BLOKUS

COMP1140 Assignment 2 thu09l

THE TEAM

- Jack Adamson
- Liyang Guan
- Faizan Siddiqui

THE GAME

- Between 1 and 4 players
- 4 colours each with 21 pieces, with different squares in each piece
- Aim is to get the most number of your squares on the board
- Can only place piece which touches the corners of another piece with the same colour

THE GAME IN JAVA

- Board was encoded in a string composed of blocks of 4 characters (or a . for pass)

- Each piece was represented by the four characters
  - First character tells us which of the 21 pieces
  - Second character tells us what rotation
  - Third and fourth character gives us the (x,y) coordinates

WHAT WE ALL DID

JACK

- Tile.setOnMouseReleased(event -> {...})
  - Handles encoding of a selected piece and allows it to snap on the piece
  - As piece is being hovered over the board, the type of piece, rotation and position (to the closest square) gets encoded
  - When released, the game checks if it is a legitimate move, and if it is, the encoded piece gets added to the game string, and then consequently drawn on the board
  - The piece goes to the nearest square, thus creating a snapping effect
  - If piece is an illegal move, it is returned back to the panel

- Tile.setOnMousePressed(event -> {...})
  - Selects a piece from the tile panel and allows it to be rotated 90 degrees or flipped along the y-axis, based of a double click or left click, respectively

LIYANG

- The methods in the Legit class (the improved legitimate game function)
  - This updated legitimate function more accurately check the corner contact with the same-color tile and also make sure no edge contact with the same-color tile, and after implementing this method, the overlapping pieces bug in the GUI was fixed
- The overall structure of the GUI
- The Launch Game button, that switched from the Menu scene to the Board scene
- The messagebox in the game, which told us how many players we are playing with, and who's turn it is

