Mathematical Problems of Computer Science 46, 107--116, 2016.

Development of a Modular Tool for Regulating and Analyzing Activities in Chess

Levon S. Berberyan

Institute for Informatics and Automation Problems of NASRA
e-mail: levon.berberyan@gmail.com

Abstract

In this work computer software tool for regulating and analyzing activities in chess and approach to its design are suggested. The article starts with analyzing some common software for regulating and analyzing activities in chess, listing some of their pros and cons, especially concerning the chess engines comparison. Further, an improved software design approach to regulating and analyzing activities in chess based on separating software modules and defining API for each of them is specified. Then the design approach and its implementation details, particularly components implementing modules API are independent from each other, providing flexible mechanisms for manipulations are described. Also usage scenarios building tips, based on manipulations with basic commands calls, for developed software are provided.

Keywords: Chess, Chess Engines, Chess Engines comparison, Chess Software, Activities in Chess, Regulating Activities in Chess, Analyzing Activities in Chess, Flexible Chess Software, Chess Tutors.

1. Introduction

1.1. The standard approach to designing tool for regulating and analyzing activities in chess includes chess software, which consists of the following components:

a. Board component - shows the chess board to a user in some predefined way, visualizes chess pieces positioning.

b. Engines component - allows to run some chess engine as a player.

c. Game component - allows to run chess games between the players, supports remote gaming.

d. Position component - allows to load chess game states or to store them.

e. Game Logging component - allows to log the whole game.

f. Analyzer component - allows to analyze some game state or the whole game.

User can open board UI, edit chess pieces positioning, select players, can use some engine from the available listing as players, setup and run a game, then analyze positioning using some available component or analyze the whole game.

One of the most popular software solutions that implements the standard approach is Arena Chess GUI [1].

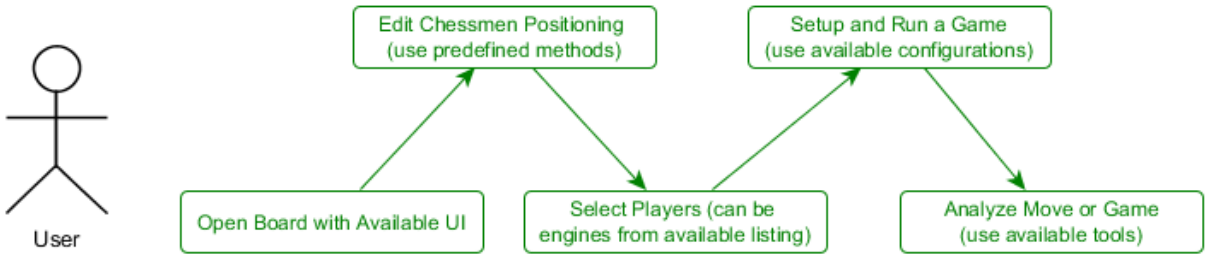

Fig. 1. Chess activities regulating and analyzing.

1.2. Questioning standard approaches to designing software tool for regulating and analyzing activities in chess.

The mentioned approach to designing software tool for regulating and analyzing activities in chess:

a) Does not contain functionality for working with custom chess engines (example: web-based chess engines).
b) Does not provide abilities to build custom usage scenarios (custom tournaments, custom game assessment methodology, tutoring).
c) Does not allow adding components or functionality without changing existing application structure.

Problems with overcoming constraints in existing stable software tools are the following:

a. Not flexible application architecture.
b. Closed source code for most solutions.
c. Paid license for many solutions.

2. An Improved Approach to Designing Tool for Regulating and Analyzing Activities in Chess

We developed an approach to designing tool that provides extended possibilities to regulate and analyze activities in chess and implemented software. In order to ensure flexibility of the software, modular design is used in its implementation. Modules are separated from each other. At the same time modules can call each other.

Every module has its own API. API is divided into a number of parts. Thus, we can modify modules or add new ones separately. API covers module's functionality and represents the mechanism for using it.

Every module has a number of components that implement API in an individual way. With this design style we can just add some components that implement API in their own manner to extend the module functionality.

So, we have more flexible tool than the popular ones that can be modified on both the backend structure and use cases.

