SJÄLVSTÄNDIGA ARBETEN I MATEMATIK

MATEMATISKA INSTITUTIONEN, STOCKHOLMS UNIVERSITET

S-Kalaha: Solving a Swedish variant of Kalah

av

Carlotta Kvitberg

2026 - No L2

S-Kalaha: Solving a Swedish variant of Kalah

Carlotta Kvitberg

Självständigt arbete i matematik 15 högskolepoäng, grundnivå

Handledare: Per Alexandersson

2026

Abstract

S-Kalaha is a Swedish variant of the Mancala game Kalah. Using concepts prevalent in combinatorial game theory, like game trees and strategies, this Bachelor's thesis studies S-Kalaha as a finite combinatorial game. It also offers a presentation of S-Kalaha, listing its rules, ways of modifying its structure, and even an algorithm for playing S-Kalaha without a physical board. The thesis delves into some programming by estimating game complexity values for S-Kalaha, and weakly solves S-Kalaha for specific configurations of the board. It also finds a relationship between first-player losses and the number of pits on the board.

Sammanfattning

S-Kalaha är en svensk variant av Kalaha, ett Mancala-spel. Denna uppsats undersöker S-Kalaha som ett ändligt kombinatoriskt spel, och använder begrepp kopplade till kombinatorisk spelteori som spelträd och vinnande strategier. Den erbjuder även en presentation av spelet S-Kalaha, med dess regler, sätt att modifiera dess struktur, och en algoritm för att spela S-Kalaha utan en fysisk bräda. I uppsatsen så används även programmering för att beräkna S-Kalahas komplexitet, och för att lösa olika konfigurationer av spelet. Uppsatsen identifierar även ett förhållande mellan förluster för den första spelaren och antalet bon på brädan.

Acknowledgements

I would like to express my gratitude to my supervisor Per Alexandersson for his support and much-needed guidance throughout this project - it would not have been possible without him. Much love and thanks to my friends and family, and anyone who contributed to this thesis by responding to my Kalaha survey.

AI Statement

AI was deliberately avoided while writing this thesis; if any AI was used, it was unbeknownst to the author. Features on Overleaf that explicitly mentioned AI use were ignored, AI-generated code that was offered by others as help was declined, and Google's inescapable AI overview was scrolled past in feeble protest.

Any mistakes or errors in this thesis are entirely and proudly the author's. The author urges readers to consider the environmental impact of generative AI use.

Contents

1 Introduction

In 1964, John B. Haggerty presented the game Kalah to his American colleagues as 'the best all round teaching aid in the country' [Hag64, p. 327]. Besides being a game that requires a lot of counting, there are no chance elements or hidden information in Kalah. If players want to win, there is plenty of room for them to problem-solve and strategize.

On Swedish schoolyards in the early second millennia, Kalah, or Kalaha as it had been dubbed, was not an uncommon sight. Students developed their own strategies, fought over who would play first, and could boast about especially impressive moves they had made for weeks. However, this game of Kalaha did not necessarily follow the official rules of Kalah. Somehow, a variant of Kalah, similar to the Mancala game Dakon, was played instead under the same name. This variant I have dubbed S-Kalaha, with *S* standing in for the word *Swedish*.

S-Kalaha still falls under the class of *sowing games*, Mancala-type games where stones are picked up and 'sown' into pits [Eri96]. In 2013, Divilly et al. described sowing games as being 'less studied in the literature' [DOH13, p. 1], and today, it is still far from exhaustive. This Bachelor's thesis aims to contribute to the body of knowledge by analyzing a Swedish variant of a popular sowing game, using concepts relevant in combinatorial game theory like strategies, perfect play, and game trees.

Due to S-Kalaha's complexity, finding a purely mathematical strategy using combinatorial game theory is difficult. This is a challenge for some combinatorial games, in which the number of possible combinations of games that can be played is daunting. A combinatorial game like Chess might be more likely to be solved using artificial intelligence and search-algorithms, than purely combinatorial game theory.

Using algorithms and game trees, games can be solved by exhaustive searches, declaring which player wins when both players play optimally. For this reason, programming and algorithms also play an important part in this thesis, since its main aim is to solve S-Kalaha, and different configurations of it.

This thesis begins by introducing the concept of combinatorial games, featuring important definitions, examples, theorems, and algorithms that the reader might need. This is followed by a presentation of the game S-Kalaha, explaining its rules, how it differs from Kalah, important notations, and eventually an algorithm for how the game can be played without a physical board. S-Kalaha's game complexity is estimated before it is eventually solved for different configurations of the game, and

interesting patterns in the data is presented to perhaps inspire future research.

The literature that this thesis has relied on is varied; regarding combinatorial game theory, I turned to Matt DeVos and Deborah A. Kent's (2016) *Game Theory: A Playful Introduction* and Aaron N. Siegel's (2013) *Combinatorial Game Theory* for definitions and relevant theory, respectively referred to as [DK16] and [Sie13].

When it comes to the more solution-focused aspect of game trees, like algorithms and definitions for solutions and complexity, Victor L. Allis's (1994) *Searching for Solutions in Games and Artificial Intelligence* has been the main resource, and it will be referred to as [All94]. Irving's (2000) article *Solving Kalah* has also been a major inspiration, and it has served as a guide for how to approach solving S-Kalaha. It will be referred to as [IDU00].

2 Combinatorial games

Definition 2.1. A *combinatorial game* must satisfy the following conditions:

- It is a two-player game.
- It is sequential.
- There is no hidden information.
- It is deterministic (no chance elements involved).

Based on the definition above we can determine that games like Chess, Go, Nim, Hackenbush, and Tic-Tac-Toe are combinatorial, whereas:

- Rock-Paper-Scissors is non-combinatorial as both players move at the same time (non-sequential),
- Poker is non-combinatorial since players keep their cards hidden from each other (hidden information),
- and Ludo (fia med knuff in Swedish) is non-combinatorial because it requires a dice to play (there are chance elements).

An objective with combinatorial game theory is looking at different *strategies* in combinatorial games, in particular identifying *winning strategies*.

Definition 2.2. A *strategy* is a set of moves a player can make during the duration of a game. If a player has the option to guarantee themselves a win with a move or a set of moves, regardless of what the other player does, those moves are referred to as a *winning strategy*.

Kayles is a combinatorial game that will be used as an example in this section. A game of Kayles always starts off with a positive integer number x of adjacent bowling pins, and each player at their turn must remove either one pin, or two adjacent pins. Once there are no more pins to remove, the game is over.

Let's look at a sequence of moves in a game of Kayles with five adjacent pins. At the first game position, the first player L could chose to remove the second pin and the third pin, resulting in a game position where the first pin stands alone with no adjacent pins, and the two last pins standing adjacent to each other.

Figure 1: Kayles game positions {5}, and {1}{2}.

We can write these game positions as sets, in which the value x in a set $\{x\}$ refers to a number of x adjacent pins. The first game position with five adjacent pins is therefore written as {5}, while the second game position, in which there is one lone pin and two adjacent pins, is written as {1}{2}. Since player L was also able to change the game position from {5} to {1}{2} with a single move, we say that {1}{2} is a *sub-position* to game position {5}.

Definition 2.3. A game position B is a *sub-position* to game position A if a player can make a move at game position A that changes the current game position to game position B.

The next player R at game position {1}{2} removes the pin standing alone, which moves us to the sub-position {2}, exemplified in figure 2. From this position, the next player L can either remove both pins, or just one pin. If player L removes one pin, they could choose between the pin on the left, or the pin on the right, but both moves would essentially lead to the same sub-position – a game position in which there is only one pin standing, {1}.

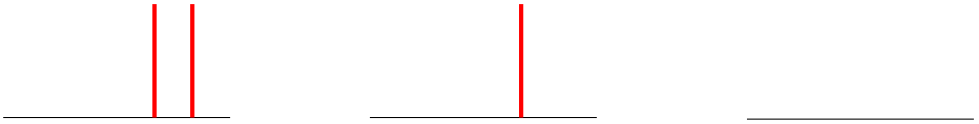


Figure 2: Kayles game positions {2}, {1}, and {}.

Although player L can make three choices, there are essentially two sub-positions from game position {2}: {1} and {}. The empty set {} represents a game position in which there are no pins left to pick up, which per the rules of the game means that the game is over. We would therefore refer to the game position {} as a *terminal position*.

Definition 2.4. A *terminal position* is a game position that has no possible sub-positions.

In Kayles, the player who cannot make a move when it is their turn is dubbed the loser. Therefore, if player L moves to terminal position {} from {2}, they ensure

that the next player R loses. However, if player L moves to sub-position $\{1\}$, the only sub-position player R can move to from there is the terminal position $\{\}$, which results in a loss for player L.

Therefore, a winning strategy in Kayles would be to always move to a terminal position when one gets the chance, since it forces the other player to lose.

2.1 Game trees

Combinatorial games are often aided by *game trees*, in order to visualize all the possible sub-positions players can move to.

Definition 2.5. A *game tree* is a tree-structured graph that represents the possible game positions players can move to throughout a game starting from the initial game position. Game positions are represented as nodes in the game tree.

Definition 2.6. Let A be a node representing a game position A_{GP}, and let the node B represent game position B_{GP}.

If B_{GP} is a sub-position to game position A_{GP}, then node B will be considered a *child node* to node A, while node A will be considered a *parent node* to node B.

Definition 2.7. In game trees, nodes are connected to their child nodes through lines referred to as *branches*. The number of branches is equal to the number of child nodes.

Definition 2.8. *Root nodes* are nodes that only have child nodes, and no parent nodes. They represent the initial game position in the game tree.

Definition 2.9. A game tree is considered *full* if the game tree contains every single game position and all of its sub-positions that players can move to during a game.

Definition 2.10. *Leaf nodes* are nodes that do not have any child nodes, and only have parent nodes. In a full game tree, they would therefore represent terminal positions.

Definition 2.11. A *ply* in a game tree refers to a level of child nodes, with the child nodes to the root node in a game tree being situated at the first ply, and a game tree with only a root node having zero plies. The *ply depth* of a game tree refers to the highest ply a leaf-node in the tree is able to reach.

Let's look at the full Kayles {1}{2} game tree in figure 3. The root node is game position {1}{2}, from which R starts playing. The root node has three branches, connected to its three child nodes. Since this is a full game tree, and there are four leaf nodes, that means that from game position {1}{2} there are a total of four unique games that can be played.

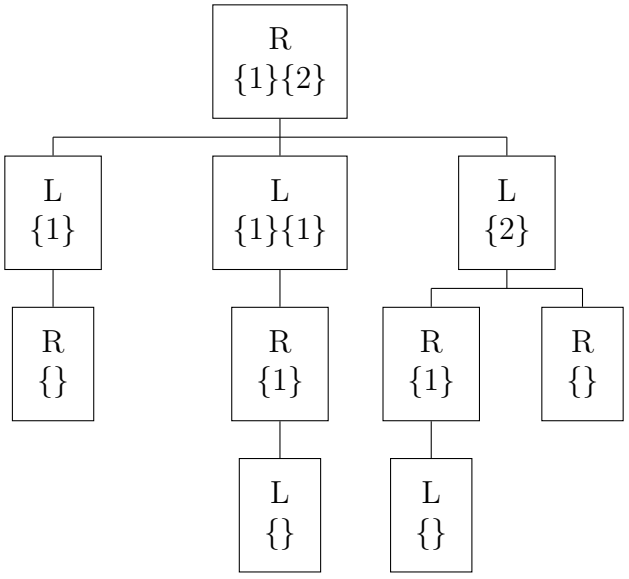


Figure 3: Full game tree for Kayles game position {1}{2}.

