Project 2 Writeup

Program Design

Architecture

- `mancala.py`: Simulation of the mancala game.
- `play.py`: Run the game.
- `strategy.py`: Four different strategies returning action given state.

State

- State: A numpy array with shape (2, PITS+1). [0,0] and [1, PITS] means the store while others mean the pits.
- Action: An integer in range [1, PITS].
- Turn taking: If stopped at store, keep the same player, otherwise change the player(`player = 3 - player`).
- Equivalent actions: Choose a random action to include randomness.

Functions:

- **Utility functions**: If win, return infinity. If lose, return negative infinity. If draw, return current heuristic.
- **Heuristics**: Stones in current player's store.

Member contributions

- Guangyu Sun: Finished the project independently.

Experiments

- Result for 100 battles between different players:

- Number in bracket means the search depth.
- Time is the time used in the last running.

```
Random vs. MiniMax(3)
--------------------------------------------------
Draw: 5, Player 1 wins: 8, Player 2 wins: 87
Player 1 times: 0.002957, Player 2 times: 0.227260
# MiniMax will win most games but take longer time.
==================================================
Random vs. AlphaBeta(3)
--------------------------------------------------
Draw: 3, Player 1 wins: 11, Player 2 wins: 86
Player 1 times: 0.004177, Player 2 times: 0.489204
# AlphaBeta will win most games but take longer time.
==================================================
Random vs. Random
--------------------------------------------------
Draw: 10, Player 1 wins: 42, Player 2 wins: 48
Player 1 times: 0.002206, Player 2 times: 0.001133
# Random players have equal win rate.
==================================================
MiniMax(3) vs. MiniMax(5)
--------------------------------------------------
Draw: 5, Player 1 wins: 47, Player 2 wins: 48
Player 1 times: 0.403912, Player 2 times: 3.770179
# Increasing the depth limitation takes more time
# but have slight effict on the result.
==================================================
AlphaBeta(3) vs. AlphaBeta(5)
--------------------------------------------------
Draw: 3, Player 1 wins: 44, Player 2 wins: 53
Player 1 times: 0.211810, Player 2 times: 1.148984
# Increasing the depth limitation takes more time
# but have slight effict on the result.
==================================================
MiniMax(5) vs. AlphaBeta(5)
--------------------------------------------------
Draw: 1, Player 1 wins: 53, Player 2 wins: 46
Player 1 times: 6.041353, Player 2 times: 0.804062
# AlphaBeta donesn't increase the win rate, but
# reduced the running time.
==================================================
AlphaBeta(3) vs. AlphaBeta(10)
--------------------------------------------------
Draw: 3, Player 1 wins: 52, Player 2 wins: 45
Player 1 times: 0.204834, Player 2 times: 81.319638
# Going too deep takes tens of time but didn't work
# under this heuristic.
==================================================
```

Discussion

- Any strategy can't always win due to the time and space limitation.
- Heuristic helps the searching and you won't need to go to very deep.
- AlphaBeta will reduce the running time but won't increase the win rate compared with MiniMax.
- Utility functions for draw controls how aggressive the strategy will be. In my setting, it will be less possible to be draw.