

Module Interface Specification for Chess Connect

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

April 5, 2023

1 Revision History

Table of Revisions

Table 1: Revision History

| Date | Developer(s) | Change |
|------------|-------------------------------|--|
| 2023-01-16 | Jonathan Cels, Rupinder Nagra | Web Application Modules |
| 2023-01-17 | Alexander Van Kralingen | Detailed Modules used by Arduino Mega 2560 |
| 2023-01-18 | Jonathan Cels, Rupinder Nagra | Finalized Web Application Modules |
| 2023-04-05 | Joshua Chapman | Team |

2 Symbols, Abbreviations and Acronyms

| symbol | description |
|---------------------|--------------------------------|
| M | Module |
| MIS | Module Interface Specification |
| R | Requirement |
| FEN | Forsyth-Edwards Notation |

Contents

| | | |
|----------|--|-----------|
| 1 | Revision History | i |
| | Table of Revisions | i |
| 2 | Symbols, Abbreviations and Acronyms | ii |
| 3 | Introduction | 1 |
| 4 | Notation | 1 |
| 5 | Module Decomposition | 2 |
| 6 | MIS of Arduino Controller Module | 3 |
| 6.1 | Arduino Controller | 3 |
| 6.2 | Uses | 3 |
| 6.3 | Syntax | 3 |
| 6.3.1 | Exported Constants | 3 |
| 6.3.2 | Exported Access Programs | 3 |
| 6.3.3 | Exception Descriptions | 3 |
| 6.4 | Semantics | 3 |
| 6.4.1 | State Variables | 4 |
| 6.4.2 | Environment Variables | 4 |
| 6.4.3 | Assumptions | 4 |
| 6.4.4 | Access Routine Semantics | 4 |
| 6.4.5 | Local Functions | 5 |
| 7 | MIS of Piece Identification Module | 6 |
| 7.1 | Piece Identification | 6 |
| 7.2 | Uses | 6 |
| 7.3 | Syntax | 6 |
| 7.3.1 | Exported Constants | 6 |
| 7.3.2 | Exported Access Programs | 6 |
| 7.3.3 | Exception Descriptions | 6 |
| 7.4 | Semantics | 6 |
| 7.4.1 | State Variables | 6 |
| 7.4.2 | Environment Variables | 6 |
| 7.4.3 | Assumptions | 6 |
| 7.4.4 | Access Routine Semantics | 7 |
| 7.4.5 | Local Functions | 7 |

| | | |
|-----------|--|-----------|
| 8 | MIS of Chess Board Module | 8 |
| 8.1 | Chess Board | 8 |
| 8.2 | Uses | 8 |
| 8.2.1 | Exported Constants | 8 |
| 8.2.2 | Exported Access Programs | 8 |
| 8.3 | Semantics | 8 |
| 8.3.1 | State Variables | 8 |
| 8.3.2 | Environment Variables | 9 |
| 8.3.3 | Assumptions | 9 |
| 8.3.4 | Local Functions | 10 |
| 9 | MIS of Communication Module | 11 |
| 9.1 | Communication | 11 |
| 9.2 | Uses | 11 |
| 9.3 | Syntax | 11 |
| 9.3.1 | Exported Constants | 11 |
| 9.3.2 | Exported Access Programs | 11 |
| 9.4 | Semantics | 11 |
| 9.4.1 | State Variables | 11 |
| 9.4.2 | Environment Variables | 11 |
| 9.4.3 | Assumptions | 11 |
| 9.4.4 | Access Routine Semantics | 12 |
| 9.4.5 | Local Functions | 12 |
| 10 | MIS of Web Application Input Module | 13 |
| 10.1 | Module | 13 |
| 10.2 | Uses | 13 |
| 10.3 | Syntax | 13 |
| 10.3.1 | Exported Constants | 13 |
| 10.3.2 | Exported Access Programs | 13 |
| 10.4 | Semantics | 13 |
| 10.4.1 | State Variables | 13 |
| 10.4.2 | Environment Variables | 13 |
| 10.4.3 | Assumptions | 13 |
| 10.4.4 | Access Routine Semantics | 13 |
| 10.4.5 | Local Functions | 14 |
| 11 | MIS of Display Module | 15 |
| 11.1 | Module | 15 |
| 11.2 | Uses | 15 |
| 11.3 | Syntax | 15 |
| 11.3.1 | Exported Constants | 15 |
| 11.3.2 | Exported Access Programs | 15 |

