Module Interface Specification for Chess Connect

Team #4,
Alexander Van Kralingen
Arshdeep Aujla
Jonathan Cels
Joshua Chapman
Rupinder Nagra

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 ArduinoController Module

[It is also possible to use LaTeXfor hypperlinks to external documents. —SS]

6.1 ArduinoController

6.2 Uses

Arduino.h SoftwareSerial.h ChessBoard.cpp

6.3 Syntax

6.3.1 Exported Constants

None

6.3.2 Exported Access Programs

None

6.4 Semantics

6.4.1 State Variables

gameMode := enumeration { beginner, normal, engine }
gameState := enumeration { init, play, end, reset }

6.4.2 Environment Variables

HALL_PINS: input pins receiving signal fomr Hall-effect sensors LED_PINS: output pins lighting up the LEDs on the board

6.4.3 Assumptions

setup() will run before any other function.

6.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

6.4.5 Local Functions

setup
loop
lightLED

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

7 MIS of ChessBoard Module

7.1 ChessBoard

[Short name for the module —SS]

7.2 Uses

Arduino.h SoftwareSerial.h

7.3 Syntax

7.3.1 Exported Constants

int numRows := Chess board rows
int numCols := Chess board columns
int RX_PIN := receiving pin of the serial communication module
int TX_PIN := sending pin of the serial communication module

int LED_PINS[8][8] := 2-D array controlling the LED output pins
int HALL_PINS[8][8] := 2-D array controlling the Hall-effect sensor input pins

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
movePiece	int, int, int, int, PieceType	boolean	-
removePiece	int, int	Piece	-
isCheck	int, int	bool	-
isCheckmate	int, int	bool	-
boardToFEN	-	string	-
sendFEN	string	-	-
recieveMoves	-	Colour	-
lightSquare	int, int, Colour	-	-

7.4 Semantics

7.4.1 State Variables

gameMode := enumeration
check := boolean
checkmate := boolean
draw := boolean

7.4.2 Environment Variables

HALL_PINS: input pins receiving signal fomr Hall-effect sensors
LED_PINS: output pins lighting up the LEDs on the board
serialToTeensy: serial communication to and from the Teensy controller

7.4.3 Assumptions

- Serial connection between both microcontrollers will remain constant
- All LED pins will remain connected
- Hall-effect sensors will function as intended

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

7.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]

- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

7.4.5 Local Functions

pieceToChar: converting the piece type into the FEN-string character representation.

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

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.

8 Appendix

[Extra information if required —SS]