Module Interface Specification for Chess Connect

Author Name

January 18, 2023

1 Revision History

Date	Version	Notes
Date 1	1.0	Notes
Date 2	1.1	Notes

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at [give url —SS]

[Also add any additional symbols, abbreviations or acronyms —SS]

Contents

3 Introduction

The following document details the Module Interface Specifications for [Fill in your project name and description —SS]

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at .... [provide the url for your repo —SS]

4 Notation

[You should describe your notation. You can use what is below as a starting point. —SS]

The structure of the MIS for modules comes from Hoffman and Strooper (1995), with the addition that template modules have been adapted from Ghezzi et al. (2003). The mathematical notation comes from Chapter 3 of Hoffman and Strooper (1995). For instance, the symbol := is used for a multiple assignment statement and conditional rules follow the form $(c_1 \Rightarrow r_1|c_2 \Rightarrow r_2|...|c_n \Rightarrow r_n)$.

The following table summarizes the primitive data types used by Chess Connect.

Data Type	Notation	Description
character	char	a single symbol or digit
integer	$\mathbb{Z}$	a number without a fractional component in (-∞, ∞)
natural number	$\mathbb{N}$	a number without a fractional component in [1, ∞)
real	$\mathbb{R}$	any number in (-∞, ∞)

The specification of Chess Connect uses some derived data types: sequences, strings, and tuples. Sequences are lists filled with elements of the same data type. Strings are sequences of characters. Tuples contain a list of values, potentially of different types. In addition, Chess Connect uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification.

5 Module Decomposition

The following table is taken directly from the Module Guide document for this project.

Level 1	**Level 2**
Hardware-Hiding	
Behaviour-Hiding	Input Parameters Output Format Output Verification Temperature ODEs Energy Equations Control Module Specification Parameters Module
Software Decision	Sequence Data Structure ODE Solver Plotting

Table 1: Module Hierarchy

6 MIS of Web Application Input Module

6.1 Module

Web Application Input

6.2 Uses

Board Module
User Mode Module

6.3 Syntax

6.3.1 Exported Constants

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
parseInput	string	seq of string	invalidInput

6.4 Semantics

6.4.1 State Variables

inputString: string #String containing FEN string, user mode, game termination state, and delimiting characters

6.4.2 Environment Variables

N/A

6.4.3 Assumptions

N/A

6.4.4 Access Routine Semantics

parseInput():

- output: sequence of strings. The first is the FEN string, the second is the user mode, the third is the game termination state.
- exception: invalidInput if any of validFen, validUserMode, or validGameTermination return false.

6.4.5 Local Functions

Name	In	Out	Exceptions
validFen	string	boolean	
validUserMode	string	boolean	
validGameTermination	string	boolean	

7 MIS of Display Module

7.1 Module

Display

7.2 Uses

Board Module

7.3 Syntax

7.3.1 Exported Constants

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
drawSquare	string		
drawBoard	seq of (seq of int)		
displayGameTermination	int		
setBackground	string		

7.4 Semantics

7.4.1 State Variables

N/A

7.4.2 Environment Variables

N/A

7.4.3 Assumptions

N/A

7.4.4 Access Routine Semantics

drawSquare():

- output: Draw board square
- exception: none

drawBoard():

- transition: Uses drawSquare to display the game board

- exception: none

displayGameTermination():

- transition: Displays game termination state (checkmate, stalemate, etc.)
- exception: none

setBackground():

- transition: Sets the background colors of the display.
- exception: none

7.4.5 Local Functions

N/A

8 MIS of Web Application Output Module

8.1 Module

Web Application Output

8.2 Uses

Engine Module
Game State Module

8.3 Syntax

8.3.1 Exported Constants

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
sendData	string	string	

8.4 Semantics

8.4.1 State Variables

N/A

8.4.2 Environment Variables

N/A

8.4.3 Assumptions

N/A

8.4.4 Access Routine Semantics

sendData(string):

- output: string #Encodes game state (none, check, checkmate, stalemate), and 3 engine-generated moves
- exception: none

