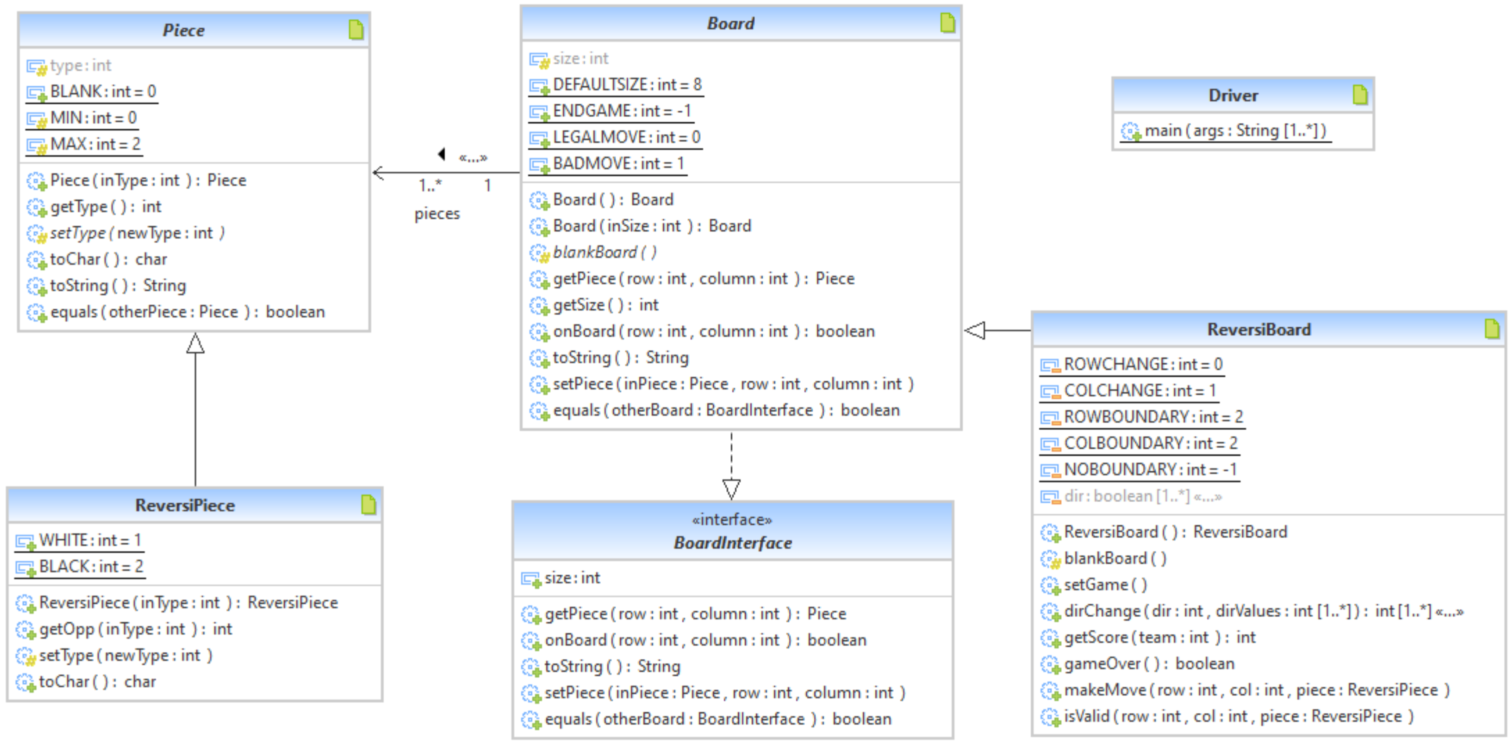
Software Development Report
for
CSC 321: Programming III: Spring 2023
Reversi Project: Non-GUI Classes
by
Noah Jackson

Problem Summary
The purpose of this program is to create a playable version of the game "Reversi" in Java.

Implementation Requirements
- The program must be playable through text and graphic implementations.
- The game board and game pieces must be implemented with subclasses.
- The game must be played correctly.
- The system must support two players across two computers.
- Games can be saved, loaded, reset, and exited.
- The system must gather statistics over the games played.
- The system must implement an AI player.

