AP AX

A Multiplayer Game Using Sockets

Rules

The rules of the game are as follows. Both players start with 12 pieces on the board; the first player's pieces are red and positioned on every second space on the top three rows; the second player's pieces are black and are on the bottom three rows. The turns alternate, with player one going first. A player has two different types of move: 'a simple move', consisting of moving a piece diagonally left or right to an unoccupied tile, and a 'jump', consisting of moving a piece to an empty space beyond an opponent's piece that is diagonally adjacent to the player's piece. Initially pieces can only move forward; however if a piece reaches the other side of the board, it becomes a king and can simple move and jump both forward and backwards. The game ends when one of the two players runs out of pieces.

Deficiencies

One deficiency is that players are unable to 'multi-jump', which is a rule in most variations of checkers. The implementation of this functionality was attempted by using the 'checkAvailableMoves' function in the board class, but was ultimately unfinished. By utilizing this function, the program could be expanded to include this feature.

In traditional checkers, the game can also end if the opponent has no legal moves, which represents another deficiency in the program. The 'checkAvailableMoves' function was also written toward this end.

Functionality and Implementation

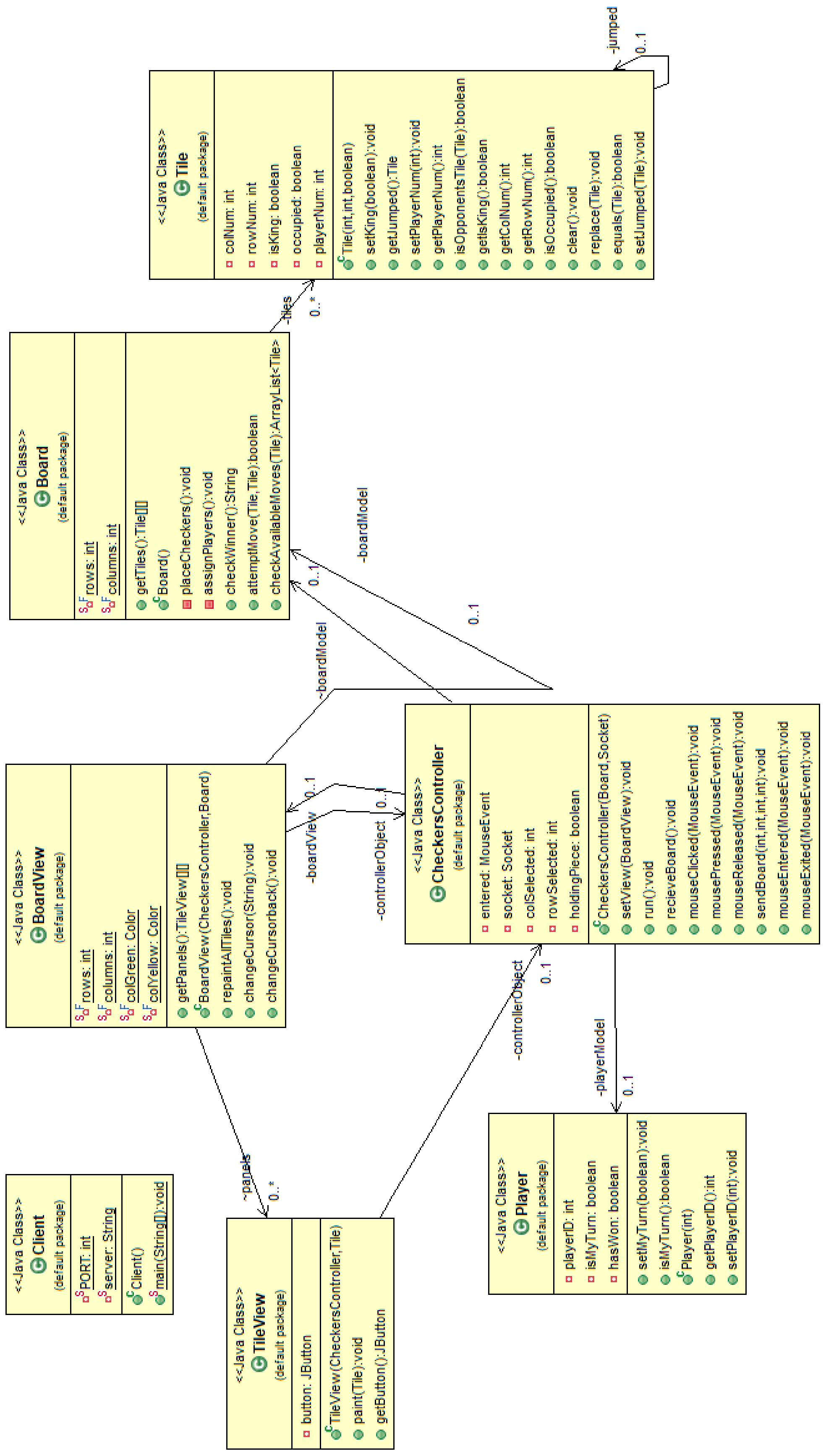
The aim was to create a GUI that was responsive, even when it is not the player's turn. The player controls pieces by dragging the piece to the selected tile. When the player is 'holding' a piece, the mouse cursor changes to the selected piece. If the player tries to move on their opponents turn or if they try and move a piece which is not theirs, they are notified.

