Universiteit Leiden
The Netherlands

Opleiding Informatica

AI agents for the

abstract strategy game Tak

Laurens Beljaards

Supervisors:
dr. W.A. Kosters
dr. J.M. de Graaf

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

25/07/2017

Abstract

Tak is an abstract strategy game in which a white and a black player place, move and stack stones to be the first to build a road that connects two opposite edges of the board. As the game progresses, more stones will be piled up into stacks with increased mobility, which adds another layer of depth to the game.

We look at a variety of strategies for agents in playing a game of Tak. Among others, we measure the competence of a Monte Carlo Tree Search agent that makes decisions based on random game sampling, and a strategic agent that uses an evaluation function to rate moves by heuristic value. We determine which of the algorithms is best-suited for the game, investigate the influence the color has on the win chance for the players, and give an in-depth analysis of the complexity of Tak.

Contents

Chapter 1

Introduction

Tak is an abstract strategy board game for two players, designed by James Ernest and Patrick Rothfuss, and published by Cheapass Games in 2016 [Wik16]. The objective of Tak is to build a road that connects two opposite edges of the board with stones. An example is shown in Figure 1.1.

The focus of this thesis is to create AI agents capable of playing Tak intelligently, on a level that is challenging to human players. This can help in gaining more insight into the game of Tak, specifically in its complexity and strategies. Another goal is to get a better understanding of the MCTS and negamax algorithms, how these algorithms can be further optimized, and which type of algorithm is the best fitting for the game Tak.

For this purpose, a C++ program was written that allows Tak to be played. Various agents were created along with the Tak program, most notably Monte Carlo and Monte Carlo Tree Search agents that make decisions based on random game sampling, and a strategic agent that uses a negamax tree search algorithm with an evaluation function to rate moves by heuristic value. The competence of the different agents and the advantage for either color will be estimated by having the agents play a large number of games against one another.

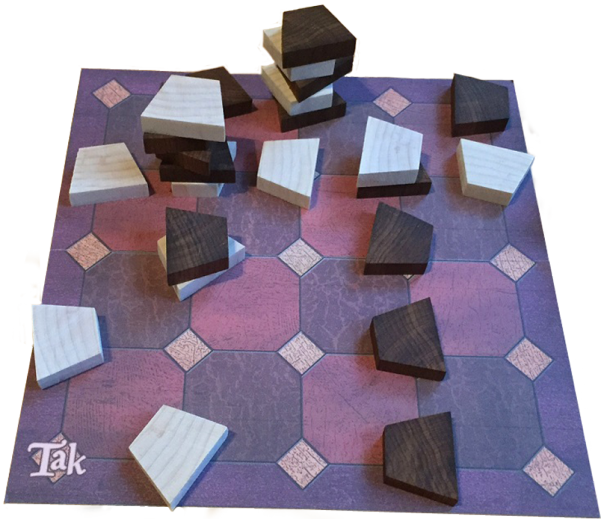

Figure 1.1: The board of Tak. White won this game by building a road that connects the left-hand side with the right-hand side of the board. Image adapted from *http://gamingtrend.com/wp-content/uploads/2016/05/Victory.jpg*

The rules of Tak are explained in Chapter 2, and previous work on the game and used techniques are mentioned in Chapter 3. In Chapter 4, an in-depth analysis of the complexity of Tak is given. Chapter 5 explains the different agents for playing Tak. We will clarify how the strategies work and how these agents are implemented. In Chapter 6 we analyze how the different agents perform against each other and we make an estimation of the discrepancy between win chances of the white and black player. Based on these results, we will draw conclusions about the agents and the game Tak in Chapter 7. Lastly, we will give suggestions for future research to expand upon this thesis and improve the agents and strategies.

This bachelor thesis was written for the Leiden Institute of Advanced Computer Science (LIACS) at Leiden University, and was supervised by Walter Kosters and Jeannette de Graaf.

Chapter 2

Game rules

In this chapter, we explain the rules of Tak. This is a concise explanation based on the official rules [Rul16] and Tak's Wikipedia page [Wik16].

Tak is played on boards of $m \times n$ squares. The amount of stones in each player's reserve depends on the size of the board. All Tak games start with an empty board.

Board Size	3×3	4×4	5×5	6×6	7×7	8×8
Normal Stones	10	15	21	30	40	50
Capstones	0	0	1	1	2	2

Table 2.1: Pieces for each player per board size.

We describe several notions of Tak:

Taking turns On each player's first turn, they must place one of their opponent's normal stones flat on any empty space on the board. During any other turn, players must either place a stone or move a stack under their control.

Placement The player plays one stone from their reserve onto an empty spot on the board. There are three types of placement:

- Flat stone: A normal stone played flat. Flat stones can be stacked upon.
- Wall: A normal stone played on its edge. Nothing can be stacked upon a wall, but walls do not count as part of a road.
- Capstone: The most powerful piece, as it counts towards a road and cannot be stacked upon. It also has the ability to "flatten" a wall into a flat stone by moving onto it.

Movement A player moves some or all of the top stones of a stack under their control in one direction (up to min(m,n) stones). The color of the stone on top determines which player has control of the entire stack.

As a stack is moved in that direction, one or more of its bottom stones must be dropped off on top of each space or stack it traverses, see Figure 2.1.

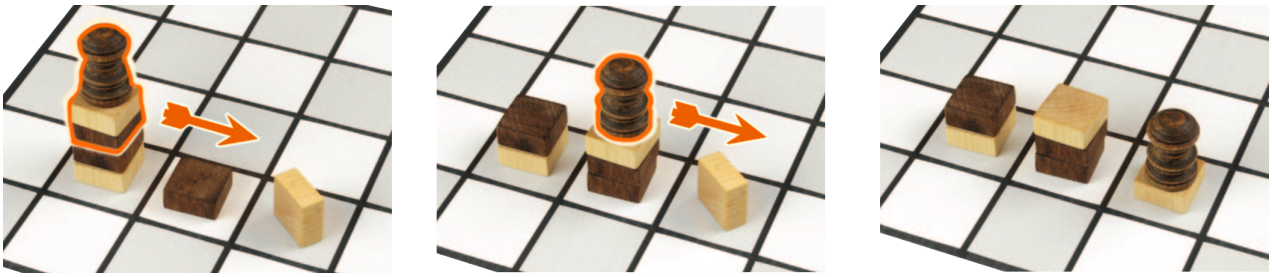

Figure 2.1: Black moves the top 3 stones of the stack to the right, dropping off at least one stone per traversed tile.

End of the game The primary way to win is to build a road of your color, see Figure 2.2, but the game also ends if either player runs out of pieces, or if the board is completely full. In that event, the winner is the player with the most flat stones on top.

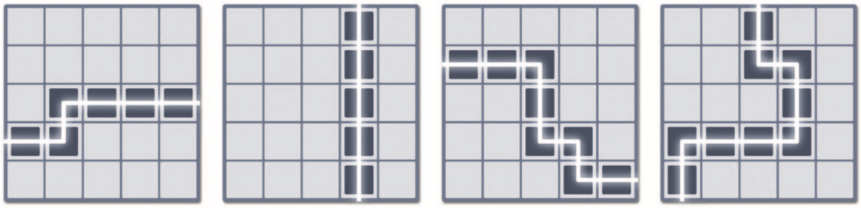

Figure 2.2: Examples of a road victory.

Tak is originally meant to be played on square boards from 3×3 to 8×8. We consider a generalized variant for which Tak is played on boards of $m \times n$.

