**Requirement document**

# The program is based on the chess game, the basic rules are, in chess, each player takes turns to make a single move. Players cannot choose to skip a turn - they must move a piece. And each piece moves in a particular way, and must be moved according to its legal movement. Lastly, the winning condition is the King of one side is captured or the contestants agree to a draw issue. Each player has 16 pieces, 6 types of piece. They are 1 King, 1 Queen, 2 Rooks, 2 Bishops, 2 Knights and 8 Pawns. The pawn moves vertically forward one square, with the option to move two squares if they have not yet moved. Pawns are the only piece to capture different to how they move. The pawns capture one square square diagonally in a forward direction. The knight moves in an ‘L’ shape, two square in a horizontal or vertical direction, then moves one square horizontally or vertically. They are the only piece able to jump over other pieces. Bishops move diagonally any number of squares. They are unable to jump over pieces. Rooks move horizontally or vertically any number of squares. They are unable to jump over pieces. Rooks move when the king castles. Queens move diagonally, horizontally or vertically any number of squares. They are unable to jump over pieces. Kings move one square in any direction, so long as that square is not attacked by an enemy piece. Additionally, kings are able to make a special move, known as castling. Once the program is running, it will show a chess board with chess pieces and have the title “Chess Game”. There will be two options on the top of the sub-menu, “game” and “help”. In the “game” option, it includes the restart, undo, and pass. Restart is to restart the whole game, undo is to recover the last movement, and pass is to pass the user’s current turn. In the “help” option, it includes the “rules”, “movement”. The “rules” is to show the user the legal movement of chess pieces, and the “movement” is to tell the user the chess piece's possible movement. When the users Select piece state: they cannot select a blank square, cannot select opponent’s pieces, cannot select pieces that are illegal to move.

**CRC card**

| Class Name:Piece |  |
| --- | --- |
| Responsibility:  Able to get and set the previous position, get and set the player’s own. Set and get the row and column, check the range. Movement of the rook and king is in check or not. Calculate the legal moves,getAlliance | Collaborator:  Row, column, board, move, myPlayer, enemyPlayer, positionHistory, board, isInCheck |

| Class Name: Board |  |
| --- | --- |
| Responsibility:  Able to display the chess board and when the user uses the reset function, it is able to print the board again. Able to check the game status, check if the game is over. Able to clear the board. Check AI. | Collaborator:  Board size, game, GUI, chessboard[] [],tilePrinter, aiRuning, selectTile |