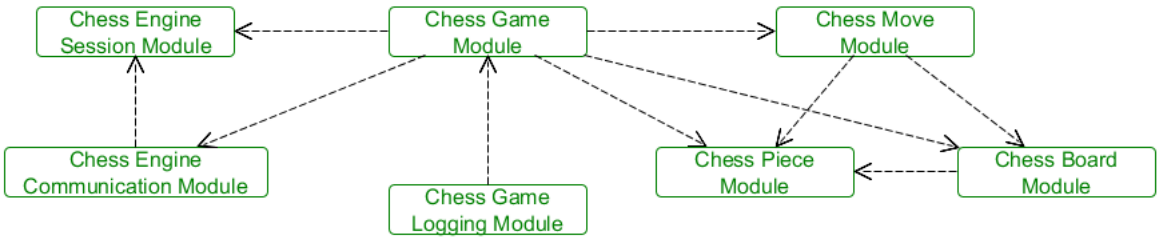

Fig. 2. Modular Tool for Regulating Activities in Chess.

Features of developed tool structure:

1. Every chess game entity has its own module (chess piece, board, move, logging, etc.), which are in relationships of type Dependency ("using" relationship).
2. Every module has one or more interfaces that represent its API.
3. Every module has a number of components that implement API in an individual ways. It also has mechanism for switching between implementations.

Listed features allow to add modules without changing existing application structure, to add new components without changing module structure, to design custom usage scenarios. These advantages allow to design custom chess scenarios like assessment tasks [2] or chess tutors [3].

3. Implementation Details of Regulating and Analyzing Chess Activities Tool

A software tool was developed to implement an improved approach with modular design [4]. Implementation was focused on tools abilities for working with chess engines. Tool contains components for running chess engines, communicating with them, representing chess pieces, chess boards, chess moves and chess games, also providing chess game logging. These components allows to regulate chess engines activities (compare engines, learn from engine) and analyze them (get info from engines, log engines activity).

Software has been tested using Ad-hoc Testing approach (some unit tests were developed for testing modules functionality on examples and some integration tests for testing modules interaction). Software was runed in Ubuntu, Windows, Mac OS platforms.

3.1. Chess Engine Session Module

This module allows running new chess engine session. Chess engines implementations can be started as native OS process, web get service or other ways. In current version starting as an OS process and web get service are implemented as most of chess engines today support these ways, the other ways can be added manually or in the next versions of product.

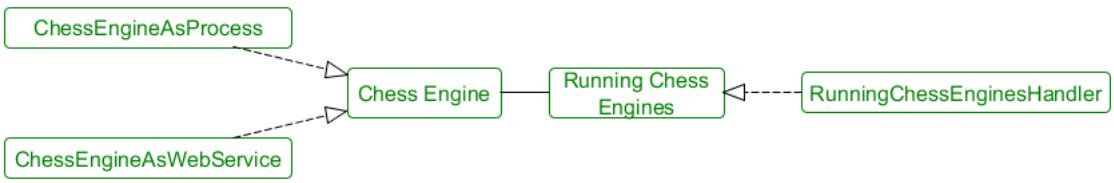

Fig. 3. Chess Engine Session Module.

Engines running implementations:

- to start native OS programs and connect to their input and output streams
- to connect to web API through HTTP request with GET method

Chess Engine Session Module consists of 2 parts:

1. Chess Engine (working with particular chess engine)
2. Running Chess Engines (working with pool of chess engines)

Chess Engine Session Module API for part 1 consists of the following methods:

- createNewEngineSession(path) - creates a new chess engine session by its path.
- writeLineToEngine(sessionId, theLine) - writes line(request) to engine with defined session id.
- readLineFromEngine(sessionId) - reads line(response) from engine with defined session id.
- destroyEngineSession(sessionId) - closes(quit) the session with defined session id.

Chess Engine Session Module API for part 2:

- getChessEngine(sessionId) - returns reference to one of running chess engines by its session id.

