Final Project

CS580 – Introduction to Artificial Intelligence

The Game of Kalaha

Jatin Mistry

Supervised By

Kenneth De Jong

Department of Computer Science

George Mason University

2014-2015

The Game of Kalaha

Abstract

This report is based on developing a game of Kalaha in Prolog, which explores the areas of Artificial Intelligence in two-player games with perfect information. In AI aspect, the player of the program finds intelligent moves by using minimax algorithm with alpha-beta pruning. The theoretical aspects that are applied in implementing the game are game trees, static evaluation function, minimax algorithm, alpha-beta pruning and logic programming using Prolog.

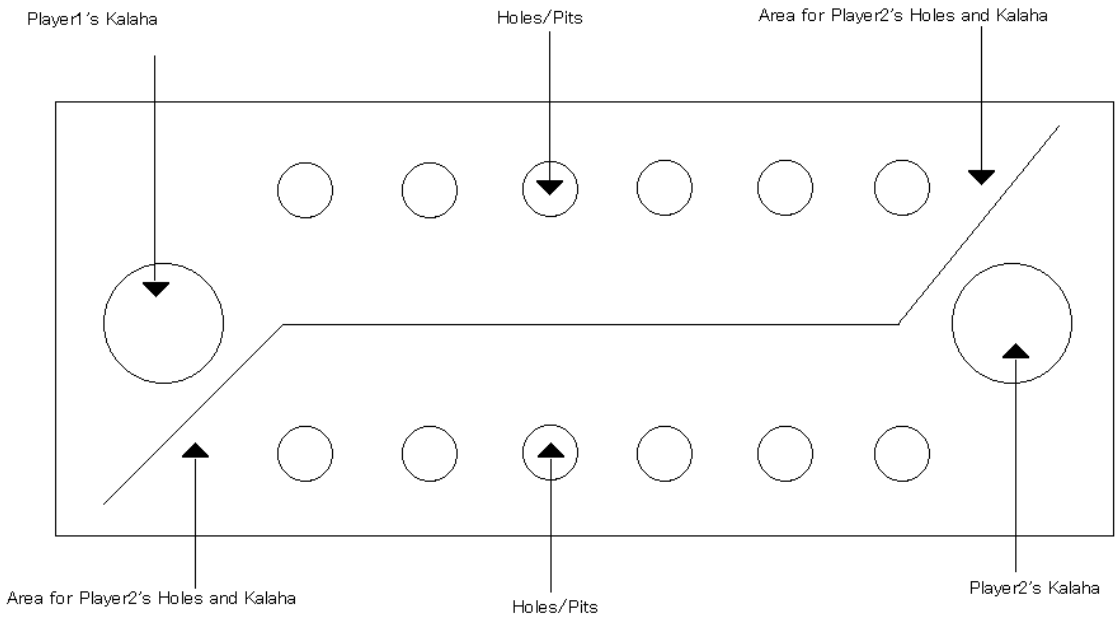
Contents

1. Introduction

Games contribute to Artificial Intelligence like Formula 1 racing contributes to designing automobiles. Games, like the real world require the ability to make some decisions, even when optimal decision is infeasible. They also have well-defined problems that are generally interpreted as requiring intelligence to play well and introduce uncertainty as opponent's moves cannot be determined in advance. Since search spaces can be very large, games become a good test domain for various search methods and development of pruning methods that ignore portions of the search tree that do not affect the outcome.

Kalaha is an abstract strategy game invented in 1940 by William Julius Champion, Jr. The notation (m,n)-Kalaha refers to Kalaha with m pits per side and n stones in each pit. For the current project we would focus on Kalaha(6,6). i.e. Kalaha with 6 pits per side and 6 stones in each pit.

