1

Table 4: Competitive – selfish.

Cooperative level 1: Every player wants most to win himself, but now there exist 'teams'. So a player that is in a team also kind of wants his teammate to win. If the player himself wins the random playout, the value of the move is increased by 3 points. If his teammate wins or if the game ends in a draw the value of the move is increased by 1 point. If another player wins the value stays the same. See Table 5 for the outcome weights.

Player	Outcome Weights		
	Team: P1, P2	Team: P1, P3	Team: P2, P3
P1	3 1 0 1	3 0 1 1	3 0 0 1
P2	1 3 0 1	0 3 0 1	0 3 1 1
P3	0 0 3 1	1 0 3 1	0 1 3 1

Table 5: Cooperative level 1.

Cooperative level 2: Every player wants most to win himself, but now there exist 'teams'. So a player that is in a team also kind of wants his teammate to win. If the player himself wins the random playout, the value of the move is increased by 3 points. If his teammate wins the value of the move is increased by 2 points, if the game ends in a draw the value is increased by 1 point and if another player wins the value stays the same. See Table 6 for the outcome weights.

Player	Outcome Weights		
	Team: P1, P2	Team: P1, P3	Team: P2, P3
P1	3 2 0 1	3 0 2 1	3 0 0 1
P2	2 3 0 1	0 3 0 1	0 3 2 1
P3	0 0 3 1	2 0 3 1	0 2 3 1

Table 6: Cooperative level 2.

Cooperative level 3: Every player wants most to win himself, but now there exist 'teams'. A player that is in a team is equally satisfied if his teammate wins as he would be if he himself won. If the player himself wins the random playout or his teammate wins, the value of the move is increased by 3 points. If the game ends in a draw the value is increased by 1 point and if another player wins the value stays the same. See Table 7 for the outcome weights.

Player	Outcome Weights		
	Team: P1, P2	Team: P1, P3	Team: P2, P3
P1	3 3 0 1	3 0 3 1	3 0 0 1
P2	3 3 0 1	0 3 0 1	0 3 3 1
P3	0 0 3 1	3 0 3 1	0 3 3 1

Table 7: Cooperative level 3.

Misère: Every player wants to lose himself. If the player himself wins the random playout, the value of the move stays the same. If any of the other players wins the value of the move is increased by 3 points and if the game ends in a draw the value of the move is increased by 1 point. See Table 8 for the outcome weights.

Player	Outcome Weights
P1	0 3 3 1
P2	3 0 3 1
P3	3 3 0 1

Table 8: Misère.

5 Experiments

These experiments were run on the terminal, with version 2.5.3 (343.7) and gcc version 4.2.1, of a MacBook Pro with a 2.6 GHz Intel Core i5 processor and with OS X Yosemite version 10.10.4. The experiments are explorative, so not exhaustive.

5.1 Number of games

For the experiments in the next section it is chosen to play 10000 games and check how many times the players win. As a first indication of statistical stability, the same experiment with 1000 playouts was run with different numbers of games. The results are shown in Table 9. The outcome weights that are used here are from misère, also used in one of the experiments in the experiments section.

Number of games	P1 wins	P2 wins	P3 wins	Draw
100	19	8	28	45
500	64	57	102	277
1000	137	107	216	540
1500	217	160	309	814
5000	736	564	1034	2666
9000	1342	1038	1808	4812
10000	1475	1149	2029	5347
11000	1600	1260	2287	5853

Table 9: Misère play and 1000 playouts per move with different numbers of games as a first indication of statistical stability.

We can see that the percentages are as follows:

Number of experiments	P1 wins	P2 wins	P3 wins	Draw
100	19.0%	8.0%	28.0%	45.0%
500	12.8%	11.4%	20.4%	55.4%
1000	13.7%	10.7%	21.6%	54.0%
1500	14.5%	10.7%	20.6%	54.3%
5000	14.7%	11.3%	20.7%	53.3%
9000	14.9%	11.5%	20.1%	53.5%
10000	14.8%	11.5%	20.3%	53.5%
11000	14.5%	11.5%	20.8%	53.2%

Table 10: Percentages derived from Table 9.

The difference between the run with 100 games and the run with 11000 games is at most 8.2 percentage points and on average 5.9 percentage points. This is a big difference, so we cannot say that 100 games are sufficient.

We see that in the next rows the values are getting more and more stable. The difference between the run with 9000 games and the run with 10000 games is at most 0.2 percentage points and on average 0.08 percentage points. The difference between the run with 10000 games and the run with 11000 games is at most 0.5 percentage points and on average 0.22 percentage points. This is negligible in comparison with the differences between the values of P1, P2, P3 and draw. So 10000 games seems to be enough to get reliable results.

5.2 Number of playouts

For the experiments it is chosen to do 1500 random playouts for each move to chose the best move with Monte-Carlo Tree Search. As a first indication of statistical stability, the same experiment with 10000 games was run with different numbers of playouts. The results of these experiments are shown in Table 11. The outcome weights that are used here are from misère, also used in one of the experiments in the experiments section.

Number of playouts	P1 wins	P2 wins	P3 wins	Draw
500	1378	1375	2148	5099
900	1052	952	2421	5575
1000	1467	1154	2039	5340
1100	1344	1354	1887	5415
1250	1413	874	1862	5851
1500	1425	944	1893	5738
1600	1410	953	1986	5651

Table 11: Misère play and 10000 games per move with different numbers of playouts as a first indication of statistical stability.

We can see that the percentages are as follows:

Number of playouts	P1 wins	P2 wins	P3 wins	Draw
500	13.8%	13.8%	21.5%	60.0%
900	10.5%	9.5%	24.2%	55.8%
1000	14.7%	11.5%	20.4%	53.4%
1100	13.4%	13.5%	18.9%	54.2%
1250	14.1%	8.7%	18.6%	58.5%
1500	14.3%	9.4%	18.9%	57.4%
1600	14.1%	9.5%	19.9%	56.5%

Table 12: Percentages derived from Table 11.

The difference between the run with 500 playouts and the run with 1600 playouts is at most 4.3 percentage points and on average 2.43 percentage points. This is a big difference, so we cannot say that 500 playouts are sufficient.

We see that in the next rows the values are getting more and more stable. The difference between the run with 1250 playouts and the run with 1500 playouts is at most 1.1 percentage points and on average 0.58 percentage points. The difference between the run with 1500 playouts and the run with 1600 playouts is at most 1.0 percentage points and on average 0.55 percentage points. This is almost negligible in comparison with the differences between the values of P1, P2, P3 and draw. Furthermore, if we would increase the number of playouts, the runtime would also increase. With this in mind, 1500 playouts seemed to be enough to get reliable results.

5.3 Results

The number of games chosen is 10000, which seems to be sufficient. The number of random playouts that is going to be used is 1500. These numbers are an estimation, it is not statistically significant, the previous subsection contains the justification for using these numbers. In this section we will first study the whole game and then we will check the first three of the moves that are done the most from the start-position.

5.3.1 Study of a whole game: sides of size 3

In Table 13 the nature of the players is shown in the first column, the second column indicates the teams and the following columns show how many times the game was won by each of the players or ended in a draw. The last column shows the runtime. The experiments were run on a $3 \times 3 \times 3$ board.

Nature of the players	Team	P1 wins	P2 wins	P3 wins	Draw	Runtime
Competitive – selfish	nA	17	0	0	9983	
Cooperative level 1	P1, P2	561	100	0	9339	268m12.201s
	P2, P3	0	358	591	9051	255m19.007s
	P1, P3	4	8	139	9849	
Cooperative level 2	P1, P2	8723	289	7	981	
	P2, P3	0	9670	90	240	255m19.007s
	P1, P3	19	0	8066	1915	
Cooperative level 3	P1, P2	9267	249	0	484	
	P2, P3	0	9578	159	263	
	P1, P3	31	0	9236	733	
Misère	nA	1425	944	1893	5738	

Table 13: Experiments on a $3 \times 3 \times 3$ board using Monte-Carlo Tree Search with 1500 playouts per move and 10000 games. The average runtime for each experiment was 4–5 hours.

