



**KTH Computer Science
and Communication**

Constructing a Kamisado playing agent

DAN SETTERQUIST
DANSET@KTH.SE

PETER SKEPPSTEDT
PETERSKE@KTH.SE

DD143X Bachelor's essay
School of Computer Science and Communication
Royal Institute of Technology
Supervisor: Per Austrin
Examiner: Mårten Björkman

Abstract

Kamisado is a two player board game. This report examines the question of how to construct a competent Kamisado playing agent.

Kamisado can be played in single round matches, or in matches consisting of several rounds. The single round match was searched exhaustively. The result shows that the starting player can force a win in a single round match regardless of the strategy of the opponent.

The longer matches were too complex to be searched exhaustively. Therefore, a set of heuristic Kamisado strategies were developed and agents using different combinations of the strategies were implemented.

The different agents were compared by playing matches between them. Agents using a specific combination of the heuristic strategies tended to perform better than other agents. This indicates that this combination of the heuristic strategies creates a more competent agent than other combinations.

Referat

Skapandet av ett Kamisado-spelande datorprogram

Kamisado är brädspel för två spelare. Den här rapporten undersöker frågan om hur man konstruerar ett habilt Kamisado-spelande datorprogram.

Kamisado kan spelas i en match om en runda, eller i matcher om flera rundor. En fullständig sökning av matchen som består av en enkel runda genomfördes. Resultatet visar att spelaren som börjar alltid kan vinna en match bestående av en enkel runda oavsett motståndarens strategi.

De längre matcherna var för komplexa för att kunna sökas fullständigt. Därför utvecklades en uppsättning heuristiska Kamisado-strategier med vars hjälp olika varianter av Kamisado-spelande program implementerades.

De olika varianterna jämfördes genom att låta de olika programmen möta varandra i ett antal matcher. De varianter som presterade bättre än andra hade alla en viss kombination av de heuristiska strategierna. Detta tyder på att den kombinationen genererar ett mer välspelande datorprogram än andra kombinationer.

Contents

1	Introduction	1
1.1	Kamisado	2
1.2	Research question	2
1.3	Purpose	2
2	Background	3
2.1	The rules of Kamisado	3
2.2	Zero-sum games	5
2.3	Game tree	5
2.4	Game tree search	5
2.4.1	The minimax algorithm	5
2.4.2	Alpha-beta pruning	7
2.4.3	Iterative deepening	8
3	Single round	9
3.1	Method	9
3.2	Result	9
4	Marathon matches	13
4.1	Method	13
4.1.1	Game state features	13
4.1.2	Heuristic evaluation functions	14
4.1.3	Search depth	15
4.1.4	Regrouping	15
4.1.5	Handling sumo wins and losses	15
4.1.6	Combining the heuristic evaluation functions	16
4.1.7	Playing different agents against each other	17
4.2	Result	18
4.2.1	Results of playing single heuristics against each other	18
4.2.2	Results of playing an even weighting of two heuristics against one heuristic	18
4.2.3	Results of playing different weightings of two heuristics against an even weighting of three heuristics	18

4.2.4	Results of playing different weightings of three heuristics against an even weighting of three heuristics	19
4.2.5	Results of testing the best weightings of three heuristics . . .	19
5	Discussion	21
5.1	Single round	21
5.2	Marathon matches	21
5.3	Sources of error	22
6	Conclusions	25
7	Further research	27
	Bibliography	29
	Appendices	29
A	Additional results	31
A.1	Results of playing different weightings of two heuristics against an even weighting of three heuristics	31
A.2	Results of playing different weightings of three heuristics against an even weighting of three heuristics	33

Chapter 1

Introduction

In 1997 a milestone in the research field of game playing agents was achieved when IBM computer Deep Blue defeated the world champion Garry Kasparov in a game of chess. Since then, chess playing agents have progressed and in 2007 the computer program Rybka won against several grand masters of chess even though they were given a handicap [1, p. 193].

There are board games where the best computer programs still have not managed to beat the highest ranked humans, such as the game of Arimaa. Inspired by the Deep Blue vs Garry Kasparov match, Omar Syed created Arimaa to show that humans still can outplay computers. The current reward to the person who develops an Arimaa agent that wins the yearly challenge against the top ranked humans is \$11,000 [2].

There are several reasons why research on game playing agents is of general interest. One rather abstract reason, reflected in the previous paragraphs, is that the board game battle between man and machine can be seen as a visualization of the general progress of the computer science field. In other words, that the strength of the best game playing agents is a measure of how "intelligent" computers have become. Other perhaps more useful reasons are found in George Luger's *Artificial Intelligence* [3, p. 20-27]:

- Games can generate extremely large search spaces. These are large and complex enough to require powerful techniques for determining what alternatives to explore in the problem space.
- Much of what we commonly call intelligence seems to reside in the heuristics used by humans to solve problems.
- Artificial intelligence research have important by-products, such as advances in programming languages and software development environments.

1.1 Kamisado

Kamisado is a board game released in 2008 [4]. Similar to chess the game is played on an 8×8 grid of tiles. Each player takes turn moving one of their eight towers. The goal of the game is to get one of the towers to the other side of the board. The game can be played for just a single round, but there are extended rules in which the game is played for several consecutive rounds. After each round the winning player is awarded points and the first player to reach some predetermined number of points wins. The longest Kamisado matches, so called marathon matches, are played until one of the players reaches fifteen points.

1.2 Research question

This report revolves around the question of how to construct Kamisado playing agents capable of playing single round matches and marathon matches competently.

This is a broad question, which involves many interesting subproblems. This report will mainly be focused on finding good heuristic evaluation functions for use in the minimax algorithm (described in section 2.4.1). We will try to answer the questions of what are good heuristics when determining which player is ahead in a given Kamisado board position, and what combination of these heuristics creates the most competent Kamisado playing agent.

1.3 Purpose

The purpose of studying Kamisado is to contribute to the extensive board game research field. Virtually every new board game offers the possibility to put existing game search methods into a new context and to study how various gameplay heuristics perform in that specific game.

Hopefully this report will give insights on the game of Kamisado, and perhaps be part of insights on other existing board games or board games released in the future.

