Names: Omkar Bhalerao, Aneesh Boreda, Denaly Min

Project Outline: We plan to implement our chess game with several java classes. We have an abstract class, Piece, which all other piece classes extend. We also have a chessboard class which ties everything together. The chess panel takes care of buttons and graphics, while the driver runs everything.

Division of Labor: In the beginning, with not much time to work and other focuses in this class and in school, labor was not strictly divided. Whoever had a good idea or time to work simply did what they could. We are working on finding our strengths and will lay out a plan as we move along and become more familiar with what we need to do. As of now, Aneesh created the structure and set up the chess piece icons while he and Omkar worked together on implementing the logic for each chess piece. Denaly and Omkar wrote up a list of rules for our instructions panel and created the classes to display this second window. Debugging programs is likely the most time-consuming part of this project, and all three of us share in this task.

Describes the purpose: Explains the purpose and identifies special factors that influence the solution.

Gives Examples: Makes up examples and explains why they are relevant.

Header: Correct.

Algorithm: Explains how the algorithm will work with diagrams or pseudo-code.

Problem: Our purpose is to code a game of chess, adding a few of our own features such as background music and an instructions panel, as well as making sure we have accounted for all the rules of chess. We want to make it easy to play, recognizing all legal moves and not allowing any illegal moves.

Examples: For the special move called “castling” to occur in chess, a very specific set of conditions must be met. Our program will ensure that castling cannot occur illegally. This means it will check to make sure that neither the king nor the involved rook has moved before, that no pieces are in the way of the king or rook, and that the king will not be in check before, during, or after castling.

Another special move, en passant, is a special type of capture performed by pawns. If a pawn moves two spaces forward on its first move to evade capture by an opposing pawn which it is now next to, the second pawn can take the first pawn on the next move only, not on any subsequent moves. Our program will ensure that this complicated move is performed correctly.

Header:

public class ChessDriver{

public static void main(String[] args) throws IOException{}

}

public class Piece{

public Piece(int a, int b, int col){}

public abstract int[][] getLegalSquares();

public abstract ImageIcon getImage();

public abstract String getName();

public abstract void addSquare(int a, int b, int num);

public abstract void clearSquares();

public void move(int a, int b){}

}

public class Pawn{

public Pawn(int a, int b, int color){}

public int[][] getLegalSquares(){}

public ImageIcon getImage(){}

public String getName(){}

public void addSquare(int a, int b, int num){}

public void clearSquares(){}

}

public class Knight{

public Knight(int a, int b, int color){}

public int[][] getLegalSquares(){}

public ImageIcon getImage(){}

public String getName(){}

public void addSquare(int a, int b, int num){}

public void clearSquares(){}

}

public class Bishop{

public Bishop(int a, int b, int color){}

public int[][] getLegalSquares(){}

public ImageIcon getImage(){}

public String getName(){}

public void addSquare(int a, int b, int num){}

public void clearSquares(){}

}

public class Rook{

public Rook(int a, int b, int color){}

public int[][] getLegalSquares(){}

public ImageIcon getImage(){}

public String getName(){}

public void addSquare(int a, int b, int num){}

public void clearSquares(){}

}

public class Queen{

public Queen(int a, int b, int color){}

public int[][] getLegalSquares(){}

public ImageIcon getImage(){}

public String getName(){}

public void addSquare(int a, int b, int num){}

public void clearSquares(){}

}

public class King{

public King(int a, int b, int color){}

public int[][] getLegalSquares(){}

public ImageIcon getImage(){}

public String getName(){}

public void addSquare(int a, int b, int num){}

public void clearSquares(){}

}

public class Chessboard{

public Chessboard(){}

public static Piece[][] Input(int a, int b){}

public static Piece getPiece(int a, int b){}

}

public class ChessPanel{

public ChessPanel(){}

public static void updateBoard(){}

private class Listener implements ActionListener{

public Listener(int a, int b){}

public void actionPerformed(ActionEvent e){}

}

}

public class RulesPanel{

public RulesPanel() throws IOException{}

private class PrevListener implements ActionListener{

public PrevListener(){}

public void actionPerformed(ActionEvent e){}

}

private class NextListener implements ActionListener{

public NextListener(){}

public void actionPerformed(ActionEvent e){}

}

public void doStuff(){}

}

Algorithm:

Our chessboard consists of an 8x8 matrix of JButtons, with different ImageIcons which represent chess pieces. We also have an 8x8 matrix containing 32 Piece objects, which represent the different pieces and their locations on the board. On every move, the function updateBoard() is called, which finds all the legal moves all the pieces can make. Then, when a piece is selected, all the legal squares for the piece to move are highlighted green. Our program will not allow the piece to move to any square that is not highlighted, and clicking on the piece a second time allows for it to be deselected. It will also detect checks, castling opportunities, and en passant captures, while checking for legal squares to move to, and end the game when checkmate or stalemate is detected. In addition to our chess window we also have a window with basic instructions on how to play chess. This is displayed above buttons which can be used to navigate to the next and previous pages, as chess has many rules, some of which are quite complicated.