The University of Kansas

Technical Report

Searching and Game Playing: An Artificial Intelligence Approach to Mancala

Chris Gifford, James Bley, Dayo Ajayi,
and Zach Thompson

ITTC-FY2009-TR-03050-03

July 2008

Searching & Game Playing: An Artificial Intelligence Approach to Mancala

Chris Gifford, James Bley, Dayo Ajayi, and Zach Thompson

Electrical Engineering and Computer Science Department
University of Kansas, Lawrence, KS 66045
Correspondence: cgifford@eecs.ku.edu

I. Introduction

The game of Mancala is a two-player strategy game whose objective is to collect the most number of stones from a board of bins into a collection (home/Mancala) bin. The game begins with a predefined number of stones in each player's array of bins using a predefined number of bins on each side (including home bins). Details on the board setup, game instructions, and our approach to studying this game follow.

Glossary of Terms

Bin – The location on the board in which stones are stored.

Capture – The ability to end a move by placing a stone in an empty bin on your side of the board, allowing you to take that stone and all of the stones in the opponent's bin opposite that bin and place them into your home bin.

End Game – When one player has no stones left on his or her side of the board, that player takes any remaining stones from the other side and places them in his or her home bin.

Go-Again – The ability to end a move by placing a stone in your home bin.

Look-ahead – The numbers of moves used between two Mancala players.

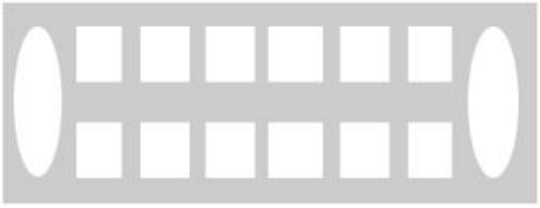
Mancala – Each player's home bin that is on each player's right on their side of the board.

Move – A player removes all of the stones from one of the holes on his side of the board, and then, moving counter-clockwise, places one stone in each hole (including his home hole, but not his opponent's) until all of the stones have been placed.

Win – The player with the greatest number of stones in his home when the game ends wins.

Board

The board consists of six (6) bins on each side, and a home position on the right of the bins. The board is laid out typically starting with four stones in each bin and home bins empty (0):



Basics of Play

A player removes all of the stones from one of the bins on his side of the board, and then, moving counter-clockwise, places one stone in each bin (including his home bin, but not his opponent's) until all of the stones have been placed. If the last stone placed is in his home bin, the player goes again. If the last stone is placed in an empty bin on the player's side of the board and if the opponent has stones in the opposite bin, then the player removes his stone and the stones in the opposite bin and places them in his home. If the opponent has no stones in the opposite bin, nothing occurs.

When one player has no stones on his side of the board, that player takes any remaining stones from the other side and places them in his home. The player with the greatest number of stones in his home wins. Keep in mind that we proved going first gives a definite advantage!

Problem Statement

To implement, test, and analyze the game of Mancala by incorporating different Artificial Intelligence (AI) techniques to its game play. This was accomplished via insertion of the techniques into pre-existing open source software (lacking AI) so as to evaluate overall heuristic performance.

Implemented Solution

We implemented the historic two-player game of Mancala. Our approach incorporated variables so that the number of starting stones per bin and other game characteristics could be altered for analysis. This allows our resulting solution the ability to be transformed into almost any desired variation of the Mancala family of games. In addition to being able to alter the variation of the game, we incorporated three versions of actual game play:

1. **Simulation/Batching Mode:** Computer vs. Computer
2. **Interaction/Testing Mode:** Computer vs. Human
3. **Demonstrational Mode:** Human vs. Human

An approach like this allowed a simplified batch processing of the test cases (Computer vs. Computer), while also allowing the group to test heuristics by hand (Computer vs. Human) as well as for presentation and demonstration of heuristics (Human vs. Human). To gather the various statistics that we present later in this report, we made use of a batch file to allow mass simulation of games and player situations in an unattended environment. Testing and results are discussed in later sections.

The main metrics for comparing these strategies are the number of “stones” in each Mancala (home bin) when the game ends and the number of moves it took to complete the game. More heuristic strategies also used are discussed in detail below. Given a game like Mancala, it is trivial to see that Mini-max, Alpha-Beta pruning, and DFS-style searches fit very well to the game setup. Thus, these approaches were considered into our design and incorporated into our final build.

We chose the game of Mancala because it is well-defined, involves searching and strategy, and has a reasonable search space for possible states. Being able to compare varying kinds of heuristics against one another in a tournament-style fashion, as well as against each other in

differing combinations, was beneficial not only for the game of Mancala but also for related and similar games. By running many combinations of heuristics, starting positions, and look-ahead depths, we effectively isolated the heuristics that perform the best in various categories. Following our analysis, we discovered a small set of dominant heuristics and played them against each other in round-robin and tournament-style fashions to determine which heuristic was the overall best. These results are covered in-depth later in the report.

We originally planned to incorporate a learning/evolution-based approach to the weights for the heuristics (i.e., how much of each heuristic contributes to the utilities). In theory, the weights would adjust themselves based on performance and eventually would converge to an optimal set of weights for the given heuristic sequence. Due to the difficulty and time required for this, we decided to leave this as an accomplishable addition to this project for the future. For more difficulties and decisions made along the course of development, see the *Testing and Developmental Difficulties* section.

We developed two distinct versions of Mancala: one for the Linux shell (batching and text-based play), one graphical user interface (GUI) developed with Microsoft Visual Studio 6.0 for Windows. Both of these solutions, their importance, use, and frameworks are discussed later in this report. Both versions were implemented in C++.

II. Solution Design

A. Heuristic Approaches

Overview of Heuristics

Based on the board setup, we have come up with various basic heuristics for choosing the next move. As design and testing proceeded, it became clear which heuristics were not needed as they were being included in other more dominant heuristics. The following seven heuristics were deemed the most important by the team and therefore were the focus of all statistical and strategic analysis.

- *H0:* First valid move (furthest valid bin from my home)
- *H1:* How far ahead of my opponent I am
- *H2:* How close I am to winning (> half)
- *H3:* How close opponent is to winning (> half)
- *H4:* Number of stones close to my home
- *H5:* Number of stones far away from my home
- *H6:* Number of stones in middle of board (neither close nor far from home)

As somewhat apparent in heuristics H2 and H3, if a player reaches *1 + half* of the stones then they are guaranteed to win. This intelligence was not included into the play of the heuristics so as to not taint how useful they are by themselves. If the goal of this research was to develop an unbeatable Mancala AI player, it would have been included. However, our goal was to develop and analyze various heuristics against each other and in combination through the searching techniques of Mini-max and Alpha-Beta pruning. Therefore, it was left out of our development and analysis.

Heuristic Comparison

The above heuristics were run through a series of simulation games (Computer vs. Computer) to separate out the best single heuristics, best combinations of those heuristics, and the overall best

sequence of heuristics to use for optimal results. To determine the best overall heuristic/sequence, we first separated the best heuristics and then ran them against each other in a round-robin tournament. Winners moved on and played other winners until the dominant heuristic was apparent. In other words, the heuristics were used to assign utility to nodes during the search.

> ***Look-ahead***
> *Represents 1-ply of adversarial game search, or the number of moves used between two Mancala players:*
> *E.g. A look-ahead of 2:*
> *Max makes 1 move, and Min makes 1 move, in that order, to find the best move for Max.*

Via use of a batch file, we were able to run all of our comparisons in one execution. Without getting too much into implementation details yet, each player was given a set of heuristics to use for that particular set of games. Each player was also provided a look-ahead depth to use during that set of games. For each execution (set of games), we were also able to provide the number of games to play going first, and then going second. Because strategy is so important in two-player games, Computers given the ability to move first gives them a better chance of winning. By varying these values, all meaningful combinations of heuristics can be simulated.

Statistics

Each set of games generated various statistics about strategy and winning/losing tendencies for the players and their respective look-ahead and heuristic array. There were two different configurations that the batching could use: AI Computer vs. Random, or AI Computer vs. AI Computer. The following are the statistical categories that were gathered for the AI Computer vs. Random configuration:

> *H1 H2 H3 H4 H5 H6* //Heuristic Array (see above Heuristics section)
> *Lookahead* //Look-ahead Depth
> *Win%* //Win Percentage
> *Win%FirstMove* //Win Percentage Going First
> *Win%SecondMove* //Win Percentage Going Second
> *Avg_Stones* //Average number of stones in Mancala at end
> *Avg_Margin* //Average stone margin in Mancalas
> *Win_Margin* //Average Win Margin
> *Loss_Margin* //Average Loss Margin
> *Avg_Captures* //Average number of captures per game
> *Avg_GoAgains* //Average number of go-agains per game
> *Avg_Moves* //Average number of total moves per game

The following are the statistical categories that were gathered for the AI Computer vs. AI Computer configuration. The main difference is that it keeps track of all of the above statistics for both players:

> *H1 H2 H3 H4 H5 H6* //Heuristic Arrays for both players
> *Lookahead* //Look-ahead Depth for both players
> *Win%* //Win Percentage for both players
> *Win%FirstMove* //Win Percentage Going First for both players

```
Win%SecondMove      //Win Percentage Going Second for both players
Avg_Stones          //Average number of stones in Mancala at end (both)
Avg_Margin          //Average stone margin in Mancalas for both players
Win_Margin          //Average Win Margin for both players
Loss_Margin         //Average Loss Margin for both players
Avg_Captures        //Average number of captures per game (both)
Avg_GoAgains        //Average number of go-agains per game (both)
Avg_Moves           //Average number of total moves per game (both)
```

B. Searching and State

Searching

Mini-max and Alpha-Beta pruning methods of traversing and searching the state space were incorporated. We first developed Mini-max as our searching technique with a basic heuristic being the most stones in our home bin. As we incorporated the look-ahead depth, we noticed that Mini-max was taking a substantial amount of time (~20 seconds) to pick a move with look-aheads larger than 10. As the look-ahead increased, it was observed that the computation time involved in expanding the search space increased nearly exponentially. As a result, Alpha-Beta pruning was used to reduce this computation time by ignoring parts of the game tree that did not improve on the current best found move, making it a more efficient overall search. After implementation, we noticed immediate results. Thus, the searching technique employed in our solutions is Alpha-Beta pruning using Mini-max as the base. Below is a brief description of these two techniques:

> ***Mini-max***
>
> *A technique used to open up the search space for the game. This algorithm returns the best move given the best utility at the end of a search depth. The move selected is based on the assumption that the opponent is using the same set of heuristics.*
>
> ***Alpha-Beta Pruning***
>
> *A technique to reduce the number of nodes evaluated in the search tree by the above Mini-max algorithm. It stops completely evaluating a move that a player can make when at least one reply has been found that proves the move to be worse than a previously examined move. Since it clearly doesn't benefit the player to play that move, it need not be evaluated any further. This algorithm reduces computation time significantly, yet does not affect the search result in any way.*

As previously stated, we originally planned to incorporate the refining of our evaluation function. We envisioned this being done by evolving the evaluation function through "breeding" of the evaluation functions which are shown to perform the best. In essence, the weights for each set of heuristics would be combined to form a new set of weights through "breeding". Then the refining process continues until they converge on values.

After reading papers on related approaches to the game, we realized that this would be an extremely difficult task requiring a substantial amount of time and work. Because of this, we decided to omit the breeding of weights and keep it discrete for contribution of weights. By discrete, we mean 0 for no use and 1 for entire use. There are no fuzzy values for the weights of heuristics, and we left it as a good future addition to the work.