Chapter 2

Background

This chapter contains a summary of the rules of Kamisado and descriptions of common algorithms and techniques when implementing game playing agents.

2.1 The rules of Kamisado

The game of Kamisado is played on an 8×8 grid of tiles. Each player has eight different towers, one in each of the colors of the tiles (see figure 2.1). The players can move their towers straight forward and diagonally forward any number of steps, but they can not jump over other towers and they can not move backwards. The player that starts a round, *the challenger*, begins by moving any one of his/her towers. The other player, *the defender*, must then move his/her tower that has the same color as the tile the challenger's move ended on. The game then proceeds with each player moving the tower that has the same color as the tile the previous player's move ended on. A player wins by getting one of the towers to the other side of the board.

Kamisado can be played for just a single round, but the game gets more interesting when it is played for several consecutive rounds. The winner of each round is awarded points, and the first player to reach some predetermined number of points wins the game. The longest Kamisado matches, so called marathon matches, are played until one of the players reaches fifteen points. When playing Kamisado for several rounds some special rules are introduced, which are described below.

When a round has been won by some player, the winning tower (the tower that moved to the opponents home row) gets upgraded, and will for the rest of the game be a so called sumo tower. Sumos are slower but more powerful pieces. A sumo can move no more than five steps at a time, but sumos have a special move, the sumo push move. Using the sumo push move a sumo can push a tower directly in front of it one step, but only if there is a free space behind the pushed tower. The sumo moves to the tile where the pushed tower stood.

If a player wins by getting one of the sumos into the opponent's home row the sumo is upgraded to a double sumo tower. Double sumos can move no more than



Figure 2.1. The starting position of the first round of Kamisado.

three steps at a time but have a stronger sumo push. Using their sumo push move they can push two towers that are directly in front of them if there is a free space behind the last pushed tower.

If a player wins by using a double sumo it is upgraded to a triple sumo. Triple sumos can only move one step at a time, but they can push three towers using their sumo push move. Triple sumos are the final sumo, and can not be upgraded.

In all rounds except the first, the arrangement of the towers at the start of the round is determined by the arrangement of the towers at the end of the previous round. At the end of each round the winning player can choose to either "fill from the left" or "fill from the right". This choice gives the winning player two set-up options for the next round. In the next round the winning player of the previous round will be the defender.

At the end of each round the winner is awarded points. If the winning tower is a normal tower the winner is awarded one point, if it is a sumo the winner is awarded

2.2. ZERO-SUM GAMES

two points, a double sumo gives four points and a triple sumo gives eight points.

The complete rules of Kamisado can be found in the references [5].

2.2 Zero-sum games

Kamisado is a zero-sum game. A zero-sum game is a game in which any gains by a player are balanced by equally large losses by another player. The net sum of the game is therefore always zero [1, p. 161]. Kamisado is a zero-sum game since if one player wins the game, the other player loses. Many classical games, such as chess, are zero sum games [1, p. 161]. Since many of these games have been analyzed extensively, a set of good techniques for dealing with zero-sum games have been developed. One approach when developing agents for zero-sum games is to search the *game tree* of the game using the *minimax algorithm* with *alpha-beta pruning*. These concepts are described in the following sections.

2.3 Game tree

Zero-sum games can be analyzed by examining their game tree. The game tree of a game is a tree structure in which the nodes are possible states of the game and the edges represent legal moves connecting different states [1, p. 162]. By starting at the node corresponding to the current position of the game a player can try to find the best possible move by looking ahead in the game tree. When searching the game tree the player can try to analyze what will happen in the future if the player selects a particular move and then select the move that seems to lead to the best game states. The leaf nodes in the game tree represent game states in which the game has ended, either because one of the players has won the game or because the game has ended in a draw.

One of the most common ways to measure the complexity of a game is by looking at the *branching factor*. This is defined as the average number of reachable game states from any game state [3, p. 158]. It follows from the definition that the number of nodes at depth n of the game tree is roughly the branching factor to the power of n . Hence, it is hard to search deep in a game with a large branching factor.

2.4 Game tree search

2.4.1 The minimax algorithm

The minimax algorithm searches the game tree of a game and returns the best move to make from a given position. It can be used when implementing a game playing agent. The algorithm is based on the assumption that the player in turn to make a move, referred to as Max, will want to maximize the minimum value he gets from the game and that the second player, referred to as Min, will want to minimize the value Max gets [3, p. 151].

The algorithm does a depth-first search of the game tree rooted in the current game state and assigns a value to each node in this game tree. Based on these values the best move to make from the current position can be found. More specifically, the best move to make is the move to the child of the root node that has been assigned the highest value. The assigned values of all nodes depend on the values assigned to the leaf nodes. The leaf nodes, states in which the game has ended, are assigned the value $+\infty$ if it is won by Max, $-\infty$ if it is won by Min, and 0 if it is drawn. The value of each other node is calculated as follows. If the node belongs to Max it is assigned the maximum value of its children, since if Max ends up in this node he would make the move leading to the child with the largest value. If the node belongs to Min it is assigned the minimum value of its children, since if Min ends up in this node he would make the move leading to the child with the lowest value.

The algorithm as stated above has to examine every node in the game tree. This works for small games, but the game tree of most games is too large to undergo an exhaustive search in reasonable time. One way of handling this is to end the minimax search at a certain depth of the game tree, this is called *n-ply look-ahead* [3, p. 153]. One *ply* is one level in the search, so *n* represents the tree depth.

The nodes reached by a *n-ply look-ahead* at depth *n* (the terminating depth) are not necessarily leaf nodes of the game tree, and therefore not end states of the game. Because of this, $+\infty$ or $-\infty$ can not be assigned to these nodes directly as when doing the complete search, as it is not known in these nodes which player wins or loses. Therefore a *heuristic evaluation function* is used to estimate what value to assign to these nodes, based on which player seems to be ahead. When the values of the bottom nodes of the search have been estimated, the minimax algorithm can be applied as before.

The heuristic evaluation function assigns a value to a certain game state. The value is an estimation of the "certainty" that the game state will be won by one of the players. A large positive value means that the game state is most probably in favor of Max, while a large negative value means that it is most probably in favor of Min. Since Max chooses the move that leads to the game state with the highest value, an accurate heuristic evaluation function will tend to lead Max to states in which he has an advantage.