Seeing as whoever plays next at a terminal position is dubbed the loser in Kayles, there are two outcomes in which player L loses, and two in which player R loses. Can we find a winning strategy for R?

If R moves to sub-position {1}, they will definitely lose because L can only move to {} from there. If R instead moves to sub-position {1}{1}, they will definitely win, because if we look at the leaf-node of that sub-tree, it ends with L playing next at a terminal position, making R win.

If player R moves to {2}, and player L chooses their next move at random, there would be a 50% chance of player R winning. However, in the field of game theory, one often likes to assume that players are rational and play optimally. As mentioned earlier, it is a winning strategy in Kayles to move to {} as soon as one gets the chance; therefore, under *perfect play*, at game position {2} player L would move to {}, giving player R a 0% chance of winning if they move to {2}.

Definition 2.12. *Perfect play* is a strategy that results in the best possible outcome for that player, regardless of what moves the other player makes.

Under perfect play, a move from $\{1\}\{2\}$ to $\{2\}$ would not be considered since player L would be able to force a win for themselves. Perfect play for player R would be to move to $\{1\}\{1\}$, since no matter what player L does, it will be a win for player R. By using the game tree and analyzing its nodes we have been able to identify a winning strategy for whoever starts the game at game position $\{1\}\{2\}$, which also signifies that we have obtained the game position's *game-theoretic value*.

Definition 2.13. A game's *game-theoretic value* refers to the outcome of the initial position under perfect play, from the perspective of the first player.

Every finite combinatorial game has a game-theoretic value, which means that either one player can force a win for themselves when both are playing optimally, or both players can force draws for themselves. This follows from Zermelo's theorem, which I prove by paraphrasing DeVos and Kent's (2016) proof in [DK16, p. 12-14].

Theorem 2.14. (Zermelo's theorem) *In a finite combinatorial game, if both players L and R cannot force a draw, then one of the players must be able to force a win.*

Proof. Let GT be a full game tree with 0 plies, in which there is only one node, that is both the root node and the leaf node. The leaf node is a terminal position, and therefore there is decidedly only one player who can force a win, or both players can force a draw.

Our induction hypothesis is that this is the case for every full game tree with $p \geq 0$ plies. We assume that this is true for every $n = p$ in which $n \geq 0$.

Let's look at a full game tree with $n+1$ plies then. The root node N in the tree will have l child nodes c_i, $1 \leq i \leq l$, situated in the first ply, who in their turn are the root nodes of their own game trees GT_i, $1 \leq i \leq l$. Since their game trees will have n plies, and we are assuming our induction hypothesis is true, that means that for all child nodes c_i, in which $1 \leq i \leq l$, there is either one player can that force a win, or both players can force a draw when one reaches that node.

Let's assume that L is the first player at the root node N. L can therefore move to any of the child nodes, and there are three different cases that can occur, in which one is always able to determine which player can force a win in GT, or at least whether both players can force a draw.

1. *There is at least one game tree GT_i that L can force a win in.* Then from N, L can move to the child node c_i and force a win from the game tree GT_i, which in turn means that L can force a win in GT.

2. *R can force a win in every game tree* GT_i, $1 \leq i \leq l$. Regardless of what child node L moves to, R is able to force a win from its game tree, which means that R is able to force a win in GT.

3. *L cannot force a win at any game tree, but there is one game tree that both players can force a draw at.* L cannot force a win in GT because none of the child nodes they can move to enable that, but if L moves to child node c_i and both players can force a draw at GT_i, then L can force a draw in GT by moving to c_i.

□

2.2 Solving games

When we find the game-theoretic value of a game, we can declare it *solved*.

Definition 2.15. A game is *solved* when the game-theoretic value of its initial game position has been found.

There are different types of solutions. The definition for a solved game counts as an *ultra-weak solution*, where simply determining the game-theoretic value is enough, and no strategy for how to achieve that value is necessary. If a strategy or algorithm is produced that can lead the player there, we might be looking at a *weak solution* or a *strong solution*. The following definitions come from Allis (1994), with Allis stating that 'reasonable resources' should mean 'a state-of-the-art computer and several minutes of computation time per move' [All94, p. 8].

Definition 2.16. A *weak solution*, in addition to meeting the definition for a solved game, also features a strategy or algorithm that can lead the first player from the initial position to the game-theoretic value, under reasonable resources.

Definition 2.17. A *strong solution*, in addition to following the criteria for a weak solution, has an algorithm or strategy that can lead each player from any legal position to the game-theoretic value, under reasonable resources.

A full game tree analysis, in which every node is considered and all winning strategies are found, can help solve a game; for example, we can say that Kayles game position {1}{2} in figure 3 has been strongly solved since we were able to determine the outcome of every node under perfect play. Although a helpful tool, manually creating a game tree is not feasible for more complex games, which leads us to the next topic of game complexity.

2.3 Game complexity

With games that have not yet been fully solved, it is of interest to look at the game's *state-space complexity* and *game-tree complexity*.

Definition 2.18. *State-space complexity* is the number of legal positions that can be reached from the initial game position. In cases where it cannot be calculated, it can be estimated with an upper bound.

Definition 2.19. *Game-tree complexity* is the number of leaf nodes in the sub-tree that needs to be searched to find the game-theoretic value of the root node.

The state-space complexity offers an insight into whether the game is possible to solve through brute-force methods, which is essentially looking at all the nodes in the game tree and evaluating who wins under perfect play [vdHUvR02]. In 1994, Allis approximated that in order for a game to be solved through an enumerative process, the limit for its state-space complexity would be at 10^{11}, so we might assume the limit is higher today [All94, p. 159].

A traditional mini-max search is a brute-force method that can be used to find the game-theoretic value of a finite combinatorial game. The algorithm assumes that player A wants to maximize the value of the game position, while player B wants to minimize the value player A can receive from it. A mini-max search therefore looks at all the leaf nodes in a game tree, and depending on whose turn it is, either returns the maximum or minimum value of the child nodes to the parent node [Qia03].

Heuristic-based approaches to game tree searches can eliminate some of the branches in the search tree by determining certain nodes as irrelevant or less important under a certain heuristic. For example, alpha-beta pruning is an adjustment to the mini-max algorithm that stops looking at other child nodes once it has found a child node that returns a desired value [Qia03]. This diminishes some of the computational load in a game tree search, and therefore an estimation of the game-tree complexity might for some games be more relevant in determining a game's ability to be solved.

Allis (1994) states that the game-tree complexity of a game can be estimated by playing that game several times and finding the average ply depth d of those sub-trees, and the average *branching factor* b of its nodes. The game-tree complexity GTC can then be estimated as $GTC \approx b^d$.

Definition 2.20. The *branching factor* refers to the number of child nodes at each non-leaf node in a game tree.

3 S-Kalaha

3.1 Board and equipment

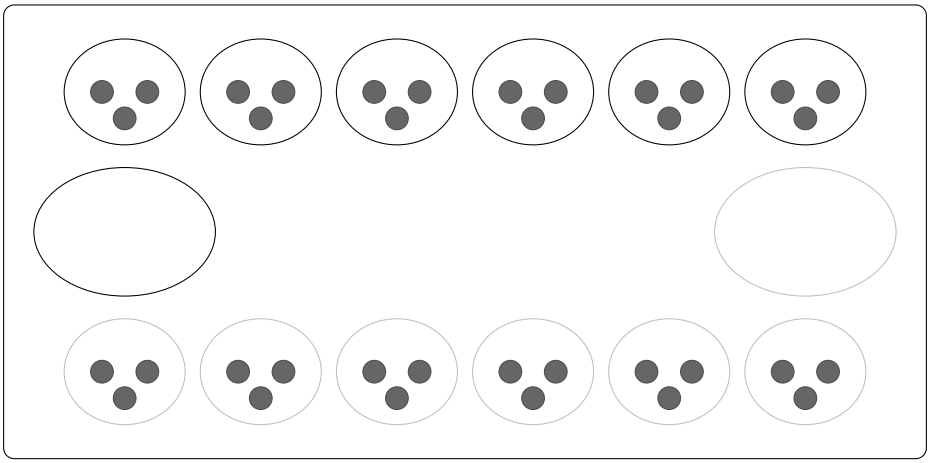


Figure 4: A standard S-Kalaha board at its starting position with 3 seeds in each house. The light gray highlights which pits count as one player's side of the board.

A standard S-Kalaha board consists of two rows with six *houses* facing each other, and two *stores* positioned between the rows near the short side. The players sit opposite each other by the long sides and own the row of houses that is closest to them and the store that is on their right side of the board. A standard S-Kalaha board also carries several *seeds*, and the game starts with an equal amount of seeds being placed in each house.

Definition 3.1. A *store* is one of the two larger pits on the board. Each player owns one of the stores, namely the one on the right side of their view of the board. The stores are where each player collects their seeds.

Definition 3.2. A *house* is a smaller pit on the board. Each player owns a row of houses, specifically the row closest to their side of the board.

Definition 3.3. *Seeds* are small stone-like objects that can be placed in the pits on the board.

When referring to a modified S-Kalaha board, where the number of houses and seeds on the board differ from the standard, the term *S-Kalaha(h,s)* will be used, a modification and notation Irving (2000) too used in his article about Kalah [IDU00].

Definition 3.4. *S-Kalaha(h,s)* is a modified S-Kalaha game, with the parameters h and s.

- h refers to the number of houses in each row on the board, $h \in \{h \in N \mid h \geq 1\}$.
- s refers to the number of seeds in each house at the starting game position, $s \in \{s \in N \mid s \geq 1\}$.

Remark 3.5. A game of S-Kalaha(h, s) will always have a total of $2h + 2$ pits on the board, since there are two rows of h houses and two stores. A game of S-Kalaha(h, s) will always have a total of $2hs$ seeds on the board, since each game starts with s seeds in each of the $2h$ houses on the board.

A standard S-Kalaha game, depicted as its initial game position in figure 4, would therefore be considered S-Kalaha$(6, 3)$.

3.1.1 S-Kalaha(h, s) board notation

S-Kalaha(h, s) game positions, when not illustrated, will be written as a list of numbers between brackets, with each number representing the number of seeds inside a pit on the list. The first number will be the number of seeds in the first house on the current player's side of the board, starting from the left. The next number in the list will be the number of seeds inside the next pit in a counter-clockwise direction, and so forth, with the last pit in the list being the opponent's store.

The pits on each player's side of the board will be separated with a vertical bar to differentiate the two. Unless stated, the game position is always represented from the perspective of the current player.

As an example, the board in figure 5 would be written as (0,0,2,1|1,1,1,0), and the standard S-Kalaha board in figure 4 would be written as (3,3,3,3,3,3,0|3,3,3,3,3,3,0).

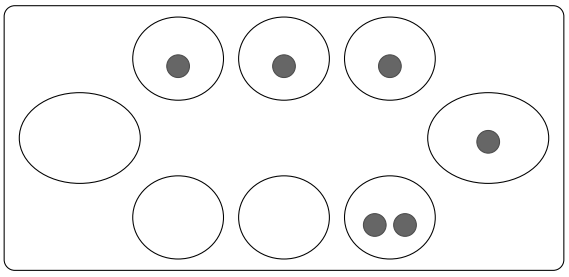


Figure 5: A S-Kalaha$(3, 1)$ game position.

