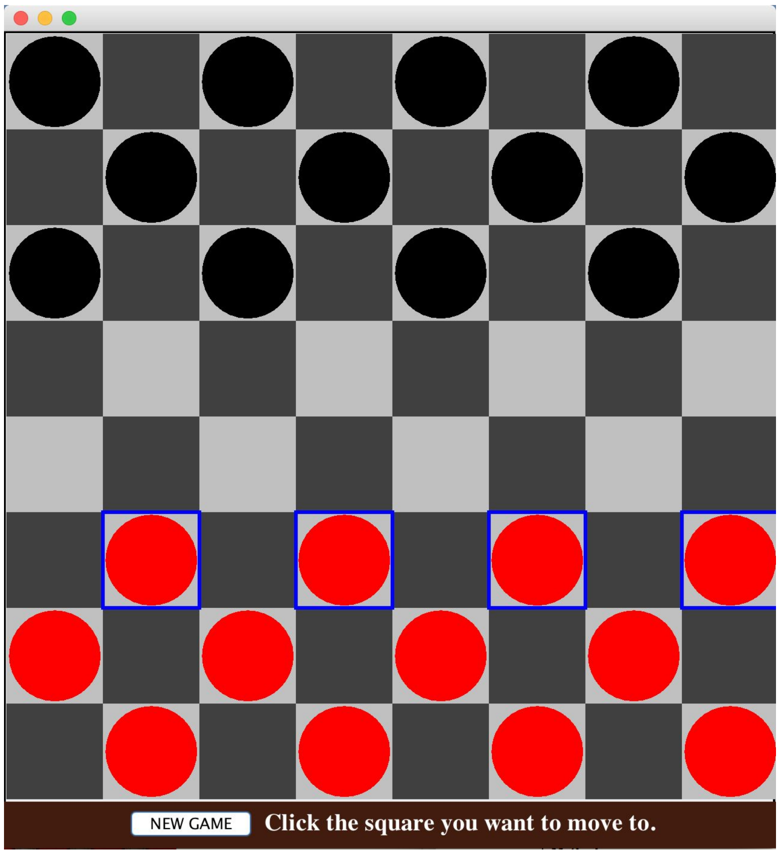
Mathew Puryear & Dulce Palacios
CS 360 – Ravikumar
December 16, 2016

Final Project: Checkers

Project Description

For our final project, we implemented the turn-based board game, Checkers. The game will be played on an eight by eight board, where there are 32 light and 32 dark squares. Each player will have their 12 checkers (Red or Black) on their side of dark colored squares. Each player will take their turn moving their pieces, only in a diagonal direction to another open dark square. A checker can only move forward and may not land on a square that already has a checker on it. When a checker reaches the last row of the board, it is "kinged" and becomes a king piece. A king piece is able to move similar to a regular checker piece, except it can move forwards and backwards. To capture an opponent's checker, one must be next to their checker and have a black square behind it in order to have a valid jump. Multiple jumps are valid. A king can be captured just like a regular checker. A player wins the game when all of their opponent's checkers have been captured or they are blocked from making any move.



Click the square you want to move to.

Classes and Methods

***** (+) = public, (-) = private, all member data are non-static unless otherwise stated***

(+) Game: This public class is responsible for managing the rules of the game and the rest of the classes.

Member Data:
- (-) JFrame
- (-) JPanel (buttonPanel, gamePanel)
- (-) Board board
- (-) Boolean gameOver
- (-) Player
- (-) Int (selectedR, selectedC)
- (-) JLabel
- (-) MoveSequence

Methods:
- (+) void Game(): Constructor initiates all data members.
- **(+) void startGame():** Initiates game by checking the type of game the user wants to play. And determines the next move of the first player.
- **(+) void clickSquare(int, int) -** Checks for valid selected square / checker
- (+) void makeMove(Move) - Handles the order and rules behind each player's move.
- **(+) void SimpleAITurn()** - Handles hypothetical mouse clicks for AI player.
- **(+) void gameOver** - Takes in string for message and handles asking user for a new game or quitting.

(+) Game() { **// set gameOver to true** **// create new instances of Board, JPanel(s), JLabel and JFrame** **// set up frame attributes** **// create button and add it to buttonPanel** **// edit button attributes** **// ad actionListener to button** **// set button to visible** **// add to buttonPanel**	**(+) startGame() {** **// Checks if game is ongoing** **// Asks user if they want to continue or not** **// If no, starts new game** **// If yes, they will continue where they left off** **// Checks whether 1, 2 or 0 users are playing**

// create label, change font and add it to buttonPanel
 	
// frame.setResizable(false) to prevent issue with sierzing.
 	
// add buttonPanel to frame bottom
// add gamePanel to frame center
// add this to frame
// add this to mouse listener
 	
// call Start Game to initialize game and mvoes
}
	// Depending on the result, it will set AI of player to red or black
	
	// Have red player move first (whether in person or in video.
	
