CS165A MP2 Report
Shunji Zhan

Architecture:

I have four classes:
Game: the main game logic, including dealing with different options, and direct player to put chess on the board.
InputHelper: This class help deal with different flags that the user can set when initialize game, such as -l, -n
Board: This class manages all the board status, it has functionalities to put chess on board and update board states, print board status as human readable format to console, judge if this is a tie or win, etc.
Player: This class represent each player, so in each game there will be two player instances, one is computer and one is human. For computer player, it has the most important function that can determine which position is the best position to play, given the current board status.

Search:

Algorithm:
I first calculate potential positions: before making decision, the computer player will first collect all potential positions to put the chess. I used all position that has other chess within one block as potential positions.

I used min max tree as my searching algorithm. For each potential move, I will first simulate this move on the board, and send the current board status to computer player's decide position function, the player will use heuristic to score the current board. It will go over all the potential positions, and will return the highest score move.

For depth d >= 2, I will simulate d moves, if it is adversaries move, I will time the return score by (-1), so the best move for the opponent is the worst move for us. With this recursive call, the top level will still return the best move, after thinking about d moves.

Heuristic:
I first scan all horizontal line, vertical line, and diagonals, and score every single six-in-a-row block. I assigned each different situation some particular score, for example if five chess are in continues, it is a win and will score 99999999, and four continues chess will score either 10000 or 5000, depending on if there is any opponent chess blocking on side. I listed out almost all situations that I can think of, and the total score of one color is the total score by each six-in-a-row block. The total score of a board is the total score of my chess minus total score of opponent's chess.

Optimization:
- I used to try two blocks to existing chess as potential positions, but that will decrease the

speed a lot, so finally I decided to use one block.
- In heuristic if there is a five-in-a-row then return 99999999 directly, don't need to check other position, since it is already a win.
- The depth I used was 2. But for the first couple steps, since the potential positions are really small, I used depth 4, later if the potential positions are relatively small, I used 3. This can increase the performance.
- There is a weight assigned to each position, so that if two position has same score, it will tend to put chess on the position closer to the center.

Challenges:

- I didn't use Java for a while, so it took me some time to get familiar with it. But fortunately, I still remember many of the concepts.
- At first I was confused which heuristic to use, I tried to use calculate the score based on how many continues chess there exist, but I found this method to naïve, because even the chesses are not directly connected, it can still be a good position to put. So, I search many algorithms online, and finally decide to use the heuristic I mentioned above, which greatly increase the power of my AI.
- Because I used an array the represent the board, indexing from 0, but the board start from 1, so I encountered many bugs when transiting array to board. So, I drew out the whole array, with arrow pointing which index correspond to which position in the board. And as I wrote more, I got familiar with the array indexing with the board and didn't have the problem anymore.

Weakness:

- there are still much more combinations of the six-in-a-row scan, I only had part of it, so some situations on board cannot be analyzed. I can research on more other's Gobang example, seeing how they deal with different combinations.
- I didn't use alpha-beta pruning, so usually if the depth exceeds 2, it will be extremely slow, I could add alpha-beta pruning to make the search much faster.
- Score assigned to each six-in-a-row situation can be adjusted, since now the score is only based on my experience, I can adjust them based on its performance, in order to find the best score for each situation.