3.2 Rules

At the start of the game, an equal number of s seeds is placed in each house. Players start their turn by picking up all the seeds from a non-empty house on their side of the board, and sowing one seed at a time in each subsequent house in a counter-clockwise direction, including their own store if they encounter it on the way. In order to not give any points to the opponent, players do not sow seeds into their opponent's store, and simply skip to the next store in the sequence.

The player's next course of action once the last seed has been sown depends on which pit it was sown into:

- If the player places the last seed in their own store, the player may make another move. They choose a house on their side of the board, pick up its seeds, and sow them counter-clockwise just like before.
- If the player places the last seed in an empty house, they forfeit their turn and the other player gets to make a move.
- If the player places the last seed in a non-empty house, they get to pick up all the seeds from that house, including the seed they just put down, and continue to sow one seed each in the following *sowing pits* in a counter-clockwise direction.

Definition 3.6. *Sowing pits* on a S-Kalaha board refers to the pits the player is allowed to sow into during their turn, which are all of the houses on the board, and their own store.

The players alternate turns until there is an empty row of houses on the board. Whoever then has the most seeds on their side of the board, including the seeds in their store, is dubbed the winner.

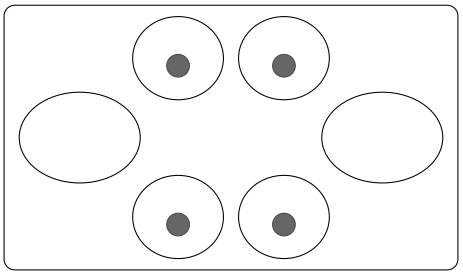


Figure 6: The starting game position in S-Kalaha$(2, 1)$.

To make things clearer, let's look at how a move can be made in S-Kalaha$(2, 1)$. At the starting game position, depicted in figure 6, the first player must choose which of the two houses on their side of the board to pick up seeds from.

The first player chooses to make their *first pick-up* from the first house; they pick up the one seed inside it, and sow it into the next pit in a counter-clockwise direction. It is sown into the second house on the first player's side of the board, and since that house is not empty, a *secondary pick-up* must be made from that house. Figure 7 shows the entirety of this move until its completion.

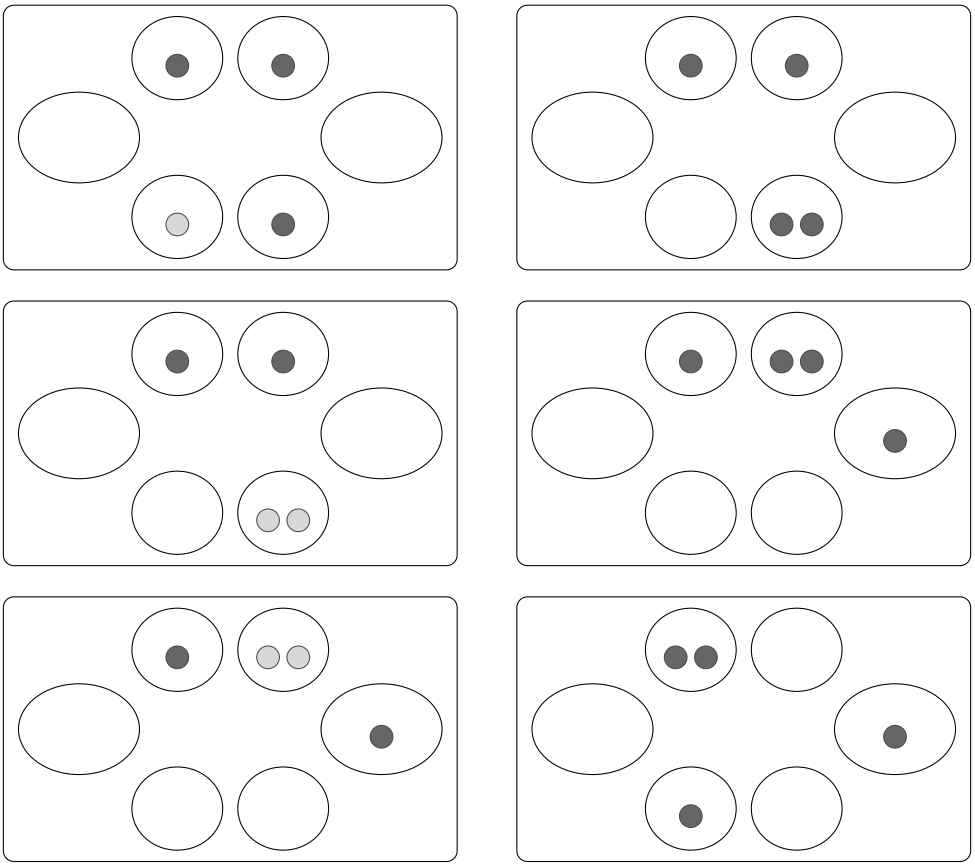


Figure 7: A move made from the game position in Figure 6. The figures on the left represent the act of picking up seeds from a house, with the figure on its right showing what the board looks like once those seeds are sown.

Definition 3.7. A *pick-up* refers to the act of picking up seeds from a house in S-Kalaha. A *first pick-up* refers to the first pick-up made during a move. A *secondary pick-up* refers to any pick-ups made in addition to that first pick-up during the same move. Pick-ups must always be followed by the seeds being sown.

The *move* ends once the player has sown its last seed of its last pick-up into an empty house. The first player made a total of three pick-ups in this move, which is also the only move they were able to make during their *turn*, since placing the last seed in an empty house forfeits one's turn. Note how the player skips their opponent's store when they are sowing the seeds.

Definition 3.8. A *move* in S-Kalaha starts with a player making a first pick-up from a non-empty house on their side of the board, and ends once the last seed of their last pick-up is placed in the player's own store, or in an empty house on the board. There can be more than one pick-up in the same move.

Definition 3.9. A *turn* in S-Kalaha starts with a player making a move on the board, and ends once the last seed of their last pick-up of their last move is placed in an empty house. There can be more than one move during the same turn.

3.3 S-Kalaha as a variant of Kalah

Kalah is a Mancala game sold in the Western world that is marketed as *Kalaha* in Sweden. However, the game Kalaha that some people grew up playing in Sweden does not exactly match the official rules of Kalah. This Kalah variant, which I refer to as *S-Kalaha*, differs from Kalah when it comes to two rules deciding what happens when one has sown the last seed of a pick-up:

1. If one sows the last seed into a non-empty house:
    - **Kalah:** The player immediately forfeits their turn.
    - **S-Kalaha:** The player picks up all the seeds in that house, including the seed they just put down, and continues sowing them counter-clockwise around the board.
2. If one sows the last seed into an empty house on one's side of the board:
    - **Kalah:** The player collects the last seed, and any seeds in the opponent's house on the opposite side of the board, and places them in their store before they forfeit their turn.
    - **S-Kalaha:** The player immediately forfeits their turn.

In an online survey I created, people were asked which of the rule variations mentioned above that they remembered playing with in Sweden. Although the survey was small, with only 80 respondents, it is significant that 62 respondents (77.5%) only remembered playing with the S-Kalaha rules. This was the most common combination of rules that respondents remembered. Only 2 respondents (2.5%) reported playing with both of the Kalah rules, which raises a lot of questions considering those are the official rules of the game.

Some respondents reported playing both variations of a certain rule. With the first rule, 95% of respondents remembered playing the S-Kalaha variant, while 8.75% remembered playing the Kalah one. With the second rule, it seemed to be more mixed, although the S-Kalaha variant was still more common: 86.25% had played the S-Kalaha variant, and 15% had played the Kalah variant (see Appendix A).

Although S-Kalaha, by any other name, is not a recognized Kalah variant even in Sweden, the survey suggests there might be reason to regard it as one.

3.4 S-Kalaha(h, s) as a combinatorial game

Seeing as there are only two players in S-Kalaha, and there are no chance elements, or hidden information, and player play sequentially, S-Kalaha fits the definition of a combinatorial game. Let's see if we can find a winning strategy to the game of S-Kalaha$(2, 1)$ that we started playing in figure 7.

The first player, from now on referred to as PL_1, moved from game position (1,1,0|1,1,0) to sub-position (1,0,1|0,2,0). Considering players in S-Kalaha can make pick-ups, moves, and turns, it is important that we define what a sub-position is in S-Kalaha:

Definition 3.10. A game position B is a sub-position to game position A in S-Kalaha if a player's **move** can change the game position A to game position B.

When PL_1 moves to sub-position (1,0,1|0,2,0), since the last seed was placed into an empty house it becomes the turn of the second player, from now on referred to as PL_2. Since S-Kalaha(h, s) game positions depict the board from the perspective of the current player, PL_2 will play from game position (0,2,0|1,0,1). It is the same game position that PL_1 moved to, but just presented from the perspective of PL_2.

Due to S-Kalaha being quite complex, it will be easier to identify strategies using a full game tree, which can be seen in figure 8. Since the changing perspectives of the game positions might make it confusing to read, in S-Kalaha(h, s) game trees

the game position is always represented from the perspective of the first player in the game tree, with the second player's side of the board being italicized.

The nodes in the game tree will state whose turn it is next at the given game position, so one can know which side of the vertical bar to look at. Leaf-nodes in the game tree will declare the outcome of that game, and also feature the terminal game position.

Let's return to PL_2, who was at the game position (0,2,0|1,0,1), which is written as (1,0,1|*0,2,0*) in the game tree. PL_2 has zero seeds in the first house on their side of the board, two seeds in the second house, and zero seeds inside their store. PL_2's only option is to pick up the seeds from the second house, and as we can see in the tree, it leads to a draw.

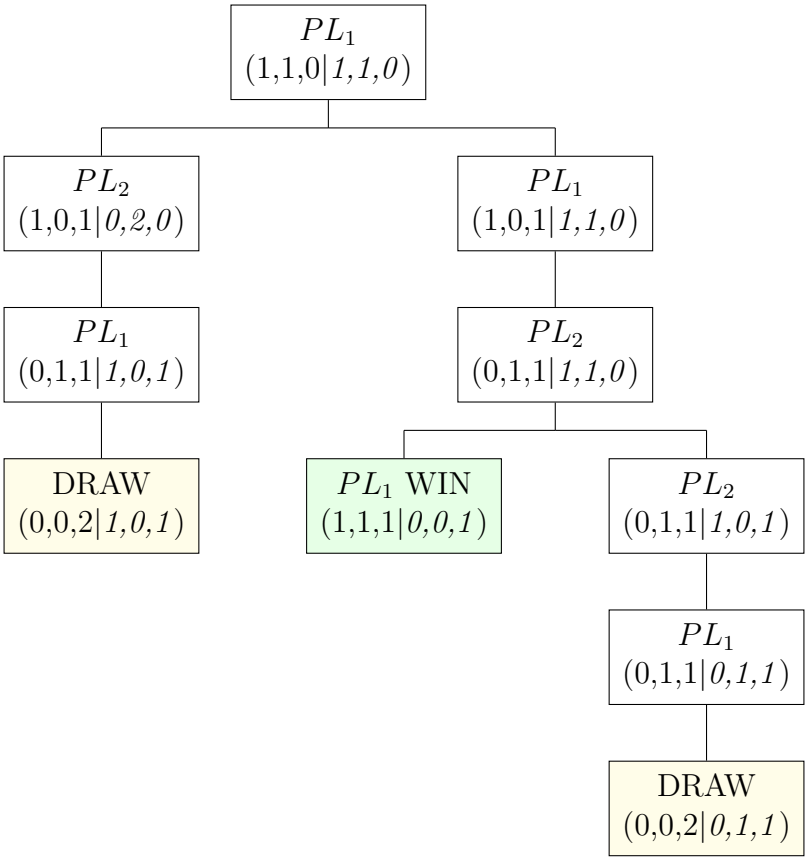


Figure 8: Full game tree for S-Kalaha$(2, 1)$.

