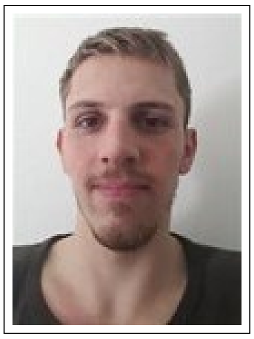
Board Game Assignment

Introduction to Artificial Intelligence



Troels Lund
DTU Diplom
s161791

Mads Stege
DTU Compute
s165243

Niklas Thielemann
DTU Diplom
s145032

Technical University of Denmark
DTU Compute

Friday, the 13th of April, 2020 - 20:00

Number of pages (Including appendix): 10

Courses: 02180

Content

Contributions

During this project, the entirety of the group has either been directly responsible for a given matter, or been on the sidelines as a valuable discussion partner. With that being said, the primary contributors of the Report are as follows:

Section name	**Section number**	**Responsible**
Game Rules	1	Mads
Chosen Game	2	Mads
State Space	3	Niklas
Game Elements	4	Niklas
Game Representation	5	Niklas
Methods & Algorithms	6	Troels
Heuristics or Evaluation Functions	7	Troels
Parameter Adjustment	8	Mads
Comments & Conclusion	9	Everyone

(Press a section Name or number to navigate to it directly.)

Everyone in the group has contributed to the overall code base in equal measures.

0.1 GitHub Repository

The project can be viewed online at:

https://github.com/trolund/KalahaAI

1 Game Rules

For the purpose of this report, the game will employ the standard Kalaha rules, as described in the official Endless Games instruction manual (2015) handed out on the 2nd of March, 2019.

The implemented board in game will follow a (6,4) setup. The notation, (6,4), describes the initial game state in terms of number of pits for each player and the starting number of stones in each pit.

There will be no changes, alterations, or simplifications to the rules and game setup, as described from the official guide[1].

2 Chosen Game

The classic Kalaha game, sometimes referred to as "Mancala" depending on the region, is an activity involving two players in a competitive non-zero-sum game setting. Kalaha is a turn-based game, where both players have perfect information (full observability) of the game's current state, as shown in figure 1.

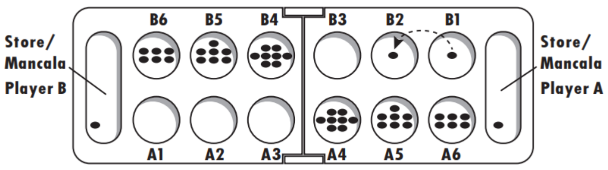


Figure 1. Example of a Kalaha game. Source: Endless Games (2015)

Due to the nature of Kalaha, it is a fully deterministic game. Both players may view the outcome of all game states following the currently active game state. There are no random elements affecting the course of the game. As such, the AI operating will therefore implement an algorithm seeking out the best suited states first. Minimax with Alpha-Beta pruning would be a likely candidate, to help quickly identify the most promising state-node.

It is worth mentioning that Kalaha is solved game, i.e. it is possible to predict the game's outcome from the initial state, and who the first player is, assuming both players play perfectly.

3 State Space

The state space is determined by factors, such as number of pits in the game, the number of stones and which player's turn it is. In the standard version of Kalaha, there are 12 pits in total and two Kalahas. There are 48 pieces, and there are only two players. If the number of pits and Kalahas are defined as 'k', the number of stones defined as 's', then the number of state spaces excluding the players, would be s^k, since any given number of pieces can be at any given pit/store. The upper bound game state space including players would then be $s^k * p$ where p are the number of players.

For the standard version of Kalaha, that would give $48^{14} * 2$ which is roughly $6,8 * 10^{23}$ number of state spaces.