To determine the move to make, the heuristics are applied and each returns a utility value that represents how good that move is based on the current board configuration. By using combinations of heuristics, the utility values they return get summed to give the total utility of a move given that particular set of heuristics.

$$TotalUtility(S_k) = \sum_{i=0}^{6} Weight(H_i) * Utility(H_i)$$

To mimic human game play and to insure Computer players do not look far ahead into the game (to keep the Computer moves interesting and not predictable), we enforced a searchable depth limit – say a 6 move look-ahead. This acts as a time-based approach in that Human players cannot count the number of stones in each bin, making the move time-limit usually short. By forcing Computer players to do this, it is giving us a "good enough, soon enough" result, and also mimics how a human actually plays the game by only being able to look ahead a few possible moves. As will become apparent after viewing the results, with the exception of a couple odd cases, a larger look-ahead directly translates to more ending stones and therefore more wins.

State

The state of the board was represented concisely as a 1D array with each player's set of bins in regions of the array, along with the heuristic array, which move it is, and the statistics variables. Each node in the search/game tree was represented by the board state at that point, the depth of the node, the Mini-max move, and evaluation function information. This provided enough information to efficiently evaluate states and compare heuristics.

III. Testing & Development Difficulties

A. Testing

To perform the tournament-style and heuristic combinations comparison and testing, we employed batch processing. Batching allowed many tests in an unattended environment, where the output was computed during program execution and printed out into a results file. The output was formatted so as to allow us to simply copy and paste them into Microsoft Excel or Open Office and immediately sort and generate graphs from the results. It also provided us with a very distinct way of evaluating if the games were being played correctly and if the searching and heuristics were working as desired. There were many instances where viewing the statistical results told us exactly what was going wrong, allowing an easy fix.

Other forms of testing were by using the Computer vs. Human GUI mode of the software. This would allow us to make certain moves and see if the heuristic that we were playing against would actually pick the move that its function told it to (the one giving it the best utility). The GUI version was also a key aspect in testing and gave us an opportunity to visualize the moves as they took place, along with the sequence of moves performed by each player for the entire game.

It should be noted that we implemented and tested the usual variation of the game: 4 starting stones per bin with 6 bins on each side (in addition to the single home bin per player). Another factor/variable could have been to increase the number of starting stones per bin from 4 to, say,

6. Although the moves would be different, the heuristic calculations might return different numbers, but all in all we would expect extremely similar results. We feel like that variable would not increase what we would find out about the game itself or the heuristics we chose to analyze. Thus, we left this as future work that could be done on this project. Both the Linux/batch and Windows/GUI versions implement it so that it is easy to change. In the GUI, this option is actually implemented so you can change the number of starting stones per bin via the Options dialog box. For the scope of this project, we felt like this would have been overkill but left it implemented in the GUI version so that anyone who feels intrigued could change the number and play the game to see how the heuristics perform with a different starting number of stones.

B. Problems Encountered

We ran into design decisions on how to represent board state, and therefore changed not only our board representation but also our entire state representation scheme a couple of times. With a software project increasing in size involving multiple team members, small changes in the code became less apparent as development progressed. Without the use of a version control system, sharing the most updated copy of the code became a challenge at times.

Early in Mini-max development, we discovered that we were thinking about the utilities in the wrong manner. We were initially not trying to maximize our utility and minimize the opponent's utility, but were instead looking at the board from the wrong perspective when deciding on moves. After some discussion this was resolved so that each agent analyzed the board using their own point-of-view/perspective.

In addition to the time complexity of a search depth and the Mini-max-to-Alpha-Beta performance improvement, most of the troubles were associated with getting Mini-max originally working correctly. Slowly stepping through the program and debugging allowed us to find out what the problems were and fix them accordingly.

IV. Statistical and Game Play Results

A. Explanation of Results

As previously described, one configuration in the batch file represented the AI Computer player being given a set of heuristics and look-aheads to use during the set of games playing first then playing second. The Random player simply chose a random, valid move for each of its moves and therefore had no special heuristic or intelligence. By using a Random player, it allowed us to compare heuristics and combinations of them against each other based on how they performed against the Random player in 100 games starting first and 100 starting second. We ran this many games so that it would give the percentages a chance to level out and converge to a stable average.

The other configuration in the batch file represented both AI Computer players being provided a set of heuristics and look-aheads to use during the set of games playing first then playing second. This did not use a Random player at all, but instead faced heuristics against each other to see how they performed when going up against each other. In such a case, we didn't need to run hundreds of games because they would play each other the same each game unless the look-ahead values or heuristic arrays were changed.

Heuristics

H0: Chooses first valid move (going left to right)
H1: (My Mancala – Opponent's Mancala)
H2: How close I am to winning
H3: How close my opponent is to winning
H4: # of stones close to my home (1/3)
H5: # of stones away from my home (1/3)
H6: # of stone in the middle (1/3)

The following statistical results and analysis is not a self-proclaimed "solution" to the game, and we are not stating that we have "figured out" the game of Mancala completely. There are many other ways to analyze its heuristics and strategy. We chose a subset of the plethora of heuristics for the game: the ones we thought were most important and interesting. Although our analysis and results are quite thorough, there is potentially much more analysis to do to determine the optimal way to play the game and which heuristics are the most beneficial in certain game situations. Keep this in mind when reading the following sections. See the later subsection on *Look-ahead Analysis* for the results of varying the look-ahead.

B. Results for AI Computer vs. Random

The results below reflect performance and statistics relating to all possible heuristic combinations being played against the Random player. The analysis in this section will mostly focus on average statistics per move. Analyzing the data with respect to the opponent's statistics could not be done, due to the fact that the data for the player making random moves was not tracked. The average results on a per-move basis will show the characteristics of the AI during each move, uncovering the overall strategy of the AI. Once the average data per move is sorted, it will be apparent which combinations of heuristics result in the best strategies. This also serves to verify that the heuristics are performing their job correctly.

Overall Analysis

The following data was gathered from the raw analysis of P1 vs. Random (AI with heuristics vs. AI making random moves) sorted by win percentage. *Figure 1* contains the count of the number of AI's (Player 1's) that used the particular heuristic with a particular percentage of winning. *Figure 2* shows the same data, only this time the count of the number of AI's who used the particular heuristic is the percentage of AI's who used the particular heuristic with a particular percentage of winning.

H1	H2	H3	H4	H5	H6	Win%
27	18	18	16	15	16	1
5	6	6	4	5	3	0.99875
0	1	1	1	1	1	0.9975
0	0	1	0	1	0	0.99625
0	1	0	0	1	1	0.995
0	2	1	2	2	2	0.99375
0	1	0	1	0	1	0.99125
0	1	0	0	0	0	0.98875
0	1	0	0	1	0	0.9875
0	1	0	0	0	1	0.98625
0	0	1	0	0	1	0.985
0	0	1	1	1	0	0.98
0	0	1	1	0	1	0.9775
0	0	1	1	0	0	0.97375
0	0	1	1	1	1	0.95875
0	0	0	1	0	0	0.65625
0	0	0	0	1	1	0.60875
0	0	0	1	1	0	0.565
0	0	0	1	0	1	0.52875
0	0	0	0	0	1	0.52125
0	0	0	1	1	1	0.5175
0	0	0	0	1	0	0.51625
0	0	0	0	0	0	0.35125

Figure 1

H1	H2	H3	H4	H5	H6	Win%
0.84375	0.5625	0.5625	0.5	0.46875	0.5	1
0.555556	0.6667	0.66667	0.4444	0.555556	0.333333	0.99875
0	1	1	1	1	1	0.9975
0	0	1	0	1	0	0.99625
0	1	0	0	1	1	0.995
0	0.6667	0.33333	0.6667	0.666667	0.666667	0.99375
0	1	0	1	0	1	0.99125
0	1	0	0	0	0	0.98875
0	1	0	0	1	0	0.9875
0	1	0	0	0	1	0.98625
0	0	1	0	0	1	0.985
0	0	1	1	1	0	0.98
0	0	1	1	0	1	0.9775
0	0	1	1	0	0	0.97375
0	0	1	1	1	1	0.95875
0	0	0	1	0	0	0.65625
0	0	0	0	1	1	0.60875
0	0	0	1	1	0	0.565
0	0	0	1	0	1	0.52875
0	0	0	0	0	1	0.52125
0	0	0	1	1	1	0.5175
0	0	0	0	1	0	0.51625
0	0	0	0	0	0	0.35125

Figure 2

Figure 3 shows the data from *Figure 1*, exhibiting how more AI's used a wide combination of many heuristics to result in a better winning percentage. In particular, it shows how H1 (comparing one's score with the opponent's score) is a large contribution to winning since H1 is used more with AI players that performed well and not as much with AI's that won less. H2 (how close one is to winning) also contributes to success in the 98% and greater range with AI's losing more who don't use H2. H3 (how close one's opponent is to winning) was a contributor to AI's winning 95% of the time and above.

H4, H5, and H6, heuristics which concentrate on the layout of the board, do contribute to winning success when combined with other heuristics, but left by themselves do not perform very well. In combinations by themselves, H4 performed the best; H5 with H6 was next, then H4 with H5, H4 with H6, H6, H4 with H5 and H6, and finally H5.

Figures 4 and *5* both depict charts gathered from *Figure 2*. *Figure 4* shows, using cones, the percentage of AI's at each winning percent that used each heuristic. *Figure 5* gives the same results at a different viewing angle. A full cone represents 100%, and no cone represents 0%.

