**CS154 Project Proposal**

**Title- Chess**

**Team members-**

* Sagar Jha- 110100024
* Raghav Gupta- 110100092
* Deepali Adlakha- 11D170020

**Brief Problem Description**

We intend to implement a one-player (i.e. Human vs Computer) version of the game chess. This would include an engine back-end with a GUI front end for the user to play Chess games implemented with Racket GUI Toolkit.   
Apart from basic chess-playing functionality, we also intend our application to-

1. Be able to save games in and retrieve games from PGN (Portable Game Notation) format.
2. Given a chess position (FEN) , construct the board and allow the game to continue from there.
3. Allow for timed/blitz or unlimited time games.
4. Evaluate and display when one player might have a threat, in particular detects checks, drawn positions, checkmates etc.
5. Offer multiple difficulty levels for Human- Computer games
6. If possible, incorporate computer-computer games, with usage of external chess engines.

**Solution Idea**

To calculate the best move at each stage of the game, our mainstay would be the Minimax algorithm (with alpha-beta pruning for efficiency).

At any stage, in a game of chess, there are typically 40-50 moves that a player can make. So, we would generate a partial game tree with the current game stage as the root, and all possibilities for the board state that can be reached from there after, say, some k plies (half-moves) as other nodes. Then the board position at the leaves can be evaluated.

Now the Minimax algorithm comes into play. First we'd assign values to the board positions at each leaf (there we involve a heuristic to “evaluate” a given board position). Then, to evaluate the position at a node, we recursively evaluate all its children and, depending on whether the node under consideration is at odd depth or even depth, the value at the current node would be the minimum or the maximum of the evaluated values of the children respectively. This allows us to pick a reasonably good move at each position.

Note that the move is “reasonably good” because we are limiting the depth of our search tree to some fixed value (varying this maximum depth would result in changing difficulty levels for the game; a deeper search tree implies a better move returned by the algorithm).

Once this is done, the move played is that edge in the game tree between the root and its child with the maximum value.

Also, to introduce some non-determinism, we might as well consider choosing a move probabilistically from some good candidate moves. This is to prevent the possibility of the exact same move being played everytime by the computer when fed a particular game position.

**Input/Output Specifications**

The program would largely contain two modules- one is a back-end which is the heart of our AI- given a FEN chess position, it builds the Minimax tree and outputs a move to be played and, if required for analysis/statistics, the Minimax-returned value at the root.

The other is the GUI which accepts the move to be played returned by the engine, exectues that move on the current board position, then takes a move as input from the user (using the mouse), translates the board to yet another position after making any move consistency checks and drawing up statistics if any, and outputs this new board position for the engine to analyze.

**Discussion**

Open for discussion, regarding further features that could possibly be implemented, algorithm optimizations, and fairness of the project load wrt to a 3 member team (as opposed to 2 members).