An Intelligent Chinese Checkers Playing Program

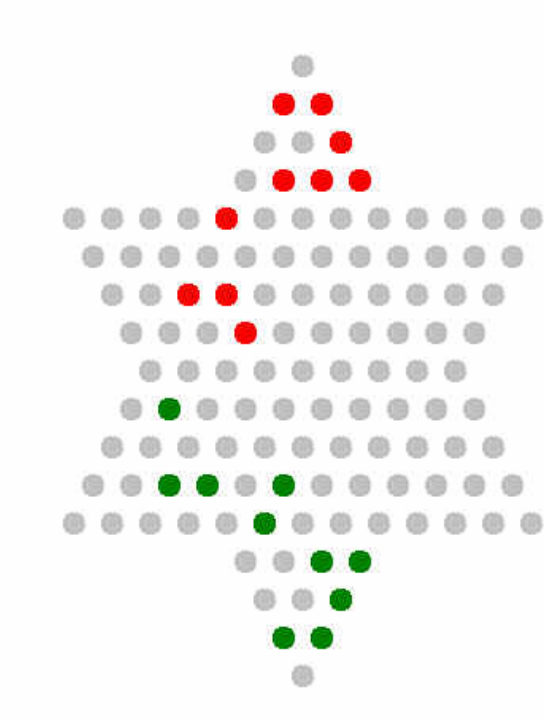
Developed By
Ashish Gupta
Ashish Gupta

URL: http://www.cse.iitd.ernet.in/~csu98130/AI/project

CONTENTS

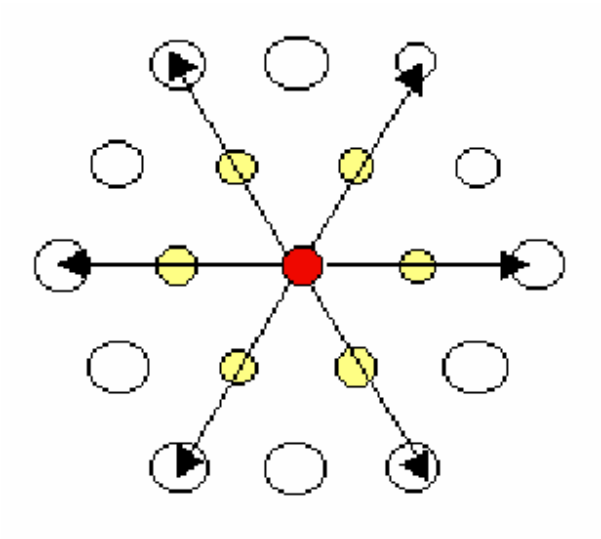
Description of the game

Chinese Checkers is played on a board shaped like a six-pointed star, with one player's home at each star point. Unlike chess, which is played on a flat board, Chinese Checkers is played with round marbles on a board with holes, where the marbles rest. The holes are connected by lines forming a hexagonal pattern. The players are only allowed to move their marbles along these lines.



Rules of the Game

Two to six players can play the game, each having ten same-colored marbles. At the start of the game, the player's marbles are in the ten holes of the star point that has the same color as his marbles. The goal is to move all marbles of your color from your starting point to the star point on the opposite side of the board, here called the goal. A marble can move by rolling to a hole next to it or by jumping over one marble, of any color, to a free hole, along the lines connecting the holes in a hexagonal pattern. The player can make several jumps in a row, but only one roll. The player cannot both roll the marble and jump with it at the same turn. When the player has moved one of his marbles, the turn passes on to the next player. Unlike chess, you never remove any game pieces from the board.



Heuristics Used

Adding Intelligence to the program

1) Since it is allowed to make several jumps in a row, it is strategically important to make it possible to do so. By building structures for the marbles to jump on, it is possible to quickly move a marble to the opposite side of the board. The ability to recognize the opportunity to make a long jump is critical for playing a good game.

2) It is also important not to leave any marbles behind when the others move over the board. Those marbles will need more turns to cross the board than if they had been moved with the others, since their opportunities to make long jumps are fewer.