System Design

- UML Diagram


Piece
type : int
BLANK : int = 0
MIN : int = 0
MAX : int = 2
Piece (inType : int) : Piece
getType ( ) : int
setType (newType : int)
toChar ( ) : char
toString ( ) : String
equals (otherPiece : Piece) : boolean
«...»
1..*
1
pieces
Board
size : int
DEFAULTSIZE : int = 8
ENDGAME : int = -1
LEGALMOVE : int = 0
BADMOVE : int = 1
Board ( ) : Board
Board (inSize : int) : Board
blankBoard ( )
getPiece (row : int , column : int) : Piece
getSize ( ) : int
onBoard (row : int , column : int) : boolean
toString ( ) : String
setPiece (inPiece : Piece , row : int , column : int)
equals (otherBoard : BoardInterface) : boolean
Driver
main (args : String [1..*])
ReversiPiece
WHITE : int = 1
BLACK : int = 2
ReversiPiece (inType : int) : ReversiPiece
getOpp (inType : int) : int
setType (newType : int)
toChar ( ) : char
«interface»
BoardInterface
size : int
getPiece (row : int , column : int) : Piece
onBoard (row : int , column : int) : boolean
toString ( ) : String
setPiece (inPiece : Piece , row : int , column : int)
equals (otherBoard : BoardInterface) : boolean
ReversiBoard
ROWCHANGE : int = 0
COLCHANGE : int = 1
ROWBOUNDARY : int = 2
COLBOUNDARY : int = 2
NOBOUNDARY : int = -1
dir : boolean [1..*] «...»
ReversiBoard ( ) : ReversiBoard
blankBoard ( )
setGame ( )
dirChange (dir : int , dirValues : int [1..*]) : int [1..*] «...»
getScore (team : int) : int
gameOver ( ) : boolean
makeMove (row : int , col : int , piece : ReversiPiece)
isValid (row : int , col : int , piece : ReversiPiece)

- Complex Methods
    - isValid(row: int, col: int, piece: ReversiPiece), void
        - Declare variables/Objects
            1. Color, rowStep, colStep: int
            2. dirValues: int[]
            3. Border: boolean
        - Initialize
            1. color = color of the piece being validated
            2. rowStep initialized to row
            3. colStep initialized to col
            4. Border = false
        - Check if the proposed move is on the board, else return.
        - Check if the proposed piece is blank
            1. Start of for loop with int i for checking all 8 directions
                - Set dirValues using the method dirChange(i, dirValues), this sets the direction being checked.
                - If the row or column are inside the boundary of the board
                    - Increase rowStep and colStep by the changes dictated by the dirValues array.
                    - Try and catch array out bounds exception are piece on the border
                        - While the next piece in the current direction is the opposite color
                        - Increase rowStep and colStep by the changes dictated by the dirValues array.
                    - If an exception is caught
                        - border: boolean = true.
                    - If border = true
                        - Boolean array dir[i] equals false for that direction.
                    - Else
                        - If the first piece checked in the current does not equal color: int
                            - If the last piece in the current direction equals color: int
                                - Boolean array dir[i] equals true for that direction.
                            - Else
                                - Boolean array dir[i] equals false for that direction.
                        - Reset rowStep to row and ColStep to col
    - makeMove(row: int, col: int, piece: ReversiPiece), void
        - Declare variables/Objects

```
                1. Color, rowStep, colStep: int
                2. dirValues: int[]
        ➢ Initialize
                1. color = color of the piece being validated
                2. rowStep initialized to row
                3. colStep initialized to col
        ➢ Call isValid(row, col, piece) to set the dir[] values true or false.
        ➢ If all directions are false.
                1. Print error "Invalid Move"
                2. Return
        ➢ Place the proposed move on the board
        ➢ Flip all of the relevant pieces in the directions that are true.
                1. Start of for loop with int i for checking all 8 directions
                        ○ Set dirValues using the method dirChange(i, dirValues),
                          this sets the direction being checked.
                        ○ If dir[i] is true.
                                ■ Increase rowStep and colStep by the changes
                                  dictated by the dirValues array.
                                ■ While the next piece in the current direction is the
                                  opposite color
                                        ● Flip the piece to the opposite color.
                                        ● Increase rowStep and colStep by the
                                          changes dictated by the dirValues array.
                                ■ Reset rowStep to row and ColStep to col
```

Testing Report

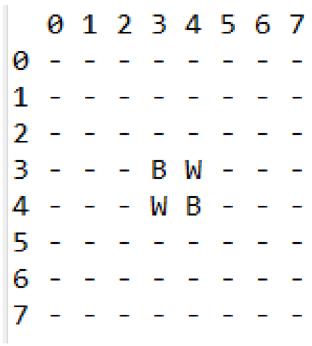
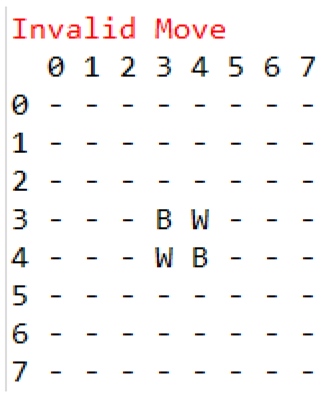
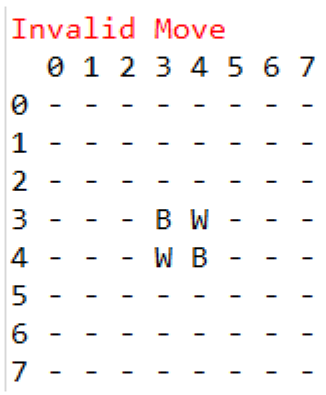
Case	Input (shows the initial board state, and the move being made)	Expected Output	Observed Output	P or F