5.3.2 Study of a whole game: sides of size 4

In Table 14 the nature of the players is shown in the first column, the second column indicates the teams and the following columns show how many times the game was won by each of the players or ended in a draw. The last column shows the runtime. The experiments were run on a $4 \times 4 \times 4$ board. We now do 1000 experiments instead of 10000, because otherwise the experiments will take a long time running.

Nature of the players	Team	P1 wins	P2 wins	P3 wins	Draw	Runtime
Competitive – selfish	nA	17	3	0	980	418m1.136s
Cooperative level 1	P1, P2	30	0	6	964	404m4.388s
	P2, P3	0	53	1	946	405m43.984s
	P1, P3	42	1	15	942	406m30.125s
Cooperative level 2	P1, P2	922	5	0	73	340m22.886s
	P2, P3	0	963	2	35	326m16.588s
	P1, P3	8	0	953	39	216m11.133s
Cooperative level 3	P1, P2	997	0	0	3	300m31.391s
	P2, P3	1	996	0	3	308m1.623s
	P1, P3	0	0	994	6	328m12.285s
Misère	nA	0	0	0	1000	551m7.533s

Table 14: Experiments on a $4 \times 4 \times 4$ board using Monte-Carlo Tree Search with 1500 playouts per move and 1000 games. The average runtime for each experiment was 6–7 hours.

5.3.3 Study of a whole game: exclude fields

In Table 15 the nature of the players is shown in the first column, the second column indicates the teams and the following columns show how many times the game was won by each of the players or ended in a draw. The last column shows the runtime. The experiments were run on a $3 \times 3 \times 3$ board, with the central field excluded.

5.3.4 Study of the best moves: sides of size 3

For the start position of the game on a $3 \times 3 \times 3$ playboard, it was checked how many times each move is selected by the algorithm of Monte-Carlo Tree Search and thus done by P1. The move that was chosen the most, will be studied further. Below in Figure 9 there is displayed how many times the moves are chosen by P1. The number of playouts is 10000 so that we get more reliable results and the number of games 1000. The outcome weights for the players that were used are shown in Table 16 below.

In Figure 9, 10 and 11 the 3D playboard is displayed in 2D. The first square is the front side, the second is the median plane, and the third is the back side. On the right of the 2D representation of the gameboard the direction in which the players play is displayed. Now we mark the first move chosen 1000 times out of 1000 by P1 and check what will be the distribution in the next

move by P2, see Figure 10. Both these moves are equivalent by symmetry so we proceed with one of the two. In Figure 11 we can see the distribution of the moves chosen by Monte-Carlo Tree Search for P3.

Nature of the players	Team	P1 wins	P2 wins	P3 wins	Draw	Runtime
Competitive – selfish	nA	0	0	32	9968	241m5.989s
Cooperative level 1	P1, P2	914	179	0	8907	241m6.484s
	P2, P3	55	1141	740	8064	239m19.577s
	P1, P3	16	60	156	9768	238m57.750s
Cooperative level 2	P1, P2	6547	763	15	2675	230m39.497s
	P2, P3	29	8153	1344	474	224m48.447s
	P1, P3	267	0	5643	4090	231m26.519s
Cooperative level 3	P1, P2	7246	545	57	2152	224m17.066s
	P2, P3	1	9457	386	156	209m32.423s
	P1, P3	150	0	5920	3930	234m9.605s
Misère	nA	32	754	1123	8091	263m34.720s

Table 15: Experiments on a $3 \times 3 \times 3$ board with the central field excluded using Monte-Carlo Tree Search with 1500 playouts per move and 10000 games. The average runtime for each experiment was 4 hours.

Player	Outcome Weights
P1	3 0 0 1
P2	0 3 0 1
P3	0 0 3 1

Table 16: Competitive – selfish.

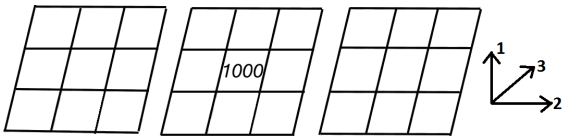

Figure 9: The first move played by P1 (remaining squares have ∅).

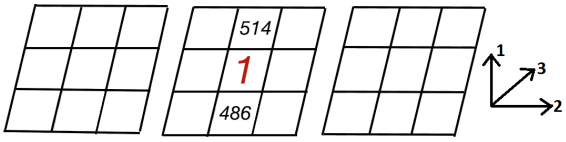

Figure 10: The second move played by P2 (remaining squares have ∅).

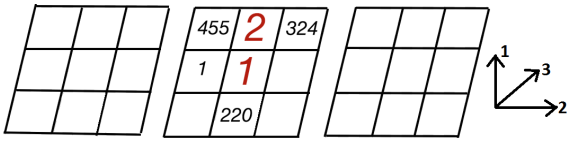

Figure 11: The third move played by P3 (remaining squares have ∅).

5.3.5 Study of the best moves: sides of size 4

For the start position of the game on a $4 \times 4 \times 4$ playboard, it was checked how many times each move is selected by the algorithm of Monte-Carlo Tree Search and thus done by P1. The move that was chosen the most, will be studied further. Below in Figure 12 there is displayed how many times the moves are chosen by P1. The number of playouts is 10000 so that we get more reliable results and the number of games 1000. The outcome weights for the players that were used are shown in Table 17 below.

In Figure 12, 13 and 14 the 3D playboard is displayed in 2D. The first square is the front side, the second and third are the following median planes and the fourth is the back side. On the right of the 2D representation of the gameboard the direction in which the players play is displayed. Now we mark the first move chosen 489 times out of 1000 by P1 and check what will be the distribution in the next move by P2, see Figure 13. We proceed with the most done move that was chosen 895 times out of 1000. In Figure 14 we can see the distribution of the moves chosen by Monte-Carlo Tree Search for P3.

Player	Outcome Weights
P1	3 0 0 1
P2	0 3 0 1
P3	0 0 3 1

Table 17: Competitive – selfish.

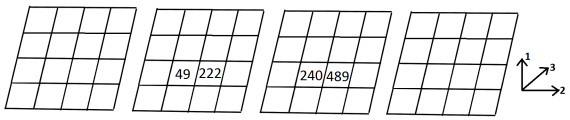

Figure 12: The first move played by P1 (remaining squares have ∅).

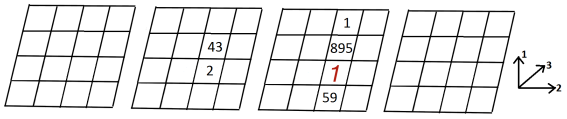

Figure 13: The second move played by P2 (remaining squares have ∅).

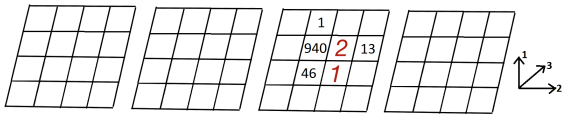

Figure 14: The third move played by P3 (remaining squares have ∅).

5.3.6 Study of the best moves: exclude fields

For the start position of the game on a $3 \times 3 \times 3$ playboard with the middle field excluded, it was checked how many times each move is selected by the algorithm of Monte-Carlo Tree Search and thus done by P1. The move that was chosen the most, will be studied further. Below in Figure 15 there is displayed how many times the moves are chosen by P1. The number of playouts is 10000 so that we get more reliable results and the number of games 1000. The outcome weights for the players that were used are shown in Table 18 below.