The two players are placed symmetrically around the board. Only the two-player game has been studied, one human against the computer as well as computer vs the computer. The computer plays with the red marbles starting at the top of the board and the human opponent plays with the opposite green marbles.

A Layered Approach

We have developed the heuristics using a layer by layer approach. We first added a simple heuristic to the Static Evaluation Function (SEF). Then added additional functionality to each heuristic to add to its intelligence.

The Heuristics used in the Program

We have added various kinds of heuristics to the program based on the above observations and added some unique features to make heuristics perform better. Following is the list of heuristics used:

1) Random
2) Vertical Displacement
3) Vertical Displacement and Horizontal displacement
4) Vertical Displacement, Horizontal displacement and Split
5) Vertical Displacement, Horizontal displacement, split and last piece move.

See the program for details.

Heuristic Details

1. Random

In this heuristic, the computer makes a move randomly without taking into consideration the current board configuration. It can be pretty hard to beat this heuristic also, as the random player may not leave its triangle at all and thus denying the chance to the opponent to occupy its winning triangle !

2. Vertical Displacement

This heuristic does the most obvious in the strategy for winning i.e. trying to maximize the vertical jumps of any piece.

It generates all moves with minimax algorithm and then sums up the vertical distance for its pieces and adds them up.
It also adds up the vertical distance of each of the opponent's pieces.
Then it uses the following SEF (static evaluation function) to make the best move :

$Total_{Self} - Total_{Opponent}$

Caveats
The player does not know anything about the horizontal displacement. Thus it can get stuck in corners and may not be able to move into its triangle easily and may even oscillate. This problem is corrected in the next heuristic

3. Vertical and Horizontal Displacement

In this heuristic we also consider the horizontal positions of the pieces in deciding the next move. The main idea is that it is best to play in the middle of the board and the opportunity is maximum to make the best move (more chances to jump and so on).

This keeps the player's pieces in the middle and also makes it very easy to move to its destination triangle .

The formula is :

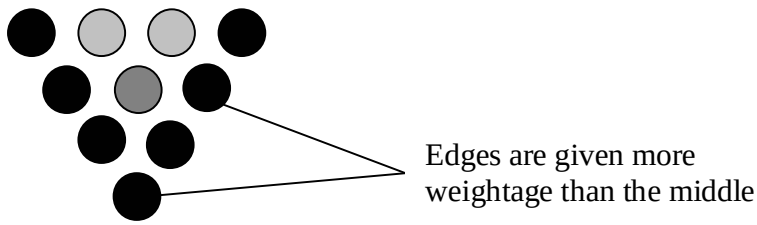
Wt. Factor * Vertical Displacement + Horizontal Displacemt

The Wt. Factor decides how much importance is to be given to keep the pieces in the middle of the board compared to jumping vertically.

4. Vertical/Horizontal Displacement with Split

This is a new idea, which makes it very easy for the heuristic to win towards the end of the play. The idea is that when the pieces start reaching its destination triangle, then we also need to provide good opportunity for other pieces to come in the destination triangle and the pieces already in the destination triangle should facilitate their entry into the destination triangle.

But the previous strategy of keeping the pieces in the middle may hinder other pieces from entering the triangle. So in this heuristic, we move the pieces to the edges of the destination triangle once they have moved in, to create space for other pieces to come in. We do this by assigning more weights to the edges then the middle of the destination triangle. This is shown below.



Destination triangle

5. The Above Strategy + Back Piece moving strategy

One problem with which the previous heuristics suffer from is that some pieces are left at the backend of the board and they have to move one by one at the end thus requiring more number of moves to win. In this heuristic, we give more weightage to the moves where the back pieces move forward than the front pieces. This produces the desired effect, i.e. no lone pieces are left behind on the board and pieces move in clusters. This further reduces the number of moves required to win. This heuristic is very strong and it is very difficult to beat it.

Optimization

List of Optimization

1) Look at only those nodes which show some progress.

In this we generated only those moves for the minimax algorithm, which advance towards the goal state and rejected others. This results in significant improvement in speed.