| | | |
|-----------|---|-----------|
| 11.4 | Semantics | 15 |
| 11.4.1 | State Variables | 15 |
| 11.4.2 | Environment Variables | 15 |
| 11.4.3 | Assumptions | 15 |
| 11.4.4 | Access Routine Semantics | 15 |
| 11.4.5 | Local Functions | 16 |
| 12 | MIS of Web Application Output Module | 17 |
| 12.1 | Module | 17 |
| 12.2 | Uses | 17 |
| 12.3 | Syntax | 17 |
| 12.3.1 | Exported Constants | 17 |
| 12.3.2 | Exported Access Programs | 17 |
| 12.4 | Semantics | 17 |
| 12.4.1 | State Variables | 17 |
| 12.4.2 | Environment Variables | 17 |
| 12.4.3 | Assumptions | 17 |
| 12.4.4 | Access Routine Semantics | 17 |
| 12.4.5 | Local Functions | 17 |
| 13 | MIS of User Mode Module | 18 |
| 13.1 | Module | 18 |
| 13.2 | Uses | 18 |
| 13.3 | Syntax | 18 |
| 13.3.1 | Exported Constants | 18 |
| 13.3.2 | Exported Access Programs | 18 |
| 13.4 | Semantics | 18 |
| 13.4.1 | State Variables | 18 |
| 13.4.2 | Environment Variables | 18 |
| 13.4.3 | Assumptions | 18 |
| 13.4.4 | Access Routine Semantics | 18 |
| 13.4.5 | Local Functions | 19 |
| 14 | MIS of Board Module | 20 |
| 14.1 | Module | 20 |
| 14.2 | Uses | 20 |
| 14.3 | Syntax | 20 |
| 14.3.1 | Exported Constants | 20 |
| 14.3.2 | Exported Access Programs | 20 |
| 14.4 | Semantics | 20 |
| 14.4.1 | State Variables | 20 |
| 14.4.2 | Environment Variables | 20 |
| 14.4.3 | Assumptions | 20 |

| | | |
|-----------|---|-----------|
| 14.4.4 | Access Routine Semantics | 21 |
| 14.4.5 | Local Functions | 21 |
| 15 | MIS of Web Application Game State Module | 22 |
| 15.1 | Module | 22 |
| 15.2 | Uses | 22 |
| 15.3 | Syntax | 22 |
| 15.3.1 | Exported Constants | 22 |
| 15.3.2 | Exported Access Programs | 22 |
| 15.4 | Semantics | 22 |
| 15.4.1 | State Variables | 22 |
| 15.4.2 | Environment Variables | 22 |
| 15.4.3 | Assumptions | 22 |
| 15.4.4 | Access Routine Semantics | 22 |
| 15.4.5 | Local Functions | 23 |
| 16 | MIS of Engine Module | 24 |
| 16.1 | Module | 24 |
| 16.2 | Uses | 24 |
| 16.3 | Syntax | 24 |
| 16.3.1 | Exported Constants | 24 |
| 16.3.2 | Exported Access Programs | 24 |
| 16.4 | Semantics | 24 |
| 16.4.1 | State Variables | 24 |
| 16.4.2 | Environment Variables | 24 |
| 16.4.3 | Assumptions | 24 |
| 16.4.4 | Access Routine Semantics | 24 |
| 16.4.5 | Local Functions | 25 |
| 17 | Teensy Output to Web App Module | 26 |
| 17.1 | Module | 26 |
| 17.2 | Uses | 26 |
| 17.3 | Syntax | 26 |
| 17.3.1 | Exported Constants | 26 |
| 17.3.2 | Exported Access Programs | 26 |
| 17.4 | Semantics | 26 |
| 17.4.1 | State Variables | 26 |
| 17.4.2 | Environment Variables | 26 |
| 17.4.3 | Assumptions | 26 |
| 17.4.4 | Access Routine Semantics | 27 |
| 17.4.5 | Local Functions | 27 |
| 18 | Appendix | 29 |

3 Introduction

The following document details the Module Interface Specifications for Chess Connect. The Chess Connect project aims to bridge the gap between physical and online chess play by enabling two players to play a game on a physical board while simultaneously transmitting the moves to a web application via Bluetooth. This central platform will eliminate the need for players to switch between different mediums and will provide a more flexible and accessible way for new players to learn the game.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at <https://github.com/ChessConnect/chess-connect>.

4 Notation

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 $(c_1 \Rightarrow r_1 | c_2 \Rightarrow r_2 | \dots | c_n \Rightarrow r_n)$. String concatenation uses the $+$ symbol between strings surrounded by ‘’, such as ‘this’ + ‘that’.