In Figure 15, 16 and 17 the 3D playboard is displayed in 2D. The first square is the front side, the second is the median planes and the third is the back side. On the right of the 2D representation of the gameboard the direction in which the players play is displayed. Now we mark the first move chosen 477 times out of 1000, that is equivalent by symmetry to the move chosen 470 times out of 1000 by P1, and check what will be the distribution in the next move by P2, see Figure 16. We proceed with the most done move that was chosen 1000 times out of 1000. In Figure 17 we can see the distribution of the moves chosen by Monte-Carlo Tree Search for P3.

Player	Outcome Weights
P1	3 0 0 1
P2	0 3 0 1
P3	0 0 3 1

Table 18: Competitive – selfish.

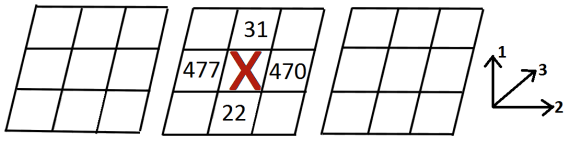

Figure 15: The first move played by P1 (remaining squares have ∅).

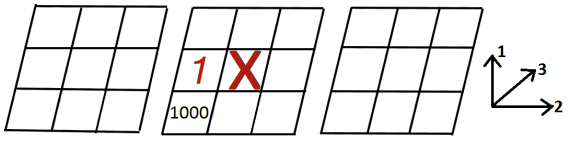

Figure 16: The second move played by P2 (remaining squares have ∅).

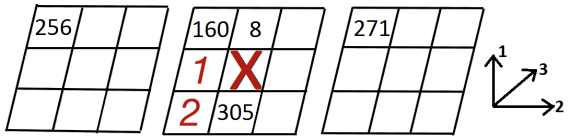

Figure 17: The third move played by P3 (remaining squares have ∅).

5.4 Discussion

From the results in the previous section we can derive some graphs. We will study them and we will also discuss and try to explain what we conclude from them.

5.4.1 Competitive – selfish.

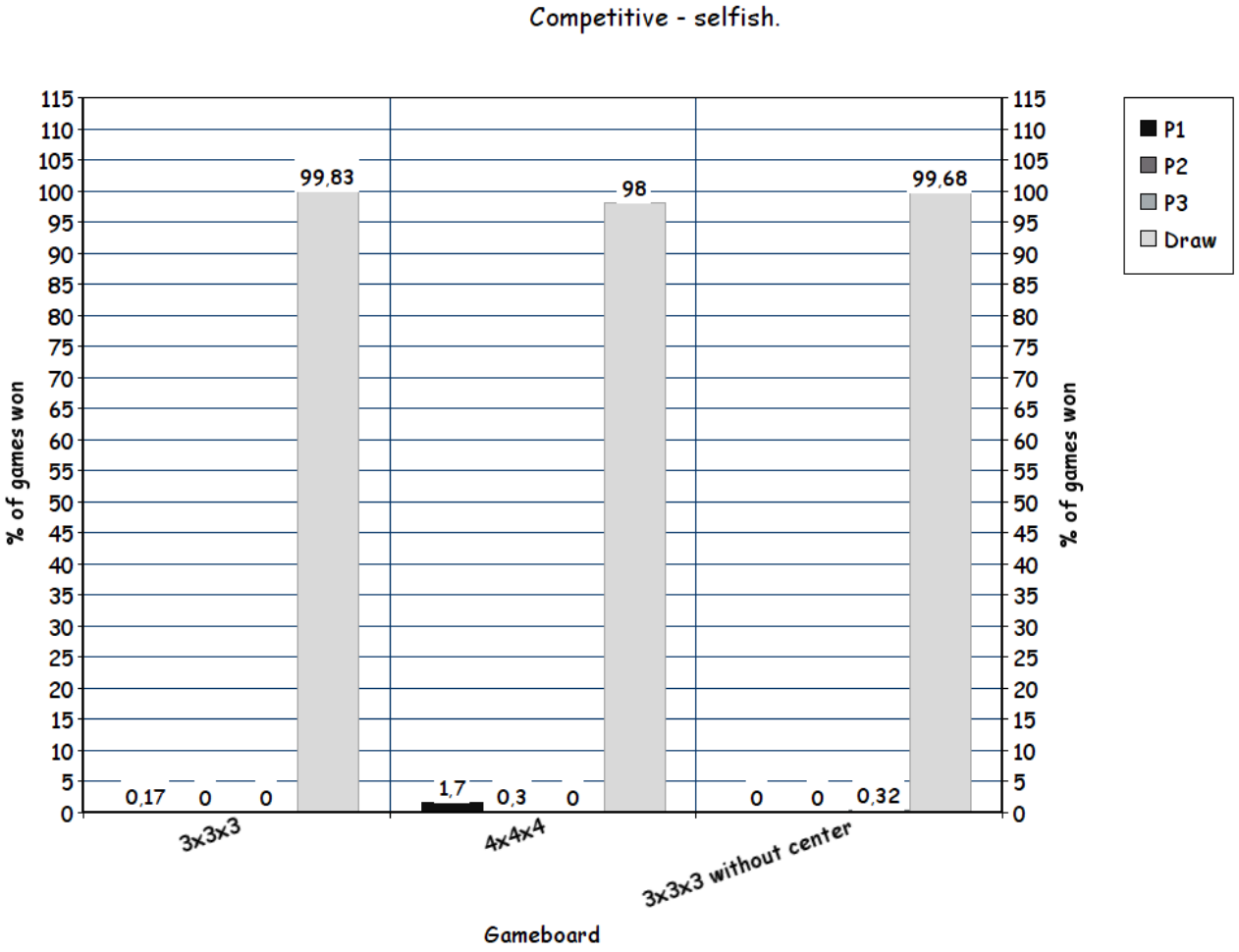

Figure 18: A graph with the results from the experiments with competitive – selfish outcome weights.

In the graph displayed in Figure 18 we can see that most of the games, on average 99.17%, ended in a draw if the players played competitive – selfish. We can also see that on a $3 \times 3 \times 3$ playboard none of the games was won by P2 of P3 and only 0.17% was won by P1. On a

$4 \times 4 \times 4$ playboard none of the games was won by P3, 0.30% was won by P2 and 1.70% was won by P1. We see that P1 has some kind of advantage over P2 and P3 by being able to do the first move. When we do the same experiment on a $3 \times 3 \times 3$ playboard where we exclude the center field, we see that none of the games is now won by P1 or P2, but now 0.32% is won by P3. This can be explained by the previous section where we studied the best moves. The best move, and most done move, by P1 from start-position is the center field. So when we exclude this field, P1 has to do another move that maybe will result in a draw or loss.