Diagram:



The Board of Kalaha (6,6)

Kalaha is played on a board with two rows of 6 holes (pits) with 72 stones and two stores called Kalahas. The two players Player1 and Player2 sit at each side of the board. Each of the players takes 36 seeds and puts six pieces in each of the six pits on their side of the board. Kalahas are left empty at the start of the game. Player1 will always start the game. A move is made by selecting a non-empty hole at the player's side of the board. The seeds are lifted from this hole and sown in an anti-clockwise manner starting from the next pit. The player's own Kalaha(home) is included in the

sowing, but the opponent's Kalaha(home) is not. Note that the captured counters never re-enter the game.

The three possible outcomes of a move in Kalaha are as follows:

1. Kalaha-move: If the last seed lands in the player's score pit, the player will have another turn.

2. Capture: If the last seed lands in an empty pit on the player's side of the board, it is added to the player's total and any seeds in the opposite pit (opponent's pit) are added as well. The player's turn is then over and the opponent moves next.

3. Null Capture: If the last seed played ends in an empty pit on the player's side and the opponent's pit that is directly opposite is also empty. For this case, the last seed is still placed in the player's Kalaha and the player's turn is over.

4. Normal-move: If the last counter is put anywhere else, the turn is over directly and it is now the opponent's turn.

5. End Game: When a player's all holes are completely empty, the game ends. The player who still has stones left in his holes captures those stones and puts them in his Kalaha. The players then compare their Kalaha. The player with most stones wins.

This project is an attempt to implement the game of Kalaha, which will be played by two players (both are computer). The AI aspect here is to study AI in two-player games, i.e a computer player always makes intelligent moves by finding the best move for a given game position.

2. Rules

Rules for playing Kalaha and implementing it:

1. A player can start his/her move from any non-empty pit from his/her side of the board.

2. The player cannot start his/her move using the pieces on the opponent's side of the board.

3. The player's Kalaha, the player's pits and the opponent's pits are included in sowing. Opponent's Kalaha is not included in sowing.

4. Seeds are sowed in counter-clockwise (anti-clockwise) direction.

5. The player cannot sow any of his/her seeds into the opponent's Kalaha. i.e. we have to wrap around opponents Kalaha without placing any stone there.

6. If the last seed land in the player's Kalaha, the player's score increases by 1 and he retain the right to continue playing.

7. If the last seed does not end up in the player's Kalaha, the player loses his/her turn.

8. If last seed on opponents side in a bowl where there are 2 or 3 stones then those stones be will moved to player's Kalaha. This shall be repeated clockwise until reaching own Kalaha or there are not 2 or 3 stones in the bowl.

9. We cannot seed into the original hole from which the stones were picked.

10. If the last counter is put into an empty hole on the player's side of the board, a capture takes place: all stones in the opponent's pit opposite and the last stone of the sowing are put into the player's kalaha and the opponent moves next.

11. If the last counter is put anywhere else, it is now the opponent's turn.

12. The seeds that are being captured or the seeds that has entered both players Kalaha do not re-enter the game. The game value of this game thus depends on the configuration of the active seeds or seeds that are not captured.

3. Goals

- The goal of this project is to learn the concepts of Artificial Intelligence such as game trees, static evaluation function, Minimax algorithm and Alpha-Beta pruning.

- The game will be developed in Prolog. A rule-based approach is taken to implement the game in prolog.

- The rules/code will be added by the developer.

- The goal of the player playing Kalaha is, to be the player with the highest number of seeds in its Kalaha.

- Success will be measured by how well the program does playing another opponent over a series of N games, which will be done by having it play another program.

4. Observations

1. The game value of this game decreases as with each turn that any of the two players get to sow.

2. The player's score increases by at least 1 for every capture the player makes.

3. If in the case for normal play, player 1 starts by moving the seed that will end up in his Kalaha, he will lead by at least one capture. Thus, he can optimise his chances of winning by leading by at least one capture.

4. There are five types of moves that a player can make:
    - Move to capture:
      This is a move to capture at least itself (1 seed) if the opponent's pit opposite to the player's pit where his last seed lands is empty. If the opponent's pit is non-empty, the player can capture at least 2 seeds.

    - Move to defend:
      This is a move to prevent a capture of a seed that is exposed to the opponent in the immediate move that the opponent might make.

