Software Engineering Group 18 Project
Design Specification

Authors:	Jasper Crabb [jac127], Muntazir Rashid [mrm19], Archie Malvern [arm36], Jason Smith [jas160], William Parry [wip24], Kacper Dziedzic [ktd1]
Config Ref:	DSGroup18
Date:	23/03/2023
Version:	1.5
Status:	Release

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB

CONTENTS

1. INTRODUCTION

1.1 Purpose of this Document

The purpose of this document is to describe how we have designed our chess tutor. This document will do this by showing relationships between the requirements, classes, methods and algorithms and how they combine into our final product.

1.2 Scope

This document specifies each aspect of our program and how they work in our final product.

This follows the standards laid down by the Design Specification Standards [1].

This document should be read by all project members. Before reading this document, the reader should familiarise themselves with the UI Specification [2].

1.3 Objectives

The objective of this document is to show how each component of our program interacts to create the final product.

2. DECOMPOSITION DESCRIPTION

2.1 Programs in System

Our system will only have one program in it. This program is structured into classes which handle graphics (using JavaFx) and the logic of the chess game. The JavaFx classes use event handlers to manage user input and we use JavaFx files for the layout.

2.2 Significant Classes

Our program has many classes but there are some significant classes that are detailed below:

- Main:
    - This class is the entry point to the application. It creates a menu object for the main menu and from here a Board object is created, which means that we can display and play the game.
- Board:
    - This class contains a 2d array of tiles and acts as a platform that ties the logic aspects of the game together. It also initialises the pieces on the chess board, whether that be a saved layout (Using the FenConverter class) or the standard starting position.
- Game:
    - The game class manages the current state of the game such as which player's turn it is, whether the king is in check and executing turns after player input.
- ChessBoard:
    - This is the main JavaFX class and contains all the methods that manage displaying the board, pieces and game. It also manages player input e.g., clicking on the board.

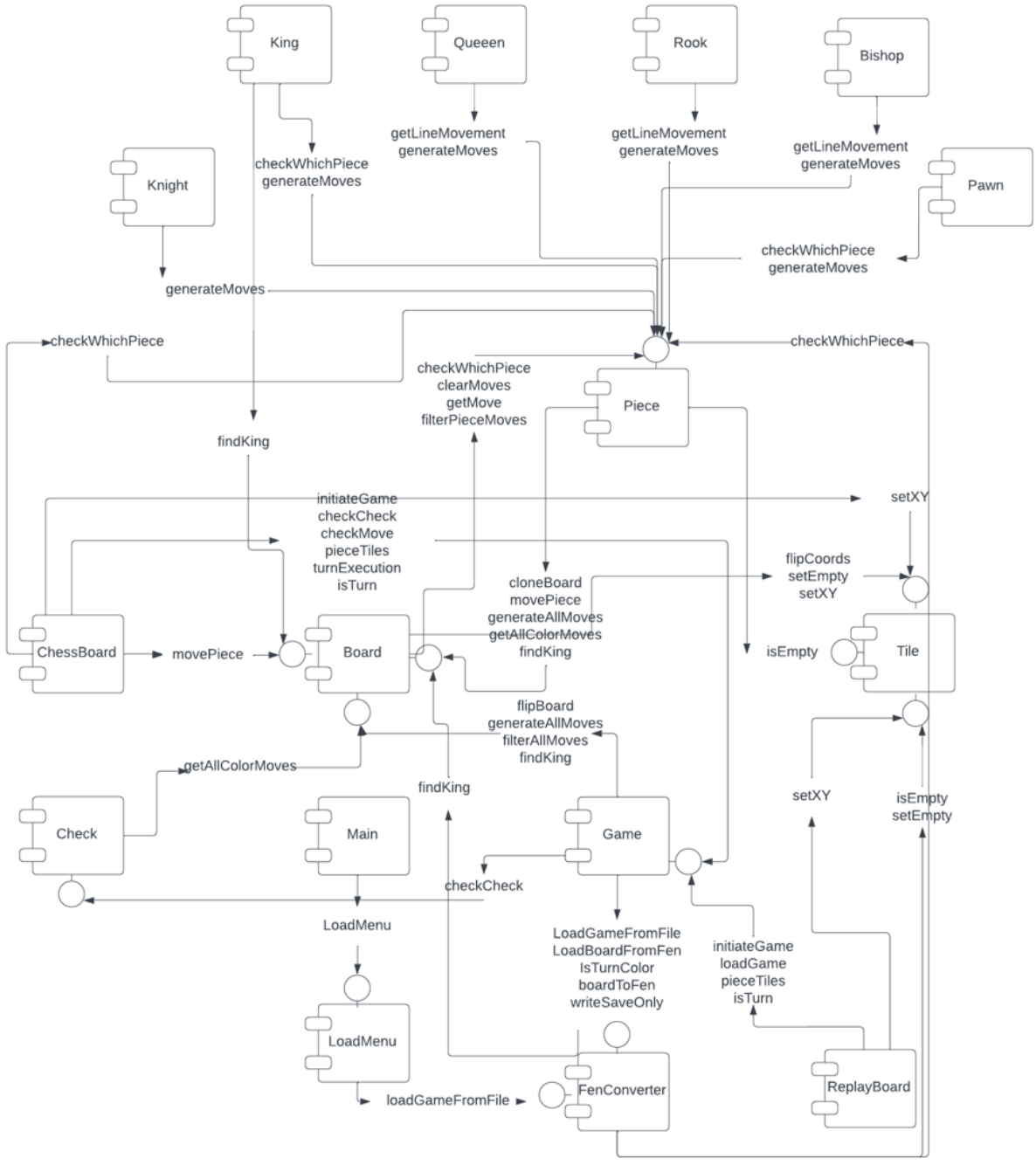
- Tiles:
    - The tiles class is used to store the chess pieces alongside their positions. It allows for us to flip the coordinates of the board without modifying the piece array. This is needed so that we can apply the piece move algorithms to both sides of the board.
- FenConverter:
    - This class is used to load and make FEN strings for the game state. It has methods which can be used to load multiple FEN strings from files and methods that can take a Board object and convert it into a FEN string based on the position of the pieces.
- Piece abstract class:
    - This class acts as a template for all the other piece classes, which are for specific pieces. It has methods for valid move generation and a way to store each valid move in an arrayList .