Chapter 3

Related Work

The game Tak was published by Cheapass Games in 2016. As of writing, no papers have been pubished on it yet. There is a fanmade site [Pla16] on which Tak can be played online and on which replays are publicly accessible.

Tak is a connection game [Bro05]. Without the rules of movement and wall placements, Tak would be similar to the connection game Hex on a square board, in the sense that players take turns playing stones with the goal of connecting two opposing sides of the board [Bro00].

The paper *Games solved: Now and in the future* [HUR02] covers the complexity and the existence of winning strategies for other abstract strategy games. The explained techniques may prove useful for Tak as well.

Chapter 4

Complexity

In this chapter, we will give an in-depth analysis of the complexity of Tak. In Section 4.1 we will determine a formula for calculating the number of possible moves resulting from specific stacks, and use it to construct boards with worst-case branching factors. We will also take a short look at boards with best-case branching factors. In Section 4.2, we calculate an upper bound for the state space complexity of Tak.

4.1 Stack movement and branching factors

If there is enough space, a stack of h stones can be moved in

$$4 * (2^h - 1) \tag{4.1}$$

different ways. Namely, for each direction applies: starting from the bottom stone you can choose for each of the h stones whether to split it, moving it and the stones above it to the next space, or not. If we exclude the possibility of never splitting, as that is not a move, this results in $2^h - 1$ possible moves in each direction, see Figure 4.1.

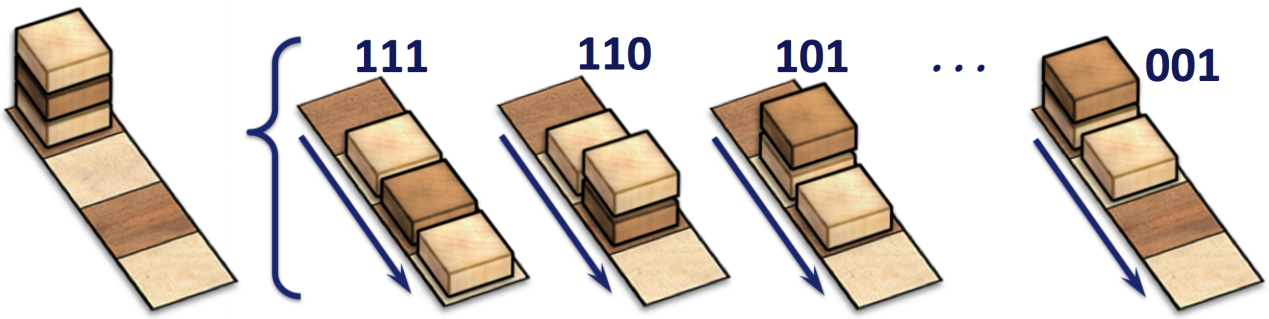

Figure 4.1: A stack of height $h = 3$ can be moved in 7 different ways per direction. A '1' denotes a split.

For normal games, the amount of possible moves in one direction is of course limited by the number of available spaces, s. In such a case, only moves with up to s splits are valid, again excluding the no splits option

$s = 0$. The number of moves in the given direction with exactly i splits is equal to h *choose* i. Also note that by the rules of Tak, the amount of stones that may be moved at once on a $m \times n$ board is bounded by min(m,n). Therefore, the number of possible moves for the stack in the given direction is:

$$\sum_{i=1}^{s} \binom{\min(h, m, n)}{i} \tag{4.2}$$

4.1.1 Worst-case boards

Using Equation 4.2, we can determine the worst-case scenario for the number of moves on a given board size. This shows us the maximum branching factor of Tak for that board size.

If not obstructed by walls or capstones, a stack of $\geq n$ stones can always be moved in $2^{n+1} - 4$ (Equation 4.4) different ways on a square board of $n \times n$, regardless of where that stack is on the board.

Proof: Given the $n \times n$ board, our stack must have k spaces to the left and ℓ spaces to the right with $k + \ell = n - 1$. Using Equation 4.2, the amount of moves to the left plus the right is equal to:

$$\begin{aligned} &\sum_{i=1}^{k} \binom{n}{i} + \sum_{i=1}^{\ell} \binom{n}{i} \\ &= \sum_{i=1}^{k} \binom{n}{i} + \sum_{i=1}^{n-k-1} \binom{n}{i} \\ &= \sum_{i=1}^{k} \binom{n}{i} + \sum_{i=k+1}^{n-1} \binom{n}{i} \\ &= \sum_{i=1}^{n-1} \binom{n}{i} \\ &= 2^n - 2 \end{aligned} \tag{4.3}$$

This works similarly for moves upwards and downwards. Hence, we can multiply Equation 4.3 by two to obtain the formula for the number of moves in all four directions:

$$2 * (2^n - 2) = 2^{n+1} - 4 \tag{4.4}$$

For constructing worst-case boards for the number of moves, we need to make sure that there are no roads and that both players have stones left in their reserve, as the number of moves is of course equal to zero after the game has ended. Additionally, we must take into account that a player always ends his turn with at least one piece of his color on top of a tile or stack. Worst-case boards for 5×5, 3×3 and 8×8 are shown in Figure 4.2, Figure 4.3 and Figure 4.4, respectively. These boards are reachable by having the players essentially skip turns by moving a piece of theirs back and forth or swap turns as illustrated in Figure 4.6.

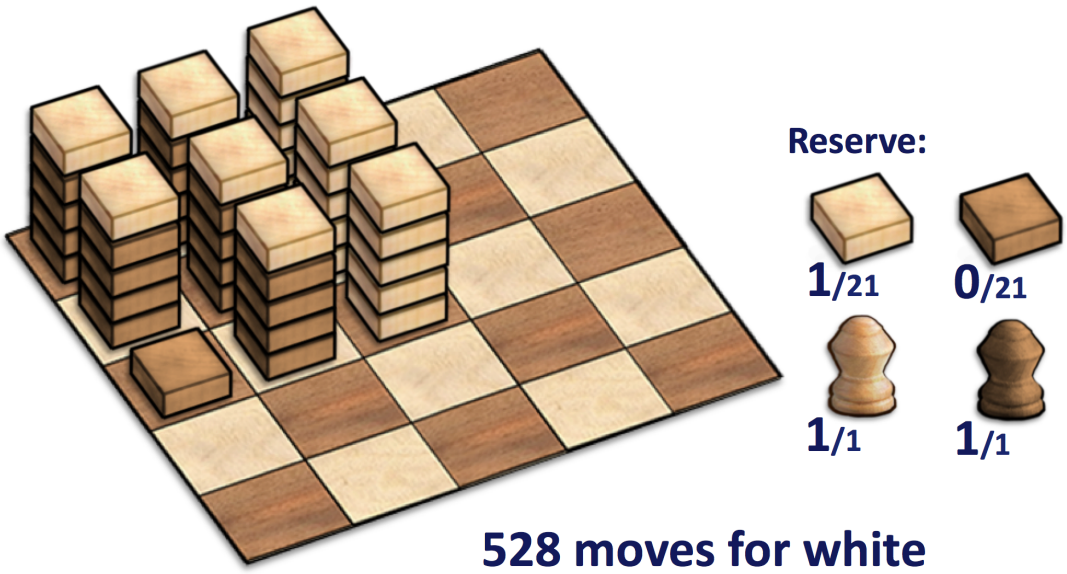

Figure 4.2: Worst case for a 5x5 board: 8 stacks of height 6 ($8 * 60$) and 16 empty tiles ($16 * 3$) = 528 moves.