However, most of these state spaces are unreachable from the initial state due to the rules of the game. Since the stones are evenly split among the pits and players are forced to place stones in their own Kalahas when passing by, suggests that a pit at most can hold half the possible stones. In that case, the state space complexity would be calculated as:

$$s^k * (s/2)^h * p$$

where

$$s = stones, k = kalahas, h = pits \text{ and } p = players$$

which when calculated equals to (roughly) $1,7 * 10^{20}$ state spaces

Considering that other rules have not been taken into account, making it harder to stack stones in pits, implies that the game complexity is even smaller than that. From third party works[2], they estimate the state space to be in the region of $1.31 * 10^{13}$ reachable states, suggesting we may yet still include a number of unreachable states.

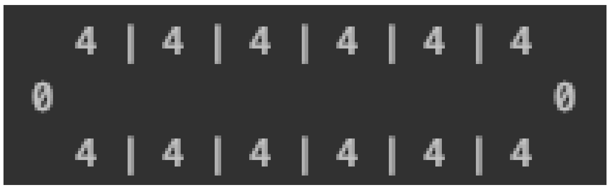
4 Game Elements

The various game elements used in the project is described as below:

- **s0 (Initial state)** is the *initial state*, the first state presented to the system. This is depicted in figure 2. The numbers on the right and left side are the players amount of points, both of which initially are set to zero. Each pit in the game contains four stones each in s0.
- **Player(s)** returns which player's turn it is.
- **Actions(s)** returns which pits a given player can select for pick up. A pit is only available for pick up if its amount of stones are greater than zero.
- **Results(s, a)** will return a state where each pit has been updated, e.g. if the current state is s0 and the left most southern pit has been selected, then that pit will have zero and the following four pits will have five stones in the resulting state.
- **Terminal-Test(s)** returns true if one of the player's side are completely empty.
- **Eval_win_loss(s, p)** defines a value of the state s for a player p. The value depends on the outcome which can either be a win, loss or draw. The player with the most stones wins.
- **Eval_max_dif(s, p)** returns a value stating how good the current state s is for player p. For a max player, the higher result given by the evaluation functions, the better.
- **Eval_sum(s, p)** returns a value stating how good one player's boardside is over the other's.

5 Game Representation

Game state is represented as an array that contains the values of each pit's amount of stones and the Kalahas as well. It also contains a boolean indicating which player's turn it is.



```
  4 | 4 | 4 | 4 | 4 | 4
0                         0
  4 | 4 | 4 | 4 | 4 | 4
```

Figure 2. A representation of the game's initial state

Actions are represented as numbers, where a specific action's value equals to the chosen pit. If a player wants to move the pieces from pit number 4, the player will choose $action = 3$[1].

Since we have implemented the rule that gives the possibility for player to get an extra turn, it is important for the A.I to know whether it has reached a min or max node in it's pathing. By default the nodes would shift between a min and max node, but since a player can get an extra turn, it is possible to reach two or more min/max nodes in a row.

If we had discarded that specific rule, the game state could simply have been an array without the boolean attached to it.

6 Methods & Algorithms

We have chosen to implement a greedy Minimax-based algorithm with alpha-beta pruning for the agent. The alpha-beta pruning allows the Minimax algorithm to exclude less promising paths, allowing for a faster exploration of more promising paths.

Since the Minimax algorithm builds a partial game tree, it is necessary to incorporate a max depth parameter, as it is impractical to do the complete tree search every turn. This especially holds true, because the program does not utilities the power of multithreading.

The current implementation of the Minimax algorithm has one notable difference. That is the game state tree does not necessarily contain layers of alternately max and min layer. In Kalaha, you can earn an extra turn, i.e. the max player having two turns in a row will result in two layers of maximising the utility value.

One example of the Minimax algorithm's thought process can be seen in section 10.1, *Gametree model.*

As shown in the gametree, the max agent (red) will chose the highest value. Meanwhile, the min agent (blue) will chose the lowest evaluation value. Since every node potentially has six children, the tree grows incredibly large, very fast. Therefore, only a small section of a few selected nodes are drawn. The boldened path shown is the path the agent at this point will try following, by executing Action 9.

