Universiteit Leiden
Opleiding Informatica

Tetris Strategies in a Multiplayer Environment

Name:	Chivany van der Werff
Date:	29/09/16
1st supervisor:	W.A. Kosters
2nd supervisor:	H.J. Hoogeboom

BACHELOR THESIS

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

Contents

Abstract

Tetris is a puzzle game which can be played in a single player or multiplayer mode. Most research done regarding Tetris was in a single player environment. Therefore, in this thesis we will try to find out which of three different strategies performs better when competing against one another in different multiplayer Tetris environments. In our research a multiplayer game has two players.

For this research we have developed a Tetris variant with a different and more restrictive set of rules in order to simplify the game and reduce the amount of possible moves during each turn. We will use different strategies, namely: Random, Rule-based, and Monte Carlo. There are three goals that the strategies will try to achieve. The goals are: (1.) The player who has cleared the most lines at the end of a game is the winner; (2.) The player who is the first to clear a line, wins the game; (3.) The player whose Tetromino sticks out above row m in an $m \times n$ sized board loses the game.

Aside from different goals we will also vary the board size. Lastly, we will implement a Hold-function which is assessed by enabling and disabling this function in the different environments and setups. Results have been gathered from each setup which show the significance of a Hold-function, but also that in the environments the Rule-based player was the most successful. Also, whenever $n = 20$ for an $m \times n$ board, all the players performed badly.

Chapter 1

Introduction

The game Tetris [12] was developed in June 1984 by the Artificial Intelligence researcher Alexey Pajitnov. It was firstly introduced as a single player game with a board that has 20 rows and 10 columns (20×10) and pieces named Tetrominoes which appear at the top of the board, one at a time. The player on the board needs to decide how to place this random Tetromino while it falls down until it is unmovable when it lands on a surface. Nowadays, Tetris often has a Hold-function implemented in the game, which allows a player to store exactly one Tetromino in its space. Figure 1.1 shows an official present-day single player Tetris game where the Hold-function can also be seen.
Tetris is a puzzle-like game where the goal is to clear as many lines as possible which will earn the player points. For players to be able to play the game well, strategies are required. Human players need a lot of practice to be able to master certain strategies on a high level but artificial players are able to do this either in a shorter amount of time or do not need to learn or practice anything beforehand. We look at the latter type of artificial players which will use the Alpha-Beta Pruning search algorithm, also referred to as the Rule-based Player. In this paper we investigate three Tetris strategies. The three strategies that will be tested are: (1.) Random; (2.) Rule-based; (3.) Monte Carlo. We created setups where the strategies play against one another and based on the results we draw a conclusion for the performances of the players.
The reason for picking a multiplayer environment is because most research on Tetris was done in a single player environment which excludes a lot of influences that are present in multiplayer Tetris games. Research in this environment would thus give different results than in a single player environment. To measure the performance, the strategies will play in environments with different board sizes and goals. We will also implement a Hold-function to see if it improves the performance of the strategies in the different environments.
For our research we will be using a simplified version of Tetris because of the fact that the number of possible states in the original game with a board size of 20×10 is around 10^{60} [3]. If played according to the rules of the official game, every newly spawned Tetromino can have a large number of possible placements depending on the present blockstructure in the field. To decrease the nodes in the state space along with the computational time, and thus reducing the complexity, we simplified the game.

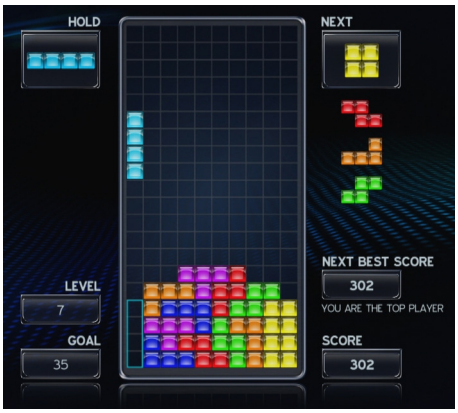

Figure 1.1: Playstation Network (PS3) Tetris® showing a present-day single player game [17].

We will give a short description of the remaining chapters of this thesis. In Chapter 2 we will be thoroughly explaining our variant of the game. Chapter 3 discusses related work to this research; Chapter 4 discusses our approach for this research; Chapter 5 discusses the experiments and the results; and lastly in Chapter 6 we draw conclusions and give suggestions for future work.

This thesis was written as a Bachelor project at the Leiden Institute of Advanced Computer Science (LIACS) of Leiden University, and has been been supervised by Walter A. Kosters and Hendrik Jan Hoogeboom.

Chapter 2

The Game

As was mentioned before, we will be working with a simplified version of Tetris to reduce the complexity of the game which will make for a faster computation. In this section we will explain the proposed alterations thoroughly.

2.1 The Players

There are two players in each game. The players will play on the same field and switch turns after they have placed a Tetromino in the field. A score is kept for each player. The score indicates the amount of cleared lines per player.

2.2 The Board

The testing will start out on an $m \times n$ board and will be altered to a smaller or bigger board depending on the strength of the opponent and the results. Tetrominoes are out of bounds if their blocks exceed beyond row number m of the board. A sequence of random Tetrominoes falls down on the playing field. In contrast to the original Tetris game, the Tetromino next in line to appear can only be rotated and moved (horizontally) at the top of the board when and where it first appears. After the player has made his choice in rotation and position, the Tetromino will fall down in a straight line. During that process the player is unable to move or rotate the piece.

2.3 Pieces

Tetris has a set of seven different pieces called Tetrominoes. In the original Tetris game they are referred to as "Tetriminos" but in this paper we will use the word "Tetrominoes" since it is the definition of geometric shapes that consist of four square-shaped blocks. The different pieces and their rotations are named and displayed on the next page:

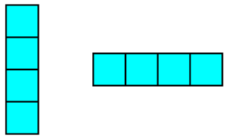

Figure 2.1: The I-Tetromino and its two possible rotations.

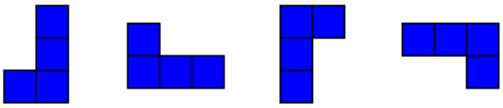

Figure 2.2: The J-Tetromino and its four possible rotations.

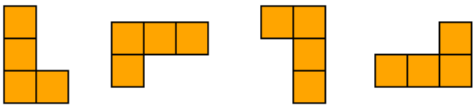

Figure 2.3: The L-Tetromino and its four possible rotations.

Figure 2.4: The O-Tetromino and its only possible state.

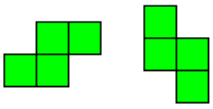

Figure 2.5: The S-Tetromino and its two possible rotations.

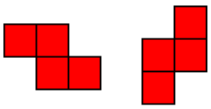

Figure 2.6: The Z-Tetromino and its two possible rotations.

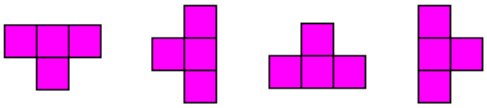

Figure 2.7: The T-Tetromino and its four possible rotations.

