CSE 140: Assignment #3

Summer 21

DUE Sunday, 11 JULY 23:59:59

MAX POINTS: 100. Released: — July 8, 2021

Introduction

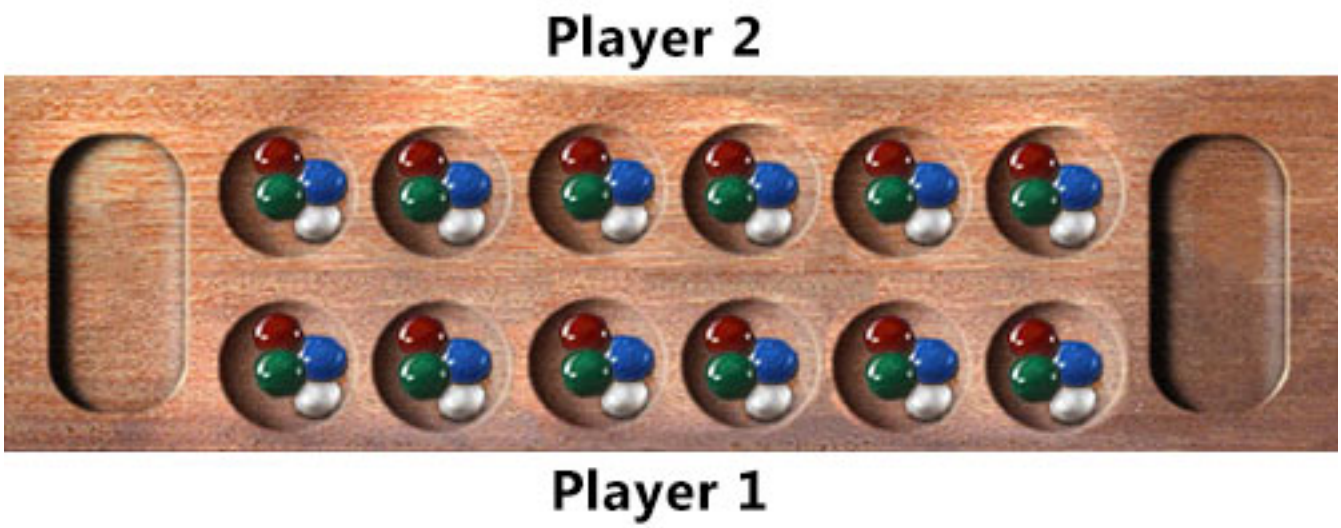
In this assignment, you are required to write a program that determines the best move for a player in a game of Mancala using minmax algorithm with alpha-beta pruning in python. Starter code is provided with key functionality already defined.

Mancala

Mancala is a generic family of two player turn based strategy board games. It is usually played on a board with 6 small pits per player and 2 Mancalas (larger pits). Stones/it's equivalent go in the pits, with some starting configuration. The objective is usually to capture some or all of the opponents stones. Here is the Wikipedia article on it: `https://en.wikipedia.org/wiki/Mancala` .

Rules for this assignment

For this assignment, we will use the following rules. Use the following image as a reference.

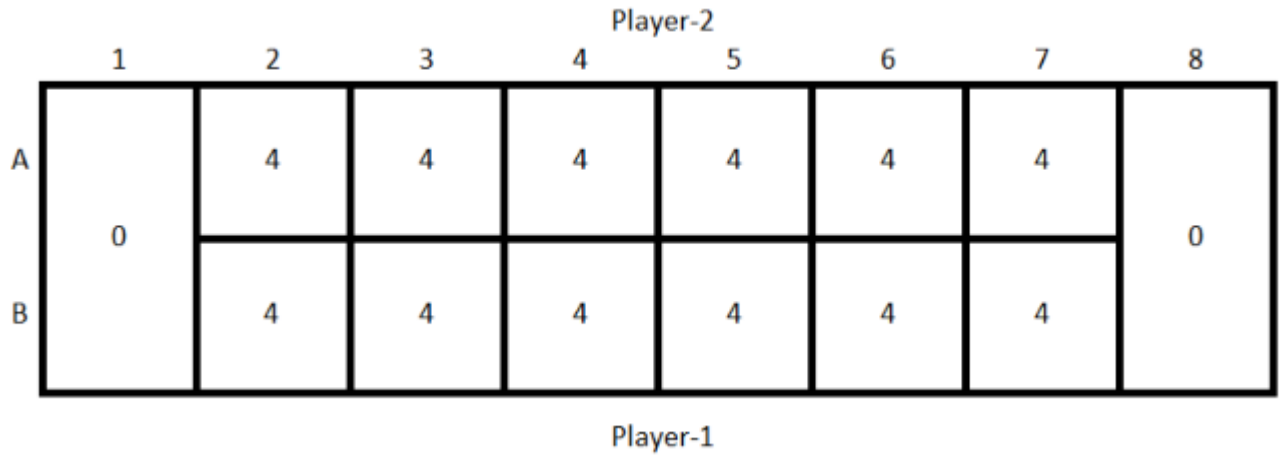


The basics of the game are the most generic version. The game is still a two player game. There are six small pits with the default starting condition of 4 stones in each pit. Player 1 owns the bottom row and Player 2 owns the top row as shown in the image. The two larger pits, called Mancalas, are special pits where the players accumulate their captured stones. Player 1's Mancala is on the right and Player 2's Mancala is on the left.