5.4.2 Misère.

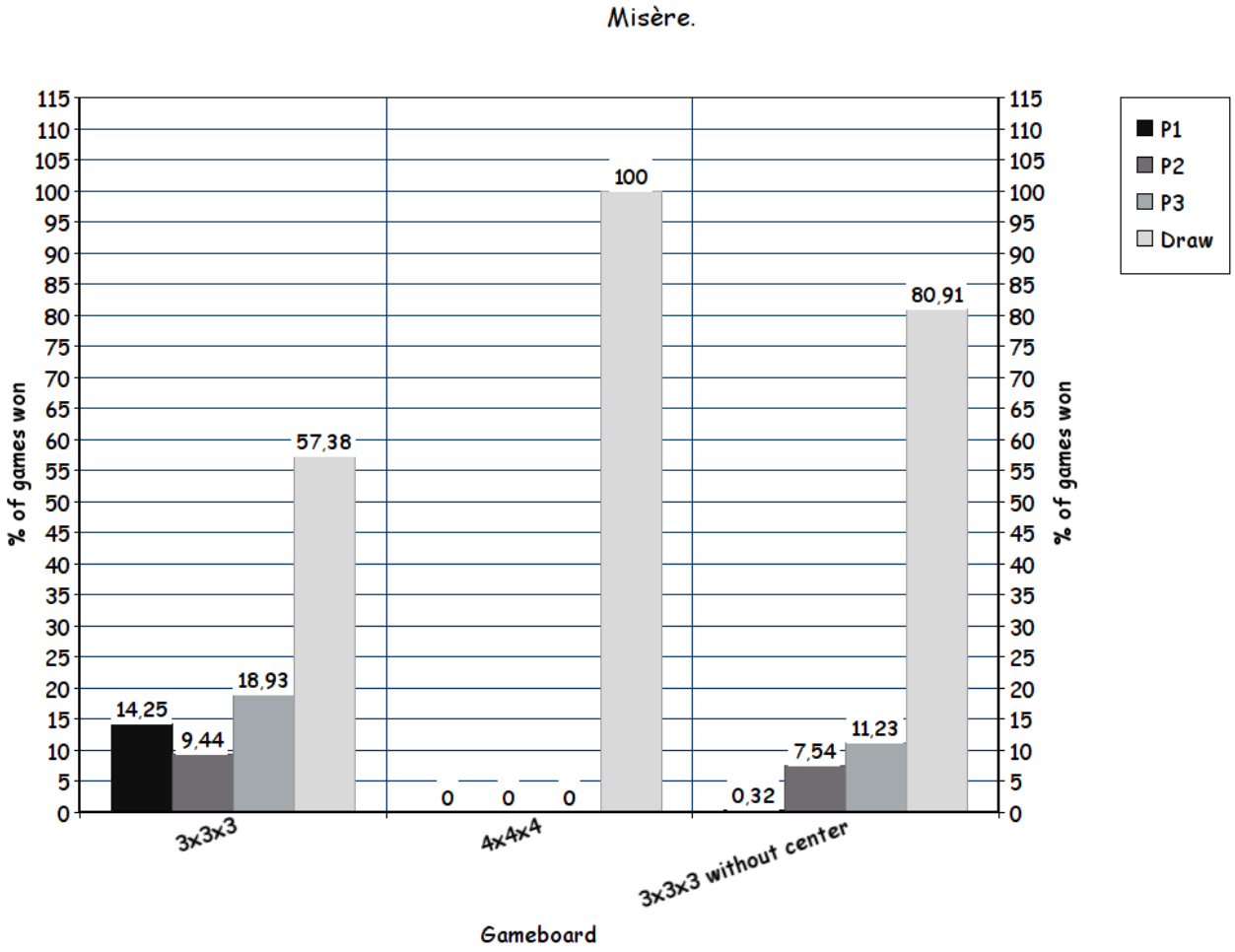

Figure 19: A graph with the results from the experiments with misère outcome weights.

In the graph displayed in Figure 19 we can see that most of the games, on average 79.43%, ended in a draw if the players played misère. We can also see that on a $3 \times 3 \times 3$ playboard P1 en P3 have the biggest advantage, P3 wins 18.93% of the games and P1 wins 14.25%. P2 wins 9.44% of the games. On a $4 \times 4 \times 4$ playboard none of the games was won by any of the players, all games ended in a draw. When we do the same experiment on a $3 \times 3 \times 3$ playboard where we exclude the center field, we see that 0.32% of the games is won by P1, 7.54% by P2 and 11.23% by P3. The low percentage for P1 in comparison with the gameboard that has the center field, can be explained by the previous section where we studied the best moves.

The best move, and most done move, by P1 from start-position is the center field. So when we exclude this field, P1 has to do another move that maybe will result in a draw or loss.

5.4.3 Cooperative.

In the next graphs displayed in Figures 20, 21, 22 and 23 the gameboard and level of cooperation are given on the x-axis. The gameboard is noted as $3 \times 3 \times 3$ or $4 \times 4 \times 4$. A $3 \times 3 \times 3$ playboard with the middle field excluded is named '$3 \times 3 \times 3$ e'. 'C L1' stands for cooperative level 1 and 'C L2' for cooperative level 2. 'C L3' stands for cooperative level 3.

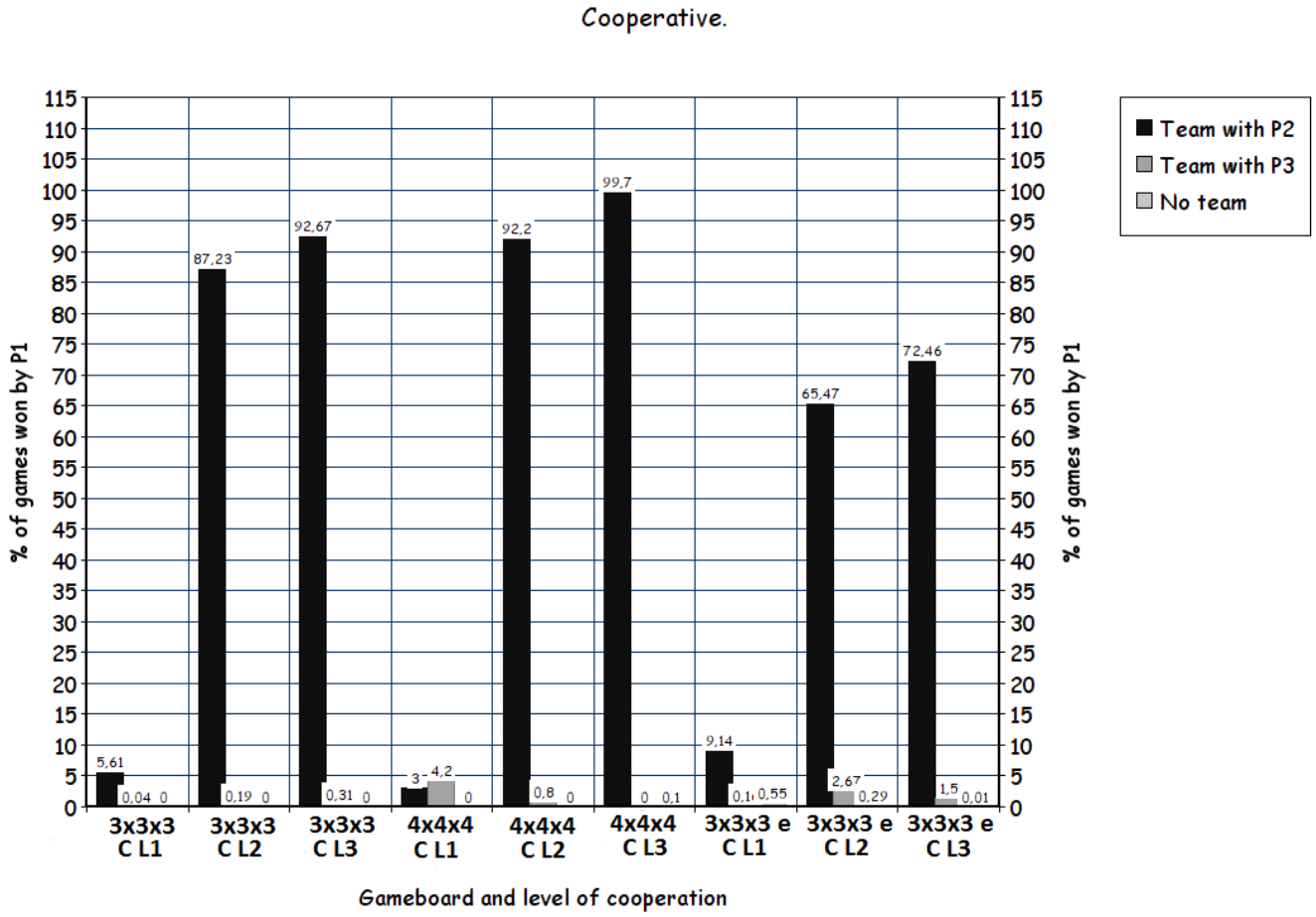

Figure 20: A graph with the results from the experiments with cooperative outcome weights that shows the wins of P1.