The game starts with one of the seven pieces being randomly introduced onto the top-most row (row m) of the field and depending on the piece and the rotation its blocks will at least touch the mth row and at most the $(m-3)$th row. After the current piece has been rotated and moved by the player, it will be dropped in a straight line on the field. After the landing, the next random Tetromino will be introduced at the top of the field. The pieces always arrive at a fixed position on the field.

2.4 Hold-Function

This function is a space which can hold a Tetromino. This gives a player when it is his turn the ability to store a recently spawned piece into this space. If the current Tetromino that has spawned makes up for a worse move than the Tetromino that is already stored in the Hold space, they are swapped. This means that the

Tetromino in the Hold space is taken out and the Tetromino that has recently spawned at the top of the field is stored within that space. The previous Hold Tetromino is then used within the field.
In the first original Tetris game this function had not been implemented yet. This feature was implemented later on and is very common in contemporary Tetris. The Hold-function is thus not always enabled.

2.5 Rule Set

Our variant of Tetris used for this research has a specific rule set. We have divided the rules into basic rules and specially induced rules, which we have developed and implemented to improve the durability of a game.

Basic Rules:

- The Tetrominoes are not allowed to go out of bounds which means the blocks are not allowed to exceed the top row for the continuation of the game.
- When blocks of Tetrominoes exceed the top row, the game will end.
- Tetrominoes are only able to be moved and rotated by the player at the top-most side of the board when and where the pieces will arrive. After a decision has been made, the piece will fall straight down. During the fall the piece is unable to be moved or rotated.
- A line is formed if an entire row is filled with blocks. This line will then be cleared which means that the blocks in that entire row will be removed.
- When a row of blocks has been removed, the blocks on top of that row according to gravity will fall down into the newly created free space shifting all the blocks down a line (or more if more lines were cleared).
- Blocks colliding vertically will stop further gravitational movement of a Tetromino.
- It is not allowed for Tetrominoes to be stacked on top of one another to end the game.

Specially Induced Rules:

- When it is allowed to use the Hold-function it should be used in a way the player deems fit.
- A player always tries to pick a move that exceeds the current top row (the highest row that has one or more blocks in it) of the block structure as little as possible. This means that Tetrominoes placed beneath this top row are the most preferred. If that deems to be impossible, a move that exceeds the top row as little as possible will be preferred instead. This rule is implemented to have a more even block structure across the field.
- A player is not allowed to stack its Tetromino on the Tetromino of the other player in the previous turn. This is to prevent a player from prematurely ending the game when for example being ahead.

2.6 Goals

The players will play to achieve certain goals during a game. We have set up three goals to test the performance of the players. The goals are as following:

1. Most lines: The player who has cleared the most lines at the end of a game is the winner of said game.
2. First line clear: The player who is the first to clear a line wins the game. This means that only one line needs to be cleared by one of the two players in order to win and end the game. The game can still end if the players manage to get blocks out of bounds. If none of the players have cleared a line when this happens, the game results into a tie which means that none of the players result into being the winner.
3. First to play out of bounds: The player whose Tetromino is the cause for the game to end, loses said game. That means that the other player is automatically the winner.

We will refer to these goals using the number above that is used to indicate them. They will thus be referred to as goal 1, goal 2 and goal 3.

Chapter 3

Related Work

In this chapter, we will mention scientific work related to Tetris and our research.

Similar to our research is the research of Rovatsou whose research was about an agent completing as many lines as possible with an adversary that changes the Tetrominos in such a way that the game is harder for Player 1. She also generated a search tree with the MiniMax search algorithm and the Alpha-Beta Pruning optimization but also included a learning component to improve the strategy of Player 1 over time [6].

Boumaza discusses the amount of possible states in the state space of a 20×10 board size which is the board size in the original Tetris game. This resulted that there are approximately 10^{60} states [3, 4].

Burgiel discusses the NP-completeness of the game. Bouzama discusses that with approximations to reduce the size of the state space, the problem of finding strategies to maximize the score is NP-complete. She also discusses that for a version of the game where the sequence of the pieces is not known in advance, is of at least of the same difficulty as the version where the sequence of the pieces is known. Both versions result into being NP-complete [4, 5, 7, 8, 14, 15].

Breukelaar, Hoogeboom, Kosters, Demaine, Hohenberger, and Liben-Nowell prove that for a natural full-information (offline) version of Tetris the game is NP-complete when the number of cleared rows, the number of Tetrises, the maximum height of an occupied square, or the number of pieces placed before the game ends is optimized [7, 8].

Hoogeboom and Kosters prove that every (reasonable) configuration in Tetris can be constructed in an initially empty playing field [14].

Hoogeboom and Kosters also prove that in a Tetris variant similar to the one we used for our research Tetris without intervention is undecidable, while Tetris with normal intervention is decidable [15].

Böhm, Kókai, and Mandl have used a evaluation function which they called a rating function to determine the best Tetris move per turn in a 2-ply search [9]. The same goes for Shahar and West as well as Flom and Robinson who also used an evaluation function which is similar to the one Böhm, Kókai, and Mandl used [9–11].

Browne, Lucas, Cowling, Rohlfshagen, Tavener, Perez, Samothrakis and Colton have written about Monte Carlo methods and approach as well as Monte Carlo Tree Search in numerous variations [13].

Chapter 4

Approach

In this chapter we will explain what methods we have used for our research.

4.1 Search Algorithms

To test the performance of the strategies we have decided to develop a program for a variant of Tetris in C++, where we will use the Alpha-Beta Pruning algorithm with evaluation functions for the Rule-based player. We are using the Alpha-Beta Pruning algorithm because the MiniMax Search algorithm works for a full information two-player game where players take turns.
Search algorithms [2] are used to find a set of states in a state space from the initial state to a goal state with a minimum amount of total path cost. Every reachable state of the game is included in the state space tree which is generated by the used search algorithm. The search tree starts with a root node, also known as the initial state. In this state the empty playing field gets its first random Tetromino. The successor states of the initial state are generated and those states are expanded until all paths lead to a goal state. The actions that result into new states in the tree are the possible choices of a player within his/her turn. The goal states, also called the terminal nodes or end states, are the states where the game has come to an end. This depends on what states cause a goal state to occur, like for example a Tetromino surpassing row m on an $m \times n$ board, or as mentioned before when a strategy has cleared the first line in the game. Getting an optimal path out of the search tree is equivalent to finding an optimal strategy, which is based on the total path costs and the evaluation of those costs. We will determine which successor state is the best based on their worth which will be calculated by an evaluation (decision) function.

