CSCI 561: Foundations of Artificial Intelligence
Instructor: Prof. Laurent Itti
Homework #2: Adversarial Search
Due on October 19th at 11:59pm, 2015

In this homework, you will write a program to determine the next move for a player in the Mancala game using Greedy, Minimax, and Alpha-Beta pruning algorithm. The rules of the Mancala game can be found at https://en.wikipedia.org/wiki/Mancala [1] and you can also try playing it online at http://play-mancala.com/ [2] to get a better understanding of the game.

Introduction to Mancala:

Mancala is a two-player game from Africa in which players moves stones around a board (shown above), trying to capture as many as possible. In the board above, player 1 owns the bottom row of stones and player 2 owns the top row. There are also two special pits on the board, called Mancalas, in which each player accumulates his or her captured stones (player 1's Mancala is on the right and player 2's Mancala is on the left).

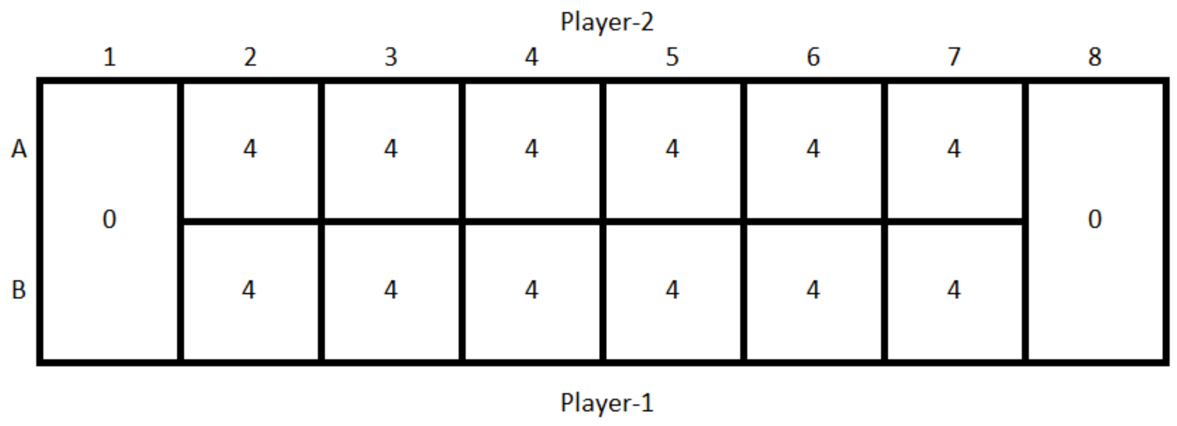
On a player's turn, he or she chooses one of the pits on his or her side of the board (not the Mancala) and removes all of the stones from that pit. The player then places one stone in each pit, moving counterclockwise around the board, starting with the pit immediately next to the chosen pit, including his or her Mancala but NOT his or her opponents Mancala, until he or she has run out of stones. If the player's last stone ends in his or her own Mancala, the player gets another turn. If the player's last stone ends in an empty pit on his or her own side, the player captures all of the stones in the pit directly across the board from where the last stone was placed (the opponents stones are removed from the pit and placed in the player's Mancala) as well as the last stone placed (the one placed in the empty pit). The game ends when one player cannot move on his or her turn, at which time the other player captures all of the stones remaining on his or her side of the board.

Tasks:

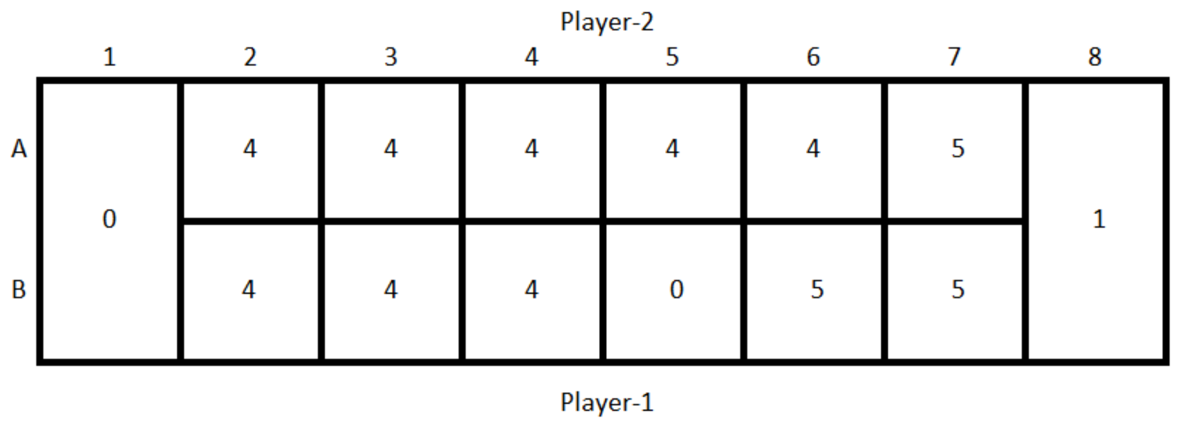
In this assignment, you will write a program to determine the next move by implementing the following algorithms:

- Greedy
- Minimax
- Alpha-Beta

Legal moves:



Assume that the current board position is as shown in the image above and the current turn is Player-1's. Blocks B2-B7 are Player-1's pits, blocks A2-A7 are Player-2's pits, B8 is Player-1's Mancala and A1 is Player-2's Mancala. We define a move for any player as to pick a pit (except Mancala) from which the stones are removed and placed in other pits as explained above. So for the current board position, Player-1's legal moves are pits B2, B3, B4, B5, B6, or B7. The image below shows the board position if Player-1 picks B5 as his/her move. Any player cannot choose an empty pit as a move, i.e. empty pits are illegal moves.



Note that a player can make more than one move during his/her turn as per the rules. For example, Player-1 can pick pit B4 and then pick any other pit (except for B4 as it will be empty) for his turn for the starting board position shown above. So keep that in mind while making decisions for any of the tasks. **YOU HAVE TO MAKE A LEGAL MOVE IF ONE IS AVAILABLE.**

End game:

If a player cannot make any valid move, i.e. all of his pits (except Mancala) are empty, the game ends and the remaining stones are moved to the other player's Mancala.

Evaluation function:

The goal of the game is to collect maximum number of stones by the end of the game and to win the game; you need to collect more stones than your opponent. So, the evaluation function for any legal move is computed as the difference between the numbers of stones in both players' Mancala if that move is chosen.

E(p) = #Stones_player - #Stones_opponent

For example, for the current board position shown above, if Player-1 chooses pit B5 as his/her move, then the value of the evaluation function would E(B5) = (1-0) = 1. Note: You can hence assume that your agent is always the "max" player. Similarly, E(B2) = (0-0) = 0.

Tie breaking and Expand order:

Ties between pits are broken by selecting the node that is first in the position order on the figure above. For example, if all legal moves for Player-1 (B2, B3, B4, B5, B5, B6, and B7) have the same evaluated values, the program must pick B2 according to tie breaker rule. Same rule applies for Player-2.

Your traverse order must be in the positional order also. For example, your program will traverse on B2, B3, B4, B5, B6, and B7 branch in order.

Board size:

The board size will be 2xN along with a mancala for each player, where N represents the number of pits for a player and $3 \leq N \leq 10$. The board size for the mancala board shown above would be 2x6. The initial number of stones in each pit can be maximum 1000.

Pseudo code:

Greedy: It is a special case of Minimax. The cut-off depth is always 1. Thus, the algorithm is very simple. You only need to pick the action which has the highest evaluation value.

Minimax: AIMA Figure 5.3 (Minimax without cut-off) and section 5.4.2 (Explanation of Cutting off search)

Alpha-Beta: AIMA Figure 5.3 (Alpha-Beta without cut-off) and section 5.4.2 (Explanation of Cutting off search)

Input:

You are provided with a file input.txt that describes the current state of the game.

<Task#> Greedy=1, MiniMax=2, Alpha-Beta=3, Competition=4

<Your player: 1 or 2>

<Cutting off depth>

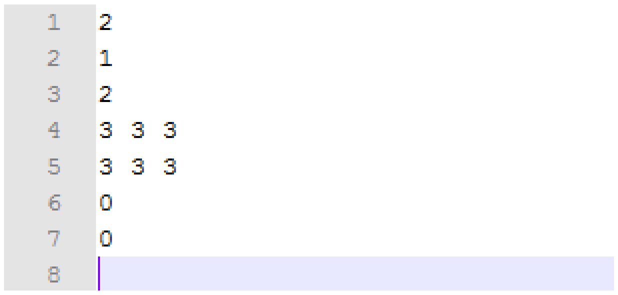
<Board state for player-2>

<Board state for player-1>

<#stones in player-2's mancala>

<#stones in player-1's mancala>

Example:



```
2
1
2
3 3 3
3 3 3
0
0

```

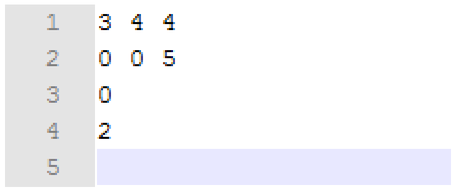
Figure 1: sample_input.txt

- The sample input file here gives you the description about a 2x3 Mancala board.
- The input file above asks you to perform task #2, i.e. **MiniMax**.
- You will be playing as player-1. Remember that player-1 is assigned the lower side of the board.
- The cut off depth for MiniMax is given as 2. You should ignore the cut off depth if the task is **Greedy**.
- Line-4 represents the board state for player-2, i.e. the upper side of the board. Each number is separated by a single white space.
- Line-5 represents the board state for player-1, i.e. the upper side of the board. Each number is separated by a single white space.
- Line-6 gives you the number of stones in player-2's mancala.
- Line-7 gives you the number of stones in player-1's mancala.
- You can assume that the input file will have no format error.
- You can assume that there will be at least one legal move available for the current player.