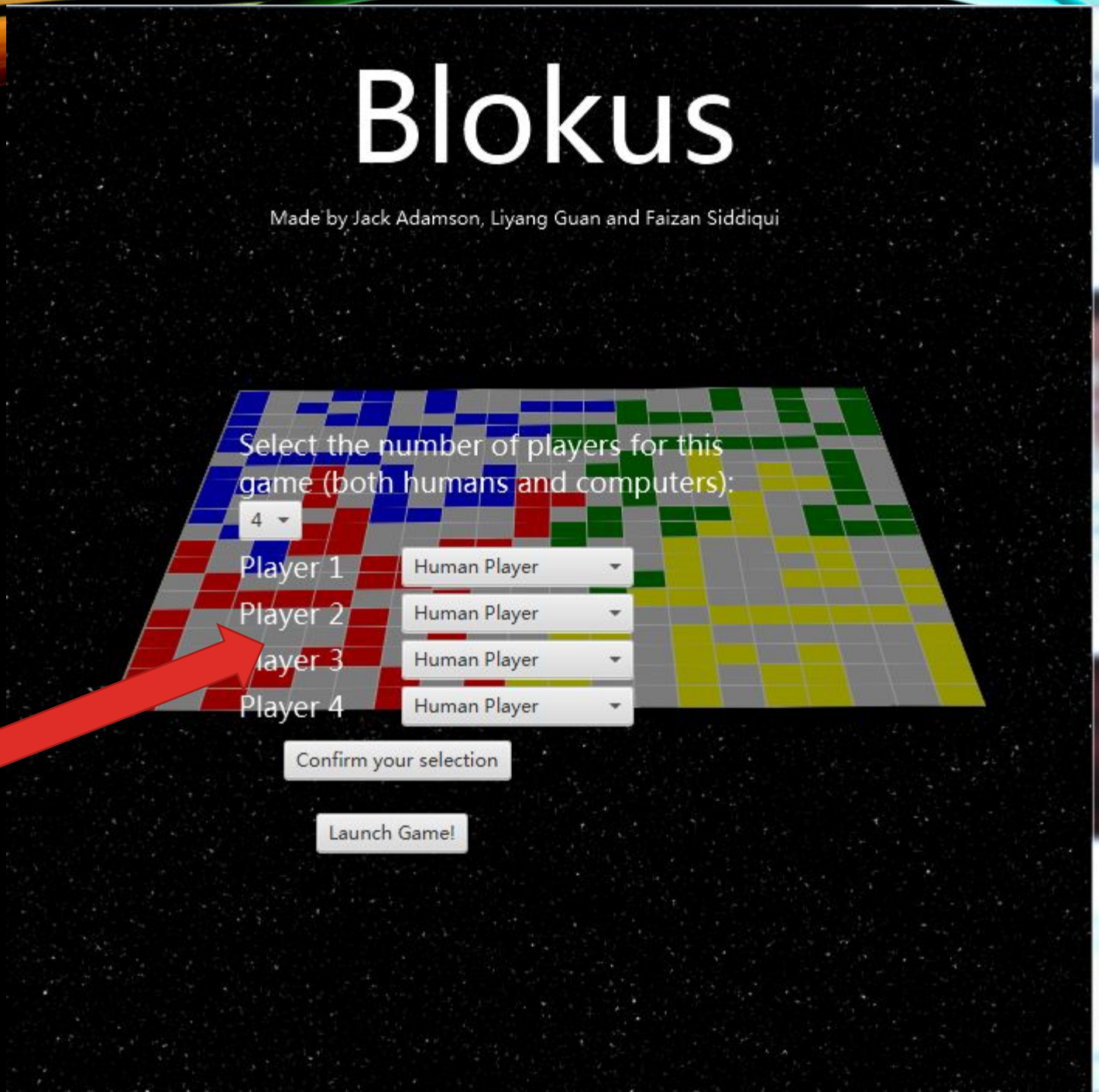
FAIZAN

- The Menu class (which made the game's menu and player selection area
  - all the choiceboxes and buttons for selecting the type of players, and number of players.
  - selectNumber.getSelectionModel().selectedItemProperty().addListener code in class Menu(), that displays the choiceboxes for different players, depending on the number of players playing.
  - confirm.setOnAction event in class Menu() which encodes the game variation and selects the number of human and computer players

- The fieldpromt textbox in class PlayBoard() and the isValidEncoding() method in class Playboard(), which allows users to input strings of encoded pieces and have it drawn on the board.

- The Scoreboard inner class in class Playboard(), which draws a scoreboard. The update score function was written by Liyang, as well as some improvements to the scoreboard.

