Mancala Players: Alpha-beta Search vs Monte Carlo Tree Search

Rosalinda Garcia | Oregon State University, garciros@oregonstate.edu

Abstract: This report describes two different approaches to creating agents to play the folk game mancala. Mancala is a two player game involving moving marbles around a game board. It is fairly simple but the board and number of marbles can be varied to increase more elements. Two search algorithms were considered for the agents: Alpha-beta search and Monte Carlo Tree search. In the investigation, each agent played against a human and against the other agent. Overall, MCTS was more efficient but Alpha-beta worked better for this particular definition of mancala.

1 INTRODUCTION

This experiment considers two approaches to making agents capable of playing the game mancala.

Mancala is a two player game where the goal is to move marbles from pots on the board into your designated pot at the end of the board (shown in Figure 1). The game ends when there are no more marbles on left in the pots on the middle of the board. The player with the most marbles in their pot wins.

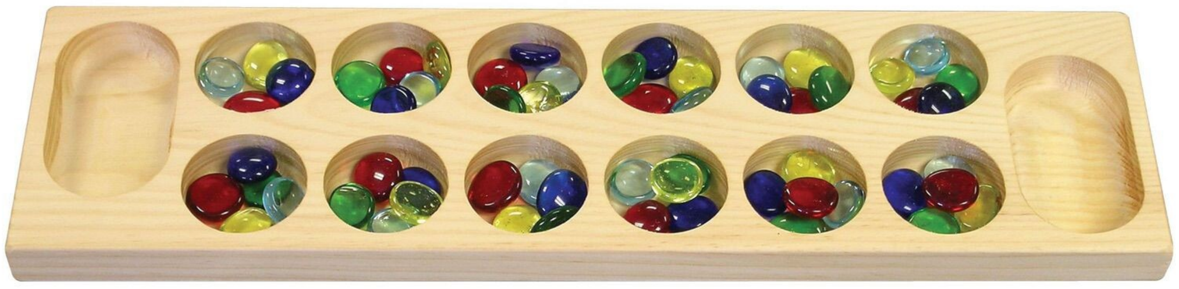


Figure 1: Mancala board. The typical set up is to have 6 pots on each long side of the board filled with 3-5 marbles.

In order to create players who can successfully engage in mancala gameplay, I consider two approaches: Alpha-beta Search and Monte Carlo Tree Search (MCTS). I compare the two agents on their ability to play against a human and against each other.

2 METHODOLOGY

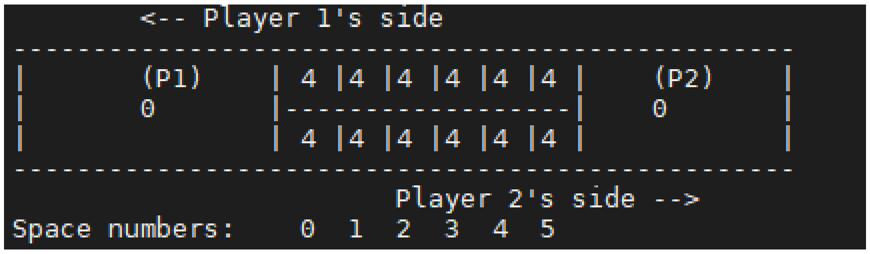
2.3 Mancala Game

In order to be able to have the agents consider the mancala game, I first had to create a virtual version of the mancala board. To do so, I had to determine the rules that it would follow. Because mancala is a traditional folk game, there are many ways to play. The following rules were used:

1. A player moves by taking marbles out of any pot on their side of the board. They they deposit the marbles they picked up sequentially in pots around the board (counterclockwise) until they run out. If they pass over their own goal pot, they deposit one marble but don't deposit in their opponents pot.
2. If a player drops their final marble of a turn into their goal pot, they get to go again

3. If a player drops their final marble of a turn into an empty pot on their side of the board, they take that marble and any marbles in the corresponding pot on the opponent's side and add them to their goal pot.
4. If a player has no marbles on their own side, their opponent gets to move until there are available moves for the player.
5. The game ends when there are no marbles left in the 12 middle pots.
6. The winner is the player with the most marbles.