    - Mirror the move of the opponent:
      This is a move where the player imitates the moves made by his opponent.

    - Move to land seed in Kalaha:
      This is a move where the last seed of the player lands in his Kalaha or moving of the seeds that are closer to his Kalaha.

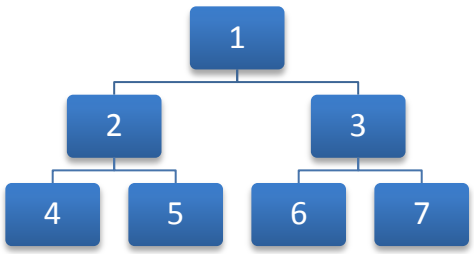
    - Random move:
      This is a move other than those described above.

5. Game Strategy

This section explains the theoretical aspects such as game trees, static evaluation function, minimax search algorithm and alpha-beta pruning.

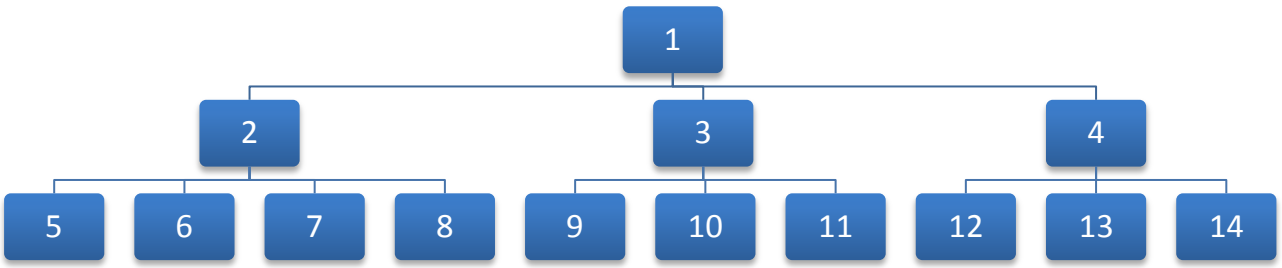
5.1 Game Trees

The simplest type of tree is a binary tree. Each node of a binary tree has up to two child nodes. Each of these children nodes have up to 2 children and so forth.



A simple binary tree with a depth of 2.

In the above figure, the top node is called the root node, and the tip nodes are called the leaf nodes. A binary tree with a depth of 2 can have maximum 7 nodes and so on. In contrast to the binary tree, many game trees contain multiple nodes in each branch. In fact, a game tree can have as many nodes as they are required, depending upon the game and the game position.



A typical game tree with a depth of 2.

In Kalaha, each node can contain up to 6 children, because there are maximum 6 possible moves a player can make each turn and hence there are 6 possible game positions resulting from moves.

5.2 Static Evaluation Function

An evaluation function, also known as a heuristic evaluation function or static evaluation function, is a function used by game-playing programs to estimate the value or goodness of a position in the minimax algorithms. The evaluation function is typically designed to prioritize speed over accuracy; the function looks only at the current position and does not explore possible moves (therefore static).

For the game of Kalaha, the static evaluation function can be the difference between the numbers of stones in players Kalaha.

5.3 Minimax Algorithm

A minimax algorithm is a recursive algorithm for choosing the next move in an n-player game, usually a two-player game. Each position/status of the game is evaluated by using a static evaluation function and the resulted value indicates how good or bad it would be for a player to reach that position. Assume that the game's participants are two players, namely– Min and Max. Min will make the move that minimizes the maximum possible value and Max will make the move that maximizes the minimum value of the position, resulting from the opponent's possible following moves. If it is a player's turn to move, that player gives a value to each of his legal moves.

If we depict this strategy in a game tree, each node will represent a (game) position and the children of a given node will represent all of the possible subsequent positions that the parent node has. Therefore, if the parent node is a position occurred by a Max's move, then each of its children nodes are the possible subsequent positions that could occur by a Min's move and vice versa. A static evaluation function can assign a value to each node and this value serves as the basis for an assessment of the best successor to the current node. The depth of the tree can be formed depending on the game level and requirements such as how many positions that should be looked ahead and searching time etc.