The first thing that stands out in the graph displayed in Figure 20 are the black bars, they denote the percentage of wins of P1 if he plays in a team with P2. We can take a closer look at these percentages to see which factors yield advantage. Let us first look at the different playboards and derive on which playboard P1 probably has the best chance of winning. The average percentage of wins on a $3 \times 3 \times 3$ playboard is 51.84%, on a $4 \times 4 \times 4$ playboard 64.97% and on a $3 \times 3 \times 3$ playboard with the middle field excluded 49.02%. So on a $4 \times 4 \times 4$ playboard P1 has the best chance of winning if he plays in a team with P2. He has the least chance on a $3 \times 3 \times 3$ playboard with the middle field excluded, this can be explained by the fact that this first center move gives P1 some kind of advantage over the rest if the players.

Moreover, the more P1 cooperates with P2, the more wins for P1. This can be explained by the fact that now also P2 wants P1 to win, so he will in most cases not block the winning moves for P1 anymore. If P1 teams up with P3, he has a really small chance of winning, but if he is not in a team at all the chances are even smaller.

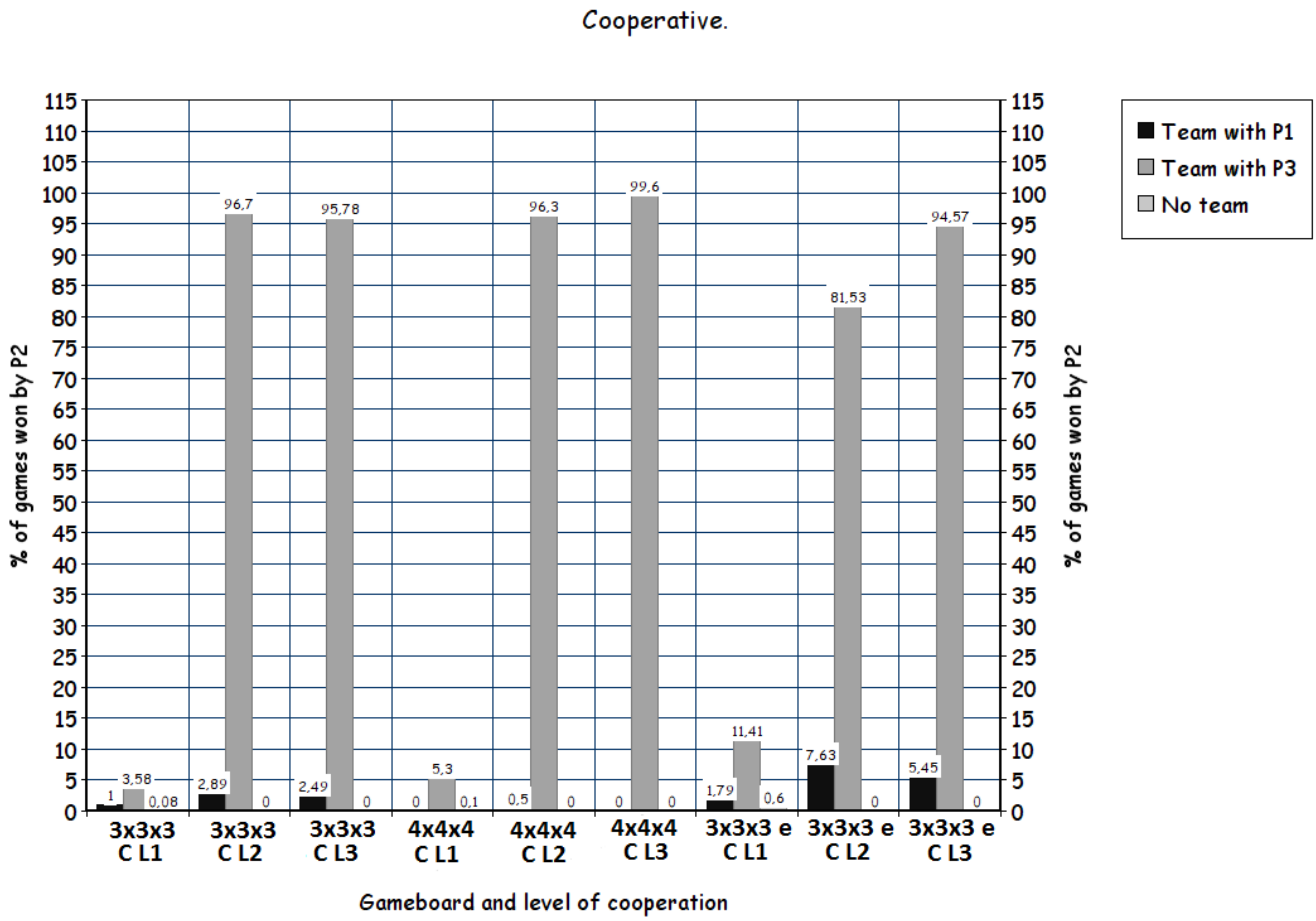

Figure 21: A graph with the results from the experiments with cooperative outcome weights that shows the wins of P2.

The first thing that stands out in the graph displayed in Figure 21 are the dark gray bars. These bars denote the percentage of wins of P2 if he plays in a team with P3. We can take a closer look at these percentages to see which factors yield advantage. Let us first look at the different playboards and derive on which playboard P2 probably has the best chance of winning. The average percentage of wins on a $3 \times 3 \times 3$ playboard is 65.35%, on a $4 \times 4 \times 4$ playboard it is 67.07% and on a $3 \times 3 \times 3$ playboard with the middle field excluded it is 62.50%. So on a $4 \times 4 \times 4$ playboard P2 has the best chance of winning if he plays in a team with P3 and the least chance on a $3 \times 3 \times 3$ playboard with the middle field excluded. We now see for the second time that playing on a $4 \times 4 \times 4$ playboard is an advantage for the players, this can be explained by the fact that the more cooperation there is on a $4 \times 4 \times 4$ playboard, the less games end in a draw and thus the more games are won by one of the players. In Figure 23 we see that the number of draws decrease faster when played on a board with size 4 in comparison with when they play on any other board. We can also see that the more P2 cooperates with P3, the more wins for P2 there are. This can be explained by the fact that

now also P3 wants P2 to win, so he will in most cases not block the winning moves for P2 anymore. When they play on a $3 \times 3 \times 3$ playboard we see that the wins stay about equal when the level of cooperation goes from level 2 to level 3, but the percentage of wins for P2 is then very high, that can be the reason why it did not increase. The last thing we will derive is that if P2 teams up with P1, he has a really small chance of winning, but if he is not in a team at all the chances are even smaller.