The most common way to construct heuristic evaluation functions is to calculate various features of a game state [1, p. 172]. A simple heuristic evaluation function in chess would be to assign a value to each piece (depending on the strength of the piece) and then sum up the values of the pieces on the board. By assigning positive values to Max's pieces and negative values to Min's pieces, the sum of all the values of the pieces would be an estimation of who is in the lead in a game state.

Another example of a game state feature in chess is pawn structure. The corresponding heuristic evaluation function would assign a value to a game state based on how the player's pawns are positioned.

A challenge when constructing the heuristic evaluation functions is to find the "right" values. How much is a pawn worth compared to a queen? And how much

2.4. GAME TREE SEARCH

is good pawn structure worth compared to the value of one pawn when combining these heuristic evaluation functions? According to Russel and Norvig [1, p. 173] the answers to these questions come from centuries of human chess-playing experience, or if this experience is not available, from estimations by machine learning methods.

Figure 2.2 shows a four-ply look-ahead where the value of each node has been determined with the minimax algorithm. The value of each bottom node has been estimated with a heuristic evaluation function, or been assigned $+\infty$ or $-\infty$ if it is an end state. The value of a node on level three is the maximum value of its children, since this node belongs to Max. The value of a node on level two is assigned the minimum value of its children, since this node belongs to Min. Finally, the value of the root node is the maximum value of its children, since this node belongs to Max. Since the maximum value of the children of the root node is 16, Max should make the move leading to the child with this value.

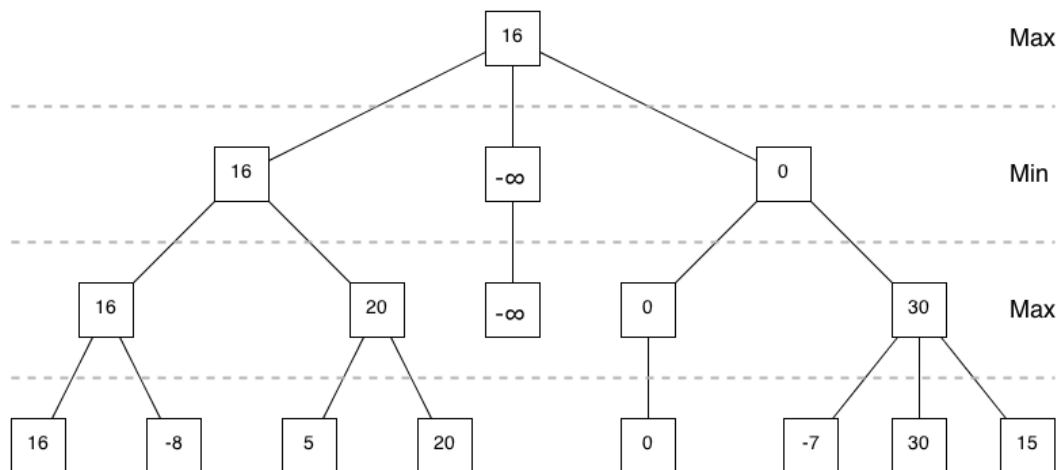


Figure 2.2. A game tree searched with the minimax algorithm.

2.4.2 Alpha-beta pruning

Alpha-beta pruning is an extension and improvement of the minimax algorithm. It is guaranteed to return the same answer as minimax, but it is faster because it evaluates fewer nodes in the game tree. Alpha-beta pruning runs like the minimax algorithm, but skips evaluating nodes in the game tree that can not affect the final result [1, p. 167].

To see that some nodes in the game tree never affect the final result and safely can be skipped when running the minimax algorithm, consider the game tree given in figure 2.2. For simplicity assume that the branches are evaluated from left to right. When the first two branches have been evaluated Max is guaranteed a score of at least 16. When the third branch is evaluated it is quickly found that Min is guaranteed a score of at least 0 in this branch, since the first subbranch evaluates to

0. Now the rest of this branch can be skipped, since it will always be better for Max to play in the first branch, where he will receive a score of at least 16, compared to the third branch, where he at most will receive a score of 0. In figure 2.3, the nodes that are skipped when using alpha-beta pruning are faded.

The name alpha-beta pruning stems from that the algorithm internally uses two values, traditionally called alpha and beta. Alpha and beta holds the lower bounds of the value that Max and Min respectively are guaranteed to get from the game. The moves that are worse than these guaranteed values are the moves that can be skipped [1, p. 168-169].

The best case performance of alpha-beta pruning can be shown to be $O(b^{d/2})$, where b is the branching factor of the game and d is the search depth, compared with $O(b^d)$ for minimax [1, p. 169].

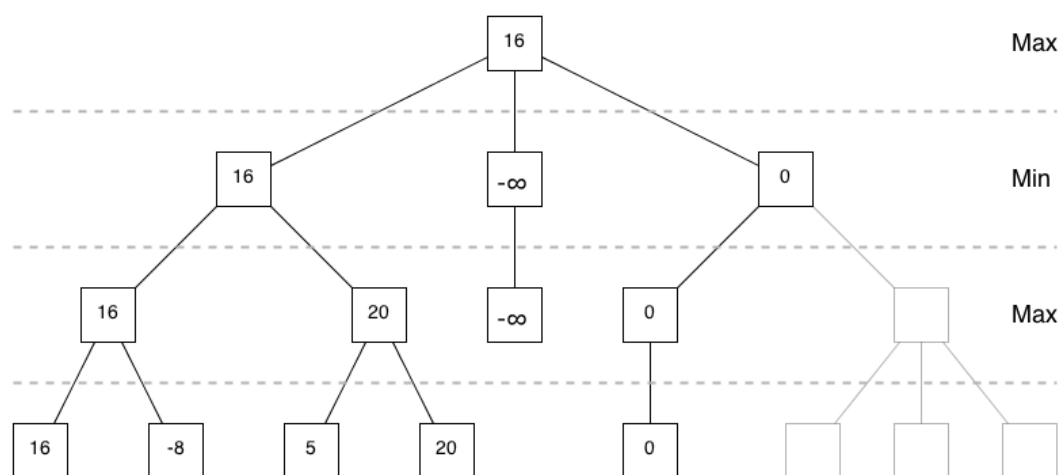


Figure 2.3. A game tree searched with the minimax algorithm using alpha-beta pruning. The faded nodes were skipped in the search.