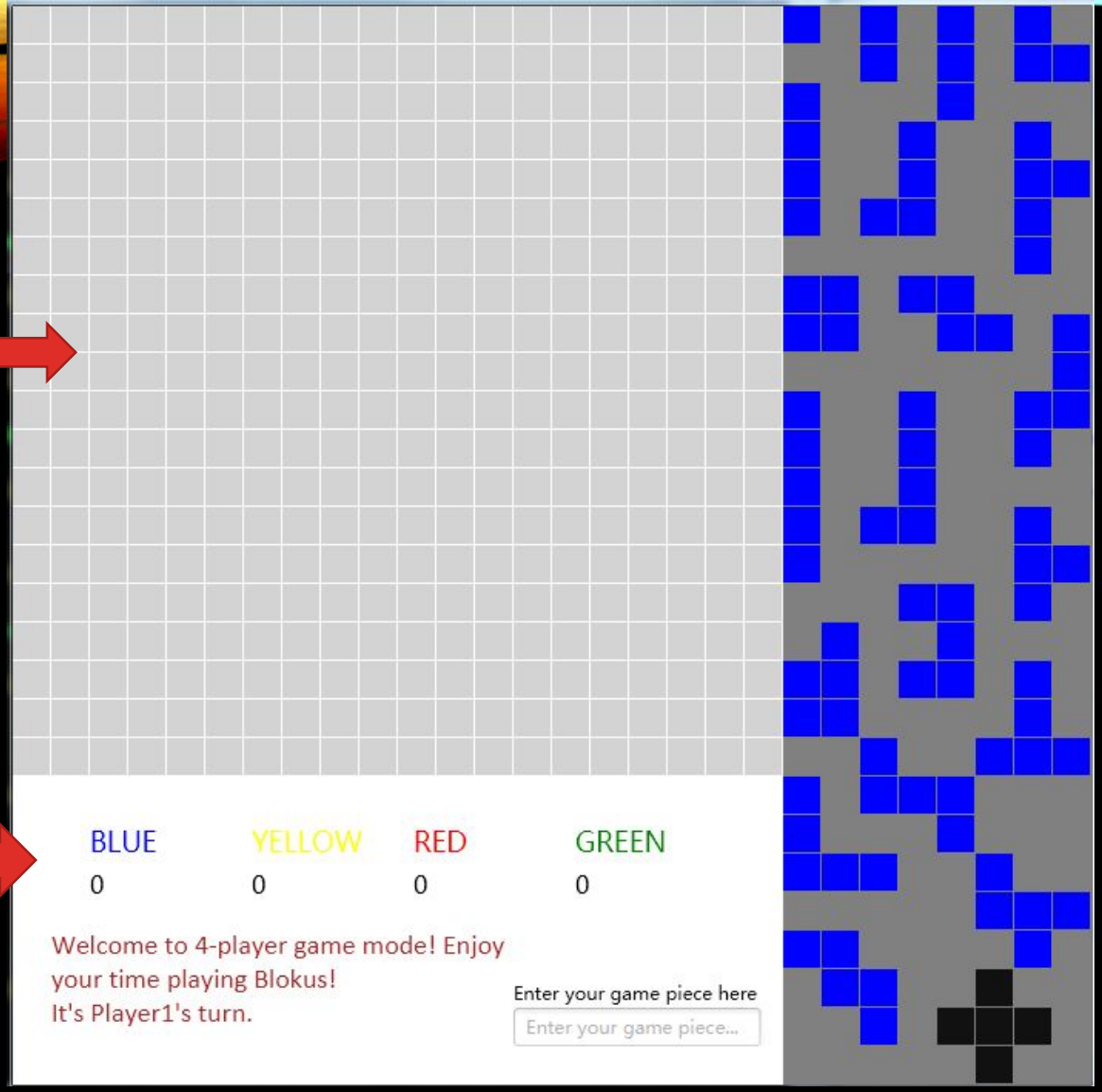
FEATURES OF THE GAME WE DESIGNED

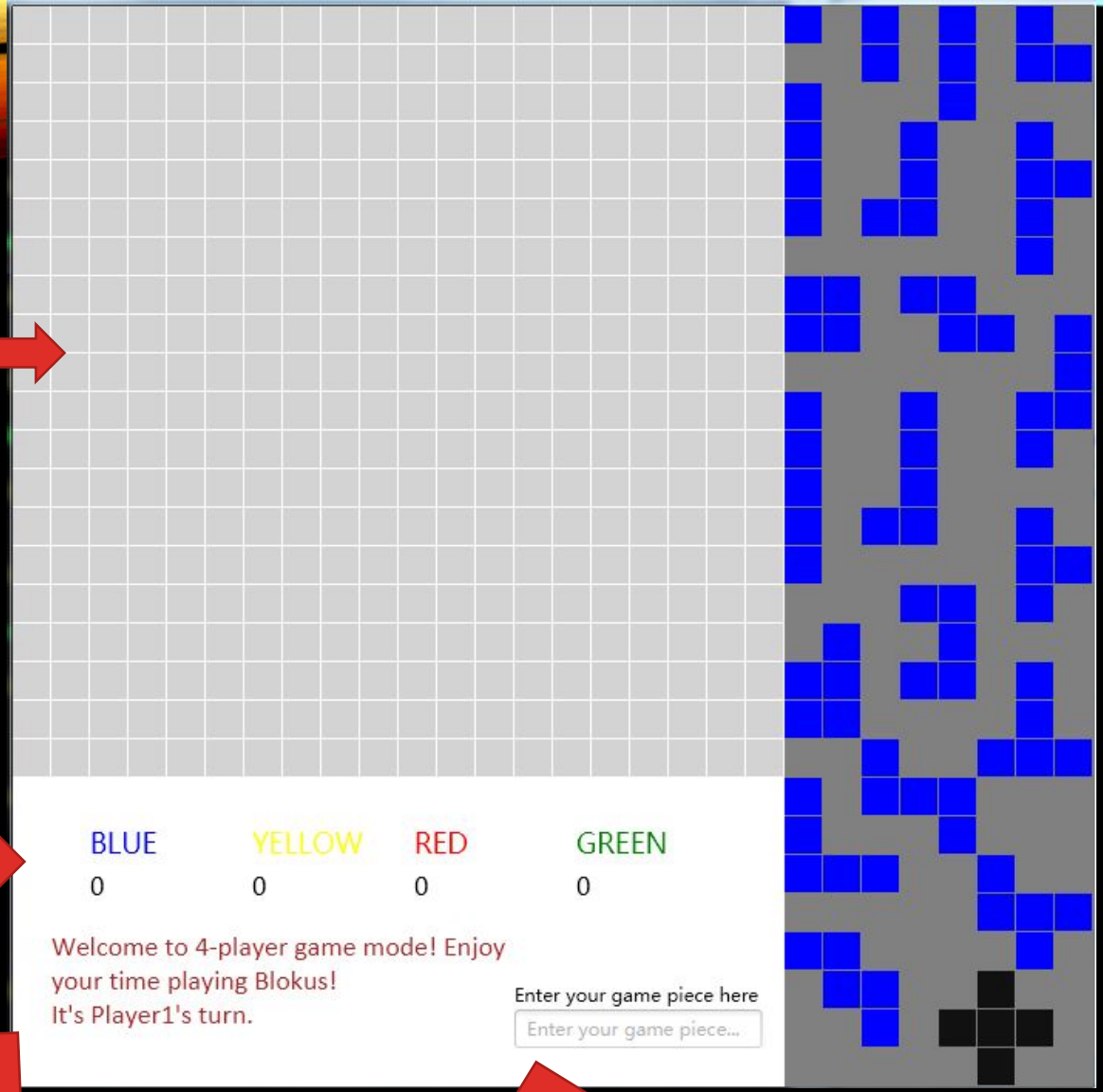


Game mode Selection

Board

Score for each color



