2020

XADREZ ECGM MULTIPLAYER

LABORATÓRIO PROGRAMAÇÃO

DIOGO AMORIM Nº18463

DUARTE REBOUÇO Nº19150

IPVC | ECGM

Índice

Descrição do Jogo de Xadrez

O jogo de Xadrez é um jogo de estratégia em que o objetivo principal é encurralar e conquistar o Rei do adversário (come-lo).

No Xadrez existem 6 tipos de peças diferentes:

- Peão;
- Torre;
- Cavalo;
- Bispo;
- Rainha;
- Rei;

Cada uma delas tem tipos de movimentos diferentes e alguns que que so podem ser efetuados em certas condições. Por exemplo, um Peão se for o primeiro movimento que esta a fazer no jogo ele tem a opção de se poder mover 1 ou 2 casas para a frente como também. Outro exemplo seria que o Peão pode se mover 1 casa na diagonal se uma peça do adversário estiver uma casa para o lado e uma para a frente.

Objetivo

Comer a peça do Rei do adversário

Regras

O Jogo é feito em turnos, primeiro começa o Jogador com as peças brancas e só se pode mover uma peça por turno, assim que for feito passa a vez para o outro Jogador.

As peças brancas podem comer as pretas (ou vice-versa) se mover a peça branca para uma casa na qual já se encontra uma peça preta, assim essa peça preta é automaticamente comida.

O Jogo acaba quando um dos Jogadores conseguir comer o Rei do adversário, conseguindo assim a vitoria.

Peças

Peão

Se for o primeiro movimento do Peão no jogo então ele tem a opção de poder mover 1 ou 2 casas para a frente, depois disso os próximos movimentos só podem ser de 1 casa para a frente, para o Peão poder comer uma peça ele tem de o fazer na diagonal.

Caso o Peão consiga atravessar o tabuleiro todo ate o lado do adversário ele pode assim ser promovido para umas das 4 peças seguintes: Torre, Bispo, Cavalo ou Rainha.

Torre

A Torre pode se mover um número ilimitado de casas para a frente, trás, esquerda ou direita, contando que não exista nenhuma peça no caminho

Cavalo

O Cavalo pode se dizer que o tipo de movimento dele é como andar a saltar pelo tabuleiro, podendo assim saltar por cima das outras peças sem problema, o movimento que ele faz pelo tabuleiro é em “L”, sempre com o topo do “L” a posição atual dele e a ponta do “L” a casa para a qual ele pode saltar.

Bispo

O Movimento do Bispo é semelhante ao da Torre em que ele pode se mover um número ilimitado de casas desde que não tenha também nenhum obstáculo pelo caminho, mas em vez de ser nos sentidos frente, trás, esquerda, direita, o Bispo movesse nas diagonais, fazendo assim um “X”.

Rainha

A Rainha é a peça com mais liberdade de movimento no Xadrez, o movimento dela pode se considerar uma combinação da Torre e do Bispo, sendo assim a Rainha pode se movimentar livremente em todas a direções na horizontal, vertical e diagonal.

Rei

O Rei pode se móvel para qualquer casa adjacente a sua posição atual, mas tem de se ter em consideração que é a peça mais importante do jogo, sendo que se ela for perdida equivale a derrota do Jogador que a perdeu.

Jogadas Especiais

En Passant

É uma forma de captura especial que o Peão pode efetuar se um Peão adversário se encontrar numa das casas ao lado do seu Peão, podendo assim comer o do adversário e ao mesmo tempo fazer um movimento na diagonal para o lado em que se encontrava o Peão do adversário.

Roque

O Roque é uma Jogada em que o Rei e a Torre se movem em simultâneo. O movimento consiste em o Rei andar 2 casas em direção a Torre e a Torre passar para a casa adjacente do lado oposto do Rei.

Desenvolvimento e construção do Jogo

Para desenvolver um jogo de Xadrez de dois Jogadores que funcionasse entre dois computadores diferentes através da conexão de rede é obvio que ao inicio não iria ser uma tarefa fácil.

Para funcionar um dos computadores tem de agir como servidor para que assim o outro se possa conectar a ele através do IP que o servidor esta associado.

Sendo assim fez-se dois softwares diferentes, O Servidor e o Client

Diagrama de Classes do Client

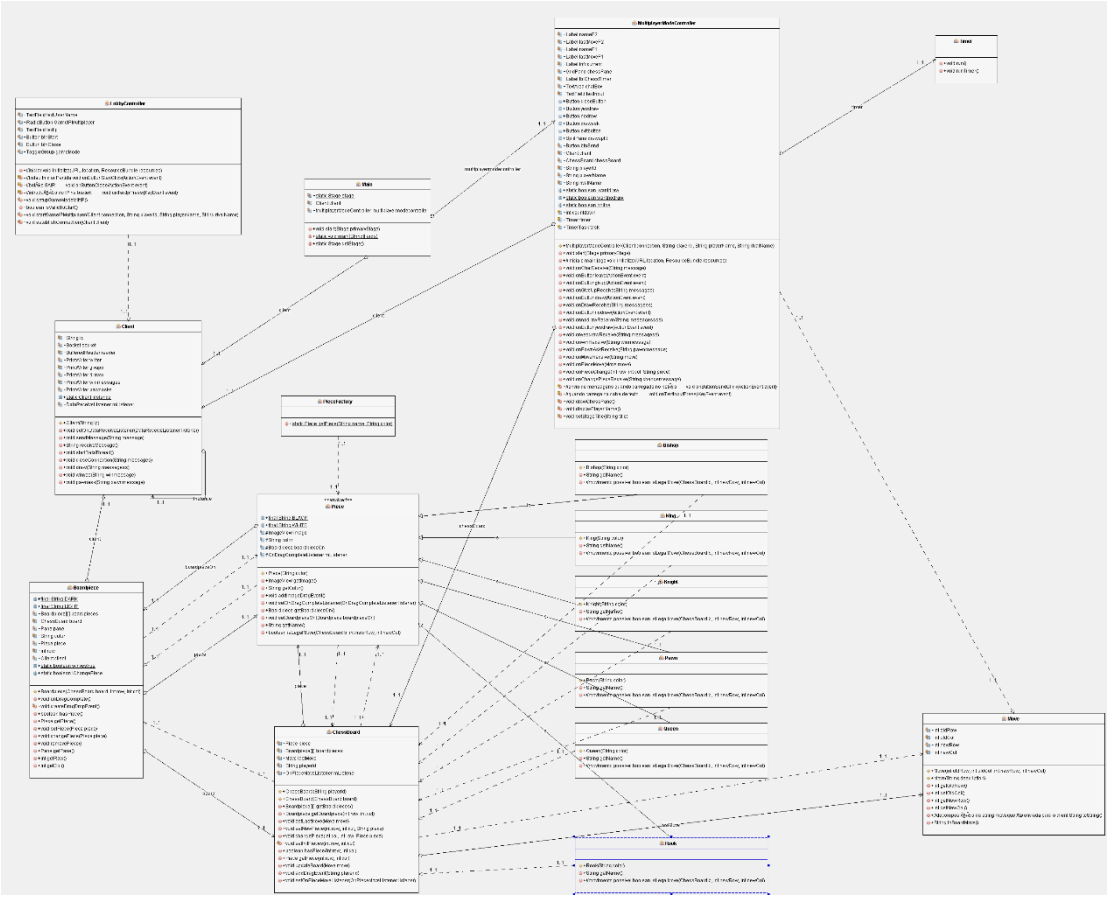
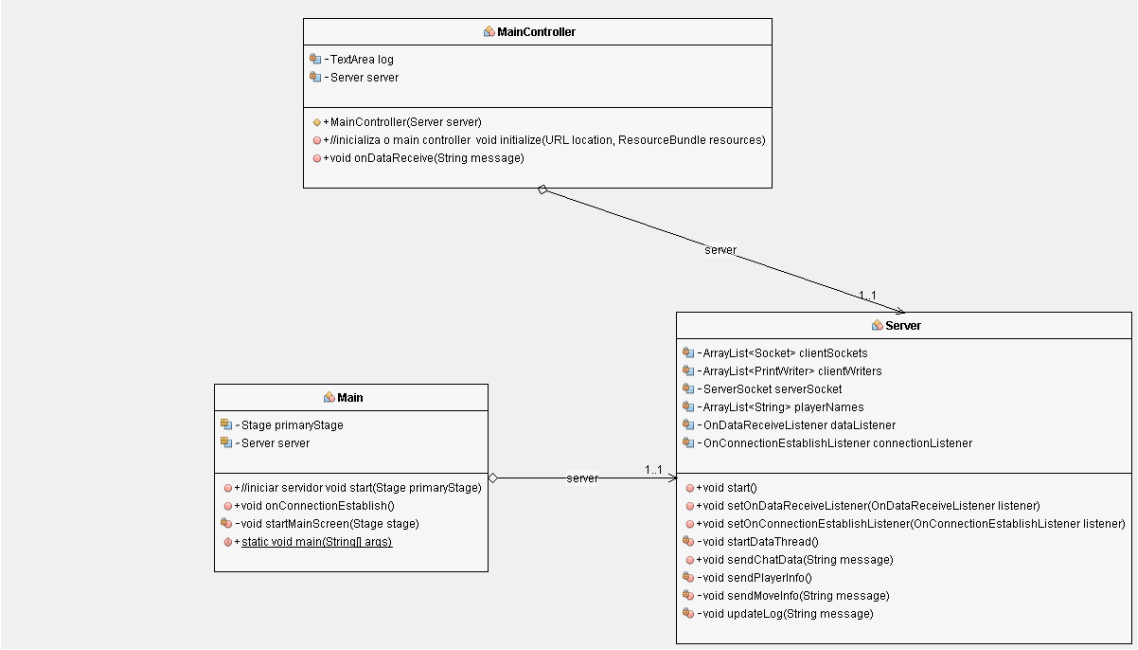


Diagrama de Classes do Servidor



Classes (Client)

Movimentos e Tabuleiro

ChestBoard

Esta Classe esta encarregada de distribuir as peças pelo tabuleiro nas suas posições corretas iniciais de ambos os lados dos jogadores.

