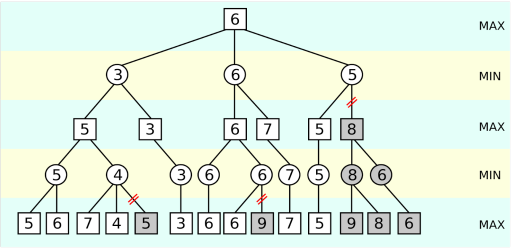
Pentago-swap Report

My program works based on alpha-beta pruning algorithm. At first, I analyzed the game and found that the center((1,1),(4,1),(4,4),(4,1)) of each quadrant is a key to win the game. Those four pieces can connect 5 pieces in a row vertically, horizontally and diagonally by placing other pieces around the center piece from 8 directions(top, bottom, left, right, top left, top right, bottom left, bottom right) no matter how I swap each quadrant. But for other points of each quadrant, that is certainly not the case. For example, I can connect 5 pieces in a row vertically, horizontally and diagonally by placing other pieces around the piece of (3,3) from 8 directions. However, after swapping bottom right quadrant and top left quadrant, the piece of (3,3) becomes (0,0). I can now only connect 5 pieces in a row vertically, horizontally and diagonally by placing other pieces around the piece of (0,0) from 3 different directions(right, bottom, bottom right). As a result, I believe placing my first two pieces at the center of any quadrant can give me a big advantage and I also assume that my opponent does the same thing. After the first two steps, I run my alpha-beta pruning algorithm with a depth of 3. As I have an intuition when I am playing the normal pantago game that if my pieces are less separate than my opponent's, I have a better chance to win. The evaluation function of the game board is based on how close each of my pieces is to each of my other pieces and also how close each of my opponent's pieces is to each of my opponent's other pieces. Specifically, I calculate the sum of the absolute difference of x and y value between each two of my pieces and each two of my opponent's pieces. The larger the sum is,

the worse the situation is for both player. When it is my turn, the value of that game board is my opponent's sum subtracts my sum. When it is my opponent's turn, the value is the other way around. If the game board is over and I win the game, the evaluation function simply returns 100000. And if the game is over and my opponent wins the game, the evaluation function simply returns -100000.

The technical approach that I am using is alpha-beta pruning algorithm. Alpha-beta pruning algorithm is an advanced version of minimax algorithm. Minimax algorithm starts from an empty board and lists all of the possible situations of the second move as a child node of that empty board node and expand all of the following moves as a tree until all of the situations are explored. Each node has a value which tells how good or bad that situation is to the player. Whenever it is the player's turn, the player choose the largest value. And whenever it is the opponent's turn, the player choose the smallest value. Alpha-beta pruning does the exact same thing plus it truncates a part of the search. For example from the graph below, unlike minimax, alpha-beta pruning does not count the second 5 node in the last row because the parent node of that 5 is currently 4 which asks for a minimum value from its children and the parent node of that 4 is currently 5 which asks for a maximum value from its children. As a result, it does not need to take the second 5 in the last row into consideration as it can not be smaller than 4 and larger than 5 at the same time.



(From Wikipedia https://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning)

The advantage of my algorithm is that it can compute very fast and make the the best move possible within the next three steps according to my evaluation function. However, as the program is based on alpha-beta pruning, it assumes my opponent takes the best move every time according to my evaluation function which is probably not the case unless my opponent runs the same algorithm as I do. Because of the limit for running time, my program is not able to find all of the possible situation of the game every turn which means it can not think ahead too much. That definitely influences the performance of my agent especially when there are still many empty spaces on the game board. Another thing that needs to be noticed is that if my opponent knows that I will only place the first two pieces at the center of any quadrant, he or she can place the first two pieces together in the same quadrant and take the advantage of that.

If there is no limit for memory and time, it is totally possible to run the alpha-beta pruning for the entire game board and make the move by discover every single one of the possibility and that can make the agent think ahead much more than now. If there is only a limit for memory but not for time, I can try monte carlo tree search and do as much random move as I can as long as it is under the memory limit. If there is a limit for both memory and time, one approach that I tried is to store all of the situations and the best move of that situation in a text file. In detail, first, my agent runs alpha-beta pruning algorithm for all the possible situations on the game board until the end of the game and store each game board together with a best move in that situation in a text file before the tournament or use the first 30 seconds during the tournament(if that is long enough). After that, when my agent tries to choose a

move from the current game board, instead of calculating all possible moves, it can simply read the data from the text file, find the specific game board and do the move that is stored with that game board. By doing that, my agent can think ahead much more than what I implemented for now because it does not need to explore anything or run the evaluation function which saves a lot of time. One thing that can also be improved is my evaluation function. Though for most of the time, the closer my pieces are to each other, the higher winning possibility I have. There can be some cases that does not follow this intuition as we can swap each quadrant after placing a piece. Thus, I should make the evaluation function not only consider the current game board but also consider the game board after doing all possible moves.