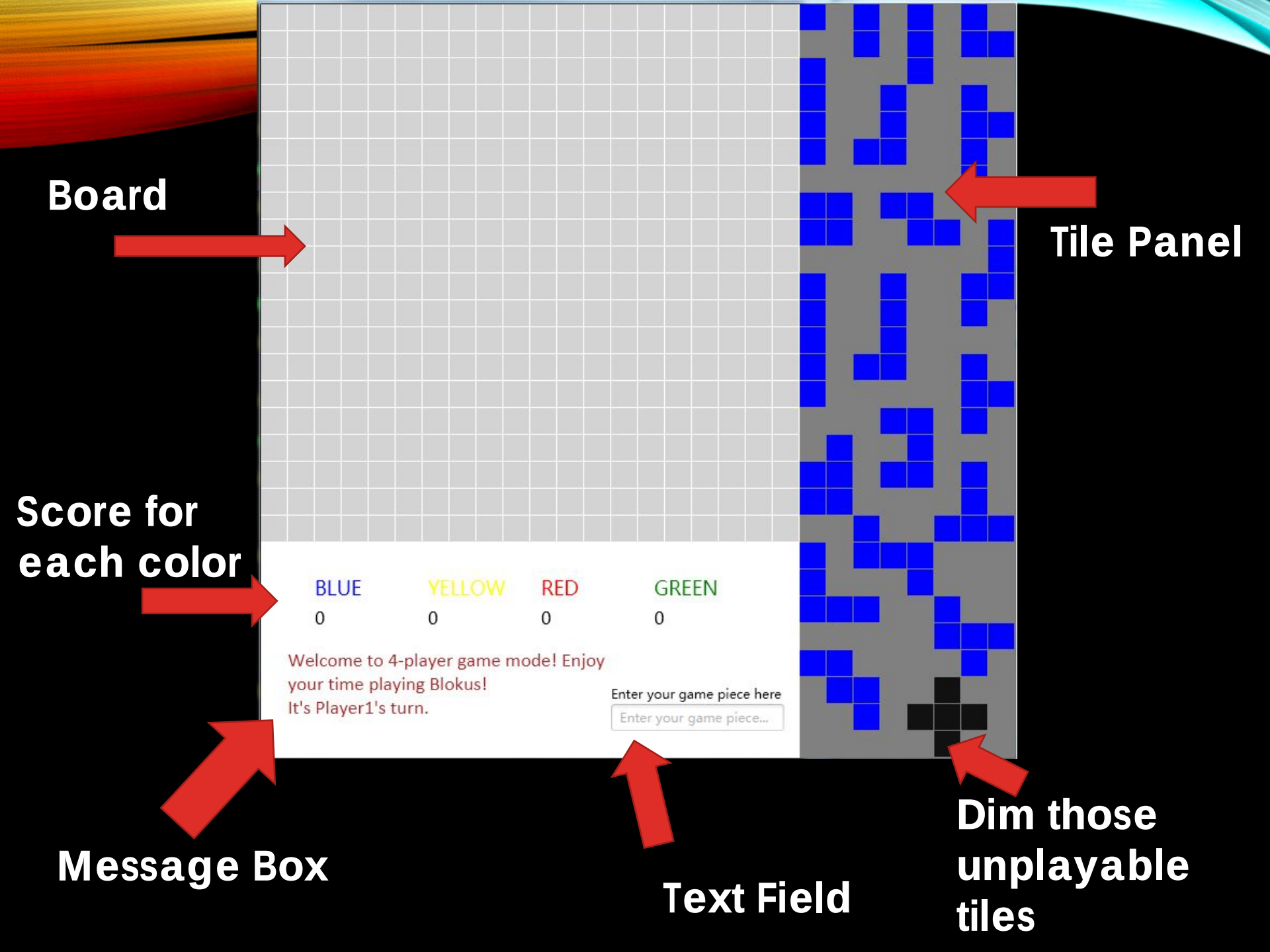


Board

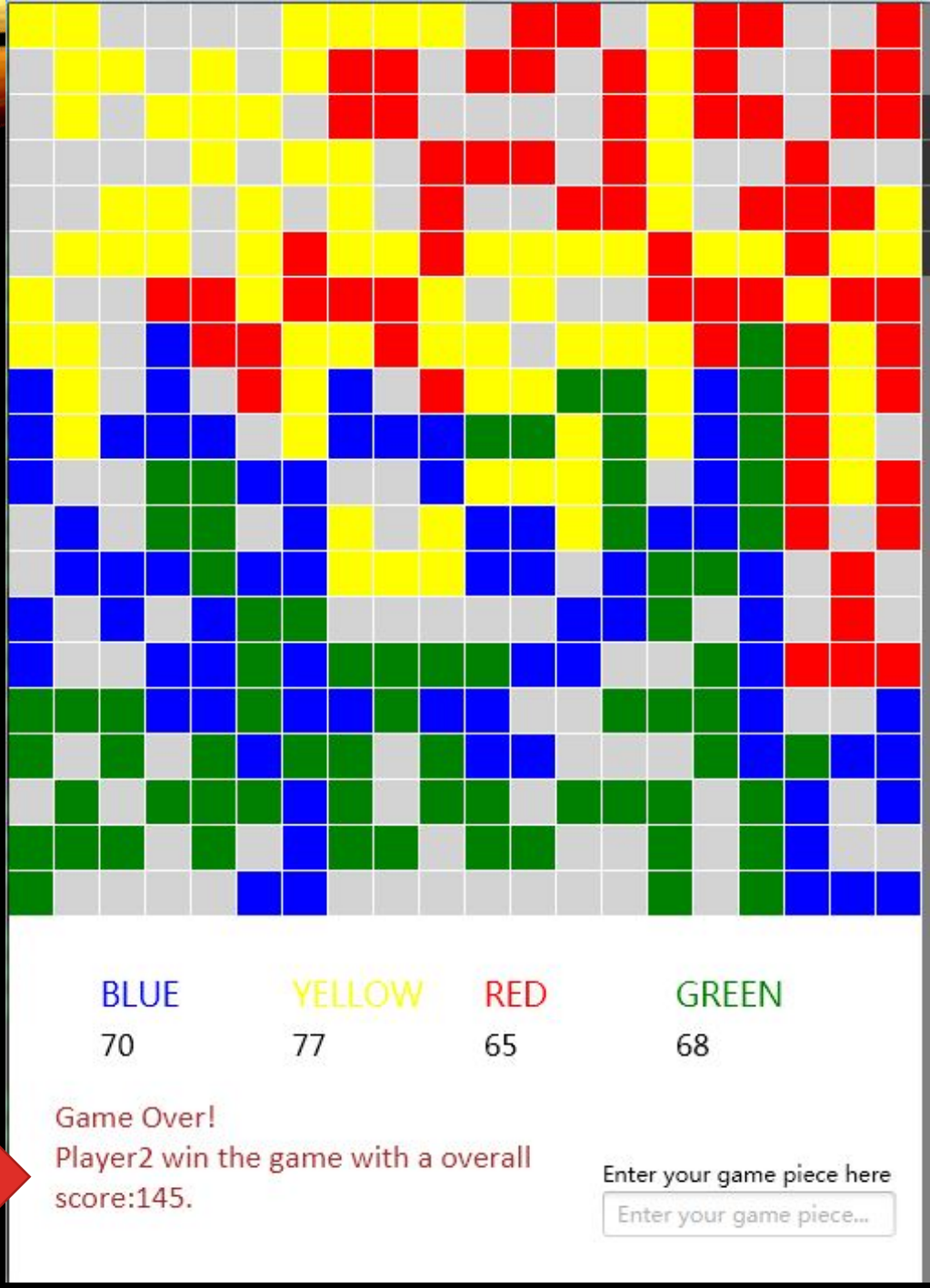
Score for each color

Message Box

Text Field