If a parent node belonged to Max and he were to choose among tip nodes that belonged to Min, he would choose the node with the largest value. Thus, the parent node is assigned a backed-up value, which is the maximum of the evaluations of the tip nodes. On the other hand, if a parent node belonged to Min and he were to choose among tip nodes that belonged to Max, he would choose the node with the smallest value. Thus, the parent node is assigned a backed-up value, which is the minimum of the evaluations of the tip nodes. After the parents of all tip nodes have been assigned backed-up values, the backing up value procedure goes to another level or depth of the tree, assuming that Max would choose the node with the largest backed-up value while Min would choose the node with the smallest backed-up value.

The backing up value procedure continues level by level until the successors of the start node are assigned backed-up values. If it is Max's turn to move at the start, then the successor node having the largest backed-up value should be his choice as the first move.

5.4 Alpha-Beta Pruning Algorithm

Minimax algorithm is not so efficient, namely because it needs to search all of the tip nodes of the tree. Hence, alpha-beta pruning was born.

Alpha–beta pruning is a search algorithm that seeks to decrease the number of nodes that are evaluated by the minimax algorithm in its search tree. It is an adversarial search algorithm used commonly for machine playing of two-player games. It stops completely evaluating a move when at least one possibility has been found that proves the move to be worse than a previously examined move. Such moves need not be evaluated further. When applied to a standard minimax tree, it returns the same move as minimax would, but prunes away branches that cannot possibly influence the final decision.

The benefit of alpha–beta pruning lies in the fact that branches of the search tree can be eliminated. This way, the search time can be limited to the 'more promising' subtree, and a deeper search can be performed in the same time. The optimization reduces the effective depth to slightly more than half that of simple minimax if the nodes are evaluated in an optimal or near optimal order (best choice for side on move ordered first at each node).

6. Design Decisions

This project covers the logical programming part learned during the course of Introduction to Artificial Intelligence. I found it interesting and wanted to get a better understanding about logical programming since it was a new programming paradigm for me. For this reasons I decided to implement this game using Prolog.

a. I had to deal with different kinds of methods that Prolog provides:
   i. Declarative Prolog for querying facts and rules. In our program I had to use lists, recursive rules etc.
   ii. Procedural Prolog was used as well since there was a need to deal with input and output, arithmetic, manipulate flow controls etc.

b. The simple game playing framework was used.
   The higher-level game playing framework consists of following steps:
   i. Initialize the board.
   ii. Display the board.
   iii. Choosing a move.
   iv. Making a move.
   v. Displaying the board again and
   vi. Allowing the opponent to make his/her move.

c. The Board information and representation
   The data structure to represent the board is of the type [Own, OwnKalaha, Opp, OppKalaha] where Own: the list of number of stones in player's own six holes; OwnKalaha: the number of stones in player's own Kalaha; Opp: the list of number of stones in opponent's holes; OppKalaha: the number of stones in opponent's Kalaha.

d. The static evaluation function
   The evaluation function is the difference between the numbers of stones in the two Kalahas. i.e static evaluation function = OwnKalaha – OpponentKalaha.

e. To control the backtracking (and avoid wasting time exploring possibilities that lead nowhere), cuts (! Predicate) were used which offers control over the way Prolog looks for solutions. According to me, a sensible way of using cut is to try and get a good, clear, cut free program working, and only then try to improve its efficiency using cuts. It's not always possible to work this way, but it's a good idea to aim for.