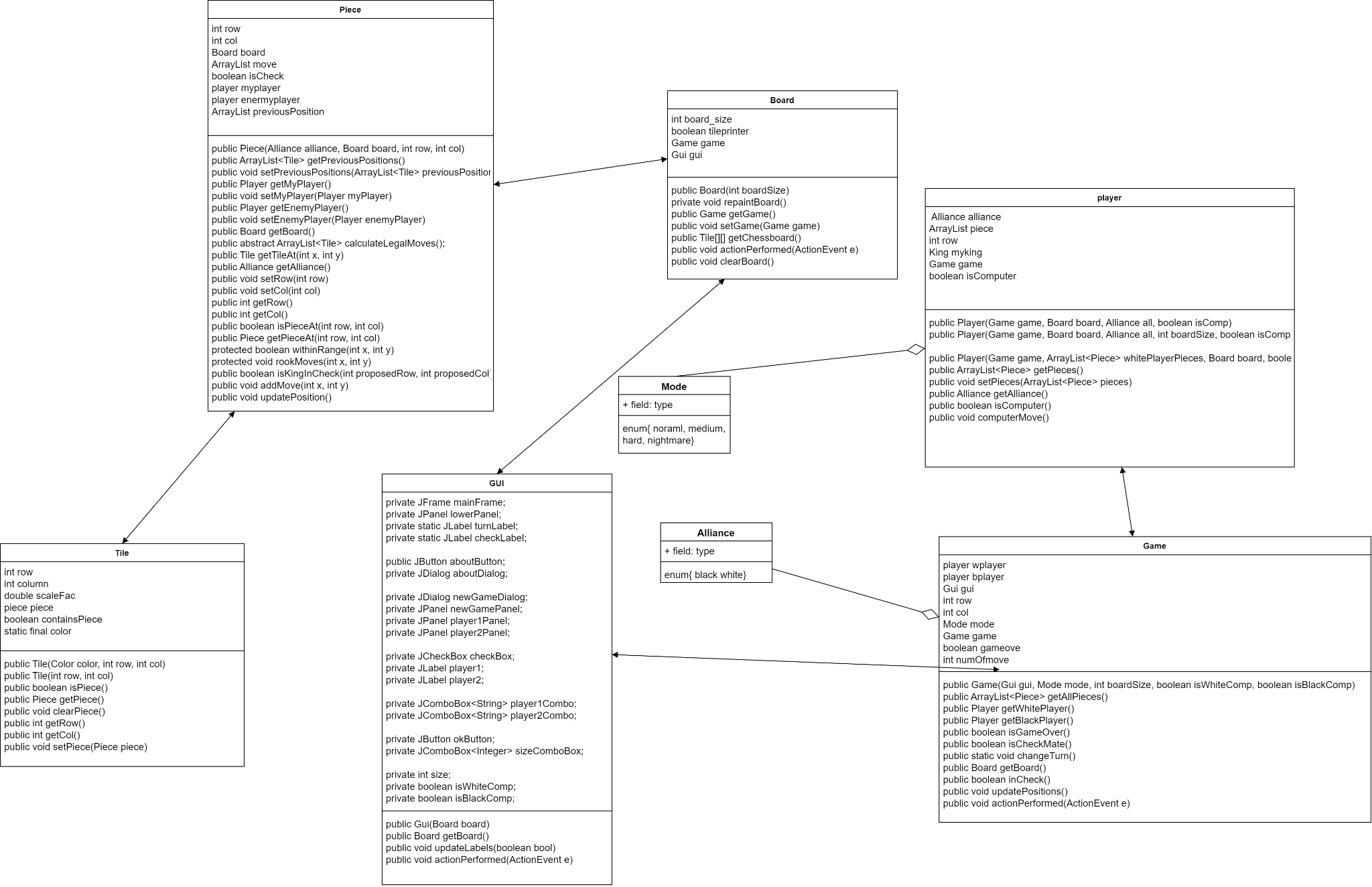
| Class Name: Tile |  |
| --- | --- |
| Responsibility:  Able to give the color to the board and pieces, and give the legal move range with color. Able to upload the image of the pieces. Able to get the row, column and piece, also able to clear the pieces. | Collaborator:  Row, column, piece, color |

| Class Name: player |  |
| --- | --- |
| Responsibility:  Have four different constructors, normal, medium, hard and nightmare modes. Able to get and set the piece, get the alliance. Able to check the movement of the computer and check if the player is computer or not. | Collaborator:  Alliance, row, king, isComputer, piece |

| Class Name: game |  |
| --- | --- |
| Responsibility:  Construct the chess game with GUI, different game modes, default size of the board, and the player is human or AI. able to get the piece, and get he white or black player. Able to check the checkmate, pass the turn. | Collaborator:  Player, gui, row, column, mode |

| Class Name:GUI |  |
| --- | --- |
| Responsibility:  Display the chess game, with piece, board, menu and statue panel. Users are able to use the menu, reset and undo. The About button can show the description of the game. | Collaborator:  JFrame. JPanel, JButton, JDialog, JLabel, size, wComputer, bComputer |

**UML Diagram**



**Javadoc class documentation**

/\*\*

\* This class is to set up the pieces, assign color, position

\* Check the legal moves, player or enemy player

\* @author GuFX

\*

\*/

public abstract class Piece

/\*\*

\* This class displays the chess board, which can be printed again when the user uses the reset function.

\* You can view the game state to see if the game is over. Clear the board. Check the AI.

\* @author GuFX

\*

\*/

public class Board extends JPanel implements ActionListener

/\*\*

\* This class can color the board and pieces, and assign a legitimate range of movement and color.

\* Tiles can also upload an image of a piece, get rows, columns, and tiles, and clear pieces

\* @author GuFX

\*

\*/

public class Tile extends JButton

/\*\*

\* This class is to create the buttons, displays chess games, pieces, boards, menus, game mode and statue panels.

\* Users can use the menu (reset and undo). "About" button can display a description of the game.

\* @author GuFX

\*

\*/

public class Gui implements ActionListener

/\*\*

\* This class has four different constructors, normal, Medium, difficult,

\* and Nightmare modes (mode can only be selected if the opponent is AI).

\* The ability to pick up and place pieces to gain an alliance (black or white).

\* The ability to determine if the player is a computer and control the computer's actions based on the player's actions.