2.3 Table Mapping Requirements onto Classes

Requirement	Classes providing requirement
FR1	Main, PlayerMenu, FenConverter
FR2	PlayerMenu, ChessBoard, Board, Tile
FR3	ChessBoard, Tile, Game
FR4	ChessBoard, Tile, Game
FR5	ChessBoard, Board Tile, Piece, Bishop, Pawn, King, Queen, Knight, Rook, Check, Game
FR6	ChessBoard, Board, Tile, Check, Game
FR7	ChessBoard, Board, Tile, Check, Game
FR8	ChessBoard, FenConverter, Game
FR9	ChessBoard, FenConverter, Game
FR10	Main, FenConverter, ChessBoard, ReplayMenu
FR11	Main, FenConverter, ChessBoard

3. DEPENDENCY DESCRIPTION

3.1 Component Diagrams



4. INTERFACE DESCRIPTION

4.1 Piece interface specification

Type: public abstract
Extends: nothing
Public methods:

- Piece(boolean colour): The constructor initializes the colour and moves ArrayList for the piece.
- Piece(): An empty constructor.
- ArrayList<Tile> getAllMoves(): Returns all moves available for the piece.
- void generateMoves(Board board, Tile piece): Generates all moves available for the piece on the board.
- int checkForPiece(Tile tile): Checks the tile for a piece and returns a value representing whether it is empty, occupied by an enemy, occupied by a friendly piece, or out of bounds.
- int checkWhichPiece(Tile tile): Returns a value representing which piece is located on the tile.
- ArrayList<Tile> getAllMoves(): Returns all the moves available for the piece.
- void clearMoves(): Clears all the moves for the piece.
- void setMoves(ArrayList<Tile> moves): Sets the moves for the piece.
- Tile getMove(int x, int y): Returns a specific move the piece can do, or null if it does not have that move.
- void removeAllMoves(ArrayList<Tile> moves): Removes all the moves of a piece by comparing the x and y of the move against the moves sent through the parameter "moves".
- void filterPieceMoves(Board board, Tile piece): Filters all the illegal moves from every possible moves of a piece.
- ArrayList<Tile> lineMovement(Tile piece, Board board, int m, int n): Gets all the straight moves in a chosen direction until it reaches the edge of the board or another piece
- boolean getColour(): Returns the colour of the piece.
- void setColour(boolean colour): Sets the colour of the piece.
- String getPieceName(): Returns the name of the piece.
- void setPieceName(String pieceChar): Sets the name of the piece.

4.2 Bishop interface specification

Type: public

Extends: Piece

Public methods:

- Bishop(boolean colour): The constructor initializes the bishop piece with its corresponding color.
- void generateMoves(Board board, Tile piece): Generates all moves available for the bishop on the board.

