SE 3XA3: Module Interface Specification
SupremeChess

Team #2, The Triple Grobs
Rupinder Nagra (nagrar5)
Pesara Amarasekera (amarasep)
Jonathan Cels (celsj)

March 18, 2021

Input Module

Module

Input

Uses

Shared Data Module

Syntax

#define BOARD_X 8
#define BOARD_Y 8

Routine Names	Inputs	Outputs	Exceptions
getScreenSize			
setBoardSize			
onClickPress	mouseDown	seq of int	
onClickRelease	mouseUp	seq of int	notOnBoard
getClickPressSquare		seq of int	noClick
getClickReleaseSquare		seq of int	noClick

Semantics

State Variables

screenX: int #Screen pixel width
screenY: int #Screen pixel height
boardX: int #Board pixel width
boardY: int #Board pixel height
boardStart: sequence of int #(x, y) pixel coordinates of bottom left board corner
squareX: int #Square pixel width
squareY: int #Square pixel height

State Invariant

$|boardBounds| \leq screenX * screenY$
$|boardX| = |boardY|$
$|squareX| = |squareY|$
$|boardX| = |squareX| * BOARD\_X$
$|boardY| = |squareY| * BOARD\_Y$

Environment Variables

mouseDown: #Left click press provided by the user mouse
mouseUp: #Left click release provided by the user mouse

Assumptions and Design Decisions

getScreenSize is called before any other access routine, followed immediately by setBoardSize

Access Routine Semantics

getScreenSize

transition: screenX = screen pixel width, screenY = screen pixel height
exceptions: none

setBoardSize

transition: boardX = board pixel width based on screen width, boardY = board pixel width based on screen height, boardStart = bottom left coordinate of the board based on screen size. $squareX = \frac{boardX}{BOARD\_X}$. $squareY = \frac{boardY}{BOARD\_Y}$
exceptions: none

onClickPress:

output: tuple of (pixel value of mouse along x axis, pixel value of mouse along y axis).
exceptions: none

onClickRelease:

output: tuple of (pixel value of mouse along x axis, pixel value of mouse along y axis).
exceptions: notOnBoard, if the tuple of coordinates is not within the bounds of the board.

getClickPressSquare

output: tuple of (integer from 1 to BOARD_X, integer from 1 to BOARD_Y) where the values represent an x and y square coordinate respectively.
exceptions: noClick, if called before a click input occurs

getClickReleaseSquare

output: tuple of (integer from 1 to BOARD_X, integer from 1 to BOARD_Y) where the values represent an x and y square coordinate respectively.
exceptions: noClick, if called before a click input occurs

Output Module

Module

Output

Uses

Shared Data Module
App Module

Syntax

N/A

Routine Names	Inputs	Outputs	Exceptions
drawSquare	string		
drawWhiteTimer	double		
drawBlackTimer	double		
drawBoard	sequence of (sequence of int)		
drawChat	sequence of (char, string)		
drawOptions	sequence of int		
setBackground	string		

Semantics

State Variables

N/A

State Invariant

N/A

Environment Variables

Assumptions and Design Decisions

N/A

Access Routine Semantics

drawSquare

output: draw board square
exceptions: none

drawWhiteTimer

transition: draw value of the white player's timer
exceptions: none

drawBlackTimer

transition: draw value of the black player's timer
exceptions: none

drawBoard

transition: use *drawSquare* routine to build game board
exceptions: none

drawChat

transition: draw strings of chat messages sent by players
exceptions: none

drawOptions

transition: draw game options (checkmate, draw, stalemate, etc.)
exceptions: none

setBackground

transition: set background colour
exceptions: none

Board Module

Module

Board

Uses

Piece Module

Syntax

#define letter ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']

Routine Names	Inputs	Outputs	Exceptions
initialize			
getXYPosition	int	tuple of int	invalidIndex
isBlack	int	boolean	
getPosition	int	tuple of int	invalidPosition

Semantics

State Variables

currTurn: string #Signifies player with the current turn

State Invariant

N/A

Environment Variables

Assumptions and Design Decisions

initialize is called before any other access routine.