The State Space Tree for our game is still enormous even with our proposed adaptations. Therefore, finding an optimal path for our strategies with the available computational resources and time will force us to not exceed a ply higher than 2 when it comes to iterative deepening. In the MiniMax search algorithm we will be using our two players as the Max-player and Min-player. They will switch turns after they have made a move with the current random Tetromino in the field. Player 1 will compete against Player 2, the adversary, where Player 1 will try to maximize the total amount of points (score) given by the evaluation function, whereas when the opponent is the Min-player, Player 2 will be trying to minimize that score. Each state in the tree holds information about the state of the board as well as which player's turn it is. The search tree is organized in levels that are based on whose turn it is. These different levels are called plies. Fringe nodes are nodes that are yet to be expanded. To reduce the amount of fringe nodes that are to be expanded in the search tree we will use the Alpha-Beta Pruning algorithm.

4.1.1 MiniMax Search Algorithm

The MiniMax algorithm functions on alternating turns between players. Each player will try his/her best to achieve the best heuristic value possible. Therefore, it is hard for a player to get to a desired goal state since a good opponent will try his/her best to get into a goal state that suits him/her the best. This implies that the optimal goal states and thus optimal paths for both players are never the same. Both players will thus

choose action states that will improve their likeability of winning and decrease their likelihood of losing. This results in the likelihood that both players will never get the highest achievable score possible. Player 1 does not know the strategy of Player 2 and thus will always assume that the adversary plays in an optimal way. The MiniMax search algorithm generates the state space tree and has a score evaluated at each state. When it is the Max player's (Player 1) turn it will choose the successor node that gives the highest score. The Min player will choose the successor node with the lowest score. The time complexity is equal to $O(p^n)$ and the space complexity is $O(pn)$ where p is the number of allowed moves of the current player and n is the number of the last ply to which the expansion has taken place.
The MiniMax algorithm computes the evaluation of each node. When the Max player for example is the one that gets the first turn, the search tree is expanded until all the goal states are expanded. When a goal state is reached the expansion is terminated and the score of the leaf node is returned.
A variation of the MiniMax algorithm, called the Expecti-MiniMax algorithm, applies to our Tetris variant.

Expecti-MiniMax Algorithm

The Expecti-MiniMax algorithm is similar to the regular MiniMax algorithm except for the fact that it uses chance nodes, which contain a value based on the occurrence of a random event. A generated tree based on the MiniMax algorithm is an Expecti-MiniMax tree when chance nodes are included. The value of the chance nodes is calculated by having the sum of the chance for a random state to occur from that node multiplied with the child node it applies to. Because the spawn of a new Tetromino is always random it means that it will need a chance node to indicate the random occurrence.

4.1.2 Alpha-Beta Pruning Search Algorithm

The Alpha-Beta Pruning Search Algorithm is an optimization of the MiniMax Search Algorithm which returns the same path as the MiniMax algorithm but prunes nodes that are not needed for expansion. In the state space tree, generated by the MiniMax Search Algorithm, the Alpha-Beta algorithm does not expand the nodes that have better nodes as options at the parent node or any other predecessor. This is called pruning which is done to minimalize the state space tree by reducing the amount of possibilities. It will also speed up the execution process by not unnecessarily expanding nodes.
The algorithm gets its name from the Alpha (α) and Beta (β) bounds it uses. Certain information is passed through to successor nodes. They need to have access to what the values for the best choices are at the parent node and the predecessors. The Alpha and Beta values are passed to the successor nodes, where Alpha contains the value of the best option for the Max Player on a path and thus creates a maximum lower bound of the set of all possible paths whereas Beta contains the value of the best option for the Min Player on a path and thus creating a minimum upper bound of the set of all possible paths. The value of the current node is then compared with the bounds where $\alpha \leq x \leq \beta$ with x being the estimated value of the current node. In every Max node Alpha gets updated before it is given to lower levels. The same goes for Beta in Min nodes where the value of the current Min node will update the Beta value and is then given to lower levels. There has to be an existing overlap between the ranges of Alpha and Beta. When a node is evaluated and no overlap is found between the Alpha and Beta ranges, it will come to light that the current node can not result into a solution path. With that information we can stop expanding the current node and jump back to its parent node. The value from the node will be passed to the parent node. This is done by passing the Beta value of a Min node to the parent which might change the Alpha value of said parent. As for Alpha values in Max nodes, those might change the Beta value of their parents. Optimal pruning is only achieved when nodes are ordered in an optimal way. The order of nodes is influential to the amount of pruning that can take place.
The Alpha-Beta Pruning Search algorithm is able to reduce the time complexity of the MiniMax Search algorithm to $O(p^{\frac{n}{2}})$ for the best case which is when the nodes are sorted in an optimal fashion. When there is no particular ordering the average time complexity is going to be $O(p^{\frac{3n}{4}})$. The worst case in Alpha-Beta pruning implies that no possibility for pruning arose at all and thus the time complexity will be equal to that of the MiniMax Search algorithm.

4.1.3 Evaluation Functions

The above named search algorithms need an evaluation function to generate a heuristic value in each node. In an evaluation function one can implement how the program should assess the value of a node and thus

how well the move is. One can give a positive amount of points for situations that are really beneficial for the current player or a negative amount of points for situations that should be avoided as much as possible by the current player. The situation derrived at the current node will result in positive and negative points being handed out. The sum of the given points for a certain node is the value of that node. The Min-player will be going for the lowest values generated by the evaluation function and the Max-player will try to achieve the highest value possible generated by the evaluation function.

4.2 Basic Monte Carlo Tree Search

The Monte Carlo algorithm we use, which is a Basic Monte Carlo Tree Search algorithm [13], also performs a search for the best heuristic value to determine what move in the current node will possibly give the best results. The simple concept of the Basic Monte Carlo Tree Search is displayed in Figure 4.1. For this the algorithm does several (often starting at 1000) random playouts of the game from the current node it is in. A node represents a (possible) situation of the game. A successor node is generated from the current node and represents one of the possible situations the parent node could result in. Successor nodes are generated for each node until a leaf node is reached. A leaf node in a game is the end situation of a game. A playout is the path from the current node to a leaf node. A playout in a Monte Carlo algorithm will return a heuristic value. After several playouts have been done, the move that led to the best average playout will be picked and done. See Section**??** for more information about getting the best average playout. The performance of the Monte Carlo algorithm improves when the amount of random playouts that have to be done for the current node increases.

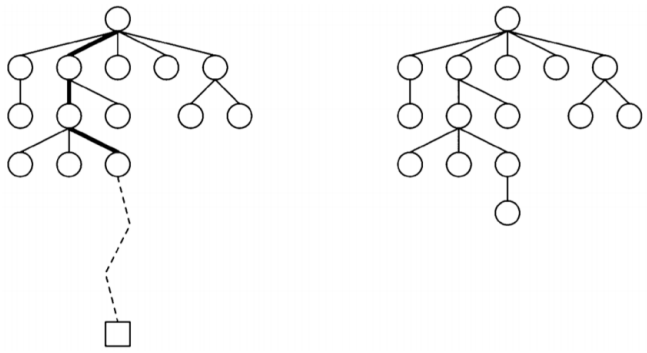

Figure 4.1: This image shows the concept of a Basic Monte Carlo Tree Search algorithm [16]

.

4.3 Strategies

We have now given a general idea of the algorithms we have used for our research. We will now explain how we have implemented these algorithms in the beforenamed Tetris environment.

