Genetic Chess

Mark Harrison

April 30, 2017

Abstract

This work is a program for evolving chess-playing AIs written in C++. Through putting a population of AIs into chess matches, killing off the losers, and breeding the winners, it is hoped that one specimen will be able to stand up against a more traditionally developed engine (if only on the easiest difficulty setting).

Contents

1 Building

Run `make` to create release and debug executables in the `bin/` subfolder. If new files are created or the `#include` files are changed within a file, run `python create_Makefile.py` to regenerate the Makefile.

2 Running

genetic_chess -genepool [file_name] This will start up a gene pool with Genetic_AIs playing against each other–mating, killing, mutating–all that good Darwinian stuff. The required file name parameter will cause the program to load a gene pool and other settings from a configuration file. A record of every genome and game played will be written to text files.

genetic_chess (-human|-genetic|-random) (-human|-genetic|-random) Starts a local game played in the terminal with an ASCII art board. The first parameter is the white player, the second is black.

- **-human:** a human player entering moves on the command line and seeing the results on a text-based drawing of the board. Moves are specified in standard algebraic notation (SAN) or in coordinates that indicate the starting and ending square.
- **-genetic:** a Genetic AI player. If a file name follows, load the genes from that file. If there are several genomes in a file, the file name can be followed by a number to load the genome with that ID. If no number is specified, the last genome in the file is loaded (presumably this one is the most evolved). If there is another file containing records of games played during a gene pool run that has the same name as the genome file name with an extra suffix "_games.txt", then the last genome with at least 3 wins will be selected.
- **-random:** an AI player that chooses moves randomly from all legal moves.

Genetic Chess can communicate with GUI chess programs through the Chess Engine Communication Protocol, including xboard, PyChess, Cute Chess, and others. When using Genetic Chess this way, only specify the arguments for a single player (-genetic or -random). The program will then wait for communication from the GUI.

3 High-Level Overview

4 Non-evolutionary aspects

4.1 Endgame Scoring

Winning gives an infinite score. Losing gives a negative infinite score. Draw gives zero.

Why not evolve these numbers? While the priorities of various genes can be varied to yield different playing styles, the only reasonable score to assign to a win is one that is larger than any other score. It can only be a disadvantage to prefer anything to a winning move. While this would result in upward evolutionary pressure on the score assigned to winning, it would stall the evolution of all other genes while the score assigned to winning was pushed high enough to always be preferred.

4.2 Minimaxing

Also mention the decision process for whether or not to recurse further down a certain play variation.

Since chess is a zero-sum game, the score assigned to one side of the board is equal to the negative value assigned from the other side. The variation of Minimax is known as Negamax.

4.3 Alpha-Beta Pruning

With additional optimization to cutoff search if alpha represents a win at a shallower depth than is possible in the current branch.

5 The Genome

5.1 Regulatory Genes

5.1.1 Piece Strength Gene

Specifies the importance or strength of each different type of chess piece.

5.1.2 Look Ahead Gene

Determines how many positions to examine based on the time left. When looking ahead to future moves, the number of positions to examine is divided equally among every legal move. This naturally limits the depth of search while allowing deeper searches for positions with fewer legal moves. The amount of time to use in examining moves is determined by genetic factors indicating an average number of moves per game and the number of positions than can be examined per second. The distribution of moves per game is modeled with a Poisson distribution.

5.1.3 Branch Pruning Gene

Cutoff searching to greater depths if the current move lowers the score of a board state to less than the amount specified within the gene.

5.2 Board-Scoring Genes

5.2.1 Total Force Gene

Sums the strength (according to the Piece Strength Gene) of all the player's pieces on the board.

5.2.2 Freedom to Move Gene

Counts the number of legal moves available in the current position.

5.2.3 Pawn Advancement Gene

Measures the progress of all pawns towards the opposite side of the board.

5.2.4 Opponent Pieces Targeted Gene

Sums the total strength (as determined by the Piece Strength Gene below) of the opponent's pieces currently under attack.

5.2.5 Sphere of Influence Gene

Counts the number of squares attacked by all pieces. Bonus points are awarded if the square can be attacked with a legal move.

5.2.6 King Confinement Gene

Counts the squares the king can reach given unlimited legal moves.

5.2.7 King Protection Gene

Counts the squares that have access to the king by any valid piece movement that are unguarded by that king's other pieces.

5.2.8 Castling Possible Gene

Returns a positive non-zero score to indicate that castling is possible or has already happened. Score can vary based on the preference for kingside or queenside castling.

6 The Gene Pool: On the care and feeding of chess AIs

In each generation, the players in a gene pool are randomly matched up with each other to play a single game of chess. If the game ends with a winner, whether through checkmate or time violation, then the two players produce an offspring by recombination of their genomes (each of the offspring's genes is picked randomly from either parent with equal probability) plus a single extra mutation. This offspring replaces the loser in the gene pool. This way, the genes of losing players are only slowly weeded out since a single game does not provide much information about the fitness of each gene.