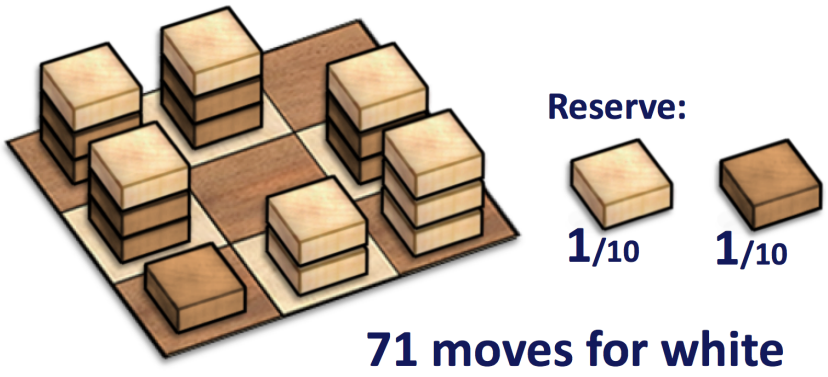

Figure 4.3: Worst case for a 3x3 board: 5 stacks of height 3 ($5 * 12$), a stack of height 2 (7) and 2 empty tiles ($2 * 2$) = 71 moves.

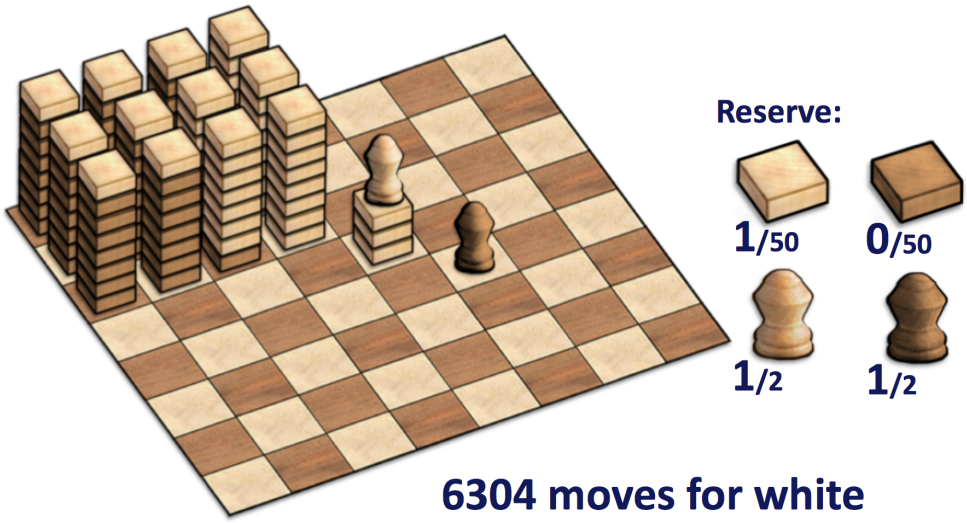

Figure 4.4: Worst case for a 8x8 board: 12 stacks of height 8 ($12 * 508$), a stack of height 4 (58) and 50 empty tiles ($50 * 3$) = 6304 moves.

4.1.2 Best-case boards

We take a short look at best-case boards with the fewest possible moves. Because the game ends when the board is full or a player has no stones left in his reserve, a player can always do a place move. For board sizes ≤ 4, the stone in the reserve is always a normal stone. As a result, there will always be at least 2 possible moves: placing the stone flat or placing the stone as a wall. For larger board sizes, there can be just 1 possible move if only capstones are left in the player's reserve. An example is shown in Figure 4.5.

The aforementioned best-case boards can be constructed easily by placing walls on all tiles but one, while making sure that the walls surrounding the empty tile are of the opponent's color, so the only move possible is a place move.

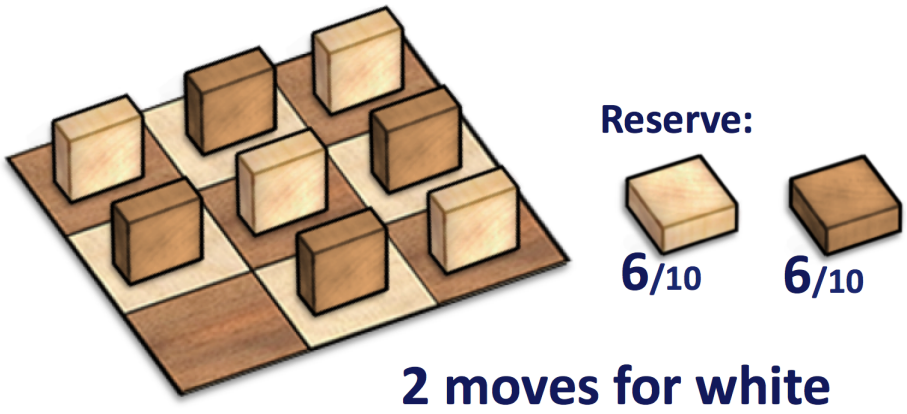

Figure 4.5: Best case for a 3x3 board: No stones can be moved. On just one tile a flat stone or wall can be placed.

4.2 State space complexity

In this section, we calculate an upper bound for the state space complexity of Tak.

4.2.1 Flat stones

Given an $m \times n$ Tak board with $m \cdot n = t$ tiles ($m, n \geq 1$). The amount of unique legal configurations of the board containing exactly ws white flat stones and exactly bs black flat stones is equal to:

$$\text{ConfigsExact}(t, ws, bs) = \frac{(t-1+ws+bs)!}{(t-1)! * ws! * bs!} = \text{multinomial}(t-1, ws, bs) = \binom{t-1+ws+bs}{t-1, ws, bs} \tag{4.5}$$

Namely, board configurations can be described as a row of $t-1+ws+bs$ elements: ws white flat stones, bs black flat stones and $t-1$ separators to denote the different tiles. This yields $(t-1+ws+bs)!$ permutations, but as stones are indistinguishable from stones of the same colour, and the tile separators can not be distinguished from one another either, we have to divide the equation by the three permutations $ws!$, $bs!$ and $(t-1)!$.

Using Equation 4.5, we find that a board with up to *maxws* white flat stones and up to *maxbs* black flat stones can have

$$\text{Configs}(t, maxws, maxbs) = \sum_{ws=0}^{maxws} \sum_{bs=0}^{maxbs} \text{ConfigsExact}(t, ws, bs) \tag{4.6}$$

possible configurations.

Note that in case of very small boards, most of these configurations are not reachable in a normal game from the initial empty board. As an extreme example, almost none of the configurations are reachable on a 1 by n board, as the game already ends after placing the very first stone.

For larger m and n, an increasingly larger part of the configurations is reachable, as there is enough space to place and order stones on such boards. Even large boards will always have some configurations that are unreachable, though. Consider a configuration with two roads, which are separated by empty tiles or walls. These roads can not have both been produced by any one single move. As the game instantly ends after one move produces a road, a normal game will end before this configuration can be reached in any way.

More unreachable configurations can be discovered by checking a number of factors: the game ending before the state can be reached, states that can not have been constructed due to the rule that walls and capstones can not be stacked upon, the rule that new stones must be placed on empty tiles, and the fact that any turn always ends with at least one piece of that player's color on top.

4.2.2 Walls and capstones

Consider an empty board without normal stones or capstones. We can place exactly *ww* white walls and exactly *bw* black walls on it in

$$\text{ConfigsExact}(t, ww, bw) = \begin{cases} 0 & \text{if } ww + bw > t \\ \binom{t}{t - ww - bw, ww, bw} & \text{otherwise} \end{cases} \tag{4.7}$$

different ways.