\* @author GuFX

\*

\*/

public class Player

\*\*

\* This class builds a chess game using a GUI that allows you to select different game modes,

\* assign default values to the board size, and determine whether a player is playing black or white.

\* The ability to determine if the game is over by checking is in check,

\* and the ability to update the Status Panel by turn when the player finishes moving pieces.

\* @author GuFX

\*

\*/

public class Game implements ActionListener

**Javadoc method documentation**

**Piece.java**

/\*\*

\* construct the piece with default color, position

\* @param alliance color

\* @param board chess board

\* @param row row

\* @param col column

\*/

public Piece(Alliance alliance, Board board, int row, int col)

/\*\*

\* get the previous position

\* @return last position

\*/

public ArrayList<Tile> getPreviousPositions()

/\*\*

\* set the row

\* @param row

\*/

public void setRow(int row)

/\*\*

\* set the column

\* @param col

\*/

public void setCol(int col)

/\*\*

\* get the row

\* @return

\*/

public int getRow()

/\*\*

\* get the column

\* @return

\*/

public int getCol()}

/\*\*

\* set the previous position

\* @param previousPositions previous position

\*/

public void setPreviousPositions(ArrayList<Tile> previousPositions)

/\*\*

\* get the user's player

\* @return own player

\*/

public Player getMyPlayer()

/\*\*

\* set the own player

\* @param myPlayer

\*/

public void setMyPlayer(Player myPlayer)

/\*\*

\* get enemy's player

\* @return

\*/

public Player getEnemyPlayer()

/\*\*

\* set enemy's player

\* @param enemyPlayer

\*/

public void setEnemyPlayer(Player EnemyPlayer)

/\*\*

\* calculate the legal moves

\* @return the number of the legal moves

\*/

public abstract ArrayList<Tile> calculateLegalMoves();

/\*\*

\* get the board

\* @return chess board

\*/

public Board getBoard()

/\*\*

\* get the tile's position

\* @param x row

\* @param y column

\* @return position of the tile

\*/

public Tile getTileAt(int x, int y)

/\*\*

\* get the alliance(black or white)

\* @return alliance

\*/

public Alliance getAlliance()

/\*\*

\* check the position of the piece

\* @param row

\* @param col

\* @return true if piece is at that position, false then not

\*/

public boolean isPieceAt(int row, int col)

/\*\*

\* get the position of the piece

\* @param row

\* @param col

\* @return position of the piece

\*/

public Piece getPieceAt(int row, int col)

/\*\*

\* get the color of the piece at specific position

\* @param row

\* @param col

\* @return position and color of the piece

\*/

public Alliance getAllianceAt(int row, int col)

/\*\*

\* set the rook's move

\* @param x

\* @param y

\*/

protected void rookMoves(int x, int y)

/\*\*

\* check the status is in check or not

\* @param proposedRow

\* @param proposedCol

\* @return true - in check, false - not in check

\*/

public boolean isKingInCheck(int proposedRow, int proposedCol)

/\*\*

\* add move if is not in check

\* @param x

\* @param y

\*/

public void addMove(int x, int y)

/\*\*

\* update the position, change the tile

\*/

public void updatePosition()

**Board.java**

/\*\*

\* constructor, print the board with default size

\* @param boardSize

\*/

public Board(int boardSize)

/\*\*

\* repaint the board, call it when need to undo or reset

\*/

private void repaintBoard()

/\*\*

\* get the game

\* @return game

\*/

public Game getGame()

/\*\*

\* set the game

\* @param game

\*/

public void setGame(Game game)

/\*\*

\* get the position of the board

\* @return position of the board

\*/

public Tile[][] getChessboard()

/\*\*

\*check the game status,

\*/

@Override

public void actionPerformed(ActionEvent e)

/\*\*

\* clear the board,

\*/

public void clearBoard()

/\*\*

\* set up the GUI

\* @param gui

\*/

public void setGui(Gui gui)

**Game.java**

// Game constructor takes a GUI, game mode, size of board and if players are

// computers or humans

public Game(Gui gui, Mode mode, int boardSize, boolean isWhiteComp, boolean isBlackComp)

/\*\*

\* get all the piece

\* @return all piece in arraylist

\*/

public ArrayList<Piece> getAllPieces()

/\*\*

\* get the white piece player

\* @return white player

\*/

public Player getWhitePlayer()

/\*\*

\* get the black piece player

\* @return black player

\*/

public Player getBlackPlayer()