2.4.3 Iterative deepening

Iterative deepening means running repeated n -ply look-aheads with increasing n [6, p. 142]. Iterative deepening is a good approach when searching for a guaranteed win or loss in a game tree.

For example, consider a game in which Max is guaranteed to win within seven moves if he plays optimally. A search of this game tree with a 10-ply look-ahead would find that Max is guaranteed to win. This scan would however not say that Max can win within seven moves. Using iterative deepening instead of searching at a fixed depth is therefore a better approach. The scans at depth one to six would not find the guaranteed win, but the search at depth seven would. This would then imply that Max can win in seven moves.

Chapter 3

Single round

When investigating single round Kamisado matches we found that the game tree was fairly small. Therefore we decided to do an exhaustive search of this game tree. The method and result of the search is presented in this chapter. The results of the search had implications on the construction of agents playing single round matches. These implications are discussed in section 5.1.

3.1 Method

The entire game tree of single round Kamisado was searched using the minimax algorithm with alpha beta pruning. Iterative deepening was used to determine at what depth the game was guaranteed to end for every opening move given that the players play optimally.

Because of the layout of the board and the arrangement of the towers at the start of a single round match, the initial placement is symmetric. For instance, moving the brown tower one step along the brown diagonal is equivalent to moving the orange tower one step along the orange diagonal. Because of this observation only half of the opening moves need to be searched. The opening moves of the challenger's brown, green, red and yellow towers were searched, and the results were extended to all the challenger's opening moves.

The implementation of Kamisado, the minimax algorithm, the alpha-beta pruning and iterative deepening was done in C#.

3.2 Result

Figure 3.1 displays the result of each north-east diagonal opening move. Each reachable tile is marked with an outcome and a number. This should be interpreted as if the challenger opens with the north-east diagonal move leading to this tile, the game will end with the given outcome within the given number of moves, assuming that the winning player plays optimally.

As can be seen, no diagonal opening move is a winning move, assuming that the defender plays optimally. The north-west diagonal opening moves are omitted, since they are identical for symmetrical reasons.

Orange	Blue	Purple	Pink	Yellow	Red	Green	Brown
						Loss (24)	Loss (4)
					Loss (24)	Loss (20)	Loss (16)
				Loss (16)	Loss (22)	Loss (18)	Loss (16)
			Loss (14)	Loss (18)	Loss (20)	Loss (20)	Loss (18)
		Loss (16)	Loss (14)	Loss (14)	Loss (16)	Loss (18)	Loss (4)
	Loss (14)	Loss (15)	Loss (16)	Loss (16)	Loss (14)	Loss (14)	Loss (12)
Brown	Green	Red	Yellow	Pink	Purple	Blue	Orange

Figure 3.1. The outcome of each diagonal opening move of single round Kamisado, given optimal play.

Figure 3.2 displays the result of each straight forward opening move. There are some winning straight forward opening moves. This means that the challenger is guaranteed to win the first round of Kamisado assuming that the challenger plays optimally.

3.2. RESULT

Orange	Blue	Purple	Pink	Yellow	Red	Green	Brown
Loss (4)	Win (17)	Win (21)	Loss (4)	Loss (4)	Win (21)	Win (17)	Loss (4)
Loss (6)	Win (23)	Loss (36)	Win (17)	Win (17)	Loss (36)	Win (23)	Loss (6)
Win (17)	Loss (26)	Loss (30)	Win (17)	Win (17)	Loss (30)	Loss (26)	Win (17)
Loss (24)	Loss (20)	Win (21)	Loss (24)	Loss (24)	Win (21)	Loss (20)	Loss (24)
Loss (18)	Loss (16)	Loss (30)	Loss (22)	Loss (22)	Loss (30)	Loss (16)	Loss (18)
Win (37)	Loss (18)	Loss (18)	Loss (18)	Loss (18)	Loss (18)	Loss (18)	Win (37)
↑	↑	↑	↑	↑	↑	↑	↑
Brown	Green	Red	Yellow	Pink	Purple	Blue	Orange

Figure 3.2. The outcome of each straight forward opening move of single round Kamisado, given optimal play.

Chapter 4

Marathon matches

The game tree of marathon matches is too large to search exhaustively in reasonable time. Therefore the agents developed for marathon matches had to rely on heuristic evaluation functions, as described in section 2.4.1. The method and results of developing agents playing marathon matches are presented in this chapter. The discussion of the results is found in section 5.2.

4.1 Method

To develop the marathon match playing agents, features of Kamisado game states that could indicate which player was most likely to win the game from a given position were identified. From these game state features, a set of heuristic evaluation functions was developed for use in the minimax algorithm. Agents using these heuristic evaluation functions were constructed and played against each other to determine what combination of the heuristic evaluation functions produced the best agent. The heuristic evaluation functions will sometimes simply be referred to as heuristics in the following chapters.

The implementation of Kamisado and the game playing agents was done in C#. To make the agents run faster they were implemented with alpha-beta pruning. A GUI was implemented capable of displaying Kamisado matches. The GUI also allowed human players to compete against the constructed agents.

4.1.1 Game state features

We used three different approaches to find game state features that could indicate which player had the advantage in a given Kamisado board position:

- We played the game to get a feeling for the basic strategies.
- We examined different board positions to find features indicating which player was ahead.

- We analyzed the result of the exhaustive search of the single round to try to determine what separated the winning opening moves from the losing opening moves.