If the game ends in a draw, then one of two things happens. With high probability–currently 95%–the players are left as they are and will participate in the next round. The other possibility is that one of the players is randomly picked to be replaced by the result of mating the other player with either a randomly generated Genetic_AI, or any past Genetic_AI, even long dead ones. This happens with low probability because it destroys genetic information. However, it is necessary to keep a pool from stagnating due to never-ending drawn games. Bringing in a randomly generated AI injects new genetic information into the pool (though with low probability of high-quality information). Bringing back dead Genetic_AIs injects good information from the past and should help to keep a gene pool from becoming trapped in a self-reinforcing, aberrant playing style that only works against similar players. In effect, both of these strategies are meant to kick a gene pool off of a local maximum of genetic fitness.

One final means of preventing gene pool stagnation and preserving genetic diversity is the use of multiple gene pools. Each gene pool evolves separately for a long time, allowing each to genetically diverge. Then, every once in a long while (the time being user-specified), the best player from each pool is transferred to the next pool over. Thus, the best genes are further spread afield so that they can be tested against a wide range of opponents. A useful measure of a pool's strength is how long its best player survives when it enters a new pool.

An example of typical output during a gene pool run is shown below:

```
Gene pool ID: 0  Gene pool size: 16  New blood introduced: 126 (*)
Games: 22024  White wins: 10469  Black wins: 9562  Draws: 2737
Time: 38.538 sec   Gene pool file name: pool.txt
    ID   Wins  Streak  Draws  Streak
 59014      9       7      1       0 T
```

```
 59031      3       3       0       0
 59054      1       0       2       2
 59055      2       2       0       0
 59074      1       1       0       0
 59077      1       1       0       0
 59078      1       1       0       0
 59081      1       1       0       0
 59095      0       0       0       0
 59096      0       0       0       0
 59097      0       0       0       0
 59098      0       0       0       0
 59099      0       0       0       0
 59100      0       0       0       0 *
 59101      0       0       0       0
 59102      0       0       0       0

59055 vs 59097: White!
59074 vs 59101: Black!
59102 vs 59100: White!
59096 vs 59014: None! 59014 mates with random / 59096 dies
59031 vs 59078: None!
59054 vs 59095: Black!
59098 vs 59099: Black!
59081 vs 59077: White!

Most wins:     18 by ID 20968
Longest lived: 27 by ID 45394
```

The `Streak` column indicates the current number of consecutive wins or draws a player has attained in the last few games. In the example above, player 59014 is on a seven-game winning streak following its last draw. Player 59054 has drawn its last two games.

The asterisk (`*`) indicates an offspring of the result of a drawn match. Note that the outcome of the game between 59096 and 59014 means that another such offspring will be brought into this pool for the next round. The (`T`) indicates that it is the best AI from another gene pool that has been copied to this pool.

7 Some Consistent Results

Here are a few results that are reliably reproduced in multiple simulations.

Piece values are rated in near-standard order

In descending order of valuation by a Genetic_AI:

1. Queen
2. Rook
3. Bishop and Knight nearly equal
4. Pawn

White has a slight advantage

Of the games ending in checkmate, white wins about 10% more often than black. Wins by time are shared by black and white equally.

The Total Force gene and the Pawn Advancement gene typically dominate.

The Pawn Advancement gene usually gains higher priority first, probably because it is the simplest gene that makes an immediate difference in the game. Push the pawns forward both threatens the opponent's pieces with low-risk attacks and increases the chances of promotion.

The Queen is the most popular piece for promotion.

Even when the Piece Strength gene has not been tuned at all, the queen is the overwhelming favorite, followed by the rook, then bishop, and finally the knight. In human games, only the queen and knight are chosen since they have different move patterns. If you need at least a rook or bishop, you might as well take a queen since that piece provides both. Only the knight provides a viable alternative.

Threefold repetition is the most common stalemate

Just like real games.

The Sphere of Influence gene typically counts legal moves as just as valuable as any other move.

This was unexpected. I thought that the legal moves would count more since they present a greater threat to the opponent. You cannot capture your opponent's Queen if your own King is in check. Perhaps Genetic_AIs find this gene more useful as a forward-looking view of the game.

8 Public Methods of Classes - Programmer's API

8.1 Piece

8.2 Move

8.3 Complete_Move

8.4 Board

8.4.1 Important Notes

The Complete_Moves returned by Board::get_complete_move() and Board::all_legal_moves(), as well as the const Piece* returned by Board::view_piece_on_square(), should be handled carefully as these structures are rended invalid by the following two operations:

1. The originating Board is modified via Board::submit_move().
2. The originating Board is destructed.

Furthermore, a Complete_Move generated by one Board is invalid for any other Board, even an unmodified copy of the originating Board.

When calling Board::view_piece_on_square(), an empty square is represented by a nullptr.

8.5 Player

8.6 Clock