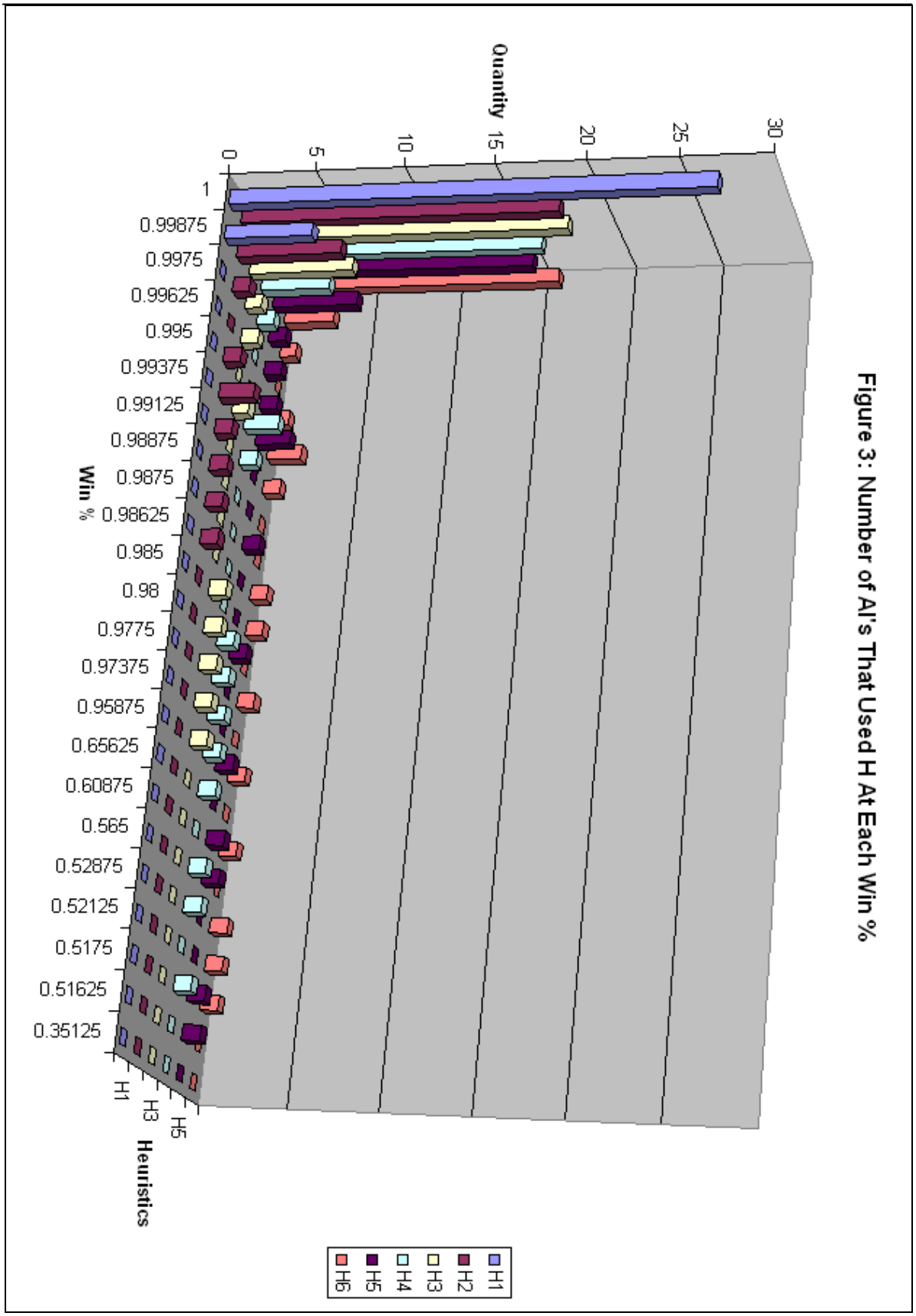


Figure 3: Number of AI's That Used H At Each Win %

Figure 4: Percentage of AI's Using Heuristic At Win %

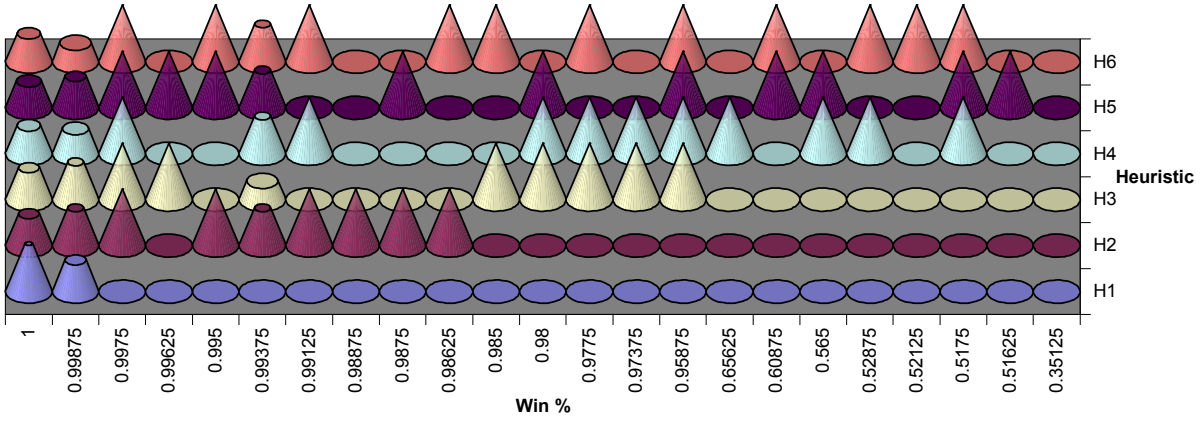


Figure 5: Percentage of AI's Using Heuristic vs. Win %

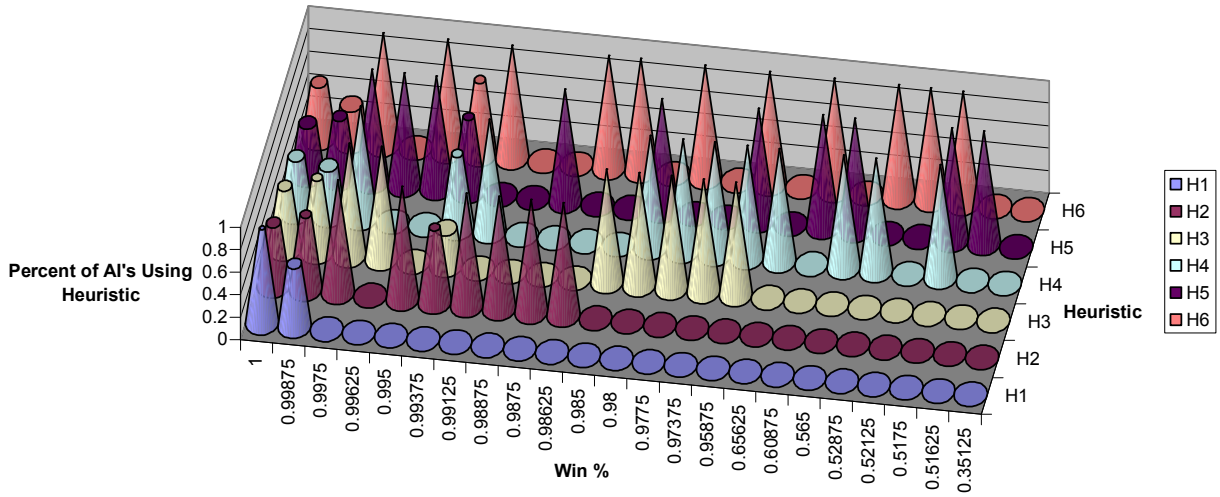


Figure 6 contains the data for sorted average stones per move. *Figure 7* shows a simple chart expressing the heuristics of each AI visually as a function of the average stones per move. Looking closely at H1 and H2, the chart shows that using one or the other or both results in a move which yields more stones on average compared to not using H1 or H2. The chart also shows that in order to have a good stone-per-move ratio one must use H1 if not using H2, and H2 if not using H1. This is exactly what should be happening, as a player using H2 will attempt to find the move which results in getting closest to winning (having one more than half the total number of stones in the home bin). Similarly for H1, the player will look for the move which puts itself farthest ahead of his opponent. This means that each move the player will be looking for the move which results in the largest gain. This is verified in the graph where the AI with the best average stones per move is the AI that ***only*** used H2. The next best AI used a combination of H1, H2, and H4. This makes sense in that the player used H2 and H1, which maximize the amount of stones in the home bin, but also H4, the move which results in a board configuration with the most stones close to home. This could have possibly allowed the player to end the game by clearing his side of the board, resulting in a final move which obtained a large amount of stones. It also could have resulted in captures. It is also obvious that H3 does not play a large role in average stones-per-move as it only relies on making moves which keep the opponent from obtaining stones in his or her home bin.

H1	H2	H3	H4	H5	H6	Stones / Move		H1	H2	H3	H4	H5	H6	Stones / Move
0	1	0	0	0	0	1.812915422		1	0	1	0	1	0	1.494733
1	1	0	1	0	0	1.758720092		1	0	0	0	0	1	1.491997
1	1	0	0	0	0	1.722028562		0	1	1	1	0	0	1.477458
1	1	0	0	1	0	1.719877814		1	0	0	1	0	1	1.473705
0	1	0	0	1	0	1.714328022		0	1	1	0	0	1	1.469149
1	1	0	1	0	1	1.699483878		0	1	1	1	0	1	1.468076
1	1	0	1	1	0	1.685279751		1	0	0	1	0	0	1.463005
1	0	0	0	0	0	1.681744047		0	1	1	0	1	1	1.4526
1	1	1	1	0	0	1.677694018		1	0	0	1	1	1	1.441361
0	1	1	0	0	0	1.665020134		1	0	1	1	0	1	1.436207
1	1	1	0	0	0	1.653737244		0	1	1	1	1	1	1.435292
1	1	1	1	0	1	1.646467333		1	0	1	1	1	0	1.434596
1	1	0	1	1	1	1.631935553		1	0	0	0	1	1	1.420888
0	1	0	1	0	0	1.618938803		1	0	1	0	0	1	1.409993
0	1	0	1	1	0	1.617834395		1	0	1	0	1	1	1.408958
1	1	1	1	1	0	1.611908365		1	0	1	1	1	1	1.399436
1	1	0	0	0	1	1.60588799		0	0	1	0	0	0	1.317492
1	0	1	0	0	0	1.593287013		0	0	1	0	1	0	1.16435
1	1	0	0	1	1	1.588946459		0	0	0	0	0	0	1.102683
1	1	1	0	1	0	1.587157862		0	0	1	0	0	1	1.09321
1	1	1	1	1	1	1.580826293		0	0	1	0	1	1	1.050529
0	1	0	0	0	1	1.573801144		0	0	1	1	0	0	1.03068
1	1	1	0	1	1	1.559069909		0	0	0	0	1	0	0.963177
1	1	1	0	0	1	1.553706888		0	0	1	1	0	1	0.935416
1	0	0	1	1	0	1.546600905		0	0	1	1	1	0	0.92717
0	1	0	1	1	1	1.536874733		0	0	0	0	1	1	0.895278
0	1	1	1	1	0	1.532111021		0	0	1	1	1	1	0.867511
0	1	1	0	1	0	1.529911055		0	0	0	0	0	1	0.858704
1	0	0	0	1	0	1.525379808		0	0	0	1	0	0	0.834206
1	0	1	1	0	0	1.520037977		0	0	0	1	1	0	0.743675
0	1	0	0	1	1	1.519610953		0	0	0	1	0	1	0.727935
0	1	0	1	0	1	1.494809869		0	0	0	1	1	1	0.702501

Figure 6

Figure 8 contains the data for sorted average captures per move. The accompanying chart in *Figure 9* shows the AI's for each average capture per move. H1 and H2 are required to have a high rate of capture; this is a side effect of their definition, which is to find the move which results in the largest gain of stones for the player. One of the largest gains of stones is a capture move. At a minimum a capture will result in two stones being placed into the player's bin. The difference between *Figure 7* and *Figure 9* is that the AI's which have the best average captures per move factor in H4, H5, and H6. This is required since capturing requires that there be an empty bin on the player's side of the board. By factoring in quantities of stones in its move decision, it allows the player to open up areas of his side of the board, resulting in capture moves. Using H4, H5, and H6 alone will not prove successful in capturing since searching using only those heuristics does not take into account obtaining any stones in his or her home bin. It is worth noting that H2 alone ranked in the top 15%, whereas H1 alone barely made the top third. For the upper 50% of the data, H3 required H2 and H4 to be successful.

Figure 10 contains the go-again per move data for each AI, and *Figure 11* is its visual representation. By themselves, H1 performed the best, then H2, then H3—all being in the top third of the results. H1 performs the best since it looks for moves which will result in the greatest difference between the player and his opponent. Going again results in a minimum of obtaining one stone with the possibility of obtaining another in the resulting turn, and the possibility of setting up a series of moves for a capture. H1 is best at this since going again keeps the opponent's score the same, and any subsequent scoring only increases the utility value of H1 at each node. H2 does not find as many moves which allow it to go again because it relies on finding moves which result in the largest gain for the player without looking at the differences between the players' scores. H3 ranks well in this analysis since it looks at how to keep the opponent's score as low as possible. One way to do this is to not let the opponent have a move—instead, go again! Another trend in the data is that H4 is not used as much as H5 or H6 in the upper third of the data. This is because H3 keeps stones close to home. When stones are close to home and there's a lot of them or they're in large piles, it is not possible to go again since the last stone dropped will not be in the player's home but somewhere far from it. A player has a better chance of going again (dropping the last stone in his or her home bin) if they pick up a large group of stones from the middle of the board or the one-third of the board farthest from home.

Figures 12 and *13* correspond to the combined data of average captures plus average go-agains per move for each AI. One overwhelming consistency is the use of H1 or H2 or both in order to effectively make a capture move or go again. Heuristics H4, H5, and H6 have no bearing on the upper 10% of the data. Alone, H1 and H2 rank very near the top and H3 falls just short of the 50% mark. H3 however makes up, in combination with H1 and/or H2, nearly 75% of the top 15%. Alone, H5 performs best since having your stones farthest from home would result in the greatest possibility for a capture or a go-again. The majority of AI's that used H4 fell in the lower 50% of the data since having stones close to home does not allow for many captures and also not very many go-agains, unless the stones in the bin are in small numbers, which is why H4 alone performs better than H6 which comes in second-to-last.