	// Set currentPlayer to whatever the result is aka getting a new office
	
	// set gameOver to false
	
	// if currentPlayer is an AI, do first move
	
	// repaint screen

(+) clickSquare(int row, int col) {
 	
// for loop that checks that location is a valid square to click on
 	
// if it isnt, return and display text asking use to try again
 	
// for loop to make sure that intended move is valid and calls makeMove to actually move
}
	(+) makeMove(Move) {
	// make move on board
	
	//checks if move was a jump
	// returns if true so that it skips the rest
	
	// checks if there are other valid jumps
	
	// checks for ai
	
	// repaints
	}

(+) SimpleAITurn() {
// Declare and initialize move to ai's next moves
// undergo triggered mouse clicks to undergo move
}
	(+) gameOver {
	
	// displays text
	
	// makes gameover true

	// shows option to play again or quit **}**

(+) Board: This public class is responsible the creation and modification of a two-dimensional array of Pieces	
Member Data: **- (+) public static final int NUM_ROW_COL -** global number of rows and columns **- (-) Piece [][] board -** 2D array of Piece	
Methods: **- (+) Piece getPiece(int, int) -** Returns Piece at location of row and column - (+) **void setPiece(int, int, Piece)** - Takes in row, col and Piece and replaces the current element, if any, with parameter Piece **- (+) Piece [] [] getBoard() -** Returns whole Piece array that represents the board **- (+) movePiece(Move) -** Handles move of Piece in the array **- (+) canJump(int, int, int, int, int, int, Player) -** Takes in the location that the Piece will move from, the location of the piece it will jump over, and the location of the place it will jump to. And returns whether it is possible or not. **- (+) getValidJumps(int, int, Player) -** Returns the sequence of valid jumps. **- (+) setup() -** Initializes board to have the corresponding empty and piece Pieces. **- (+) getValidMoves(Player) -** Returns the sequence of valid moves.	

(+) movePiece(Move m) {
// initalize toR, toC, fromR, fromC from m
 	
// create Piece from board at that location (fromR, fromC)
// move piece to new location
// make old location empty piece
 	
// if move was a jump
// erase the jumped over piece
	(+) setup() {
	// for r in rows and c in cols
	// where r % 2 == c % 2
	// if r is less than 3
	    //piece is black
	// else if r > 4
	    // piece is red
	// else
	   // piece is empty
	// else
	   // piece is empty
	}

// if red piece makes it to the end, king them
// if black piece makes it to the end, king them
}	

(+) canJump(int, int, int,int,int,int, Player) {
// checks if move was off the board
// checks if move landed on a piece
// checks if middle piece is empty
// checks if player is red
    // checks if is king
    // checks if middle piece is red
// return true otherwise
// checks if player is black
   // checks if is king
   // checks if middle piece is black
// return true otherwise
}
	(+) getValidJumps(int, int, Player) {
	// creates arraylist of move
	// adds possible jump moves to array
	}
	
	(+) getValidMoves(Player) {
	// returns null if it is neither red or black
	
	// create arraylist of move
	
	// for r in row and c in col
	    //if player if parameter player
	    //add possible moves to arraylist
	
	// Don't get more moves unless arraylist is emtpy
	// if moves is empty, return empty move sequence
	}

(+) canMove(int, int, int, int) {
// checks if move was off the board
// checks if to-go place is empty
/ checks that if normal piece, it is going in the right direction	

(+) Piece: This public class is responsible for holding the attributes of the "checkers"
Member Data: **- String color -** Stores the color of the checker piece - **String type -** Stores the type of the checker piece
Methods:

- (+) isPlayer(Player) - Returns color that represents the piece.
- (+) isRed() - Returns true/false if the Piece is red
- (+) **isBlack() -** Returns true/false if the Piece is black
- (+) **isEmpty()** - Returns true/false if the Piece is empty
- (+) isKing() - Returns true/false if the Piece is a king
- (+) getType() - Returns the type of the piece
- (+) getColor() - Returns the color of the piece
- (+) getColorType() - Returns the color and type of piece concatenated

(+) getColorType() { return new String(color + "_" + type); }

(+) Player: This public class is responsible storing the attributes of a player.

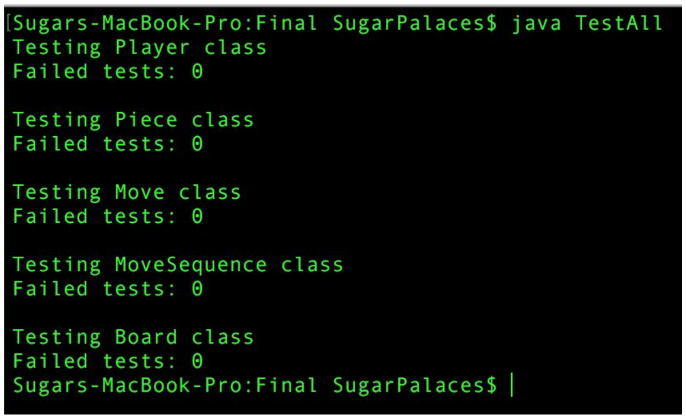
Member Data:
- (+) String color - Stores the color that represents the Player
- (+) isAI - Returns true/false depending if Player is AI

Methods:
- (+) getColor - Returns the color of the Player
- (+) isRed() - Returns true if Player is red
- (+) **isBlack()** - Returns true if Player is black
- (+) isAI() - Returns true if Player is AI
- (+) setAI(boolean) - Sets isAI boolean to whatever the parameter is

Test Results:

We have included test classes for all of our functions and have tested all of the functions within the classes.

Output:



```
[Sugars-MacBook-Pro:Final SugarPalaces$ java TestAll
Testing Player class
Failed tests: 0

Testing Piece class
Failed tests: 0

Testing Move class
Failed tests: 0

Testing MoveSequence class
Failed tests: 0

Testing Board class
Failed tests: 0
Sugars-MacBook-Pro:Final SugarPalaces$ |
```