In fact, there are only three unique ways this game can be played if we look at the leaf nodes. Two games end with the players drawing, and one game ends with PL_1 winning. Could PL_1 perhaps win under perfect play then?

If we work backwards and start at the leaf node in which PL_1 wins, we find that

it is the child node to a node from which PL_2 starts playing. Under perfect play, players always want to find the best outcome for themselves, so if PL_2 can reach a better outcome by going to the other node, then they will choose that instead.

From the game position (1,1,0|0,1,1) (note that matrices in the text always represent game positions from the current player's perspective), the only other outcome that PL_2 can reach is a draw, which is better than losing. Therefore, PL_2 has a drawing strategy from that node, and under perfect play, that is what they would choose.

Working our way backwards to the root node, we find that the two child nodes PL_1 can move to both result in the players forcing a draw. With a lack of any other option, the game-theoretic value of S-Kalaha$(2, 1)$ is a draw.

Do all S-Kalaha(h, s) games have a game-theoretic value? Zermelo's theorem 2.14 states that for every finite combinatorial game, there is either one player that can force a win, or both players can force a draw. Since that theory was proven, if S-Kalaha(h, s) is finite, then we know this is the case for all S-Kalaha(h, s) games, and that means that we should theoretically be able to solve S-Kalaha.

A terminal position in S-Kalaha is whenever there is a row of houses that is empty, since that is when the rules state that the game ends. Although players of S-Kalaha might not reach a game position in which all the houses are empty, since it might have terminated at an earlier position before that, if it is possible for the houses to run out of seeds, then the game must end at some point, which is how I attempt to prove that S-Kalaha(h, s) is a finite game.

Theorem 3.11. *S-Kalaha(h, s) is a finite game.*

Proof. Players move in S-Kalaha(h, s) by picking up seeds from a house on their side of the board, and then sowing the seeds into the next sowing pit in a counter-clockwise direction. Let's label the sowing pits in figure 9 with indexes, in which the index refers to how many sowing pits it is away from the current player's store in the sowing sequence.

If a player picks up seeds from a sowing pit sp_i, then at least one of those seeds will be sown into the next sowing pit sp_{i-1}. For example, consider the seed inside sowing pit sp_{h-1} in figure 9. If the current player picks it up, then it will be dropped into the next pit sp_{h-2}. As long as the opposing player does not intervene (which I take into consideration further down), it will take that seed a maximum of $h - 2$ moves until it makes its way to the current player's store.

Since the h parameter is always a finite number, so is the maximum number of moves one must make with a certain seed for it to reach the store. Same goes for the number of seeds, defined by the fixed s parameter value. As long as every move either results in seeds getting closer to a store, or the number of seeds in houses is decreasing, then the game must be finite.

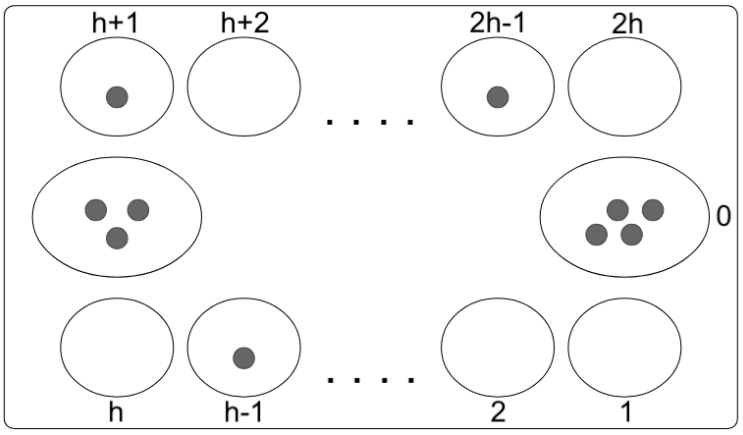


Figure 9: A game position in S-Kalaha(h, s), with the sowing pits indexed.

Moves in S-Kalaha(h, s) are made by picking up seeds, and there are three cases that could occur when a pick-up is made:

1. A seed or seeds are picked up from a house sp_i, $i \neq 1$. A guarantee of one seed will be sown into the house sp_{i-1}, decreasing the number of moves it would need to reach a player's store.

2. A seed or seeds are picked up from a house sp_1 by the current player. A guarantee of one seed will be sown into sp_0, decreasing the number of active seeds on the board by one.

3. A seed or seeds are picked up from the house sp_1, but it is picked up by the opposing player, so the sowing pit sp_0 will be skipped, and the first seed to be sown will be sown into the house sp_{2h}.

In the two first scenarios, the number of moves to get a seed to a store decrease, or the number of active seeds on the board decreases.

The third scenario may seem like a loop-hole that negates the statement I made earlier, but in order for a player to pick up seeds from a house on the opposing player's side of the board, it must be a secondary pick-up, since players are only

allowed to make a first pick-up from their side of the board. In order for a player to get to the opposing player's side of the board during a pick-up, they must cross their own store by sowing a seed into it. So although the seed in the third scenario went from requiring one move to requiring $2h$ moves to reach the player's store, the only way this can occur is if it is directly preceded by a player sowing a seed into its store, decreasing the number of seeds in houses.

The number of moves to make a particular seed reach a store might seem trivial in combination with several other seeds, but if one considers a sparse board, like in figure 9, where players might not be able to get any seed into a store for several moves, it is still a confirmation that since there is a finite number of sowing pits on the board, seeds will always eventually reach a store.

Therefore, for any move made in the game, there will either be one seed that will require one less move to enter a store, decreasing the number of total moves to end the game, or a seed is placed into a store, making the finite number of seeds left in houses on the board decrease by one.

Therefore, after a finite number of moves, a terminal position must be reached.

□

3.5 Algorithm for playing S-Kalaha(h, s)

The codes I have written to collect S-Kalaha(h, s) data have all been built on the same algorithm, which takes a list of numbers, meant to represent the number of seeds in each pit on a S-Kalaha(h, s) board, and the index of the pit on the board that the player will make their first pick-up from.

The list has $2h + 2$ items, one for each pit on the board, and it starts with the house furthest to the left on the current player's row of houses, and continues with the next pit in a counter-clockwise direction around the board, finally ending with the opponent's store. If (0,0,2,1|1,1,1,0) is our game position, the list would be written as [0,0,2,1,1,1,1,0].

In Python, list indexes start with the index 0, which means that since there are h houses in a row on a S-Kalaha(h, s) board, and those rows are followed by a store, the index of the current player's store will be h. When one makes a first pick-up therefore, one must make sure that one chooses to pick up seeds from a non-empty house on the board with an index x, where $0 \leq x \leq h - 1$.

When index x has been selected, one can send the S-Kalaha list K and the index x to algorithm 1. It collects the number of the item in the list at index x as a

variable s, and then changes that item in the list to zero as a way of emptying the pit.

Algorithm 1 Pick-up algorithm, empties the chosen pit.

```
1: function UpdateKalahaList(List K, chosen index x)
2:     s = K[x]                          ▷ s is the number of seeds picked up
3:     K[x] ← 0                          ▷ Empties pit at index x
4:     return SowingSeeds(K, x, s)
```

Algorithm 2 Pick-up algorithm, adds seeds before first vector rotation

```
1: function SowingSeeds(List K, chosen index x, seeds s)
2:     indexlimit = length(K) − 2        ▷ length(X) counts items in X
3:     lastseedpit = x + s
4:     if x < indexlimit then
5:         if lastseedpit > indexlimit then
6:             for x + 1 ≤ i ≤ indexlimit do
7:                 K[i] ← K[i] + 1
8:             return AdditionalSeeds(K, lastseedpit)
9:         else
10:            for x + 1 ≤ i ≤ lastseedpit do
11:                K[i] ← K[i] + 1
12:            return NextAction(K, lastseedpit)
13:    else
14:        return AdditionalSeeds(K, lastseedpit)
```

The algorithm then sends the updated list K, the index x, and the number of seeds s to algorithm 2. This algorithm adds any seeds that would be sown before the player reaches the opponent's store at index $2h + 1$ in the list.

The *indexlimit* in the second line will be equal to $2h$, which is the index of the last house before the opponent's store, so if the seeds were picked up from the house at index $2h$, the next seed would have to be placed into the house at index 0, since players must skip their opponent's store. For this reason, an if-statement is added in line 4 so that this function only applies to pick-ups from houses with indexes $x < 2h$. If $x \geq 2h$, it is sent to algorithm 3.

If the sum $x + s = lastseedpit$ on line 3 is smaller than $2h + 1$, then *lastseedpit* is actually the index of the pit the last seed will be sown into. However, if it is equal to or greater than $2h + 1$, that means it will have to cross the threshold from the opponent's store to the first house in the list at index 0. In that case, every pit with an index $x + 1 < i \leq 2h$ can receive a seed, and it is sent to algorithm 3 to sow its

remaining seeds. If not, then since *lastseedpit* is the index of the pit the last seed is sown into, one can adjust the range of the pits that will receive a seed by letting *lastseedpit* be its upper limit. Since those were all the seeds, that list is sent to algorithm 4 instead.

At algorithm 3 the remaining seeds are sown. With a modular operation, in which the modulus $m = 2h + 1$ is the index of the opponent's store $2h + 1$, one can get the accurate index y of the pit the last seed is sown into when one skips the opponent's store by finding the smallest positive non-negative congruent number y when $x + s$ modulo $2h + 1$. Meanwhile, r refers to the number of times that the sowing crosses the threshold from the opponent's store at index $2h + 1$ to the house at index 0. If $r = 2$, then that means that the sowing has made at least one full rotation of the board after the first crossing of the threshold was made. Every sowing pit on the board should therefore receive $r - 1$ seeds, which they do in line 7.

Algorithm 3 Pick-up algorithm, additional seeds

```
1:  function AdditionalSeeds(Vector K, lastseedpit)
2:      m = length(K) − 1                          ▷ length(X) counts items in X
3:      y = Mod(lastseedpit, m)                    ▷ Mod(a, b) finds integer remainder of (a/b)
4:      r = IntQ(lastseedpit, m)                   ▷ IntQ(a, b) finds integer quotient of (a/b)
5:      for 0 ≤ i ≤ y do
6:          K[i] ← K[i] + 1
7:      if r > 1 then
8:          seed = r − 1
9:          for 0 ≤ i < m do
10:             K[i] ← K[i] + seed
11:         return NextAction(K, y)
12:     else
13:         return NextAction(K, y)
```

Even if $r \not> 1$, one must account for the houses that are sown into the last time the threshold is crossed. So at line 5, for all of the pits with an index i where $0 \leq i \leq y$, in which y is the pit the last seed is sown into, one seed is added. Regardless of whether $r = 0$ or $r > 1$, lists in both instances are sent to algorithm 4, alongside the index y of the pit the last seed was sown into.