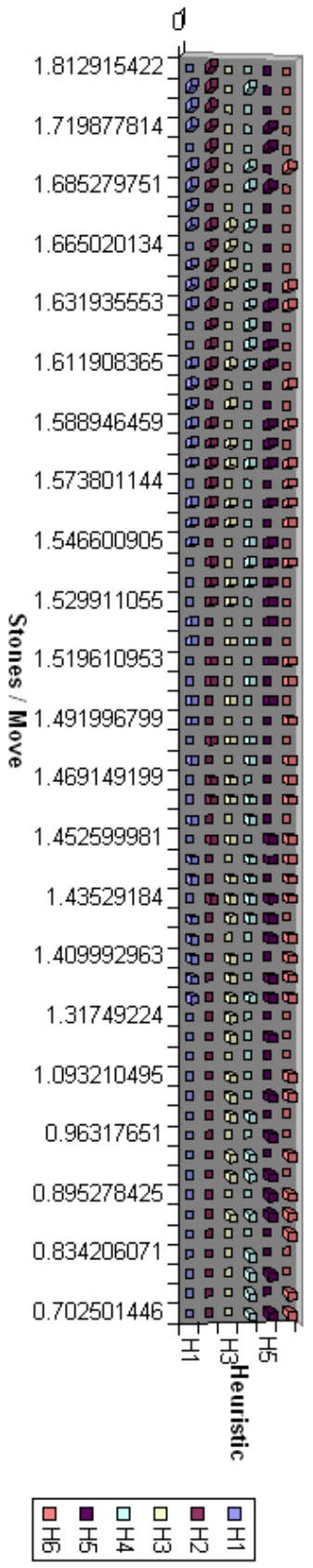


Figure 7: Average Stones / Move

H1	H2	H3	H4	H5	H6	Captures/Move		H1	H2	H3	H4	H5	H6	Captures/Move
1	1	0	1	0	0	0.226578265		1	0	0	1	1	0	0.184522296
0	1	0	1	0	0	0.223442203		1	0	1	1	1	1	0.184362526
1	1	0	1	0	1	0.223319829		0	1	1	0	0	1	0.182786683
0	1	0	1	1	1	0.22071398		1	1	1	0	0	0	0.182080462
1	1	0	1	1	0	0.220137655		1	1	1	0	0	1	0.180337817
0	1	0	1	0	1	0.219915497		1	1	1	0	1	0	0.179593315
0	1	0	0	0	0	0.213366669		1	1	1	0	1	1	0.179012032
0	1	0	0	1	0	0.209382219		1	0	0	0	1	1	0.177943552
0	1	0	1	1	0	0.208418721		0	1	1	0	1	1	0.177447468
1	1	0	1	1	1	0.208013231		1	0	1	0	0	0	0.175845509
1	1	1	1	0	0	0.206844189		1	0	0	0	1	0	0.175753265
1	1	1	1	0	1	0.204830815		1	0	1	1	1	0	0.175483383
0	1	0	0	0	1	0.204465464		0	1	1	0	1	0	0.174231258
1	0	0	1	0	0	0.19993158		1	0	1	0	0	1	0.169082806
1	1	0	0	1	0	0.19928725		1	0	1	0	1	1	0.164998371
0	1	0	0	1	1	0.198646791		1	0	1	0	1	0	0.161090519
1	1	0	0	0	1	0.197508518		0	0	0	0	0	0	0.127586699
1	1	1	1	1	0	0.197463863		0	0	1	0	1	0	0.125124131
1	0	1	1	0	0	0.197081751		0	0	1	1	0	0	0.123951035
1	0	0	1	1	1	0.194927219		0	0	1	0	0	0	0.123595506
0	1	1	1	0	0	0.194284306		0	0	1	0	0	1	0.116683357
1	1	0	0	0	0	0.193522996		0	0	1	1	0	1	0.115957638
1	1	1	1	1	1	0.191891613		0	0	1	0	1	1	0.115506451
0	1	1	1	1	1	0.191752976		0	0	1	1	1	0	0.104455287
1	0	0	0	0	0	0.191203653		0	0	1	1	1	1	0.09948725
1	0	0	1	0	1	0.191070548		0	0	0	0	1	0	0.098696642
0	1	1	1	0	1	0.189481445		0	0	0	0	1	1	0.091848401
0	1	1	0	0	0	0.18738255		0	0	0	0	0	1	0.089856213
1	1	0	0	1	1	0.187051866		0	0	0	1	0	0	0.086263239
0	1	1	1	1	0	0.185980551		0	0	0	1	1	1	0.070018797
1	0	1	1	0	1	0.18591954		0	0	0	1	0	1	0.069159017
1	0	0	0	0	1	0.18507403		0	0	0	1	1	0	0.068513985

Figure 8

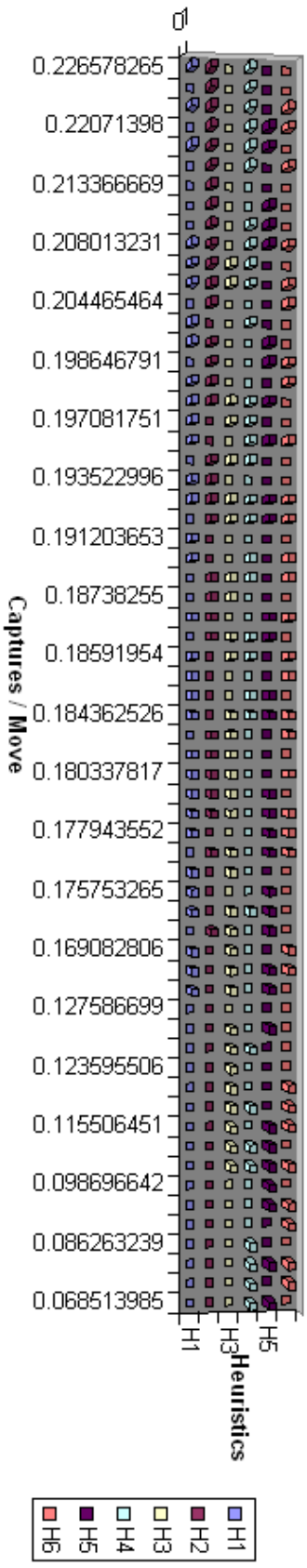


Figure 9: Captures Per Move For Each AI

H1	H2	H3	H4	H5	H6	GoAgain/Move		H1	H2	H3	H4	H5	H6	GoAgain/Move
1	1	0	0	0	0	0.482795457		1	0	0	1	1	0	0.36579041
1	1	1	0	0	0	0.475236416		0	1	0	0	1	1	0.361454699
0	1	1	0	0	0	0.472536913		0	1	1	1	1	0	0.359754862
1	0	1	0	0	0	0.469653622		1	0	1	1	0	0	0.358135119
1	0	0	0	0	0	0.462487768		0	0	1	0	1	0	0.357373386
1	1	1	0	1	0	0.462099653		0	1	0	0	0	1	0.350472943
1	1	1	0	0	1	0.453777688		1	0	1	1	0	1	0.337691571
1	0	1	0	1	0	0.452936012		1	0	1	1	1	1	0.334585583
0	1	0	0	0	0	0.449417647		0	1	1	1	0	0	0.323084504
1	1	1	0	1	1	0.443316241		0	1	1	1	1	1	0.319959346
1	1	0	0	1	0	0.442074895		0	1	1	1	0	1	0.31884984
1	1	0	0	1	1	0.438007013		1	0	0	1	0	1	0.318598915
1	1	1	1	1	0	0.431038981		1	0	0	1	0	0	0.318101847
1	0	1	0	1	1	0.429861725		1	0	0	1	1	1	0.313421937
0	1	1	0	1	0	0.426480305		0	0	1	0	1	1	0.313335394
1	1	0	0	0	1	0.425574957		0	1	0	1	1	0	0.308335641
1	0	0	0	1	0	0.425435699		0	0	1	0	0	1	0.295293411
1	1	1	1	1	1	0.424376674		0	1	0	1	0	0	0.271485899
0	0	1	0	0	0	0.424037074		0	1	0	1	1	1	0.269559641
1	1	1	1	0	0	0.418786268		0	1	0	1	0	1	0.250691148
1	1	0	1	1	1	0.415653009		0	0	1	1	1	0	0.247195413
1	0	1	0	0	1	0.41543514		0	0	1	1	1	1	0.221583675
1	1	1	1	0	1	0.406735476		0	0	1	1	0	1	0.217160294
1	1	0	1	1	0	0.40464032		0	0	1	1	0	0	0.214027254
0	1	1	0	1	1	0.400455118		0	0	0	0	1	1	0.203175947
1	1	0	1	0	0	0.394984145		0	0	0	0	1	0	0.194776879
1	0	0	0	1	1	0.392045994		0	0	0	1	1	0	0.189108984
1	1	0	1	0	1	0.389033798		0	0	0	1	0	1	0.180571556
0	1	0	0	1	0	0.382514956		0	0	0	1	0	0	0.177881019
1	0	1	1	1	0	0.381506169		0	0	0	1	1	1	0.16902834
1	0	0	0	0	1	0.380802321		0	0	0	0	0	1	0.148201565
0	1	1	0	0	1	0.378051788		0	0	0	0	0	0	0.058084772