Mas também é a classe encarregada de poder permitir ao jogador mover as peças pelo tabuleiro de uma forma *drag and drop* verificando se no local aonde o Jogador prime com o rato existe uma peça ou não.

```
//inicia as peças todas nas suas posições por tipo de peça e cor
private void setInitPieces(int row, int col) {
    String pieceName = "";
    String pieceColor = "";

    if (row == 1) { //linha todos peões pretos
        pieceName = "pawn";
        pieceColor = Piece.BLACK;
        boardpieces[row][col].setPiece(PieceFactory.getPiece(pieceName, pieceColor));
    } else if (row == 6) {//linha todos os peãos brancos
        pieceName = "pawn";
        pieceColor = Piece.WHITE;
        boardpieces[row][col].setPiece(PieceFactory.getPiece(pieceName, pieceColor));
    } else if (row == 0 || row == 7) { //caso as linhas sejas 7 ou 0
        if (row == 0)   pieceColor = Piece.BLACK; //linha 0 só brancas
        if (row == 7)   pieceColor = Piece.WHITE;// linha 7 só pretas

        switch (col) {//colunas com as seguintes peças após já terem as linhas defenidas
            case 0://2 torres na coluna 0
            case 7://2 torres na coluna 7
                pieceName = "rook";
                break;
            case 1://2 cavalos na coluna 1
            case 6://2 cavalos na coluna 6
                pieceName = "knight";
                break;
            case 2://2 bispos na coluna 2
            case 5://2 bispos na coluna 5
                pieceName = "bishop";
                break;
            case 3://2 rainhas na coluna 3
                pieceName = "queen";
                break;
            case 4:// 2 reis na coluna 4
                pieceName = "king";
                break;
        }

        boardpieces[row][col].setPiece(PieceFactory.getPiece(pieceName, pieceColor));
    }
}
  //atualização do tabuleiro
  public void updateBoard(Move move) {
      int oldRow = move.getOldRow();//antiga linha
      int oldCol = move.getOldCol();//antiga coluna
      int newRow = move.getNewRow();//nova  linha
      int newCol = move.getNewCol();//nova coluna

      Piece piece = getPiece(oldRow, oldCol); //busca a peça na posição antiga
      getBoardpiece(oldRow, oldCol).removePiece();//elimina a peça antiga no quadrado
      if (hasPiece(newRow, newCol)) {// se caso no monimento tiver uma peça
          getBoardpiece(newRow, newCol).removePiece();// remove a peça
      }
      getBoardpiece(newRow, newCol).setPiece(piece);//alteral o valor da variavel e indica a nova posição da
  }

  //Adicionar evento de arrasto para peças
  public void addDragEvent(String playerId) {
      for (int row = 0; row < 8; row++) { //verificação linhas
          for (int col = 0; col < 8; col++) { //verificação colunas
              if (hasPiece(row, col)) {// caso exista peça
                  Piece piece = getPiece(row, col); //seleção da peça
                  //ID do jogador - 1 para o lado branco e 2 para o lado preto
                  if ((playerId.equals("1") && piece.getColor().equals(Piece.WHITE))
                          || (playerId.equals("2") && piece.getColor().equals(Piece.BLACK))) {
                      piece.addImageDragEvent(); //Imagem Arrasto
                  }
              }
          }
      }
```

Move

Regista a localização antiga e nova da peça que foi movimentada.

```
public class Move {
    private int oldRow;
    private int oldCol;
    private int newRow;
    private int newCol;
    //movimento contem antiga linha, antiga coluna, nova linha, nova coluna
    public Move(int oldRow, int oldCol, int newRow, int newCol) {
        this.oldRow = oldRow;
        this.oldCol = oldCol;
        this.newRow = newRow;
        this.newCol = newCol;
    }
    //move sting description que contem oldRow_oldCol_newRow_newCol
    public Move(String description) {
        String[] s = description.split("_");
        this.oldRow = Integer.parseInt(s[0]);
        this.oldCol = Integer.parseInt(s[1]);
        this.newRow = Integer.parseInt(s[2]);
        this.newCol = Integer.parseInt(s[3]);
    }
    // retorna o valor da variavel linha antiga
    public int getOldRow() {
        return oldRow;
    }
    // retorna o valor da variavel coluna antiga
    public int getOldCol() {
        return oldCol;
    }
    // retorna o valor da variavel linha nova
    public int getNewRow() {
        return newRow;
    }
    // retorna o valor da variavel coluna nova
    public int getNewCol() {
        return newCol;
    }


    @Override
    //decomposição da string moveque é enviada para o client
    public String toString() {
        return (Integer.toString(oldRow) + "_"
                + Integer.toString(oldCol) + "_"
                + Integer.toString(newRow) + "_"
                + Integer.toString(newCol));
    }
    //string historico do movimento das peças
    public String toBoardMove() {
        String[] r = {"8", "7", "6", "5", "4", "3", "2", "1"}; //linha
        String[] c = {"A", "B", "C", "D", "E", "F", "G", "H"}; //lcoluna
        return (c[oldCol] + r[oldRow] + " -> " + c[newCol] + r[newRow]);//envio
    }
}
```