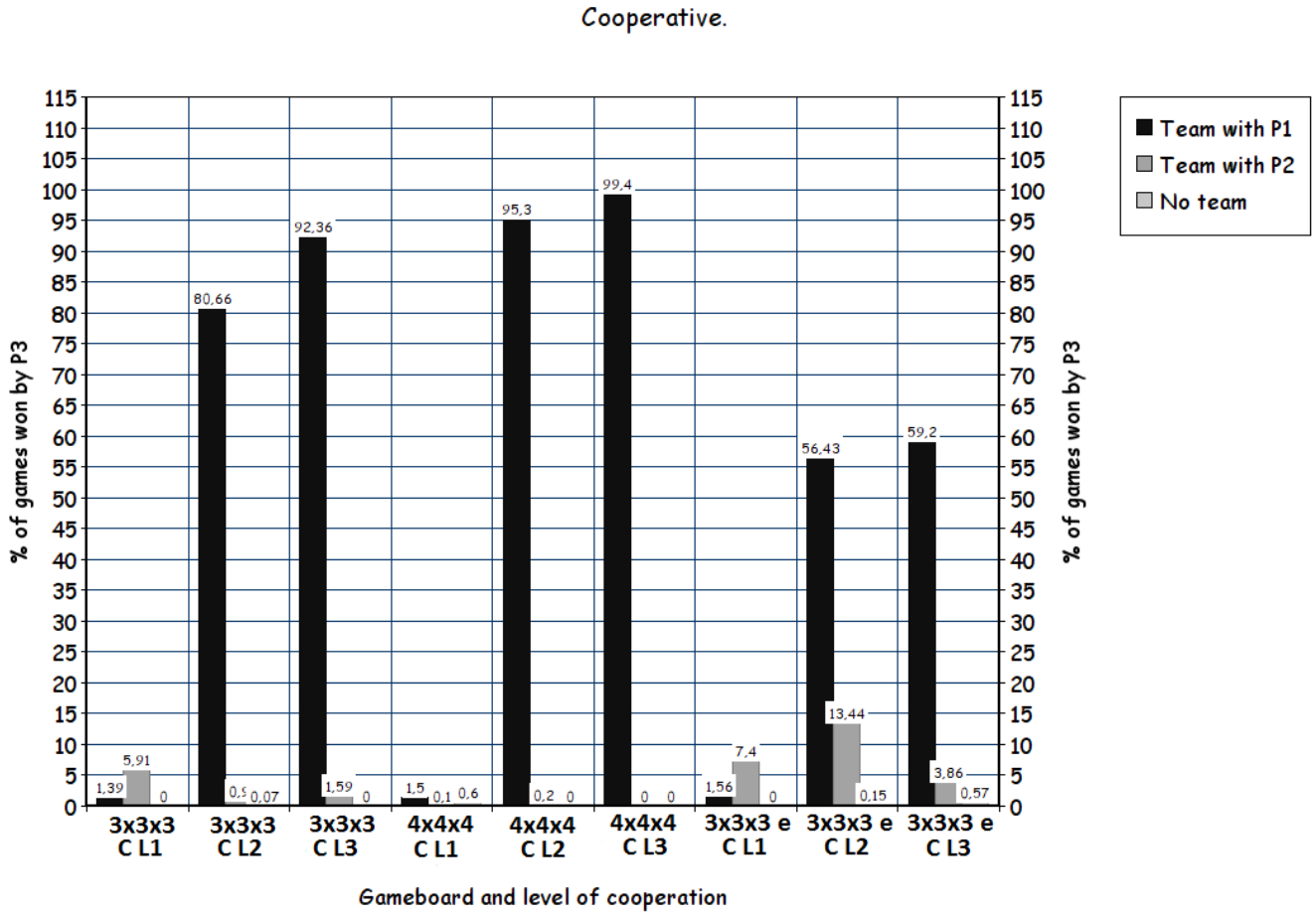

Figure 22: A graph with the results from the experiments with cooperative outcome weights that shows the wins of P3.

The first thing that stands out in the graph displayed in Figure 22 are the black bars. These bars denote the percentage of wins of P3 if he plays in a team with P1. We can take a closer look at these percentages to see which factors yield advantage. Let us first look at the different playboards and derive on which playboard P3 probably has the best chance of winning. The average percentage of wins on a $3 \times 3 \times 3$ playboard is 58.14%, on a $4 \times 4 \times 4$ playboard it is 65.40% and on a $3 \times 3 \times 3$ playboard with the middle field excluded it is 39.06%. So on a $4 \times 4 \times 4$ playboard P3 has the best chance of winning if he plays in a team with P1 and the least chance on a $3 \times 3 \times 3$ playboard with the middle field excluded. We just saw that P1 en P2 have both the best chance of winning if they play on a $4 \times 4 \times 4$ playboard, and now we see that also P3 has the best chance of winning on that board. Also they all have the least chance of winning on a $3 \times 3 \times 3$ playboard with the middle field excluded. When we look at Figure 23, we can see that the number of draws on a $3 \times 3 \times 3$ playboard with the middle

field excluded is the highest, so here there will be less players winning. We can also see that the more P3 cooperates with P1, the more wins for P3. This can be explained by the fact that now also P1 wants P3 to win, so he will in most cases not block the winning moves for P3 anymore. The last thing we will derive is that if P3 teams up with P2, he has a really small chance of winning, but if he is not in a team at all the chances are even smaller.

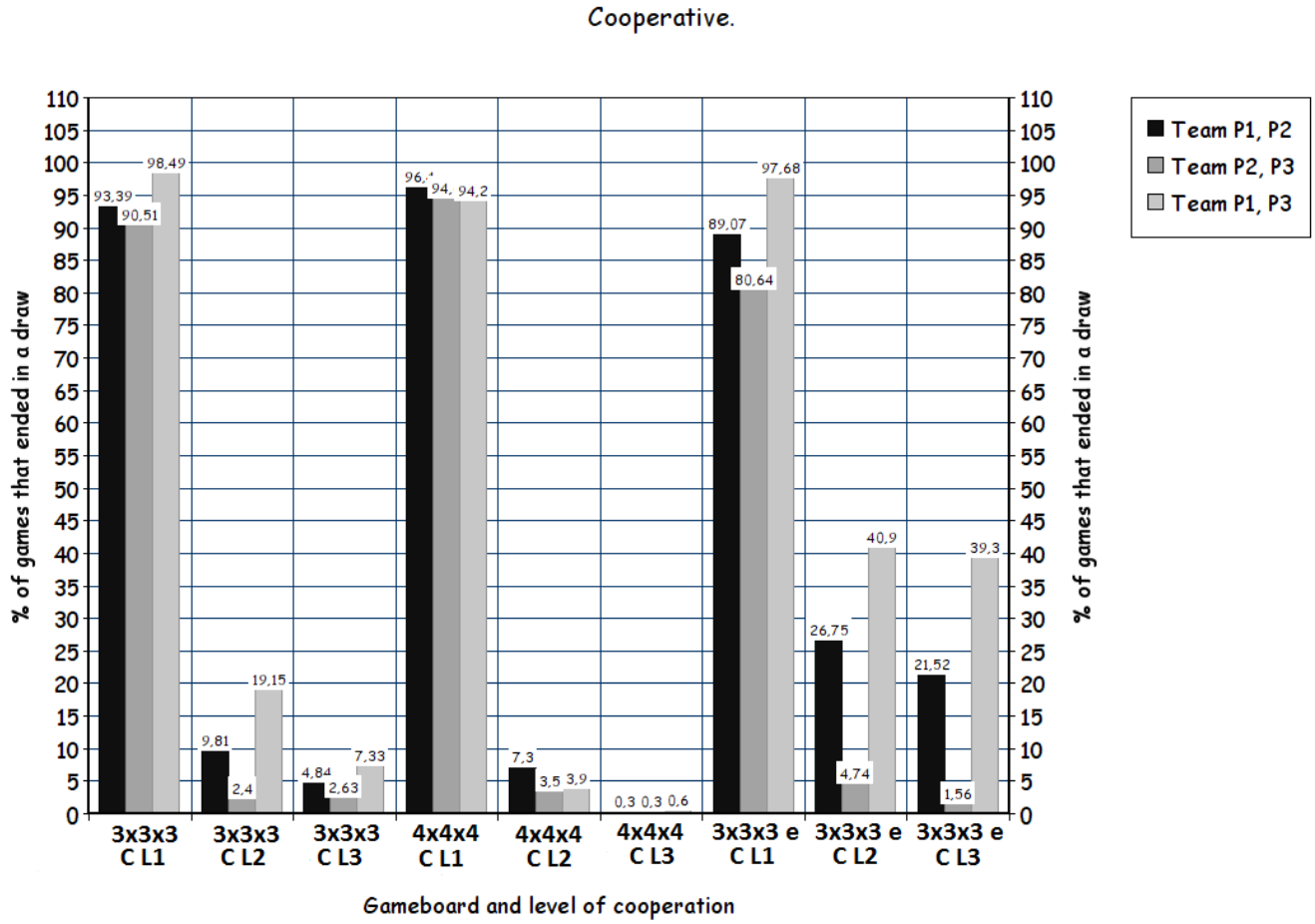

Figure 23: A graph with the results from the experiments with cooperative outcome weights that shows the number of times the game ended in a draw.