4.3.1 Random Player

The Random strategy does a random move that abides to the aforementioned Rule Set. When the Hold-function is enabled the Tetromino that will be used will be chosen at random.

4.3.2 Rule-based Player: Alpha-Beta Pruning Search Algorithm

The Alpha-Beta Pruning search algorithm has every move evaluated by the means of an evaluation function that gives back the heuristic value for every move. The performance of the search algorithm thus depends on what is implemented in the evaluation function. We will therefore discuss the evaluation function thoroughly. The algorithm will have a maximum depth of 2 because the Expecti-Minimax algorithm requires a very large expansion of the state space when the depth increases.

Evaluation Function

There is an evaluation function for each goal we are experimenting with. This results into three slightly different evaluation functions. We will discuss what aspects in a playing field are taken into consideration by the evaluation function. In the formulas listed below we have a function F which indicates whether a position is occupied. Function F is defined as follows:

$$F(i)(j) = 1 \quad \textit{if and only if the position } (i,j) \textit{ on the board contains a block.}$$

We will now list the different aspects of the evaluation function:

1. **More blocks in lower rows** The aim is to credit rows for the amount of blocks they contain but lower rows get a higher value for the amount of residing blocks. The lower the row, the higher the value per block. The number of positive points assigned for this aspect is:

$$score \leftarrow score \ + \sum_{i=0}^{rows-1} \Big(rows - i \Big) \sum_{j=0}^{columns-1} F(i)(j)$$

2. **Full row** When a move results in a row having n blocks, and thus having the entire row filled with blocks, it will result in a much higher heuristic value. More rows cleared in one turn will slightly increase the heuristic value per extra cleared line. The number of positive points assigned for this aspect is shown below. We initialize *FullRow* to a value of 0:
   if:

$$\Big(\sum_{j=0}^{columns-1} F(row)(j) \Big) = columns$$

   then:

$$FullRow \leftarrow FullRow \ + \ 1$$

$$score \leftarrow score \ + \ (columns * 200) * FullRow$$

3. **Blocks evenly spread across the field** When a row does not result in a line clearance, it will get points for every block that is occupying space in that row. There is an exception when there are $n-1$ blocks in that row which gives the opponent in the next turn a higher chance of clearing a line. If that is the case, the heuristic value will strongly decrease. This exception does not apply to goal 3 since line clearing overall will delay the game from ending.
   if:

$$\Big(\sum_{j=0}^{columns-1} F(row)(j) \Big) < columns$$

   then:

$$score \leftarrow score \ + \sum_{i=0}^{rows-1} 25 \Big(\sum_{j=0}^{columns-1} F(i)(j) \Big)$$

   if:

$$\Big(\sum_{j=0}^{columns-1} F(row)(j) \Big) = columns - 1$$

   then:

$$score \leftarrow (score/6)$$

   if:

$$\Big(\sum_{j=0}^{columns-1} F(row)(j) \Big) = columns - 2$$

then:

$$score \leftarrow (score/3)$$

4. **Top row** The top row is the highest row with a block or blocks present. The top row can range from 0 to m. The higher the top row, the lower the heuristic value because a high top row will result in the game to end soon after:
   if:

   $$TopRow < rows$$

   then:

   $$score \leftarrow score \ / \ TopRow$$

5. **Top row is higher than** m When a move results in the top row being higher than m, which means that the game will end, it will strongly reduce the heuristic value of the move. There is a slight exception: if a move results in the game ending no matter where and how it is placed but can still clear a line, the heuristic value will be slightly higher for the move that results into a full row for a line clear. We define:
   if:

   $$FullRows > 0 \ \textbf{\textit{and}} \ TopRow > rows$$

   then:

   $$score \leftarrow (score/200) \ + \ (FullRows * 100)$$

   if:

   $$FullRows = 0 \ \textbf{\textit{and}} \ TopRow > rows$$

   then:

   $$score \leftarrow (score/200)$$

When the Hold-function is enabled and the Tetromino residing in it results in a move with a higher heuristic value than the Tetromino that spawned in the current player's turn, the Tetromino in the Hold-function will be used and the recently spawned Tetromino will reside in the Hold space.

4.3.3 Monte Carlo Player

Our Monte Carlo strategy determines the heuristic value of a playout on the amount of Tetrominoes that is used throughout an entire game. The move with the highest heuristic value of all generated playouts will result in the best move for this strategy during his/her current turn. When multiple moves result into having the highest heuristic value, a move out of this set of moves is randomly chosen. When the Hold-function is enabled and the Tetromino residing in it results in a move where a playout had a higher heuristic value than the best heuristic value of the playouts of the recently spawned Tetromino in the player's turn, the Tetromino in the Hold-function will be used, and the recently spawned Tetromino will reside in the Hold space.

Chapter 5

Experiments

To measure the performance of the strategies we have tested them in different environments. The strategies will try to achieve the three different goals mentioned before. We will also check if enabling the Hold-function will make any significant difference. Lastly, we will also test in different board sizes. We have set up each strategy against one another. The setups are described below.

The strategies that will be examined in the beforenamed Tetris environments will be the Random Algorithm, Alpha-Beta Pruning algorithm, and the Monte Carlo Algorithm. We examine how well these algorithms perform against one another. We will do so by letting the algorithms compete in setups as given in Table 5.2. When it is not mentioned otherwise we assume that the Alpha-beta Pruning strategy has a depth of 2. The same goes for the Monte Carlo strategy where the default is generating 100 playouts.

Each strategy will compete against either itself or one of the other strategies. The setups are mentioned in Table 5.1.

Table 5.1: Setups

Setup	**Player** 1	**Player** 2
1.	Random	Random
2.	Alpha-beta Pruning	Random
3.	Alpha-beta Pruning (Depth $= 2$)	Alpha-beta Pruning (Depth $= 2$)
4.	Alpha-beta Pruning (Depth $= 3$)	Alpha-beta Pruning (Depth $= 3$)
5.	Alpha-beta Pruning	Monte Carlo
6.	Monte Carlo	Random
7.	Monte Carlo(100)	Monte Carlo(1000)

Table 5.2 shows the Tetris environments in which the setups will be examined in.