We can extend this equation to include *wc* white capstones and *bc* black capstones, resulting in equation Equation 4.8:

$$\text{ConfigsExact}(t, ww, bw, wc, bc) = \begin{cases} 0 & \text{if } ww + bw + wc + bc > t \\ \binom{t}{t - ww - bw - wc - bc, ww, bw, wc, bc} & \text{otherwise} \end{cases} \tag{4.8}$$

This means that a board with up to *maxww* white walls, up to *maxbw* black walls, up to *maxwc* white capstones and up to *maxbc* black capstones can have

$$\text{Configs}(t, maxww, maxbw, maxwc, maxbc) = \sum_{ww=0}^{maxww} \sum_{bw=0}^{maxbw} \sum_{wc=0}^{maxwc} \sum_{bc=0}^{maxbc} \text{ConfigsExact}(t, ww, bw, wc, bc) \tag{4.9}$$

possible configurations.

4.2.3 Total number of configurations

As there is no limit to stack height, placing any amount of flat stones on a tile does not invalidate the placement of a wall or capstone on that tile. Therefore, a board with t tiles and any amount of flat stones will still have t tiles remaining that can be stacked upon. This means that we can combine all possible configurations of flat stones (Equation 4.6) with all possible configurations of walls and capstones (Equation 4.9) to get the total number of configurations for a board size, as long as the number of flat stones plus the number of walls on the board does not exceed the initial size of the owner's stone reserve:

$$\text{Configs}(t, maxws, maxbs, maxwc, maxbc) = \sum_{ws=0}^{maxws} \sum_{bs=0}^{maxbs} \left(\binom{t-1+ws+bs}{t-1, ws, bs} * \text{Configs}(t, ww, bw, wc, bc) \right) \tag{4.10}$$

We can further lower this bound by looking at the forced stone placement during the first round: Each player's first turn consists of playing an opponent's stone flat on the board. Therefore, any board after the initial two turns will contain at least one white and one black flat stone. The number of possible configurations until the end of white's first turn is equal to $1 + m * n$. Namely the empty board, or any configuration with only one black flat stone on it.

Finally, we account for which player is to move in a given configuration. After all, one configuration can have different playouts depending on who is to move. Note that in almost any configuration a turn can essentially be "swapped" between players, as illustrated in Figure 4.6.

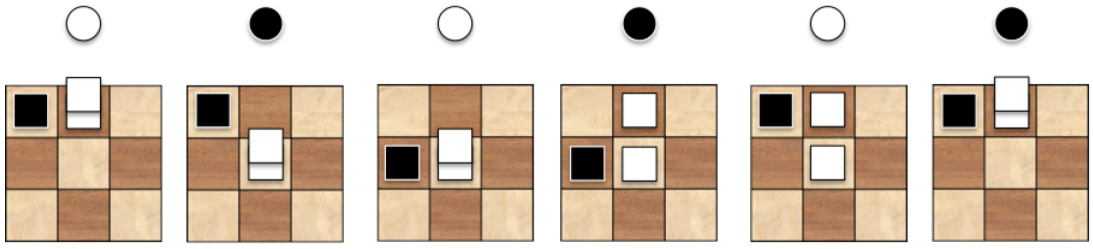

Figure 4.6: Depending on play history, one specific board setup can have different players to move.

Having taken this into account, the upper bound for the number of states is:

$$\text{Configs}(t, maxws, maxbs, maxwc, maxbc) = 1 + m * n + 2 * \sum_{ws=0}^{maxws} \sum_{bs=0}^{maxbs} \left(\binom{t-1+ws+bs}{t-1, ws, bs} * \text{Configs}(t, ww, bw, wc, bc) \right) \quad (4.11)$$

Final states in which the last player to move no longer has stones left in his reserve are reachable in normal games. States in which the opponent has no stones left in his reserve, however, are not reachable. The upper bound after excluding such states is:

$$\text{Configs}(t, maxws, maxbs, maxwc, maxbc) = 1 + m * n + \sum_{p=1}^{2} \sum_{ws=1}^{maxws} \sum_{bs=1}^{maxbs} \binom{t-1+ws+bs}{t-1, ws, bs} * \sum_{ww=0}^{maxws-ws} \sum_{bw=0}^{maxbs-bs} \sum_{wc=0}^{maxwc} \sum_{bc=0}^{maxbc} \begin{cases} 0 & \text{if } ww+bw+wc+bc > t \\ & \text{or } (p = 1 \text{ and } bs+bw+bc = maxbs+maxbc) \\ & \text{or } (p = 2 \text{ and } ws+ww+wc = maxws+maxwc) \\ \binom{t}{t-ww-bw-wc-bc, ww, bw, wc, bc} & \text{otherwise} \end{cases} \quad (4.12)$$

Board Size	Stones	Capstones	Number of States	Exact Number of States
3×3	10	0	$9.63171 \cdot 10^{13}$	96317109784544
4×4	15	0	$1.86068 \cdot 10^{23}$	186068001400694400139264
5×5	21	1	$1.73738 \cdot 10^{37}$	173737646960094203027341667496791900 16
6×6	30	1	$2.34954 \cdot 10^{53}$	234953877228339135788770198553992010653918615167827968
7×7	40	2	$1.28701 \cdot 10^{74}$	128700537923578263555452692444323607426853504501698758061991500643148234752
8×8	50	2	$5.55774 \cdot 10^{93}$	555773973645223473071469906718460888132914465305140809038585175165518414179421639013386158080 0

Table 4.1: The results of Equation 4.12 for various board sizes: an upper bound for the number of reachable states of Tak.

Configurations can be grouped in sets of between 1 to 8 states that can be considered "similar" by rotation and reflection. On boards where $m \neq n$, these groups contain only between 1 and 4 similar states. If you also consider configurations where both the turn and the colors of all pieces are swapped similar, the size of these groups of states that play out similarly can be twice as large.

4.3 Winning strategies

For some two-player games, a strategy stealing argument [HUR02] can be used to prove that with optimal play, the game is not winnable for the second player to move, using the fact that doing a move is never

disadvantageous in that game: If a winning strategy for the second player were to exist, player one could do a random move, which is not disadvantageous to himself, and after that adopt the hypothetical strategy of the second player to win. Therefore, a winning strategy for the second player can not exist in such games. In the game of Tak however, moves are not unconditionally beneficial to the player who does them. For example, an extra stone on the board could give the opponent more mobility in a stack or end the game by depletion of the first player's reserve, creating the possibility of the second player winning by top stone count. The existence of zugzwang positions in Tak, as shown in Figures 4.7 and 4.8, proves that there exist situations in which making a move is disadvantageous to the moving player. Therefore, a strategy stealing argument can not be used to prove or disprove the existence of a strategy for Tak that guarantees at least a draw for the white player.

Figure 4.7: Mutual zugzwang on a hypothetical 2×3 board. If the first player to move places a new capstone, the opponent will win by flat stone count. If the first player moves one of his capstones instead, the opponent will win by making a horizontal road.