The game functionality was implemented via two programs, a client and a server. First, the server waits for both clients to connect and then assigns them each a player number and begins the game loop. The bulk of the project is done client side, with the server only keeping track of the moves. An earlier build of the project had the server pass a serialized board object to represent the game state, but this was scrapped in favour of simply transmitting the co-ordinates of each move and updating the model client side, which saved a substantial amount of code lines. The client adheres where possible to the MVC design pattern, with classes for the model (Board, Tile, Player), the view (BoardView, TileView) and the controller (CheckersController).

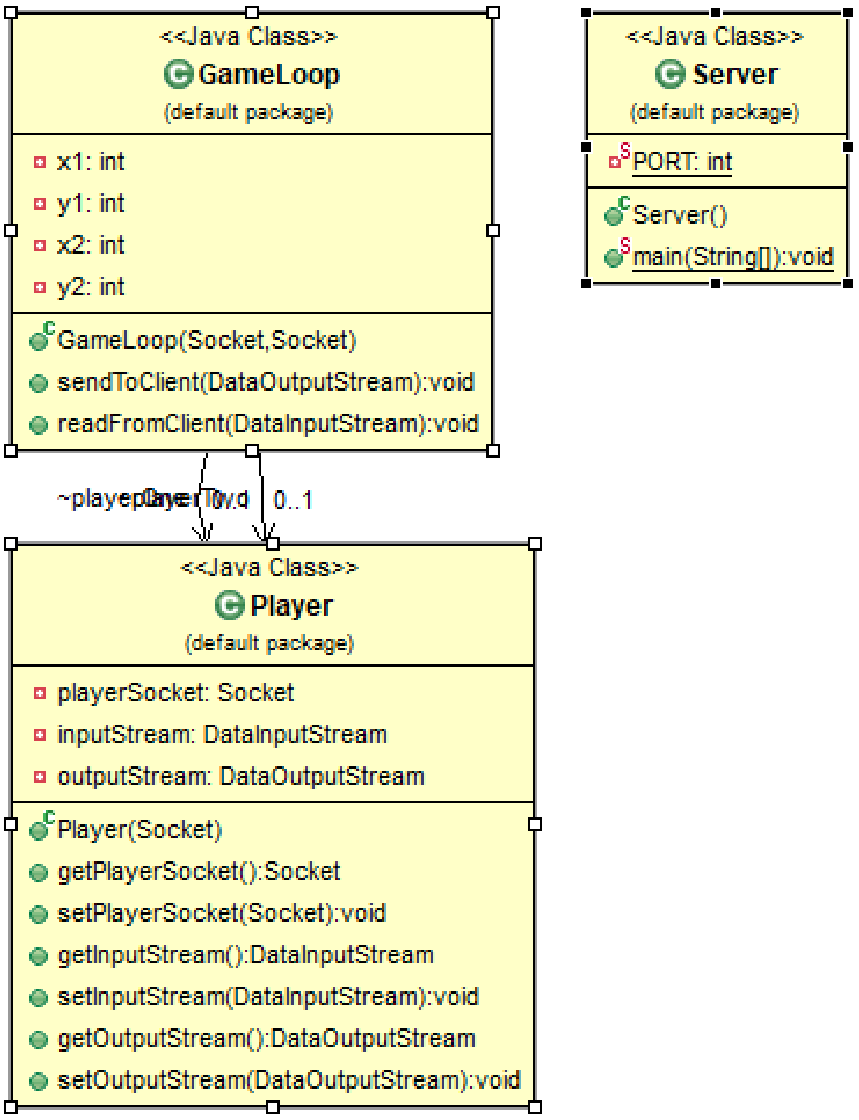
In order to run the game correctly, an images folder is supplied with the client java files.