The outcome of these analyses was three basic features:

- **Tower progress.** In general, it seems to be a good idea to move the towers quite far. Towers on the opponent's half of the board are more likely to be in striking position (see below), and they are also more likely to block the opponent's towers. The fact that most winning opening moves in the single round are on the opponent's half of the board is also an indication that it is good to try to position the towers there.
- **Number of striking positions.** When playing it became clear that it is good to have many towers in striking position. A tower is said to be in striking position if it can reach the opponent's home row in one move. This is perhaps the most obvious way to estimate which player is most likely to win from a certain board position. The more towers a player has in striking position, the harder it is for the opponent to move, since all moves to tiles of the same color as the towers in striking position result in a loss.
- **Number of possible moves.** Since the opponent chooses which tower you move, it seems to be a good idea to have many possible moves to choose from. Towers with many possible moves are probably less likely to run into trouble when it is their turn to move. With many options it is also more likely that there is a good move among them.

4.1.2 Heuristic evaluation functions

From the features found in section 4.1.1, three heuristic evaluation functions were constructed. Throughout the descriptions we denote by G a Kamisado game state, by A the set of towers in G that belong to Max and by B the set of towers in G that belong to Min (Max and Min are described in section 2.4.1).

- **The tower progress heuristic, f_1 ,** is the heuristic evaluation function defined as

$$f_1(G) = \sum_{a \in A} p(a) - \sum_{b \in B} p(b) \quad (4.1)$$

where $p(x)$ is the number of vertical steps tower x has progressed. It does not matter if the tower has moved straight forward or diagonally forward, $p(x)$ is only based on the number of steps on the vertical axis the tower has made.

- **The striking position heuristic, f_2 ,** is the heuristic evaluation function defined as

$$f_2(G) = \sum_{c \in C} s(c) - \sum_{d \in D} s(d) \quad (4.2)$$

4.1. METHOD

where C is the set of possible moves of the towers in A , D is the set of possible moves of the towers in B and

$$s(y) = \begin{cases} 1 & \text{if move } y \text{ is a winning move} \\ 0 & \text{otherwise} \end{cases}$$

- **The possible moves heuristic**, f_3 , is the heuristic evaluation function defined as

$$f_3(G) = \sum_{a \in A} m(a) - \sum_{b \in B} m(b) \quad (4.3)$$

where $m(x)$ is the number of possible moves of tower x .

4.1.3 Search depth

Since the main focus of this report is the heuristic evaluation functions it was decided that all agents we tested should use the same game tree search depth. Choosing the search depth was a tradeoff between the time limitations of this project and accuracy of the results. On the one hand we would not be able to run enough matches between the different agents if we searched the game tree too deep, on the other hand it is doubtful that the results would be accurate with a too shallow search depth, since the idea behind the minimax algorithm is to search at least a few plies deep. Based on these factors it was decided that the agents should use 5-ply look-ahead.

4.1.4 Regrouping

At the end of each round of a marathon match, the winning player decides how to regroup the towers as described in section 2.1. In this report we do not pay much attention to that decision. However, we did not want to add uncertainty to the results by letting the agents make the regroup decision by random. Therefore, all the different agents make the regroup decision in the same way. First they find the best opening move of the two possible regroupings (left and right) with the minimax algorithm, searching at depth five using the tower progress heuristic. Then they choose the regrouping with the lowest estimated value of the best opening move, since the opponent of the agent that makes this decision will start the next round.

4.1.5 Handling sumo wins and losses

As described in section 2.4.1, winning game states should be assigned the value of $+\infty$. In a marathon match there is, however, a difference between winning game states, since the points awarded to the winning player depends on the sumo rank of the winning tower. This issue was handled by giving the winning game states values larger than any possible evaluation of a non-winning game state, but proportional to the sumo rank. A game state won by a normal tower was assigned a value larger

than any non-winning game state, but a smaller value than a game state won by a level one sumo, and so on. This way the agents preferred to win with a sumo tower over a normal tower, and a double sumo tower over a sumo tower, and so on. Losing game states were handled similarly, game states lost by a normal tower were assigned values smaller than any non losing game states, but larger than any game states lost by a level one sumo tower, and so on.

4.1.6 Combining the heuristic evaluation functions

The basic idea when constructing agents combining the three heuristics (4.1) - (4.3) defined in section 4.1.2, was to rank the possible moves with the heuristic evaluation functions and elect the best possible move by *range voting* [7, p. 27]. This was done in the following way:

For a player with n possible moves (y_1, y_2, \dots, y_n) , three vectors were calculated:

$$\mathbf{v}_i = (g_i(y_1), g_i(y_2), \dots, g_i(y_n)) \quad i = 1, 2, 3 \quad (4.4)$$

where $g_i(y_j)$ is the minimax value of move y_j using heuristic f_i in the minimax algorithm.

Then a ranking function, $r(\mathbf{v})$, was used to transform the values of \mathbf{v}_i to a ranking by replacing the lowest value of \mathbf{v}_i with 1, the second lowest value with 2 and so on. If two or more values of \mathbf{v}_i were equal, they were replaced with the same ranking, d , and the next higher value of \mathbf{v}_i was replaced with the next consecutive rank, $d + 1$.

The integer on index k in vector $r(\mathbf{v}_i)$ represents the ranking of move y_k based on the minimax value of y_k using evaluation function f_i . Hence, the three vectors, $r(\mathbf{v}_1)$, $r(\mathbf{v}_2)$ and $r(\mathbf{v}_3)$, were the basis for the range voting.

Range voting was used for simplicity reasons, since the winning candidate of a range voting is simply the candidate with the highest average rank. This allowed for the use of a weight vector

$$\mathbf{w} = (w_1 \ w_2 \ w_3) \quad (4.5)$$

to give the three evaluation functions different amounts of influence in the range voting. Introducing the ranking matrix

$$R = \begin{pmatrix} r(\mathbf{v}_1) \\ r(\mathbf{v}_2) \\ r(\mathbf{v}_3) \end{pmatrix} \quad (4.6)$$

the weighted range voting vector, \mathbf{u} , was calculated by

$$\mathbf{u} = \mathbf{w}R \quad (4.7)$$

Finally, the index l of the highest value of \mathbf{u} was used to find the best move y_l . Whenever there were more than one highest value in \mathbf{u} a random choice was made

4.1. METHOD

between the best moves. This made the agents non-deterministic, which was preferable since a deterministic agent can easily be beaten once a winning strategy against it has been found. Deterministic agents would also make it meaningless to play several matches between two agents, since all the matches would be exactly the same.