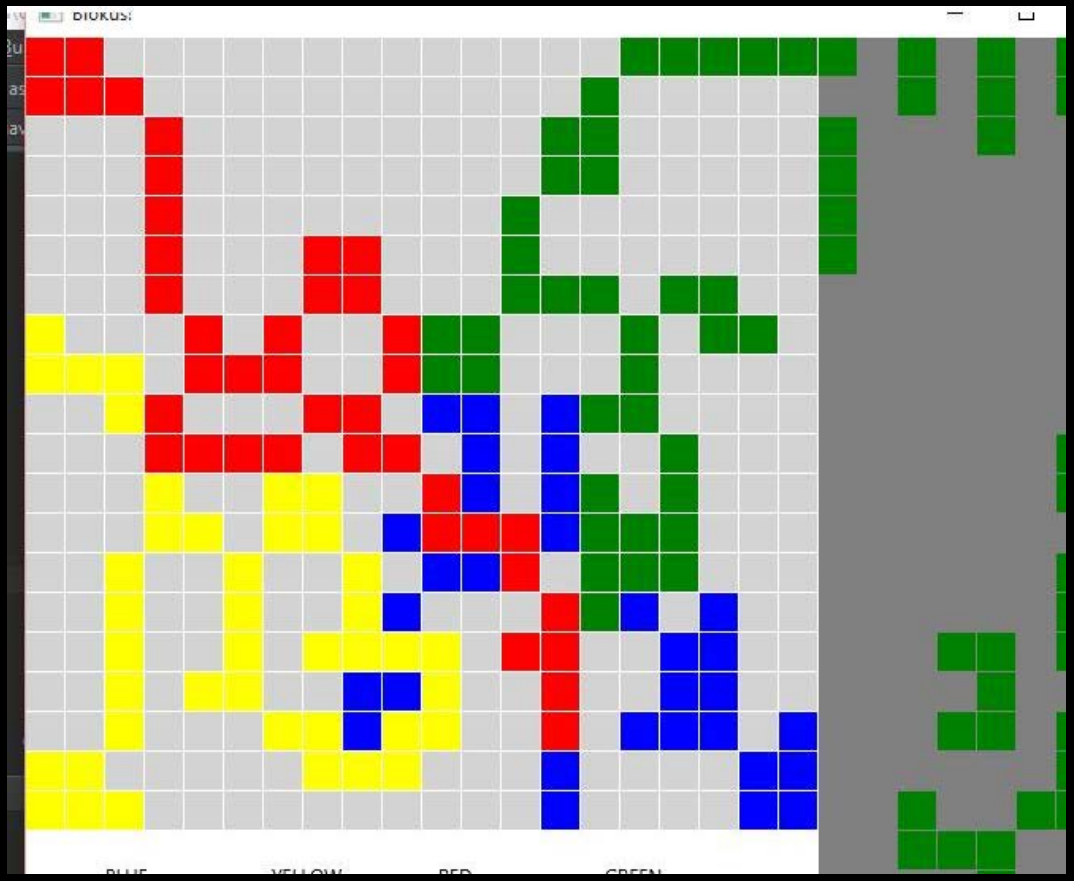

Board
Tile Panel
Score for each color
BLUE
YELLOW
RED
GREEN
0
0
0
0
Welcome to 4-player game mode! Enjoy your time playing Blokus!
It's Player1's turn.
Enter your game piece here
Enter your game piece...
Message Box
Text Field
Dim those unplayable tiles

The result for a 2-player game.