BoardPiece

Esta classe esta encarregada de gerenciar a informação de cada quadrado do tabuleiro, como por exemplo se tem alguma peça ou não e também esta encarregada de actualizar os quadrados de como quando uma peça é comida ou se move para um quadrado diferente.

```
public class Boardpiece implements Piece.OnDragCompleteListener {
    public static final String DARK = "dark";
    public static final String LIGHT = "light";

    private Boardpiece[][] boardpieces ;
    private ChessBoard board;
    private Pane pane;
    private String color;
    private Piece piece;
    private int row, col;
    private Client client;
    public static boolean winyestrue=false;
    public static boolean pawnaskmessage=false;
    public static boolean iChangePiece =false;
    public static boolean iRemovePiece;
    //defenir tabuleiro
    public Boardpiece(ChessBoard board, int row, int col) {
        this.board = board;
        this.row = row; //Linha
        this.col = col; //coluna
        this.pane = new Pane();
        if ((row % 2 == 0 && col % 2 == 1) || (row % 2 == 1 && col % 2 == 0)) {
            this.color = DARK;
            this.pane.setStyle("-fx-background-color: #1389C8"); //azul fundo tabuleiro escu
        } else {
            this.color = LIGHT;
            this.pane.setStyle("-fx-background-color: #F37723"); //laranja fundo tabuleiro cl
        }

        createDragDropEvent(); //criação do evento de Drag and Drop de peças
    }

    @Override
    public void onDragComplete() {
        //quando o evento de Drag andDrop está Completo este remove a pecça da posição anteri
        removePiece();
    }


    //Drag and Drop EVENT
    private void createDragDropEvent() {
        this.pane.setOnDragOver(new EventHandler<DragEvent>() {
            @Override
            public void handle(DragEvent event) {

                if (event.getGestureSource() != pane &&
                        event.getDragboard().hasString()) {
                    event.acceptTransferModes(TransferMode.MOVE);
                }

                event.consume();
            }
        });
```

```
        this.pane.setOnDragEntered(new EventHandler<DragEvent>() {
            @Override
            public void handle(DragEvent event) {
                  Dragboard db = event.getDragboard();
                  boolean success = false;
                //string base de dados com a info das peças linha_coluna
                if(db.hasString()) {
                    String[] pieceInfo = db.getString().split("_");
                    int oldRow = Integer.parseInt(pieceInfo[2]);
                    int oldCol = Integer.parseInt(pieceInfo[3]);
                    Piece newPiece = PieceFactory.getPiece(pieceInfo[1], pieceInfo[0]);
                    newPiece.setBoardpieceOn(board.getBoardpiece(oldRow, oldCol));
                    //caso o moviemnto seja legas
                    if (newPiece.isLegalMove(board, row, col)) {

                        success = true; //sucesso
                    }

                }


                //caso possa fazer drop da peça aparecerá
                if (success) {
                    pane.setStyle("-fx-background-color: #669438"); // verde para sucesso do Drop
                }else{
                    pane.setStyle("-fx-background-color: #b50b0b"); //Vermelho para não insucesso do
                }


                event.consume();
            }
        });

        //Apos o Drag concluido
        this.pane.setOnDragExited(new EventHandler<DragEvent>() {
            @Override
            //as cores voltam ao normal após o drag concluido
            public void handle(DragEvent event) {
                if (color.equals(DARK)) {
                    pane.setStyle("-fx-background-color: #1389C8"); //azul fundo tabuleiro escuro
                } else {
                    pane.setStyle("-fx-background-color: #F37723"); //laranja fundo tabuleiro claro
                }

                event.consume();
            }
        });
        //Drag Dropped concluido
        this.pane.setOnDragDropped(new EventHandler<DragEvent>() {
            @Override
            public void handle(DragEvent event) {
                Dragboard db = event.getDragboard();
                boolean success = false;
                 //string base de dados com a info das peças linha_coluna
                if(db.hasString()) {

                    String[] pieceInfo = db.getString().split("_");
                    int oldRow = Integer.parseInt(pieceInfo[2]);
                    int oldCol = Integer.parseInt(pieceInfo[3]);
```

```
                Piece newPiece = PieceFactory.getPiece(pieceInfo[1], pieceInfo[0]); //string que contem a info da peça nova coluna, n
                newPiece.setBoardpieceOn(board.getBoardpiece(oldRow, oldCol));// nova peça que altera o valor da peça antiga no no lo
                 //caso o moviemnto seja legas
                if (newPiece.isLegalMove(board, row, col)) {

                    //caso hajo uma peça para ser comida no movimento legal
                    if (hasPiece()) {

                        // e essa peça for um rei
                        if(piece.getName()=="king"){//Xeque-Mate
                        winyestrue=true;//serve para não imprimir para o jogar que ganhou a mensagem a baixo
                        Client.Instance.winyes("Adversário ganhou o jogo!"); //envia uma mensagem para o client com a seguinte inform
                        Alert alert = new Alert(AlertType.CONFIRMATION);
                        alert.setTitle("Xeque-mate!");//alert pop up a avisar o vencedor
                        alert.setHeaderText("Parabéns ganhas-te o jogo conseguiste comer o Rei do teu adversário!");
                        alert.setContentText("Esperemos que voltes a jogar...");
                        ButtonType buttonTypeOne = new ButtonType("Sair");
                        ButtonType buttonTypeTwo = new ButtonType("Ok");
                        alert.getButtonTypes().setAll(buttonTypeOne, buttonTypeTwo);

                        Optional<ButtonType> result = alert.showAndWait();

                        if (result.get() == buttonTypeOne){

                            System.exit(0);        // quando o jogador carrega em Sair a aplicação dá shut down
                        }else {
                            System.exit(0);

                        }

                        }  else{
                            removePiece();// caso a peça não seja um rei remove a peça normalmente

                        }}

                    if(Piece.onPassant==true){ //caso se verifique o enpassat
                    board.getBoardpiece(Piece.rowToEat, Piece.colToEat).removePiece();  //ele remove a peça no local
                    iRemovePiece =true; // troca de informações no client para a remoção da peça

                    board.setRemovePiecePawn(Piece.rowToEat, Piece.colToEat); //envia a informação de eliminara a peça para o outro j
                    board.setLastMove(new Move(oldRow, oldCol, row, col)); // ultimo movimento (antigalinha, antigacolua, linha, colu
                    success = true; //fim da jogada

                    Piece.onPassant = false;
                    }


                    newPiece.addImageDragEvent(); // nova peça dropada com uma nova imagem
                    setPiece(newPiece); // nova peça

                    board.setLastMove(new Move(oldRow, oldCol, row, col)); // ultimo movimento (antigalinha, antigacolua, linha, colu
                    success = true;// sucesso no drop

                }

            // caso os peão consigam chegar ao outro lado do tabuleiro row= 0 ou row=7 alert change Piece -> promoção de peão BRANCAS
            if(  getRow()==0 && piece.getName().equals("pawn") && piece.getColor().equals(Piece.WHITE)){
```

```
                Alert alert = new Alert(AlertType.CONFIRMATION);
                alert.setTitle("PROMOÇÃO DO PEÃO A OUTRA PEÇA...");
                alert.setHeaderText("Carregue no botão com a peça que deseja substituir...");
                alert.setContentText("Escolha a sua peça!");

                ButtonType buttonTypeOne = new ButtonType("Bispo");
                ButtonType buttonTypeTwo = new ButtonType("Torre");
                ButtonType buttonTypeThree = new ButtonType("Rainha");
                ButtonType buttonTypeFour = new ButtonType("Cavalo");
                pawnaskmessage=true;//serve para não imprimir para o jogar que não trocou a peça a mensagem a baixo
                Client.Instance.pawnask("Troca peão por outra peça."); //envia mensagem para o adversario

                alert.getButtonTypes().setAll(buttonTypeOne, buttonTypeTwo, buttonTypeThree, buttonTypeFour);

                Optional<ButtonType> result = alert.showAndWait();
                if (result.get() == buttonTypeOne){

                    removePiece(); //remove o peao
                    Piece newPieces = PieceFactory.getPiece("bishop", Piece.WHITE); // cria uma nova peça
                    newPieces.setBoardpieceOn(board.getBoardpiece(oldRow, oldCol));// nova peça que altera o valor da pe
                    newPieces.addImageDragEvent();// nova peça dropada com uma nova imagem

                    //mudança da peça para o adversário
                    iChangePiece=true; //mudança da peça
                    board.setNewPiece(row, col, "bishop#"+Piece.WHITE); // nova peça string (linha,coluna,peça,cor)
                    board.setLastMove(new Move(oldRow, oldCol, row, col));// ultimo movimento (antigalinha, antigacolua,
                    newPieces.setBoardpieceOn(board.getBoardpiece(oldRow, oldCol));
                    success = true; //sucesso na conclusao da jogada
                    setPiece(newPieces); // nova peça

                } else if (result.get() == buttonTypeTwo) {

                    removePiece();
                    Piece newPieces = PieceFactory.getPiece("rook", Piece.WHITE);
                    newPieces.setBoardpieceOn(board.getBoardpiece(oldRow, oldCol));
                    newPieces.addImageDragEvent();

                    iChangePiece=true;
                    board.setNewPiece(row, col, "rook#"+Piece.WHITE);
                    board.setLastMove(new Move(oldRow, oldCol, row, col));
                    success = true;
                    setPiece(newPieces);
                } else if (result.get() == buttonTypeThree) {

                    removePiece();
                    Piece newPieces = PieceFactory.getPiece("queen", Piece.WHITE);
                    newPieces.setBoardpieceOn(board.getBoardpiece(oldRow, oldCol));
                    newPieces.addImageDragEvent();

                    iChangePiece=true;
                    board.setNewPiece(row, col, "queen#"+Piece.WHITE);
                    board.setLastMove(new Move(oldRow, oldCol, row, col));
                    success = true;
                    setPiece(newPieces);
                } else if (result.get() == buttonTypeFour){

                    removePiece();
                    Piece newPieces = PieceFactory.getPiece("knight", Piece.WHITE);
                    newPieces.setBoardpieceOn(board.getBoardpiece(oldRow, oldCol));
```

```
                newPieces.addImageDragEvent();

                iChangePiece=true;
                board.setNewPiece(row, col, "knight#"+Piece.WHITE);
                board.setLastMove(new Move(oldRow, oldCol, row, col));
                success = true;
                setPiece(newPieces);
            }
            // caso os peão consigam chegar ao outro lado do tabuleiro row= 0 ou row=7 alert change Piece -> promoção d
           }else if(  getRow()==7 && piece.getName().equals("pawn") && piece.getColor().equals(Piece.BLACK)){

            Alert alert = new Alert(AlertType.CONFIRMATION);
            alert.setTitle("PROMOÇÃO DO PEÃO A OUTRA PEÇA...");
            alert.setHeaderText("Carregue no botão com a peça que deseja substituir...");
            alert.setContentText("Escolha a sua peça!");

            ButtonType buttonTypeOne = new ButtonType("Bispo");
            ButtonType buttonTypeTwo = new ButtonType("Torre");
            ButtonType buttonTypeThree = new ButtonType("Rainha");
            ButtonType buttonTypeFour = new ButtonType("Cavalo");
            pawnaskmessage=true;//serve para não imprimir para o jogar que não trocou a peça a mensagem a baixo
            Client.Instance.pawnask("Troca peão por outra peça.");

            alert.getButtonTypes().setAll(buttonTypeOne, buttonTypeTwo, buttonTypeThree, buttonTypeFour);

            Optional<ButtonType> result = alert.showAndWait();
            if (result.get() == buttonTypeOne){

               removePiece();
               Piece newPieces = PieceFactory.getPiece("bishop", Piece.BLACK);
               newPieces.setBoardpieceOn(board.getBoardpiece(oldRow, oldCol));
               newPieces.addImageDragEvent();

               iChangePiece=true;
               board.setNewPiece(row, col, "bishop#"+Piece.BLACK);
               board.setLastMove(new Move(oldRow, oldCol, row, col));
               success = true;
              setPiece(newPieces);

            } else if (result.get() == buttonTypeTwo) {

                removePiece();
                Piece newPieces = PieceFactory.getPiece("rook", Piece.BLACK);
                newPieces.setBoardpieceOn(board.getBoardpiece(oldRow, oldCol));
                newPieces.addImageDragEvent();

                iChangePiece=true;
                board.setNewPiece(row, col, "rook#"+Piece.BLACK);
                board.setLastMove(new Move(oldRow, oldCol, row, col));
                success = true;
                setPiece(newPieces);
            } else if (result.get() == buttonTypeThree) {

                removePiece();
                Piece newPieces = PieceFactory.getPiece("queen", Piece.BLACK);
                newPieces.setBoardpieceOn(board.getBoardpiece(oldRow, oldCol));
                newPieces.addImageDragEvent();
```

```
                    iChangePiece=true;
                    board.setNewPiece(row, col, "queen#"+Piece.BLACK);
                    board.setLastMove(new Move(oldRow, oldCol, row, col));
                    success = true;
                    setPiece(newPieces);
                } else if (result.get() == buttonTypeFour){

                    removePiece();
                    Piece newPieces = PieceFactory.getPiece("knight", Piece.BLACK);
                    newPieces.setBoardpieceOn(board.getBoardpiece(oldRow, oldCol));
                    newPieces.addImageDragEvent();

                    iChangePiece=true;
                    board.setNewPiece(row, col, "knight#"+Piece.BLACK);
                    board.setLastMove(new Move(oldRow, oldCol, row, col));
                    success = true;
                    setPiece(newPieces);
                }

                    }


            }
            //EVENTO DROP CONCLUIDO COM SUCESSO APOS TODAS AS VERIFICAÇÕES
            event.setDropCompleted(success);
            event.consume();
        }
    });
}

//função que retorna se existe alguma peça numa determina casa
public boolean hasPiece() {

    return (this.piece != null);
}


public boolean hasPiecePawnBlack() { //VERIFICA QUE A PEÇA É UM PEÃO PRETO
    if (piece.getName().equals("pawn") && piece.getColor().equals(Piece.BLACK)) {
        return (this.piece != null);
    }else{
       return (this.piece == null);
    }}

public boolean hasPiecePawnWhite() { //VERIFICA QUE A PEÇA É UM PEÃO BRANCA
    if (piece.getName().equals("pawn") && piece.getColor().equals(Piece.WHITE)) {
        return (this.piece != null);
    }else{
        return (this.piece == null);
    }}


//vais buscar uma peça especifica
public Piece getPiece() {
    return this.piece;
}
```

```
    // alteral o valor da variavel peça pelo que foi passado pelo parametro
    public void setPiece(Piece piece) {
        this.piece = piece;
        this.piece.setBoardpieceOn(this);
        this.pane.getChildren().add(piece.getImage());
        this.piece.setOnDragCompleteListener(this);//drag completo
    }
    //variavel igual ao setPiece a unica diferença é que serve para mudar os peões
    public void changePiece(Piece piece){
        this.piece = piece;
        this.piece.setBoardpieceOn(this);
        this.pane.getChildren().add(piece.getImage());
        this.piece.setOnDragCompleteListener(this);//drag completo
    }
    //remove uma peça selecionada
    public void removePiece() {
        this.pane.getChildren().remove(this.piece.getImage());
        this.piece = null;

    }
    //retorna o valor da variavel tabuleiro
    public Pane getPane() {
        return this.pane;
    }
    //retorna o valor da variavel linha
    public int getRow() {
        return row;
    }
    // retorna o valor da variavel coluna
    public int getCol() {
        return col;
    }
}
```

Pawn

Das classes relacionadas com as peças eu diria que esta é a mais complexa já que o Peão pode se movimentar de tantas formas diferentes conforme a situação que se encontra.

```
if (this.getColor().equals(Piece.WHITE)) {//peças brancas
    if (oldRow == 6) { //O peão pode dar 2 passos apenas se estiver na posição original
        if (Math.abs(newCol - oldCol) == 1 && newRow == oldRow - 1 && b.hasPiece(newRow, newCol)) { //come lateralmente
            return true;
        } else if (newCol == oldCol && newRow == oldRow - 1 && !b.hasPiece(newRow, newCol)) { //primeira casa só anda para a frente caso não exista peça
            return true;
        //segunda casa só anda para a frente caso não exista peça
        } else if (newCol == oldCol && newRow == oldRow - 2 && !b.hasPiece(newRow, newCol) && !b.hasPiece(newRow + 1, newCol)) {

            return true;
        }

        return false;
    }else  if ( oldRow == 3 ) { //"EN PASSANT BRANCAS"
        if (Math.abs(newCol - oldCol) == 1 && newRow == oldRow - 1 && b.hasPiece(newRow, newCol)) {//come lateralmente
            onPassant = false; //necessário para não eliminar caso coma uma peça lateral e tenha passado por uma area onde poderia ter feito enpassant
            return true;
        } else if (newCol == oldCol && newRow == oldRow - 1 && !b.hasPiece(newRow, newCol) ) {//primeira casa só anda para a frente caso não exista peça
            onPassant = false;//necessario pois se passa-se em drag por cima de uma area enpassant este ficaria true e eliminaria a peça
            return true;
        }else if (newCol == oldCol - 1 && newRow == oldRow - 1  && b.hasPiecePawnBlack(newRow + 1, newCol )  ) {//movimento diagonal para a esquerda
            //caso tenha um peão ao seu lado onpassant ativa hasPiecePawn...
            colToEat = newCol; //envia a posição da peça preta da coluna preta
            rowToEat = newRow + 1; //envia a posição da peça preta da linha
            onPassant = true;
            return true ;
        }else if (newCol == oldCol + 1 && newRow == oldRow - 1  && b.hasPiecePawnBlack(newRow + 1, newCol )  ) { //moviemnto diagonal para a direita
            //caso tenha um peão ao seu lado onpassant ativa hasPiecePawn...
            colToEat = newCol; //envia a posição da peça preta da coluna preta
            rowToEat = newRow + 1; //envia a posição da peça preta da linha
            onPassant = true;
             return true;
        }
        return false;
    }

    else  { //O peão não está mais na posição original
        if (Math.abs(newCol - oldCol) == 1 && newRow == oldRow - 1 && b.hasPiece(newRow, newCol)) { //come lateralmente
            return true;
        } else if (newCol == oldCol && newRow == oldRow - 1 && !b.hasPiece(newRow, newCol)) { // só anda para a frente caso não exista peça
            return true;
        }
        return false;
    }
} else { //peças pretas
    if (oldRow == 1) { //O peão pode dar 2 passos apenas se estiver na posição original
        if (Math.abs(newCol - oldCol) == 1 && newRow == oldRow + 1 && b.hasPiece(newRow, newCol)) {//come lateralmente
            return true;
        } else if (newCol == oldCol && newRow == oldRow + 1 && !b.hasPiece(newRow, newCol)) {//primeira casa só anda para a frente caso não exista peça
            return true;
        //segunda casa só anda para a frente caso não exista peça
        } else if (newCol == oldCol  && newRow == oldRow + 2 && !b.hasPiece(newRow, newCol) && !b.hasPiece(newRow - 1, newCol)) {
```

Esta classe faz várias verificações como por exemplo se o movimento que o Peão selecionado esta prestes a fazer é o primeiro ou não, sendo que o Peão no primeiro movimento pode se mover ate duas casas para a frente em vez de uma só.