Output:

Greedy:

The program should output one file named ***"next_state.txt"*** showing the *next state* of the board after the greedy move in the following format.



```
3 4 4
0 0 5
0
2
```

Figure 2: next_state.txt

- Line-1 represents the board state for player-2, i.e. the upper side of the board. Each number is separated by a single white space.
- Line-2 represents the board state for player-1, i.e. the upper side of the board. Each number is separated by a single white space.
- Line-3 gives you the number of stones in player-2's mancala.
- Line-4 gives you the number of stones in player-1's mancala.

MiniMax:

The program should output two files named ***"next_state.txt"*** showing the *next state* of the board after the greedy move and ***"traverse_log.txt"*** showing the *traverse log* of your program in the following format.

- The format of "next_state.txt" should be the same as shown above.
- The format of "traverse_log.txt" should be as shown below. (The traverse log shown below is for "sample_input.txt" shown above.)

```
Node,Depth,Value
root,0,-Infinity
B2,1,Infinity
B3,1,Infinity
A2,2,1
B3,1,1
A3,2,1
B3,1,1
A4,2,1
B3,1,1
B2,1,1
```

```
B4,1,Infinity
A2,2,1
B4,1,1
A3,2,1
B4,1,1
A4,2,1
B4,1,1
B2,1,1
root,0,1
B3,1,Infinity
A2,2,0
B3,1,0
A3,2,0
B3,1,0
A4,2,0
B3,1,0
root,0,1
B4,1,Infinity
A2,2,0
B4,1,0
A3,2,0
B4,1,0
A4,2,0
B4,1,0
root,0,1
```

Note: The MiniMax traverse log requires 3 columns. Each column is separated by "," (a single comma). Three columns are node, depth and value. Everything shown here is case sensitive.

"Node": is the name of the pit you chosen as the next move. For example, player-1 chooses pit "B2" as the first move to be explored as per the evaluation order. The depth of the node "B2" is 1. Then player-1 chooses pit "B3" as his/her next move as move from pit "B2" ends up putting the last stone in the Mancala of player-1 and according to the rules of the game, player-1 needs to make another move. As this is the turn from the same player, the depth of the node "B3" is also 1. Next, player-2 chooses pit "A2" as his/her move and the depth of the node "A2" becomes 2. "root" is the special name assigned to the root node.

"Depth": is the depth of the node. The depth of root node is 0.

"Value": is the value of the node. The value is initialized to "-Infinity" for the max node and "Infinity" for the min node. The value will be updated when its children return the value to the node. The value of the leaf nodes is the evaluated value, for example, node "A2" has value 1.

The algorithm traverses from root node. The log should show both when:

1) The algorithm traverses down to the node.

2) The value of the node is updated from its children.

For example, the log shows value of node "B3" when traversing from node "B2". The log shows the node "B3" again when the node is updated from its children "A2", "A3", and "A4". In general, the leaf nodes have no children, but if the move from the leaf node ends up putting the last stone in the corresponding player's mancala, then that leaf node will have children as the player has to choose another move. Thus, for any node that has children, the log will show its value after each update. You can relate the reporting of the traverse log to DFS traversal of the Minimax game tree.

Alpha-Beta:

The program should output two files named ***"next_state.txt"*** showing the *next state* of the board after the greedy move and ***"traverse_log.txt"*** showing the *traverse log* of your program in the following format.

- The format of "next_state.txt" should be the same as shown above.

The format of "traverse_log.txt" will be similar to the one for MiniMax, but with two additional columns.

```
Node,Depth,Value,Alpha,Beta
........
........
........
```

Note: The Alpha-Beta traverse log requires 5 columns. Each column is separated by "," (a single comma). Five columns are node, depth, value, alpha, and beta. The description is same as with the MiniMax log. However, you need to show the alpha and beta values in the Alpha-Beta traverse log.

Guidelines:

- You can use C++, JAVA, or PYTHON to implement your code. Make sure that you code runs on Vocareum.
- You need to create **"mancala.xxx"** where 'xxx' is the extension for the programming language you choose. ("py" for python, "cpp" for C++, and "java" for Java). If you are using C++11, then the name of your file should be "**mancala11.cpp**" and if you are using

python3.4 then the name of your file should be "**mancala3.py**". The command to run your program would be the same as was for Homework-1.

- Input file is a text file ending with .txt extension.
- You will use Vocareum.com to submit your code.
- For any test case, it will be marked as correct only if both next state and traverse log are correct. (For greedy only next state will be checked).
- With the given description, we don't believe that multiple outputs are possible for any test case. If you are able to think of any such case, please let us know and we will make the necessary changes in the grading guidelines.
- The announcement about the competition will be done later. Note that the performance of your program in the competition will not affect your grade for this homework or the course in any manner.