Figure 10

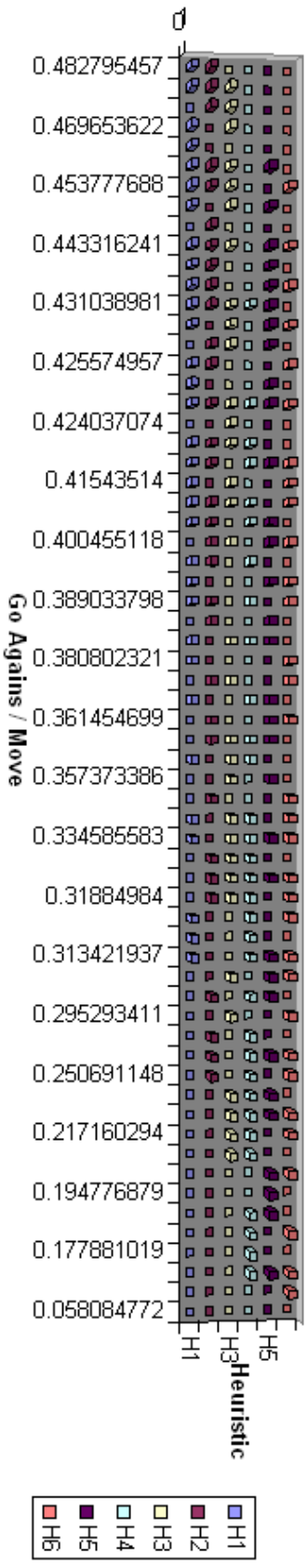


Figure 11: Go Agains Per Move for Each AI

H1	H2	H3	H4	H5	H6	CapAgain/Moves		H1	H2	H3	H4	H5	H6	CapAgain/Moves
1	1	0	0	0	0	0.676318453		1	0	1	1	0	0	0.55521687
0	1	0	0	0	0	0.662784316		0	1	0	0	0	1	0.554938407
0	1	1	0	0	0	0.659919463		1	0	0	1	1	0	0.550312707
1	1	1	0	0	0	0.657316878		0	0	1	0	0	0	0.54763258
1	0	0	0	0	0	0.653691421		0	1	1	1	1	0	0.545735413
1	0	1	0	0	0	0.645499132		1	0	1	1	0	1	0.523611111
1	1	1	0	1	0	0.641692968		1	0	1	1	1	1	0.51894811
1	1	0	0	1	0	0.641362145		1	0	0	1	0	0	0.518033428
1	1	1	0	0	1	0.634115505		0	1	1	1	0	0	0.51736881
1	1	1	1	1	0	0.628502844		0	1	0	1	1	0	0.516754362
1	1	1	1	0	0	0.625630457		0	1	1	1	1	1	0.511712322
1	1	0	0	1	1	0.625058879		1	0	0	1	0	1	0.509669462
1	1	0	1	1	0	0.624777975		1	0	0	1	1	1	0.508349155
1	1	0	1	1	1	0.62366624		0	1	1	1	0	1	0.508331285
1	1	0	0	0	1	0.623083475		0	1	0	1	0	0	0.494928102
1	1	1	0	1	1	0.622328273		0	1	0	1	1	1	0.490273621
1	1	0	1	0	0	0.62156241		0	0	1	0	1	0	0.482497517
1	1	1	1	1	1	0.616268288		0	1	0	1	0	1	0.470606645
1	0	1	0	1	0	0.614026531		0	0	1	0	1	1	0.428841845
1	1	0	1	0	1	0.612353627		0	0	1	0	0	1	0.411976767
1	1	1	1	0	1	0.611566291		0	0	1	1	1	0	0.3516507
1	0	0	0	1	0	0.601188964		0	0	1	1	0	0	0.337978289
0	1	1	0	1	0	0.600711563		0	0	1	1	0	1	0.333117932
1	0	1	0	1	1	0.594860096		0	0	1	1	1	1	0.321070925
0	1	0	0	1	0	0.591897175		0	0	0	0	1	1	0.295024349
1	0	1	0	0	1	0.584517945		0	0	0	0	1	0	0.293473521
0	1	1	0	1	1	0.577902585		0	0	0	1	0	0	0.264144258
1	0	0	0	1	1	0.569989547		0	0	0	1	1	0	0.257622969
1	0	0	0	0	1	0.565876351		0	0	0	1	0	1	0.249730573
0	1	1	0	0	1	0.560838471		0	0	0	1	1	1	0.239047137
0	1	0	0	1	1	0.560101491		0	0	0	0	0	1	0.238057777
1	0	1	1	1	0	0.556989552		0	0	0	0	0	0	0.185671471

Figure 12

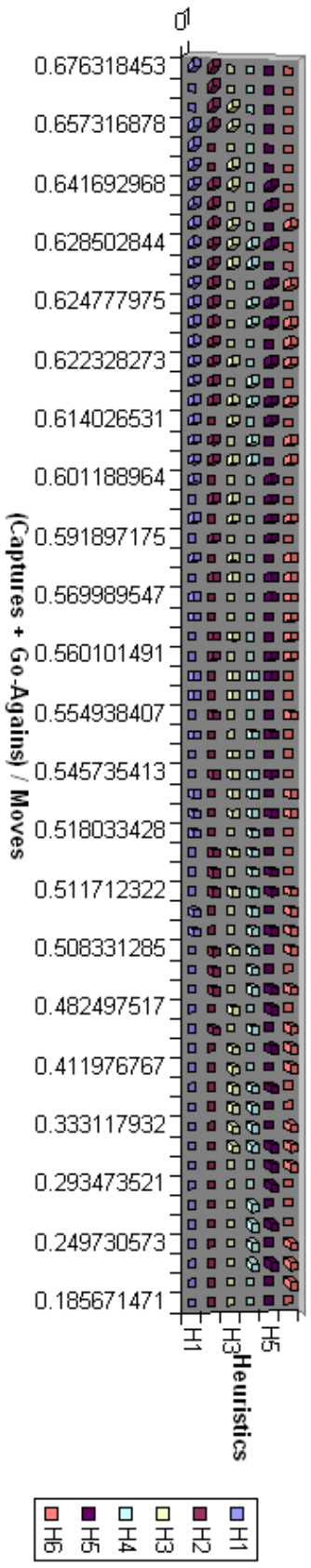


Figure 13: Avg. Captures + Avg. Go-Agains Per Move for Each AI

C. Round-Robin Results for AI Computer vs. AI Computer

The results below reflect performance and statistics relating to heuristic combinations being played against other heuristic combinations. This is the round-robin portion of the results where the results below describe the tournament and games of it. It is not important, in this section, to perform analysis of average statistic per move, since the trends of the heuristics on an average per move basis were already discovered in the analysis of AI's versus an opponent making random moves (see the previous section). Figure numbers start back at 1 for this section.

The data/analysis in this section pertains to round-robin results for all possible AI's given all possible combinations of heuristics used in the evaluation function. With six heuristics being used, this results in 63 possible AI's, given an AI must use at least one heuristic. Each AI played two matches versus the other 62 AI's—one match as the player given the first turn and one match as the player given the second turn. Ultimately each AI played 124 games.

One important distinction between the results of this data and that of the analysis of AI's versus an opponent making random moves, is that these are results of games where the moves of one AI versus another AI will never change if all initial variables of the game remain the same. This is why it is only necessary to play two matches against each opponent since the matches only vary in the aspect of which AI is given the first move. This also explains why an AI who performed well against an AI who made random moves may not perform well against an AI that performs moves based on heuristic evaluation.

Since each AI only has the possibility of playing 124 distinct matches, this only provides a small subset of the total possible distinct matches that an AI might experience when playing an opponent which makes moves based on decisions other than the given heuristics analyzed in this research—such as making random moves. Even the analysis of an AI versus a random player only looked at 800 distinct matches—four tests of each AI playing 200 total matches (100 matches given the first move and 100 matches given the second move). These are minuscule observations given the size of the search space and the given number of possible paths/moves each AI could choose.

There is reason to believe that a predictable AI, given heuristics, might never be able to visit every possible node in the search space, even versus a random opponent, since it could be shown that the AI can limit the actions/options of the opposing player merely because its own moves may take away/add to the opponent's choices. Using heuristics would limit the available options of a player, making him or her follow a given direction through the search space. The question becomes: will the player's moves, due to the heuristics, allow the player to reach a state in the search space that will result in the player's win? The data should help in making this determination.

To help simplify the analysis, the use of the various heuristics ultimately creates 63 uniquely defined AI's, as was stated above. The heuristics each AI uses can be divided into two parts—heuristics H4, H5, and H6, which deal with quantities of stones in various parts of the board (attempting to utilize a strategy of stone location), and heuristics H1, H2, and H3, which deal with blanket strategies of keeping ahead of the opponent or keeping the opponent from winning.

This means that there are seven AI's who use distinct combinations of H1, H2, and H3. Utilizing the other combinations of H4, H5, and H6, gives the complete field of AI's.

The raw data from the results gathered for the AI round-robin matches will now be presented. The table in *Figure 1* is composed of data which has definite similarities. It is very apparent that an AI which uses H2 and H3 along with combinations of H4, H5, or H6, is the same as an AI which only uses H1 with combinations of H4, H5, or H6. This makes sense because H1 is essentially a conjunction of H2 and H3. H2 maximizes how close a player is to winning (essentially finding the move which results in obtaining the most stones), H3 minimizes this for the opposing player, and combining the two strategies results in H1: find the move with the greatest margin over the opponent. H1 finds the difference in scores and makes a move based on the test outcome; whereas, H2 finds the maximum score the player could obtain, H3 finds the minimum score the opponent could obtain, and combining the two results in the maximum difference between these two values.

This simplifies the amount of possible distinct AI's which must be analyzed for effectiveness. The total combination of AI's using heuristics H1, H2, or H3 is only seven for a total of 56 AI's. This now becomes six combinations for a total of 48 AI's—a reduction of eight. *Figure 2* is color coded to indicate identical AI's.

It is inefficient to analyze the success of AI's which do not use H1, H2, or H3. This excludes the seven AI's which just use combinations of H4, H5, and H6. *Figure 3* shows these AI's removed. It is also not successful to just use H3 without any combination of H1 or H2 since H3 only deals with making moves which keep the opponent's score low—not an effective way to win the game when it requires the player to have the most stones. This excludes another eight AI's which are removed from *Figure 4*.

The raw data with respect to a player's win percentage indicates that H2, by itself without being combined in some what with H1 or H3, is not very effective. This follows logically since it will not be a successful strategy to concentrate on the move which will give the player the largest increase in score alone, but a move which takes into account how the opponent is doing also. This makes another eight possibilities which do not have to be considered. *Figure 5* removes these.

Analysis of the remaining AI's which just used H6 with a combination of H1, H2, or H3, shows that the only combinations where using H6 by itself performed better than just using a combination of H1, H2, and H3, was the AI that used H1 and H2, and the AI that used H1, H2, and H3. There are three of these cases. Since two of theses AI's were already covered above by the recognition that H1 is the same as H2 used with H3, this is only a reduction of two more AI's (*Figure 6*).

H1	H2	H3	H4	H5	H6	P1Win%	P1Win%FirstMove	P1Win%SecondMove	P1Avg_Stones	P1Avg_Margin	P1Win_Margin	P1Loss_Margin	P1Avg_Captures	P1Avg_GoAgains	P1Avg_Moves
1	1	0	0	0	1	0.7581	1	0.516129032	27.87903226	7.758064516	10.5967742	2.838709677	3.129032258	11.95967742	28.0403226
1	1	1	0	0	1	0.7581	1	0.516129032	28.54032258	9.080645161	11.7741935	2.693548387	2.596774194	12.89516129	29.4274194
0	1	1	0	0	0	0.7419	1	0.483870968	29.2016129	10.40322581	12.2258065	1.822580645	2.983870968	12.48387097	30.016129
1	0	0	0	0	0	0.7419	1	0.483870968	29.2016129	10.40322581	12.2258065	1.822580645	2.983870968	12.48387097	30.016129
1	1	1	0	0	0	0.7419	1	0.483870968	29.2016129	10.40322581	12.2258065	1.822580645	2.983870968	12.48387097	30.016129
0	1	1	0	1	0	0.7258	1	0.451612903	27.92741935	7.85483871	10.8225806	2.967741935	2.725806452	12.99193548	29.9032258
1	0	0	0	1	0	0.7258	1	0.451612903	27.92741935	7.85483871	10.8225806	2.967741935	2.725806452	12.99193548	29.9032258
0	1	1	0	1	1	0.6532	0.887096774	0.419354839	26.89516129	5.790322581	9.51612903	3.725806452	2.766129032	11.28225806	30.1370968
1	0	0	0	1	1	0.6532	0.887096774	0.419354839	26.89516129	5.790322581	9.51612903	3.725806452	2.766129032	11.28225806	30.1370968
0	1	1	1	1	0	0.5403	0.677419355	0.403225806	24.93548387	1.870967742	6.85483871	4.983870968	2.951612903	7.701612903	27.8387097
1	0	0	1	1	0	0.5403	0.677419355	0.403225806	24.93548387	1.870967742	6.85483871	4.983870968	2.951612903	7.701612903	27.8387097
0	1	1	1	0	1	0.5323	0.693548387	0.370967742	24.62096774	1.241935484	6.32258065	5.080645161	2.685483871	7.60483871	28.4677419
1	0	0	1	0	1	0.5323	0.693548387	0.370967742	24.62096774	1.241935484	6.32258065	5.080645161	2.685483871	7.60483871	28.4677419
0	1	1	1	1	1	0.4516	0.564516129	0.338709677	24.56451613	1.129032258	6.16129032	5.032258065	3.201612903	6.806451613	29.5322581
1	0	0	1	1	1	0.4516	0.564516129	0.338709677	24.56451613	1.129032258	6.16129032	5.032258065	3.201612903	6.806451613	29.5322581
0	1	1	0	0	1	0.3952	0.435483871	0.35483871	24.18548387	0.370967742	6.01612903	5.64516129	2.572580645	9.120967742	30.1451613
1	0	0	0	0	1	0.3952	0.435483871	0.35483871	24.18548387	0.370967742	6.01612903	5.64516129	2.572580645	9.120967742	30.1451613
0	1	1	1	0	0	0.3629	0.419354839	0.306451613	23.58064516	-0.83870968	5.0483871	5.887096774	3.137096774	6.298387097	27.5
1	0	0	1	0	0	0.3629	0.419354839	0.306451613	23.58064516	-0.83870968	5.0483871	5.887096774	3.137096774	6.298387097	27.5