Access Routine Semantics

initialize

transition: currTurn is set to starting chess board state, $currTurn = `w'$
exceptions: none

getXYPosition

output: tuple of (x, y) coordinates from board square index
exceptions: the board index does not exist

isBlack

output: string of piece image location
exceptions: none

getPosition

output: tuple of (board *letter* index, y coordinate index)
exceptions: the board position does not exist

Shared Data Module

Module

Shared Data

Uses

App Module
Timer Module
Chat Module
AI Module

Syntax

#define BOARD_X 8
#define BOARD_Y 8

Routine Names	Inputs	Outputs	Exceptions
initialize	double		
getBoardState		sequence of (sequence of int)	
getTurnState		char	
getWhiteTimer		double	
getBlackTimer		double	
getChatHistory		sequence of (char, string)	
setBoardState	sequence of (sequence of int)		invalidBoardState
setTimerValues	double, double		
addMessage	char, string		
clearChat			

Semantics

State Variables

boardState: sequence of (sequence of int) #Chess board state with different integers for different pieces
turnState: char #'w' or 'b', representing which player's turn it is, white or black.
whiteTimer: double #Value of the player with the white pieces' turn timer.
blackTimer: double #Value of the player with the black pieces' turn timer.
chatHistory: sequence of (char, string) #Sequence of tuples where the character is 'w' or 'b' representing the player, and the string is a chat message they sent.

State Invariant

$|boardState| = |boardState[i]| = BOARD\_X = BOARD\_Y$
$|chatHistory| \geq 0$

Environment Variables

Assumptions and Design Decisions

initialize is called before any other access routine.

Access Routine Semantics

initialize

transition: boardState is set to starting chess board state, $turnState = `w'$,
$whiteTimer = blackTimer =$ input value, chatHistory is set to an empty sequence.
exceptions: none

getBoardState

output: current board state
exceptions: none

getTurnState

output: turnState
exceptions: none

getWhiteTimer

output: whiteTimer
exceptions: none

getBlackTimer

output: blackTimer
exceptions: none

getChatHistory

output: chatHistory
exceptions: none

setBoardState

transition: updates boardState to match the given board state
exceptions: invalidBoardState, if the given board state is impossible to reach

setTimerValues

transition: updates both whiteTimer and blackTimer to match the given timer values
exceptions: none

addMessage

transition: adds the given message to chatHistory
exceptions: none

clearChat

transition: sets chatHistory to an empty sequence exceptions: none

Piece Module

Module

Piece

Uses

N/A

Syntax

Routine Names	Inputs	Outputs	Exceptions
getPiece	array of strings	string	noPiece
dragPiece	object of int		

Semantics

State Variables

pieceImg: string #Image location of piece
id: string #Unique piece ID with piece type, colour, position

State Invariant

N/A

Environment Variables

type: #The piece type (PAWN, KNIGHT, BISHOP, ROOK, QUEEN, KING)
colour: #The colour of the piece

Assumptions and Design Decisions

N/A

Access Routine Semantics

getPiece

output: string of piece image location
exceptions: the piece type does not exist

dragPiece

transition: id = concatenated string of piece type, colour, and position
exceptions: the piece type does not exist

Game Module

Module

Game

Uses

N/A

Syntax

Routine Names	Inputs	Outputs	Exceptions
initGame			
resetGame			
handleMove	tuple of int		
getGameResult		string	

Semantics

State Variables

currMoves: array of strings #Shows set of moves of current game
boardState: string #Saves current board state

State Invariant

N/A

Environment Variables

Assumptions and Design Decisions

initGame is called before any other access routine.

Access Routine Semantics

initGame

transition: set to default board state
exceptions: none

resetGame

transition: reset to default board state
exceptions: none

handleMove

transition: append move to currMoves and update board state
exceptions: none

getGameResult

output: string of game result (checkmate, draw, stalemate, etc.)
exceptions: none

App Module

Module

App

Uses

Game Module
Board Module

Syntax

Routines	Inputs	Outputs	Exceptions
initialize			
setBoard	Board, Game	Board	
setGame	Board, Game	Game	inValidMove

Semantics

State Variables

Board: Board Module
Game: Game Module

State Invariant

N/A

Environment Variables

Assumptions and Design Decisions

initialize is called before any other access routine

Access Routine Semantics

initialize

transition: Board and Game modules are initialized to begin functions
exceptions: none

setBoard

transition: updates the Board using the Game for representation purposes
exceptions: none

setGame

transition: updates the Game to a new valid configuration based on the moves played on the Board
exceptions: invalidMove, if the played move is illegal

Chat Module

Module

Chat

Uses

N/A

Syntax

Routines	Inputs	Outputs	Exceptions
initialize			
setMessage	String		noChatInitialized
getMessage		String	noChatInitialized
endChat			noChatInitialized