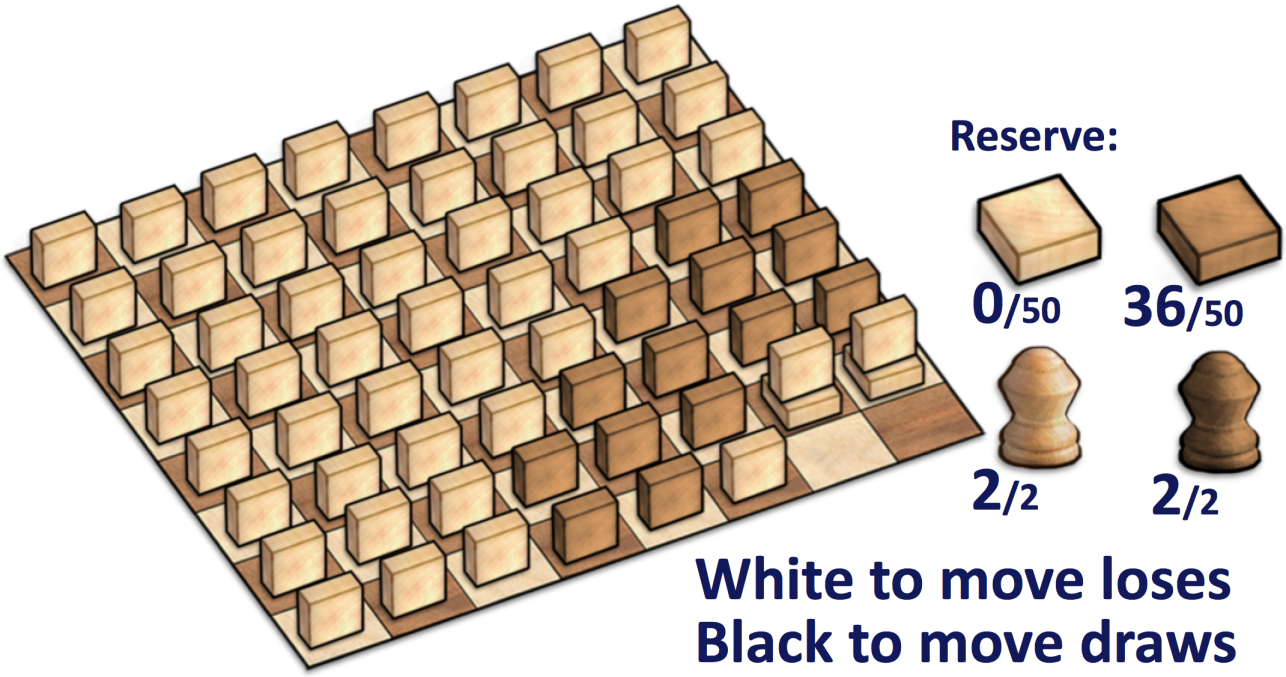

Figure 4.8: Zugzwang on an 8×8 board. White to move loses; Black to move draws.

Chapter 5

Agents

In this chapter, the different agents for playing Tak will be explained, such as the Monte Carlo Tree Search agent that makes decisions based on random game sampling, and the negamax tree search agent that uses an evaluation function to rate moves by heuristic value. We will explain how the strategies work, and how these different agents are implemented.

5.1 Random player

The *pure random player* counts all possible moves and picks one uniformly at random. Because the number of possible moves resulting from a stack increases exponentially with stack height, splitting out a higher stack becomes exponentially more probable relative to doing a place move. As this may be undesirable, we also created a *weighted random player*, which has heuristic weights assigned to categories of moves, influencing the probability of them being selected. It has an increased probability of doing place moves when the amount of stones on the board is small. Also, its weight of placing a flat stone is three times as high as placing a wall. As walls are often played defensively, but generally do not contribute as much to road potential as flat stones do, this should increase the quality of the average move. Additionally, as the weighted random player assigns weights per tile instead of individual moves, a high stack of stones does no longer have an exponentially larger chance to be selected. This is also an improvement, as placing a stone practically always improves the player's board, and usually has a higher probability of being a decent move than splitting a stack.

5.2 Monte Carlo strategy

The *Monte Carlo player* probabilistically determines its move by the use of pure Monte Carlo game search: For each possible move, k games are played out with the weighted random player. Each move is assigned a score, depending on the amount of won, tied and lost playouts. The move with the highest score is chosen, see

Figure 5.1. If the highest score is tied between multiple moves, one of them is chosen at random. The Monte Carlo player also has the functionality to play out games until a certain depth. If a playout would prolong after that, the playout is interrupted and rated as the average score between a win and a loss.

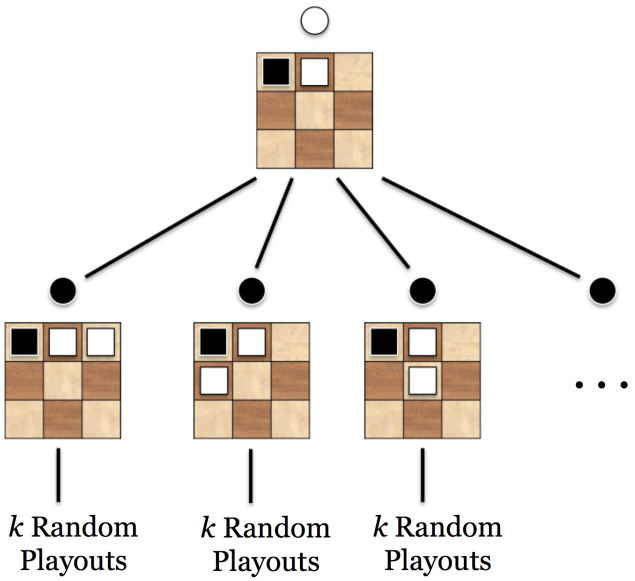

Figure 5.1: The Monte Carlo Player (white) does playouts for every possible move, and picks the move with the best results.

Monte Carlo Tree Search [MCT17, BPW+12] makes use of a gradually expanding tree that keeps track of the score and the number of playouts done for all explored moves, see Figure 5.2. The tree favors expanding towards moves with a high win rate, but also tries to explore moves for which the expected score has low certainty due to a small number of simulations. On each level of the tree, more playouts are assigned to such moves.

In one MCTS playout, the algorithm selects the child node that has the highest value for Equation 5.1, where w_i and n_i are the number of wins and the number of playouts for the i^{th} child node, and $c = \sqrt{2}$. The left component of the expression rates moves according to exploitation, prioritizing moves with a high win rate. The right component rates exploration, prioritizing moves that have not been played out often. This is repeated on all levels of the tree, until a node is reached that has not yet been expanded. In that case, child nodes corresponding to possible moves are generated, and one of them is selected at random. One Monte Carlo playout is done from this node. The result of this playout is propagated from this node back to all parent nodes, adjusting the score and number of playouts variables in each visited node. In order to account for the fact that the opponent will make the play that leads to the least value for the agent, the scores of moves are inverted on every other level of the tree. Many of these MCTS-playouts are done until the timer expires, after which the agent will choose the move that has had the most playouts assigned to it.

$$\frac{w_i}{n_i} + c \cdot \sqrt{\frac{1}{n_i}} \tag{5.1}$$

We use expression 5.1 above in our MCTS-agent, as it yielded better results than the standard UCT formula

for balancing exploration and exploitation in Monte Carlo Tree Search [KS06], where t is the total number of playouts done for the parent node, see Equation 5.2.