In addition, it is worth mentioning that an algorithm like Monte Carlo tree search could also have been applied to Kalaha, in which case the evaluation function would be defined by a stochastic function based on random wins and losses, called Monte Carlo simulation.

[1]The array is 0-indexed.

While Minimax is capable of finding the most valid path, the Monte Carlo tree search algorithm would have made for a more capable agent.

7 Heuristics or Evaluation Functions

For our game, we have considered a couple of heuristics.They are simple in nature, because the objective of the game is to have more points in your Kalaha than the opponent's Kalaha.

7.1 Winner Or Loser Heuristic

The AI has only one heuristic point, that only checks to see if the player is winning - a simple check whether it has more points than the opponent.

The evaluation of game states therefore becomes binary, because there is only winning or losing. This is despite the fact that there is also a possibility of the players having the same amount of points.

The evaluation function is therefore implemented so that it will return '1' in every state that it got more points than the opponent, '-1' in a losing state and '0.5' when the scores are even.

This also means that the AI will choose to move to the node with the most successful sub tree, i.e. the tree with the most terminal states where it will win. This is regardless of the number of steps taken or the difference in points between the players.

7.2 Maximise Point Difference Heuristic

Another simple approach is to look at the difference between the opponents points and the players own points so that the score will be as high as possible for one player and as low as possible for the opposing player.

This guides the AI towards maximising the amount it wins with, thereby looking like a more competent agent.

The evaluation function would look like so:

$$f(n) = player^{1}_{points} - player^{2}_{points}$$

This approach appears superior, to the Winner Or Loser Heuristic.

7.3 Sum Difference Heuristic

Finally, we have a heuristic, where it takes the sum of one side of the board associated with the player in action, and subtracts it from the opposing side of the board.

This lets the AI more easily determine whether or not it is heading towards a losing node, letting it work preemptively to stop this.

7.4 Evaluation of heuristics

We have tried to evaluate different heuristics by letting them play 1,000 times against a random agent, an agent selecting random pits to choose from. The starting player of the game is equally divided, giving each of them 500 games to play first, and 500 games to play second. The results are as follows:

	Winner/loser	Maximise score	Sum difference
Win%	75,4	97,5	95,1

As shown, the Maximise point difference heuristic is the best performing, winning 97,5% of its games. For that reason, this is the heuristic evaluation chosen. None the less, the suggested heuristics are of some level of competence, as they all perform noticeably better than a completely random agent.

It could be beneficial to use step count as a factor in the evaluation function to increase AI awareness of the amount of steps needed to arrive by a given state.

Further more, combining heuristics with different focus areas could give the agent a better change of using the game mechanics to its advantage.

8 Parameter Adjustment

Due to the nature of the implemented algorithms (see section 6, Methods & Algorithms), there are a number of variables one might tweak to change the behaviour and proficiency of the AI. As a baseline, the AI interprets moves which puts its own score above its opponent as favourable. Thus, the variables one might tweak is:

- **Algorithm variation**
  The implemented algorithm comes in two varieties, with or without alpha-beta pruning. While they both might arrive at the same path with the same level of depth, the algorithm with alpha-beta will arrive within a much shorter time frame.
- **Search depth**
  The deeper the AI goes into its search tree nodes, the more likely it is to find a favourable path of actions.
- **Initial turn**
  Whosoever starts the game has an monumental advantage over their opponent. In fact, it is mathematically impossible for the first player to lose a game of Kalaha[2]. In a 6-pit setup[3], no matter the initial amount of stones in the various pits, the player with the initial draw wins.
- **Value of paths**
  The algorithm considers a number of paths available to it, choosing the path returning the highest amount of stones in its Kalaha. However, a more lenient AI opponent will choose a less favourable path (while still working in its own interest). Implementing such

[2]Assuming, of course, they play optimally.

an algorithm. Interestingly, this "flawed" decision making will also make the AI seem more approachable, and human-like, instead of an always-correct machine.

