Alice Xia (mx48) & Quinn Beightol (qeb2)
CS 4701 Project Proposal

Playing Kalah with Minimax

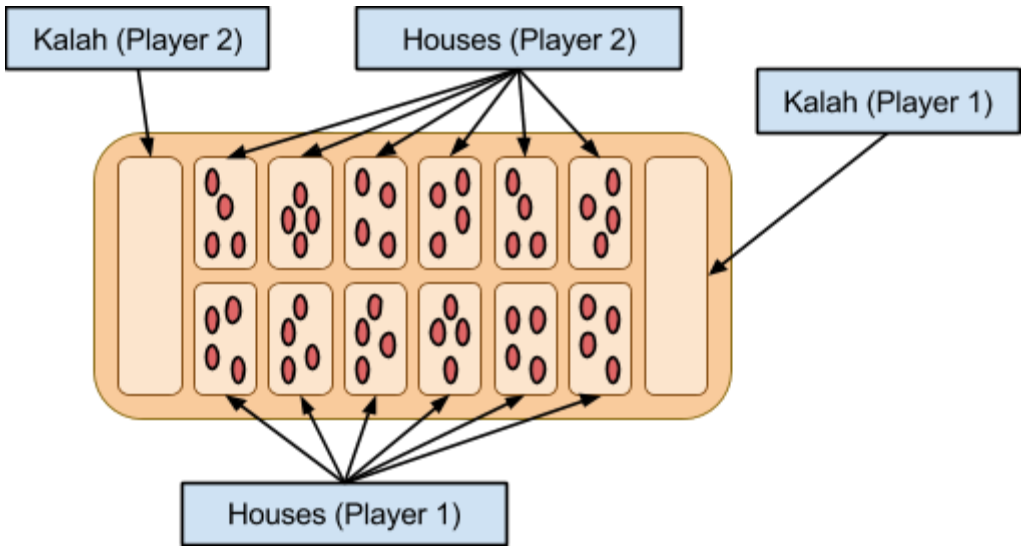
Description

We will implement several programs for playing Kalah (a variant of Mancala typically played in the US). We'll also include a simple GUI that will show the board state and allow a user to play against the bots we implement.

Specifically, we'll implement:

- a MiniMax class that works with any game representation and evaluation function
- a representation of the game of Kalah (including successor functions, state, termination conditions, etc.)
- an evaluation function for Kalah
- A GUI for monitoring/playing games
- A bot that makes random moves
- A bot that follows intermediate-level strategy

Overview of the game:



When the game begins, there are four stones (also called "seeds") in each "house". Player 1 begins the game by picking one of the piles of seeds and "sowing" them across the next four adjacent homes (and possibly one of the kalahs) in counter-clockwise order. From there, each player takes turns picking up a pile of n seeds and sowing them across the next n pits. Ultimately, the players work towards maximizing the number of stones in their respective kalahs.

There are a few additional rules that make the game more interesting (e.g. capturing opponent's stones, and restrictions on which stones can be moved), but we'll omit descriptions of those rules for brevity.

Approach

We'll create a program that uses minimax with iterative deepening and alpha-beta pruning to play the game effectively. Additionally, we'll write a state-evaluation function for board states. Generally, we'll write the evaluation function to reflect the number of seeds each player can expect to receive in their kalah based on the current game state. Features of the board examined will likely include:

1) the number of seeds in each kalah
2) whether the house closest to each player's kalah is empty
3) whether houses hold the exact number of seeds necessary to reach a kalah when sown

Plan for Evaluation

To measure our MiniMax bot's performance, we'll have it play against two other bots. One will simply make random moves, and beating this AI will confirm that our bot's performance is intelligent, in the sense that it is better than not using a strategy at all. The other will follow a set of heuristics (see this website for examples of rules the bot could follow) and will roughly represent the performance of an intermediate player. We plan to have the minimax bot repeatedly play against the others, and then check whether there is a statistically significant difference between the observed win-rate and a 50% win rate. As a side note, we'll also make sure that the bots alternate between going first and second, since the first player is guaranteed to win under optimal play.[1]

Because the first player is guaranteed to win, we'll also have the minimax bot play itself. Ideally, it will adopt an optimal strategy, and should win whenever it plays first. If this doesn't occur, however, we'll know that improvements to the bot are possible.

Finally, we'll look at the maximum depth by the AI and see how that depth changes when we turn various features on and off (e.g. we'll compare the depth the bot reaches with and without alpha-beta pruning). As a side note, the bot will have a fixed amount of time to make a decision--around 10-30 seconds. Consequently, we'll look at the maximum depth of analysis as opposed to comparing the quality of analysis at different depths.

Timeline

By November 1st, 2014: Kalah representation class (Quinn), evaluation function (Alice)
By November 15th, 2014: MiniMax implementation(Quinn), random bot (Alice)
By December 1st, 2014: Intermediate-level bot (Quinn), Finish GUI, any final changes (Alice)
We will jointly update the powerpoint and write up as we implement features of our project.

---

[1] http://naml.us/~irving/papers/irving2000_kalah.pdf (page 8; labeled 146 in the document header)