Kalah Game-Playing AI

Game Description

Kalah is a board game that involves moving stones from holes to try and collect as many stones as possible into your own Kalah. The board has two sides, each with 6 holes, and one hole on the end of either side of the board known as a Kalah. Each of the holes starts with 6 stones and the Kalahs start empty. To move, the players take turns picking a hole, taking all the stones out of that hole, and redistributing them into the following holes in a counterclockwise order, putting one stones in each hole including the own player’s Kalah. This can wrap around the board if enough stones are selected from a hole in a move, but stones are not placed in an opposing players Kalah during a move.

If a move ends with a stone placed in an empty hole, the player takes all the stones in the hole across the board from that hole and places them in their Kalah including the stone that landed in the empty hole. If a move ends with a stone placed in the players Kalah, that player gets another turn. If a player runs out of stones on their side, the opposing player takes all their remaining stones and puts them in their Kalah.

The game ends when more than half of the total stones are in someone’s Kalah, and that person would win the game.

6
6
6
6
6
6
6
6
6
6
6
6

Example Kalah Board

Heuristic Function

My Heuristic Function for this game is as follows. There are five different heuristics that are evaluated at any state of the game that are added up to provide a final score for that state of the game.

The first heuristic takes the difference between the AI's Kalah and the opponent's Kalah and multiplies it by a weight factor determined by how many pieces are in play at that point in the game. The function F_1 is as follows with A representing the AI's Kalah, and B representing the opponent's Kalah:

$$F_1 = (A - B) * (1 + \frac{A + B}{72})$$

The second heuristic counts the stones on the AI players side of the board as a positive integer to evaluate the number of stones the player can move, with a higher number being more favorable. The function F_2 is as follows with A(i) being the number of stones in hole i on the AI player's side:

$$F_2 = \sum_{i=1}^{6} A(i)$$

The third heuristic counts the stones on the opponents side of the board as a negative integer to evaluate the number of stones the opponent can move, with a higher number being less favorable for the AI. The function F_3 is as follows with B(i) being the number of stones in hole i on the opponent's side:

$$F_3 = -\sum_{i=1}^{6} B(i)$$

The fourth heuristic counts the number of non-empty pits on the AI player's side of the board. A higher number is more favorable because the player has more moves to choose from. The function F_4 is as follows with A(m) being the number of non-empty pits on the AI player's side of the board:

$$F_4 = A(m)$$

The fifth heuristic counts the number of non-empty pits on the opponent's side of the board. A higher number is less favorable because the opponent has more moves to choose from. The function F_5 is as follows with B(m) being the number of non-empty pits on the opponent's side of the board:

$$F_5 = -B(m)$$

The full heuristic function evaluates as $H = F_1 + F_2 + F_3 + F_4 + F_5$. However, I also have a heuristic for win conditions and tie conditions. If a player has more than 36 stones in their Kalah during a state, the Heuristic Function gives this state a value of 1000 for the AI player, and -1000 for the opponent.

If both players have 36 stones in their Kalah, the game is a tie, and the Heuristic Function gives the state a value of 0. An example of the Heuristic Function in action is as follows:

- Opponent opened game with move A2

Move	B1	B2	B3	B4	B5	B6
Heuristic Value	-2.111	-19.833	-3.000	-25.444	-24.625	-46.944

- AI selects move B1 as the best move from heuristic evaluation at depth 3
- Opponent moves A5

Move	B1	B2	B3	B4	B5	B6
Heuristic Value	19.000	4.500	4.500	-10.555	3.000	2.931

- AI selects move B1 as the best move from heuristic evaluation at depth 3
- Opponent moves A1

Move	B1	B2	B3	B4	B5	B6
Heuristic Value	-13.208	1.500	-7.875	-7.875	0.931	2.264

- AI selects move B6 as the best move from heuristic evaluation at depth 3

This Heuristic Function is effective because as demonstrated, it provides a more specific value than simply comparing the two Kalahs of the players. Although this is included as part of the function, it adds a weight to this value as well as four more heuristics that represent different aspects of the game that can affect player advantage, giving different values to states with equal scores that would otherwise be seen as identical.

Program Design/Experiments

My program design uses a minimax depth-first search with alpha-beta pruning and my Heuristic Function $H = F_1 + F_2 + F_3 + F_4 + F_5$. I ran two different experiments on my program to test both the efficiency of my program at different depths and the Heuristic Function effectiveness.

Experiment 1: Program Efficiency

The first experiment was to test my program efficiency at different depths of the minimax algorithm. To test this, I ran the program 5 times at depths starting at 3 and found the average time it took the AI to make a first move at each of these depths, since the first move is the most search intensive. I increased the depths until I reached a move time of over 1 second (1000ms), which is the time limit for the tournament. The results are as follows on my computer which has a CPU Speed of 4.10 GHz:

	Move Time (ms)							
Depth	3	4	5	6	7	8	9	10
Trial 1	2.99	4.01	10.02	30.02	63.42	155.58	344.89	825.15
Trial 2	1.99	4.56	12.46	34.07	77.45	187.34	424.48	1006.66
Trial 3	2.30	5.03	11.43	32.48	68.33	165.93	455.36	1047.36
AVG	2.43	4.53	11.30	32.19	69.73	169.62	408.24	959.72

These results support minimax search being O(b^m) time complexity, however, the alpha-beta pruning does help with the efficiency. They also show that a depth of 9 is optimal for the tournament, because with a time limit of 1000ms, a depth of 10 has a chance of running out of time, whereas with depth 9 there is a comfortable remainder of time. The more depths the algorithm searches, the better its move will be, therefore, because a depth of 10 takes too long, a depth of 9 is the best choice for the tournament.

Experiment 2: Heuristic Function Effectiveness

The second experiment was to test my heuristic against a naïve model that does not employ any heuristic when selecting moves. The naïve model simply selects a random legal move to play. I played 5 games against the AI that uses my Heuristic Function at depth 9 and an AI with the naïve model. I measured how many moves the game took and whether the AI won or lost. The results are as follows:

	Number of Moves and AI Win/Loss				
AI Player	Game 1	Game 2	Game 3	Game 4	Game 5
Naïve Model	24, Loss	15, Loss	22, Loss	19, Loss	17, Loss
Heuristic Model	19, Win	15, Win	20, Win	29, Loss	23, Win

These results support my Heuristic Function, as the naïve model lost 5 out of 5 games, whereas my Heuristic AI only lost 1 out of the 5 games. This is a significant improvement from the baseline naïve model, and demonstrates the effectiveness of implementing my Heuristic Function into the AI, as it became much harder to beat.