8.1 Benchmarks

For the purpose of highlighting the various differences changing the parameters described, benchmarks has been logged and ordered. Unless otherwise specified, the AI is set to run the Minimax algorithm **with** alpha-beta pruning, with a search depth of 4.

8.1.1 Algorithm variation

The Minimax algorithm and its alpha-beta variation offers no difference in terms of the quality of its output, as both the original Minimax and Alpha-Beta performed identically in terms of score. The time between each AI's move was drastically different however.

We put two AI's up against one another, with the only difference being their algorithm implementation.

How much faster is Alpha-Beta pruning? (ms)			
Depth level:	Minimax	Alpha-beta	Percentile difference
1	5,84	4,35	25,51%
2	15,57	7,0	55,04%
3	71,42	24,57	65,59%
4	256,0	70,27	72,55%
5	820,0	158,26	80,69%
6	4211,05	537,0	87,24%
7	17322,14	1393,07	91,95%

Table 1: Table showcasing the percentile difference between Minimax and Minimax with Alpha-Beta pruning.

Table 1 highlights the performance difference between the two algorithms. Note that the times are averages over the course of one game. This means that as the game goes one, certain pits become empty of stones, and thus can be immediately disqualified from the list of available pits.

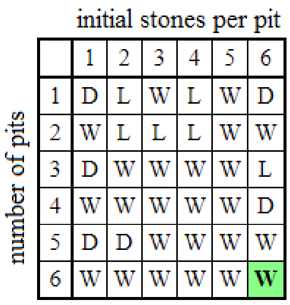
8.1.2 Search depth

Changing the search depth is one of the simplest methods of changing the AI's behaviour. For a search depth of 1, the AI will only look one step ahead for its options. While it will correctly scrap the paths that will not be beneficial and pick a path where it gets a point, it might not be the *best* path. The deeper the search, the better the AI is able to discern which paths return the best score

This is the key difference between a search depth of 1 and, say, 4. We were not able to generate a game in which the AI with a search depth of 4 lost a game, as it would always choose the much more beneficial path.

8.1.3 Initial turn

As previously mentioned, whichever party starts has the option of winning if they perform perfectly. This hypothesis is confirmed in the works of Carstensen and Larsen[3] and Irving, Donkers and Uiterwijk[2], and highlighted in figure 3.



initial stones per pit

number of pits	1	2	3	4	5	6
1	D	L	W	L	W	D
2	W	L	L	L	W	W
3	D	W	W	W	W	L
4	W	W	W	W	W	D
5	D	D	W	W	W	W
6	W	W	W	W	W	**W** 🟩

Figure 3. Matrix detailing outcome of various initial states for Kalaha. Note the bottom row with 6 pits for each player. Source: *Solving (6,6) Kalaha* (2011)

In all tests conducted, with both players on equal fotting, the initial player always won the game. Unless the initial player is handicapped in some way, the theory of the always winning first player holds firm.

8.1.4 Value of paths

Changing the AI's preferred choice, to instead select the second or third best path immediately lowered the overall difficulty. It made "mistakes", or suboptimal turns. This resulted in the human player's perception of a more lenient opponent. Still, with a search depth of 4, and the alpha-beta pruning, the AI was admittedly much faster and could still walk in circles around the human player.

9 Comments & Conclusion

Over the course of the project, various implementations and structures were discussed, both in relation to the program's overall structure, as well as the handling of various attributes. Through meticulous discussions in group, and in cooperation with TA's, the end result was established.

However, a number of changes could have been made to better the programs functionality, and results. These are:

- At the current time, the heuristic evaluation function does not take the step cost into consideration, one of many factors left out. Expanding the heuristic evaluation function by combining additional functions and aspects of the game into the evaluation would provide a more game aware agent.
- The game's logic was originally intended to have an object oriented approach. However, this was scrapped for a more simplistic version, as the number of variables proved too insurmountable to keep track of during development. A better structure of the code would be preferable for future iterations.