$$\frac{w_i}{n_i} + c \cdot \sqrt{\frac{\ln(t)}{n_i}} \tag{5.2}$$

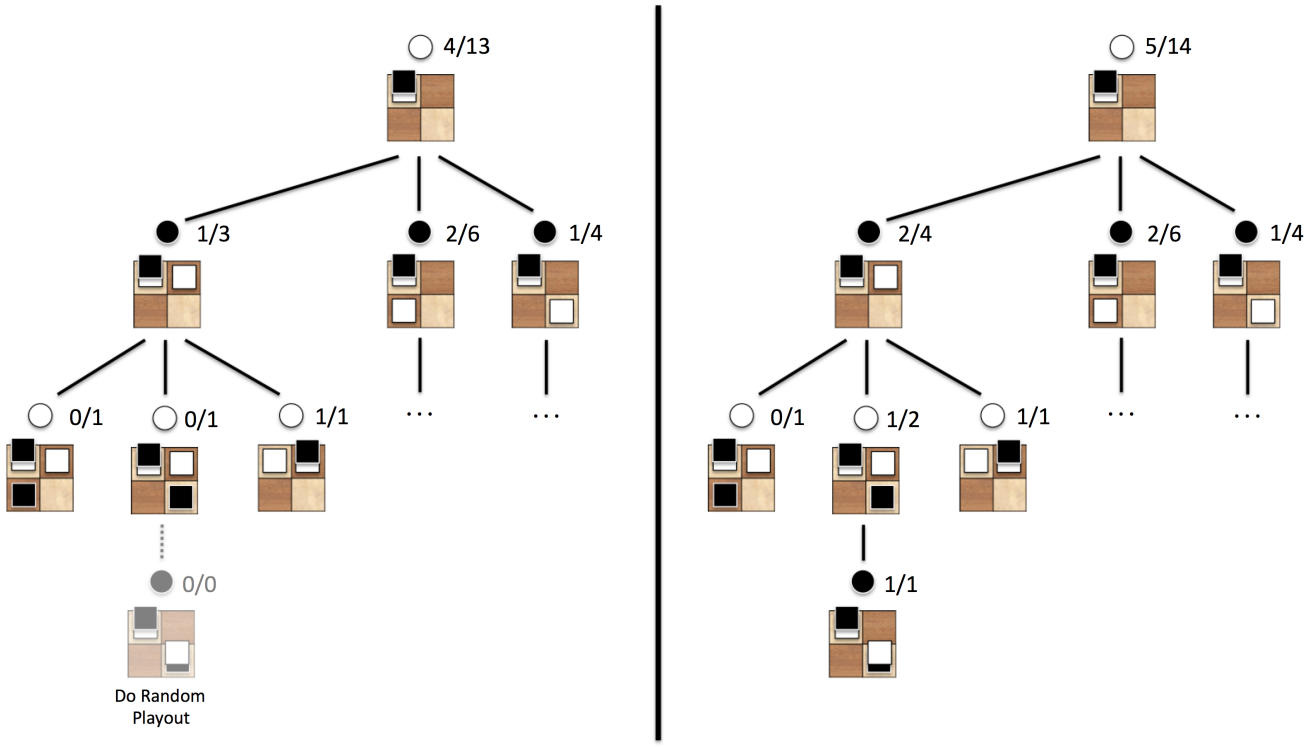

Figure 5.2: (Left:) MCTS expands a selected node and does a random playout for one child. (Right:) White won the random playout and the result is backpropagated. In each node the score for white and the total number of playouts is remembered.

5.3 Negamax agent

The *negamax agent* uses the *negamax* [Neg17, DP13] game tree search algorithm. Each node in the tree corresponds to a board state. A path from the root to a leaf node corresponds to a valid sequence of moves. Assuming optimal play by both players, white is guaranteed to choose the transition that leads to the child node with the highest return value whenever he is to move. As Tak is a *zero-sum game*, black will in turn always choose the move that leads to the least value for white. Using this information, a tree can be generated to calculate the optimal responses to every move possible from the root, after which the agent can select the optimal move. The negamax algorithm simplifies the code of the decision process by inverting the values between every level of the tree. Thus in nodes where white is to move, the move with the highest value for white is selected, while in black nodes the move with the highest negated value, or the lowest value for white, is chosen.

The algorithm needs to be able to compare non-final board states. Therefore, it makes use of an *evaluation*

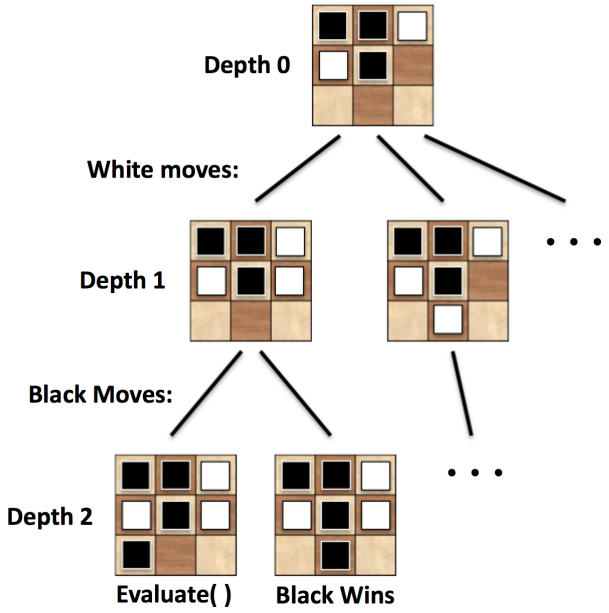

Figure 5.3: Negamax tree example for depth 2. Black will always choose the winning move (∞ value). Therefore, white will not do the move that leads up to this (as that move is valued $-\infty$).

function that determines a heuristic value for any given board state. This value is calculated by rating various factors, including pieces on the board, area control, stack control and mobility.

The tree is explored using *iterative deepening*. Multiple searches are done after one another, with every search having a higher maximum search depth than the previous one. The agent may be interrupted at any time, after which the best move found in the deepest search completed so far is returned.

Alpha-beta pruning is used to greatly reduce the number of nodes that negamax needs to explore, by pruning branches that can never be reached with optimal play, whenever it becomes known the other player had a strictly better move available previously.

If better moves are evaluated first, alpha-beta is more likely to prune subtrees. Therefore, ordering moves may increase the speed of the algorithm. However, the ordering of moves may also be time expensive itself. On lower levels of the tree, the amount of time spent ordering moves is higher than the time it would take to evaluate the pruned subtrees. Therefore, we decided to use ordering functions of various speeds for different remaining search depths: From leaf nodes up to nodes at height 3, moves are not ordered. Between height 3 and 8, moves are ordered with a fast ordering function, which orders moves that place a stone before moves that place a wall, and favors moves that result in a greater number of stones for the moving player and a lower number of exposed stones for the opponent. Above height 8, moves are ordered more extensively, exploring moves that leave fewer possible plays for the opponent first.

Another improvement is the use of a statemap. After the children of a node have been visited and the value of the node is calculated, the state corresponding to the node is saved, along with its value and the maximum search depth for which this value was calculated. If later on in the search a similar state is reached, the past result can immediately be used, under the condition that the search depth of the stored state is at least as high

as the current required search depth.

Contrary to convergent games such as chess, Tak has a branching factor that diverges as the game goes on. Therefore, given that the available memory is of course limited, it is more efficient to store only the states that occur during the early turns of the game when doing an exhaustive search. For real-time searches though, the timer will not last long enough for memory to become an issue.

Additionally, board configurations that are rotations or reflections of one another are mapped to the same state in the statemap, greatly reducing the number of nodes that need to be evaluated.

