Board Game Assignment

Compute

02180 - Introduction to Artificial Intelligence F19



Caroline Ankjær Andersen *s175201*	Daniel Greibe *s175207*	Mathias Høgsgaard *s175211*

Introduction

For this assignment we have chosen to work with the board game kalaha. We have chosen to implement the full game with all the original rules. The players take turn taking seeds from a hole on their own side of the board, placing a seed in every following hole counter clockwise on the board except in their opponents “home” hole (kalaha) on the left side of the current player. If the player ends in a hole with other seeds in them the turn ends. If the player ends in an empty hole on their own side of the board the player can take the seed from their own hole and the seeds in the opposite hole (if there are any) and place it in their kalaha which is on the right side of the board from the current player’s view. The player gets an extra turn every time their turn ends in their own kalaha. The game ends when there are no seeds left on one of the sides of the board. The rest of the seeds are placed in the kalaha of the player that ended the game. The player that ends up with the most seeds in their kalaha wins the game. Our task with this game was to implement an AI for the game such that a human player can play against the AI. The board game and the AI was implemented in the programming language Java.

The game

Kalaha is a competitive, turn-based, zero-sum, multiplayer game of perfect information. The environment is deterministic, as the board is fully observable and the next state is determined by the current state and the move made by the other player. Since the boardgame is a two-player game i.e. multi agent environment where each player needs to consider the actions of the other and how this will affect the outcomes of its own actions, we need to consider this as an adversarial search problem. For these types of problems, the optimal algorithm would be the minimax algorithm as the goal of the AI is to get maximum points from the actions so it can lead to the best possible outcome for a goal state. For this algorithm we have chosen to apply alpha-beta pruning to make the search for the best possible action of the current state faster.

The state space of the game can be calculated using the following formula (Rovaris, n.d., 26):

$$p(k, n, m) = k \cdot \binom{n + m - 1}{m}$$

Where k is number of players, n describes the total number of holes and m is the total number of seeds.

Here our state space can be calculated as:

$$2 \cdot \binom{14 + 72 - 1}{72} = 1.48 \cdot 10^{15}$$

Since we're working on a big state space we have to limit our depth in the search three or in some other way limit the search tree. Since we can't hold the enormous state space in memory we only work on a part of the state space. In practise we do this by limiting how deep we go into the search tree to 10. That means we check the children of the state we wish to move from 10 levels deep. We have chosen 10 because the computation is almost instantaneously. If we choose 11 and above the computational time increase rapidly and we preferred the faster algorithm vs the better algorithm. Instead of limiting by the depth you can limit by the breadth as with AlphaBeta pruning.

The initial state s0 of kalaha would be how the board is set up at the beginning of the game. Therefore s0 would be when both players have no seeds in their own kalaha and there are 6 seeds in every other hole on the board. We called the first state in each turn the initial state in our program. We explicitly state this as it could be confusing.

Player(s) defines the player that has the move in the current state. There are only two players in this game so the player would be either player 1 or player 2. Or in this case where we have the AI play against human competitors it would be the AI and the human. From the initial state it would be the AI that has to make a move and then the players would take turns. The only time a player would get another turn was if the player got a seed into their kalaha.

Actions(s) defines the legal moves that the current player can make from a state s. In kalaha this would be which hole with x number of seeds the player can choose from to make their move. This has an upper limit of 6 and a lower limit of 1, if there is no legal actions the game is over (There are no seeds left in either side). We have considered a legal action as an action from a hole that is non-empty on your own side of the board.

Results(s, a) is the result of the move made from a state s. This is when a player made a move and what the new state looks like after this move was made.

Terminal-Test(s) is true if the game is over and false if its not. In our case this is when all of the holes are empty except for the players own kalahas.

Utility(s, p) defines the final value when the game has ended in a terminal state s for a player p. As we defined before, kalaha is a zero-sum game. Therefore the values that the utility function would return either 1+0, 0+1 or ½ + ½.

Implementation of the game and AI

In the computer we represent a state of the game by an integer array with 15 indices. Each indice contains a number that represents the number of seeds in the given hole. Index 0 - 5 is Player1's side, Index 6 is the Player 2's kalaha, Index 7-12 is Player 2's side of the board and Index 13 is Player 1's kalaha. We also added a 15th number (Index 14) which is 1 if the previous player got an extra turn and 0 if the player didn't.

It is not necessary to represent game states as belief states as all information is given, there are no random or nature elements in the game, only the choice of the two players which can be solved by the minimax algorithm. Since the minimax plays out the best possible move by the max player and the move that gives the max players the least amount of points, this is expected by the max player in the algorithm and the max player should win if the heuristic function is good enough. The algorithm relies on the heuristic value, since the chosen action is based on the heuristic of that action, if the heuristic is wrong, one action with a lower heuristic might be a better move but since it gets a lower heuristic value it is not chosen.

Since Kalaha is a multiplayer game we chose between the following algorithms: Minimax, AND-OR, Minimax with AlphaBeta cutoff and Expectimax.
Minimax is an extension of the AND-OR algorithm and AlphaBeta pruning is an even further extension reducing the calculations needed to find the best move.
Expectimax is used when the nature element has chance involved or said differently when the environment is non-deterministic.. We have chosen to use the Minimax with AlphaBeta pruning. We chose this one because Minimax alone can take a lot of time to calculate, and AlphaBeta is a strictly better version of Minimax.
Since there is no chance involved in our chosen game we did not choose Expectimax. Since Kalaha is a two player game it did not make sense to use either DFS, BFS, A* or Uniform Cost Search. We use the algorithm as described in the book where the AI is the max player, and we are the min player with the AI being the one starting the game.

For the minimax algorithm to work we need a heuristic or evaluation function. We decided for a very simple evaluation function once that takes into account how many points you have and how many points your opponent have. The value of a given state is the difference in points between you and your opponent. We could also have

used one that took only your own points into account and not your opponents but preferred using the difference in points.

We did not succeed in making a successful AI that could beat us every single time in Kalaha but our AI will beat us most of the time. We believe the heuristic function works like it should and the problem lies in our implementation not being very efficient and we are limited to having to searching with a depth of 10. We thought about going with the traditional Minimax but decided that we would use Minimax with AlphaBeta pruning since it a strictly better version of Minimax we saw no reason not to. We got it all to work very late in the project and we ran out of time to test different heuristics and therefore we don't have any test results using other heuristics. We did try out other search depths and as described earlier we ended up using a search depth of 10 because of the computational time.

If we had succeeded in implementing the Minimax algorithm and the AI had the starting move we shouldn't be able to win against it. Playing perfectly in Kalaha the starting player always wins.

Obviously our solution can be improved. Had we spent more time on the project we could've explored different heuristics and a bigger search depth. The first and biggest problem would be why our search depth is softly capped at a depth of 10. It's a soft cap since it is possible to have a bigger search depth but the computational time quickly gets very big.
Differently you could explore the sub-symbolic part of AI's and use reinforcement learning in Kalaha. You would provide the agent with the rules for Kalaha and have it play and explore different plays in Kalaha. As it became better you would have it play against it itself and make it better and better. Since Kalahas is classified as a 'solved' game where there exists a best possible combination of moves for all states, this would probably be considered an over-engineering of the problem and could probably be used as an exercise but not for research.

Bibliography

1. Rovaris, Gabriele. n.d. “Design of Artificial Intelligence for Manchala Games.” Edited by Pier Luca Lanzi. Master thesis, Scuola di Ingegneria Industriale e dell’Informazione.

2. Russell, Stuart, and Peter Norvig. 2016. *Artificial Intelligence: A Modern Approach, Global Edition*.