The first thing that stands out in the graph displayed in Figure 23 are the larger bars when the cooperation is low. We see that the less the cooperation is, the more the game ends in a draw. This can be explained as following: the more cooperation there is, the more chance that one of the players wins the game and so the less games ending in a draw. Let us now take a look at the different playboards and derive on which playboard the most games end in a draw. We see that for cooperation level 2 or higher on a $3 \times 3 \times 3$ playboard with the middle field excluded the number of draws are approximately the highest and for cooperation level 2 or higher on a $4 \times 4 \times 4$ playboard approximately the lowest. This is in accordance with Figure 20, 21 and 22, we saw that P1, P2 and P3 have the best chance of winning when playing on a $4 \times 4 \times 4$ playboard, so the games that end in a draw should be low. We can also see that the average percentage of draws when P1 en P2 are in a team is 38.82%, the average percentage of draws when P2 en P3 are in a team is 31.21% and the average percentage of draws when

P1 en P3 are in a team is 44.55%. So P1 and P3 as a team yields in the most games ending in a draw, P2 and P3 together in a team yields in the least games ending in a draw.

5.4.4 The best moves

We can derive from the best first moves given in the previous section that the most reached configurations on a $3 \times 3 \times 3$ playboard after the third turn are the three displayed in Figures 24, 25 and 26.

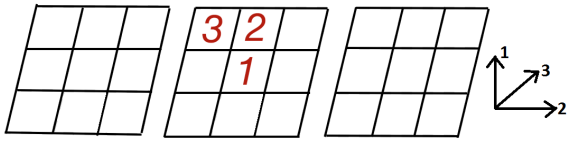

Figure 24: First most reached configuration: 455 out of 1000 times, so 45.5%.

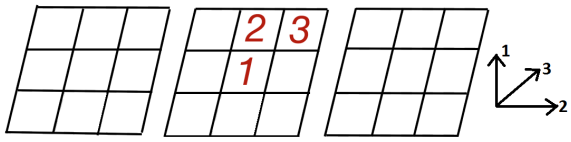

Figure 25: Second most reached configuration: 324 out of 1000 times, so 32.4%.

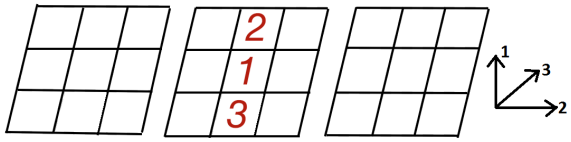

Figure 26: Third most reached configuration: 220 out of 1000 times, so 22.0%.

The percentage of games with this configurations as third configuration can now be calculated. As we could see in the previous section in 1000 out of 1000 cases the first move of P1 was the center move on a $3 \times 3 \times 3$ gameboard. The second move was 514 out of 1000 times one move that blocks P1 from winning and in 486 times out of 1000 it was the other move on the other side that also blocks P1 from winning. These turns are equivalent by symmetry. So 1000 out of 1000 times P2 blocks P1 from winning. So the first and second move played by P1 and P2 are probably almost always the same. The different options of the third move by P3 are given in Figure 24, Figure 25 and Figure 26. Here also the percentage of games that have a third configuration like this is given.

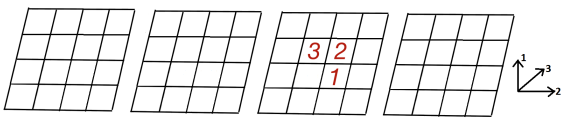

Figure 27: Most reached configuration after third move: 411 out of 1000 times, so 41.1%.

For a $4 \times 4 \times 4$ playboard we can derive from the best first moves given in the previous section that the most reached configuration after the third is the one displayed in Figure 27.

The percentage of games with this configuration as third configuration can now be calculated. As we could see in the previous section in 489 out of 1000 cases the first move of P1 was the move marked with '1' on a $4 \times 4 \times 4$ gameboard. The second move was 895 out of 1000 times one move that blocks P1 from winning. And the third move was 940 times out of 1000 the move marked with '3'. So the percentage of games that have a third configuration like this, as given in Figure 27, is $0.489 \times 0.895 \times 0.940 = 0.4114$ this is 41.1%.

We can derive from the best first moves given in the previous section for a $3 \times 3 \times 3$ gameboard with the middle field excluded that the most reached configurations after the third turn are the ones given in Figure 28, 29, 30 and 31.

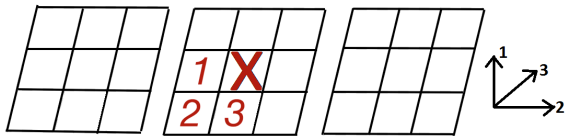

Figure 28: First most reached configuration: 289 out of 1000 times, so 28.9%.

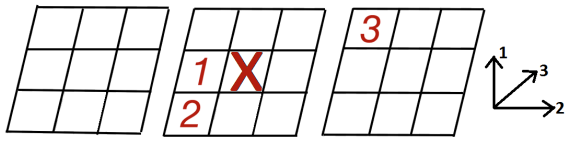

Figure 29: Second most reached configuration: 242 out of 1000 times, so 24.2%.

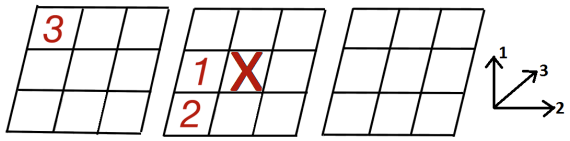

Figure 30: Third most reached configuration: 257 out of 1000 times, so 25.7%.

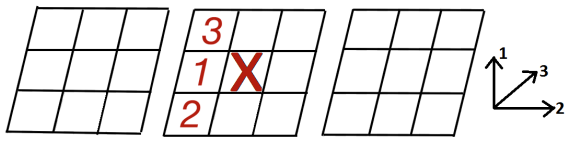

Figure 31: Fourth most reached configuration: 152 out of 1000 times, so 15.2%.

The percentage of games with this configurations as third configuration can now be calculated. As we could see in the previous section in $477 + 470 = 947$ out of 1000 cases the first move of P1 was the move on the left of the center move on a $3 \times 3 \times 3$ gameboard with the middle field excluded. The second move was 1000 out of 1000 times one move that blocks P1 from

winning. So the first and second move played by P1 and P2 are probably almost always the same. The different options of the third move by P3 are given in Figure 28, Figure 29, Figure 30 and Figure 31. Also the percentage of games that have a third configuration like these are given, calculated as follows:

$0.947 \times 1.000 \times 0.305 = 0.2888$ this is 28.9%.
$0.947 \times 1.000 \times 0.256 = 0.2424$ this is 24.2%.
$0.947 \times 1.000 \times 0.271 = 0.2566$ this is 25.7%.
$0.947 \times 1.000 \times 0.160 = 0.1515$ this is 15.2%.

5.4.5 Runtime

What we can derive from the runtimes given in the previous section is that: the more games ending in a draw, the more the runtime will be. This can be explained by the fact that if the game ends in a draw, then all moves are done and so the algorithm of Monte-Carlo Tree Search has to run for 27 moves (on a $3 \times 3 \times 3$ gameboard). When the game does not end in a draw, then Monte-Carlo Tree Search has to run for less than 27 moves (on a $3 \times 3 \times 3$ gameboard). This can be an explanation of the variation in runtime.

6 Conclusion and future work

We have solved the game of $2 \times 2 \times 2$ 3D Hex. The outcome of this game, given optimal play, is a draw. Solving games with larger playfields takes a long time, so the rest of the conclusions are based on experiments with Monte-Carlo Tree Search.