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

| Data Type | Notation | Description |
|-------------|----------|--|
| character | char | a single symbol or digit |
| integer | int | a number without a fractional component in $(-\infty, \infty)$ |
| boolean | boolean | true (value of 1) or false (value of 0) |
| enumeration | enum | keywords assigned an integer value in order of declaration beginning at 0 |
| structure | Piece | C++ struct data-type containing Piece-Type enumeration and int colour (0 for white, 1 for black) |

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 Module | Arduino Controller Module |
| | Arduino Module |
| | Software Serial Module |
| Behaviour-Hiding Module | Web Application Input Module |
| | Display Module |
| | Web Application Output Module |
| | Piece Identification Module |
| | Communication Module |
| | Teensy Input Bluetooth from Web App Module |
| | Teensy Output Bluetooth from Web App Module |
| Software Decision Module | User Mode Module |
| | Board Module |
| | Web Application Game State Module |
| | Engine Module |
| | Chess Board Module |

Table 2: Module Hierarchy

6 MIS of Arduino Controller Module

6.1 Arduino Controller

6.2 Uses

Arduino
Software Serial
Chess Board
Piece Identification
Communication

6.3 Syntax

6.3.1 Exported Constants

None

6.3.2 Exported Access Programs

None

6.3.3 Exception Descriptions

- **TeensyConnectionFailed:** Possible causes are breaks in the communication chain from the Teensy to
- **InvalidAction:**
- **UnknownAction:**

6.4 Semantics

| Name | In | Out | Exceptions |
|-------------------|-----------|------------|-----------------------------------|
| setup | - | - | TeensyConnectionFailed |
| loop | - | - | TeensyConnectionFailed |
| changeGameState | gameState | gameState | InvalidAction |
| changeGameMode | gameMode | string | InvalidAction |
| competeUserAction | string | userAction | InvalidAction, Un- knownAction |
| lightLED | int, int | int | - |

6.4.1 State Variables

gameMode := enum { beginner, normal, engine }
gameState := enum { init, play, end, reset }
userAction := enum { wait_white, wait_black, piece_lifted, remove_piece, promoting, valid_move, invalid_move, draw, resign, reset }
boardState := [FEN string](#) playerWarning := enum { check, checkmate, stalemate }

6.4.2 Environment Variables

HALL_PINS: input pin addresses for receiving signal from Hall-effect sensors

LED_PINS: output pin addresses for lighting up the LEDs on the board

rx_from_Teensy: input pin for communication with Teensy controller

tx_from_Teensy: output pin for communication with Teensy controller

6.4.3 Assumptions

- setup() will run before any other function.
- Connection exists between both controllers and remains constant

6.4.4 Access Routine Semantics

loop():

- transition:
 - Main control loop.
 - Polling sensors to update boardState FEN string.
 - Checking for check/checkmate/stalemate signal from Web App to update playerWarning.
 - Wait for userAction based on Hall-effect sensor inputs.
- exception: TeensyConnectionFailed

changeGameState():

- transition: Change gameState based on user input button presses (game start, draw, reset).
- exception: InvalidAction

changeGameMode():

- transition: Change gameMode based on user input button presses (beginner, normal, engine).
- exception: InvalidAction

completeUserAction():

- transition: Update boardState based on completed userAction
- exception: InvalidAction, UnknownAction

lightLED():

- output: LED_pin := HIGH ($\mathbb{Z} := 1$) or LOW ($\mathbb{Z} := 0$).
- exception: TeensyConnectionFailed

6.4.5 Local Functions

setup():

- transition: initialize serial connection; read board state; game state set to "init"
- exception: TeensyConnectionFailed

7 MIS of Piece Identification Module

7.1 Piece Identification

7.2 Uses

None

7.3 Syntax

7.3.1 Exported Constants

None

7.3.2 Exported Access Programs

| Name | In | Out | Exceptions |
|--------------|-----------------|-------|---------------------|
| readSensors | int | Piece | SensorOffline |
| waitForPiece | int, int, Piece | bool | PieceMissingTimeout |

7.3.3 Exception Descriptions

- **SensorOffline:** Can occur when a sensor returns a null value or produces outputs that do not align with the intended state included in the state machine.
- **PieceMissingTimeout:** A strict timer of 5 minutes is placed on the player to return the piece to the square. The PieceMissingTimeout occurs if this timer is finished without registering a placed piece.

