

CS5346: Fall 2018

Project 2: Kalah

Submitted by: Gentry Atkinson

Teammate: Vishal Kumar



Table of Contents

1.	Introduction	2
1.1	Rules of Kalah	3
2.	Contributions	3
3.	Analysis of the Problem	4
3.1	Domain and Goal	4
3.3	Proposed solution	4
4.	Evaluation Function Design	5
5.	MinMaxAB	6
5.1	AlphaBetaSearch	7
6.	Methodology	10
6.1	Algorithm	10
6.2	Data Structures	10
7.	Program Implementation	14
7.1	Header Files	
7.2	Source Files	
7.3	Driver Function	
8.	Sample Runs	28
9.	Analysis of Program and Result	31
10.	Comparison of Two Evaluation Functions	33
10.1	Comparison of Search Algorithms	34
11.	Summary	

1. Introduction:

Mancala is one of the oldest known games. It and its many variations are still played in many parts of the world. It is conceptually and materially very simple. However, the strategy can still be very challenging. This makes it very well suited to the exploration of two-player searches implementing intelligent evaluation functions.

Kalah is a variation of Mancala which was developed in the mid-20th century for American audiences. The Kalah game board gives each player a certain number of holes (or houses) and starts with a certain number of stones (or seeds) in each hole. A standard notation was introduced to describe versions of Kalah with varying numbers of holes and stones. Generally $\text{Kalah}(m, n)$ describes a game being played with m holes per player and n stones per hole. This project has chosen to focus on $\text{Kalah}(6, 6)$, meaning that all games described in this project are being played with 2 players, 12 holes, 72 stones, and 2 kalahs (or stores). The use of each of these will be described in the following section.

The easy rule set and surprisingly difficult strategy of Kalah make it ideal as a vehicle with which to study intelligent 2-player searches. Human players can quickly develop an intuitive sense for ideal board states but computers do not share the benefits of this intuition. The understanding of the human players must be distilled into tight, manageable rule sets which a computer can then apply. There are several conditions which can easily be seen as being desirable but it is not always easy to say which of these desirable conditions is more desirable than its mates. This is why an intelligent computer or human must look past the immediate state of the board and choose a strategy which benefits them several moves into the future.

$\text{Kalah}(6,6)$ has been solved by computer systems. This means that a player who has the first move can guarantee a win if they play an optimal strategy. However this optimal strategy depends on the existence of a full game database. Any turn of Kalah can have up to 6 legal moves, turns can repeat, and the game can last for more than a dozen turns. This means that the full tree generated by Kalah can be extremely large. In order to avoid the large expenditures of space and time which are required by full game databases, this project will attempt to use smart searching algorithms which can limit their search depth to a manageable level while still producing a near optimal result. In order to allow the system to judge the optimality of an unfinished game, a pair of evaluation functions have also been developed.

The purpose of this project is to develop and compare two methods of evaluating a board's state and two methods of searching a tree of board states to back-propagate the values of the

leaf nodes in the tree. These search algorithms also each include a cut-off which determines when a branch is no longer worth searching in order to optimize the execution time of this search.

Each author of this project has developed and contributed on evaluation algorithm. These algorithms are simply referred to by the author's first names in the code base for simplicity's sake. The performance of these algorithms is compared by playing a series of games with each algorithm choosing the moves for one player. In order to account for all factors which might affect the algorithms' performances these games are played over a variety of tree depths and with each of the two search algorithms.

Similarly, two optimal search algorithms have been chosen from the literature: Alpha Beta Search and MinMax AB. These algorithms have been implemented in the code base of the project and their performances will be evaluated against one another in a similar fashion to the two evaluation algorithms. While the evaluation algorithms were compared with each search algorithm in place to assure fairness, the search algorithms will be compared with each evaluation algorithm in place to insure that they also each have a fair chance.

After the test cases have been executed, this project will collect and present the results. Some ad hoc conclusions can be drawn about the performance of each algorithm from analysis alone but in the end data speaks loudest. The results will be tabulated and presented so as to present whatever clear distinctions in the code's performances can be made.

1.1 Rules of Kalah:

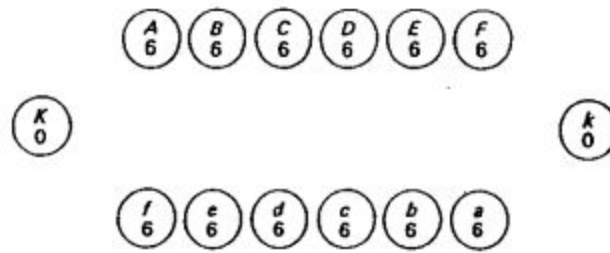


Figure 2-8 Starting position in kalah. There are six stones in each hole and none in each kalah.

1. Each player begins the games with 6 holes, 1 kalah, and 6 stones in each hole. Each player owns the kalah that is to his or her right and the six holes closest to him or her.
2. The objective of the game is to have more than half of the stones placed in the kalah owned by a player.
3. If all of the holes of one player become empty, all of the stones in his or her opponent's holes are placed in the first player's kalah and the game ends.
4. A player makes a move by picking up all of the stones in one of his or her own holes. These stones are then distributed one at a time into the holes around the board. A player places stones into his or her own kalah while distributing in this fashion but not into his or her opponent's.
5. If the last stone of a player's move is into the kalah, that player goes again.
6. If the last stone of a player's move is placed into an empty hole belonging to that player, all of the stones in the adjacent hole (across the board) are placed into that player's kalah along with the one stone from the player's side.

2. Contributions:

Gentry Atkinson:

- Structure and object design
- Evaluation algorithm
- Code consolidation

Vishal Kumaar:

- Search algorithm implementation
- Code review
- Evaluation algorithm

This project would like to acknowledge John McCarthy for his development of the simplified rules for the game of Kalah as used in this project.

3. Analysis of the Problem:

Adversarial searches are most often considered with regard to game playing but they are actually much more broadly applicable. Any time that some actor must choose an optimal strategy in conditions that are not wholly determined by his or her actions then it is best to assume that those outside conditions are being determined by some adversary who will inevitably choose the “worst” possible conditions for the actor. This is the origin of min-max searching. An actor always chooses the maximal path while assuming that some adversary will “choose” the minimal path. Some examples of real world problems that benefit from min-max searching outside of gameplay: logistics, code optimization, personnel management, and epidemiology.

But min-max searching is not a simple problem so it is natural that many algorithms have been generated in order to accomplish this task. Two of the most well known are MinMaxAB and AlphaBeta Search. It is possible that each of these algorithms have an ideal application at which they excel but in order to derive meaningful results regarding their applicability this project will apply each of these two algorithms to the task of optimally playing a game of Kalah in conditions of restricted time and memory usage.