Mancala Utility Function

Problem 1

Part B

The search tree for the board game *mancala* grows exponentially, making it unfeasible to determine the best action for a player by looking at all the paths that leads to a terminal state. One of the ways to solve this problem is by creating a *evalation function* that estimates how good or bad a particular game state is compared to other possible states. A evaluation function analyzes a game state based on the various aspects of the board configurations and predicts which game state is more likely to lead to a winning terminal state. Different evaluation functions have different methods for analyzing the board and often, the heuristic function that encourages the player to take actions that gives a *strategic advantage* to the player performs the best.

The player that has the most stones in her mancala at the terminal state wins the game. Based on this rule, one simple evaluation function can be to return the total number of stones at the player's mancala at the given state. This function encourages the player to take actions that will increase the number of stone at her mancala. While this evaluation function has proved to be a formidable opponent, it does not account for the subtleties in the game that allows a player to win the game by temporarily having fewer number of stones in her mancala during the game play. For example, the simple evaluation function does not account for the possibilities of capturing opponent's stones. So, the simple evaluation function is vulnerable to more complicated evaluations functions that *understands* that a game-state is not necessarily better just because it has more stones in the mancala.

Process of making (best-mancala-eval)

We identified the two major weaknesses of the simple evaluation function. Firstly, the simple-mancala function ignores the rule that all stones in the holes in the player's side of the board will be transferred to the player's mancala after the terminal state. Secondly, the simple-mancala function does not explore moves that allows for capturing the opponent's stones. Whenever a player takes an action that places the last stone in an empty hole in the player's side of the board, all the stones directly in front of the previously empty hole is placed in the mancala. With these two key insights, we proceeded to make an evaluation function to exploit these *blind spots* of the simple-mancala function.

Accounting for the distribution of stones: (stones-diff player)

We created the function `stones-diff` that calculates the total number of stones in the player's side of the board, the total number of stones in the opponent's side of the board and returns the difference. The reason for including the difference between the number of stones in each side of the board is to *incentivize* the player to take actions that will result in more stones in the player's side of the board. This strategy of *"hoarding the stones"* proved to be very useful as the evaluation function that accounts for both the total number of stones in the mancala as well as the difference in the stones distribution performed better than the simple mancala function.

Incentivizing to empty (some) holes

Try 1

A player needs to have empty holes to capture the opponents stones. Moreover, there is a higher chance of capture if a player has multiple empty holes. With this hypothesis, we made added the number of empty holes in our utility value by returning `(+ mancala-val stones-val empty-holes)`, where mancala-val is

the number of stones in mancala, stones-val is the difference of stone distribution and empty-holes is the number of empty-holes. This evaluation function, to our surprise, proved to be a disaster. It lost against the simple-mancala function both as player MAX and player MIN. When we analyzed the game-play, we found out a critical error in our incentives. Inadvertently, this evaluation function was encouraging moves that lead to a state with 6 empty holes. When the player empties all the holes, she has no more moves and the game reaches terminal state, with all the stones in the opponent's side of the board going to his mancala. This evaluation function naively rolled over when the opponent was "hoarding the stones".

Try 2

Try 1 was a massive disaster. We had to figure out a way to incentivize emptying some holes without self-destructing. After playing mancala online with each other, we developed the intuition of keeping at most 3 holes empty[1]. So, we modified our evaluation function to behave the same way as before for states with less than 4 number of empty-holes but when the number of empty-holes was more that 3, we charged a penalty of `(- empty-holes 3)`. This meant that our player was more likely to keep at most 3 of her holes empty and rarely explore states with more than 3 empty holes. This modification proved to be a success.

Preventing captures

During the process of developing (best-mancala-eval), my partner and I worked on different ideas and after completing the implementation of an idea, we skirmished. After multiple battles, we found out that our evaluation function can be very shortsighted because of a fundamental flaw in our evaluation model. Both of our evaluation functions were overly *self-absored* and thus, failed to see if the opponent could capture our stones. Our evaluation function also had to *keep an eye* our for the number of empty holes in the opponent's side of the table so that our player can sense danger 5 (the number of plies) steps ahead. We introduced a penalty of the magnitude of the maximum the number of stones in front of an opponent's empty hole. This addition made our evaluation more aware of any external threats of capture and hence, a little more competent.

It is important to note here that just because an opponent has an empty hole does not mean that he can capture our stones. Most of the time, the opponent does not have a move that allows him to take actions that place the last stone in an empty hole. In this sense, it would be more effective to keep track of the indices that the opponent's last stone can reach through each of his actions. We, however, did not proceed with this idea fearing that it would be computationally demanding and potentially disqualify our evaluation function from the **tournament**.

Explored and discarded ideas

Offensive mode vs. Defensive mode

We explored the idea of using different evaluation strategy during different points of the game play. For example, during the earlier stages of the game, when both players have very few stones in their mancala, we posited that it would benefit to play more *aggresively*. Aggressive play would comprise of valuing potential capture more that the number of stones in the mancala. To contrast, in the end game, we would play more defensively if we have more stones in the mancala compared to the opponent. Defensive play would comprise of removing all incentives for having empty holes and focusing more on mancala-val. We wrote multiple functions for different stages of the game to value board configurations differently depending on other circumstances. Unfortunately, we could not make any substantial progress by adding more components in our evaluation function. In fact, beating the simple-evaluation function was the *litmus test* for our evaluation functions and surprisingly, simple-evaluation function is a very formidable opponent!

[1] http://play-mancala.com

Testing different evaluation functions

Any evaluation that wants to be the (best-mancala-eval) has to be first beat the simple mancala function. After it passes this test, it gets to participate in a battle tournament along with other evaluation functions that have won against the simple mancala function as well. My lab partner and I worked on different ideas and played each other.After we agreed on what components to include in the calculation of (best-mancala-eval), we tested the evaluation function with different weighting for each component against a different weighting of the the same components. The (best-mancala-eval) was developed by multiple iterations of such skirmishes both to decide what components to include and what weights to apply for each of the components.