4-Player Chess Bot:

Stuff I Did

Jacob Brazeal

# Introduction

Chess.com has a new feature: four-player chess. The standard layout is shown in figure 1.

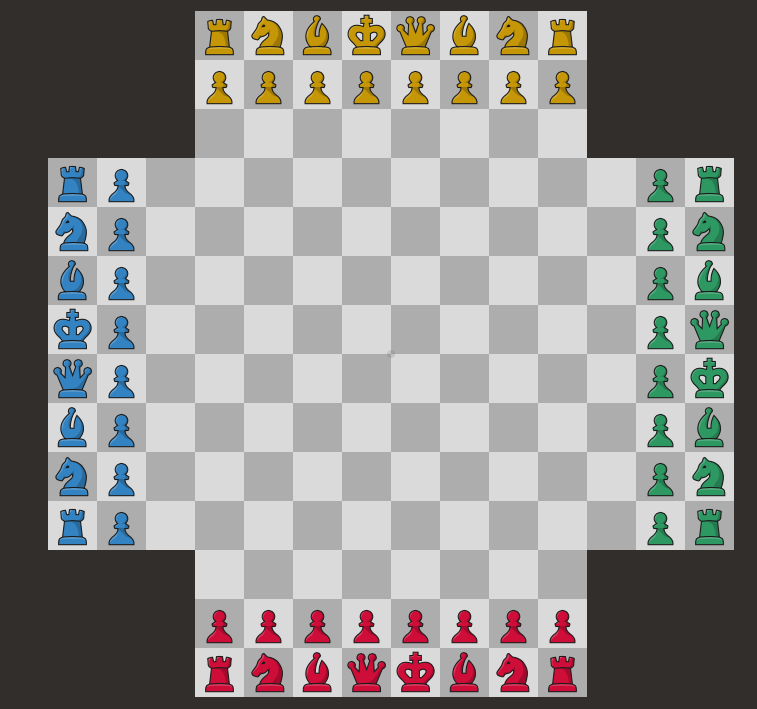


Figure : 4-player chess board

This game appears to originate from the website <http://hellochess.com>, and has broadly similar rules (described below).

## Rules

The following rules are excerpted from <https://chess.com/4-player-chess?t=rules>.

|  |
| --- |
| The goal is to finish with the most points of any player.  Points may be earned in the following ways:   * Checkmating an opponent (+20 points). * Stalemating yourself (+20 points). * Capturing active (colored) pieces: Pawns: +1,  Knights: +3,  Bishops: +5,  Rooks: +5,  Queens: +9,  Kings: +20,  Promoted Queens: +1. * Checking two players simultaneously: +5. * Checking three players simultaneously: +20.   Pawns promote to Queens on your **8th** rank.  Pieces become “dead” (grey) when a player is eliminated. Capturing dead pieces does not award any points.  Gameplay begins with the Red player. Clockwise move order: Red, Blue, Yellow, Green.  The game is finished when three players have been eliminated. |

## Time Controls

Each player has 1 minute plus a 15 second increment, in chess-speak: that is, every turn one has 15 seconds to spend on the move, and then one’s 1 minute reserve begins to run down. If a player runs out of time on his turn, he is out of the game. His pieces remain on the board but have no capture value, except for his king, which is worth 20 points.

## Other 4PC Bots

Hellochess.com apparently has a “very hard AI”, which you can see in this video from 2017: <https://www.youtube.com/watch?v=m0iKVxx95YM>. I have not found any other AIs, and I don’t have code for Hellochess.com’s AI either, so I was starting fresh.

## Ethics

It’s illegal to use computers to play standard chess on Chess.com, for good reason: bots are much better than people, and it’s deceptive to use a bot to gain rating. However, 4 player bots are not specifically prohibited, and since my bot is a worse player than I am, using it was actually a sacrificial act.

# What I Did

I wrote a four player chess bot in both JavaScript and C++. The JS version was 1234 lines of code; the C++ version was 1499 lines (surprisingly terse!). I also wrote a program that let the bot play on the Chess.com website. There was no API for this, so this interface was tricky. It was 200 lines of JavaScript.

I also implemented the Glicko2 rating system and created a genetic algorithm tournament in C++ in order to optimize the arbitrary parameters in my bot. I ran the tournament on a AWS EC2 server (a c5d.2xlarge instance). The tournament code was a total of 592 lines of C++ and Python. I spent $24.19 running the tournament.

The bot performed above average (where an average rating is 1200): its rating varied from 1200-1300 in testing over the week of Monday, April 29, to Thursday, May 2, 2019. A video of one of its games is available on <http://cps.bju.edu/>.

# Strategy

In standard chess, a bot has two components:

1. An evaluation function that can estimate how good a position is for a player by counting material, considering king safety, etc.
2. A minimax engine that considers a tree of possible moves. Usually the tree is constructed by *iterative deepening:* All moves are considered at first, the ones that are obviously bad after a few levels in the tree are discarded, and a deeper search is performed on the remaining candidate moves.

The first chess engine to be super-human was probably Deep Blue, which narrowly defeated then-World Champion Garry Kasparov in 1997. Today, the best chess engines are much better than humans. The best chess engine is generally considered to be Stockfish, an open source program.