SOME PROBLEMS WE INCURRED ALONG THE WAY

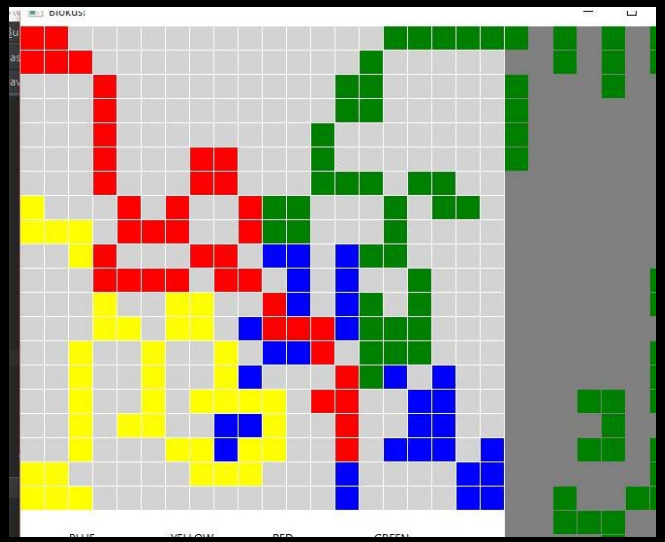
OVERLAPPING PIECES!



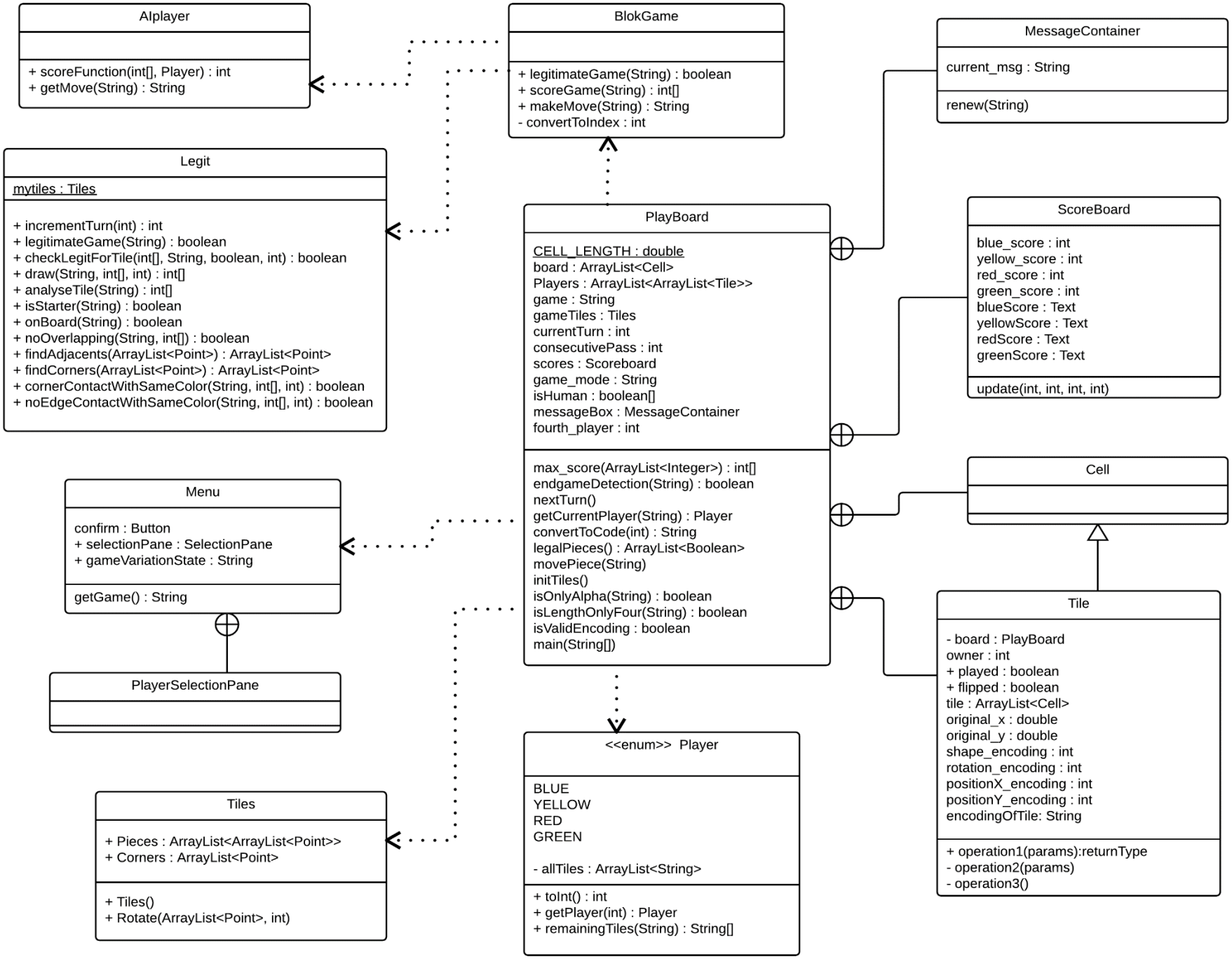
OVERLAPPING PIECES!