Table 5.2: Environments

Abbreviation	Board size	Goal	Hold
10x10-1	10 rows & 10 columns	Most lines	Disabled
10x10-1H	10 rows & 10 columns	Most lines	Enabled
10x10-2	10 rows & 10 columns	First line clear	Disabled
10x10-2H	10 rows & 10 columns	First line clear	Enabled
10x10-3	10 rows & 10 columns	First out of bounds	Disabled
10x10-3H	10 rows & 10 columns	First out of bounds	Enabled
10x20-1	10 rows & 20 columns	Most lines	Disabled
10x20-1H	10 rows & 20 columns	Most lines	Enabled
10x20-2	10 rows & 20 columns	First line clear	Disabled
10x20-2H	10 rows & 20 columns	First line clear	Enabled
10x20-3	10 rows & 20 columns	First out of bounds	Disabled
10x20-3H	10 rows & 20 columns	First out of bounds	Enabled
20x10-1	20 rows & 10 columns	Most lines	Disabled
20x10-1H	20 rows & 10 columns	Most lines	Enabled
20x10-2	20 rows & 10 columns	First line clear	Disabled
20x10-2H	20 rows & 10 columns	First line clear	Enabled
20x10-3	20 rows & 10 columns	First out of bounds	Disabled
20x10-3H	20 rows & 10 columns	First out of bounds	Enabled
20x20-1	20 rows & 20 columns	Most lines	Disabled
20x20-1H	20 rows & 20 columns	Most lines	Enabled
20x20-2	20 rows & 20 columns	First line clear	Disabled
20x20-2H	20 rows & 20 columns	First line clear	Enabled
20x20-3	20 rows & 20 columns	First out of bounds	Disabled
20x20-3H	20 rows & 20 columns	First out of bounds	Enabled

5.1 Results

In this section we will present and discuss the results from the beforementioned experiments. We have generated 50 games per setup per board size, goal, and the usage of the Hold-function. The output of each game shows the winner of the game by either giving the number 1, 2 or 0, where 1 means Player 1 had won the game, the same goes for when the output was 2 which means that Player 2 had won; 0 is an exceptional case which can only occur when the goal was either **[Most lines]** or **[First line clear]**. This means that the game resulted in a tie and thus had no winner. Table 5.2 shows the meaning of the abbreviations mentioned on the horizontal axis of a chart. We refer to the goals in the following way: **[Most lines]** $= 1$, **[First line clear]** $= 2$, and **[First to play out of bounds]** $= 3$.

First we will discuss the setups and experiments involving the Alpha-beta Pruning algorithm (Rule-based Player) to get an idea of how well the algorithm and its evaluation function perform. See Figures 5.1, 5.2, 5.3, and 5.4.

In the setups of the charts, Player 1 is always the Rule-based Player using the Alpha-beta Pruning algorithm with the previously discussed evaluation function. Player 2 in Figure 5.2 and 5.3 uses the same algorithm but with a simple evaluation function for performance testing of Player 1. Player 2 in Figure 5.4 is using the Monte Carlo algorithm and in Figure 5.1 it is using the Random algorithm.

We will now discuss the results per board size.

- **Goal 1:** The results show that Player 1 has won more games than Player 2 on almost every board size except when Alpha-beta vs. Alpha-beta had depth $= 3$. It was then Player 2 who had won the most games. There are a few exceptions for the wins of Player 1. For the results of the Alpha-beta vs. Alpha-beta setup where the depth $= 3$, Player 1 won more games except for on the 20×20 board. There is also a noticeable improvement for Player 2 in Figure 5.4, where Player 2 also shows that the performance and thus the wins strongly increase when the Hold-function is enabled. Furthermore, there is a noticeable improvement for the players in every setup when the Hold-function is enabled, and especially for Player 1.

- **Goal 2:** The results show that Player 1 has won more games than Player 2 on almost every board size except when Alpha-beta vs. Alpha-beta had depth $= 3$. It was then Player 2 who had won the most games. There are a few exceptions for the wins of Player 1. For the results of the Alpha-beta vs. Alpha-beta setup where the depth $= 2$, Player 1 won more games except for on the 20×20 board. Player 2 makes a lot of improvement in his/her performance on board sizes where $n = 10$. There is an exception when Alpha-beta vs. Alpha-beta had depth $= 3$, where on a 20×10 board, Player 2 did not improve in his/her amount of wins and actually had a lower amount of wins than in the setup with the Hold-function disabled. There is also a noticeable improvement for Player 2 in Figure 5.4, where Player 2 also shows that the performance and thus the wins strongly increase when the Hold-function is enabled.

- **Goal 3:** The results show that Player 1 has won more games than Player 2 on every almost board size. The charts also show an increased performance when the Hold-function is enabled, especially for Player 1. An exception is noticeable on a board size of 20×20 in the Alpha-beta vs. Alpha-beta setup where the depth $= 3$, where the performance has slightly decreased for Player 1. It improved slightly for Player 2.

  A significantly high amount of games has resulted in a tie between the players, namely when $n = 20$ for an $m \times n$ board.

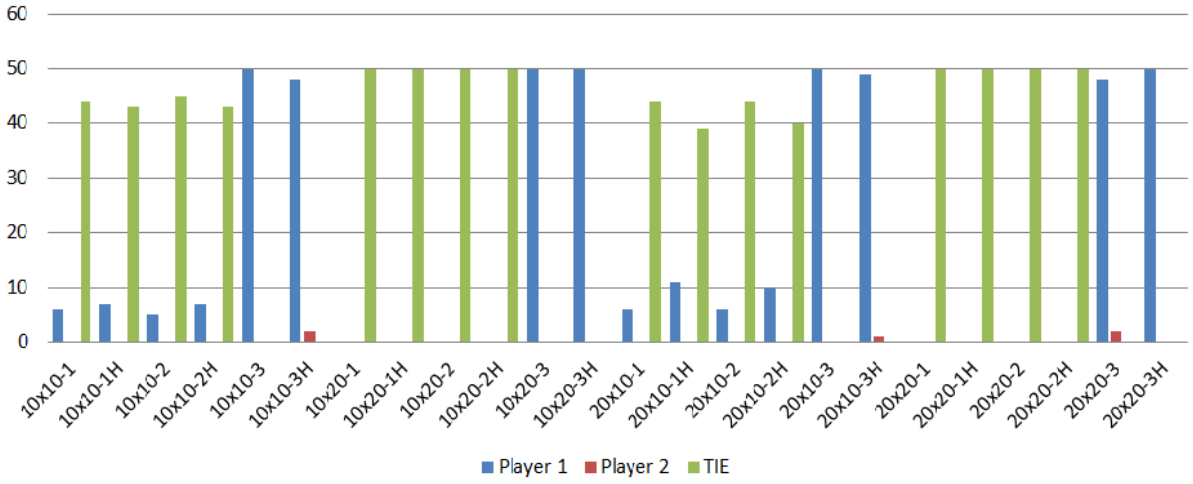

Figure 5.1: This chart shows the results of the different setups for AlphaBeta(max-player) against Random