Figure 1

H1	H2	H3	H4	H5	H6	P1Win%
0	0	0	0	0	1	0.0323
0	0	0	0	1	0	0.0403
0	0	0	0	1	1	0.0806
0	0	0	1	0	0	0.0565
0	0	0	1	0	1	0.0403
0	0	0	1	1	0	0.0484
0	0	0	1	1	1	0.0645
0	0	1	0	0	0	0.5
0	0	1	0	0	1	0.2823
0	0	1	0	1	0	0.3387
0	0	1	0	1	1	0.25
0	0	1	1	0	0	0.1935
0	0	1	1	0	1	0.1694
0	0	1	1	1	0	0.1935
0	0	1	1	1	1	0.2258
0	1	0	0	0	0	0.5242
0	1	0	0	0	1	0.3145
0	1	0	0	1	0	0.3226
0	1	0	0	1	1	0.3548
0	1	0	1	0	0	0.2016
0	1	0	1	0	1	0.2097
0	1	0	1	1	0	0.3387
0	1	0	1	1	1	0.2903
0	1	1	0	0	0	0.7419
0	1	1	0	0	1	0.3952
0	1	1	0	1	0	0.7258
0	1	1	0	1	1	0.6532
0	1	1	1	0	0	0.3629
0	1	1	1	0	1	0.5323
0	1	1	1	1	0	0.5403
0	1	1	1	1	1	0.4516

H1	H2	H3	H4	H5	H6	P1Win%
1	0	0	0	0	0	0.7419
1	0	0	0	0	1	0.3952
1	0	0	0	1	0	0.7258
1	0	0	0	1	1	0.6532
1	0	0	1	0	0	0.3629
1	0	0	1	0	1	0.5323
1	0	0	1	1	0	0.5403
1	0	0	1	1	1	0.4516
1	0	1	0	0	0	0.75
1	0	1	0	0	1	0.4597
1	0	1	0	1	0	0.7258
1	0	1	0	1	1	0.7016
1	0	1	1	0	0	0.4839
1	0	1	1	0	1	0.6694
1	0	1	1	1	0	0.5887
1	0	1	1	1	1	0.5645
1	1	0	0	0	0	0.75
1	1	0	0	0	1	0.7581
1	1	0	0	1	0	0.7097
1	1	0	0	1	1	0.6935
1	1	0	1	0	0	0.5806
1	1	0	1	0	1	0.7097
1	1	0	1	1	0	0.7661
1	1	0	1	1	1	0.6774
1	1	1	0	0	0	0.7419
1	1	1	0	0	1	0.7581
1	1	1	0	1	0	0.7016
1	1	1	0	1	1	0.7419
1	1	1	1	0	0	0.7177
1	1	1	1	0	1	0.6452
1	1	1	1	1	0	0.7823
1	1	1	1	1	1	0.7419

Figure 2

H1	H2	H3	H4	H5	H6	P1Win%
0	0	1	0	0	0	0.5
0	0	1	0	0	1	0.2823
0	0	1	0	1	0	0.3387
0	0	1	0	1	1	0.25
0	0	1	1	0	0	0.1935
0	0	1	1	0	1	0.1694
0	0	1	1	1	0	0.1935
0	0	1	1	1	1	0.2258
0	1	0	0	0	0	0.5242
0	1	0	0	0	1	0.3145
0	1	0	0	1	0	0.3226
0	1	0	0	1	1	0.3548
0	1	0	1	0	0	0.2016
0	1	0	1	0	1	0.2097
0	1	0	1	1	0	0.3387
0	1	0	1	1	1	0.2903
0	1	1	0	0	0	0.7419
0	1	1	0	0	1	0.3952
0	1	1	0	1	0	0.7258
0	1	1	0	1	1	0.6532
0	1	1	1	0	0	0.3629
0	1	1	1	0	1	0.5323
0	1	1	1	1	0	0.5403
0	1	1	1	1	1	0.4516

H1	H2	H3	H4	H5	H6	P1Win%
1	0	0	0	0	0	0.7419
1	0	0	0	0	1	0.3952
1	0	0	0	1	0	0.7258
1	0	0	0	1	1	0.6532
1	0	0	1	0	0	0.3629
1	0	0	1	0	1	0.5323
1	0	0	1	1	0	0.5403
1	0	0	1	1	1	0.4516
1	0	1	0	0	0	0.75
1	0	1	0	0	1	0.4597
1	0	1	0	1	0	0.7258
1	0	1	0	1	1	0.7016
1	0	1	1	0	0	0.4839
1	0	1	1	0	1	0.6694
1	0	1	1	1	0	0.5887
1	0	1	1	1	1	0.5645
1	1	0	0	0	0	0.75
1	1	0	0	0	1	0.7581
1	1	0	0	1	0	0.7097
1	1	0	0	1	1	0.6935
1	1	0	1	0	0	0.5806
1	1	0	1	0	1	0.7097
1	1	0	1	1	0	0.7661
1	1	0	1	1	1	0.6774
1	1	1	0	0	0	0.7419
1	1	1	0	0	1	0.7581
1	1	1	0	1	0	0.7016
1	1	1	0	1	1	0.7419
1	1	1	1	0	0	0.7177
1	1	1	1	0	1	0.6452
1	1	1	1	1	0	0.7823
1	1	1	1	1	1	0.7419

Figure 3

H1	H2	H3	H4	H5	H6	P1Win%
0	1	0	0	0	0	0.524194
0	1	0	0	0	1	0.314516
0	1	0	0	1	0	0.322581
0	1	0	0	1	1	0.354839
0	1	0	1	0	0	0.201613
0	1	0	1	0	1	0.209677
0	1	0	1	1	0	0.33871
0	1	0	1	1	1	0.290323
0	1	1	0	0	0	0.741935
0	1	1	0	0	1	0.395161
0	1	1	0	1	0	0.725806
0	1	1	0	1	1	0.653226
0	1	1	1	0	0	0.362903
0	1	1	1	0	1	0.532258
0	1	1	1	1	0	0.540323
0	1	1	1	1	1	0.451613
1	0	0	0	0	0	0.741935
1	0	0	0	0	1	0.395161
1	0	0	0	1	0	0.725806
1	0	0	0	1	1	0.653226
1	0	0	1	0	0	0.362903
1	0	0	1	0	1	0.532258
1	0	0	1	1	0	0.540323
1	0	0	1	1	1	0.451613

H1	H2	H3	H4	H5	H6	P1Win%
1	0	1	0	0	0	0.75
1	0	1	0	0	1	0.459677
1	0	1	0	1	0	0.725806
1	0	1	0	1	1	0.701613
1	0	1	1	0	0	0.483871
1	0	1	1	0	1	0.669355
1	0	1	1	1	0	0.58871
1	0	1	1	1	1	0.564516
1	1	0	0	0	0	0.75
1	1	0	0	0	1	0.758065
1	1	0	0	1	0	0.709677
1	1	0	0	1	1	0.693548
1	1	0	1	0	0	0.580645
1	1	0	1	0	1	0.709677
1	1	0	1	1	0	0.766129
1	1	0	1	1	1	0.677419
1	1	1	0	0	0	0.741935
1	1	1	0	0	1	0.758065
1	1	1	0	1	0	0.701613
1	1	1	0	1	1	0.741935
1	1	1	1	0	0	0.717742
1	1	1	1	0	1	0.645161
1	1	1	1	1	0	0.782258
1	1	1	1	1	1	0.741935

Figure 4

H1	H2	H3	H4	H5	H6	P1Win%
0	1	1	0	0	0	0.741935
0	1	1	0	0	1	0.395161
0	1	1	0	1	0	0.725806
0	1	1	0	1	1	0.653226
0	1	1	1	0	0	0.362903
0	1	1	1	0	1	0.532258
0	1	1	1	1	0	0.540323
0	1	1	1	1	1	0.451613
1	0	0	0	0	0	0.741935
1	0	0	0	0	1	0.395161
1	0	0	0	1	0	0.725806
1	0	0	0	1	1	0.653226
1	0	0	1	0	0	0.362903
1	0	0	1	0	1	0.532258
1	0	0	1	1	0	0.540323
1	0	0	1	1	1	0.451613

H1	H2	H3	H4	H5	H6	P1Win%
1	0	1	0	0	0	0.75
1	0	1	0	0	1	0.459677
1	0	1	0	1	0	0.725806
1	0	1	0	1	1	0.701613
1	0	1	1	0	0	0.483871
1	0	1	1	0	1	0.669355
1	0	1	1	1	0	0.58871
1	0	1	1	1	1	0.564516
1	1	0	0	0	0	0.75
1	1	0	0	0	1	0.758065
1	1	0	0	1	0	0.709677
1	1	0	0	1	1	0.693548
1	1	0	1	0	0	0.580645
1	1	0	1	0	1	0.709677
1	1	0	1	1	0	0.766129
1	1	0	1	1	1	0.677419
1	1	1	0	0	0	0.741935
1	1	1	0	0	1	0.758065
1	1	1	0	1	0	0.701613
1	1	1	0	1	1	0.741935
1	1	1	1	0	0	0.717742
1	1	1	1	0	1	0.645161
1	1	1	1	1	0	0.782258
1	1	1	1	1	1	0.741935

Figure 5

With every combination of AI's using H1, H2, and H3, with no other heuristics, adding H4 by itself caused the AI to perform worse. There are five of these cases. Two are covered with the AI pairs that perform the same, making a reduction of four distinct AI's. This follows logically, when it's assumed that a good AI player is one that will make many captures and go-agains during the game, which H4 by itself (choosing a move with more stones close to home) does not accommodate very well. It's also noticeable that AI's (H1) and (H2 with H3) perform worse in every instance when any combination of heuristics H4, H5, or H6 are added (10 cases), thus reducing the set by five more distinct AI's (*Figure 7*).

Figure 7 shows the two AI's who performed exactly the same, as discussed earlier. For the sake of clarity, removing the AI which uses H2 and H3 makes the resulting AI's have one large distinction—they all contain H1 as a heuristic. *Figure 7* also shows that the AI who used H1 and H3 performed best when those were the only heuristics used. This means that all other combinations of AI's which used H1 and H3 exclusively, with combinations of H4, H5, and H6, can be removed—a reduction of six distinct AI's. *Figure 8* shows the remaining AI's.

The data does not show that since the AI who uses H1 is the same as the AI that uses H2 and H3 together, their domains combine to form the same AI—a third instance of the first two. Meaning, their search domains, when exclusive, cover the exact same nodes; however, when the heuristics are combined, they appear to reduce or expand the search domain so that the AI's move choices are different. Moreover, if H4, H5, and H6 are not used, the AI which uses H1, H2, and H3 performs exactly the same as the AI using just H1 and the AI using just H2 and H3. *Figure 8* shows the color coded AI's which performed the same, reducing the number of distinct AI's by one more. It also shows that there are four distinct AI's left which used different combinations of H1, H2, and H3. *Figure 9* shows the best combinations of H4, H5, and H6 with those AI's. One need only look at the AI's which performed best for the distinct sets of H1, H2, and H3. Since the AI which used H1, H2, and H3 only performed better when some combination of H4, H5, and H6 were used, we can not only eliminate that AI, but also the AI which used just H1, since that AI is exactly the same as the AI that only used H1, H2, and H3.

This results in three AI's (H1, H3), (H1, H2, H4, H5), and (H1, H2, H3, H4, H5), the best AI out of each distinct set of AI's as discussed above. (H1, H3) only had a 75% win percentage, which is by far not the best out of the other possible AI's. It also only had a 97% win percentage when given the first move compared to other AI's that were left out that had 100% win percentages when given the first move. This is the best AI which used just H1 and H3 with other combinations of H4, H5, and H6.

