Checkers Design Document

me 3

Contents

1 Introduction

This document contains the decomposition, uses relationship, and traceability.

2 Module Guide

Modules are stuff.

2.1 Hardware Hiding Module

2.1.1 Input Module

Type	Hardware Module
Secret	This module translate mouse clicks and keyboard presses to be used by the rest of the software.
Responsibilites	This module will take mouse and keyboard input and convert it to software usable states.
Uses	None
Design	None
Explanation	The input module is a hardware hiding module since it translates hardware inputs to software.

2.2 Behaviour Hiding Module

2.2.1 Piece Module

Type	Software Module
Secret	This module hides and separates specific piece information.
Responsibilites	This will hold the necessary components to describe what a game piece will contain, which will be seperate from the game board.
Uses	None
Design	3.1
Explanation	I DONT KNOW: The piece is a part of behaviour hiding since the piece module holds specific piece information and outputs values needed by other modules.

2.3 Software Decision Hiding Module

2.3.1 Board Module

Type	Software Module
Secret	This module serves to hide the secret of how the board is defined internally.
Responsibilites	This module is responsible for holding the necessary components and attributes to setup the board and describe piece locations.
Uses	2.2.1
Design	3.2
Explanation	The board is a part of software decision hiding since the board implements a data structure that holds the placement of the pieces, this data structure might be changed for increased performance. Another software decision is deciding how to take user input to parse the placement of pieces.

3 Module Design (MIS and MID)

3.1 Piece Module

3.1.1 Interface

Types

typeState	enumerate if the piece is normal or king
player	enumerate if piece owned by Black or White

Constants

None

Access Programs

getType()	Retrieves the piece's current type.
setType(newType : typeState)	Changes the piece's type.
getOwner()	Says who owns the piece.

3.1.2 Implementation

Variables

pieceType : typeState	holds current piece type
owner : player	holds information of the piece's owner

Access Programs

getType() : typeState

Inputs	None
Updates	None
Outputs	pieceType

setType(newType : typeState)

Inputs	newType
Updates	pieceType
Outputs	None

getOwner() : player

Inputs	None
Updates	None
Outputs	owner

3.2 Board Module

3.2.1 Interface

Types

None

Constants

None

Access Programs

setUpBoard()	Sets up board based on user input.
getPiece(col : int, row : int)	This method is used to determine if a piece exists on a square of the board. If the piece does exist, we pass it along to the caller.
placePiece(col : int, row : int, piece : Piece)	Places the piece on the board while checking if the placement is legal (in terms of checkers).
movePiece(fromCol : int, fromRow : int, toCol : int, toRow : int)	Moves the piece from starting to end positions while checking if the movement is valid (in terms of checkers).
clear()	Removes all pieces from the board.

3.2.2 Implementation

Types

None

Constants

None

Variables

pieceArray[] The board will be implemented as an array.

Access Programs

setUpBoard(input : string)

Inputs	input
Outputs	pieceArray[]
Updates	None
Desc	Parses input to be interpreted as Piece locations. Place Piece on correct Piece location using the PlacePiece() access program.

getPiece(col : int, row : int) : Piece

Inputs	col, row
Outputs	piece
Updates	None

placePiece(col : int, row : int, piece : Piece)

Inputs	col, row, piece
Outputs	pieceArray[]
Updates	None
Desc	If piece placement is valid, it will put it there in the data structure.

movePiece(fromCol : int, fromRow : int, toCol : int, toRow : int)

Inputs	None
Outputs	None
Updates	None

clear()

Inputs	None
Outputs	pieceArray[]
Updates	None