Bishop

A classe associada a peça do Bispo, como as restantes classes da mesma categoria, estão todas encarregadas de fazer os cálculos para saber quais são as casas para aonde as peças se podem mover livremente.

```
//Verifique o movimento nas linhas diagonais
//Verifique se a nova posição na linha diagonal está bloqueada por outra peça
if (Math.abs(newRow - oldRow) == Math.abs(newCol - oldCol)) {

    //Verifique à direita - parte inferior

    if (newRow > oldRow && newCol > oldCol) {
        for (int i = 1; i < newRow - oldRow; i++) {
            if (b.hasPiece(oldRow + i, oldCol + i)) {
                return false;
            }
        }
        return true;
    }

    //Verificar à direita - parte superior

    if (newRow < oldRow && newCol > oldCol) {
        for (int i = 1; i < newCol - oldCol; i++) {
            if (b.hasPiece(oldRow - i, oldCol + i)) {
                return false;
            }
        }
        return true;
    }

    //Verifique a parte inferior esquerda

    if (newRow > oldRow && newCol < oldCol) {
        for (int i = 1; i < newRow - oldRow; i++) {
            if (b.hasPiece(oldRow + i, oldCol - i)) {
                return false;
            }
        }
        return true;
    }

    //Verifique à esquerda - parte superior

    if (newRow < oldRow && newCol < oldCol) {
        for (int i = 1; i < oldCol - newCol; i++) {
            if (b.hasPiece(oldRow - i, oldCol - i)) {
                return false;
            }
        }
        return true;
    }
    return true;
```

Knight

Na classe Cavalo tem de se fazer um calculo para saber que casas ele pode saltar, sendo o movimento dele em “L”.

```
    //Verifique se o boardpiece/tabuleiro  tem a peça do mesmo lado
     if (b.hasPiece(newRow, newCol)) {
         if (b.getPiece(newRow, newCol).getColor().equals(getColor())) {
             return false;
         }
     }
     // o cavalo move-se em L
     return ((Math.abs(newRow - oldRow) == 2 && Math.abs(newCol - oldCol) == 1)
             || (Math.abs(newRow - oldRow) == 1 && Math.abs(newCol - oldCol) == 2));
}
```

Tower

Esta Classe como as outras trata do movimentos da Torre como também, mas esta tem uma diferença já que também tem uma condição especial que esta associada a jogada especial Roque.

```
    // Verifique o movimento na linha horizontal
    if (newRow == oldRow) {
        if (newCol > oldCol) {
            for (int i = oldCol + 1; i < newCol; i++) {
                if (b.hasPiece(oldRow, i)) {
                    return false;
                }
            }
            return true;
        }
        if (newCol < oldCol) {
            for (int i = oldCol - 1; i > newCol; i--) {
                if (b.hasPiece(oldRow, i)) {
                    return false;
                }
            }
            return true;
        }
        if (newCol == oldCol) {
            return true;
        }
        return true;
    }

    //Verifique o movimento na linha vertical
    if (newCol == oldCol) {
        if (newRow > oldRow) {
            for (int i = oldRow + 1; i < newRow; i++) {
                if (b.hasPiece(i, oldCol)) {
                    return false;
                }
            }
            return true;
        }
        if (newRow < oldRow) {
            for (int i = oldRow - 1; i > newRow; i--) {
                if (b.hasPiece(i, oldCol)) {
                    return false;
                }
            }
            return true;
        }
        return true;
    }

    return false;
```

Queen

A Classe da Rainha calcula as casas para qual a peça se pode mover.

```
//Verifique o movimento na linha horizontal
if (newRow == oldRow) {
    if (newCol > oldCol) {
        for (int i = oldCol + 1; i < newCol; i++) {
            if (b.hasPiece(oldRow, i)) {
                return false;
            }
        }
        return true;
    }
    if (newCol < oldCol) {
        for (int i = oldCol - 1; i > newCol; i--) {
            if (b.hasPiece(oldRow, i)) {
                return false;
            }
        }
        return true;
    }
    if (newCol == oldCol) {
        return true;
    }
    return true;
}

//Verifique o movimento na linha vertical
if (newCol == oldCol) {
    if (newRow > oldRow) {
        for (int i = oldRow + 1; i < newRow; i++) {
            if (b.hasPiece(i, oldCol)) {
                return false;
            }
        }
        return true;
    }
    if (newRow < oldRow) {
        for (int i = oldRow - 1; i > newRow; i--) {
            if (b.hasPiece(i, oldCol)) {
                return false;
            }
        }
        return true;
    }
    return true;
}
```

```
//Verifique o movimento nas linhas diagonais
//Verifique se a nova posição na linha diagonal está bloqueada por outra peça
if (Math.abs(newRow - oldRow) == Math.abs(newCol - oldCol)) {

    //Verifique à direita - parte inferior

    if (newRow > oldRow && newCol > oldCol) {
        for (int i = 1; i < newRow - oldRow; i++) {
            if (b.hasPiece(oldRow + i, oldCol + i)) {
                return false;
            }
        }
        return true;
    }

    //Verificar à direita - parte superior

    if (newRow < oldRow && newCol > oldCol) {
        for (int i = 1; i < newCol - oldCol; i++) {
            if (b.hasPiece(oldRow - i, oldCol + i)) {
                return false;
            }
        }
        return true;
    }

    //Verifique a parte inferior esquerda

    if (newRow > oldRow && newCol < oldCol) {
        for (int i = 1; i < newRow - oldRow; i++) {
            if (b.hasPiece(oldRow + i, oldCol - i)) {
                return false;
            }
        }
        return true;
    }

    //Verifique à esquerda - parte superior

    if (newRow < oldRow && newCol < oldCol) {
        for (int i = 1; i < oldCol - newCol; i++) {
            if (b.hasPiece(oldRow - i, oldCol - i)) {
                return false;
            }
        }
        return true;
    }
    return true;
}
```

King

NA Classe do Rei não temos só os movimentos como também temos a verificação da posição do mesmo para se poder efetuar uma das jogadas especiais que é o roque.

```
//Verifique se o boardpiece/tabuleiro  tem a peça do mesmo lado
if (b.hasPiece(newRow, newCol)) {
    if (b.getPiece(newRow, newCol).getColor().equals(getColor())) {
        return false;
    }
}

//o rei só se pode mover uma casa em toda a sua volta
if (Math.abs(newRow - oldRow) < 2 && Math.abs(newCol - oldCol) < 2) {
    return true;
}

/*
```

Interação

Piece

Nesta classe é aonde se faz o desenho da peça como também é aonde se trata do *drag and drop*.

```
public abstract class Piece {
    public static final String BLACK = "black";
    public static final String WHITE = "white";
    public static boolean onPassant;

    public static int rowToEat=-1;
    public static int colToEat=-1;


    protected ImageView image;
    protected String color;
    protected Boardpiece boardpieceOn;
    protected OnDragCompleteListener mListener;


    //funçaõ que destingue cor, tamanho e tipo das peças
    public Piece(String color) {
        this.color = color;
        String filePath = "chessecgm/designs/pieces/" + this.getColor() + "_" + this.getName() + ".png";
        this.image = new ImageView(filePath);
        this.image.setFitWidth(62.5);
        this.image.setFitHeight(62.5);
    }
    //retorna a imagem da peça
    public ImageView getImage() {
        return this.image;
    }
    //retorna a cor da peça
    public String getColor() {
        return this.color;
    }
    //função que premite arrastar a peça ao longo da board
    public void addImageDragEvent() {
        this.image.setOnDragDetected(new ImageDragDetectedEvent());// deteta que a imagem vaiser arrastada
        this.image.setOnDragDone(new ImageDragDoneEvent());// determina quadndo o drag acaba
    }
     // alteral o valor da variavel setOnDragCompleteListener/ Drag completo  que foi passado pelo parametro
    public void setOnDragCompleteListener(OnDragCompleteListener listener) {
        this.mListener = listener;
    }
    //retorna a peça que está no tabuleiro
    public Boardpiece getBoardpieceOn() {
        return boardpieceOn;
    }
    // alteral o valor da variavel setBoardpieceOn/ da peça que está no tabuleiro que foi passado pelo parametro
    public void setBoardpieceOn(Boardpiece boardpieceOn) {
        this.boardpieceOn = boardpieceOn;
    }
```

```
    //retorna o nome de cada uma da função das peças
    public abstract String getName();
    //se o movimento é legal
    public abstract boolean isLegalMove(ChessBoard b, int newRow, int newCol);
    //interface que teteta que o Drag esta a funcionar
    public interface OnDragCompleteListener {
        void onDragComplete();
    }
    //decteta quando a imagem é amarrada
    private class ImageDragDetectedEvent implements EventHandler<MouseEvent> {
        @Override
        public void handle(MouseEvent event) {//funçao de quando a amarra
            Dragboard db = image.startDragAndDrop(TransferMode.MOVE);
            Image dragShadow = image.getImage();//amarrar a imagem e chamala peça sombra
            db.setDragView(dragShadow, dragShadow.getWidth()/2, dragShadow.getHeight()/2 );//tamanho da peça sombra
            ClipboardContent content = new ClipboardContent();// string que contem cor, tipo da peça ,linha e coluna
            content.putString(getColor() + "_"
                    + getName() + "_"
                    + Integer.toString(boardpieceOn.getRow()) + "_"
                    + Integer.toString(boardpieceOn.getCol()));
            db.setContent(content);

            event.consume();
        }
    }
    //quando o drag está completo ou seja drop
    private class ImageDragDoneEvent implements EventHandler<DragEvent> {
        @Override
        public void handle(DragEvent event) {
            if (event.getTransferMode() == TransferMode.MOVE) {
                onPassant=false;
                mListener.onDragComplete();

            }
            event.consume();
        }
    }
}
```

