Hing! The ultimate chess playground on web

# Function Briefing

1. Practice (练习)

Place pieces on the board to form a legal position, to resolve end games,

Ask computer to evaluate the position, to give best move.

1. Gaming (对弈)

Computer to computer

Human to computer

Human to human

1. Game Records (对局记录/棋谱)

Browse and replay personal game records

# Use cases

1. Open the page, default in practice mode, the mode is keep in cookie
2. User can switch mode
3. practice mode, use cases:
   1. Place pieces from the piece box to the board
   2. Set the board in Start position
   3. Set the board by a fen string
   4. Clear the board
   5. Move pieces on the board freely
   6. Ask computer gives best move to a position
   7. Change side
4. gaming mode, use cases:
   1. Select player. There are two players for a game, usually, player1 is the user sit before the screen and player2 is the player behind the screen. The application provides many fictitious players backed by computer chess engines on the server end, user can choose a fictitious player for player2, and available human players are also candidates for player2. If user wants to watch gaming by two fictitious players, user can choose two fictitious players both for player1 and player2.
   2. Set side, timing or other game options
   3. If the user want to play with a human player, there are two ways, look up if somebody is available for play2, or, user can put himself/herself in a status of available for gaming and wait for others.
5. Gaming history, use cases
   1. Browse the past game list, date, player info, etc.
   2. Replay a game, move by move, or auto play
   3. Add comments
   4. Ask computer for best move for a position

# The Architecture

Browser

Board View

Client AI

Node

Application

Redis

Position Cache Database

UCCI.js

UCCI Engines

Mongodb

User Data

Game Records

## BoardView

This is the graphical representation of the chess board. The board view is the epicenter of the UI in our application. The BoardView is implemented in javascript, could be show in all popular browsers and can fit into various device screen size.

## Client AI

Provides simple AI on client side, it is implemented in javascript, running in web worker. So user can play game when they are offline.

## Application

The application will be hosted in nodejs. Client modules will be managed using seajs. The bootstrap will be used as the front-end framework. We adopt responsive web design to support small-screen mobile devices.

## UCCI.js

The Universal Chess Interface (UCI) is an open [communication protocol](http://en.wikipedia.org/wiki/Communication_protocol) that enables a [chess program](http://en.wikipedia.org/wiki/Chess_program)'s engine to communicate with its [user interface](http://en.wikipedia.org/wiki/User_interface). UCCI is the variant of UCI for Chinese Chess. UCCI.js is a nodejs module, which enables nodejs loading and communicating with chess engine. UCI/UCCI is designed for the standalone desktop application which only a single user is gaming. We will make new design on how to communicate between UI and chess engine in a web based chess game. However, it is still valuable to leverage the existing code base by a simple bridging as UCCI.js does.

## UCCI Engines

UCCI engines will be loaded by the UCCI.js. The application supports loading multiple engines, each engine runs in a separated child process where the engine waiting for request of evaluating position from the application.

## Position Cache/Database

In traditional chess program, usually there is an opening book, or an end game table. UCI/UCCI does not specify where the opening shall be. It could be in the chess UI or in the chess engine. In our new design, the opening book and end game table will be hosted by the Redis which is a high performance key-value database. Reasons and benefits are following:

1. High throughput high performance requirements. The system need to handle hundreds of thousands of players gaming online.
2. Why not use simple memory caching system like the memcached? Because we have to change and save the position database. The rank of moves for a position will be adjusted dynamically according to the game result. By this way, the AI of the system will grow and getting more and more powerful as more people playing the game.
3. Move the opening book and end game table out of the chess engine. That will simplify the design of chess engine which could only focus on search and position evaluation.
4. The goal is to unify the opening book and end game table. End game table is more complicated, let’s try to do it.
5. The structure of position database is very simple, for each position, there is a list of moves, and each move has a rank number. We can simply use the fen string as the key of position.

Position

Move1

Move2

Move3

Move4

Move5

## User/Game Database

* User profile, user level, stored fen strings while practicing, etc.
* Personal game records, for replaying
* Public game records, for browsing and learning.

Mongodb is very suitable for host above data.

# TODO list

1. Client AI
2. App framework
3. Position cache/database
4. Port UCCI engines to linux