To conclude, we added another combination of an AI which used H1 and H2; this was (H1, H2), and can be seen in *Figure 10*. As important as it is to performance measures to be able to win, there is also value in being able to not lose. The "not lose" percentage statistic is not the same as the win percentage statistic when ties are allowed in a match, which is possible in Mancala. The "not lose" value is calculated using the tie percent, which can be derived from the win percentages of Player 1 and Player 2. If they do not add up to 100%, then the rest of the games had to end in a tie. Overall, (H1, H2, H3, H4) was able to win more matches, but (H1, H2) was able to lose less matches. These four AI's represent the best AI's from each distinct set out of all combinations of AI's.

H1	H2	H3	H4	H5	H6	P1Win%
0	1	1	0	0	0	0.74194
0	1	1	0	1	0	0.72581
0	1	1	0	1	1	0.65323
0	1	1	1	0	0	0.3629
0	1	1	1	0	1	0.53226
0	1	1	1	1	0	0.54032
0	1	1	1	1	1	0.45161
1	0	0	0	0	0	0.74194
1	0	0	0	1	0	0.72581
1	0	0	0	1	1	0.65323
1	0	0	1	0	0	0.3629
1	0	0	1	0	1	0.53226
1	0	0	1	1	0	0.54032
1	0	0	1	1	1	0.45161
1	0	1	0	0	0	0.75
1	0	1	0	1	0	0.72581
1	0	1	0	1	1	0.70161
1	0	1	1	0	0	0.48387
1	0	1	1	0	1	0.66935
1	0	1	1	1	0	0.58871
1	0	1	1	1	1	0.56452

Figure 6

H1	H2	H3	H4	H5	H6	P1Win%
1	1	0	0	0	0	0.75
1	1	0	0	0	1	0.7581
1	1	0	0	1	0	0.7097
1	1	0	0	1	1	0.6935
1	1	0	1	0	0	0.5806
1	1	0	1	0	1	0.7097
1	1	0	1	1	0	0.7661
1	1	0	1	1	1	0.6774
1	1	1	0	0	0	0.7419
1	1	1	0	0	1	0.7581
1	1	1	0	1	0	0.7016
1	1	1	0	1	1	0.7419
1	1	1	1	0	0	0.7177
1	1	1	1	0	1	0.6452
1	1	1	1	1	0	0.7823
1	1	1	1	1	1	0.7419

H1	H2	H3	H4	H5	H6	P1Win%
0	1	1	0	0	0	0.7419
1	0	0	0	0	0	0.7419
1	0	1	0	0	0	0.75
1	0	1	0	1	0	0.7258
1	0	1	0	1	1	0.7016
1	0	1	1	0	1	0.6694
1	0	1	1	1	0	0.5887
1	0	1	1	1	1	0.5645
1	1	0	0	0	0	0.75
1	1	0	0	0	1	0.7581
1	1	0	0	1	0	0.7097
1	1	0	0	1	1	0.6935
1	1	0	1	0	1	0.7097
1	1	0	1	1	0	0.7661
1	1	0	1	1	1	0.6774
1	1	1	0	0	0	0.7419
1	1	1	0	0	1	0.7581
1	1	1	0	1	0	0.7016
1	1	1	0	1	1	0.7419
1	1	1	1	0	1	0.6452
1	1	1	1	1	0	0.7823
1	1	1	1	1	1	0.7419

Figure 7

H1	H2	H3	H4	H5	H6	P1Win%
1	0	0	0	0	0	0.7419
1	0	1	0	0	0	0.75
1	1	0	0	0	0	0.75
1	1	0	0	0	1	0.7581
1	1	0	0	1	0	0.7097
1	1	0	0	1	1	0.6935
1	1	0	1	0	1	0.7097
1	1	0	1	1	0	0.7661
1	1	0	1	1	1	0.6774
1	1	1	0	0	0	0.7419
1	1	1	0	0	1	0.7581
1	1	1	0	1	0	0.7097
1	1	1	0	1	1	0.7419
1	1	1	1	0	0	0.7177
1	1	1	1	0	1	0.6452
1	1	1	1	1	0	0.7823
1	1	1	1	1	1	0.7419

Figure 8

H1	H2	H3	H4	H5	H6	P1Win%
1	0	1	0	0	0	0.75
1	1	0	0	0	0	0.7661
1	1	1	0	0	1	0.6452

Figure 9

Figure 11 shows all of the AI's sorted by win percentage. After logically digging through the data and observing consistencies among various AI's, the results of *Figure 10* should be the basis upon which further AI development is made.

H1	H2	H3	H4	H5	H6	P1Win%	Win%FirstMove	Win%SecondMove	AvgTie%	AvgNot_Lose%
1	0	1	0	0	0	0.75	0.967741935	0.532258065	0.03226	0.782258065
1	1	0	1	1	0	0.766129	1	0.532258065	0.01613	0.782258065
1	1	1	1	1	0	0.782258	1	0.564516129	0.02419	0.806451613
1	1	0	0	0	0	0.75	1	0.5	0.05645	0.806451613

Figure 10

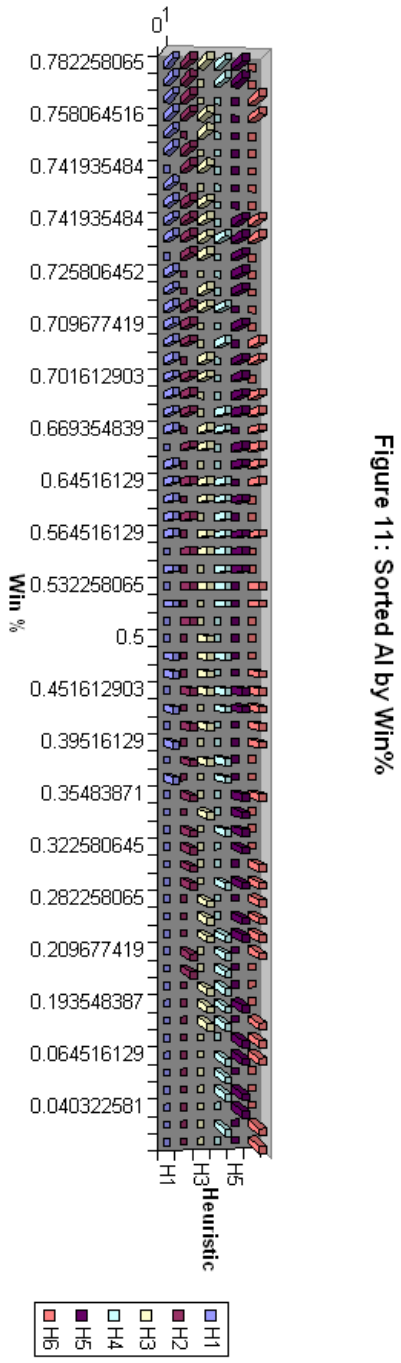


Figure 11: Sorted AI by Win%

D. Single Heuristic Effectiveness

The results below reflect performances and statistics of single heuristics (one at a time) playing against the Random player, and then playing against other heuristics. The associated charts and graphs are shown along with the statistics.

H1, H2, and H3 perform significantly better than H4, H5, and H6:

H1	H2	H3	H4	H5	H6	Win%	Win%FirstMove	Win%SecondMove
0	0	0	0	0	1	52.13%	50.00%	54.25%
0	0	0	0	1	0	51.63%	55.50%	47.75%
0	0	0	1	0	0	65.63%	64.50%	66.75%
0	0	0	1	1	1	51.75%	56.25%	47.25%
1	1	1	0	0	0	100.00%	100.00%	100.00%
0	0	1	0	0	0	100.00%	100.00%	100.00%
0	1	0	0	0	0	98.88%	99.00%	98.75%
1	0	0	0	0	0	100.00%	100.00%	100.00%

Instances where H1 win% isn't 100% are the following:

H1	H2	H3	H4	H5	H6	Win%	Win%FirstMove	Win%SecondMove
1	0	0	0	0	1	99.88%	100.00%	99.75%
1	0	1	1	0	0	99.88%	100.00%	99.75%
1	1	1	0	0	1	99.88%	100.00%	99.75%
1	1	0	0	1	1	99.88%	100.00%	99.75%

Regardless of combination, heuristic sets with H1 in them win 100% when going first!

Best Single Heuristic: Heuristic 1

By utilizing such a deep look-ahead (nine steps), this heuristic can very accurately calculate the move that will maximize the number of stones captured. It also displayed an average of 38 stones in a game with a win margin of 29. The first three heuristics we used are based on the single most important aspect of winning: having more stones than your opponent by the end of the game. Since we have seen that maximizing the number of stones captured is the most effective strategy against a random player, we can conclude that doing so will always result in a win. This is mainly because random has at best a 1/6 chance of choosing the move that will maximize its own number of stones.

Worst Single Heuristic: Heuristic 5

Heuristic 5 attempts to search out the course of action that will result in it ending with the most stones in the first two bins. H5 would seem effective for capturing and going again, since it would open up the middle/end of the board for captures, and large stacks of stones far from home/within striking distance of going again; but would seem like a bad heuristic since keeping your stones far from home allows the opportunity for the opponent to make large captures. However, there seems to be no inherent advantage to this. The greatest disadvantage of this particular heuristic is that it overtly sets itself up for devastating captures, as the highest volume of stones are passing through that area of the opponents board. With no way to foresee or even understand stone loss, this heuristic just can't stand up to the rest.

E. Heuristic Combination Effectiveness

The results below reflect performances and statistics of heuristics combinations (using multiple at a time combining into overall utility) playing against other combinations. The associated charts and graphs are shown along with the statistics.

Best 5 heuristic combinations:

H1	H2	H3	H4	H5	H6	Win%	Win%FirstMove	Win%SecondMove	Avg_Stones
1	0	1	0	0	0	100.00%	100.00%	100.00%	38.98375
1	1	1	0	0	0	100.00%	100.00%	100.00%	38.69125
1	1	1	1	0	1	100.00%	100.00%	100.00%	38.68375
1	1	1	1	0	0	100.00%	100.00%	100.00%	38.66875
1	0	0	0	0	0	100.00%	100.00%	100.00%	38.6675

Noticeably lacking from any of the top five heuristics is H5, which is expected as it is by far the worst of all the individual heuristics. Also notable is the inclusion of H1 in all of the top five, and H3 in the top four. H3 was the second best of the single heuristics, and chooses the move that keeps your opponent the furthest from securing more than half the available stones. Similar in nature is H2, which also makes a sizable and similarly understandable showing in the top five. H4 tended to increase the number of captures when combined with one of the first three heuristics, which is why we would say it appears in some of the best configurations. We would guess that H6 produced some kind of balance in the system which increased overall stone count.

Worst 5 heuristic combinations:

H1	H2	H3	H4	H5	H6	Win%	Win%FirstMove	Win%SecondMove	Avg_Stones
0	0	0	0	1	1	51.63%	55.50%	47.75%	24.84875
0	0	0	1	1	1	51.75%	56.25%	47.25%	24.2925
0	0	0	0	0	1	52.13%	50.00%	54.25%	24.56
0	0	0	1	0	1	52.88%	52.25%	53.50%	24.485
0	0	0	1	1	0	56.50%	59.25%	53.75%	25.06

Noticeably lacking from the worst five combinations is any mention of the first three heuristics. The lowest win percentage of any combination containing the first three heuristics is 97%. There is a dead even distribution of each of the heuristics throughout these combinations. However, we notice that H5 is not only the worst single heuristic, but in the overall worst combination as well. Somewhat surprising is the fact that the combination of all three heuristics here is the second worst combination on average. What this says is that there is no real advantage to merely trying to keep stones on your side of the board. H4, which is the best of these heuristics, shows an increase in first move win percentage when used, even with H5. This means that it has some potential, or at least more than any of the others, which seem more decisively detrimental to success.