Algorithm 4 mainly determines the next step of the game. If the index of the pit the last seed was sown into is $y = h$, then it is the same index as the current player's store. This terminates the players move, but the code also announces that the player gets to make an additional move. Players can then add that new list into

algorithm 1 with a new chosen house index to play again.

If the last seed was not placed into the player's store, one must determine if the house it was sown into was empty or not when it was added. If the house was empty, then the number of seeds at index y in the current list should be equal to one, since the last seed was added to it. In that case, it is the opponent's turn, which the code announces, but if it wasn't, then a secondary pick-up must be made from the house at index y. The algorithm therefore automatically sends the list K and the index y back to algorithm 1. This way, the algorithm only ends once a full move has been completed, returning a sub-position to the game-position K.

Algorithm 4 Pick-up algorithm, fourth step: Determining next act

```
1: function NextAction(List K, last seed index y)
2:     h = (length(K) − 2) ÷ 2
3:     if y = h then
4:         output Current player moves again!
5:         return K
6:     else
7:         seedsinpit = K[y]
8:         if seedsinpit = 1 then
9:             output Opponent's turn!
10:            return K
11:        else
12:            return UpdateKalahaList(K, y)
```

4 S-Kalaha Complexity

4.1 State-space complexity

The state-space complexity was found for certain cases of S-Kalaha(h, s) by creating a code in Python that looked at every single node in the full S-Kalaha(h, s) game tree and collected unique game positions. Note that although the same board configuration could show up twice, it would be treated as separate game positions if PL_1 and PL_2 had both gotten to play next at it, since each game position would have different sub-positions depending on whose turn it was.

h \ s	1	2	3	4	5	6
1	2	3	4	3	3	3
2	10	19	105	179	269	212
3	45	1'219	11'184	44'242		
4	1'220					
5						
6						

Table 1: State-space complexity values for S-Kalaha(h, s).

The state-space complexity of the remaining cases in the table was estimated by counting the ways one can choose k pits to place 1 seed into n times. We can interpret this as there being a set with k pits, and in order to disperse our n seeds, we must choose n of them, with one seed going to each of those chosen pits. Repetitions are allowed; for example, if we have a set $\{P_1, P_2, P_3, P_4, P_5, P_6\}$ with six pits, and we have four seeds, we could choose P_2 four times, leaving us with a board that looks like (0,4,0|0,0,0).

The number of ways that we can choose $n = 2hs$ pits from a board with $k = 2h+2$ pits, including repetition, can be found with the following theorem, from [BXGS23]:

Theorem 4.1. *The number of ways of selecting n objects from a set with k objects, if order does not matter and repetitions are allowed, is equal to:*

$$\binom{k+n-1}{n}.$$

With $n = 2hs$ and $k = 2h+2$, we can find the number of ways the board can be configured.

$$\binom{2h+2+2hs-1}{2hs} = \binom{2h+2hs+1}{2hs}.$$

As previously mentioned, it is possible for players to encounter the same board configuration in S-Kalaha(h, s), which counts as two separate game positions since they will not have the same sub-positions that they can move to, and it is not certain what those two game positions might have been sub-positions to. We must therefore multiply the number of possible board configurations by two, and that will be the upper bound for the S-Kalaha(h, s) state-space complexity. We could consider the symmetry of the board, but since this is not done in [All94] nor in [IDU00], neither shall I. Table 2 features the estimated state-space complexity for the remaining cases.

h \ s	1	2	3	4	5	6
3					20'590'944	64'448'228
4		4'085'950	77'134'200	$7.01 \cdot 10^8$	$4.11 \cdot 10^9$	$1.8 \cdot 10^{10}$
5	705'432	$1.69 \cdot 10^8$	$6.32 \cdot 10^9$	$9.53 \cdot 10^{10}$	$8.36 \cdot 10^{11}$	$5.12 \cdot 10^{12}$
6	10'400'600	$7.13 \cdot 10^9$	$5.25 \cdot 10^{11}$	$1.31 \cdot 10^{13}$	$1.73 \cdot 10^{14}$	$1.48 \cdot 10^{15}$

Table 2: Estimated state-space-complexity for S-Kalaha(h, s).

Solving S-Kalaha(h, s) through brute-force methods should be possible, since the estimated state-space complexity is smaller than 10^{11} for most cases [All94]. The largest full-game database for Kalah that was created in [IDU00] was for Kalah$(4, 3)$, which had a state-space complexity of $4.6 \cdot 10^9$. The larger cases had to be solved with an 'optimized searching program' [IDU00, p. 143].

Awari, a Mancala-type game, has a state-space complexity of around 10^{12} [All94], and it was through a state-space search of its positions using retrograde analysis that led to it being solved, requiring 144 processors [RB02]. This leads me to think that a state-space search with my limited resources might not be viable for all cases of S-Kalaha(h, s).

4.2 Game tree-complexity

Since players of S-Kalaha$(1, s)$ only have one house on their side of the board, the branching factor b at each non-leaf node will always be 1. The estimated game-tree complexity b^d is therefore trivial, as $1^d = 1$ for all integers d. Therefore, we will only estimate the game-tree complexity for S-Kalaha(h, s) in which $2 \leq h \leq 6$ and $1 \leq s \leq 6$.

The game-tree complexity was calculated by running 100 games of S-Kalaha(h, s) in a program where moves were chosen randomly. The program collected the ply depth of every game, and its average branching factor by collecting the branching

factor of every non-leaf node it encountered, and dividing the sum of the branching factors by the number of non-root-nodes it had encountered. With the data from the 100 games, an average branching factor b and an average ply depth d was defined, with which one could approximate the game-tree complexity GTC with $GTC \approx b^d$. The results are in table 3.

h \ s	1	2	3	4	5	6
2	$3.72 \cdot 10^0$	$4.71 \cdot 10^0$	$5.61 \cdot 10^1$	$5.59 \cdot 10^1$	$9.4 \cdot 10^1$	$4.4 \cdot 10^1$
3	$7.85 \cdot 10^0$	$9.98 \cdot 10^2$	$1.72 \cdot 10^4$	$2.14 \cdot 10^5$	$6.00 \cdot 10^5$	$4.2 \cdot 10^6$
4	$2.47 \cdot 10^3$	$3.73 \cdot 10^6$	$2.49 \cdot 10^9$	$4.77 \cdot 10^{11}$	$1.36 \cdot 10^{13}$	$1.73 \cdot 10^{14}$
5	$2.83 \cdot 10^6$	$6.96 \cdot 10^{11}$	$2.07 \cdot 10^{16}$	$2.64 \cdot 10^{19}$	$5.36 \cdot 10^{22}$	$1.33 \cdot 10^{24}$
6	$6.14 \cdot 10^9$	$3.18 \cdot 10^{18}$	$1.4 \cdot 10^{25}$	$2.7 \cdot 10^{29}$	$1.97 \cdot 10^{34}$	$3.09 \cdot 10^{37}$

Table 3: Estimated game-tree complexity of S-Kalaha(h, s) games.

For the smaller cases, the results do not cause any worry. However, it took 300 hours of CPU time to solve connect-four using an alpha-beta search, and connect-four has an estimated game-tree complexity of 10^{21} [All94]. Irving (2000) applied alpha-beta pruning and several other optimizations when trying to solve larger cases of Kalah(h, s), and he was unsuccessful in the case of Kalah$(6, 6)$, which had an estimated game-tree complexity of $2 \cdot 10^{33}$ [IDU00].

Considering that S-Kalaha$(6, 5)$ and S-Kalaha$(6, 6)$ have greater estimated game-tree complexities than Kalah$(6, 6)$, and that there are several other S-Kalaha(h, s) configurations in table 3 that have a greater estimated game-tree complexity than connect-four, it does cause some concern. It does seem though like Irving's (2000) solution of the larger Kalah(h, s) cases also considered the amount of seeds that a player could collect in addition to winning, and since my aim is simply to find who wins under perfect play, in which the best outcome is simply a win, it might not be as hard to solve it.

5 Solving S-Kalaha

The large state-space complexities for some of the (h, s)-values in table 3 led me to write a code in Python that would examine the game tree with a forward-search, starting from the root node instead of looking at every possible leaf nodes first.

In order to make the game tree smaller, I terminated nodes once a player had more than half of the seeds on the board inside their store, since that would be the amount of points needed to win the game. The code was not interested in finding 'better' wins that might have collected even more seeds, and that is where my algorithm might differ from some others. As long as the player found that they could win from a child node, no other child nodes would be analyzed.

Initial versions of the code struggled to find a solution for some larger cases. However, I had earlier written a code that looked at every way the first player could move during their first turn at the initial game position of an S-Kalaha(h, s) game. This in turn created a much smaller game tree, in which the leaf nodes were simply the game positions that PL_2 would start playing at once PL_1 had terminated their turn. In several cases, I found that the first player had a *winning opening*.

Definition 5.1. A *winning opening* is a move or a set of moves the first player can make during their first turn that leads to them winning.

h \ s	1	2	3	4	5	6
1		✓			✓	✓
2		✓				✓
3						
4			✓	✓	✓	✓
5				✓	✓	✓
6				✓	✓	✓

Table 4: S-Kalaha(h, s) games with winning openings.

This made me edit my code so that it would first look every possible *turn sub-position* the player could move to, before moving on to the next player's turn. If the player was able to force a win during their turn, it would find that reasonably quick instead of perhaps getting stuck in an unnecessary sub-tree.

Definition 5.2. Let there be two players L and R. Let A be a game position that player L plays next at.

Game position B is a *turn sub-position* to A if player L can make a move or several moves from A that leads them to game position B, and player R plays next at game position B.

A game tree in which the child nodes are turn sub-positions will therefore be referred to as a *turn game tree*.

Definition 5.3. A *turn game tree* is a game tree in which each child node represents a turn sub-position to the game position in the parent node.

If the player was unable to win during their turn, the program would instead sort the turn sub-positions it had encountered by the number of seeds in that player's store. The first child node whose game tree would be analyzed would be the one featuring the most seeds in the player's store, and this was a heuristic that made a big difference and that finally led most configurations of S-Kalaha(h, s), in which $1 \leq h \leq 6$ and $1 \leq s \leq 6$, to be solved.

Since the code was able to return what house the first player at the initial game position would have to pick up seeds from in order to force their win/draw/loss from the initial game position, a solution with this code would be considered a weak solution.

Table 5 shows the game-theoretic values of the S-Kalaha(h, s) games. Using the program, S-Kalaha(h, s) was weakly solved for every case in the table, except for S-Kalaha$(5, 3)$ and S-Kalaha$(6, 3)$. The search-trees for S-Kalaha$(5, 3)$ and S-Kalaha$(6, 3)$ were much too great, but they were eventually solved by breaking down the tree further and making a physical game tree, which is further explained in the next section.

h \ s	1	2	3	4	5	6
1	D	W	D	L	W	W
2	D	W	W	L	L	W
3	W	W	W	W	W	L
4	W	W	W	W	W	W
5	W	W	W	W	W	W
6	W	W	W	W	W	W

Table 5: Game theoretic values for S-Kalaha(h, s).

There seems to be a prominent first-player advantage in S-Kalaha, but for certain values of (h, s), the first player has no winning strategy from the initial game position.

The code written to solve games of S-Kalaha(h, s) at their initial game positions can be found at https://github.com/krtct/S-Kalaha-solution-code.git. The code consists of several different functions, many of them used recursively, and it is 641 lines long.