2) Taking shortest depth first in case of a win.

In case two paths results in a win (same value of SEF) , then we choose the path which takes us to the goal faster.

3) Expand the node with maximum value to improve alpha - beta pruning.

4) Sort the generated nodes to increase the gain from alpha-beta pruning.

Significant improvements are observed when the above optimizations are implemented.

Results

Metrics used to study

We studied various metrics to evaluate the optimizations like:

1) Number of nodes generated
2) Total number of nodes generated
3) Average number of nodes generated
4) Time taken for each move (in milliseconds)
5) Total time till now
6) Average time per move

Comparison of Heuristics

Player 1 vs Player 2	Heuristic 2	Heuristic 3	Heuristic 4	Heuristic 5
Heuristic 2	X	3	4	5
Heuristic 3	3	X	4	5
Heuristic 4	4	4	X	5
Heuristic 5	5	5	5	X

Who Wins

We see that as the intelligence improves, the computer moves better and better. It is very difficult for a human player to beat the computer with Heuristic 4 or 5.

Optimization Results

1) Optimization of generation of moves

Player 1 is computer	**Player 2 is computer**
Heur : VH	Heur : VH
Optimization : No	Optimization : Yes
Mode : None	Mode : None
Plies : 3	Plies : 3
Nodes : 11432	Nodes : 235
Total Nodes : 2196551	Total Nodes : 586895
Average Nodes : 46735	Average Nodes : 12487
Time : 270	Time : 60
Total Time : 54230	Total Time : 15110
Average Time : 1153	Average Time : 321

From the stats we see, that Player 2 is optimized. We can see the difference in the average number of nodes generated is 4 : 1. And it is also much faster than the other player. (321 ms vs 1153 ms average move time)

2) Best Move first: Improving Alpha-Beta Pruning

Player 1 is computer	**Player 2 is computer**
Heur : VHSBack	Heur : VHSBack
Optimization : Yes	Optimization : Yes
Mode : None	Mode : Best
Plies : 3	Plies : 3
Nodes : 106	Nodes : 124
Total Nodes : 550919	Total Nodes : 217675
Average Nodes : 10802	Average Nodes : 4353
Time : 0	Time : 0
Total Time : 11160	Total Time : 8240
Average Time : 218	Average Time : 164

The player with the best move first generated much less number of nodes due to heavy alpha-beta pruning (4353 vs 10802). It is faster also.

3) Sorting the generated moves: Improving Alpha-Beta Pruning

Player 1 is computer	Player 2 is computer
Heur : VHSplit	Heur : VHSplit
Optimization : Yes	Optimization : Yes
Mode : None	Mode : Sort
Plies : 3	Plies : 3
Nodes : 72	Nodes : 60
Total Nodes : 655889	Total Nodes : 188771
Average Nodes : 11116	Average Nodes : 3254
Time : 0	Time : 0
Total Time : 18620	Total Time : 9070
Average Time : 315	Average Time : 156

The player 2 first sorts all of its moves. We see that this results in heavy alpha-beta pruning. (3254 vs. 11116 average number of nodes). It is also very fast (156 ms vs. 315 ms on the average)

4) Best move vs. Sorted Moves

Player 1 is computer	Player 2 is computer
Heur : VHSplit	Heur : VHSplit
Optimization : Yes	Optimization : Yes
Mode : Best	Mode : Sort
Plies : 3	Plies : 3
Nodes : 66	Nodes : 26
Total Nodes : 262059	Total Nodes : 193441
Average Nodes : 2239	Average Nodes : 1667
Time : 0	Time : 0
Total Time : 12500	Total Time : 9400
Average Time : 106	Average Time : 81

We see that sorting of moves results in more alpha-beta pruning than the best move first approach. The tradeoff is the Order n^2-sorting algorithm vs. the order n best move first. Still sorting performs better as it results in more alpha-beta pruning and thus decreases in number of nodes generated.

Other Assignments Done

1) Tic-Tac-Toe
2) A* Algorithm
3) K-Means Clustering Algorithm