CS221 Project Progress Report
AI Agent for Chinese Chess

Li Deng[a]

[a]*SUID: dengl11*

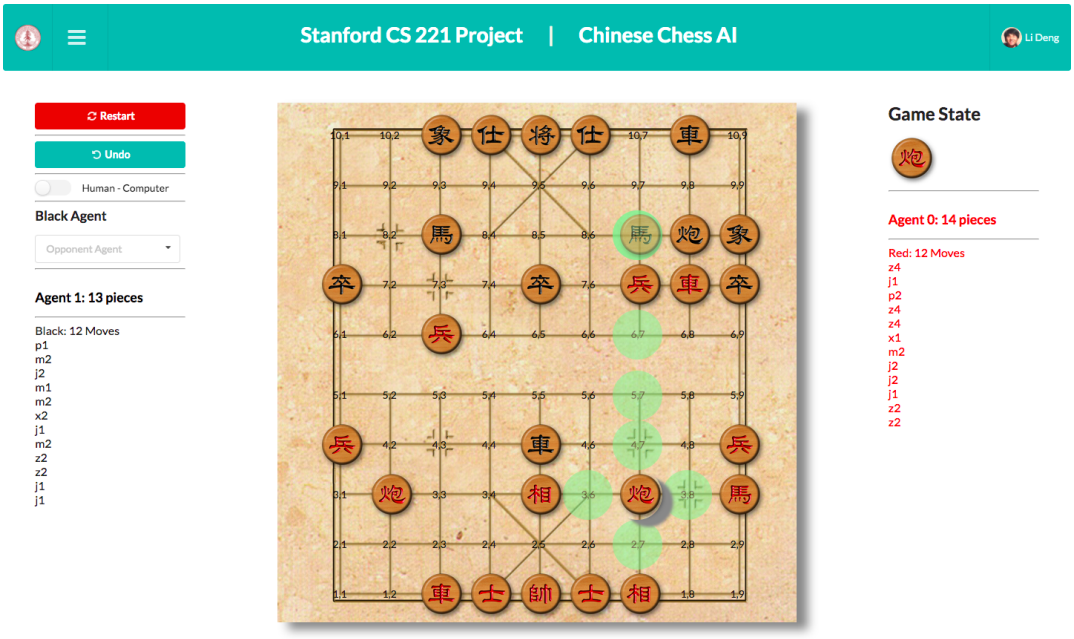
Abstract

As an intermediate step toward the final AI agent for Chinese Chess, two different strategies and two game modes have been implemented: Greedy Agent and Evaluation Function Agent in Human-mode and simulation-mode. In Human-mode, where human play plays against computer with selected strategy, the computer with search tree depth up to 3 can already have a winning rate of 0.82 against the writer. In simulation-mode, where computer agents with different strategies play against each other, search tree agent with depth up to 3 can beat the greedy agent almost all the time. To further speed up the game tree search speed, *node.js* is experimented for asynchronous search by its *async* library functions, i.e., since search of branches of game tree are completely independent from each other, search on each branch can be asynchronously completed, which greatly decreases the computation time. The agent is now reasonably intelligent in playing the game, and satisfactory results are obtained so far.

Keywords: Chinese Chess, alpha-beta pruning search, async

1. Introduction

Chinese chess (Xiang Qi) is one of the most popular board games worldwide. Having a long history, the modern form of Chinese chess was popular during the Southern Song Dynasty (11271279 A.D.).Chinese chess is a two-player, zero-sum game with complete information. Chinese-chess expert knowledge started to be developed some 800 years ago. Nowadays, the world has many excellent human players. Yet, already now, the strength of the best Chinese-chess programs can be compared to that of human players notwithstanding the fact that the game is considered rather complex. Detailed game rules can be found at https://en.wikipedia.org/wiki/Xiangqi#Rules For this project, Chinese Chess is implemented with Web interface by *Angular2* and web server by *Node.js*. For the reader's overview, below is current web interface for Human-Mode.