Semantics

State Variables

N/A

State Invariant

N/A

Environment Variables

Assumptions and Design Decisions

initialize is called before any other access routine

Access Routine Semantics

initialize

transition: The chat is initialized with no chat data
exceptions: none

setMessage

transition: updates the chat with a new message
exceptions: noChatInitialized, when no chat is initialized before updating the message data

getMessage

output: gets the chat data in string format
exceptions: noChatInitialized, when no chat is initialized to get data from

endChat

transition: ends the chat upon request
exceptions: noChatInitialized, when no chat is initialized to get end

Timer Module

Module

Timer Data

Uses

N/A

Syntax

Routine Names	Inputs	Outputs	Exceptions
initialize	double, double		
getTime		double	
startTimer			timeUp
pauseTimer			

Semantics

State Variables

startValue: double #Starting time value of the timer incrementValue: double #Value added to time after every move currentValue: double #Current time value of the timer player: char #'w' or 'b', representing which player's timer it is, white or black.

State Invariant

$(incrementValue = 0) \Rightarrow |startValue| \geq |currentValue|$

Environment Variables

Assumptions and Design Decisions

initialize is called before any other access routine.

Access Routine Semantics

initialize

transition: startValue is set to the first input representing the total time on the timer. incrementValue is set to the second input representing the time increment added after each move. currentValue is set equal to startValue.
exception: none

getTime

output: currentValue.
exceptions: timeUp, if currentValue is 0.

startTimer

transition: currentValue starts decreasing. exceptions: timeUp, if currentValue reaches 0 at any point.

pauseTimer

transition: currentValue stops decreasing and timeIncrement is added to currentValue. exceptions: none

AI Module

Module

AI

Uses

Game Module

Syntax

#define WPAWN 1
#define WBISHOP 2
#define WROOK 3
#define WKNIGHT 4
#define WQUEEN 5
#define WKING 6
#define BPAWN 11
#define BBISHOP 12
#define BROOK 13
#define BKNIGHT 14
#define BQUEEN 15
#define BKING 16
#define PawnWeight 10
#define RookWeight 50
#define BishopWeight 40
#define KnightWeight 30
#define QueenWeight 90
#define KingWeight 420
#define MobilityWeight 10

Routines	Inputs	Outputs	Exceptions
initialize	Game		
setBoard	Game		
setGame	Game		
evaluateBoard	Game	Integer	
makeMove		(PieceVal, BoardXPos, BoardYPos)	

Semantics

State Variables

internalBoard: sequence of sequence of integers
GameTree: Tree of Boards

State Invariant

N/A

Environment Variables

Assumptions and Design Decisions

initialize is called before any other access routine

Access Routine Semantics

initialize

transition: internal Board representation (internalBoard) initialized to begin functions

setBoard

transition: updates internal Board using the external Game state for representation purposes
exceptions: none

setGame

transition: updates the internalBoard to a new valid configuration based on the moves played (the new Game state)
exceptions: none

evaluateBoard

output: evaluates the board according to the defined weights and cost function and returns *eval* (defined below).

$$\begin{aligned} materialScore = {} & KingWeight(wK - bK) + QueenWeight(wQ - bQ) \\ & + RookWeight(wR - bR) + KnightWeight(wN - bN) \\ & + BishopWeight(wB - bB) + PawnWeight(wP - bP) \end{aligned}$$

$$\begin{aligned} mobilityScore &= mobilityWeight(wMobility - bMobility) \\ eval &= (mobilityScore + materialScore)whoMoved \end{aligned}$$

where $wK = \#ofWhiteKings$, $wQ = \#ofWhiteQueens$, $wP = \#ofWhitePawns$,
$wB = \#ofWhiteBishops$, $wR = \#ofWhiteRooks$, $wN = \#ofWhiteKnights$
$bK = \#ofBlackKings$, $bQ = \#ofBlackQueens$, $bP = \#ofBlackPawns$,
$bB = \#ofBlackBishops$, $bR = \#ofBlackRooks$, and $bN = \#ofBlackKnights$.

The mobility of a side is defined by the number of squares reachable by the pieces of a side. $wMobility = \#squaresreachablebywhite$ and $bMobility = \#squaresreachablebyblack$.

The *whoMoved* value is 1 if the current player is white and -1 if the current player is black. exceptions:none

makeMove

output: Evaluate the current game state and makes a move, returns a tuple that represents the piece being moved, and the position on the Board where it should be moved to. exceptions: none