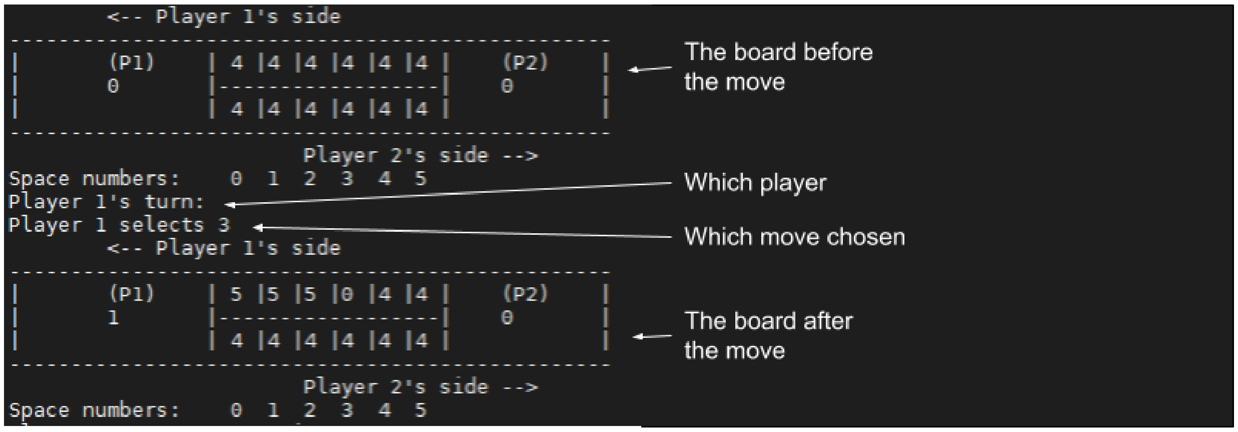
Because it was important to make sure these rules were implemented correctly, I first implemented a playable version of mancala for 2 humans. Then, I adapted the program for the two agents. The display is shown in Figure 2.



```
        <-- Player 1's side
-------------------------------------------------
|       (P1)    | 4 |4 |4 |4 |4 |4 |    (P2)    |
|       0       |------------------|    0       |
|               | 4 |4 |4 |4 |4 |4 |            |
-------------------------------------------------
                        Player 2's side -->
Space numbers:    0  1  2  3  4  5
```

Figure 2: The virtual mancala board display

Human players are asked to input the space number of the pot on their row that they want to 'pick up'. Agent players funnel their answers directly into the 'play' function, but the interface shows whose turn it was and what move was chosen in order for humans to follow the play-by-play. An example is shown in Figure 3.



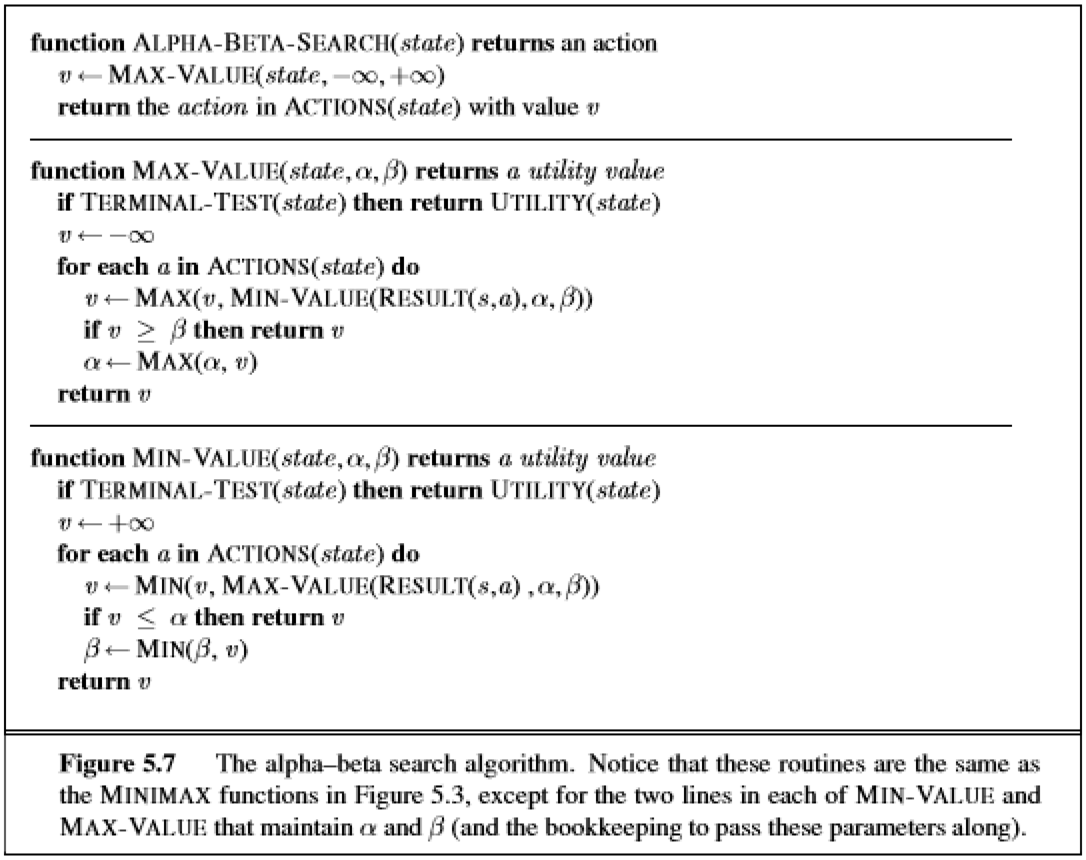
```
        <-- Player 1's side
-------------------------------------------------
|       (P1)    | 4 |4 |4 |4 |4 |4 |    (P2)    |
|       0       |------------------|    0       |
|               | 4 |4 |4 |4 |4 |4 |            |
-------------------------------------------------
                        Player 2's side -->
Space numbers:    0  1  2  3  4  5
Player 1's turn:
Player 1 selects 3
        <-- Player 1's side
-------------------------------------------------
|       (P1)    | 5 |5 |5 |0 |4 |4 |    (P2)    |
|       1       |------------------|    0       |
|               | 4 |4 |4 |4 |4 |4 |            |
-------------------------------------------------
                        Player 2's side -->
Space numbers:    0  1  2  3  4  5
```