A benefit of the range voting method was that it was very straightforward to combine the heuristics. For instance, when combining the striking position heuristic with the tower progress heuristic it was easy to put different weights on the range voting rankings and to interpret the meaning of these weights. As an example, using weight vector $\mathbf{w} = (1 \ 2 \ 0)$ in the calculations above would mean that the rankings of the striking position heuristic is twice as important as the rankings of the tower progress heuristic and that the rankings of the possible moves heuristic is left out completely. Note that an agent using weight vector $(1 \ 0 \ 0)$, $(0 \ 1 \ 0)$ or $(0 \ 0 \ 1)$ is equivalent to an agent using a single heuristic without range voting.

4.1.7 Playing different agents against each other

To find the combination of the heuristics that produces the best agent, agents using different weight vectors were played against each other in 40 marathon matches. The agents opened 20 of the matches each.

One problem with playing different agents against each other to find the best combination of the heuristics is the sheer number of matches that need to be played when testing every agent against every other agent. Combining three heuristics with four different weights for every heuristic produces $4^3 = 64$ different agents. Playing each pair against each other would produce $\binom{64}{2} = 2016$ match-ups. This was not possible due to time constraints. Instead, the following procedure was adhered to:

1. First all combinations of agents using only one heuristic were played against each other.
2. Then all agents using only one heuristic were played against all agents using an even weighting of two heuristics.
3. Then agents using various combinations of two heuristics were played against an agent using an even weighting of all three heuristics.
4. Step 3 produced a set of agents that played best against the agent using an even weighting of all three heuristics. These were investigated further. Various weights of the unused heuristic were added to these agents and they were played against an agent using an even weighting of all three heuristics.
5. Step 4 produced a set of agents that performed best. These were played against the best agents using two heuristics, found in step 3. Finally the best agents were also tested against each other.

4.2 Result

The tables in this section should be interpreted as follows: the cell in row i and column j holds the result of the matches played between player i and player j . The first number is the number of matches player i won and the second number is the number of matches j won. The agents are described by their weight vector used in the range voting. A weight vector $(a\ b\ c)$ should be interpreted as an agent using the tower progress heuristic with weight a , the striking position heuristic with weight b and the possible moves heuristic with weight c .

4.2.1 Results of playing single heuristics against each other

Table 4.1 shows the results of playing agents using one heuristic against each other. As can be seen, none of the three heuristics stand out as clearly better than the others.

Table 4.1. Results when playing single heuristics against each other.

	(1 0 0)	(0 1 0)	(0 0 1)
(1 0 0)	-	27 - 13	19 - 21
(0 1 0)	13 - 27	-	27 - 13
(0 0 1)	21 - 19	13 - 27	-

4.2.2 Results of playing an even weighting of two heuristics against one heuristic

Table 4.2 shows the results of playing agents using an even weighting of two heuristics against agents playing a single heuristic. The results are a good indication of that combining two heuristics is better than using a single heuristic.

Table 4.2. Results when playing an even weighting of two heuristics against one heuristic.

	(1 0 0)	(0 1 0)	(0 0 1)
(1 1 0)	26 - 14	27 - 13	33 - 7
(1 0 1)	34 - 6	28 - 12	36 - 4
(0 1 1)	25 - 15	33 - 7	34 - 6

4.2.3 Results of playing different weightings of two heuristics against an even weighting of three heuristics

Agents using different weightings of two heuristics were played against an agent using all three heuristics with an even weighting. None of the agents using two heuristics won a majority of the matches but some were fairly close. The agents

4.2. RESULT

using two heuristics that performed best were (1 2.5 0), (1 2 0), (1 1 0), (0 2.5 1), (0 3 1) and (0 3.5 1). The complete results of these match-ups are found in section A.1 of Appendix A.

4.2.4 Results of playing different weightings of three heuristics against an even weighting of three heuristics

Different weightings of the unused heuristic were added to the best weightings of two heuristics. Agents using these weightings were played against an agent using an even weighting of the three heuristics. The agents that performed best were (2 2.5 1), (1.5 2.5 1) and (1.5 3.5 1). The complete results of these match-ups are found in section A.2 of Appendix A.

4.2.5 Results of testing the best weightings of three heuristics

Table 4.3 shows the results of playing agents using the best weightings of three heuristics against agents using the best weightings of two heuristics.

Table 4.3. Results when playing the best weightings of the three heuristics against the best weightings of two heuristics

	(2 2.5 1)	(1.5 2.5 1)	(1.5 3.5 1)
(1 1 0)	13 - 27	10 - 30	12 - 28
(1 2 0)	18 - 22	15 - 25	16 - 24
(1 2.5 0)	19 - 21	21 - 19	15 - 25
(0 2.5 1)	11 - 29	9 - 31	10 - 30
(0 3 1)	14 - 26	10 - 30	8 - 32
(0 3.5 1)	12 - 28	12 - 28	13 - 27

Table 4.4 shows the results of playing agents using the best weightings of the three heuristics against each other. As can be seen, none of the weightings stand out as clearly better than the others.

Table 4.4. Results when playing the best weightings of the three heuristics against each other.

	(2 2.5 1)	(1.5 2.5 1)	(1.5 3.5 1)
(2 2.5 1)	-	19 - 21	22 - 18
(1.5 2.5 1)	21 - 19	-	26 - 14
(1.5 3.5 1)	18 - 22	14 - 26	-

Chapter 5

Discussion

5.1 Single round

The exhaustive search of the single round shows that the challenger can win in at most 17 moves, regardless of the strategy of the defender. An agent that always wins single round matches as the challenger could be implemented by storing a winning opening move along with some best responses to the defender's moves in a lookup table. When the lookup table has been exhausted the agent can be set to search the game tree at a sufficient depth to be able to spot the win. This is feasible as we found that it only took a couple of minutes to search a particular move for a win at depth 17 with the minimax algorithm using alpha-beta pruning.

The result of the exhaustive search implies that it is not possible to construct an agent that wins single round matches as the defender when the challenger plays optimally. When the challenger plays suboptimally however, the same techniques that works well when playing marathon matches should work for single round matches, as the first round in a marathon match is a single round match.