2. Strategies

2.1. Base Case: ***Greedy Agent***

Each piece is given a *PieceValue*, and the *Greedy Agent* takes the move that can capture one opponent piece with the greatest value. If no moves can capture opponents' pieces, then just take a random move.
Value of Pieces are based on Ong, Chin Soon, et al. "Discovering chinese chess strategies through coevolutionary approaches." 2007 IEEE Symposium on Computational Intelligence and Games. IEEE, 2007.:

Piece:	King	Assistant	Elephant	Rook	Horse	Cannon	Pawn
Value:	6000	120	120	600	270	285	30

2.2. Evaluation Function Agent with alpha-beta pruning

Since Chinese Chess is a 0-sum game, evaluation of state for an agent are defined as difference between scores of agent and its opponent. Scores of agent is the sum of piece values and positions values of all pieces.

$$\begin{aligned} Eval(state, agent) = & \sum_{piece_i \in agent} (pieceVal(piece_i) + positionVal(piece_i)) \\ & - \sum_{piece_i \in opponent} (pieceVal(piece_i) + positionVal(piece_i)) \end{aligned} \tag{1}$$

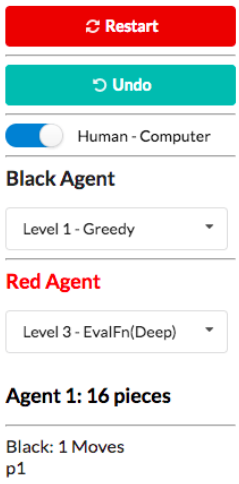
Where position values of pieces are also based on the same paper above. For example, position values for Rook is the following matrix:

14	**14**	**12**	**18**	**16**	**18**	**12**	**14**	**14**
16	**20**	**18**	**24**	**26**	**24**	**18**	**20**	**16**
12	**12**	**12**	**18**	**18**	**18**	**12**	**12**	**12**
12	**18**	**16**	**22**	**22**	**22**	**16**	**18**	**12**
12	**14**	**12**	**18**	**18**	**18**	**12**	**14**	**12**
12	**16**	**14**	**20**	**20**	**20**	**14**	**16**	**12**
6	**10**	**8**	**14**	**14**	**14**	**8**	**10**	**6**
4	**8**	**6**	**14**	**12**	**14**	**6**	**8**	**4**
8	**4**	**8**	**16**	**8**	**16**	**8**	**4**	**8**
-2	**10**	**6**	**14**	**12**	**14**	**6**	**10**	**-2**

3. Implementation and Results

3.1. Game Design

At a certain time point of the game, we have a corresponding *State*, which contains two *Agent*s and one number *playingTeam* indicating which team is in control. Each agent has a certain number of *Piece*s, containing their names and current positions. For a state a a legal move from agent, a successor state is generated for game tree search. Agents with different strategies are child classes of the parent class **Agent**, sharing all common methods like getting legal moves, and capture opponent piece, and each strategy has its own method *computeNextMove* for the next move to be taken, which shall be called when the strategy selected. Also, in Human-Mode, last state is stored for user to *Undo*, like the following:



4. Current Result

4.1. Human-Mode

Greedy Agent is fast in taking a move (less than 1s), and has a winner rate agiant the author less than 10%. **Evaluation Function Agent** with search tepth 3

takes around 5s for taking a move, and toward the end of the game, it may take less than 3s for a move with less legal moves. Since ordering of branches affects pruning effect for alpha-beta pruning, moves by more important pieces are considered before moves by less important pieces, since more important pieces tend to have greater effect for the game state, thus imposing a tighter bound on the search. This is also usually the ordering of human players in thinking about next move.

4.2. Simulation-Mode

Evaluation Function Agent can beat **Greedy Agent** almost all the time in the simulations.

5. Next Step

- **Monto Carlo Tree Search**
  Due to the large branching factor of Chinese Chess(38), Evaluation function is limited to very shallow tree search. And with Monto Carlo Tree Search, more deep tree search can be realized.

- **Fully Implemnetation of Game Tree Search with Async**
  Due to limited time so far, asynchronous game tree search by *Anysc* is still being experimented. If it proved to be practical, then game tree search can be speed up significantly, since branch factor will not matter any more.

- **Reinforcement Learning**
  Right now evaluation of state is merely based on piece values and their positions. More features, like the relative positions between Cannon and Horse, are also pretty important in evaluating the state. Domain knowledge may be needed for defining the features, and Q-learning can be used for learning their feature values from simulations. With more features added and learned, the agent is expected to be more intelligent.

6. Github Repo

The reader is welcome to git clone this project, and have fun with it. The project is hosted on Github: https://github.com/dengl11/ChineseChessAI