ECSE 526, Winter 2015, Assignment 1
Yulin Shi, ID 260629628

1 Design of game-playing agent

1.1 Connect-3 game analysis:

1. the rule of the game is simple and deterministic, thus it's possible for an agent to build an accurate model to mirror the environment.
2. in every round of game, there are only limited legal options of move for each players. The maximum option is 16 which equals 4 pieces identified from 1 to 4 for white player and -1 to -4 for black player timing 4 directions. Therefore, it's possible to predict several steps by searching the a tree.
3. there is no end for the game if no player win (connect 3 pieces in line), thus the search tree can grow into infinity.

1.2 agent architecture design

1. As is shown in Figure 1, our designed agent is a model-based, goal-based agent. It keeps track of the world state as well as a set of goals (win) it is trying to achieve, and chooses an action that will (eventually) lead to the achievement of its goals.

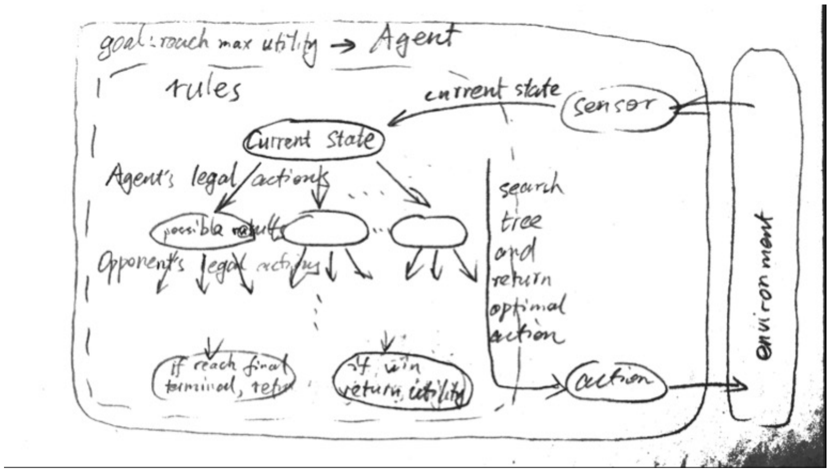

Figure 1: time-consuming between minimax search and alpha-beta pruning

2. The *sensed information* of the agent can be: 1. the initial state and all of the movements of its component, i.e. which piece and which direction did the component move; or 2. current state of the game board. My designed agent sense the first.
3. The *action* is the identity of the four pieces of its part to move, and the direction to move.
4. *Planning by search.* It will use minimax recursive algorithm suggested by Russell's textbook Fig 5.7 to explore the possible several steps ahead, to determine how to reach a possible win solution.

If someone win, terminate this branch of search, label the opponent win as 10, label the agent win as -10; if not, go on search the next depth. Until reach the search depth limit (or limited by admissible search time). It's definitely possible that there is no win or lose in the terminal depth of this multi-leaf tree. At this case, calculate and return the heuristic of this state (game board) valued by the number of runs.

"minimax" search is applied in my design. It's a depth-first search. Before reaching the depth limit, it will dynamically expand branches; after searching all leafs of a root, a node will release all of it's leafs and return a 3-dimensional vector containing action and minimax utility information to it's root. This search method is complete, optimal and foremost memory-saving. What's more, a maximum time limit is set here, before reaching which the searching algorithm will update it's searching limit one level by one level.

1.3 Java script realization

My designed agent is realized with a framework of three hierarchical level.

1. The highest *main class* will create two players (human to human, human to agent, or agent to agent), and provide environment for two player to play successively. It will call "Agent" class.
2. The *"Agent" class* has a reference "gameBoard" to model the environment (game board state). a "Agent" collects action of its opponent, calls "Minimax" class to search for action which give optimal utility, then outputs the optimal action.
3. The lowest *"Minimax" class* will import public reference "gameBoard" (state) from "Agent" class, search for a piece and a direction which gives maximum or minimum utility value, then output the identity of selected piece and selected direction.

More tutorial can be found in the "README.txt" file archived with my Java files.

2 Part I

2.1 Graph the average time

Graph the average time (during the opening few moves) that it takes your program to perform the state space expansion and evaluation of the game tree for different depth cutoffs.

In figure 2, for each depth, the average search and evaluation time of the first 5 gaming turn is counted and plot in the logarithmic coordinates. (Experiment environment: OSX V10.9.4; processor: 2.6GHz Intel Core i5)

2.2 alpha-beta pruning

Analyze how the performance changes as a result after you have implemented alpha-beta pruning.

According to the complexity estimation of "minimax search" whose complexity is $O(b^m)$, and "alpha-beta pruning" whose complexity is $O(b^{m/2})$, the magnitude of search time of "minimax search" can be the square of that of "alpha-beta pruning". In fact, it can be seen that, in the logarithmic coordinate, the height of dashed curve is approximately twice the height of solid curve, which reinforced the complexity estimation.

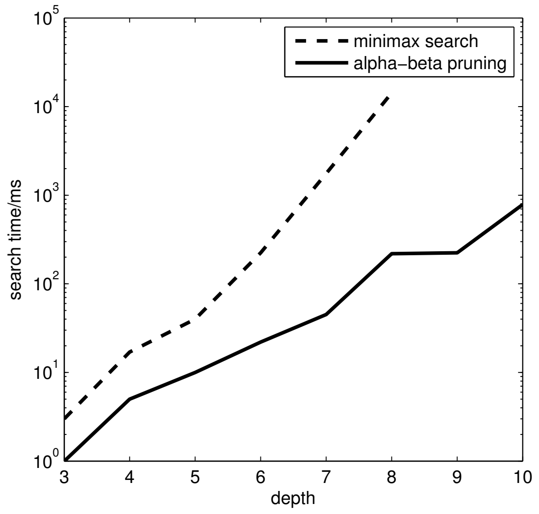

Figure 2: time-consuming between minimax search and alpha-beta pruning

3 Part II

3.1 improved evaluation function

Design an improved evaluation function for non-terminal nodes and demonstrate that it leads to superior performance.

My designed improved heuristic evaluation function is:

$$H = (\sum pair_{white} - \sum pair_{black} + 10 \times win_{white} - 10 \times win_{black})(1 + \frac{1}{depth + 1})$$

wherein, $win_{white}, win_{black} \in 0, 1$. They are added to "$pair$" so as to consider them together; they are weighted by 10 because their utilities are more important.

The term "1+1/(depth+1)" is added here to increase the utility of leafs in the shallower levels. In fact, it leads to similar effect to that of the additive term "h" of "f=g+h" in the “8-puzzle” problem. (see Russell’s textbook)

3.2 complexity of the heuristic evaluation

Discuss the tradeoff between the (computational) complexity of the heuristic evaluation function and the depth to which your agent can expand the game tree within the time limit allowed.

Before applying heuristic evaluation function, only "win" or "lose" node in a tree can terminate the depth-first "minimax" search and return with utility value 10 or -10. However, it always possible that there is no win or loss to terminate all tree expansion before the time limit, i.e. the majority of the returning utility is 0, which do no help to decision making. Therefore, in my heuristic evaluation function "Utility", each successive white or black piece pair will also add or toll one credit to/from the scalar utility variable. Consequently, there is always some utility returned from shallow leafs to do "minimax" search. The only *disadvantage* of heuristic evaluation is, when checking if there is "connect-3", "connect-2" will be evaluated too. This doubled the time consumption.