5.2 Marathon matches

Table 4.1 shows that none of the three developed heuristics stands out as clearly better than the others. From these results it is not possible to draw any conclusions on how accurate the individual heuristics are. We did however play matches against agents using only one heuristic to estimate their skill level. All three agents played surprisingly well, they were at or above our skill level.

Table 4.2 shows that combining two of the heuristics is most likely better than just using one, and table 4.3 indicates that the best combinations of all three heuristics are better than the best combinations of two heuristics. This suggests that all the heuristics contribute to the skill level of the agent. The result that the best agents used all three heuristics is also an indication that range voting is a viable method when combining several heuristics. The combined value that the range voting produces seems to be a more accurate evaluation than what any one of the

heuristics can produce on its own.

It seems likely that the weights of the heuristics can make a significant difference in the range voting. For instance, the agents (0 1 2.5) and (0 2.5 1) performed very differently when playing against (1 1 1). These two agents use the same heuristics but with different weights, so the difference in performance most likely stems from the different weights.

From the results, it is hard to draw any clear conclusion of which exact combination of the heuristics is the best. But when looking at the top three agents, (2 2.5 1), (1.5 2.5 1) and (1.5 3.5 1), it can be noted that they are all fairly similar. They all put most weight on the striking position heuristic, second most weight on the tower progress heuristic and least weight on the possible moves heuristic. This suggests that of all the investigated combinations of heuristics, this type creates the most competent Kamisado playing agent. Putting most weight on the striking position heuristic seems reasonable since having many pieces in striking position gives a player a more direct way of winning. Having many possible moves or good tower progress may be important, but does not necessarily mean that there are any direct ways to win.

5.3 Sources of error

Although the results are fairly consistent, there are areas of potential problems in the chosen method. Some of these are described in this section.

One main problem with the chosen method is that the performance of an agent is determined by playing it against other similar agents. When developing a game-playing agent one usually strives to make the agent good against human players. Since the agent developed in this report has not been tested against any expert Kamisado players, it is hard to know how well it would perform in that situation. The chosen method might only produce the agent that is best when playing against variants of itself, the developed agent might not generalize well to also be good against other types of players.

Although this is a problem, there are some indications that the developed agent is a decent Kamisado player. The authors had not played Kamisado before starting this project, but during it we have played a number of matches to get a feel for the game. Even so the developed agent was capable of easily defeating us, which is an indication that it is playing above novice level.

Another small indication that the agent is decent is that when playing as the challenger in the single round, it will open with one of the winning moves. This means that the chosen heuristics are able to separate this move from the losing moves.

Because of time constraints, not all of the different agents could be tested against each other, and not all possible combinations of weights could be tested. For instance, the combinations of three heuristics that were tested were based on the best combinations of two heuristics. But it is not necessarily true that weights that were

5.3. SOURCES OF ERROR

best when testing two heuristics also are best when testing three. Given more time, more of the possible combinations of the three heuristics could have been tested.

When choosing which combinations of two heuristics to base the combinations of three heuristics on, the six combinations of two heuristics that performed best against (1 1 1) were chosen. There were some other combinations of two heuristics that essentially performed equally well to the ones chosen, but because of time constraints these could not be tested. Given more time other combinations of two heuristics could have been extended to use all three heuristics. Similarly, when the three best agents using three heuristics were chosen, there were other agents using three heuristics that essentially performed equally well. These could have been tested given more time.

When doing the testing all of the agents used 5-ply look-ahead. It is therefore hard to know if the heuristics and weights that were found to be good will also be good when using other search depths. It is possible that a different weighting is superior when searching at other depths.

Chapter 6

Conclusions

The challenger is guaranteed to win single round matches of Kamisado in 17 moves when playing optimally. Developing an agent that always wins single round matches when playing as the challenger is therefore not hard.

All of the heuristics found in this report, the tower progress heuristic, the striking position heuristic and the possible moves heuristic, can to some extent indicate which player is ahead in a given Kamisado board position.

In general, agents using combinations of all three heuristics outperformed agents using only one or two of the heuristics. This indicates that range voting is a viable method when combining several heuristics.

The agents that played marathon Kamisado matches most competently were the agents that put most weight on the striking position heuristic, second most weight on the tower progress heuristic and least weight on the possible moves heuristic. It is hard to say how good these agents are when playing against skilled human players, but there are indications that they play above novice level.

Chapter 7

Further research

One obvious area of further research is to try to find other heuristics that accurately estimates the value of a Kamisado board position. For instance, this report included no heuristic specifically targeting the sumo towers. The sumo towers become more important as the game progresses, and although they have similarities with ordinary towers, they also have differences that might require specialized heuristics.

The choice when regrouping the towers at the end of a round was not investigated in this report. All the agents tested behaved the same when regrouping. But this choice might be very important, and could have consequences for the rest of the game. Therefore it may be an important part of developing a Kamisado playing agent, and should be investigated. Accurately choosing the best way to regroup might require completely separate heuristics.

In this report, range voting was used when combining several heuristics. Range voting is quick and easy to implement, but there might be a way to combine the heuristics that would produce a better agent. Comparing range voting with other ways to combine heuristics would be interesting on its own.

Bibliography

- [1] Russel S, Norvig P. Artificial Intelligence: a modern approach. 3rd ed. Prentice hall; 2009.
- [2] Arimaa;. [Online; accessed 05-March-2013]. <http://http://www.arimaa.com/arimaa/>.
- [3] Luger GF. Artificial intelligence: Structures and Strategies for Complex Problem Solving. 5th ed. Pearson Education; 2005.
- [4] Kamisado;. [Online; accessed 04-April-2013]. <http://www.burleygames.com/board-games/kamisado/>.
- [5] Burley P. Kamisado Rules of the Game; 2008. [Online; accessed 03-April-2013]. http://www.huchandfriends.de/page/en/Downloads/Spielanleitungen/Strategiespiele/Kamisado_engl.pdf.
- [6] Dean TL, Allen J, Aloimonos J. Artificial intelligence: Theory and practice. Benjamin/Cummings Pub. Co.; 1995.
- [7] Felsenthal DS. Electoral Systems: Paradoxes, Assumptions and Procedures. Springer; 2012.