S1	This will test a move that will flip pieces in one direction.	`  0 1 2 3 4 5 6 7`
		`0 - - - - - - - -`
		`1 - - - - - - - -`
		`2 - - - - - - - -`
		`3 - - - B W - - -`
		`4 - - - W W - - -`
		`5 - - - - W - - -`
		`6 - - - - - - - -`
		`7 - - - - - - - -`
			`  0 1 2 3 4 5 6 7`
			`0 - - - - - - - -`
			`1 - - - - - - - -`
			`2 - - - - - - - -`
			`3 - - - B W - - -`
			`4 - - - W W - - -`
			`5 - - - - W - - -`
			`6 - - - - - - - -`
			`7 - - - - - - - -`
				pass

	`  0 1 2 3 4 5 6 7`
	`0 - - - - - - - -`
	`1 - - - - - - - -`
	`2 - - - - - - - -`
	`3 - - - B W - - -`
	`4 - - - W B - - -`
	`5 - - - - - - - -`
	`6 - - - - - - - -`
	`7 - - - - - - - -`
	Row = 5, col = 4, White			

S2	This will test a move that flips pieces in multiple directions.
	`  0 1 2 3 4 5 6 7`
	`0 - - - - - - - -`
	`1 - - - - - - - -`
	`2 - - - - - - - -`
	`3 - - - B W - - -`
	`4 - - - W B - - -`
	`5 - - - - - B W -`
	`6 - - - - - - - -`
	`7 - - - - - - - -`
	Row = 5, col = 4, White
		`  0 1 2 3 4 5 6 7`
		`0 - - - - - - - -`
		`1 - - - - - - - -`
		`2 - - - - - - - -`
		`3 - - - B W - - -`
		`4 - - - W W - - -`
		`5 - - - - W W W -`
		`6 - - - - - - - -`
		`7 - - - - - - - -`
			`  0 1 2 3 4 5 6 7`
			`0 - - - - - - - -`
			`1 - - - - - - - -`
			`2 - - - - - - - -`
			`3 - - - B W - - -`
			`4 - - - W W - - -`
			`5 - - - - W W W -`
			`6 - - - - - - - -`
			`7 - - - - - - - -`
				pass

S3	This will test a move that flips pieces in multiple directions.
	`  0 1 2 3 4 5 6 7`
	`0 - - - - - - - -`
	`1 - - - - - - - -`
	`2 - - - - - - - -`
	`3 - - - B W - - -`
	`4 - - - W B - - -`
	`5 - - - - - B W -`
	`6 - - - - B - - -`
	`7 - - - - W - - -`
	Row = 5, col = 4, White
		`  0 1 2 3 4 5 6 7`
		`0 - - - - - - - -`
		`1 - - - - - - - -`
		`2 - - - - - - - -`
		`3 - - - B W - - -`
		`4 - - - W W - - -`
		`5 - - - - W W W -`
		`6 - - - - W - - -`
		`7 - - - - W - - -`
			`  0 1 2 3 4 5 6 7`
			`0 - - - - - - - -`
			`1 - - - - - - - -`
			`2 - - - - - - - -`
			`3 - - - B W - - -`
			`4 - - - W W - - -`
			`5 - - - - W W W -`
			`6 - - - - W - - -`
			`7 - - - - W - - -`
				pass

Er1	An invalid move will be made.
	
	`  0 1 2 3 4 5 6 7`
	`0 - - - - - - - -`
	`1 - - - - - - - -`
	`2 - - - - - - - -`
	`3 - - - B W - - -`
	`4 - - - W B - - -`
	`5 - - - - - - - -`
	`6 - - - - - - - -`
	`7 - - - - - - - -`
	Row = 1, col = 1, Black
		
		`Invalid Move`
		`  0 1 2 3 4 5 6 7`
		`0 - - - - - - - -`
		`1 - - - - - - - -`
		`2 - - - - - - - -`
		`3 - - - B W - - -`
		`4 - - - W B - - -`
		`5 - - - - - - - -`
		`6 - - - - - - - -`
		`7 - - - - - - - -`
			
			`Invalid Move`
			`  0 1 2 3 4 5 6 7`
			`0 - - - - - - - -`
			`1 - - - - - - - -`
			`2 - - - - - - - -`
			`3 - - - B W - - -`
			`4 - - - W B - - -`
			`5 - - - - - - - -`
			`6 - - - - - - - -`
			`7 - - - - - - - -`
				pass

S1 - S4 are Standard cases
Er1 - Error cases

Analysis of Time Used
- This program and development report took approximately 15 hours to complete. This code went through several editing phases, being checked by others on the team, causing the time used to increase as testing and debugging took place.

Outside resources used

- Online Reversi game (https://cardgames.io/reversi/)

Security Report

- There are no security issues with the project at this time.

Ethical Report

- This program creates a game that could be used for personal enjoyment.
- There are no negative ethical concerns with the project at this time.

Future Improvements
- Implement the game and player classes to allow for full gameplay.
- Fully implement the graphical user interface with the game.

Lessons learned

- Learned how to create and implement abstract classes.
- Learned how to use interfaces.
- Learned how to apply Listov's substitution principal.

Improvement of work over last time

- This is the first SDR created for this project.