The amount of possible states after the first move is equal to n^2 on an $n \times n$ board. If we count board states that are similar by rotation or reflection only once, the number of states that need to be evaluated is equal to:

$$\frac{\lceil \frac{n}{2} \rceil}{2} \cdot (\lceil \tfrac{n}{2} \rceil + 1) \tag{5.3}$$

After the second move, there are $n^2 \cdot (n^2 - 1)$ possible board configurations. After excluding similar states, this number is equal to the result of Equation 5.4 for odd n:

$$\begin{aligned} 1 \ * \ & \left(\frac{\lceil \frac{n}{2} \rceil}{2} \cdot (\lceil \tfrac{n}{2} \rceil + 1) - 1 \right) \ + \\ (\lceil \tfrac{n}{2} \rceil - 1) \ * \ & (n \cdot \lceil \tfrac{n}{2} \rceil - 1) \ + \\ (\lceil \tfrac{n}{2} \rceil - 1) \ * \ & \left(\frac{n}{2} \cdot (n+1) - 1 \right) \ + \\ \left(\frac{\lceil \frac{n}{2} \rceil - 2}{2} \cdot (\lceil \tfrac{n}{2} \rceil - 1) \right) \ * \ & (n^2 - 1) \\ = \ & \frac{n^4 + 3 \cdot n^2 - 4 \cdot n}{8} \end{aligned} \tag{5.4}$$

The components of the equation above correspond to the number of unique states resulting from the first stone being placed on the center tile; on the horizontal or vertical; on the diagonal; and on other tiles, respectively. For even n we have:

$$\begin{aligned} \frac{n}{2} \ * \ & \left(\frac{n}{2} \cdot (n+1) - 1 \right) \ + \\ \frac{n}{4} \cdot (\frac{n}{2} - 1) \ * \ & (n^2 - 1) \\ = \ & \frac{n^4 + n^2 - 2 \cdot n}{8} \end{aligned} \tag{5.5}$$

The first component of this equation corresponds to the number of unique states resulting from the first stone being placed on the diagonal. The second component accounts for the other cases. Table 5.1 shows the number of states calculated with Equations 5.4 and 5.5 for square boards of size 3×3 to 8×8.

	1st Move:		2nd Move:	
Board Size	#States	Reduced #States	#States	Reduced #States
3×3	9	3	72	12
4×4	16	3	240	33
5×5	25	6	600	85
6×6	36	6	1260	165
7×7	49	10	2352	315
8×8	64	10	4032	518

Table 5.1: The number of states that would need to be evaluated, without and with rotation and reflection detection.

Chapter 6

Results

In this chapter, we will have the agents play games against themselves in order to investigate the win rate for either color. This gives us a way to measure the advantage of the white player, who always moves first, over the black player. If the white player is indeed favored, we expect to see an increase in win rate for white as we move on to better agents. The difference in win rate between white and black should decrease as the board size increases.

Additionally, agents will be put up against agents of other types. We will discuss the results of the Monte Carlo agents and the negamax agent. In order to be able to make a fair assessment, players will be white for half of their games, and black for the other half.

6.1 White-black discrepancy

Tables 6.1 to 6.7 show the results of a large number of games between white and black, played by the random player, the weighted random player or the Monte Carlo player against itself. The number of draws is lower when the weighted random player plays against itself compared to games played by the uniformly distributed random player against itself. The reason for this is that the weighted random player places less walls, thus increasing the probability that a road is made for either player.

As predicted, the win rate for the white player increases when both players are better. The better the players, the closer they are to optimal play, and as determined by game tree search, Tak is winning for white on 3×3 with optimal play.

The difference in win rate between the two colors decreases when the board size increases, as the first-stone advantage of white becomes relatively smaller. Also, the probability of both players having an exactly equal amount of flat stones on top of the board when the game ends with a count is smaller on a large board. Therefore, draws occur less often for larger board sizes.

3×3	100 000 000	Road	No Stones	Board Full
White wins	42 286 908	14 683 548	580 667	27 022 693
Black wins	38 751 156	12 367 916	533 352	25 849 888
Draws	18 961 936	-	356 237	18 605 699

Table 6.1: White versus black, random players on 3×3.

5×5	10 000 000	Road	No Stones	Board Full
White wins	4 392 850	652 193	2 643 812	1 096 845
Black wins	4 254 315	615 459	2 559 404	1 079 452
Draws	1 352 835	-	951 071	401 764

Table 6.2: White versus black, random players on 5×5.

8×8	1 000 000	Road	No Stones	Board Full
White wins	452 685	1 926	449 634	1 125
Black wins	445 855	1 915	442 742	1 198
Draws	101 460	-	101 219	241

Table 6.3: White versus black, random players on 8×8.

3×3	100 000 000	Road	No Stones	Board Full
White wins	56 528 183	47 136 188	36 875	9 355 120
Black wins	37 296 967	31 592 135	32 692	5 672 140
Draws	6 174 850	-	19 376	6 155 474

Table 6.4: White versus black, weighted-random players on 3×3.

5×5	10 000 000	Road	No Stones	Board Full
White wins	5 145 631	4 282 157	386 955	476 519
Black wins	4 575 802	3 783 384	364 408	428 010
Draws	278 567	-	110 622	167 945

Table 6.5: White versus black, weighted-random players on 5×5.

8×8	1 000 000	Road	No Stones	Board Full
White wins	494 080	277 620	215 554	906
Black wins	470 199	260 979	208 323	897
Draws	35 721	-	35 565	156

Table 6.6: White versus black, weighted-random players on 8×8.

5×5	100.0%	Road	No Stones	Board Full
White wins	59.8%	59.8%	0.0%	0.0%
Black wins	40.0%	40.0%	0.0%	0.0%
Draws	0.2%	-	0.0%	0.2%

Table 6.7: White versus black, MonteCarlo players, 1.0 second per turn.

Table 6.8 shows the win rate of white and black for games played by the MCTS agent against itself. The large difference in win rate between the two identical players suggests that white is also hugely favored on 5×5 boards.

The rule of players placing an opponent's flat stone during the first turn has helped to reduce the discrepancy between the win rates of the two colors by 2.6 percentage points (see Table 6.9), but its effect is too weak to even the odds between the colors.

5×5	100.0%	Road	No Stones	Board Full
White wins	63.6%	62.8%	0.8%	0.0%
Black wins	36.4%	36.2%	0.2%	0.0%
Draws	0.0%	-	0.0%	0.0%

Table 6.8: White versus black, MCTS players, 1.0 second per turn.

5×5	100.0%	Road	No Stones	Board Full
White wins	66.2%	65.8%	0.4%	0.0%
Black wins	33.7%	33.4%	0.3%	0.0%
Draws	0.1%	-	0.1%	0.0%

Table 6.9: White versus black, MCTS players. 1.0 second per turn. Shown is the win rate for for either color if the rule of placing an opponent's flat stone during the first turn were not to exist.

6.2 Random players

As can be seen in Tables 6.10 to 6.12, the weighted random player wins significantly more often than the uniformly distributed random player. We can attribute this to the fact that the weighted random player prefers playing flat stones over playing walls or moving stones, which are often poor moves.

In order to win a game, the random player needs to make better moves than the weighted random player. The probability of the random player doing more good moves than the weighted random player in a game is smaller for larger boards, as those games take more moves to play out. Therefore, the winrate of the random player decreases as the board size increases. Additionally, most wins by the random player are as white. On smaller boards, the advantage for white can be high enough to carry the white player to victory, even if he is a worse player, but this does no longer hold on larger boards where the advantage is much smaller.

3×3	100 000 000	Road	No Stones	Board Full
Random wins	8 319 492	6 552 230	48 894	1 718 368
Weighted random wins	87 996 612	68 859 703	920 284	18 216 625
Draws	3 683 896	-	116 287	3 567 609

Table 6.10: Random player versus weighted random player, 3×3.