Appendix A

Additional results

A.1 Results of playing different weightings of two heuristics against an even weighting of three heuristics

Table A.1 shows the results of playing agents of the form $(1\ a\ 0)$, $a \in \{1, 1.5, 2, 2.5, 3, 3.5\}$ and $(b\ 1\ 0)$, $b \in \{1, 1.5, 2, 2.5\}$ against an agent using $(1\ 1\ 1)$. Or more simply, agents using different combinations of the tower progress and striking position heuristics against an agent using the three heuristics with equal weights. Note that we let a range over more values as the results in that direction were not as clear cut as in the other direction. The results suggest that the agent using $(1\ 1\ 1)$ is slightly better than all the agents using two heuristics with various weights.

Table A.1. Results when playing different weightings of the tower progress heuristic and the striking position heuristic against an even weighting of three heuristics.

	$(1\ 1\ 1)$
$(1\ 3.5\ 0)$	13 - 27
$(1\ 3\ 0)$	14 - 26
$(1\ 2.5\ 0)$	17 - 23
$(1\ 2\ 0)$	17 - 23
$(1\ 1.5\ 0)$	13 - 27
$(1\ 1\ 0)$	17 - 23
$(1.5\ 1\ 0)$	15 - 25
$(2\ 1\ 0)$	12 - 28
$(2.5\ 1\ 0)$	11 - 29

Table A.2 shows the results of playing agents of the form $(1\ 0\ a)$ and $(a\ 0\ 1)$, $a \in \{1, 1.5, 2, 2.5\}$ against an agent using $(1\ 1\ 1)$. The results indicate that the agent using $(1\ 1\ 1)$ is slightly ahead in all of these match-ups.

Table A.3 shows the results of playing agents of the form $(0\ 1\ a)$, $a \in \{1, 1.5, 2, 2.5\}$ and $(0\ b\ 1)$, $b \in \{1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5\}$ against an agent using $(1\ 1\ 1)$. We let

APPENDIX A. ADDITIONAL RESULTS

Table A.2. Results when playing different weightings of the tower progress heuristic and the possible moves heuristic against an even weighting of three heuristics.

	(1 1 1)
(1 0 2.5)	5 - 35
(1 0 2)	15 - 25
(1 0 1.5)	13 - 27
(1 0 1)	15 - 25
(1.5 0 1)	15 - 25
(2 0 1)	14 - 26
(2.5 0 1)	8 - 32

b range over more values as the results in that direction were not as clear cut as in the other direction. The results suggest that the agent using (1 1 1) is slightly ahead in all of these match-ups.

Table A.3. Results when playing different weightings of the striking position heuristic and the possible moves heuristic against an even weighting of three heuristics.

	(1 1 1)
(0 1 2.5)	4 - 36
(0 1 2)	13 - 27
(0 1 1.5)	7 - 33
(0 1 1)	10 - 30
(0 1.5 1)	15 - 25
(0 2 1)	12 - 28
(0 2.5 1)	17 - 23
(0 3 1)	18 - 22
(0 3.5 1)	17 - 23
(0 4 1)	16 - 24
(0 4.5 1)	16 - 24

A.2. RESULTS OF PLAYING DIFFERENT WEIGHTINGS OF THREE HEURISTICS AGAINST AN EVEN WEIGHTING OF THREE HEURISTICS

A.2 Results of playing different weightings of three heuristics against an even weighting of three heuristics

Table A.4. Results when playing agents of the form $(a \ 2.5 \ 1)$, $a \in \{0.5, 1, 1.5, 2\}$, against an agent using $(1 \ 1 \ 1)$.

	$(1 \ 1 \ 1)$
$(0.5 \ 2.5 \ 1)$	21 - 19
$(1 \ 2.5 \ 1)$	18 - 22
$(1.5 \ 2.5 \ 1)$	25 - 15
$(2 \ 2.5 \ 1)$	24 - 16

Table A.5. Results when playing agents of the form $(a \ 3 \ 1)$, $a \in \{0.5, 1, 1.5, 2\}$, against an agent using $(1 \ 1 \ 1)$.

	$(1 \ 1 \ 1)$
$(0.5 \ 3 \ 1)$	23 - 17
$(1 \ 3 \ 1)$	18 - 22
$(1.5 \ 3 \ 1)$	17 - 23
$(2 \ 3 \ 1)$	21 - 19

Table A.6. Results when playing agents of the form $(a \ 3.5 \ 1)$, $a \in \{0.5, 1, 1.5, 2\}$, against an agent using $(1 \ 1 \ 1)$.

	$(1 \ 1 \ 1)$
$(0.5 \ 3.5 \ 1)$	19 - 21
$(1 \ 3.5 \ 1)$	19 - 21
$(1.5 \ 3.5 \ 1)$	26 - 14
$(2 \ 3.5 \ 1)$	23 - 17

Table A.7. Results when playing agents of the form $(1 \ 1 \ a)$, $a \in \{0.5, 1, 1.5, 2\}$, against an agent using $(1 \ 1 \ 1)$.

	$(1 \ 1 \ 1)$
$(1 \ 1 \ 0.5)$	23 - 17
$(1 \ 1 \ 1)$	21 - 19
$(1 \ 1 \ 1.5)$	21 - 19
$(1 \ 1 \ 2)$	14 - 26

APPENDIX A. ADDITIONAL RESULTS

Table A.8. Results when playing agents of the form $(1\ 2\ a)$, $a \in \{0.5, 1, 1.5, 2\}$, against an agent using $(1\ 1\ 1)$.

	$(1\ 1\ 1)$
$(1\ 2\ 0.5)$	16 - 24
$(1\ 2\ 1)$	21 - 19
$(1\ 2\ 1.5)$	14 - 26
$(1\ 2\ 2)$	21 - 19

Table A.9. Results when playing agents of the form $(1\ 2.5\ a)$, $a \in \{0.5, 1, 1.5, 2\}$, against an agent using $(1\ 1\ 1)$.

	$(1\ 1\ 1)$
$(1\ 2.5\ 0.5)$	22 - 18
$(1\ 2.5\ 1)$	23 - 17
$(1\ 2.5\ 1.5)$	20 - 20
$(1\ 2.5\ 2)$	19 - 21