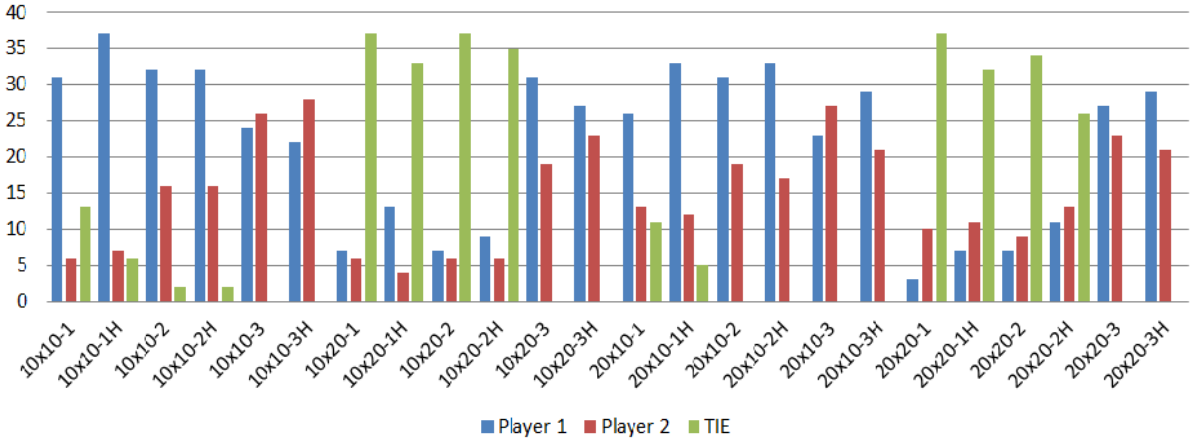

Figure 5.2: This chart shows the results of the different setups for AlphaBeta(max-player) against AlphaBeta(min-player) with the maximum depth set to 2.

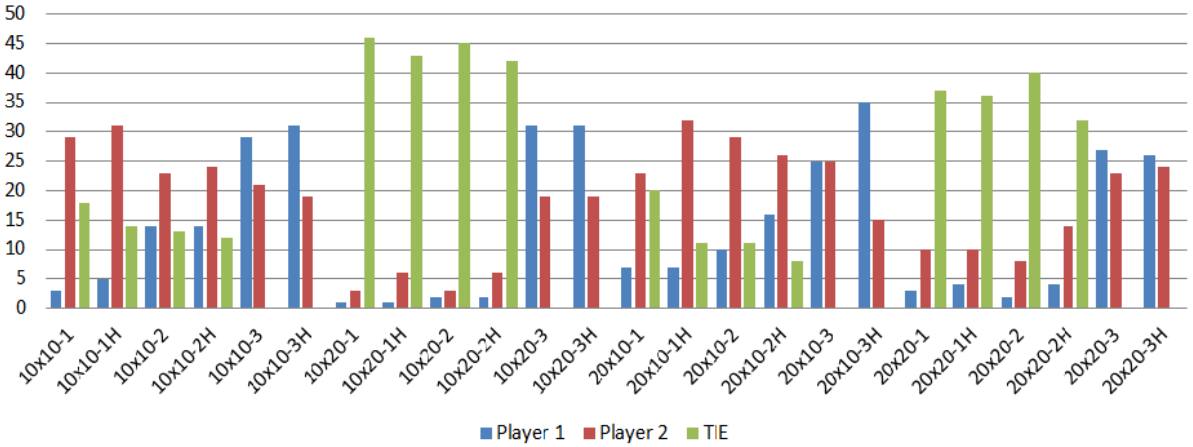

Figure 5.3: This chart shows the results of the different setups for AlphaBeta(max-player) against AlphaBeta(min-player) with the maximum depth set to 3.

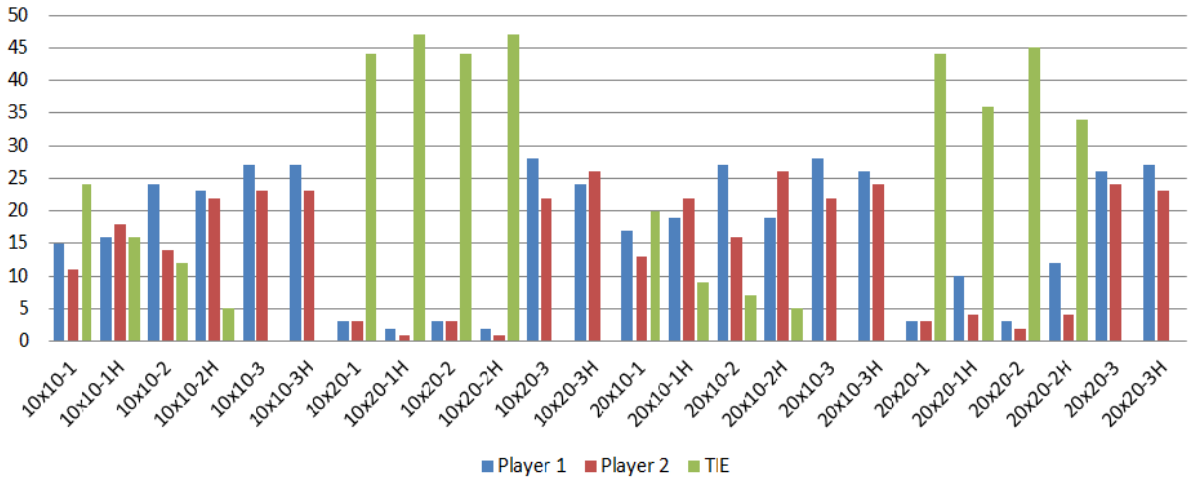

Figure 5.4: This chart shows the results of the different setups for AlphaBeta(max-player) against Monte Carlo

We will now discuss the setups and experiments involving the Monte Carlo algorithm and lastly the Random algorithm competing against itself. To get an idea of how well the algorithms perform, see Figures 5.5, 5.6, and 5.7.

In the setups of the charts, Player 1 uses the Monte Carlo algorithm except for in the last chart which then uses the Random algorithm. Player 2 in Figure 5.7 also uses a the Monte Carlo algorithm but instead of generating 100 playouts per turn it generates 1000 playouts. Player 2 uses the Random algorithm in both Figure 5.6 and 5.5. The chart for the setup of the Random algorithm against itself is shown to indicate that the algorithm is a good default to examine the performance of the other strategies with.

We will now discuss the results per board size.

- **Goal 1:** The results show that Player 2 has won more games than Player 1 on almost every board size for this goal in Figure 5.7. The opposite is true in Figure 5.6. As for Figure 5.5, the players play random and thus their statistics are almost equal. When the Hold-function is enabled in Figure 5.7 we can see that the performance of Player 1 almost always increases, whereas in comparison Player 2 his/her performance either only increases slightly, not at all, or actually decreases. In Figure 5.6 the overall performance of Player 1 increases. When the board size is 10×10, we notice that the number of wins have decreased but the overall performance of Player 1 has increased, because when the Hold-function is enabled, Player 2 stops winning games. In Figure 5.5 every game resulted in a tie, which is explained by the fact that the Random algorithm often causes a game to end before lines have been cleared.

- **Goal 2:** The results show that Player 2 has won more games than Player 1 on almost every board size for this goal in Figure 5.7. The opposite is true in Figure 5.6. As for Figure 5.5 the Players play at random and thus their statistics are almost equal. Player 1 seems to be winning more games in total for this goal than for goal 1 in Figure 5.7. When the Hold-function is enabled in Figure 5.7, we can see that the performance of Player 1 almost always increases, whereas in comparison Player 2 his/her performance either only increases slightly, not at all, or actually decreases. In Figure 5.6 the overall performance of Player 1 increases. When the board size is 10×10, we notice that the number of wins has decreased but the overall performance of Player 1 has increased because when the Hold-function is enabled, Player 2 stops winning games. In Figure 5.5 every game resulted in a tie, which is explained by the fact that the Random algorithm often causes a game to end before lines have been cleared.