On a player's turn, they choose one of the pits on their side of the board (never the Mancala). They the play their turn by removing all the stones in that pit, then placing one stone from the removed bunch into each subsequent pit (including their Mancala but not the opponents Mancala) next to the chosen location

in a counter clockwise manner until they run out of stones from the removed bunch. If the player's last stone ends in their own Mancala, the player gets another turn. If the player's last stone ends in an empty pit on their own side, the player captures all the stones in the pit directly across the board from where the last stone was placed (ie. they remove all of the opponents stone from the corresponding pit and place it in their Mancala), as well as the last placed stone (in the empty pit). The game ends when one player cannot move on their turn, at which time the other player captures all of the remaining stones on their side of the board.

Allowed moves



Consider the game state shown above. Blocks B2-B7 are player 1's pits and blocks A2-A7 are player 2's pits. Block 8 is player 1's Mancala and block 1 is player 2's Mancala. In the current game state, Player 1 can choose blocks B2, B3, B4, B5, B6, or B7 and player 2 can choose blocks A2, A3, A4, A5, A6, A7. Assuming it's player 1's turn and they choose B5, the updated game state looks as shown below.

As mentioned above, it is possible for a player to make multiple moves on their turn. If a legal move is available, it has to be made (there is no way to pass a player's turn).

Evaluation of moves

The goal of the game is to collect the maximum number of stones in the player's Mancala (more than your opponent). The evaluation function for legal moves is simplified because of this to

$$E(p) = Number\ of\ stones\ in\ players\ Mancala - Number\ of\ stones\ in\ opponents\ Mancala$$

There can be ties when selecting the the best moves based on the above evaluation function. Ties are broken by selecting the node that is first in the position order in the above image (lower number per player).

Starter Code

You have been given starter code in the form of `mancala.py`. Code is to be written in python3. Comments are provided in the code to let you know what different functions are doing/need to do. You are required to write your code in place of the comments `#YOUR CODE GOES HERE`. The functions you need to write have been defined, along with their purpose in comments above their definition. You can define helper functions to make your implementation easier. Modules that are required have already been imported and the program can be completed without importing other modules. If you need to import other modules, you are allowed to do it as long as it doesn't implement the functionality that you need to implement in code. A brief explanation of the code is given below.

class Board

This class represents the board and hence the current board state. This is where most of your code will be written. Most of the code you will be writing will be functions in this class, with a few exceptions. The current state of the board is in `self.board` which keeps track of the number of stones in each pit including Mancalas. The `__init__` constructor constructs the default board state. Some useful things like print functionality (of the current board state) `print` and the current score of a player (number of stones in a Mancala) (`player1_points` and `player2_points`) have already been implemented . Another function that's already implemented is `no_moves_remaining` which returns `True` or `False` depending on if a move can be made.

find_moves()

You are to implement this function where you need to find all possible moves that can be made given the game state. This function will be helpful when implement the minmax and alpha-beta algorithms.

update_board()

This function updates the current state of the board given a position. This is where you would remove the stones from that position and place them in pits in a counter-clockwise manner according to the rules. You can define helper functions to make this job easier.

min_max()

This function is where you would define the code for the minmax algorithm. a default depth has been provided. you are allowed to add parameters and helper functions as necessary to implement this.

alpha_beta()

This is where you would implement the alpha beta pruning. default depth and default values for alpha and beta have been provided. You are allowed to define helper functions/additional parameters but not allowed to change the values already given.

calculate_heurestic_score()

This function is already defined, implementing the evaluation function mentioned earlier.

play_mancala()

This is most likely the only place you would need write code other than in the `class Board`. This function is where you use the functions defined in the class to calculate the best move and update the board based on that. As mentioned above, you are allowed to define helper functions and add parameters other than the ones provided. This function also takes in the `starting_player` as an integer argument and starts the game using that player. **Given a starting board (default is None and is initialized by the constructor), you are required to print up to 15 moves total(counting both players).** In other words, given a starting board state and starting player, what you need to do is calculate the best move for the player, update the board, switch players and repeat either until 15 moves have been done or the game ends.

__main__

This is the main function. This is where `play_mancala()` is called from. The starting player is received through the command line (already implemented). In case you want to test different starting conditions other than the default starting condition, you can create an `initial_board` condition and pass it to `play_mancala()` function.

Summary of deliverables

You must submit mancala.py on Canvas with the following functionality implemented:

- `find_moves()` function
- `update_board()` function
- `min_max()` function
- `alpha_beta()` function
- `play_mancala()` function

Other caveats:

- You must not change the parameters already defined in any function but you are allowed to add parameters.
- You are required to print the game state up to 15 moves(counting both players) that can be done legally according to the rules defined in this document.
- You must turn in code that works.
- When needed you are allowed to define helper functions.
- Please comment your code, similar to comments provided in the starter code.