Components of part 1:

- ChessEngineAsProcess - implementation of running engine as native OS application. Process has its input and output streams (data stream) available for reading and writing.
  createNewEngineSession(path) - creates a new chess engine session running new process by the file located on specified path.
  writeLineToEngine(sessionId, theLine) - writes string line to input stream of engine using session id.
  readLineFromEngine(sessionId) - reads string line from output stream of engine using session id.
  destroyEngineSession(sessionId) - destroys the process of engine using session id.
- ChessEngineAsWebGetResource - implementation of connec to chess engine through Http.
  url - url for requests, that can be set and modified.
  response - string that contains response string received after request.
  createNewEngineSession(path) - creates a new chess engine session by its path, that is url.
  writeLineToEngine(sessionId, theLine) - makes an HTTP GET request to specified url and writes theLine.
  readLineFromEngine(sessionId) - reads response string received from engine after last request using session id.
  destroyEngineSession(sessionId) - closes the engine session using specified session id.

Components of part 2:

- RunningChessEnginesHandler - handling the pool of runned chess engines sessions.
  getChessEngine(sessionId) - returns reference to one of running chess engines by its session id.

3.2. Chess Engine Communication Module

This module allows to communicate with running chess engines. For communication with chess engines specific protocols are used. In current version of the tool UCI [5] protocol is partly supported as one of the most used, the others can be added manually or in the next versions of software.

Chess Engine Communication Module calls Chess Engine Session Module to refer to the particular engine, adds additional functionality.

Fig 4. Chess Engine Communication Module.

Chess Engine Communication Module API:

- setProtocol(sessionId) - sends command to set protocol as UCI to the engine with specified session id.
- setNewGame(sessionId) - sends command to start a new game to engine with specified session id.
- checkIsReady(sessionId) - sends command to check if engine is ready to engine with specified session id for requests.
- getMoveCalculations(sessionId) - sends command to calculate move to engine with specified session id.
- setPositionFen(sessionId, fen) - sends command to set game state using board FEN positioning to engine with specified session id.
- quit(sessionId) - sends command to end game to engine with specified session id, this command also stops the related session.

Component:

- UCIChessEngineCommunicationHandler - handles commands that are based on UCI protocol.

3.3. Chess Piece Module

This module allows to represent chess pieces, converts one representation form to another one.

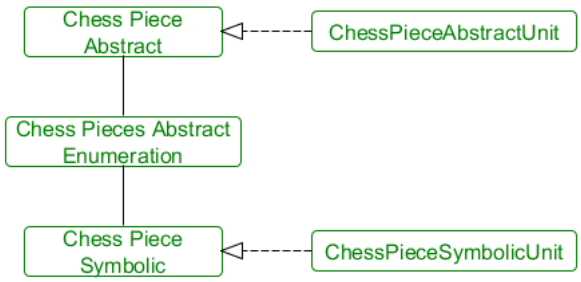

Fig 5. Chess Piece Module.

Chess Piece can be represented in different ways. In current version 2 common ways are supported, that are presented by 2 parts of module:

1. Abstract Chess Piece (example: King);
2. Symbolic Chess Piece representation (example: K).

Chess Piece Module API for part 1:

- ConvertSymbolicChessPieceToAbstract(theChessPieceSymbolic) - converts representation of chess piece from symbolic to abstract.

Chess Piece Module API for part 2:

- GetXFromChessPieceSymbolic(letter) - gets the X coordinate from symbolic chess piece first letter.
- getYFromChessPieceSymbolic(theOrderLetter) - gets the Y coordinate from symbolic chess piece second letter.
- convertAbstractChessPieceToSymbolic(ChessPieceAbstracttheChessPiece) - converts chess piece representation from abstract to symbolic.

Component of part 1:

- ChessPieceAbstractUnit - allows to handle abstract chess piece representation.

Component of part 2:

- ChessPieceSymbolicUnit - allows to handle symbolic chess piece representation.

3.4. Chess Board Module

This module allows to represent chess board. Chess Board can be represented graphically and non-graphically. There are some GUI implementations like the concept in XBoard [6].

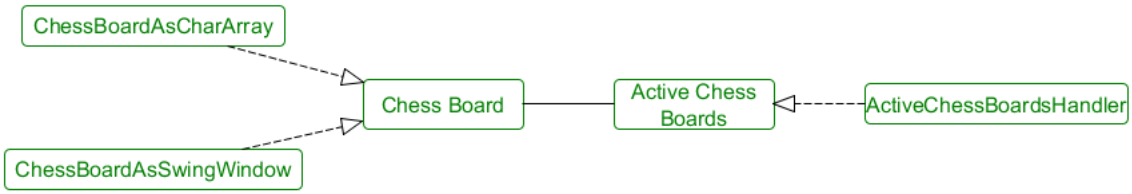

Fig 6. Chess Board Module.