8.4.5 Local Functions

N/A

9 MIS of User Mode Module

9.1 Module

User Mode

9.2 Uses

Engine Module

9.3 Syntax

9.3.1 Exported Constants

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
getUserMode		string	
setUserMode	string		

9.4 Semantics

9.4.1 State Variables

userMode: string #Represents the current user mode (Normal, Beginner, Engine)

9.4.2 Environment Variables

N/A

9.4.3 Assumptions

N/A

9.4.4 Access Routine Semantics

getMode():

- output: string

$$output := userMode$$

- exception: none

setMode(string):

- transition: Sets userMode to the input user mode

$$userMode := input$$

- exception: none

9.4.5 Local Functions

N/A

10 MIS of Board Module

10.1 Module

Board

10.2 Uses

Engine Module
Game State Module

10.3 Syntax

10.3.1 Exported Constants

#define letters [‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’, ‘g’, ‘h’]
#define startFEN = ‘rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1’
#define boardDimension = 8

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
initialize			
getXYPosition	int	tuple of int	invalidIndex
getPosition	int	tuple of int	
getFenString		string	
setFenString	string		

10.4 Semantics

10.4.1 State Variables

fenString: string #Stores FEN string of current game position

10.4.2 Environment Variables

N/A

10.4.3 Assumptions

initialize is called before any other access routine.

10.4.4 Access Routine Semantics

initialize():

- transition: #Initializes fenString to the starting chess board position

$$fenString := startFEN$$

- exception: none

getXYPosition(int: squareInd):

- output: #X and Y number coordinate for an input square number. Eg. getXYPosition(14) returns (0, 6).

$$out := (squareInd \;//\; boardDimension,\; squareInd \;\%\; boardDimension)$$

- exception: none

getPosition(int: squareInd):

- output: #letter and number coordinate for an input square number. Eg. getPosition(14) returns 'g7'.

$$out := `letters[squareInd \;\%\; boardDimension]' + `boardDimension - (squareInd \;//\; boardDimension)'$$

- exception: none

getFenString():

- output:

$$out := fenString$$

- exception: none

setFenString(string: fen):

- transition:

$$fenString := fen$$

- exception: none

10.4.5 Local Functions

N/A

11 MIS of Web Application Game State Module

11.1 Module

Web Application Game State

11.2 Uses

N/A

11.3 Syntax

11.3.1 Exported Constants

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
isCheck	string	boolean	
isCheckmate	string	boolean	
isStalemate	string	boolean	

11.4 Semantics

11.4.1 State Variables

N/A

11.4.2 Environment Variables

N/A

11.4.3 Assumptions

N/A

11.4.4 Access Routine Semantics

isCheck():

- output: True if the position is 'check', false otherwise
- exception: none

isCheckmate():

- output: True if the position is 'checkmate', false otherwise
- exception: none

isStalemate():

- output: True if the position is 'stalemate', false otherwise
- exception: none

11.4.5 Local Functions

N/A

12 MIS of Engine Module

12.1 Module

Engine

12.2 Uses

N/A

12.3 Syntax

12.3.1 Exported Constants

#define depth #How many layers of depth the chess engine should use to evaluate the position #define maxSearchTime #The maximum time the chess engine should take to evaluate the position

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
evaluatePosition	string	string	

12.4 Semantics

12.4.1 State Variables

N/A

12.4.2 Environment Variables

N/A

12.4.3 Assumptions

The depth and maxSearchTime values will determined experimentally after the system is built. There is a trade-off between move quality and speed/depth of the search.

12.4.4 Access Routine Semantics

evaluatePosition(string):

- output: String containing 3 possible moves, calculated by a chess engine from the FEN input string
- exception: none

12.4.5 Local Functions

N/A

References

Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.

Daniel M. Hoffman and Paul A. Strooper. *Software Design, Automated Testing, and Maintenance: A Practical Approach*. International Thomson Computer Press, New York, NY, USA, 1995. URL http://citeseer.ist.psu.edu/428727.html.

13 Appendix

[Extra information if required —SS]