7.4 Semantics

7.4.1 State Variables

None

7.4.2 Environment Variables

sensorInput: readings from various hall-effect sensors

7.4.3 Assumptions

Hall-effect sensors will give accurate readings.

7.4.4 Access Routine Semantics

readSensors():

- output: Piece
- exception: SensorOffline

waitForPiece():

- transition: Waiting to send signal based on a sensor transition from $HALL_PIN[\mathbb{Z}][\mathbb{Z}] := \mathbb{R} \Rightarrow 0$
- output: bool value of $(PieceNotPlaced \Rightarrow false | PiecePlaced \Rightarrow true)$
- exception: PieceMissingTimeout

7.4.5 Local Functions

None

8 MIS of Chess Board Module

8.1 Chess Board

8.2 Uses

Arduino

Piece Identification

8.2.1 Exported Constants

int numRows : Chess board rows

int numCols : Chess board columns

int LED_PINS[numRows][numCols] : 2-D array controlling the LED output pins

int HALL_PINS[numRows][numCols] : 2-D array controlling the Hall-effect sensor input pins

8.2.2 Exported Access Programs

None

8.3 Semantics

| Name | In | Out | Exceptions |
|-----------------------------|------------------------------------|---------|--------------------|
| movePiece | int, int, int, int, Piece- Type | boolean | InvalidMove |
| removePiece | int, int | Piece | InvalidMove |
| isCheckmateCheckOrStalemate | int, int | bool | - |
| boardToFEN | - | string | - |
| recieveMoves | - | Colour | InvalidMove |
| lightSquare | int, int, Colour | - | DigitalWriteFailed |
| pieceToChar | Piece | char | - |

8.3.1 State Variables

gameMode := enumeration

check := boolean

checkmate := boolean

draw := boolean

8.3.2 Environment Variables

HALL_PINS: input pins receiving signal from Hall-effect sensors

LED_PINS: output pins lighting up the LEDs on the board

serialToTeensy: serial communication to and from the Teensy controller

8.3.3 Assumptions

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

movePiece():

- transition: Update Piece type and colour on the "to" square, while removing the piece from the "from" square.
- exception: InvalidMove

removePiece():

- transition: Update Piece type and colour on the "to" square, while removing the piece from the "from" square. Remove the piece taken by the opponent.
- output: returns the Piece that was removed.
- exception: InvalidMove

isCheckmateCheckOrStalemate():

- transition: Update game state based on a command sent from the Web Application.
- exception: None

boardToFEN():

- output: FEN string representation of the current board state.
- exception: None

recieveMoves():

- transition: Process best moves recieved from the web application and light appropriate LED's.
- exception: InvalidMove

lightSquare():

- transition: Light appropriate LED's based on various conditions such as game mode, game state, check/mate/stalemate warning, etc.
- exception: DigitalWriteFailed

8.3.4 Local Functions

pieceToChar():

- output: Converting the Piece type into the FEN-string character representation.
- exception: None

9 MIS of Communication Module

9.1 Communication

9.2 Uses

Arduino.h SoftwareSerial.h

9.3 Syntax

9.3.1 Exported Constants

None

9.3.2 Exported Access Programs

| Name | In | Out | Exceptions |
|----------------|--------|--------|----------------|
| encodeMessage | string | - | UnknownAction |
| decodeMessage | - | string | UnknownCommand |
| processCommand | string | string | InvalidCommand |

9.4 Semantics

9.4.1 State Variables

command: The decoded message to update values (game state, game mode, light specific LED, etc.).

9.4.2 Environment Variables

messageEncoder: The string formatting to send a message to the Teensy Controller via Serial Communication.

messageDecoder: The string formatting to read a message from the Teensy Controller via Serial Communication.

9.4.3 Assumptions

- Communication string format remains consistent
- Connection exists between both controllers and remains constant

9.4.4 Access Routine Semantics

encodeMessage():

- output: Translate game state or action into encoded string to be read by Teensy or the Web Application
- exception: UnknownAction

decodeMessage():

- output: Translate encoded message from Teensy or the Web Application and convert into state change command
- exception: UnknownCommand

processCommand():

- transition: Command received from Web Application or Teensy controller will be used to change the chess board accordingly.
- -¿ This could be to change the game state, game mode, player warning (check, check-mate, stalemate) or to light appropriate LED's
- exception: InvalidCommand

9.4.5 Local Functions

None

10 MIS of Web Application Input Module

10.1 Module

Web Application Input

10.2 Uses

[Board Module](#)

[User Mode Module](#)

10.3 Syntax

10.3.1 Exported Constants

10.3.2 Exported Access Programs

| Name | In | Out | Exceptions |
|------------|--------|---------------|--------------|
| parseInput | string | seq of string | invalidInput |

10.4 Semantics

10.4.1 State Variables

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

10.4.2 Environment Variables

N/A

10.4.3 Assumptions

N/A

10.4.4 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.