Player	Best cooperation partner	Best gameboard	Worst gameboard
P1	P2	$4 \times 4 \times 4$	$3 \times 3 \times 3$ e
P2	P3	$4 \times 4 \times 4$	$3 \times 3 \times 3$ e
P3	P1	$4 \times 4 \times 4$	$3 \times 3 \times 3$ e

Table 19: The best cooperation partner for the players and also the best gameboard on which this cooperation yields in the best results and the worst gameboard on which this cooperation yields in the worst results.

When all of the players play competitive – selfish we see that nearly all of the games end in a draw. This makes us think that if would solve 3D Hex on a $3 \times 3 \times 3$ gameboard or a $4 \times 4 \times 4$ gameboard the result could possibly be a draw. This is left for future work to examine. We also saw that P1 has some kind of advantage by being able to do the first center field move, because when the game was played on a $3 \times 3 \times 3$ gameboard with the center field excluded, the wins of P1 decreased.

When all of the players play misère we see that now also most of the games end in a draw. We also saw that P1 has some kind of advantage by being able to do the first center field move, because when the game was played on a $3 \times 3 \times 3$ gameboard with the center field excluded,

the wins of P1 decreased a lot. We can see that on a $4 \times 4 \times 4$ gameboard the percentage of games that ended in a draw is 100%, so on larger gameboards it seems to be easier to cause the game to end in a draw.

From the experiments where the players cooperate we can conclude who the player is with whom the players need to cooperate to get the highest percentage of wins. Table 19 shows for each player his best cooperation partner. We can see that the best cooperation partner is the player who's turn is after yours. So when two players play together, the player playing right before the other has the best chance of winning. Also in Table 19 can be found the best gameboard on which this cooperation yields in the best results and the worst gameboard on which this cooperation yields in the worst results.
We can see that P1, P3 and P3 have the best chance of winning if they cooperate with the player playing after them on a $4 \times 4 \times 4$ gameboard and if they cooperate with the player playing after them on a $3 \times 3 \times 3$ gameboard with the middle field excluded they do not get the best out of the cooperation. We can also see in Figure 23 that the number of draws on a $4 \times 4 \times 4$ gameboard is the lowest, this is a consequence of the better profit for P1, P2 and P3.

The reason why if P1, P2 and P3 cooperate with the player playing after them on a $3 \times 3 \times 3$ gameboard with the middle field excluded they do not get the best out of the cooperation, can be that this middle field plays an important role in the game, as stated in the study of the best moves in the previous section.

For a cooperation level of 2 or higher on a $4 \times 4 \times 4$ gameboard the number of draws is approximately the lowest and on a $3 \times 3 \times 3$ gameboard with the middle field excluded the number of draws is approximately the highest. So the winning chances on a larger playboard are bigger. If P1 and P3 play together in a team it yields in the most games ending in a draw, if P2 and P3 play together in a team it yields in the least games ending in a draw.

Furthermore we saw that the more cooperation there is and the higher its level, the less games ending in a draw. This gives the players a better opportunity to win the game. This is because the player playing after you is no longer blocking all of your moves. And the higher the cooperation, the less win chances of yours he will be blocking, so the more chance you get to win the game.

The last thing we can derive from the results of cooperation is that if you do not play in a team, you have the least chance of winning if the other players are in a team. So playing in a team is always better.

With regard to the best moves: the best first move for P1 seems to be the center move. This is why P1 has less wins on a $3 \times 3 \times 3$ gameboard with the middle field excluded in comparison with a normal $3 \times 3 \times 3$ gameboard. So we can say that it might be an idea to introduce the pie rule in this game, so P2 or P3 can take over the move done by P1. The best move for P2 is the move that blocks this first move of P1. So one of the two moves that blocks this first move of P1 is the best move for P2, and these moves are equivalent by symmetry. Then there are three options for the third move played by P3, and these three options are all moves wherein P3 blocks the other players from winning. For a $4 \times 4 \times 4$ gameboard the best first moves for P1 are the moves in the middle of the cube. Then P2 blocks P1 and P3 blocks P2.

For a $3 \times 3 \times 3$ gameboard the best first moves for P1 is the move on the right of the center move. Then P2 blocks P1 and P3 blocks P2, P1 or tries making his own chain.

About the runtime we can say that the more games ending in a draw, the longer the runtime will be. We presume that if $3 \times 3 \times 3$ or $4 \times 4 \times 4$ 3D Hex will be solved then the game would end in a draw. By presuming that, we also know that the runtime of this experiment will take a long time, because a run that ends in a draw takes a long time running. This should be taken into account in future work.

The most interesting thing that we did not manage to do is solving 3D Hex for larger gameboards, this can be done in future work. Also the results of the experiments with Monte-Carlo Tree Search are an estimation and are not statistically significant, so in future work these experiments can be repeated to be statistically significant or they can be repeated with more random playouts per move or more games.

References

[1] Anshelevich, V. V. A hierarchical approach to computer Hex. *Artificial Intelligence*, 134, pages 101–120, 2002.

[2] Anshelevich, V. V. The game of Hex: An automatic theorem proving approach to game programming. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence (AAAI/IAAI)*, pages 189–194, 2000.

[3] Arneson, B., Hayward, R. B., Henderson, P. Solving 8x8 Hex. *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI-09)*, pages 505–510, 2009.

[4] Arneson, B., Hayward, R. B., Henderson, P. Monte-Carlo Tree Search in Hex. *IEEE Transactions on Computational Intelligence and AI in Games*, 2 (4), pages 251–257, 2010.

[5] Bakkes, S., Chaslot, G., Spronck, P., Szita, I. Monte-Carlo Tree Search: A new framework for game AI. *Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference*, pages 216–217, 2008.

[6] Baldick, R., Lee, K-H. Solving three-player games by the matrix approach with application to an electric power market. *IEEE Transactions on Power Systems*, 18 (4), pages 1573–1580, 2003.

[7] Björnsson, Y., Müller, M., Schaeffer, J. New winning and losing positions for 7x7 Hex. *Computers and Games, Third International Conference (CG)*, pages 230–248, 2003.

[8] Browne, C. Hex strategy: Making the right connections. A. K. Peters, Natick, Massachusetts, 2000.

[9] Cazenave, T., Saffidine, A. Utilisation de la recherche arborescente Monte-Carlo au Hex. *Revue d'Intelligence Artificielle*, 23 (2-3), pages 183–202, 2009.

[10] Chaslot, G. M. J-B. Monte-Carlo Tree Search, doctoral dissertation, Universiteit Maastricht, 2010.

[11] Daskalakis, C., Papadimitriou, C. H. Three-player games are hard. *Electronic Colloquium on Computational Complexity 139*, pages 81–87, 2005.

[12] Fossel, J. D. *Monte-Carlo Tree Search applied to the game of Havannah*, BSc thesis, Maastricht University, 2010.

[13] Frühwirth, T., Schrijvers, T. *Optimal union-find in Constraint Handling Rules*, Theory and Practice of Logic Programming, 6, pages 213–22, 2006.

[14] Gale, D. The game of Hex and the Brouwer fixed-point theorem. *The American Mathematical Monthly*, 10, pages 818–827, 1979.

[15] Gardner, M. The game of Hex. *Hexaflexagons and other mathematical diversions — The first scientific American book of puzzles and games*, University of Chicago Press, 1988.

[16] Lagarias, J., Sleator, D. Who wins misère Hex? *The mathemagician and pied puzzler: a collection in tribute to Martin Gardner*, AK Peters, pages 53–84, 1999.

[17] Liao, S., Pawlak, M., Yang, J. On a decomposition method for finding winning strategy in Hex game. *Proceedings ADCOG: International Conference Application and Development of Computer Games*, University of Honkong, pages 96–111, 2001.

[18] Mulvaney, E. *A beginners guide to Hex*, MicJames, 2014.

[19] Nau, D., Wilson, B., Zuckerman, I. Modeling social preferences in multi-player games. *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, pages 337–344, 2011.