- The Monte Carlo tree search algorithm is arguably the better option for this sort of task. Thus, future iterations should strive towards implementing a new agent based on this.
- We believe that the focus on a array-oriented data structure proved to be the most optimal, and obvious. The entirety of the game state can be held within the minuscule structure, allowing for easy data manipulation and debugging.

In conclusion, the report describes a functional implementation of a true Minimax algorithm with alpha-beta pruning, solving the stated objective of the project.

10 Appendix

10.1 Gametree model

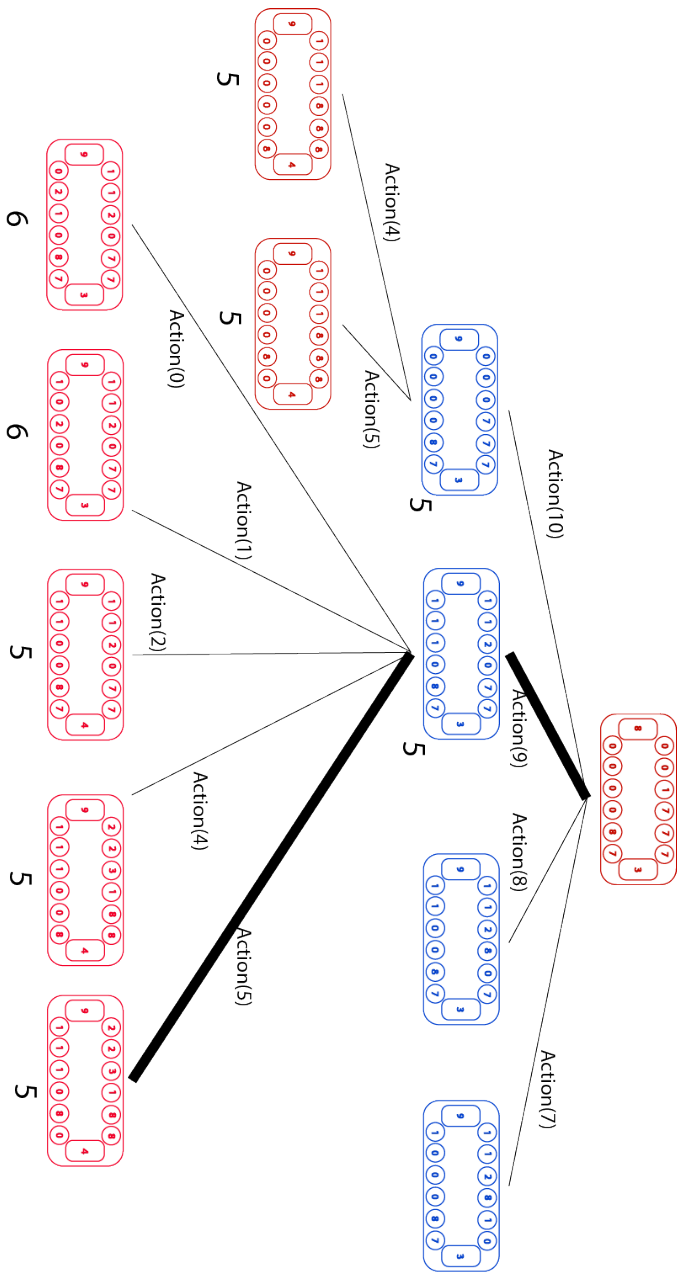


Figure 4. Game tree from a random point in a game.

Bibliography

[1] **Endless Games**. *Classic Mancala Instructions*. URL: https://cn.inside.dtu.dk/cnnet/filesharing/download/2ebfdfe3-1008-49f9-80ba-4b14509aa246. Published 2015, accessed on the 2nd of March, 2019.

[2] Geoffrey Irving, Jeroen Donkers, and Jos Uiterwijk. “Solving Kalah”. In: *ICGA journal* 23 (May 2003). DOI: 10.3233/ICG-2000-23303.

[3] **Anders Carstensen & Kim S. Larsen**. *Solving (6,6)-Kalaha*. URL: http://kalaha.krus.dk/. Published on the 14th of April, 2011.