Figure 3: The mancala board display when player 1 is a 'bot' player. Also shows an example of the board changing which each move.

2.4 Alpha-Beta Search

Alpha-beta Search is similar to Minimax search. For two player games, there is a 'min' player and a 'max' player. The 'min' player tries to minimize the score of the game while the 'max' player tries to maximize it. Alpha-beta Search differs from Minimax by pruning branches that it knows will never need to be considered. It does this by using an alpha value and a beta value to keep track of the best options for each player so far. Then it can determine which branches are unneeded. The psudocode is shown in Figure 4.

Additionally, note that I bounded this search in order to maximize time efficiency.



```
function ALPHA-BETA-SEARCH(state) returns an action
  $v \leftarrow$ MAX-VALUE(state, $-\infty$, $+\infty$)
  return the action in ACTIONS(state) with value $v$

function MAX-VALUE(state, $\alpha$, $\beta$) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  $v \leftarrow -\infty$
  for each a in ACTIONS(state) do
    $v \leftarrow$ MAX($v$, MIN-VALUE(RESULT(s,a), $\alpha$, $\beta$))
    if $v \geq \beta$ then return $v$
    $\alpha \leftarrow$ MAX($\alpha$, $v$)
  return $v$

function MIN-VALUE(state, $\alpha$, $\beta$) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  $v \leftarrow +\infty$
  for each a in ACTIONS(state) do
    $v \leftarrow$ MIN($v$, MAX-VALUE(RESULT(s,a) , $\alpha$, $\beta$))
    if $v \leq \alpha$ then return $v$
    $\beta \leftarrow$ MIN($\beta$, $v$)
  return $v$
```

Figure 5.7 The alpha–beta search algorithm. Notice that these routines are the same as the MINIMAX functions in Figure 5.3, except for the two lines in each of MIN-VALUE and MAX-VALUE that maintain α and β (and the bookkeeping to pass these parameters along).

Figure 4: Alpha-beta search algorithm psuedocode [1].

2.1 Monte Carlo Tree Search

Monte Carlo Tree Search is similar to a tree search that uses a heuristic function. It uses an Upper Confidence Bound function to select nodes instead of randomly choosing.

The search uses nodes to store information in the tree. Each node has the following:

- State: the state of the mancala board at this node
- Parent: the parent node of this node
- Action: the action to take from the parent to get to this node
- Utility: the estimated cost of the node
- Node count: the number of times the node has been selected before

Starting at the root of the tree, MCTS selects a leaf node from the available children (using the UCB function). Then it begins to expand the tree off of this node (or selects a new child node if the selected one doesn't work). As it's unfolding the tree, it is simulating the gameplay of the different moves that it selects. Then, it propagates the utility value that it finds at the bottom back up the tree so that it can determine which one is the preferred move.

Additionally, I bounded this function in order to maximize time efficiency.

2.2 Utility Function

The utility function used for both agents was the same. The first utility function was simple, just the difference in score between the two players. This was calculated simply by comparing the number of marbles each player had in their goal pots.

2.3 Testing Procedure

Each agent was tested against two human study participants. Then, the agents were tested against each other, with each agent having the opportunity to go first.

It's important to note here that in a perfect scenario, the player who goes first has the advantage. However, since the both players were bounded as time is somewhat valuable in this scenario, it's hard to make them perfect players.

Additionally, agents' decision making was tested on both on 'normal' problems and on problems that were scaled up, as described later in Section 3.2.

3 RESULTS

In this section, I present the results of the two players as they played against humans and against each other.

3.1 Alpha-beta Search

Alpha-beta did a reasonable job selecting moves. It's decision making process took approximately 19 seconds, so at times it felt a little longer than an adult playing the game but was not an unreasonable amount of time.