Client

A Classe do Client esta encarregada de enviar como também receber a informação enviada de um Jogador para o outro como pedidos de desistir e a jogada do jogador oposto.

```
public class Client {
    private String ip;
    private Socket socket;
    private BufferedReader reader;
    private PrintWriter writer;
    private PrintWriter givupp;
    private PrintWriter draww;
    private PrintWriter winmessages;
    private PrintWriter pawnasks;
    public static Client Instance;

    private DataReceiveListener mListener;

    //comunicação com o servidor
    public Client(String ip) {

        Instance= this;
        this.ip = ip;
        try {
            socket = new Socket(ip, 5000); //ip e porta predefenida
            InputStreamReader isReader = new InputStreamReader(socket.getInputStream());
            reader = new BufferedReader(isReader);
            writer = new PrintWriter(socket.getOutputStream());
            givupp = new PrintWriter(socket.getOutputStream());
            draww = new PrintWriter(socket.getOutputStream());
            winmessages = new PrintWriter(socket.getOutputStream());
            pawnasks = new PrintWriter(socket.getOutputStream());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    //receção de mensagens pelo servidor
    public void setOnDataReceiveListener(DataReceiveListener listener) {
        this.mListener = listener;
    }
    // envia mensagens ENTRE JOGADORES
    public void sendMessage(String message) {
        writer.println(message);
        writer.flush();

    }
    //recebe mensagens entre jogadores
    public String receiveMessage() {
        String result = null;

        try {
            result = reader.readLine();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return result;
```

```
    public void startDataThread() {

        Thread t = new Thread(new IncomingDataReader());
        t.start();
    }
    // mensagem de envio de encerramento de sessao / desistir
    public void closeConnection(String messages) {
        givupp.println(messages);
        givupp.flush();

        try {
            socket.close();

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    //mensagens de empate
      public void draw(String messagess) {
         draww.println(messagess);
        draww.flush();

    }
   //mensagens de vitoria
        public void winyes(String winmessage) {
         winmessages.println(winmessage);
        winmessages.flush();

    }
    //mensagens de troca de peao por outra peça
         public void pawnask(String pawnmessage) {
         pawnasks.println(pawnmessage);
        pawnasks.flush();

    }
   //esta classe lê todas as mensagens enviadas e e define diferentes tipos de feedbacks para
   //Leitor de dados recebidos
    private class IncomingDataReader implements  Runnable {

        @Override
        public void run() {
            String message;
             //quando estas mensagens são enviadas entre jogadores acontece o seguinte:
            while ((message = receiveMessage()) != null) {

                final String s = message;
                if (message.contains("_")) {// movimento de peças
                    Platform.runLater(new Runnable() {
                        @Override
                        public void run() {
                            System.out.println("recebi movimento");
```

```
                    mListener.onMoveReceive(s);

                }
            });
        } else if(message.contains("desistiu!")){//
            Platform.runLater(new Runnable() {
                @Override
                public void run() {//desistiu
                    mListener.onGiveUpReceive(s);

                }
            });

        }else if(message.contains("?-")){//trocar peça peão por outra peça
            Platform.runLater(new Runnable() {
                @Override
                public void run() {
                    System.out.println(s);
                    if(!Boardpiece.iChangePiece){
                        mListener.onChangePieceReceive(s);
                    }
                     Boardpiece.iChangePiece=false;
                }
            });
        }else if(message.contains("!-")){//remoção peão enpassant
            Platform.runLater(new Runnable() {
                @Override
                public void run() {
                    System.out.println(s);
                    if(!Boardpiece.iRemovePiece){
                        mListener.onRemovePieceReceive(s);

                    }
                     Boardpiece.iRemovePiece=false;
                }
            });
        }else if(message.contains("Pedido de Empate")){ //pedido de empate
          if(MultiplayerModeController.iwantdraw==true){
            MultiplayerModeController.iwantdraw=false;//para enviar uma mensage
            }else{
            Platform.runLater(new Runnable() {
                @Override
                public void run() {
                       mListener.onDrawReceive(s);
                }
            });}
        }else if(message.contains("aceitou o Empate!")){//aceita empate
            Platform.runLater(new Runnable() {
                @Override
                public void run() {
```

```
                    mListener.onyesdrawReceive(s);
                }
            });
        }else if(message.contains("REJEITADO!")){//pedido de empate rejeitado
          if(MultiplayerModeController.wantnodraw==true){
            MultiplayerModeController.wantnodraw=false;//para enviar uma mensagem ao ad
            }else{
            Platform.runLater(new Runnable() {
                @Override
                public void run() {

                    mListener.onnodrawReceive(s);
                }
            });}
        }else if(message.contains("Adversário ganhou o jogo!")){//vitoria
          if(Boardpiece.winyestrue==true){
            Boardpiece.winyestrue=false;//para enviar uma mensagem ao adversário e outr
            }else{
            Platform.runLater(new Runnable() {
                @Override
                public void run() {

                    mListener.onwinReceive(s);
                }
            });}
        }else if(message.contains("Desculpe! Servidor Caiu!")){//vitoria

            Platform.runLater(new Runnable() {
                @Override
                public void run() {

                    mListener.onGiveUpReceive(s);
                }
            });
        }

        else if(message.contains("Troca peão por outra peça.")){ //troca peão por outr
            if(Boardpiece.pawnaskmessage==true){
            Boardpiece.pawnaskmessage=false;//para enviar uma mensagem ao adversário e
            }else{
            Platform.runLater(new Runnable() {
                @Override
                public void run() {

                    mListener.onPawnAskReceive(s);
                }
            });}
        }else{
             mListener.onChatReceive(s);
        }
```

```
                }
            }
        }
        //Interface de Rececao de dados
        public interface DataReceiveListener {
            void onChatReceive(String message);
            void onMoveReceive(String move);
            void onGiveUpReceive(String messages);
            void onDrawReceive(String messagess);
            void onyesdrawReceive(String messagess);
            void onnodrawReceive(String messagessss);
            void onwinReceive(String winmessage);
            void onPawnAskReceive(String pawnmessage);
            void onChangePieceReceive(String changemessage);
            void onRemovePieceReceive(String changemessage);
        }
    }
```

Interface

LobbyController

Esta Classe contem o código para a interface aonde se introduz os dados necessários para se conectar e de quando o jogador esta a espera de se conectar com o outro.

```
public class LobbyController implements Initializable {

    @FXML
    private TextField textUserName;

    @FXML
    private RadioButton GameIPMultiplayer;

    @FXML
    private TextField textIp;

    @FXML
    private Button btnStart;

    @FXML
    private Button btnClose;

    private ToggleGroup gameMode;


    @Override
    //iniciar
    public void initialize(URL location, ResourceBundle resources) {
        setupGameModeMiltiP();// escolha do modo de jogo já predefenido num unico toogle
        btnStart.setDisable(true);// botao start true
    }

    @FXML
    //botao Iniciar Partida
    void onButtonStartClick(ActionEvent event) throws IOException {

        if (isValidToStart()) {//inicia o MultiplayerModeController
                Client client = new Client(textIp.getText());
                //envia usernames
                client.sendMessage(textUserName.getText());
                //conexão estabelecida
                establishConnection(client);
            }
       btnStart.setDisable(true);

    }

     @FXML
    //botão SAIR
    void onButtonClose(ActionEvent event) throws IOException {
         Platform.exit();
         System.exit(0);
    }
```

```
@FXML
//introdução do IP na boxtext
void onTextIpPress(KeyEvent event) {
    if (isValidToStart()) {
        if (event.getCode() == KeyCode.ENTER) { //caso carregue enter inicia  o MultiplayerModeController
            Client client = new Client(textIp.getText()); //ip dos jogadores
            //envia usernames
            client.sendMessage(textUserName.getText());
            //conexão estabelecida
            establishConnection(client);
        }
    }else{
        //sem IP não deixa começar
            btnStart.setDisable(false);
        }
}
//seleção do toggle, visto que só existe um modo de jogo está predefenidamente selecionado
private void setupGameModeMiltiP() {
    gameMode = new ToggleGroup();
    GameIPMultiplayer.setToggleGroup(gameMode);
    GameIPMultiplayer.setSelected(true);

    gameMode.selectedToggleProperty().addListener(new ChangeListener<Toggle>() {
        @Override
        public void changed(ObservableValue<? extends Toggle> observable, Toggle oldValue, Toggle newValue) {
            if (gameMode.getSelectedToggle() == GameIPMultiplayer) {
                textIp.setDisable(false);
            }
        }
    });
}
//validação de iniciar
public boolean isValidToStart() {

    //ter username
    if (!textUserName.getText().equals("")) {
        //seleção do modo de jogo multiplayer
        if (gameMode.getSelectedToggle() == GameIPMultiplayer) {
            //envio do IP
            return (!textIp.getText().equals(""));
        }
        return true;
    }
    return false;
}
```

```
    //iniciar o Toggle GameIPMultiplayer
    private void startGameIPMultiplayer(Client connection, String playerId, String playerName, String rivalName) {
        try {
            //inicir o jogo
            MultiplayerModeController MultiplayerModeController = new MultiplayerModeController(connection, playerId, playerName, rivalName);
            //envia para uma nova pagina
            FXMLLoader loader = new FXMLLoader(Main.class.getResource("view/main_screen.fxml"));
            //iniciar toda a nova cena
            Stage stage = (Stage) btnStart.getScene().getWindow();
            loader.setController(MultiplayerModeController);
            Parent root = loader.load();
            stage.getScene().setRoot(root);
            stage.show();

            //barrar recise
            stage.sizeToScene();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    //conexão estabelecida entre jogadores
    private void establishConnection(Client client) {
        Thread t = new Thread(new Runnable() {
            @Override
            public void run() {
                //string com indormação enviada por cada jogador separadas por _ ou seja Client_playerID_playerName_RivalName
                String[] info = client.receiveMessage().split("_");
                Platform.runLater(new Runnable() {
                    @Override
                    public void run() {
                        startGameIPMultiplayer(client, info[0], info[1], info[2]);
                    }
                });
            }
        });
        t.start();
    }


}
```

MultiplayerModeController

Esta Classe já esta encarregada pelo código da interface de durante o jogo.

