- **The program is written in Python2.7**

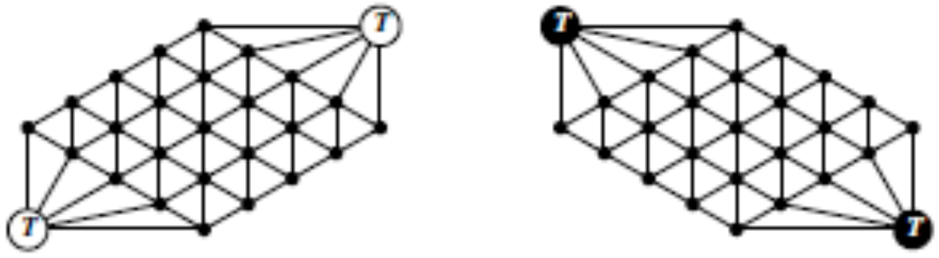
- **Game winner: get the shannon graph of the two players, if the shortest path length is 1, then the unbroken chain of pieces between one player's two sides exists, the player is the winner.**
- **termination conditions: for Alpha-beta search, the termination is when there is a winner. The winner is predicated when the heuristic is +/- 512(8^3). And for Monte-carlo tree search, the termination is when the board is fully occupied by nodes. Because it makes no difference for the final winner to search until the board is fully occupied, this will save the time used for checking winner after each move.**
- **I applied "must-win" and "must-occupied" move for both the algorithms. "must-win" move is the move that leads the player to win in the next configuraiton. "must-occupied" move is the move that is between the bridge of the player and when one of the connection is occupied by the opponent. Besides, for the first move of the first player, I assume it would occupy the most promising cell—cells in the middle of the board.**

Alpha-beta search

1. **Evaluation function**

I use the shannon graph to represent the board, and for the heuristic, when the player is the first one to move, I assume it is an attack player, and I use "size of board/8 + 1 – (second) shortest-path between the source and the target” as the heuristic. The (second) shortest-path is large when the node is inferior, and the smaller the (second) shortest-path, the bigger the heuristic is. I use second shortest-path because I assume the opponent will occupy the shortest-path node.

And when the player is the second one to move, I assume it is a conservative player, and I use “Length of the shortest path between the opponent’s source and target” as heuristic, because the larger this heuristic is, the inferior the move is for the opponent, so as to make this move a better choice for the current player.



Strategy: The opponent tries to maximize the player’s shortest-path length, i.e, minimize the player’s heuristic. The player tries to maximize the heuristic.

Cut-off strategies:

According to [1], Theorem 3.2, “For a board state of the Shannon game, an uncoloured node is live if and only if some minimal short set contains it.” Therefore, I will calculate every shortest path of the current hex board, if one cell is not contained by any of the minimal paths, I will cut-off this node because it is not live (dead or inferior).

[1] Yngvi BjÅNornsson, Ryan Hayward, et.al, Dead Cell Analysis in Hex and the Shannon Game, Graph Theory Trends in Mathematics, 45–59, 2006.

2. Threshold and Search Expansion function

Threshold of search depth is 3.

The search will first expand the more promising nodes, i.e., node that is a win move, node that connects a bridge of the current player when the other connection node is occupied by the opponent, node that forms a bridge of the current player, node that forms the opponent's bridge…

3. Statistics and games played by human against the program

This part is attached in the statistics folder. (name of the file is made as "first player-second player_*who* wins", e.g., "alphaBeta-human_human wins")

Although the heuristic is not perfect, alpha-beta can win people even when people implements very good moves.

Monte-Carlo tree search

1. Search parameters

(a) threshold for the expansion depth: 3. For small-size hex games, such as 4*4 or 5*5, MCTS can explore more than 3 depths. But for 8*8 hex games, for the first 3 depths, the simulation times is about 60^3, but the simulation times for 60s is only 4000-5000. Therefore, depth 3 is not expected to be explored for the first several rounds. Only after the board is occupied by some nodes, nodes after depth 3 can be expanded. That's why in the first several rounds, the simulation times is the same as the max size of the search tree.
(b) the simulation method employed: random choose next move.
(c) the evaluation function used for selection steps: UCB1
(d) your expansion policy: expand one node at each iteration
(e) the policy for implement for picking the most promising move for each iteration: highest win rate
I choose to use highest number of simulations, max number of wins, highest win rate and some combination approaches, but the highest win rate comes out the most reasonable moves. In most of the cases, highest win rate move is also the highest visited move.

2. Statistics and games played by human against the program

This part is attached in the statistics folder. (name of the file is made as “first player-second player_*who* wins”, e.g., “alphaBeta-human_human wins”)

For small size hex game (less than 6), monte-carlo tree search works perfectly, it can find optimal move. But for 8*8 case, it seems at first cannot identify the optimal move, or maybe it is the optimal move but as a human, I do not know. If do has a high win rate when playing with human.

Monte-Carlo tree search v.s Alpha-Beta Search

As illustrated in the homework spec, MCTS is knowledge-light, whereas Alpha-beta is knowledge-heavy. The heuristic of Alpha-beta is its indicator of the intelligence. The heuristic should take into consideration the dead cells (and captured cells, and other vulnerable cells), bridge, both the minimum path of the player and the minimum path of the opponent. I tried to combine some of them as the heuristic, but the effect is not good. So I finally choose to apply different heuristic approach for first/second player. And I tried to make those bridge cells, must win cells and cells between bridges as the first moves to be evaluated, so that alpga-beta can cut lots of branches after those more promising moves have been evaluated. And I also cut off the dead cells.

However, MCTS does not rely on heuristic, it only depends on the random simulation results. Therefore, the selection step is very important for MCTS. UCB1 seems work for small size HEX game, but for 8*8 board, sometimes the selected node does not make good sense. There are approaches such as AMAF (All moves as first) that can make MCTS work better, but the time is not enough for me to explore it. Basically, UCB1 works not bad. Expecially as the game continues, MCTS will work better and better, because nodes to be evaluated decrease.

I limit the time as 60s. And play AI against MCTS. In fact, as I set the depth limit of 3 for AI, it only uses 30s or so to get next move. However, since the heuristic takes a lot of time per evaulation, search more than 3 depths will cost minutes. Considering the time cost, I only search 3 depths. But it will definitely find a better solution if the depth limit is set to be large.

The statistics of the win and loss between alpha-beta and MCTS is attached in the statistics folder. (name of the file is made as "first player-second player_*who* wins", e.g., "alphaBeta-human_human wins")

Because there are both improvement space for alpha-beta's heuristic and mcts's selection policy, neither alpha-beta nor mcts search is perfect. When mcts is the first player to move, it has 60% or so win rate. And when alpha-beta is the first player to move, it has 90% or so win rate. For my implement, alpha-beta is better than mcts, it is because the heuristic is chosen well and also because I did not implement inferior node cut-off strategy for mcts. This is becaues implementing such strategy will sacrifies time used to simulate (also because the due is tonight and I do not have enough time…). I think mcts can be improved using AMAF and inferior cells cut-off. And if both set optimal, mcts should beat alpha-beta search algorithm. That's why for most of the HEX competitions, the winners are mcts players. Statistics are approximately the real probabilities when samples are large.