- **Goal 3:** For the Monte Carlo setups, Player 1 has won more games in total in comparison to Player 2. The charts also show an increased performance when the Hold-function is enabled, especially for Player 1. An exception is noticeable in Figure 5.7 on a board size of 10×20, where the performance of Player 1 decreases when the Hold-function is enabled. An exception also occurs in Figure 5.5, but we can assume that this result is due to the fact that the performance of Player 2 is completely random, and thus a slight decrease in performance can be coincidental. In Figure 5.6 there is no clear indication that the Hold-function improves the performance of the players. In Figure 5.5, the performance of the

players is random and thus the player who exceeds the other in wins on different board sizes is also random.

A significantly high amount of games has resulted in a tie between the players when $n = 20$ in an $m \times n$ board.

We have thoroughly discussed the outcome of our research. In the next chapter we will draw conclusions based on these results.

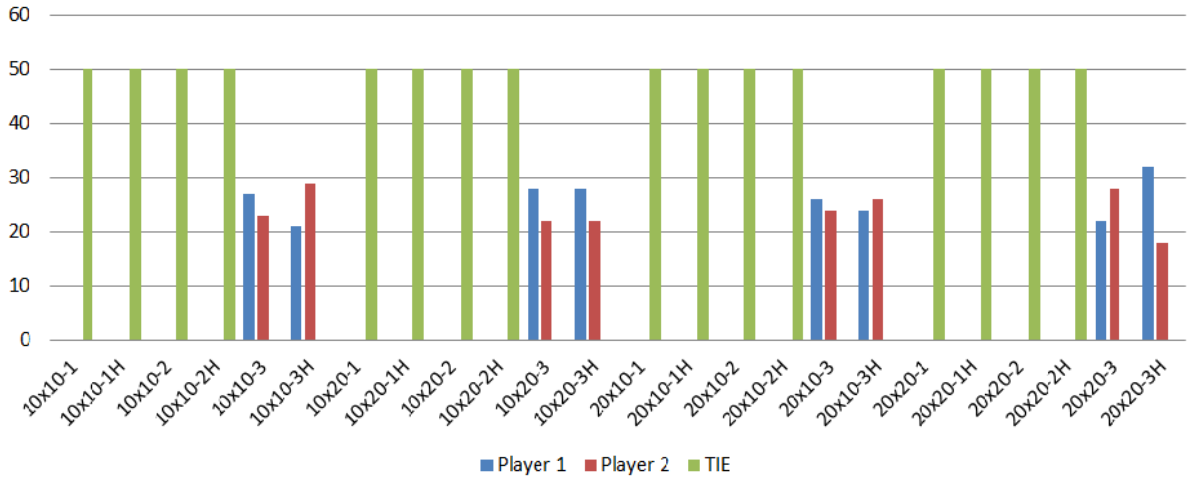

Figure 5.5: This chart shows the results of the different setups for Random against Random

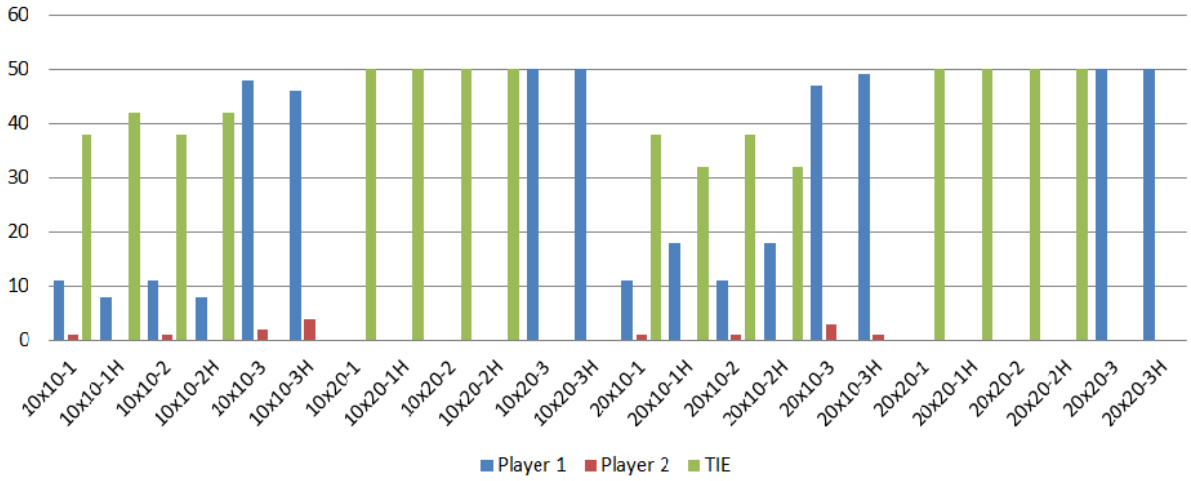

Figure 5.6: This chart shows the results of the different setups for Monte Carlo against Random

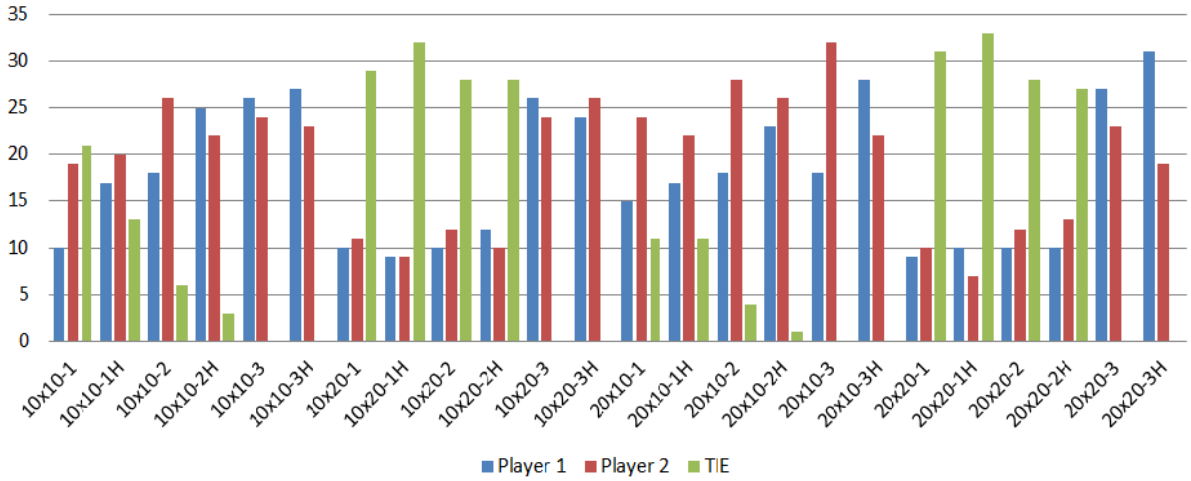

Figure 5.7: This chart shows the results of the different setups for Monte Carlo(100) against Monte Carlo(1000)

Chapter 6

Conclusions and Future Work

We have examined different strategy players for multiplayer Tetris. We have monitored their performance by having them perform against one another in different environments. The environments consisted of three different goals, board sizes and the possibility of using the Hold-function. The three different goals for the players to achieve were: (1.) The player who has cleared the most lines at the end of a game is the winner; (2.) The player who is the first to clear a line, wins the game; (3.) The player whose Tetromino sticks out above row m in an $m \times n$ sized board loses the game.
We examined how the Random algorithm, Alpha-beta algorithm, and the Monte Carlo algorithm performed under different environments. We did this by letting the strategies compete against one another in two-player setups. We also examined whether having a Hold-function in the game can result in a better performance of the strategies. After running several experiments we collected and analysed the results.