5×5	10 000 000	Road	No Stones	Board Full
Random wins	158 570	113 537	39 466	5 567
Weighted random wins	9 782 433	6 998 436	2 441 512	342 485
Draws	58 997	-	51 988	7 009

Table 6.11: Random player versus weighted random player, 5×5.

8×8	1 000 000	Road	No Stones	Board Full
Random wins	280	177	103	0
Weighted random wins	999 613	436 416	563 180	17
Draws	107	-	107	0

Table 6.12: Random player versus weighted random player, 8×8.

In Table 6.13 the winrates of the MCTS player using the uniformly distributed random player against the MCTS player using the weighted-random player are shown. The usage of the weighted-random player greatly increased the performance of the MCTS agent.

5×5	100.0%	Road	No Stones	Board Full
MCTS wins (random playouts)	8.6%	8.5%	0.1%	0.0%
MCTS wins (weighted random playouts)	91.4%	85.7%	2.5%	3.2%
Draws	0.0%	-	0.0%	0.0%

Table 6.13: MCTS player with random playouts versus MCTS player with weighted random playouts, 5×5.

6.3 Agents

In Tables 6.15 to 6.17, the win rates of the Monte Carlo, MCTS and negamax agents against one another on a 5×5 board are shown.

It is notable that no draws have occurred. We rated the score of a win higher than the combined score of two draws. Because of this, the Monte Carlo agents prefer moves with even odds of winning over a guaranteed draw. The negamax agent avoids draws as well if it can find an other sequence of plays that does not result in a loss.

In Tables 6.14 and 6.15, the performance of the Monte Carlo agent against the weighted random player and the MCTS agent is shown. It makes sense that the Monte Carlo agent performs better than the weighted random agent, and that the MCTS agent performs better than the Monte Carlo agent, as it is a more advanced variant of the same principle.

In Table 6.17, it becomes apparent that the negamax agent outclasses the UCT-MCTS agent. UCT works well for games such as Go where victory is defined through overall territory, and games do not end abruptly. On the other hand, the game of Tak can also be won or lost with the sudden creation of a road. Because of this, there exists a class of moves for Tak that seem favorable to the player who makes them in terms of pieces and territory, but on which the opponent can follow up with a sequence of moves that guarantees a road win. These moves are called traps if the winning response is shallow enough to be computationally detectable by the opponent.

Because the probability of a winning road strategy being executed in a random playout is very small, UCT may initially overlook the road threat in a trap. It may also find the action leading to the trap to not be promising enough, causing it to not visit the trap state enough to detect the trap. Furthermore, even if UCT visits the trap state often, the amount of effort UCT spends beyond what it needs to had it identified the winning strategy in a trap, generally increases exponentially with trap depth [RSS10]. In contrary, negamax would quickly be able to identify the winning strategy in a trap and prune it. In other words, because trap states are common in Tak and UCT has difficulties detecting traps, the MCTS agent performs poorly.

5×5	100.0%	Road	No Stones	Board Full
Weighted random wins	0.2%	0.2%	0.0%	0.0%
Monte Carlo wins	99.8%	99.8%	0.0%	0.0%
Draws	0.0%	-	0.0%	0.0%

Table 6.14: Win rate of the weighted random player against the Monte Carlo agent, 1.0 second per turn.

5×5	100.0%	Road	No Stones	Board Full
Monte Carlo wins	20.2%	20.2%	0.0%	0.0%
MCTS wins	79.8%	79.2%	0.4%	0.2%
Draws	0.0%	-	0.0%	0.0%

Table 6.15: Win rate of the Monte Carlo agent against the MCTS agent, 1.0 second per turn.

5×5	100.0%	Road	No Stones	Board Full
Monte Carlo wins	0.7%	0.6%	0.1%	0.0%
Negamax wins	99.3%	99.3%	0.0%	0.0%
Draws	0.0%	-	0.0%	0.0%

Table 6.16: Win rate of the Monte Carlo agent against the negamax agent, 1.0 second per turn.

5×5	100.0%	Road	No Stones	Board Full
MCTS wins	3.9%	3.2%	0.7%	0.0%
Negamax wins	96.1%	96.0%	0.1%	0.0%
Draws	0.0%	-	0.0%	0.0%

Table 6.17: Win rate of the MCTS agent against the negamax agent, 1.0 second per turn.

Chapter 7

Conclusions and Future Work

In this thesis we have looked at a variety of strategies for agents in playing the abstract strategy game Tak. We created simple random and weighted random players, probabilistic Monte Carlo and Monte Carlo Tree Search agents that make decisions based on random game sampling, and a negamax tree search agent that uses an evaluation function to rate moves by heuristic value. We have not only gained more understanding of the techniques and optimizations behind these algorithms, but also gained insight into the game of Tak, specifically its strategies, fairness, and complexity. Because trap states (states that lead to losses within a small number of moves) are very common in Tak, which leads to poor UCT Monte Carlo Tree Search performance, tree search algorithms are better suited for competitive agents in Tak.

For future work, it would therefore be best to focus on optimizing the negamax agent rather than the MCTS agent, for example by implementing the negascout variant of the algorithm. Additionally, its evaluation function could be updated to better detect partial roads or road threats. Lastly, it would be interesting to investigate whether the white player also has a winning strategy for larger boards.

Bibliography

[BPW+12] Cameron Browne, Edward Powley, Daniel Whitehouse, Simon Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of Monte Carlo Tree Search methods. *IEEE Transactions on Computational Intelligence and AI in Games, Vol. 4, No. 1*, pages 1–43, 2012.

[Brooo] Cameron Browne. *Hex Strategy: Making the Right Connections*. Taylor & Francis, 2000.

[Broo5] Cameron Browne. *Connection Games: Variations on a Theme*. Taylor & Francis, 2005.

[DP13] Ding-Zhu Du and Panos M. Pardalos. *Minimax and Applications*. Springer Science & Business Media, 2013.

[HURo2] H. Jaap van den Herik, Jos W.H.M. Uiterwijk, and Jack van Rijswijck. Games solved: Now and in the future. *Artificial Intelligence 134*, pages 277–311, 2002.

[KSo6] Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. *In 15th European Conference on Machine Learning (ECML-06). Number 4212 in LNCS*, pages 282–293, 2006.

[MCT17] MCTS. Monte Carlo Tree Search — Wikipedia, The Free Encyclopedia. `https://en.wikipedia.org/wiki/Monte_Carlo_tree_search`, 2017. [Accessed 24-may-2017].

[Neg17] Negamax tree search algorithm. Negamax — Wikipedia, The Free Encyclopedia. `https://en.wikipedia.org/wiki/Negamax`, 2017. [Accessed 24-may-2017].

[Pla16] Play Tak online. PlayTak. `https://www.playtak.com`, 2016. [Accessed 02-jul-2017].

[RSS10] Raghuram Ramanujan, Ashish Sabharwal, and Bart Selman. On adversarial search spaces and sampling-based planning. *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling*, pages 242–245, 2010.

[Rul16] Rules of Tak. Tak Web Rules — Cheapass Games. `http://cheapass.com/wp-content/uploads/2016/05/TakWebRules.pdf`, 2016. [Accessed 22-may-2017].

[Wik16] Wikipedia page of Tak. Tak (game) — Wikipedia, The Free Encyclopedia. `https://en.wikipedia.org/wiki/Tak_(game)`, 2016. [Accessed 22-may-2017].