É a classe que contem como a anterior os botões necessários para todas a funcionalidades impostas.

```
public class MultiplayerModeController implements Initializable, Client.DataReceiveListener, ChessBoard.OnPieceMoveListener{
    @FXML
    private Label nameP2;

    @FXML
    private Label lastMoveP2;

    @FXML
    private Label nameP1;

    @FXML
    private Label lastMoveP1;

    @FXML
    private Label infocurrent;

    @FXML
    private GridPane chessPane;

    @FXML
    private Label lblChessTimer;

    @FXML
    private TextArea chatBox;

    @FXML
    private TextField textInput;

    @FXML
    public Button closeButton;

    @FXML
    public Button yesdraw;

    @FXML
    public Button nodraw;

    @FXML
    public Button drawask;

    @FXML
    public Button exitbotton;

    @FXML
    public SplitPane drawsplit;


    @FXML
    private Button btnSend;
```

```
private Client client;
private ChessBoard chessBoard;
private String playerId;
private String playerName;
private String rivalName;
public static boolean iwantdraw=false;
public static boolean wantnodraw=false;
public static boolean online=false;
int countdown= 240;

Timer timer = new Timer();
TimerTask task = new TimerTask(){

    //timer impressões na tela
    public void run()
       {
            if(countdown >0)
            countdown--;

            if(countdown > 8){
            Platform.runLater(() ->lblChessTimer.setText("Falta: " + countdown+ " (seg)"));}
            else{
            Platform.runLater(() ->lblChessTimer.setText("Atenção! Falta: " + countdown+ " (seg)"));
            }
            if(countdown == 0){
            Platform.runLater(() ->lblChessTimer.setText("À espera de jogada!"));
            }
        }

};

//comunicação com o client , o outro jogador
public MultiplayerModeController(Client connection, String playerId, String playerName, String rivalNa
    this.client = connection;
    this.playerId = playerId;
    this.playerName = playerName;
    this.rivalName = rivalName;

    client.setOnDataReceiveListener(this);
    client.startDataThread();

    chessBoard = new ChessBoard(playerId);
    chessBoard.setOnPieceMoveListener(this);

    timer.scheduleAtFixedRate(task,1000,1000);
}


public void start(Stage primaryStage) throws Exception {
    FXMLLoader loader = new FXMLLoader(getClass().getResource("view/main_screen.fxml"));
    Parent root = (Parent) loader.load();
    primaryStage.setResizable(false);


    primaryStage.initStyle(StageStyle.DECORATED);

    primaryStage.setScene(new Scene(root));
    primaryStage.show();
```

```
    @Override
    //inicia o main jogo
    public void initialize(URL location, ResourceBundle resources) {
        //cabeçalho
        setStageTitle("Jogo Xadrez - Jogador: " + playerName +"  --> Id: "+ playerId);
        //desenha o tabuleiro
        drawChessPane();
        //mostra os nomes
        displayPlayerName();
        //alerta para enviar mensagem de desistir só quando está online
        online=true;

      //quando o jogo começa as pretas não podem se mexer
        if (playerId.equals("2"))
            chessPane.setDisable(true);
         lblChessTimer.setText("Falta: 240");
    }

     //receber mensagens no chat
    @Override
    public void onChatReceive(String message) {
        chatBox.appendText(message + "\n");
    }


     //botão de sair
     public void onButtonleave(ActionEvent event) {

        Stage stage = (Stage) closeButton.getScene().getWindow();
        stage.close();

    }
    // botão desistir
      @FXML
    public void onButtongivup(ActionEvent event) {

        Stage stage = (Stage) closeButton.getScene().getWindow();
        stage.close();
        client.closeConnection(playerName + " desistiu! " );
    }
  //informação recebida quando alguem desite
    public void onGiveUpReceive(String messages) {
        chessPane.setDisable(true);
        infocurrent.setText(messages );
        yesdraw.setVisible(false);
        nodraw.setVisible(false);
        drawask.setVisible(false);
        closeButton.setVisible(false);
        lblChessTimer.setVisible(false);
        exitbotton.setVisible(true);
        btnSend.setDisable(true);
        textInput.setDisable(true);


     }
```

```
    // botão de empate
    public void onButtondraw(ActionEvent event) {
        iwantdraw=true;
        client.draw("Pedido de Empate de "+ playerName+ "!!" );
        infocurrent.setText(" Pedido de Empate. Aguarde!!" );
        chessPane.setDisable(true);
        drawask.setVisible(false);
        closeButton.setVisible(false);
        exitbotton.setVisible(false);


    }

    //receção informação de empate
    public void onDrawReceive(String messagess) {

        chessPane.setDisable(true);
        infocurrent.setText(messagess );
        yesdraw.setVisible(true);
        nodraw.setVisible(true);
        drawask.setVisible(false);
        closeButton.setVisible(false);
        exitbotton.setVisible(false);



    }

    //botão de negar empate
public void onButtonnodraw(ActionEvent event) {

        chessPane.setDisable(false);
        wantnodraw=true;
        yesdraw.setVisible(false);
        nodraw.setVisible(false);
        drawask.setVisible(false);
        closeButton.setVisible(true);
        exitbotton.setVisible(false);


      if (playerId.equals("1")) {

          infocurrent.setText("É a sua vez " +playerName+"!");
          client.draw("REJEITADO! É a sua vez " +playerName+"!" );

       } else if (playerId.equals("2")) {

          infocurrent.setText("É a sua vez " +playerName+"!");
          client.draw("REJEITADO! É a sua vez " +playerName+"!" );
       }

    }

    // botão de receção de negação de empate
    public void onnodrawReceive(String messagessss) {

        chessPane.setDisable(true);
        infocurrent.setText(messagessss );
        nodraw.setVisible(false);
```

```
            drawask.setVisible(false);
            closeButton.setVisible(false);
            exitbotton.setVisible(true);
            btnSend.setDisable(true);
            textInput.setDisable(true);
            lblChessTimer.setVisible(false);
            client.draw(playerName + " aceitou o Empate!" );

        }
        //receção de aceitar empate
        public void onyesdrawReceive(String messagess) {

            chessPane.setDisable(true);
            infocurrent.setText( messagess );
            yesdraw.setVisible(false);
            nodraw.setVisible(false);
            drawask.setVisible(false);
            closeButton.setVisible(false);
            exitbotton.setVisible(true);
            btnSend.setDisable(true);
            textInput.setDisable(true);
            lblChessTimer.setVisible(false);


        }
       //receção de quando alguem faz xeque-mate
        public void onwinReceive(String winmessage) {

            lblChessTimer.setVisible(false);
            chessPane.setDisable(true);
            infocurrent.setText( winmessage );
            yesdraw.setVisible(false);
            nodraw.setVisible(false);
            drawask.setVisible(false);
            closeButton.setVisible(false);
            exitbotton.setVisible(true);
            btnSend.setDisable(true);


            Alert alert = new Alert(AlertType.CONFIRMATION);
            alert.setTitle("Xeque-mate!");//alert pop up a avisar o vencedor
            alert.setTitle("Xeque-mate!Adversário ganhou o jogo!");//alert pop up a avisar o vencec
            alert.setHeaderText("Desta vez perdeste o jogo, Tenta outra vez!");
            ButtonType buttonTypeOne = new ButtonType("Sair");
            ButtonType buttonTypeTwo = new ButtonType("Ok");
            alert.getButtonTypes().setAll(buttonTypeOne, buttonTypeTwo);

            Optional<ButtonType> result = alert.showAndWait();

            if (result.get() == buttonTypeOne){

            System.exit(0);        // quando o jogador carrega em Sair a aplicação dá shut down
            }else {
            System.exit(0);

            }
```

```
//receção da informação que o adversário vai trocar o peão por alguma peça
public void onPawnAskReceive(String pawnmessage) {

    infocurrent.setText( pawnmessage );
    yesdraw.setVisible(false);
    nodraw.setVisible(false);


}

//receção de movimento de peça pelo adversário
@Override
public void onMoveReceive(String move) {
    chessPane.setDisable(false);
    Move m = new Move(move);
    yesdraw.setVisible(false);
    nodraw.setVisible(false);
    drawask.setVisible(false);

    if (playerId.equals("1")) {
      //histórico da ultima peça do jogador atualizado para o adversário
      lastMoveP2.setText(m.toBoardMove());
      //atualização do status de quem joga para o rival
      infocurrent.setText("É a sua vez " +playerName+"!");


    } else if (playerId.equals("2")) {
      //histórico da ultima peça do jogador atualizado para o adversário
      lastMoveP1.setText(m.toBoardMove());
      //atualização do status de quem joga para o rival
      infocurrent.setText("É a sua vez " +playerName+"!");
    }


    chessBoard.updateBoard(m);
    countdown = 240;
}
//
@Override
public void onPieceMove(Move move) {
    chessPane.setDisable(true);
    yesdraw.setVisible(false);
    nodraw.setVisible(false);
    drawask.setVisible(true);

    //envia a posição da peça para o Client
    client.sendMessage(playerId + "_" + move.toString());

    if (playerId.equals("1")) {
        //histórico da ultima peça do jogador atualizado para o jogador
        lastMoveP1.setText(move.toBoardMove());
        //atualização do status de quem joga para o jogador a jogar
        infocurrent.setText("É a sua vez " +rivalName+"!");
    } else if (playerId.equals("2")) {
        //histórico da ultima peça do jogador atualizado para o jogador
        lastMoveP2.setText(move.toBoardMove());
        //atualização do status de quem joga para o jogador a jogar
        infocurrent.setText("É a sua vez " +rivalName+"!");
```

```
        countdown = 240;
    }

    //envio da substituição do peão por outra peça  na promoção do mesmo
    @Override
    public void onPieceChange(int row, int col, String piece){
        client.sendMessage("?-"+row+"-"+col+"-"+piece); //envia uma mensage para o adversario com a
    }
    //receção da substituição do peão por outra peça  na promoção do mesmo pelo adversário
    @Override
    public void onChangePieceReceive(String changemessage){
        String[] s = changemessage.split("-"); //desfaz a string em partes
        int col = Integer.parseInt(s[1]); //linha
        int row = Integer.parseInt(s[2]); //coluna
        String[] t = s[3].split("#");//peça trocada começa por #peca.cor
        Piece piece = PieceFactory.getPiece(t[0], t[1]);//nova peça criada na fabrica de peças
        chessBoard.changePiece(col,row,piece);
    }
    //envio da remoção do peão enpassant
    public void onPieceRemove(int row, int col){
        client.sendMessage("!-"+row+"-"+col);
    }
     //receção da remoção do peão enpassant envio paraoadversario
       @Override
    public void onRemovePieceReceive(String changemessage){

        String[] s = changemessage.split("-");
        int col = Integer.parseInt(s[1]);
        int row = Integer.parseInt(s[2]);
        chessBoard.removePiece(col,row);
    }

    @FXML
    //envio de mensagens quando carregado no botão
    void onButtonSendClick(ActionEvent event) {
        client.sendMessage(playerName + ": " + textInput.getText());
        textInput.clear();
    }

    @FXML
    //quando carrega na caixa de texto
    void onTextInputPress(KeyEvent event) {
        // caso carregue na tecla ENTER
        if (event.getCode() == KeyCode.ENTER) {
            client.sendMessage(playerName + ": " + textInput.getText());
            textInput.clear();
        }
    }
    //desenho do tabuleiro de xadrez
    private void drawChessPane() {
        for (int row = 0; row < 8; row++) {
            for (int col = 0; col < 8; col++) {
                chessPane.add(chessBoard.getBoardpiece(row, col).getPane(), col, row);
            }
        }
    }
```

```
    //mostrar os nome dos jogadores no painel quando inicia
    private void displayPlayerName() {
        if (playerId.equals("1")) {
           //mostra quem começa a jogar quando o jogo começa
            infocurrent.setText("É a sua vez " +playerName+"!");
            namePl.setText(playerName);
            namePl2.setText(rivalName);

        } else {
            //mostra quem começa a jogar quando o jogo começa
            infocurrent.setText("É a sua vez " +rivalName+"!");
            namePl.setText(rivalName);
            namePl2.setText(playerName);

        }
    }

    private void setStageTitle(String title) {
        Main.getStage().setTitle(title);
    }
}
```

Timer

Esta classe esta encarregada de gerir o Timer do jogo.

```
public class Timer implements java.lang.Runnable{

    @Override
    public void run() {
        this.runTimer();
    }

    public void runTimer(){
        int i = 240;
        while (i>0){
         System.out.println("Remaining: "+i+" seconds");
         try {
           i--;
           Thread.sleep(1000L);     // 1000L = 1000ms = 1 second
          }
          catch (InterruptedException e) {

          }
        }
    }

}
```