We will discuss the outcome. Every setup comes with interesting results as has been seen in the previous chapter. We wanted to test the setups for different goals to see if there was going to be an improvement in performance for the strategies. We were able to determine the following:

1. The AlphaBeta Pruning search algorithm, where the maximum depth was 2, played significantly better than the Monte Carlo algorithm which generated 100 Random playouts per turn. This showed in the AlphaBeta vs. Monte Carlo setup but also when both of the strategies played against the Random strategy, which showed that the AlphaBeta Pruning algorithm held up slightly better against the Random algorithm than the Monte Carlo algorithm.

2. The evaluation function we had set up for our AlphaBeta Pruning Max-player held up well against the simple evaluation function of the Min-player when the maximum depth was 2. This turned around when the depth was set to 3.

3. For the AlphaBeta Pruning strategies on depth 3 it also showed that the Max-player with a more thorough evaluation only won when it came down to goal 3, which is the goal where the losing player is the one whose Tetromino causes the game to end. The evaluation function was apparently strong enough to perform well for that goal.

4. It also showed that the Monte Carlo algorithm performed slightly better when the random playouts were increased to 1000 per turn. This shows that the Monte Carlo algorithm might give a significantly different outcome in the setups that were tested if the Random playouts per turn are increased.

5. Based on the results we can also conclude that having a Hold-function in a Tetris game significantly improves the performance of a decent strategy-based player.

6. Many games for goals 1 and 2 resulted in ties when $n = 20$ for an $m \times n$ board. This shows that the width of the board influences the performance of a strategy-based player.

7. Using goal 2 showed an improvement of the performance of the strategies at times compared to goal 1, but this did not occur frequently enough to say anything about how much better goal 2 is compared to goal 1.

There are still some aspects that can be examined in this field of research. The board size can be increased or decreased to see if this would result in the strategies performing significantly better. The Monte Carlo algorithm can also be tested with an increased amount of random playouts per turn. The AlphaBeta Pruning search algorithm can also be tested with an increased maximum depth. Monte Carlo tree search can also be applied on these setups and environments. It can also be examined whether the player who is the first to make a move during a game will affect the performance of the strategies. Lastly, it might be interesting to implement existing Tetris gameplay strategies like Tetris or T-Spin to see which one performs better in certain environments. These slight changes might improve the performance of the strategies and might result in different outcomes. The research behind the performance of strategies in a multiplayer Tetris environment is still anything but done.

References

[1] Webpage Artificial Intelligence — Assignment 1: Tetris, LIACS—Informatica, Leiden University, http://liacs.leidenuniv.nl/~kosterswa/AI/teet2015.html [Accessed: 24.5.2016]

[2] S.J. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, Englewood Cliffs, New Jersey, Third Edition, 2009

[3] A. Boumaza, How to Design Good Tetris players, Inria Nancy, In: Proceedings Grand Est, LORIA - AIS - Department of Complex Systems, Artificial Intelligence & Robotics, 2011, [Online], Available from: https://hal.inria.fr/hal-00926213 [Accessed: 01.09.2016]

[4] A. Boumaza, On the Evolution of Artificial Tetris players, In: Proceedings 2009 IEEE Symposium on Computational Intelligence and Games, 387–393, 2009

[5] H. Burgiel, How to Lose at Tetris, Mathematical Gazette, 81:194–200, 1997

[6] M. Rovatsou and M.G. Lagoudakis, Minimax Search and Reinforcement Learning for Adversarial Tetris, In: Proceedings: 6th Hellenic Conference on AI, SETN 2010, Athens, Greece, Lecture Notes in Computer Science, 6040: 417–422, 2010

[7] E.D. Demaine, S.Hohenberger and D. Liben-Nowell, Tetris is Hard, Even to Approximate, In Proceedings: Computing and Combinatorics, 9th Annual International Conference, COCOON 2003, Big Sky, Montana, USA, 267–272, 2004

[8] R. Breukelaar, H.J. Hoogeboom, W.A. Kosters, E.D. Demaine, S. Hohenberger, and D. Liben-Nowell, Tetris is Hard, Even to Approximate, International Journal of Computational Geometry & Applications, 14: 41–68, 2004

[9] N. Böhm, G. Kókai, S. Mandl, An Evolutionary Approach to Tetris, In: Proceedings The Sixth Metaheuristics International Conference (MIC 2005), Vienna, Austria, 1–6, 2005

[10] E. Shahar and R. West, Evolutionary AI for Tetris, Student Reports, Massachusetts, U.S.A., Fall 2010, [Online], Available from: http://www.cs.uml.edu/ecg/pub/uploads/AIfall10/eshahar_rwest_GATetris.pdf [Accessed: 08.09.2016]

[11] L. Flom and C. Robinson, Using a Genetic Algorithm to Weight an Evaluation Function for Tetris, Colorado, U.S.A., [Online], Available from: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.132.1712 [Accessed: 09.09.2016]

[12] Wikipedia, Tetris, [Online], Available from: https://en.wikipedia.org/wiki/Tetris [Accessed: 01.09.2016]

[13] C. Browne, E.Powley, D.Whitehouse, S. Lucas, P.I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis and S. Colton, A Survey of Monte Carlo Tree Search Methods, IEEE Transactions on Computational Intelligence and AI in Games, 4: 1–43, 2012.

[14] H.J. Hoogeboom and W.A. Kosters, How to Construct Tetris Configurations, International Journal of Intelligent Games & Simulation, 3: 97–105, 2004.

[15] H.J. Hoogeboom and W.A. Kosters, Tetris and Decidability, Information Processing Letters, 89: 267–272, 2004.

[16] H. Baier and P. D. Drake, The Power of Forgetting: Improving the Last-Good-Reply Policy in Monte Carlo Go, IEEE Transactions on Computational Intelligence and AI in Games, 2: 303–309, 2010.

[17] Giant Bomb, Tetris (PlayStation Network (PS3)) Review, [Online], Available from: http://www.giantbomb.com/images/1300-1677792 [Accessed: 14.09.2016]