AI Project Report

Project Description:

Mancala AI is a simple project where 2 players can play Mancala.

Both players can be either human players or AI with various levels of difficulty.

The game rules are as explained in this video and it has 2 modes:

- With Stealing
- Without Stealing

The Game and its AI are implemented entirely from scratch in python. The AI uses minimax with alpha-beta pruning where the depth of the tree search is given by this formula:

$$depth = 2 * difficulty\_level - 1$$

Utility Functions Used:

Minimax:

My implementation of minimax is a variation of the pseudocode given in the Wikipedia article on minimax.

The main differences is that instead of using the $max()$ function, we used an if condition so we will be able to also save the best move not just the best value. Also, the function that returns the children of the current state node returns both the child state and the move required to reach that state. This is used to save the best move as discussed above.

Minimax with Alpha-Beta pruning:

As with minimax, the alpha-beta algorithms is a variation of the pseudocode given in the Wikipedia article on alpha-beta pruning.

The differences are exactly the same as what was discussed above.

Both algorithms can be found in “/AI/minimax.py”

Static Evaluation:

We used a very simple static evaluation function. It runs under the assumption that the number of marbles in the mancala pockets are non-decreasing and that they alone would affect the winning player.

Therefore, the function returns the number of marbles in player0’s mancala minus the number of marbles in player1’s mancala.

This rule is only broken at the terminal states (at the end of the game) where, after moving all marbles to their respective mancalas, the function returns 100 (max score) if the number of beads in player0's mancala is more than the number of beads in player1's mancala. Otherwise, the function returns -100 (min score).

This is the simplest evaluation function that we could think of, however, we tried to search for more complex static evaluation functions but we weren't able to find any so we stuck with our function.

The static evaluation function can be found in "/AI/static_eval.py"

Players:

The game supports 2 types of players, human players, and AI players. That is why we decided to create 2 classes with an identical interface for each player type. The classes only have one function that takes the game state as an input, decides on a move somehow, makes the move and returns the new game state after the move has been played.

The human player class takes the user input to choose the move to use. If the user enters an invalid move, the class will ask the user again until the user selects a valid move.

The AI player class chooses its move based on the minimax with alpha-beta pruning algorithm discussed above. The depth of the tree is chosen based on the difficulty level entered at the start of the game.

Both player classes can be found in "/AI/player/human_player.py" and "/AI/player/ai_player.py" respectively.

Bonus Features:

We implemented 2 bonus features.

1. *Bonus Feature #2:*
   Support of various difficulty levels corresponding to different game tree depths.
2. *Bonus Feature #5:*
   Support a mode where 2 instances of the AI player can play the game against each other without any human intervention.

User Guide:

Here's a YouTube video that shows our game in action!

The simplest way to use Mancala AI is to run the binary executable in "/dist/mancala/mancala.exe". You can also run the game by running main.py using python.

Firstly, you'll have to select what type of player do you want player0 to be (either Human `h` or an AI `a`) If an AI was selected, you'll have to select a difficulty level for the AI. A good difficulty level to start with is level 5 as it's fast enough to play without any significant delay but is still not too stupid as to not make the game fun. If the AI is too hard for you you can always select a lower difficulty and vice versa.

Repeat for the other player (player1).

Finally, you can select whether you want stealing mode on or off.

You will find can also use one of the functions given in game.py to skip the UI part of the project.

After selecting the 2 players of the game, the start board state is shown, and the game begins.

To enter a move, you can press a number between 0 and 5 which represents the pocket that you will take marbles from. The rest of the rules are applied automatically.

Note that the pockets are numbered on the screen for both players so it should be easy for you to select the move you are looking for.

The game continues for each player until the end state is reached, where all extra marbles are relocated to their respective mancala and the final score for both players is calculated.

The Winner of the game is then declared and the game ends.

Team Members:

Name	ID	Section	GitHub Account
Pierre Nabil Attya Kamel	16E0056	1	https://github.com/PierreNabil
Michael Magdy Nasr Zaky	16T0076	3	https://github.com/Michael-M-Mike
Christine Magdy Gad El-Rab	16E0129	3	https://github.com/ChristineMagdy99
John Bahaa Helmy Shouhdy	1600459	1	https://github.com/John-Bahaa
Adham Nour El Wafaa	1600226	1	https://github.com/AdhamNour

The Project was split as follows:

The Game itself was created by Pierre Nabil and Adham Nour including all the rules of the game.

John Bahaa created the Human and AI player interfaces for the game and minimax functions. He also worked on the UI for the game.

The implementation of Minimax, alpha-beta pruning and the static evaluation functions were created by Christine Magdy and Michael Magdy.