Main

É a Classe encarregada de iniciar o pograma do Client todo como também faz aparecer um pop-up de quando algum dos Jogadores carrega no “X” para ter certeza se quer fechar a aplicação ou não.

```
public class Main extends Application {
    private static Stage stage;
     private Client client;
     private MultiplayerModeController multiplayermodecontroller;

    @Override
    public void start(Stage primaryStage) throws Exception{
        stage = primaryStage;
        Parent root = FXMLLoader.load(getClass().getResource("view/lobby_screen.fxml"));
        primaryStage.setTitle("Jogo Xadrez");
        primaryStage.setScene(new Scene(root));

        primaryStage.setResizable(false);
        stage.initStyle(StageStyle.DECORATED);
      // caso deseje sair do jogo pelo X boton


       primaryStage.setOnCloseRequest(new EventHandler<WindowEvent>() {
    @Override
    public void handle(WindowEvent event) {
        System.out.println("hidding");
        Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
        alert.setTitle("Sair do Jogo!");
        alert.setHeaderText("Deseja mesmo fechar o jogo??");
        alert.setContentText("Se sim carregue em SIM, caso pretenda continuar NÃO...Também podes usar os butões Sair ou Desistir...");
        Toolkit.getDefaultToolkit().beep();



        ButtonType buttonTypeOne = new ButtonType("SIM");
        ButtonType buttonTypeTwo = new ButtonType("NÃO");



        alert.getButtonTypes().setAll(buttonTypeOne, buttonTypeTwo);

        Optional<ButtonType> result = alert.showAndWait();

        if ( result.get() == buttonTypeOne ) {

         //desistir caso esteja em jogo
          if(MultiplayerModeController.online==true){
              Client.Instance.closeConnection(" Adversário desistiu!");
              Platform.exit();
              System.exit(0);
```

```
          }else{//caso ainda esteja no lobby
                Platform.exit();
                System.exit(0);
          }



        }else if (result.get() == buttonTypeTwo){

        }
         event.consume();
    }
});
        primaryStage.show();
    }


    public static void main(String[] args) {
        launch(args);
    }



    public static Stage getStage() {
        return stage;
    }
}
```

Classes (Server)

Server

A Classe server tem uma função bastante a classe Cliente em que esta encarregada de receber a informação do Client e enviar para o outro

```
public class Server {
    private ArrayList<Socket> clientSockets = new ArrayList<>(); //array com
    private ArrayList<PrintWriter> clientWriters = new ArrayList<>();
    private ServerSocket serverSocket;
    private ArrayList<String> playerNames = new ArrayList<>(); //array com os nomes dos jogadores

    private OnDataReceiveListener dataListener = null;
    private OnConnectionEstablishListener connectionListener = null;

    //iniciação do server
    public void start() {
        try {
            serverSocket = new ServerSocket(5000);//porta em que vai usar para comunicar

            for (int i = 1; i <= 2; i++) {
                Socket socket = serverSocket.accept();
                clientSockets.add(socket);
                PrintWriter writer = new PrintWriter(socket.getOutputStream());
                clientWriters.add(writer);

                Thread t = new Thread(new ConnectionHandler(socket));
                t.start();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    //verifica se recebe informaçao
    public void setOnDataReceiveListener(OnDataReceiveListener listener) {
        this.dataListener = listener;
    }
    //verifica de conecçao foi estabelicida
    public void setOnConnectionEstablishListener(OnConnectionEstablishListener listener) {
        this.connectionListener = listener;
    }
    //????
    private void startDataThread() {
        for (Socket socket : clientSockets) {
            Thread t = new Thread(new DataHandler(socket));
            t.start();
        }
    }
     //envia as mensagens do jogo, empate, vitoria, chat, etc etc
    public void sendChatData(String message) {
        Iterator it = clientWriters.iterator();
        while (it.hasNext()) {
            try {
                PrintWriter writer = (PrintWriter) it.next();
                writer.println(message);
                writer.flush();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
```

```
    private void sendPlayerInfo() {
        clientWriters.get(0).println("1" + "_" + playerNames.get(0) + "_" + playerNames.get(1)); //id_playerinimigo_pl
        clientWriters.get(0).flush();
        clientWriters.get(1).println("2" + "_" + playerNames.get(1) + "_" + playerNames.get(0)); //id_playername_playe
        clientWriters.get(1).flush();
    }
    //passar a informaçao de qual foi o movimento executado por algum dos jogadores
    private void sendMoveInfo(String message) {
        String[] s = message.split("_"); //separa a mensagem
        String data = message.substring(2);
        if (s[0].equals("1")) {
            clientWriters.get(1).println(data);     //envia a string com a mensagem do movimento
            clientWriters.get(1).flush();
        } else if (s[0].equals("2")) {
            clientWriters.get(0).println(data);
            clientWriters.get(0).flush();
        }
    }
    //actualizar o registo das jogadas
    private void updateLog(String message) {
        Platform.runLater(new Runnable() {
            @Override
            public void run() {
                dataListener.onDataReceive(message);
            }
        });
    }
    //gerente de conecção
    private class ConnectionHandler implements Runnable {
        BufferedReader reader;
        Socket socket;
        //????
        public ConnectionHandler(Socket clientSocket) {
            try {
                socket = clientSocket;
                InputStreamReader isReader = new InputStreamReader(socket.getInputStream());
                reader = new BufferedReader(isReader);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        //inicia a coneçao entre os jogadores e assim começa a permitir enviar e receber a informaçao de ambos
        @Override
        public void run() {
            String playerName;
            try {
                if ((playerName = reader.readLine()) != null) {
                    playerNames.add(playerName);
                    if (playerNames.size() == 2) {
                        Platform.runLater(new Runnable() {
                            @Override
                            public void run() {
                                connectionListener.onConnectionEstablish();
                            }
                        });
```

```
                    updateLog("Jogador " + playerNames.get(0) + " conectado!");
                    updateLog("Jogador " + playerNames.get(1) + " conectado!");
                    sendPlayerInfo();
                    startDataThread();
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

}
//Manipulador de dados
private class DataHandler implements Runnable {
    Socket socket;
    BufferedReader reader;

    public DataHandler(Socket socket) {
        try {
            this.socket = socket;
            InputStreamReader isReader = new InputStreamReader(socket.getInputStream());
            reader = new BufferedReader(isReader);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @Override
    //diferentes mensagens conforme o que acontece no jogo se decidiu jogar, desisistir, se perdeou ou venceu etc...
    // ESTAS MENSAGENS SÃO TANTO IMPRIMIDAS NO SERVER COMO ENTRE JOGADORES
    public void run() {
        String message;
        try {
            while ((message = reader.readLine()) != null) {
                if (message.contains("_")) {
                    String name = (message.split("_")[0].equals("1")) ? playerNames.get(0) : playerNames.get(1);
                    String data = "Jogador " + name + " fez um movimento...";
                    updateLog(data);
                    sendMoveInfo(message);
                } else if(message.contains("desistiu!")) {
                    String data = "Um dos Jogadores desistiu...";
                    updateLog(data);
                    sendChatData(message);
                }else if(message.contains("Troca peão por outra peça.")) {
                    String data = "Troca de Peão por peça...";
                    updateLog(data);
                    sendChatData(message);
                }else if(message.contains("Pedido de Empate")) {
                    String data = "Pedido de Empate por jogador...";
                    updateLog(data);
                    sendChatData(message);
                }else if(message.contains(" aceitou o Empate!")) {
                    String data = "Jogador aceitou o Empate...";
                    updateLog(data);
                    sendChatData(message);
```

```
                }else if(message.contains("REJEITADO!")) {
                    String data = "Jogador Rejeitou empate...";
                    updateLog(data);
                    sendChatData(message);
                }else if(message.contains("Adversário ganhou o jogo!")) {
                    String data = "Fim do jogo, Xeque-Mate...";
                    updateLog(data);
                    sendChatData(message);
                }else {
                    String data = "Mensagens enviadas...";
                    updateLog(data);
                    sendChatData(message);
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }

    }
}

interface OnDataReceiveListener {
    void onDataReceive(String message);
}

interface OnConnectionEstablishListener {
    void onConnectionEstablish();
}
}
```

MainController

Esta Classe regista o histórico do chat e recebe e apresenta a informação

```
public class MainController implements Initializable, Server.OnDataReceiveListener{
    @FXML
    private TextArea log;  //registo do historico de texto
    private Server server = null;

    public MainController(Server server) {
        this.server = server;
    }

    @Override
    //inicializa o main controller
    public void initialize(URL location, ResourceBundle resources) {
        log.setEditable(false);
        server.setOnDataReceiveListener(this);
    }

    @Override
    public void onDataReceive(String message) {
        log.appendText(message + "\n");  //recebe a informaçao e apresenta
    }
}
```