5.1 Solving S-Kalaha$(5, 3)$ and S-Kalaha$(6, 3)$

What seemed to make S-Kalaha$(5, 3)$ and S-Kalaha$(6, 3)$ particularly problematic in comparison to the other (h, s)-values seemed to be two things: the sizes of their game trees, and their lack of winning openings. S-Kalaha$(5, 2)$ and S-Kalaha$(6, 2)$ had both small enough search-trees to be solved, while S-Kalaha$(5, 4)$ and S-Kalaha$(6, 4)$ had winning openings.

I chose to assist the code by cutting down the search tree manually, and hand-picking turn sub-positions for the first player PL_1 that I thought would be particularly prosperous. While the code followed a heuristic that examined child nodes in order of how many seeds there were in the current player's store, I myself followed an additional heuristic; if there were several turn sub-positions PL_1 could move to that enabled them to collect several seeds in their store, I would often choose the turn sub-position A that lead to the least number of turn sub-positions B the next player PL_2 could move to from A.

If PL_1 moves to a turn sub-position A, then in order for A to have a winning game-theoretic value for PL_1 one must prove that PL_2 cannot force a draw or win when they get to play from game position A. One has to therefore look at every turn sub-position B that PL_2 could move to, to eliminate that possibility. The code was therefore maybe getting stuck trying to determine the game-theoretic value of a node with 300 child nodes, hence my reason for looking at smaller sub-trees.

Every turn sub-position in the tree was sent to the solution code, and if the code could not find a solution for that game position under a reasonable time, I would look at the child nodes of that turn sub-position as well. If a game-theoretic value was returned, I would apply that to the game tree, or either continue looking at the other child nodes.

Eventually, for both S-Kalaha$(5, 3)$ and S-Kalaha$(6, 3)$ I was able to prove that there was a way for the first player to force a win from the initial position. Figures 10 and 11 depict their modified turn game trees in which every leaf node is a confirmed PL_1 win by the solution code. Here are the ways in which the tree has been modified that may require an explanation:

- Child nodes to nodes that PL_1 played from are connected with red lines, to indicate that although there were more child nodes PL_1 could have moved to, this is a child node PL_1 must move to in order to force a win.
- If the code found that PL_1 could force a win from a certain node, a green arrow will descend from that node in the game tree to a tinted leaf-node featuring the text PL_1.
- The game position distinguishes between PL_1's side of the board and PL_2's side of the board by italicizing PL_2's side of the board. Note that the game position is always represented from the perspective of PL_1.
- For the S-Kalaha$(6, 3)$ game tree in figure 11, the game positions have been given a value n_i since they could not fit into the nodes. The corresponding game position of that value can be found in table 6 instead.

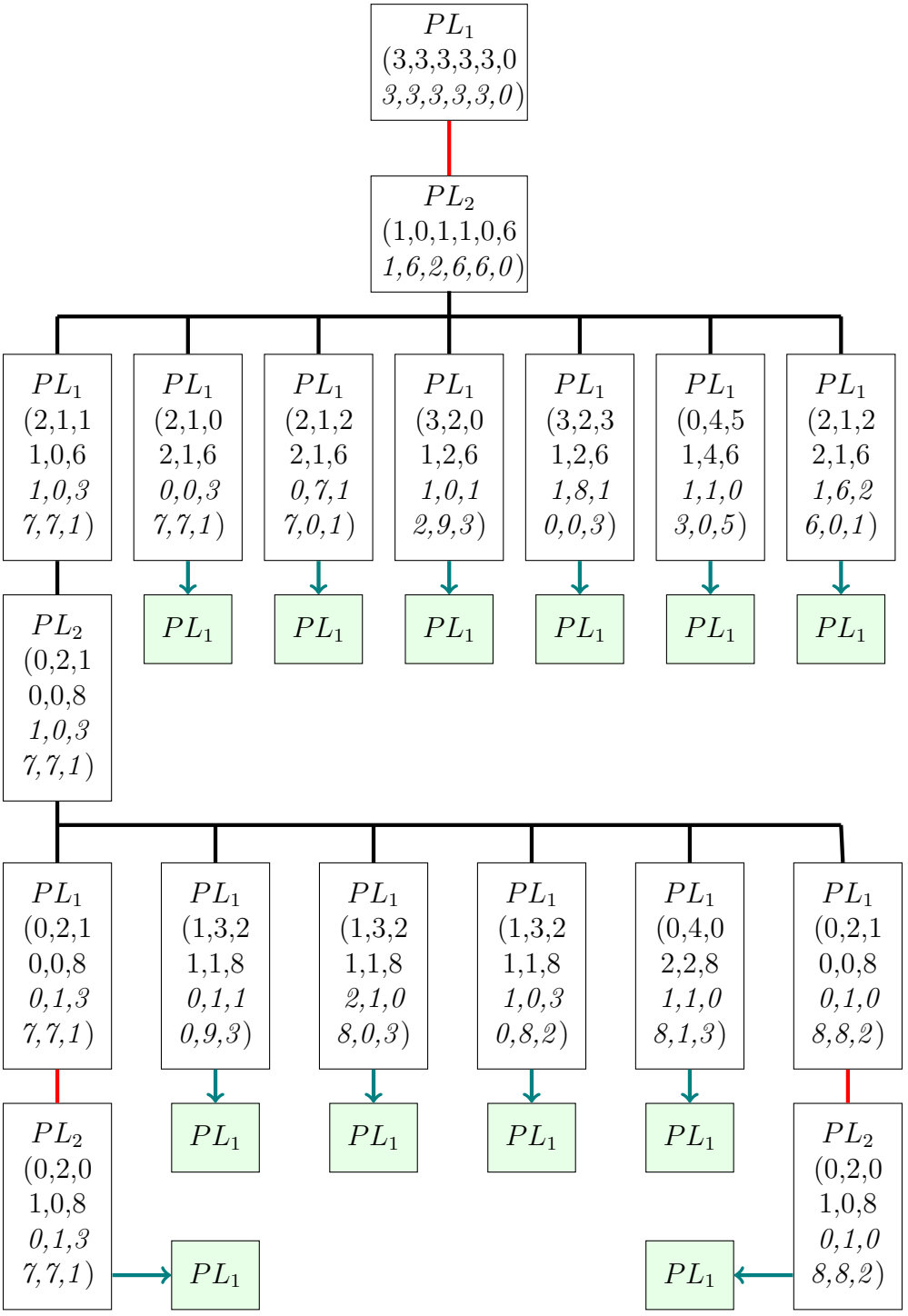


Figure 10: Modified turn game tree for S-Kalaha$(5, 3)$.

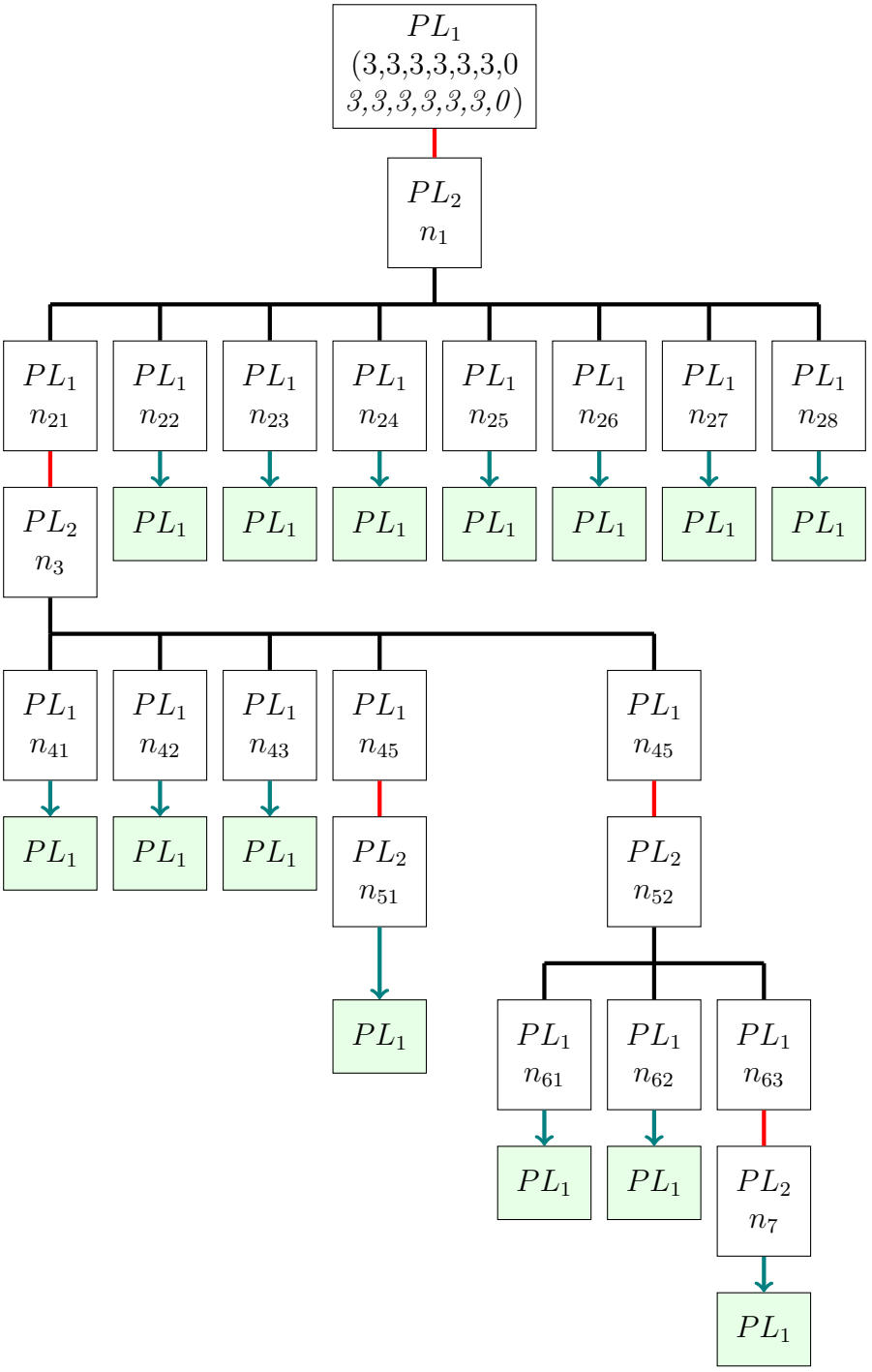


Figure 11: Modified turn game tree for S-Kalaha$(6, 3)$.