In 2017, Google’s DeepMind released the results of a new machine-learning chess program called AlphaZero that (on their own hardware) convincingly defeated Stockfish. Deepmind did not release the code or the network weights, but they did describe AlphaZero in a paper. Since then, the LeelaZero project has attempted to make an open source version of AlphaZero. Training LeelaZero has required volunteers to play millions of training games a day for over a year. In recent results, LeelaZero defeated StockFish in a computer tournament, and is probably the new state of the art.

I chose the simpler, older way: a minimax engine patterned after StockFish.

## Multiplayer minimax

Game trees are more complicated with multiple players. My search tree would widen by a custom number of moves for each player (the player whose turn it was got the most widening), and would search down to a fixed number of plies. My bot searched to depth 10, with widenings (in order of move) 3, 2, 1, and 1.

# Playing on Chess.com

To make my bot play on Chess.com, I used the browser extension Tampermonkey to write a userscript that taps the websocket connection from the website to the server. While it would have been nice to use Chess.com’s built-in APIs to make moves, the local JavaScript did not expose any methods that allowed one to read board state or submit moves, etc. So I manually reviewed the network connection between the browser and the server to deconstruct the protocols. Then, when I want to make a move, I send the corresponding message to the server.

My userscript is asynchronous in two ways:

First, I have to find the right move. Calculating a move is a time-consuming task and initially froze the browser when I turned on the minimax engine. To avoid this, I run a Node.js server with my code in JavaScript and query it with the current board state. If the server takes too long to respond, I simply choose the highest-valuated move with search depth zero, which can be calculated quickly.

Second, I have to send the move to the server. However, the server sometimes rejects legal moves: even if it is one’s turn, the server may enforce some latency before it accepts one’s move. I do not understand what the objective of this behavior is. However, to get around it, I use an exponential backoff, submitting my move after one second, then two seconds, then 4, etc.

I maintain a private version of the state of the board, and update it when Chess.com sends updated moves through the websocket. It’s important to remain in sync with the current player and where the pieces are on the board, or the bot might attempt illegal moves, which the server will reject, causing the bot to time out.

# Evaluation Function

The most critical part of the bot is its evaluation function: how it decides to rank the current state of the board. The easiest evaluation function is to simply add the value of the pieces on the board. However, 4PC introduces several complications.

First, it’s hard to calculate the ‘pressure’ against a piece. For example, a pawn might be mutually attacked by two opposing queens, meaning (ironically) that it’s perfectly safe. In addition, trades are usually bad (as two players lose equally, and two players don’t). I decided to be conservative: I considered all attacks against a piece to be ‘pressure’, but then multiplied that pressure by a constant to decide how concerned I should be. This constant and others were tuned as described below.

Second, checks are a serious problem: If one is checked by the player to one’s right, one could be attacked twice by the other players before being forced to move out of check. I had special constants to determine how aggressively to attack players on the board depending on their orientation.

# The Tuning Tournament

I had 6 parameters from 0 to 1 describing how much value I gave to checking another player, how aggressive to be in attacking, etc. To tune them, I created a genetic algorithm. My algorithm had the following structure:

1. Take *n* different sets of parameters.
2. In each epoch:
   1. Run *k* games of 300 moves each, where each game consists of four randomly selected parameters from the list.
   2. To select the sets of parameters for the next epoch:
      1. Calculate each parameter set’s rating using the Glicko2 rating system (described below).
      2. Keep the highest-rated parameter set from the last 3 epochs.
      3. Add 3 random parameter sets.
      4. For the next *n­ ­*– 6 slots, use the *n* – 6 highest-rated parameter sets from the current epoch with random noise added in the range (-0.15, +0.15) to each parameter.

This procedure had several benefits: The performance of a parameter set could be normed against the performance of the new random parameter sets; several different parameter sets could “evolve” at once; and we could avoid feedback loops. In retrospect, I should perhaps have reduced the “training rate” (that is, the range of random noise) in later epochs.

With 1500 representing a random set of parameters, I was pleased to generate parameter sets performing in the range 1700 – 1800 after five epochs.

## Using AWS

I ran the tournament on AWS EC2, spending $24.19 for about 40 hours of compute time at ~3.6 GHZ. To do this, I created an account on AWS and used the remote terminal to manage my EC2 Linux server. Unfortunately, all files are deleted when you stop running the server, so I downloaded them. I ran the tournament several times because I found bugs in the code.

## Glicko2 Rating System

I used the same rating system as Chess.com to evaluate the parameter sets’ performance. Each game is viewed as 6 games for the purposes of rating: Player 1 vs Player 2, Player 1 vs Player 3, Player 1 vs Player 4, Player 2 vs Player 3, Player 2 vs Player 4, and Player 3 vs Player 4. While chess traditionally uses the ELO system to score rating, Chess.com uses the system of Dr. Mark Glicko, which is better for games in which results can vary wildly. The Glicko2 system also makes special allowances for the ways that players can change in rating over time, but fortunately the ‘true’ rating of a parameter set cannot change over time, which was a helpful simplification. I implemented the Glicko2 system directly from the 2013 source paper: <http://www.glicko.net/glicko/glicko2.pdf>.

# Future Work

I still want to add an opening book, but I did not get to it this semester. In addition, while I fixed some of the glaring strategy problems, my bot did not take adequate advantage of checks. Forming alliances is a big part of multi-player games, and the best way to form them in 4 player chess is to attack players who are in check. Much of my own success at 4 player chess has come from exploiting this psychological element, and I’d like to be able to incorporate it in the bot.

I intend to open source the code on my GitHub account (<http://github.com/polkerty>) and on my website (<http://jacobbrazeal.com>).