Nathan Ford

November 8, 2023

EE 562

Assignment 3 Report

# Game description – Mancala:

Mancala is a centuries-old two-player strategy boardgame. The version of mancala used in this implementation used two kalahs separated by two rows by six columns of holes. The kalahs start empty while the separating holes contain six stones each. Facing from one kalah to the other, the six holes on the right and the far kalah belong to each player.



Figure 1: Mancala board with four stones per hole instead of the six used in this implementation.

Two players take turns moving stones around the board. On a player’s turn, they select one of their holes to perform an action. The action begins by removing all stones from the hole. Once the stones are removed, the player visits each sequential hole in a counterclockwise fashion and places a single stone in each one. If the player’s kalah is reached (the kalah opposite to theirs), a stone is placed in the kalah. The path continues counterclockwise fashion through the opponent’s holes. The path skips the opponent’s kalah (the kalah closest to them) and goes back to the player’s holes. This circular path continues repetitively.

According to the Kalah rules stated in the EE562 10/06 lecture, different actions occur depending on where the final stone is placed:

* If the final stone lands in the player’s kalah, the player gets another turn.
* If the last stone lands in the player’s empty hole, take all the stones from the opponent’s hole directly across from it and put them plus the last stone in the player’s kalah.
* If all player holes are empty, the opponent places the remaining stones in their kalah.

The goal is to have more stones in the player’s kalah than the opponent’s. A player wins when they have more than 36 stones in their kalah. The game results in a tie if both players end with 36 stones in their kalah.

# Heuristic function design and justification:

The assignment instructions asked for multiple heuristic functions to be used to test AI game ability and timing. Two heuristic functions were used to determine a numeric value for how likely a given move is to result in a desired outcome – a simple and more complicated heuristic.

## Simple heuristic design and justification:

The simple heuristic function uses the game’s definition of how to win – the number of stones in each player’s kalah. The heuristic function takes the difference in stones between the player’s and the opponent’s kalah and uses that to determine the value of a move. The more stones the player has than the opponent, the better the move.

## Complicated heuristic design and justification:

The complicated heuristic function builds off the simple heuristic. The new function looks at possible captures and assigns positive weights to player captures and negative weights for opponent captures. The function also looks at moves that warrant another move – assigning positive weights to player captures and negative weights for opponent captures. The total weight is a combination of all three. The weight multipliers were determined by hand through trial and error.

# Overall program design, implementation, and results:

The program design and implementation were conceptually easy. The skeleton code provided meant only the AI to determine a move (hole to perform actions from) was to be completed. The AI used a minimax alpha-beta pruning search algorithm limited by a time and search depth limit to determine the move that results in the highest heuristic to play against an assumed-to-be optimal opponent.

## Program design:

The AI was designed as a minimax alpha-beta pruning algorithm to determine the best move based upon optimal opponent play. The minimax search uses a recursive depth-first search algorithm to examine potential board states given a chosen move and pick the board state most likely to lead to the best result (a hopeful win) and backpropagate the best move to the current board state.

## Program implementation:

To implement the search, a minimax function was called on the current state to find a move that maximizes the heuristic recursively until the time or depth limit is reached. Once a state is terminal (end state or terminated), the heuristic is backpropagated up the tree. The maximization function assumes an optimal player, so it plays as the opponent as well by calling a minimization function on the next depth of the search. The minimizer calls the maximizer and so on until a terminal state is reached. The search tree is generated by generating each state’s valid successors. A valid successor has at least one stone in the hole. The successors have their state updated to match what would happen after an input move happens.

## Program results:

Program testing was done on a 1.8GHz CPU. The program testing involved running the program at different search depths: 1, 3, 5, 7, 9, 11, 13, and 15 moves ahead of the current state. The test numbers were taken at the AI’s first turn to reduce the number of terminal states reached by a game ending. The collected data is shown in Figure 2.

Figure : Heuristic function depth testing. Blue represents the simple heuristic function and orange represents the complex heuristic function.

The orange trendline represents the complex heuristic function and the blue trendline represents the simple heuristic function. The simple heuristic function takes less time to complete than the complex heuristic function, but not by a large margin.

No widespread data was collected on AI win rate. A lack of internet connection when testing the AI meant all validity checks were done by a human and not against preset AIs. Neither the complex nor the simple AI could beat me by the end of testing.

Looking at the heuristic value and board setup of an example game against the complex AI shows some of its tendencies. The complex AI sometimes chooses to ignore the opportunity to score to go again or protect a hole from capture as captures have large weights and going again can chain moves. I rarely leave holes capturable for the AI, so that aspect was not as impactful.

A screen shot of a computer

Description automatically generated

Figure : Example game of a human player vs. the complex AI.

Overall, both AI versions are bad and unable to beat a human who learned Mancala for this assignment. The complex AI plays significantly better than the simple AI and scores higher against a human, but both AI versions are unable to win against a human. The complex AI outscores the simple AI which outscores the random move decisions.

# References:

All information in this report is derived from the lecture material for EE 562.