Main

A Classe Main do server para alem de ser a que inicia tudo também tem uma função de quando o jogador for a fechar o server e uma conexão ainda estiver ativa faz aparecer primeiro um pop-up para verificar se quer mesmo fechar e se sim envia uma mensagem para o outro Client a dizer que o server foi encerrado.

```
public class Main extends Application implements Server.OnConnectionEstablishListener{
    Stage primaryStage = null;

    Server server = new Server();  //criar o servidor
    @Override
    //iniciar servidor
    public void start(Stage primaryStage) throws Exception {
        this.primaryStage = primaryStage;

        server.setOnConnectionEstablishListener(this);  // alteral o valor da variavel setOnConnectionEstablishListener/conexção estabelecida que foi passado p

        FXMLLoader loader = new FXMLLoader(getClass().getResource("loading_screen.fxml"));//screen inicial esperando os jogadores

        Parent root = loader.load();
        primaryStage.setTitle("Jogo Xadrez - Server");
        primaryStage.setScene(new Scene(root));

        primaryStage.setResizable(false);



       primaryStage.setOnCloseRequest(new EventHandler<WindowEvent>() {
    @Override
    // pop-up de quando se fecha a janela do servidor
    public void handle(WindowEvent event) {
        System.out.println("hidding");
        Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
        alert.setTitle("Desligar o servidor!");
        alert.setHeaderText("Deseja mesmo desligar o servidor??");
        alert.setContentText("Se sim carregue em SIM, caso pretenda continuar NÃO...Avisamos que os jogos abertos ficaram Desconectados...");
        Toolkit.getDefaultToolkit().beep();


        //opções sim ou nao da janela pop-up
        ButtonType buttonTypeOne = new ButtonType("SIM");
        ButtonType buttonTypeTwo = new ButtonType("NÃO");



        alert.getButtonTypes().setAll(buttonTypeOne, buttonTypeTwo);

        Optional<ButtonType> result = alert.showAndWait();
        // verificar opção selecionada e se sim fechar a janela do servidor
        if ( result.get() == buttonTypeOne ) {

                server.sendChatData("Desculpe! Servidor Caiu!" ); // caso o jogador esteja em jogo e fechar a janela
                Platform.exit();
                System.exit(0);


            }else if (result.get() == buttonTypeTwo){

        }
         event.consume();
    }});

        primaryStage.show();
```

```
        Thread t = new Thread(new Runnable() {
            @Override
            public void run() {
                server.start();
            }
        });
        t.start(); //inicia o server thread
    }

    @Override
    public void onConnectionEstablish() { // quando a coneção é estabelecida iniciar o ecra principal do servidor
        try {
            startMainScreen(primaryStage);
        } catch (Exception e) {
            e.printStackTrace();
        }

    }
     //apresentar o ecra principal apos a conecção é estabelicida
    private void startMainScreen(Stage stage) throws Exception {
        MainController controller = new MainController(server);
        FXMLLoader loader = new FXMLLoader(getClass().getResource("main_screen.fxml"));
        loader.setController(controller);

        Parent root = loader.load();
        stage.getScene().setRoot(root);
        stage.show();
        stage.sizeToScene();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

Finalização do Jogo

Dificuldades

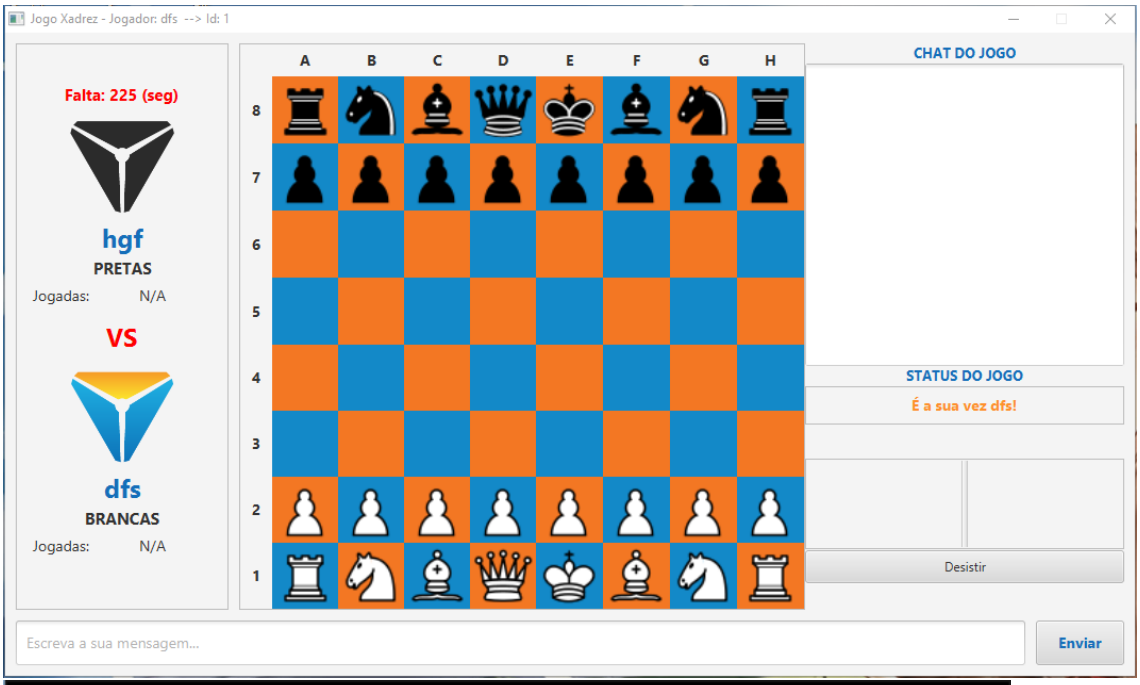
Sem dúvida foram encontradas várias dificuldades ao longo do desenvolvimento com muitos bugs encontrados a medida que se foi avançado levando assim a ter de se fazer várias alterações no código, mas cujo todas foram superadas

A parte mais difícil teria de dizer que foi a implementação das jogadas especiais, sendo que elas só podem ser efetuadas em certas condições.

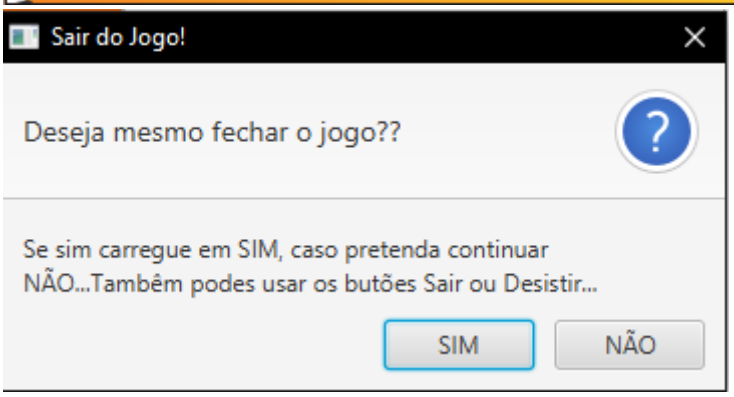
Interface

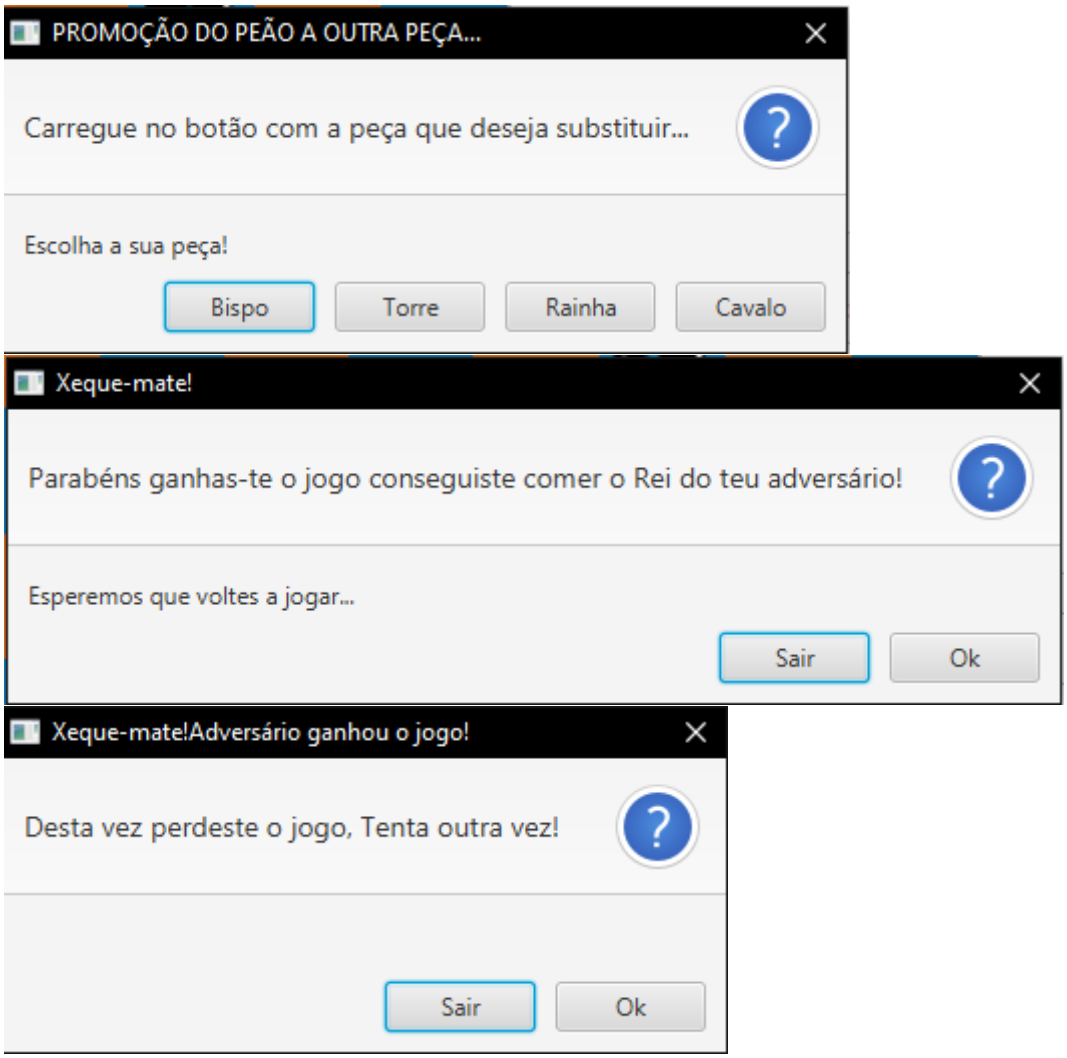

Jogo Xadrez - Server
ecgm
ENGENHARIA DA COMPUTAÇÃO GRÁFICA E MULTIMÉDIA
Espera de Jogadores....


Jogo Xadrez
ecgm
ENGENHARIA DA COMPUTAÇÃO GRÁFICA E MULTIMÉDIA
Nome Client:
Modo de Jogo:
SERVIDOR MULTYPLAYER
IP do Servidor:
Iniciar Partida!
Sair


Jogo Xadrez - Jogador: dfs --> Id: 1
Falta: 225 (seg)
hgf
PRETAS
Jogadas: N/A
VS
dfs
BRANCAS
Jogadas: N/A
A B C D E F G H
8 7 6 5 4 3 2 1
CHAT DO JOGO
STATUS DO JOGO
É a sua vez dfs!
Desistir
Escreva a sua mensagem...
Enviar


Jogo Xadrez - Server
Xadrez ECGM - Server
Jogador dfs conectado!
Jogador hgf conectado!


Sair do Jogo!
Deseja mesmo fechar o jogo??
Se sim carregue em SIM, caso pretenda continuar
NÃO...Também podes usar os butões Sair ou Desistir...
SIM
NÃO



Conclusão

Para concluir, podemos afirmar que os objetivos que foram expostos na *Descrição do Jogo* foram concluídos com sucesso. No final destas semanas de Trabalho sentimo-nos realizados apos termos conseguido cumprir este objetivo. Por vezes achamos que se iria tornar impossível poias as dificuldades neste projeto foram imensas e o tempo contrarrelógio e os vários trabalhos não ajudaram na rápida realização do mesmo.

Em conclusão foi um trabalho pratico que apresentou um enorme desafio e que nos ajudou a melhor as nossas técnicas de programador e de por a prova as matérias que aprendemos durante o semestre e a nossa capacidade de adaptamento e resolução de problemas.

Bibliografia:

https://docs.oracle.com/javase/tutorial/uiswing/events/actionlistener.html

https://docs.oracle.com/javase/tutorial/uiswing/events/listdatalistener.html

https://stackoverflow.com/questions/4818851/java-events-handlers-and-listeners-question

https://www.youtube.com/watch?v=OmSnFvv7IfQ

https://www.youtube.com/watch?v=VVUuo9VO2II

https://www.javaworld.com/article/2077351/events-and-listeners.html

https://youtu.be/WZGyP57IH6M

https://www.youtube.com/watch?v=ikkMrdT9AyY

https://www.youtube.com/watch?v=a-5Rb1_A4ew

https://www.youtube.com/watch?v=Ol2pAXgVE7c&list=PLOJzCFLZdG4zk5d-1_ah2B4kqZSeIlWtt&index=2

https://www.youtube.com/watch?v=0-xL405eNwM

https://www.youtube.com/watch?v=9vz-Dcdl8JA

https://www.youtube.com/watch?v=gN6VzPSFgBs

https://ntnuopen.ntnu.no/ntnu-xmlui/bitstream/handle/11250/2504707/17809_FULLTEXT.pdf?sequence=1&isAllowed=y

https://docs.oracle.com/javafx/2/drag_drop/jfxpub-drag_drop.htm

https://www.genuinecoder.com/drag-and-drop-in-javafx-html/

https://www.youtube.com/watch?v=jbe4vKUxo90

https://examples.javacodegeeks.com/desktop-java/javafx/event-javafx/javafx-drag-drop-example/

(Alguns dos links que estavam no meu histórico e que nos baseamos para fazer o jogo)

Plataformas onde se encontra o Projeto disponível

- Google Drive:
  https://drive.google.com/drive/folders/1fw3V0YN6IjiKPqzCiGXbHBlfGxoXcy3c?usp=sharing

- Github:

  https://github.com/diogomaa/CHESS_ECGM_LABPROG_GAME

  https://github.com/diogomaa/CHESS_ECGM_LABPROG_SERVER