In current version the next popular representation methods are supported:

1. Symbolic Array;
2. Swing Window.

Chess Board Module consists of 2 parts:

1. Chess Board
2. Active Chess Boards

Chess Board Module API for part 1:

- create() - creates new chess board and activates it.
- getBoardMatrix() - gets matrix of chess piece with their positioning on board.
- setBoardMatrix(theBoardMatrix) - sets the positioning of chess piece on board based on chess piece matrix.
- initialize() - sets the board default state.
- getChessPieceOnXY(y, x) - gets the chess piece located on (y;x) coordinate.

- setChessPieceOnXY(y, x, theChessPiece) - sets the chess piece located on (y;x) coordinate to theChessPiece.
- show() - shows the board state.
- applySymbolicMoveToBoard(theSymbolicMove) - applies symbolic move to board.

Chess Board Module API for part 2:

- getActiveChessBoard(theActiveChessBoardId) - returns active chess board y its id.

3.5. Chess Move Module

This module allows to define chess moves. Chess moves are entities related to chess board: Chess Move -> Chess Board.

Fig 7. Chess Move Module.

Chess Move Module has 2 parts:

1. Chess Move (represents chess moves)
2. Chess Move Validator (provides mechanism for validating chess moves)

Chess Move Module API for part 1:

- uciMoveToSymbolicMove(theUciMove) - converts the move received using UCI to symbolic representation.
- symbolicMoveToAbstractMove(theSymbolicMove) - converts the symbolic representation of chess move to abstract one.

Chess Move Module API for part 2:

- isValid(theMove) - checks if the move is valid.
- checkEndOfGame(theChessGameState) - checks theend of game for specified chess game state.

3.6. Chess Game Module

This module allows to handle chess games. Chess Game is related to Chess Board, Chess Piece, Chess Move, Chess Player, Chess Communication. Chess game is a sequence of game states.

Fig. 8. Chess Game Module.

Chess game can have different properties:

- simple attack
- move with no effect
- castling possibility
- etc.

Chess Game Module API:

- create() - creates new chess game
- initialize() - initializes new chess game.
- run() - runs chess game with specified options.
- checkEffect() - checks the effect on game like castling effect, etc.
- registerMove() - registers move for particular chess game.

3.7. Chess Game Logging Module

This module allows to handle chess games logging. Logging can be done for game state or for the whole game.

Fig 9. Chess Game Logging Module.

There are different approaches for description of chess board state. Currently software supports FEN [7]. The logging of the whole game can be done in PGN format [8].

Chess Game Logger Module API:

- convertToLoggingFormat() - converts chess data to logging format.
- convertFromLoggingFormat() - converts logging data to chess data.

4. Usage Scenarios

The developed tool usage scenarios are based on calling modules API functions related to particular components. Calling modules API functions can be organized in any sequence of basic commands that provides flexible use cases mechanism.

Basic commands for UI:

- modules - shows available modules.
- module [module name] - shows API for module.
- module [module name] -component [component name] -function [module API function name] -options [list of options values separated by coma]: calls the function from particular module API by component name that implements it with specified options values.
- quit - quits the application.

Example (Run chess engine as Process from file with location "./engine" and make a new session for it): module ChessEngineSessionModule -component ChessEngineAsProcess -function createNewEngineSession -options "./engine"

5. Conclusion

1. Software is developed for analyzing and regulating activities in chess. The software and its design provide users with the following advantages:

   a. Possibility to add modules to software without violating the existing ones.

   b. Possibility to add components to modules without violating the module structure.

c. Possibility to make custom scenarios for defining new software functionality call paths.

2. The software is validated by running under popular desktop pc platforms (Windows 7, Ubuntu, Mac OS).

3. Developed approach and software can be used for.

- Chess engines comparison through organizing games between them.
- Chess engines communicating tasks.
- Chess engines assessment.

In future specified software modules and their components can be improved, also new ones can be added to provide more developed tool for particular usage, for example, tutoring.

