Project Light Blue

Phillip Huang: phillphuang@college.harvard.edu
Christopher Chen: cchen02@college.harvard.edu
Javier Cuan-Martinez: cuanmartinez@college.harvard.edu
Collin Styring: collinstyring@college.harvard.edu

Brief Overview

We seek to develop a move-recommender for Chess. A user can input each move in a game of Chess using the standardized algebraic chess notation, and the program will return a recommended move for the given configuration of the board. To implement this, we plan to develop a collaborative filtering and machine learning engine. We will feed the engine data from previously recorded games of chess, likely from records of past world championships. The engine will process this data, and, based on the previous moves in past games, devise the most advantageous move to make for the turn. We will implement this application in Python, a language we are currently learning.

Feature List

Core Features

- Collaborative filtering engine
  - The engine will determine the “best” option out of a list of possible options by analyzing past behavior recorded as chess move data. Actual technical implementation remains ambiguous at this point, but will likely resolve to a system that determines the best move for any given board configuration.
- Insertion of data from past games
  - Chess game data will be obtained from various online databases. The application will parse the information from the standardized Portable Game Notation (PGN) files used by the industry for recording chess moves and games. The parser will then feed each move into the filtering engine.
- Basic command line interface

Cool Extensions

- Better recommendations
  - Consider weighting different moves based on who won or lost the match, ELO rating, or other factors
- Saving and resuming games
  - Save data will be a PGN file
- Undo input
  - Allow user to retract a move input and revert to the previous state.
- Graphical user interface
  - Simulate the board using a GUI with images of each piece in the current configuration of the board. Recommendations would be given in the form of arrows point from the current location of the piece to the recommended move location.

Technical Specification

THE FILTER
We have yet to solidify our concept of the algorithm used for collaborative filtering. We've looked into correlation-based algorithms (relatively simple), and item- and user-based algorithms (far more complex). Our decision will likely alter the spec here substantially. Notably, we could implement the correlation algorithm first, then tackle more difficult algorithms given ample extra time. This is a natural place for modules and the ability to switch out different algorithms to provide better data.

The module interface for the filter will likely consist of:
- "Chess move" abstract type - will likely be implemented as strings so it can be easily parsed and displayed to the user. Will use the standard PGN format.
- Recommend - outputs the recommendation for the current game configuration
- Move - takes in a Chess move, and records it, changing the simulation of the board
- Clear - resets the board simulation
- Clean - erases all filter information and history, for debugging purposes

To import data to the module, we will simply feed in all the moves in the previous game. The module should then save the moves and automatically adapt itself with each new game played. Thus, the system learns from each use.

The filter itself will rely on two main sub-systems, each a module:

FILTER SUBSYSTEM: BOARD SIMULATION
Tracks the current configuration of the board. Likely going to be implemented as a matrix. This will implement the move and clear functions from the filter. This will contain a function that parses and interprets Chess moves to change the respective piece in the matrix.

FILTER SUBSYSTEM: THE KNOWLEDGE BASE
The accumulated data from all imported games. This module contains the data structure representing the "memory" of the collaborative filter. It will produce the recommended move by comparing current board configuration with the configurations stored in the memory, and retrieving the most preferred move for that configuration.

We will likely implement this as a graph. Each node will represent a configuration of the board. Each edge will represent a possible move. Each edge will be weighted, determining the superiority of that move over the other possible moves. If a node does not exist for a particular configuration, we will have to prompt the user to make an arbitrary move and continue the game until the system recognizes another configuration, which will prompt recommendations to resume. If a node does not exist for a given configuration, it is created to allow the system to learn.

Weighting of each edge will first be determined entirely by popularity. Edges used often will be deemed advantageous. "Popular" edges will be tracked with an integer for each edge. The highest integer will be the best move for a given configuration. We can expand this implementation later with ELO ratings, whether the move was made by a winner or loser (for completed games), or other forms of weighting.

PGN PARSING ENGINE
To avoid manually inputting hundreds of thousands of past games, we will develop an engine that parses and inputs PGN files. This PGN parser could also help us implement saving and resuming games.

IMPORT ENGINE
After feeding a PGN file to the parsing engine, the import module will feed the list of moves generated into the knowledge base to allow the system to learn.

COMMAND LINE INTERFACE
Allow the user to use all the functionality we plan to implement.

Next Steps

- Set up Github
- Set up Asana/Slack/Trello/Bitrix24
- Set up Python environment for OS X and Windows
- Learn Python
- Research basic collaborative filtering algorithms
- Research the graph data structure
- Research PGN files
- Research command line inputs