10.4.5 Local Functions

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

11 MIS of Display Module

11.1 Module

Display

11.2 Uses

[Board Module](#)

11.3 Syntax

11.3.1 Exported Constants

11.3.2 Exported Access Programs

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

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

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

11.4.5 Local Functions

N/A

12 MIS of Web Application Output Module

12.1 Module

Web Application Output

12.2 Uses

[Engine Module](#)

[Game State Module](#)

12.3 Syntax

12.3.1 Exported Constants

12.3.2 Exported Access Programs

| Name | In | Out | Exceptions |
|----------|--------|--------|------------|
| sendData | string | string | |

12.4 Semantics

12.4.1 State Variables

N/A

12.4.2 Environment Variables

N/A

12.4.3 Assumptions

N/A

12.4.4 Access Routine Semantics

sendData(string):

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

12.4.5 Local Functions

N/A

13 MIS of User Mode Module

13.1 Module

User Mode

13.2 Uses

[Engine Module](#)

13.3 Syntax

13.3.1 Exported Constants

13.3.2 Exported Access Programs

| Name | In | Out | Exceptions |
|-------------|--------|--------|------------|
| getUserMode | | string | |
| setUserMode | string | | |

13.4 Semantics

13.4.1 State Variables

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

13.4.2 Environment Variables

N/A

13.4.3 Assumptions

N/A

13.4.4 Access Routine Semantics

getMode():

- output: string

output := userMode

- exception: none

setMode(string):

- transition: Sets `userMode` to the input user mode

userMode := input

- exception: none

13.4.5 Local Functions

N/A

14 MIS of Board Module

14.1 Module

Board

14.2 Uses

[Engine Module](#)

[Game State Module](#)

14.3 Syntax

14.3.1 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
```

14.3.2 Exported Access Programs

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

14.4 Semantics

14.4.1 State Variables

fenString: string #Stores FEN string of current game position

14.4.2 Environment Variables

N/A

14.4.3 Assumptions

initialize is called before any other access routine.

14.4.4 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

14.4.5 Local Functions

N/A

15 MIS of Web Application Game State Module

15.1 Module

Web Application Game State

15.2 Uses

N/A

15.3 Syntax

15.3.1 Exported Constants

15.3.2 Exported Access Programs

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

15.4 Semantics

15.4.1 State Variables

N/A

15.4.2 Environment Variables

N/A

15.4.3 Assumptions

N/A

15.4.4 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

15.4.5 Local Functions

N/A

16 MIS of Engine Module

16.1 Module

Engine

16.2 Uses

N/A

16.3 Syntax

16.3.1 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

16.3.2 Exported Access Programs

| Name | In | Out | Exceptions |
|------------------|--------|--------|------------|
| evaluatePosition | string | string | |

16.4 Semantics

16.4.1 State Variables

N/A

16.4.2 Environment Variables

N/A

16.4.3 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.

16.4.4 Access Routine Semantics

evaluatePosition(string):

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

16.4.5 Local Functions

N/A

17 Teensy Output to Web App Module

17.1 Module

Teensy Output via bluetooth to the Web Application

17.2 Uses

Sends current game state and game mode to the Web Application via Bluetooth

17.3 Syntax

17.3.1 Exported Constants

`#define baud rate` #The baud rate of the serial communication system
`#define stringFormat` #The format of the string remains constant to perform proper communication between the two Arduinos.

17.3.2 Exported Access Programs

| Name | In | Out | Exceptions |
|--------------|--------|--------|------------|
| copyFEN | string | string | |
| copyGameMode | string | string | |

17.4 Semantics

17.4.1 State Variables

local lastGameState
local currGameState

17.4.2 Environment Variables

N/A

17.4.3 Assumptions

The size of the string passed to the Web Application aligns with the designed format that the web application is expecting

17.4.4 Access Routine Semantics

sendCurrState(pin):

- output: String containing the current game state
- exception: none

sendCurrMode(pin):

- output: String containing the current game mode
- exception: none

17.4.5 Local Functions

N/A

References

- FEN. Fen (forsyth-edwards notation) - chess terms. <https://www.chess.com/terms/fen-chess>. Accessed: 2023-01-18.
- 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>.

18 Appendix

[Extra information if required —SS]