Developed software can be used particularly for new chess engine assessment, for example, the Solver [9] designed under the guidance of Edward Pogossian.

Acknowledgements

Author expresses gratitude to Edward Pogossian and Sedrak Grigoryan for counseling this research, also to Emma Danielyan for helping to improve the article.

References

[1] Arena Chess GUI website. [Online]. Available: https://www.playwitharena.com

[2] E. Pogossian, "On assessment of performance of systems by combining on-the-job and expert attributes scales", *Proceedings of International Conference CSIT 2015*, Yerevan, Armenia, pp. 331—334, 2015.

[3] S. V. Grigoryan and L. S. Berberyan, "Developing interactive personalized tutors in chess", *Transactions of the IIAP NAS of RA, Mathematical Problems of Computer Science*, vol. 44, pp. 116-132, 2015.

[4] E. Gamma, R. Helm, R. Johnson and J. Vlissides, Design *Patterns, Software Engineering, Object-Oriented Programming*, Addison-Wesley, 1994.

[5] R. Huber and S. Meyer-Kahlen, UCI (universal chess interface), *CCC*, November 28, 2000.

[6] XBoard website. [Online]. Available: https://www.gnu.org/software/xboard/

[7] S. J. Edwards, "Forsyth-Edwards Notation", *Portable Game Notation Specification and Implementation Guide*, 03.12.1994.

[8] S. J. Edwards, "Portable Game Notation", *Portable Game Notation Specification and Implementation Guide*, 03.12.1994.

[9] K. Khachatryan and S. Grigoryan, "Java programs for matching situations to the meanings of SSRGT games", *Proceedings of SEUA Annual Conference*, Yerevan, Armenia, pp. 135-141, 2013.

Submitted 01.08.2016, accepted 15.11.2016.

Շախմատային գործունեության վերահսկման և վերլուծման մոդուլային գործիքի կառուցում

Լ. Բերբերյան

Ամփոփում

Տվյալ աշխատանքում առաջարկվում են ծրագրային ապահովում շախմատային տարբեր տեսակի գործունեության վերահսկման և վերլուծման ապահովման համար և իր կառուցվածքի մոտեցում: Աշխատանքը սկսվում է տարածված շախմատային գործունեության վերահասկման և վերլուծման ծրագրային ապահովման քննարկմամբ՝ նշելով դրանց առավելություններն ու թերությունները, հատկապես շախմատային շարժիչների համեմատության վերաբերյալ: Ապա ներկայացվում է բարելավված ծրագրային ապահովման կառուցման մոտեցում շախմատային տարբեր տեսակի գործունեության վերահսկման և վերլուծման ապահովման համար հիմնված ծրագրային մոդուլների առանձնացման և նրանցից յուրաքանչյուրի համար API սահմանելու վրա: Այնուհետև նկարագրվում է առաջարկվող մոտեցման իրականացումը, մասնավորապես բաղադրիչները իրականացնող մոդուլների API-ն, մասնավորապես մոդուլների API-ն իրականացնող բաղադրիչները նույնպես անկախ են իրարից, որն ապահովում է ճկուն ձևախման մեխանիզմներ: Տրամադրվում են նաև մշակված ծրագրային ապահովման օգտագործման սկզբունքներ՝ հիմնված բազային հրամանների կանչերի մանիպուլյացիաների վրա:

Разработка модульного инструмента регулирования и анализа активностей в шахматах

Л. Берберян

Аннотация

В данной работе предлагаются программное обеспечение для регулирования и анализа различных видов активности в шахматах и подход к его конструированию. Статья начинается с анализа распространенных программ для регулирования и анализа различных видов активности в шахматах, перечисляя некоторые их достоинства и недостатки, в особенности, касающиеся сравнения шахматных движков. Далее определяется улучшенный подход к конструированию программного обеспечения для регулирования и анализа различных видов активности в шахматах, основанный на разделении модулей программного обеспечения и определении API для каждого из них. Затем описывается реализация предложенного подхода, в частности компоненты, реализующие API модулей, независимы друг от друга, что обеспечивает гибкие механизмы для различных видов манипуляций. Также приводятся сведения для использования разработанного программного обеспечения, основанные на манипуляции вызовов базовых команд.