$n_1 =$	(0,1,1,0,2,1,10 *2,6,6,6,0,1,0,*)
$n_{21} =$	(2,3,1,1,0,2,10, *1,0,1,8,2,3,2*)
$n_{22} =$	(1,1,1,0,2,1,10, *2,0,7,7,1,2,1*)
$n_{23} =$	(1,0,2,1,2,1,10, *2,6,0,7,1,2,1*)
$n_{24} =$	(2,0,1,2,1,2,10, *3,0,7,1,2,3,2*)
$n_{25} =$	(2,3,1,1,0,2,10, *1,0,1,8,2,2,3*)
$n_{26} =$	(1,1,1,0,2,1,10, *2,0,7,7,1,1,2*)
$n_{27} =$	(1,0,2,1,2,1,10, *2,6,0,7,1,1,2*)
$n_{28} =$	(2,0,1,2,1,2,10, *3,0,7,1,2,2,3*)
$n_3 =$	(3,0,2,2,1,1,11 *0,1,0,9,0,4,2,*)
$n_{41} =$	(3,0,2,2,1,1,11, *0,0,1,9,0,4,2*)
$n_{42} =$	(4,1,3,3,2,0,11, *1,0,1,1,1,5,3*)
$n_{43} =$	(5,2,1,4,3,1,11, *0,0,1,1,0,2,5*)
$n_{44} =$	(5,2,1,4,3,1,11, *0,1,0,0,1,2,5*)
$n_{45} =$	(0,3,2,5,4,2,11, *1,1,0,1,0,0,6*)
$n_{51} =$	(5,2,1,4,0,1,13 *1,1,0,0,1,2,5,*)
$n_{52} =$	(0,3,2,5,4,0,12 *0,2,1,1,0,0,6,*)
$n_{61} =$	(0,3,2,5,4,0,12, *0,0,2,0,1,1,6*)
$n_{62} =$	(0,3,2,5,4,0,12, *0,2,0,0,1,1,6*)
$n_{63} =$	(0,3,2,5,4,0,12, *0,2,1,0,1,0,6*)
$n_7 =$	(1,3,2,0,5,1,13 *1,0,2,1,0,1,6,*)

Table 6: Game positions of the S-Kalaha$(6, 3)$ turn game tree in figure 11, represented from the perspective of the first player, with the italicized numbers being PL_2's side of the board.

5.2 Solving larger cases of S-Kalaha, and loss polynomials

I sent more cases of S-Kalaha(h, s) through the solution code and found that there was a significant first-player advantage for larger values of s. However, at certain s, there would be a loss, like for example when the (h, s)-parameters were $(h, 2h + 1)$.

If a player picks up $2h + 1$ seeds from a house in a S-Kalaha(h, s) board, the last seed of that pick-up will return to the same pit they picked up the seeds from, because there are $2h + 1$ sowing pits on the board. Since they just emptied that pit, that means that the last seed of that pick-up will be into an empty house.

For the first player at the initial position of S-Kalaha$(2, 5)$, in which all houses have $5 = 2 \cdot 2 + 1$ seeds in them, whatever house PL_1 picks up seeds from, their pick-up will result in them immediately forfeiting their turn, and only sowing one seed into their store as a result of the singular rotation.

This is also the case for S-Kalaha$(3, 7)$, and S-Kalaha$(4, 9)$, and so on. I ran a program to see for which $s \geq 2h$ when $2 \leq h \leq 6$ a game of S-Kalaha(h, s) would not have a winning game-theoretic value, from now on shortened as GTV.

I have listed the ten first combinations of those (h, s)-values in table 7. They also all happen to have a losing GTV. Consider that for every s that is not listed in a specific h column, up until the s in the last row, that corresponding S-Kalaha(h, s) game has a winning GTV – the first player advantage is clearly quite strong.

h	2	3	4	5	6
s	4	6	8	10	12
	5	7	9	11	13
	20	42	24	110	124
	25	49	42	121	156
	100	160	72	1'109	169
	104	257	81	1'118	982
	125	294	583	1'210	1'883
	229	300	648	1'220	1'894
	284	343	656	1'331	2'028
	500	1'041	729	6'685	2'040

Table 7: S-Kalaha(h, s) games where GTV is a loss, for the first ten s-values in which $2h \leq s$.

With these s-values I began to identify a pattern, that seemed to hold for all $2 \leq h \leq 6$, but not necessarily $h = 1$. If the s-parameter for a S-Kalaha(h, s) game was the value of a specific polynomial in which the only variable was the expression

$2h+1$, then for every $2 \leq h \leq 6$, that S-Kalaha(h, s) game would have a losing game-theoretic value. These polynomials I will refer to as loss polynomials, shortened as l-polynomials, in which $l = 2h + 1$.

First I must explain how table 8 is organized. The left column in the table showcases the identified loss polynomials, and the columns to their right represent the values of those polynomials depending on what number h is, since $l = 2h + 1$.

What makes the loss polynomials important is that for all $2 \leq h \leq 6$ in a game of S-Kalaha(h, s), if the s-parameter was the value of one of the l-polynomials on the left with $l = 2h+1$, then that game of S-Kalaha(h, s) would have a losing game-theoretic value. In other words, if you look at a column in table 8 where the top row is $h = x$, and choose a number y that is listed underneath that row in the same column, then the game-theoretic value of S-Kalaha(x, y) will be a loss.

$l = 2h + 1$	$h = 2$	$h = 3$	$h = 4$	$h = 5$	$h = 6$
$l^1 - l^0 =$	4	6	8	10	12
$l^1 =$	5	7	9	11	13
$l^2 - l^1 =$	20	42	72	110	156
$l^2 =$	25	49	81	121	169
$l^3 - l^2 =$	100	294	648	1'210	2'028
$l^3 - l^2 + l^1 - l^0 =$	104	300	656	1'220	2'040
$l^3 =$	125	343	729	1'331	2'197
$l^4 - l^3 =$	500	2'058	5'832	13'310	26'364
$l^4 =$	625	2'401	6'561	14'641	28'561
$l^5 - l^4 =$	2'500	14'406	52'488	146'410	342'732
$l^5 - l^4 + l^2 - l^1 =$	2'520	14'448	52'560	146'520	342'888
$l^5 - l^4 + l^3 - l^2 + l^1 - l^0 =$	2'604	14'706	53'144	147'630	344'772
$l^5 =$	3'125	16'807	59'049	161'051	371'293
$l^6 - l^5 =$	12'500	100'842	472'392	1'610'510	4'455'516
$l^6 =$	15'625	117'649	531'441	1'771'561	4'826'809

Table 8: Loss l-polynomials and their value, depending on h.

Since a vast majority of the game-theoretic values when $s \geq 2h + 1$ in S-Kalaha(h, s) games are a win, it is noteworthy that the number of sowing pits $l = 2h + 1$ seems to have a consistent role in the few cases where the game-theoretic value is a loss.

What these specific (h, s)-parameters also had in common was that the first player's first turn at the initial game position always ended after a single move. After trying several games out myself, I found that it seemed to be because the

move was always setting up for a scenario in which the first player would eventually be forced to make a secondary pick-up from a house with $l = 2h + 1$ seeds, which would terminate their turn once they had sown the last seed.

Why this always seems to enable PL_2 to force a win at their turn is nothing I am certain of, but my theory is that the multiple rotations around the board that PL_1 makes during its first turn adds so many seeds to the other pits on the board that once it is PL_2's turn, one single move could amass even more seeds than PL_1.

Table 9 consists of all the s-parameter values, $2h \leq s \leq 100'000$, for which the game-theoretic value of the S-Kalaha(h, s) game is a loss, and for which I have not been able to rewrite the s-value as an l-polynomial that also applies for every $2 \leq h \leq 6$.

h	2	3	4	5	6
s	229	32	24	1'109	124
	284	160	42	1'118	982
	1420	257	583	6'685	1'883
	1'704	1'041	22'407	7'002	1'894
	2'045	1'302	37'843	8'789	8'149
	2'170	1'838	42'515	11'184	8'824
	19'693	2'762	47'239	64'235	22'732
		12'605			
		15'503			
		77'207			

Table 9: Games of S-Kalaha(h, s), where $2h \leq s \leq 100'000$ and GTV is a loss, excluding those already in table 8.

Since this is related to the number of sowing pits on the board, perhaps there is potential for a more mathematical ultra-weak solution to be proven one day for S-Kalaha(h, s). For now, it is simply an interesting piece of data.

6 Conclusion

S-Kalaha is a finite combinatorial game with a fairly similar game complexity to its Mancala peers, Kalah and Awari. S-Kalaha(h, s), for parameters $1 \leq h \leq 6$ and $1 \leq h \leq 6$, was able to be weakly solved using a Python code that applied a forward-search to its game tree, using alpha-beta pruning and a heuristic that prioritized collecting as many seeds as possible. The code ceased looking for other outcomes once a win had been found, making it possible to perhaps modify the code in the future so that certain wins are preferred over others.

S-Kalaha(h, s) was found to have a prominent first-player advantage, but the code was also able to solve additional S-Kalaha(h, s) games for which the game-theoretic value was a loss. The number of pits on the board seems to have a particular effect on the game-theoretic value of an S-Kalaha(h, s) game; if $2 \leq h \leq 6$, and the s-parameter was able to be written as a certain polynomial consisting of the value of the expression $2h+1$, which I refer to as a loss polynomial, the game-theoretic value of that S-Kalaha(h, s) game would be a loss.

However, no ultra-weak solution based on the game's structural constraints was able to be proven, and this is something that could be of interest if one wants to take a more combinatorial game theory approach to S-Kalaha moving forward. Another suggestion for future research is to look at the list of loss polynomials for which when set as the s-parameter in S-Kalaha(h, s) when $2 \leq h \leq 6$, always leads to a losing game-theoretic value. It might be of interest to find why the polynomial

$$(2h+1)^5 - (2h+1)^4 + (2h+1)^2 - (2h+1)^1$$

is a loss polynomial, while

$$(2h+1)^5 - (2h+1)^4 + (2h+1)^3 - (2h+1)^2$$

isn't.

Since the Mancala-game Dakon shares some similarities with S-Kalaha, it might also be interesting to compare the two games with Kalah to see if there are any shared or unique aspects that make one game more complex than the other, since there is a great overlap between their rules. This may help solve other Mancala games in the future – perhaps even those not officially recognized yet.

A Appendix: Kalaha survey

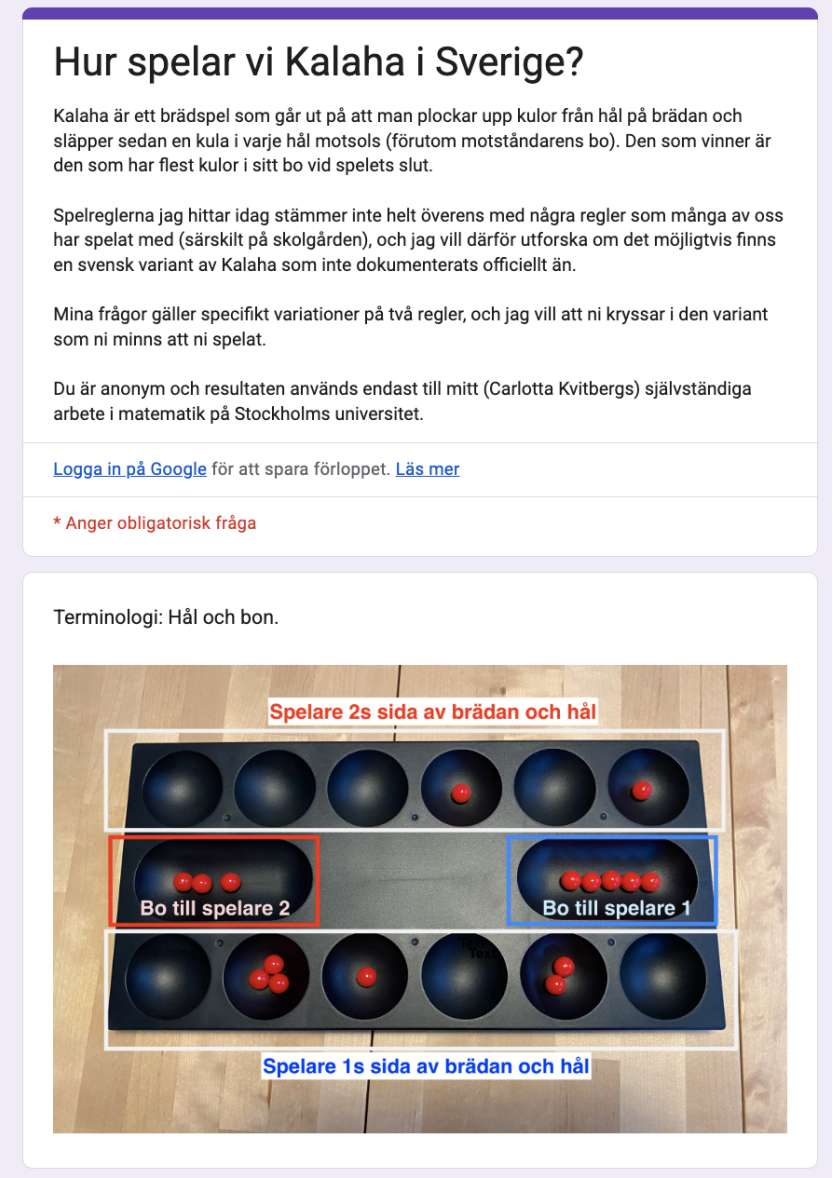
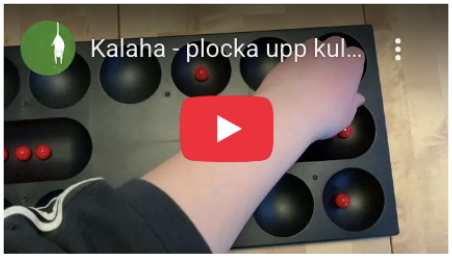