/\*\*

\* check the game is over or not

\* @return game over or not

\*/

public boolean isGameOver()

/\*\*

\* check the check mate status

\* @return check mate or not

\*/

public boolean isCheckMate()

/\*\*

\* change the turn when player finish one move

\*/

public static void changeTurn()

/\*\*

\* this method is to check whether to update the labels or not

\* @return true update, false haven't update

\*/

public boolean inCheck()

/\*\*

\* update the position

\*/

public void updatePositions()

/\*\*

\*undo function

\*/

@Override

public void actionPerformed(ActionEvent e)

/\*\*

\* get the type of the mode

\* @return type of the mode

\*/

public Mode getMode()

**Tile.java**

/\*\*

\* this constructor is to make 'ghost' tiles that captured pieces live on

\* @param row

\* @param col

\*/

public Tile(int row, int col)

/\*\*

\* check if it contains the piece

\* @return true contain, false not

\*/

public boolean isPiece()

/\*\*

\* get the piece

\* @return piece

\*/

public Piece getPiece()

/\*\*

\* clear the piece and the the contains piece to false

\*/

public void clearPiece()

/\*\*

\* get the row

\* @return row

\*/

public int getRow()

/\*\*

\* get the column

\* @return column

\*/

public int getCol()

/\*\*

\* set the piece, upload the image

\* @param piece

\*/

public void setPiece(Piece piece)

**Player.java**

/\*\*

\* constructor for normal chess

\* @param game

\* @param board

\* @param all alliance

\* @param isComp check if it is computer

\*/

public Player(Game game, Board board, Alliance all, boolean isComp)

/\*\*

\* this is the constructor for medium chess

\* @param game

\* @param board

\* @param all

\* @param boardSize

\* @param isComp

\*/

public Player(Game game, Board board, Alliance all, int boardSize, boolean isComp)

/\*\*

\* this is the constructor for hard chess

\* ONLY TO GENERATE BLACK PIECES!

\* @param game

\* @param whitePlayerPieces

\* @param board

\* @param isComp

\*/

public Player(Game game, ArrayList<Piece> whitePlayerPieces, Board board, boolean isComp)

/\*\*

\* this is the constructor for nightmare mode chess

\* @param game

\* @param all

\* @param board

\* @param isComp

\*/

public Player(Game game, Alliance all, Board board, boolean isComp)

/\*\*

\* get the piece

\* @return piece list

\*/

public ArrayList<Piece> getPieces()

/\*\*

\* set the piece

\* @param pieces

\*/

public void setPieces(ArrayList<Piece> pieces)

/\*\*

\* get the alliance

\* @return

\*/

public Alliance getAlliance()

/\*\*

\* check if it's computer

\* @return true->computer, false->human

\*/

public boolean isComputer()

/\*\*

\* set the computer's move

\*/

public void computerMove()

**GUI.java**

/\*\*

\* set up the board with menu, piece, board and status panel

\* @param board

\*/

public Gui(Board board)

/\*\*

\* get the board

\* @return board

\*/

public Board getBoard()

/\*\*

\* update the status panel, white turn or black turn

\* check or not

\* @param bool check the turns

\*/

public void updateLabels(boolean bool)

/\*\*

\* new Game(reset) function, and description of the game

\*/

@Override

public void actionPerformed(ActionEvent e)

/\*\*

\* set the game

\* @param game

\*/

public void setGame(Game game)

**Test plan**

There are four tests, chess, board, path and all tests. For the test piece, I want to set two pieces and several pieces for the test. The safe board has many legal actions, while the unsafe board is in the check state and has no legal actions. For board testing, I'll set up 3 boards and a few for testing; Safety board, unsafe board, danger board. The first one, as before, has a lot of legal actions, the second one is in check, so no legal actions, and the last one is in stalemate, and no legal actions. Path test: The default value of the board. It checks paths in the vertical, horizontal, and diagonal directions. The last test is the test of each type of chess piece. The default value is set for different types of chess pieces to judge whether the move conforms to a legal move.

**User guide**

Open the project -> Click the src folder -> gameplay folder -> Main.java, then you can run the program. Once the program runs it will show a “New Game” window, since I didn’t finish the AI opponent function you can just click “ok”, then you can play the game. There is a status panel on the bottom, you can know the current turn belongs to white or black. On the top there is a menu bar, three options, first one is “about”, description of the game; second one is “new game” reset the game; third one is “undo” you can go back to your last move.