Jeremie Curfman

Michael Molina

CSE 415

Homework 3: Kalah

**Kalah**

Kalah game is a two player game where each player has 6 holes and 1 hole for storage. Each player’s storage is on their right side of the board. Each players holes starts with 6 stones in them. The object of the game is to collect the most stones in your storage. A turn consists of selecting one of your own holes to pick up all the stones in that hole and distribute them in each consecutive hole including your storage but not the opponents storage, in a counterclockwise direction. If your last stone lands in your own storage you get another turn and if your last stone lands in your own empty hole, you take all the stones in the opponent’s opposite hole and put them in your storage. When you run out of stones on your side the opponent takes all the stones left in their holes and puts them into their storage. There is a total of 72 stones on the board so when one player reaches a score of greater than 36 they automatically win. The game is over when one player has no more stones to grab from on their own side.

**Design**

This program implemented a minimax search with alpha-beta pruning. It would go down to a depth of 9 and stop searching and return the best possible move to make depending on the heuristic function given. When the ai or its opponent got another turn, the minimum value function and the maximum value function would call itself instead of the other to take an account the extra move each player gets so the right evaluation value would be pick to optimize the choice of the ai.

**Heuristic Function**

The heuristic function has three main components that contribute to the score which can be positive or negative. The first part of the function checks to see the difference in score. If the ai has more points then it will result in a positive number and vice versa.

Another factor in the function is the amount of empty holes on the board that are reachable by the opposing player. The function takes points away for any holes on your opponents side that they can reach on their next turn based on how many tokens they would get in return.

The amount of points added or subtracted depend on a defensive scalar. On the other end, if the opponent reaches 31 tokens the ai will play more defensively by increasing the amount of points that are subtracted from the function for any empty holes that are reachable on the opponents side. The ai will try to prevent the opponent from scoring over scoring itself. The defensive scalar will also give more points if there is a possibility of an extra turn on its next turn which gives more time to prevent the opponent from winning.

**Experiments**

1. Find a case where the AI could fill one of their empty holes with their last stone landing on it to benefit them the most.

2. Use a specific sequence of moves to observe what AI would do differently changing the heuristic function.

3. Find cases where AI could play more than one moves and see if it takes them.

**Results**

1. Figure 1 shows the state where the user gives the AI a chance to collect the stones at index 4 and figure 2 shows the that the user selects index 0 to allow it to take the opposite stones on the users slot at index 4, which it does.

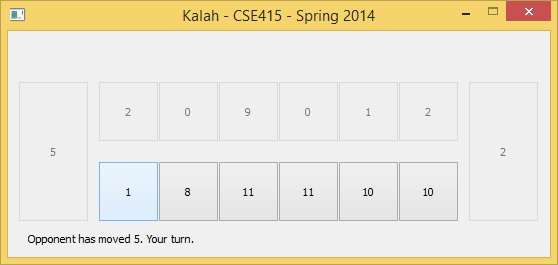


Figure 1.



Figure 2.

2. Building from a simple heuristic function to a slightly more complex version showed a change of behaviour where it would pick moves that would benefit it to win.

3. Playing exhaustively with the final AI, the AI would sometimes go on do up to around 7 moves in a row in one situation, thus the function to test if it should pick the hole that will lead it to another turn works.

|  |  |
| --- | --- |
| Depth | Average Time for One Move |
| 1 | < 0.0001 seconds |
| 3 | < 0.002 seconds |
| 5 | < 0.009 seconds |
| 7 | < 0.08 seconds |
| 9 | < 0.7 seconds |

**Improvements**

The main improvements of this ai would be its heuristic function. Being to give to value to certain states gets complicated the more factors you apply. For instance, a future improvement would be to find a proper value for states that lead to combos. Combos would include states where the ai could string together a few moves that would result in extra turns. This is difficult to do because the ai sees every move that results in an extra move as the same amount of value. So the first one will always be chosen. Another improvement would be to prevent the opponent from getting combos which would have similar difficulties as the previous improvement.

There is an offensive scalar in the code which was supposed to increase the amount of points that are given for any empty tiles that the ai can potentially reach on its next turn as the ai gets closer to 36 points. This is designed to make the ai slightly more aggressive, but failed because the ai would choose a good future state rather than choosing a good current state. For instance, if the ai could move to an empty tile and receive over 10 tokens from the opponent it would not do that, but instead move tokens in a way so that on its next turn it can move to an empty tile. This resulted in the ai getting very few points and was commented out for future improvements.

**Conclusion**

The AI in this project is relatively smart. It uses a minimax search with alpha-beta pruning and depth cutoff. Moves are chosen based on the difference in points between players, whether or not the opponent has the potential of scoring points from empty holes, and if a move results in an extra turn. Currently with a cpu speed of 3.1GHz the minimax search can go to depth 9 and stay under one second per move.