Figure 12: Introduction to the survey.

Fråga 1. Vad händer när man plockar upp kulor från ett hål och stoppar den sista kulan i ett hål som inte är tomt eller som är sitt bo?
Nedan finns det två olika varianter man kan välja mellan. Det finns även korta videoklipp man kan titta på om man vill se variationen "in action".

Fråga 1, variant A: Stoppar man sista kulan i ett hål (och inte sitt bo) så är ens tur slut och motståndaren får spela.

Fråga 1, variant B: Stoppar man sista kulan i ett hål med kulor (obs: inte sitt bo) får man plocka upp kulorna i det hålet och fortsätta så tills man stoppar sista kulan i ett tomt hål eller i sitt bo.



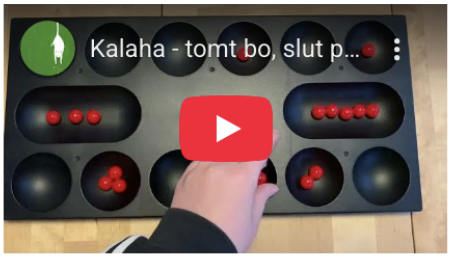
Fråga 1: Vilken variant minns du att du spelat med? *

- ◯ Variant A: Så fort man stoppar sista kulan i ett hål som inte är sitt bo så blir det motståndarens tur.
- ◯ Variant B: Stoppar man sista kulan i ett hål där det finns kulor får man plocka upp dem och fortsätta spela.
- ◯ Jag minns att jag har spelat båda varianter.
- ◯ Jag minns varken av varianterna.

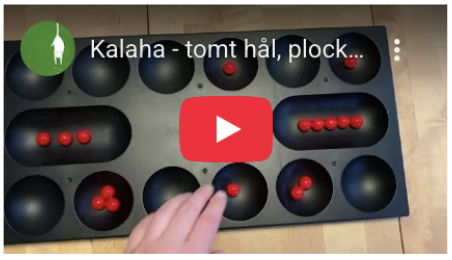
Figure 13: Question 1.

Fråga 2: Vad sker när man stoppar sista kulan i ett tomt hål på sin sida av brädan?
Nedan finns det två olika varianter man kan välja mellan. Det finns även korta videoklipp man kan titta på om man vill se variationen "in action".

Fråga 2, variant A: Stoppar man sista kulan i ett tomt hål så är ens tur över.



Fråga 2, variant B: Om man stoppar sista kulan i ett tomt hål på sin sida av boet, och det finns kulor i hålet mittemot på motståndarens sida av brädan, så får man plocka upp kulan man precis lade ner och kulorna i hålet mittemot och stoppa dem i sitt bo.



Fråga 2: Vilken variant minns du att du har spelat med? *

- ◯ Variant A: Stoppar man sista kulan i ett tomt hål är det nästa spelares tur.
- ◯ Variant B: Stoppar man sista kulan i ett tomt hål på sin sida av brädan så kan man plocka på sig kulor om det finns kulor i hålet mittemot hos motståndaren.
- ◯ Jag minns att jag har spelat båda varianter.
- ◯ Jag minns varken av varianterna.

Figure 14: Question 2.

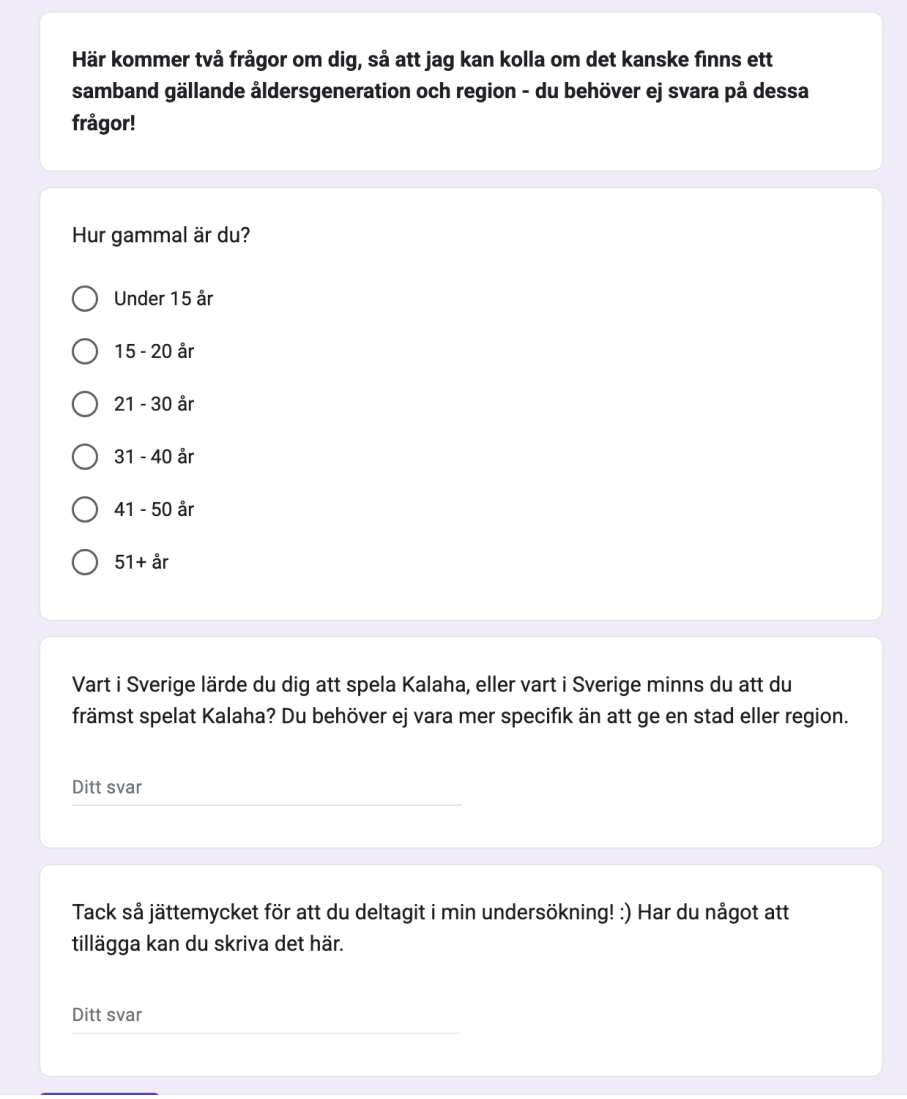


Figure 15: Questions about age and location.

Figure 16: Q1 pie graph.

Figure 17: Q2 pie graph.

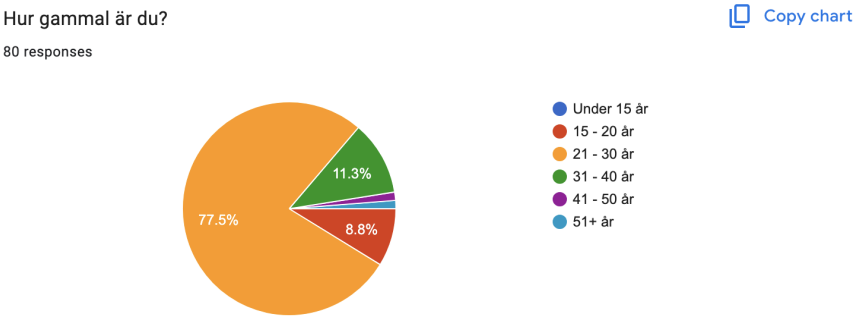


Figure 18: Age pie graph.

Figure 19: Location responses.

References

[All94] L. Victor Allis. *Searching for Solutions in Games and Artificial Intelligence*. PhD thesis, Rijksuniversiteit Limburg, 1994. doi:10.26481/dis.19940923la.

[BXGS23] Rikard Bögvad, Qimh Xantcha, Håkan Granath, and Olof Sisask. *Algebra 1*. Matematiska institutionen, Stockholms universitet, 11 edition, 2023.

[DK16] Matt DeVos and Deborah A. Kent. *Game Theory: A Playful Introduction*. American Mathematical Society, 2016.

[DOH13] Colin Divilly, Colm O'Riordan, and Seamus Hill. Exploration and analysis of the evolution of strategies for mancala variants. In *2013 IEEE Conference on Computational Inteligence in Games (CIG)*, 2013. doi:10.1109/CIG.2013.6633628.

[Eri96] Jeff Erickson. Sowing games. In Richard Nowakowski, editor, *Games of No Chance*, volume 29 of *MSRI Publications*, pages 287 – 297. Cambridge University Press, 1996.

[Hag64] John B. Haggerty. Kalah — an ancient game of mathematical skill. *The Arithmetic Teacher*, 11(5):326–330, May 1964.

[IDU00] Geoffrey Irving, Jeroen Donkers, and Jos Uiterwijk. Solving kalah. *ICGA Journal: The Journal of the Computer Games Community*, 23(3):139–147, 2000. doi:10.3233/ICG-2000-23303.

[Qia03] Qian. *The Evolution of Mulan: Some Studies in Game-Tree Pruning and Evaluation Functions in the Game of Amazons*. PhD thesis, The University of New Mexico, 2003. URL: https://api.semanticscholar.org/CorpusID:127721616.

[RB02] John W. Romein and Henri E. Bai. Awari is solved. *ICGA Journal: The Journal of the Computer Games Community*, 25(3):162–165, 2002. doi:10.3233/ICG-2002-25306.

[Sie13] Aaron N. Siegel. *Combinatorial Game Theory*. American Mathematical Society, 2013.

[vdHUvR02] H. Jaap van den Herik, Jos W.H.M Uiterwijk, and Jack van Rijswijck. Games solved: Now and in the future. *Artificial Intelligence*, 132(1-2):277–311, 2002. doi:10.1016/S0004-3702(01)00152-7.