7. Results

The following are the results that are achieved:

- The game works as per the rules of Kalaha.
- Played a series of games against a player employing Minimax algorithm and found that it is very difficult to win against that player.
- The Minimax algorithm and Alpha-Beta algorithm work well under all tests. It is very difficult for the player with manual algorithm (i.e making manual moves) to beat the player with Minimax or alpha-beta algorithm.
- The optimal value of state-space depth is 3. Any value greater than 3 can take a lot of time for the program to compute optimal moves.
- The game can be automated to be played by 2 players both as computer by setting the player1 and player2 algorithms as Minimax or Alpha-Beta (but not manual algorithm).
- When a series of games were played between two players both employing Minimax or Alpha-Beta algorithm, then the game used to take some time to finish as both players would be equally efficient in making their choices of move (maximizing their chances of winning and minimizing the opponents chances of winning).
- The two possible outcomes of this game are either Player1 wins the game or either Player2 wins the game.
- The Minimax algorithm generates all possible moves and evaluates the best possible move and executes it.
- The Alpha-Beta algorithm extends Minimax algorithm by replacing the Minimax flag with alpha and beta. The algorithm does not search all possibilities but decides whether to cutoff the entire current branch or continue searching and update the alpha, beta values and also the current best move.

8. Conclusions

This project was a good opportunity to learn the concepts of Artificial Intelligence like game trees, static evaluation function, Minimax algorithm and Alpha-Beta pruning.

Since it is very difficult to win against a player employing Minimax algorithm or Alpha-Beta pruning algorithm, we have increased the chances of that player using the concepts of Artificial Intelligence in the game of Kalaha.

During this project, studied the AI aspect in two-player games, i.e a computer player always makes intelligent moves by finding the best move for a given game position.

Programming the game was a good opportunity to learn Prolog. I followed the strategy which was introduced and described previously. Recommendations for how to program in prolog were used:

- For every procedure found out what facts we have and what can we use as an input and how to join the facts in the rule to get the output.
- Data objects and data structures were analysed.
- Operations with list were performed.
- Recursive rules were used.
- Issues of procedural semantics were analysed and understood.
- Arithmetical operations were performed.
- Input output mechanism was used.
- Control flow manipulations using cuts were used.
- Dynamic function included in the program.
- Used tracing mechanism to understand what steps the program performs while debugging.

9. References

- SWI-Prolog for implementing the game.
  http://www.swi-prolog.org/

- SWI-Prolg Documentation and Debugging documents.
  http://www.swi-prolog.org/pldoc/man?section=gensym

- Wikipedia
  http://en.wikipedia.org/wiki/Kalah

- Solving Kalaha by Geoferry Irving, Jeroen Donkers and Jos Uiterwijk (ICGA Journal)
  http://naml.us/~irving/papers/irving2000_kalah.pdf

- Solving(6,6) Kalaha
  http://kalaha.krus.dk/

- Programming in Prolog 4th Edition (Johnson Center)
  W.F Clocksin and C.S. Mellish

- Prolog for Programmers
  Feliks Kluzniak, Stanislaw Szpakowicz, Janusz S. Bien

- Play Kalaha online
  http://www.flashanywhere.net/en/puzzlegames/1450-mancala.html

- Demo Kalaha game on internet

  http://www.gamesmuseum.uwaterloo.ca/VirtualExhibits/countcap/pages/kalah.html

- Kalaha in java: An experiment in Artificial Intelligence and Networking
  Stephen O' Bryan & Christos Karavasileiadis

  http://diggy.ruc.dk/bitstream/1800/4946/1/Kalaha%20in%20Java%20An%20experiment%20in%20Artificial%20Intelligence%20and%20Networking.pdf

- Artificial Intelligence – A Modern Approach (Second Edition)
  Stuart Russell, Peter Norvig

10. Implementation Details for the Game of Kalaha

10.1 The Game Playing Framework

The play/1 predicate is the entry point to the game of Kalaha. The game is started using play(kalaha).

```
play(Game) :-
```

The play/2 predicate is the main game playing predicate which employs the game-playing framework discussed previously. It also checks for the game end condition and when found declares the winner of the game.

```
play([Own, OwnKalaha, Opp, OppKalaha], Player) :-
```

10.2 Initialization Code

The initialize/3 predicate is used for initializing the game board and the player who will make the first move.

```
initialize(kalaha, [[S, S, S, S, S, S], 0, [S, S, S, S, S, S], 0], player1) :-
        settings(stones, S).
```