Against human players, it was a bad but not always a terrible player. Against one participant, it lost 23-25 after making 23 moves (while the human player made 20 moves). This game play was fairly advanced, with both players playing multiple moves in a row.

However, against a second human player, the bot did worse. It played two games against this second participant and lost 18-30 both times. The first round it made 14 moves while the human made 20 and the second round both players made 15 moves. This human player was particularly observant and able to play the game both offensively and defensively while Alpha-beta was only really playing offensively (as per it's utility function). This result suggests that a more complex heuristic/utility is needed in order to capture defensive strategies (especially considering that a player can 'steal' marbles by landing on an empty pot). These defensive moves may actually result in fewer marbles at the given moments but also fewer marbles for the opponent in the long run.

3.2 MCTS

MCTS was much more time efficient than Alpha-beta. It's longer decision making took approximately 0.5 seconds. This was much faster than the human players that played against it.

Against the first participant, MCTS did very poorly and lost 9 to 33. However, this game revealed an area that MCTS struggled with.This problem was with the repeated turns when the opponent has no moves and this scenario resulted in a game crash. I expected that this would be a problem area when determining the game rules and was not surprised by this outcome. I was unable to determine how to improve the algorithm to avoid this problem.

However, when this scenario was not happening, it was able to complete a game. Still, it did not play well. Against the second participant, MCTS lost 14-32. In this game the human participant made 22 moves while MCTS made 10 moves. Again, this participant noted that the bot seemed worse than Alpha-beta and seemed unable to act defensively.

3.2 Alpha-beta vs MCTS

With Alpha-beta going first and MCTS going second, Alpha-beta was the winner: 34 to 14. In total, Alpha-beta made 26 moves while MCTS made 17 moves. Additionally, the MCTS agent continued to struggle with the issue where the opponent has no moves which resulted in a few crashed games (not included in the table).

Table 1: Game results for Alpha-beta vs MCTS with Alpha-beta going first.

Alpha-beta	**MCTS**
34	14
31	17
38	10
34	14
36	12

In the first game with MCTS going first, MCTS won 30 to 18. In this case, Alpha-beta and MCTS made the same number of moves (16 moves). The details of other rounds with MCTS going first are shown in the Table. Despite struggling with the same cases, MCTS was able to beat Alpha-beta by going first.

Interestingly, in this game with MCTS going first the agents made fewer repeated moves (e.g. player 1 gets a repeat move) but when they did repeat, they got 3 turns in a row instead of just 2.

Table 2: Game results for MCTS vs Alpha-beta with MCTS going first

MCTS	**Alpha-beta**
30	18
29	19
31	17
35	13

29	19

3.2 Scaling Up

First, recall that with the ‘normal’ problems, the agents’s max problem solving per term were approximately as follows:

MCTS: ~0.5 seconds

Alpha-beta: ~19 seconds

Some mancala games use different numbers of marbles. When we increase the number of marbles in each pot, the agents are still able to play the game, but the time for Alpha-beta decision increased. So, as the problem scales up, Alpha-beta would need to have a significantly better heuristic and possible a more restrictive bound on it.

MCTS: ~0.5 seconds

Alpha-beta: ~21.6 seconds

Mancala boards can also have more pots. For example, when we increase the number of pots in each row to 7, the problem gets a little trickier as there are more possible moves. In this case, the time results under utility function 1 are given below:

MCTS: ~0.5 seconds

Alpha-beta: ~128.8 seconds

Here, we see that the number of pots has a much larger effect on Alpha-beta search than the increase in number of marble. This might be because the number of marbles extends the game but doesn’t add much to the number of moves available. Adding more pots directly impacts the number of choices from each state in the game.

4 DISCUSSION AND CONCLUDING REMARKS

Because MCTS struggled with the repeated turns when there were no opponent moves possible, I would ultimately recommend that MCTS be used in variations of mancala where the rules specify that the game ends when either play has no moves left. This is a common rule for mancala, just not one that I chose in defining the problem for this investigation.

With all of this in mind, I would recommend Alpha-beta search for this form of mancala. However, MCTS did a good, fast job at making decisions and playing. MCTS also seemed to scale better in terms of extended games with more marbles/pots.

If I were to extend this project with future work, I would want to experiment with different mancala rule sets as there are a variety available on the internet. This rule set was the one I was most familiar with, so it was the easiest for me to implement. However, even this rule set is a fairly complex set, so the agents might be even more successful with a simplified mancala game.

As noted by the experiences of the agents with the participants, there is also a need for developing heuristics that can allow the agents to act ‘defensively’ even when that might not maximize the number of marbles collected.

Overall, there’s room for improvement but the agents are off to a good start.

REFERENCES

[1] Stuart Russell and Peter Norvig. Artificial Intelligence: A Modern Approach, 4th US ed.