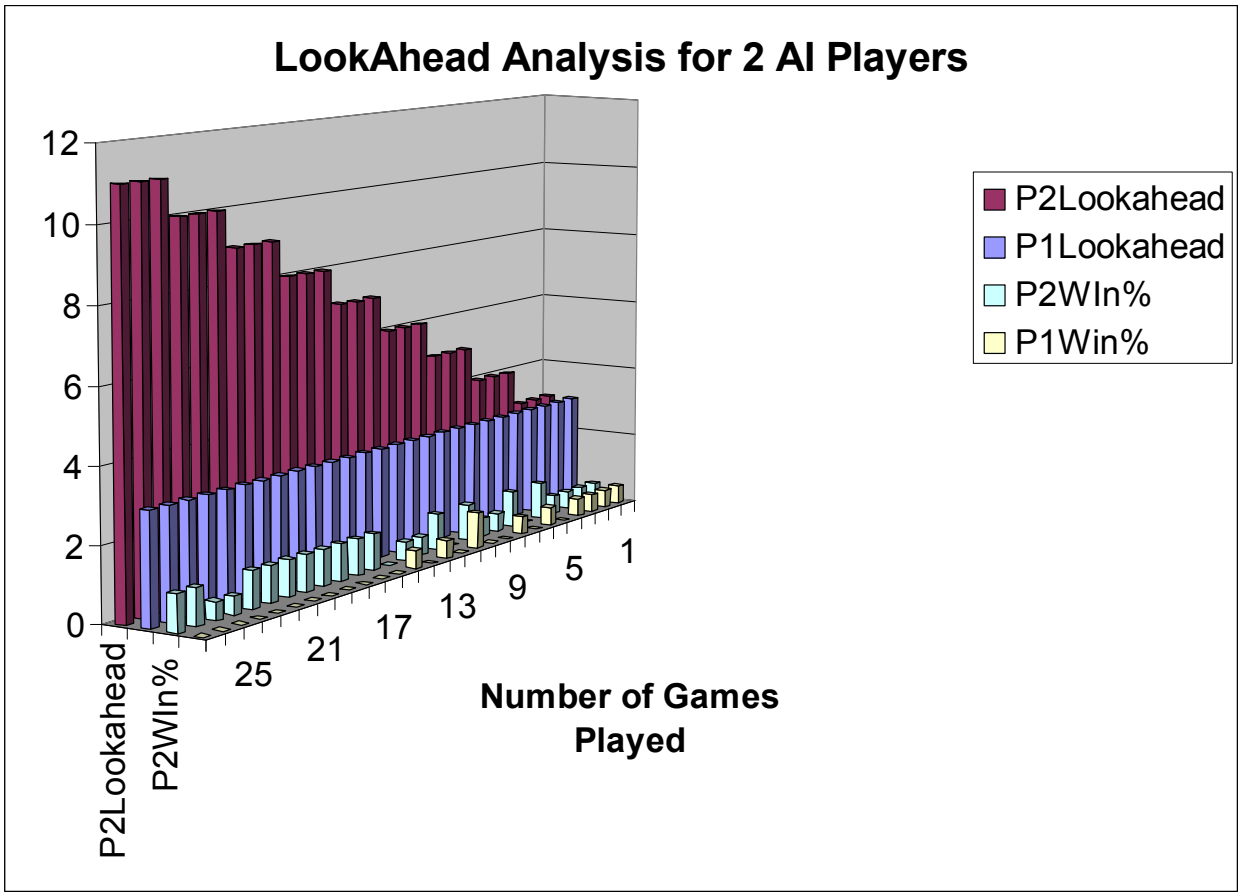
F. Look-ahead Analysis

The results below reflect performances and statistics of keeping the look-ahead static, varying it, and how the look-ahead effected computation time.

H1	H2	H3	H4	H5	H6	H1	H2	H3	H4	H5	H6	P1Lookahead	P2Lookahead	P1Win%	P2Win%
1	0	0	0	0	0	1	0	0	0	0	0	3	3	0.5	0.5
0	1	0	0	0	0	0	1	0	0	0	0	3	3	0.5	0.5
0	0	1	0	0	0	0	0	1	0	0	0	3	3	0.5	0.5
1	0	0	0	0	0	1	0	0	0	0	0	3	4	0.5	0.5
0	1	0	0	0	0	0	1	0	0	0	0	3	4	0	1
0	0	1	0	0	0	0	0	1	0	0	0	3	4	0.5	0
1	0	0	0	0	0	1	0	0	0	0	0	3	5	0	1
0	1	0	0	0	0	0	1	0	0	0	0	3	5	0.5	0.5
0	0	1	0	0	0	0	0	1	0	0	0	3	5	0	0.5
1	0	0	0	0	0	1	0	0	0	0	0	3	6	0	1
0	1	0	0	0	0	0	1	0	0	0	0	3	6	1	0
0	0	1	0	0	0	0	0	1	0	0	0	3	6	0	1
1	0	0	0	0	0	1	0	0	0	0	0	3	7	0.5	0.5
0	1	0	0	0	0	0	1	0	0	0	0	3	7	0	0.5
0	0	1	0	0	0	0	0	1	0	0	0	3	7	0.5	0
1	0	0	0	0	0	1	0	0	0	0	0	3	8	0	1
0	1	0	0	0	0	0	1	0	0	0	0	3	8	0	1
0	0	1	0	0	0	0	0	1	0	0	0	3	8	0	1
1	0	0	0	0	0	1	0	0	0	0	0	3	9	0	1
0	1	0	0	0	0	0	1	0	0	0	0	3	9	0	1
0	0	1	0	0	0	0	0	1	0	0	0	3	9	0	1
1	0	0	0	0	0	1	0	0	0	0	0	3	10	0	1
0	1	0	0	0	0	0	1	0	0	0	0	3	10	0	1
0	0	1	0	0	0	0	0	1	0	0	0	3	10	0	0.5
1	0	0	0	0	0	1	0	0	0	0	0	3	11	0	0.5
0	1	0	0	0	0	0	1	0	0	0	0	3	11	0	1
0	0	1	0	0	0	0	0	1	0	0	0	3	11	0	1

The look-ahead is the key into looking into the future, in that we can force the opponent to go down the right section of the tree by choosing a move that will generate the best utility for "me" (assumes that the opponent is using the same set of heuristics). Therefore, the smaller the look-ahead, the less that player looks into the future, and the less likely it is to predict a favorable move for itself. This allows weaker heuristics to perform closer to superior ones if they are given a larger look-ahead than the superior heuristic. A larger look-ahead implies a longer search time for a move, but ultimately guarantees a better end result. Summarily, the right combination of heuristics and a large enough look-ahead creates a very formidable AI player.

From the data acquired (see chart on next page), it was observed that when the two players have the same value of look-ahead, and playing with the same heuristics, there is always a tie (both win when going first). This is because they can both predict the future with the same accuracy. However, we observe a significant improvement in performance for Player 2 as its look-ahead value increases. There are a few ties recorded, and out of 27 games played, there is only one recorded instance of full win percentage for Player 1 and 14 for Player 2. The losses and ties could be attributed to other factors, such as which player had the first move in the game. Overall, there is a clear advantage in the value of the higher look-ahead value.



V. Conclusions and Future Work

A. Research Summary

In this work we incorporated various AI techniques, including Mini-max, Alpha-Beta pruning, heuristics, and look-ahead via search. In summary, we implemented a batch version and GUI version of our intelligent Mancala program. Using these programs we analyzed how different heuristics and combinations of heuristics fare against one another in route to distinguishing those that perform extremely well and others that are inferior. To simulate games, we batched hundreds of games where the Computer players would play Random, and then Computer players played other Computer players. By incorporating different combinations and changing variables, we effectively found patterns in the statistics that point toward successful or unsuccessful heuristic combinations. Our goals of comparing Mancala heuristics, singling out the "best" ones, and observing how game values can drastically change the winner were accomplished. We feel like this work was a success, and feel confident about our findings.

The game of Mancala proved to be an interesting application of the above algorithms and concepts, and also turned out to be extremely useful and fun to work on. Throughout the development process, we learned various aspects of the algorithms we were using and truly how effective they can be (Alpha-Beta pruning in particular). Although the analysis was quite thorough, there is plenty of room for extremely detailed heuristic and strategy analysis. We feel our work could be used for further research and development for Mancala and related games.

B. Suggestions for Improvement / Advancement

The following are simple improvements that could be done to make it better and more robust. These specific topics are beyond the scope of this report, but we include them to show that the presented work could be used for further development and research with the Mancala family of games.

Optimize code for time and space efficiency

Using large look-aheads takes time to search and expand the game tree, so making the code more efficient will allow larger look-aheads to complete in less time than they do now. As with any program that is recursive and opens a state space, efficiency in time and space is key to performance. This would be a good extension to this work.

Make heuristic contribution (weight) vary between certain fuzzy range [0..1]

Allowing "part of" each heuristic to contribute to the total utility would allow us to find out the optimal set of weights for our set of heuristics. The evaluation function results could be bred in a way to mathematically combine each two sets of weights to form a new set of weights that resulted from "breeding" the previous two sets. This extension would actually be quite easy to implement given our current structure, and would be an excellent addition. In fact, our entire architecture was designed in such a way to make additions easy; thus, it along with many other things could be easily inserted.

Implement a learning or genetic approach to find best/optimal set of weights

This was discussed in the previous extension. This would most likely occur using our current batch processing approach to keep knowledge of weights and situations at each step/state. The learning approach would more than likely be reinforcement due to how we could evaluate how "good" the set of weights or the weight adjustment was by seeing how close it got us to winning (*1 + half*) or if it resulted in a substantial amount of stones down the road. This addition wouldn't be too difficult to add and would provide another level of AI. The learning, testing, and running time would take quite a while and thus could be an entire study in itself. A genetic algorithm approach could also be done using weights as the focus, while also introducing on reproduction, crossover, and mutation. This is also very difficult in that very few people have done this for this game, and we are unsure on how effective or useful it would be for a game like Mancala.

Develop more/better heuristics

The game of Mancala has many heuristics. During our research we saw and thought of a total of 25 heuristics. We narrowed the list down to 6 or 7 based on their empirically-found importance or if one heuristic inherently encompassed another (and thus that heuristic wouldn't be as useful to test). We mainly tested heuristic performance to find the best single heuristic and best combination of heuristics, as well as tried to determine if board position was important. Look-ahead analysis was also performed. There could be more heuristics and even better ones than included in this report, but more research and testing would be required.

Make board configurations expandable (number of bins on each side, etc.)

Our current game allows us to adjust the number of starting stones per bin in both the GUI and batch versions. A good extension would be to allow the game boards to have a dynamic number of bins on each side. Ours is hard-coded at 6 bins per side (excluding the home bins). An extension like this would require quite a bit more analysis, as well as making the entire program dynamic (search method(s), heuristics, etc). Although it might be time-consuming, developing

an architecture that would allow this would greatly increase the amount of research and statistical analysis that could be accomplished.

Analyze different, specific board configurations

We analyzed heuristics and tried to determine the best board position and look-ahead, but we didn't analyze specific board situations. Are certain board configurations optimal for utility? Do certain moves at certain situations guarantee a large win/capture? Does board configuration matter in the long run, or can simple heuristics do just as good as or better than making sure the stones on the board are in a specific manner? With more research, these and more questions could be answered; a "solution" to Mancala depends on it.

Use different or extend search methods

We started off with Mini-max and then focused on Alpha-Beta pruning due to the decrease in search time it provided. Other approaches such as DFS-ID, A*, or other search techniques might increase performance and/or provide an easier way to look at the states and evaluate them. We also used recursion to expand the state space, but a non-recursive solution could provide better performance.

Source Code

All source code for this research, including the Linux batch and Windows GUI versions, are available at the following website:

https://www.cresis.ku.edu/~cgifford/mancala.html

If this source code is used to extend this work, or used as a basis for other work, please acknowledge this by referencing this report.