# Genetic Chess

Mark Harrison

April 21, 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** **2**

**2 Running** **2**

**3 High-Level Overview** **3**

**4 Non-evolutionary aspects** **3**
- 4.1 Endgame Scoring . . . 3
- 4.2 Minimaxing . . . 3
- 4.3 Alpha-Beta Pruning . . . 3

**5 The Genome** **3**
- 5.1 Regulatory Genes . . . 3
  - 5.1.1 Piece Strength Gene . . . 3
  - 5.1.2 Look Ahead Gene . . . 3
  - 5.1.3 Branch Pruning Gene . . . 3
- 5.2 Board-Scoring Genes . . . 4
  - 5.2.1 Total Force Gene . . . 4
  - 5.2.2 Freedom to Move Gene . . . 4
  - 5.2.3 Pawn Advancement Gene . . . 4
  - 5.2.4 Opponent Pieces Targeted Gene . . . 4
  - 5.2.5 Sphere of Influence Gene . . . 4
  - 5.2.6 King Confinement Gene . . . 4
  - 5.2.7 King Protection Gene . . . 4
  - 5.2.8 Castling Possible Gene . . . 4

**6 The Gene Pool: On the care and feeding of chess AIs** **4**

**7 Public Methods of Classes - Programmer's API** **6**
7.1 Piece . . . . . 6
7.2 Move . . . . . 6
7.3 Complete_Move . . . . . 6
7.4 Board . . . . . 6
7.5 Player . . . . . 6
7.6 Clock . . . . . 6

# 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. Moves are specified in algebraic notation indicated 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.
- **-random** an AI player that chooses moves randomly from all legal moves.

A barely functional implementation of the Chess Engine Communication Protocol allows for play through xboard and similar programs (PyChess, etc.). When used this way, arguments are ignored. Future feature: specify a specific AI to use like -genetic or -random.

# 3 High-Level Overview

# 4 Non-evolutionary aspects

## 4.1 Endgame Scoring

Winning gives an infinite score 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 playstyles, 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

## 4.2 Minimaxing

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

## 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

Multiple pools.

What happens to winners, losers, and games ending in a draw.

Typical output during a gene pool run:

```
Gene pool ID: 0  Gene pool size: 16  New blood introduced: 126 (*)
Games: 22026  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       9      0       0 T
 59031      3       3      0       0
 59054      2       2      0       0
 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

Result of 59055 vs. 59097: White! mating 59055 59097 / killing 59097
Result of 59074 vs. 59101: Black! mating 59074 59101 / killing 59074
Result of 59102 vs. 59100: White! mating 59102 59100 / killing 59100
Result of 59096 vs. 59014: None! 59014 mates with random / 59096 dies
Result of 59031 vs. 59078: None! 59031 mates with 4653 / killing 59078
Result of 59054 vs. 59095: Black! mating 59054 59095 / killing 59054
Result of 59098 vs. 59099: Black! mating 59098 59099 / killing 59098
Result of 59081 vs. 59077: White! mating 59081 59077 / killing 59077

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

The asterisk indicates an offspring of the result of a drawn match, in which one of the players in that match is replaced by the result of mating the other player with a randomly generated Genetic_AI. This happens with low probability because it destroys genetic information. However, it is necessary to keep a pool from stagnating due to neverending drawn games.

The `T` indicates that it is the best AI from another gene pool. One measure of a pool's strength is how fast its best gets killed off when it enters a new pool.

# 7 Public Methods of Classes - Programmer's API

## 7.1 Piece

## 7.2 Move

## 7.3 Complete_Move

## 7.4 Board

**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`.

## 7.5 Player

## 7.6 Clock