- The culprit: legitimateGame() was allowing certain illegal moves to be placed (despite passing the tests)

- How we fixed it: And even more pedantic legitimate game function that checks for even more cases



THE OVERALL UML DIAGRAM


AIplayer
+ scoreFunction(int[], Player) : int
+ getMove(String) : String
BlokGame
+ legitimateGame(String) : boolean
+ scoreGame(String) : int[]
+ makeMove(String) : String
- convertToIndex : int
MessageContainer
current_msg : String
renew(String)
Legit
mytiles : Tiles
+ incrementTurn(int) : int
+ legitimateGame(String) : boolean
+ checkLegitForTile(int[], String, boolean, int) : boolean
+ draw(String, int[], int) : int[]
+ analyseTile(String) : int[]
+ isStarter(String) : boolean
+ onBoard(String) : boolean
+ noOverlapping(String, int[]) : boolean
+ findAdjacents(ArrayList<Point>) : ArrayList<Point>
+ findCorners(ArrayList<Point>) : ArrayList<Point>
+ cornerContactWithSameColor(String, int[], int) : boolean
+ noEdgeContactWithSameColor(String, int[], int) : boolean
PlayBoard
CELL_LENGTH : double
board : ArrayList<Cell>
Players : ArrayList<ArrayList<Tile>>
game : String
gameTiles : Tiles
currentTurn : int
consecutivePass : int
scores : Scoreboard
game_mode : String
isHuman : boolean[]
messageBox : MessageContainer
fourth_player : int
max_score(ArrayList<Integer>) : int[]
endgameDetection(String) : boolean
nextTurn()
getCurrentPlayer(String) : Player
convertToCode(int) : String
legalPieces() : ArrayList<Boolean>
movePiece(String)
initTiles()
isOnlyAlpha(String) : boolean
isLengthOnlyFour(String) : boolean
isValidEncoding : boolean
main(String[])
ScoreBoard
blue_score : int
yellow_score : int
red_score : int
green_score : int
blueScore : Text
yellowScore : Text
redScore : Text
greenScore : Text
update(int, int, int, int)
Menu
confirm : Button
+ selectionPane : SelectionPane
+ gameVariationState : String
getGame() : String
PlayerSelectionPane
Cell
Tile
- board : PlayBoard
owner : int
+ played : boolean
+ flipped : boolean
tile : ArrayList<Cell>
original_x : double
original_y : double
shape_encoding : int
rotation_encoding : int
positionX_encoding : int
positionY_encoding : int
encodingOfTile: String
+ operation1(params):returnType
- operation2(params)
- operation3()
<<enum>> Player
BLUE
YELLOW
RED
GREEN
- allTiles : ArrayList<String>
+ toInt() : int
+ getPlayer(int) : Player
+ remainingTiles(String) : String[]
Tiles
+ Pieces : ArrayList<ArrayList<Point>>
+ Corners : ArrayList<Point>
+ Tiles()
+ Rotate(ArrayList<Point>, int)