Attached are two UML diagrams, one for the client and one for the server.

UML (Client)


<<Java Class>>
Tile
(default package)
colNum: int
rowNum: int
isKing: boolean
occupied: boolean
playerNum: int
Tile(int,int,boolean)
setKing(boolean):void
getJumped():Tile
setPlayerNum(int):void
getPlayerNum():int
isOpponentsTile(Tile):boolean
getIsKing():boolean
getColNum():int
getRowNum():int
isOccupied():boolean
clear():void
replace(Tile):void
equals(Tile):boolean
setJumped(Tile):void
-jumped
0..1
-tiles
0..*
<<Java Class>>
Board
(default package)
rows: int
columns: int
getTiles():Tile[][]
Board()
placeCheckers():void
assignPlayers():void
checkWinner():String
attemptMove(Tile,Tile):boolean
checkAvailableMoves(Tile):ArrayList<Tile>
-boardModel
0..1
0..1
~boardModel
<<Java Class>>
BoardView
(default package)
rows: int
columns: int
colGreen: Color
colYellow: Color
getPanels():TileView[][]
BoardView(CheckersController,Board)
repaintAllTiles():void
changeCursor(String):void
changeCursorback():void
0..1
-boardView
0..1
-controllerObject
<<Java Class>>
CheckersController
(default package)
entered: MouseEvent
socket: Socket
colSelected: int
rowSelected: int
holdingPiece: boolean
CheckersController(Board,Socket)
setView(BoardView):void
run():void
recieveBoard():void
mouseClicked(MouseEvent):void
mousePressed(MouseEvent):void
mouseReleased(MouseEvent):void
sendBoard(int,int,int,int):void
mouseEntered(MouseEvent):void
mouseExited(MouseEvent):void
0..1
-controllerObject
-playerModel
0..1
~panels
0..*
<<Java Class>>
Client
(default package)
PORT: int
server: String
Client()
main(String[]):void
<<Java Class>>
TileView
(default package)
button: JButton
TileView(CheckersController,Tile)
paint(Tile):void
getButton():JButton
<<Java Class>>
Player
(default package)
playerID: int
isMyTurn: boolean
hasWon: boolean
setMyTurn(boolean):void
isMyTurn():boolean
Player(int)
getPlayerID():int
setPlayerID(int):void

UML (Server)



<<Java Class>>
GameLoop
(default package)
x1: int
y1: int
x2: int
y2: int
GameLoop(Socket,Socket)
sendToClient(DataOutputStream):void
readFromClient(DataInputStream):void
<<Java Class>>
Server
(default package)
PORT: int
Server()
main(String[]):void
~playerOne
~playerTwo
0..1
0..1
<<Java Class>>
Player
(default package)
playerSocket: Socket
inputStream: DataInputStream
outputStream: DataOutputStream
Player(Socket)
getPlayerSocket():Socket
setPlayerSocket(Socket):void
getInputStream():DataInputStream
setInputStream(DataInputStream):void
getOutputStream():DataOutputStream
setOutputStream(DataOutputStream):void