10.3 Choose a move based on Algorithms

The chooseMove/3 predicate is used for getting the player specific algorithm and direct the flow of the program towards that algorithm implementation.

```
chooseMove(Player, Board, LN) :-
```

The algorithm/3 predicate performs the actual search operations for the right move depending upon the manual (user) algorithm or Alg (minimax or alphabeta) algorithm.

```
algorithm(manual, Board, LN) :-
algorithm(Alg, Board, LN) :-
```

10.4 Searching for a move

The performSearch/3 predicate performs the actual search for the right and legal moves according to the specified algorithm. It then sorts these moves and selects the best move from it. The performSearch/3 has three variants depending upon the algorithm used.

```
performSearch(manual, Board, List) :-
performSearch(minimax, Board, BestList) :-
performSearch(alphabeta, Board, BestList) :-
```

10.5 Check for Legality of a move

The checkLegalMove/2 predicate checks if the specified position would yield a legal move or not.

```
checkLegalMove([X, _, _, _, _, _], 1) :- X > 0.
```

10.6 Find all possible moves

The findAllMoves/2 is a very important predicate which finds all possible moves and also analyses whether that move will result in one more turn to the player or not. It internally implements the prolog's in-built predicate findall/3. It backtracks over goalMove/3 predicate and finds all possible moves and fills it in the Bag.

```
findAllMoves(Board, Bag) :-
```

10.7 Implementation of Minimax algorithm

The miniMax/5 is the predicate which has the implementation of minimax algorithm.
The inputs are Position P, state-space Depth D, points Pts, Value of the move V and minimum win value MW.

```
miniMax(P, D, Pts, V, MW) :-
```

10.9 Implementation of Alpha-Beta Pruning Algorithm

The alphaBeta/6 is the predicate which has the actual implementation of Alpha-Beta pruning algorithm. The inputs are Position P, state-space Depth D, Alpha value for the move, Beta Value for the move, Value of the move V and minimum win value MW.

```
alphaBeta(P, D, Alpha, Beta, V, MW) :-
```

10.10 Execution of a Move

The makeMove/4 is the predicate which performs the actual move for the player. It picks up the stones from the specified hole for the player and seeds it in the anti-clockwise manner and then performs the final checks as to game end stage has reached or not.

```
makeMove([NOwn, NOwnKalaha, NOpp, NOppKalaha], [Own, OwnKalaha, Opp, OppKalaha], MoreMoves
, StartHole) :-
        pickStones(StartHole, Picked, TOwn, Own),
        seedStones(Board, [TOwn, OwnKalaha, Opp, OppKalaha], TMoreMoves, Picked, StartHol
e),
        finalCheck([NOwn, NOwnKalaha, NOpp, NOppKalaha], Board, MoreMoves, TMoreMoves).
```

10.11 Game End condition check

The finalCheck/4 is the predicate which checks for the game end condition has reached or not and performs the final game end moves.

```
finalCheck([NOwn, NOwnKalaha, NOpp,  NOppKalaha], [[0, 0, 0, 0, 0, 0], OwnKalaha, Opp, Opp
Kalaha], noMoreTurn, _) :-
```

10.12 Game Playing Rules

The pickStones/4 is the predicate which performs the actual picking up of the stones from the specified hole by the player.

```
pickStones(1, Picked, [0|L], [Picked|L]) :-
pickStones(N, Picked, [X|L2], [X|L1]) :-
```

The seedStones/5 is the predicate which performs the seeding of the stones in the holes in an anti-clockwise direction. This predicate also makes sure that the game rules are obeyed and seeding and capture of holes are done in the right way.

```
seedStones([NOwn, NOwnKalaha, NOpp, NOppKalaha], [Own, OwnKalaha, Opp, OppKalaha], MoreMov
es, Picked, StartHole) :-
```

10.13 Switching between 2 players

The nextPlayer/2 is the predicate which does the switching of moves between the two players.

```
nextPlayer(player1, player2).
nextPlayer(player2, player1).
```