4.3 King interface specification

Type: public

Extends: Piece

Public methods:

- King(boolean colour): The constructor initializes the king piece with its corresponding colour.
- King(boolean color, boolean hasMoved): The constructor initializes the king piece with its corresponding colour and hasMoved flag.
- boolean hasMoved(): Returns true if the king has moved, false otherwise.
- void setHasMoved(boolean hasMoved): Sets the hasMoved flag of the king.
- void generateMoves(Board board, Tile piece): Generates all moves available for the king on the board.
- ArrayList<Tile> castlingMoves(Board board, Tile piece): Returns valid castling moves for the king on the board.
- boolean checkCastlingValidity(Board board, Tile move, Tile king): Checks if the king can castle to the specified tile.

4.4 Pawn interface specification

Type: public

Extends: Piece

Public methods:

- Pawn(boolean colour): The constructor initializes the pawn piece with its corresponding colour and move status.
- Pawn(boolean colour, int moveStatus): Constructor used for testing purposes.
- void updateMoveStatus(): Updates the move status of the pawn.
- void setMoveStatus(int moveStatus): Sets the move status to whatever the pawn has done.
- int getMoveStatus(): Returns the move status.
- boolean doubleMoveLast(): Returns true if the pawn performed a double move in its last turn.
- void generateMoves(Board board, Tile piece): Generates all moves available for the pawn on the board.
- ArrayList<Tile> enPassantMoves(Board board, Tile piece, int direction): En passant method which completes all the pre-requisite checks to see if en passant can be done on either the left or right square.

4.6 Queen interface specification

Type: public

Extends: Piece

Public methods:

- Queen(boolean colour): The constructor initializes the queen piece with its corresponding colour.
- void generateMoves(Board board, Tile piece): Generates all moves available for the queen on the board.

4.7 Rook interface specification

Type: public

Extends: Piece

Public methods:

- Rook(boolean colour): The constructor initializes the rook piece with its corresponding colour.
- Rook(boolean color, boolean hasMoved): The constructor initializes the rook piece with its corresponding colour and hasMoved flag.
- void generateMoves(Board board, Tile piece): Generates all moves available for the rook on the board.
- boolean hasMoved(): Returns true if the rook has moved.
- void setHasMoved(boolean hasMoved): Sets the hasMoved variable of the rook.

4.8 Knight interface specification

Type: public

Extends: Piece

Public methods:

- Knight(boolean colour): The constructor initializes the rook piece with its corresponding colour.
- void generateMoves(Board board, Tile piece): Generates all moves available for the Knight on the board.

4.9 EmptyPiece interface specification

Type: public

Extends: Piece

Public methods:

- EmptyPiece(): The constructor initializes an empty piece.
- void generateMoves(Board board, Tile piece): Generates no moves for an empty piece.

4.9 Tile interface specification

Type: public

Extends: nothing

Public methods:

- Tile(int x, int y, Piece piece): The constructor initializes the tile with its coordinates and the piece on it.
- Tile(int x, int y): An empty constructor that initializes the tile with its coordinates and an empty piece.
- Piece getPiece(): Returns the piece on the tile.
- void setPiece(Piece piece): Sets the piece on the tile.
- void setXY(int x, int y): Sets the x and y - coordinates of the tile.
- int getX(): Returns the x-coordinate of the tile.
- void setX(int x): Sets the x-coordinate of the tile.
- int getY(): Returns the y-coordinate of the tile.
- void setY(int y): Sets the y-coordinate of the tile.
- void flipCoords(): Flips the coordinates of the tile.
- boolean isEmpty(): Returns true if the tile is empty, false otherwise.
- void setEmpty(boolean empty): Sets the tile to be empty or not.
- getMoveType(): Returns the move type of the piece on the tile, or 0 if the tile is empty.
- setMoveType(moveType): Sets the move type of the piece on the tile.

4.10 Main interface specification

Type: public

Extends: Application

- This is required by JavaFX

Public methods:

- void main(String[] args): The entry point for the application. Launches the application with the specified arguments.
- void start(Stage stage): The main method that initializes and starts the application. Creates a menu screen with buttons for starting, loading, replaying, and quitting a game.

4.11 ChessBoard class interface specification

Type: public

Extends: Application

- This is required by JavaFX

Public methods:

- ChessBoard(Stage parentStage, String nameOne, String nameTwo, boolean turn): Constructor of the object ChessBoard.
- ChessBoard(Stage parentStage, String fenSave, String nameOne, String nameTwo, boolean turn): Constructor of the object ChessBoard and loads the fenstring file.
- void start(Stage stage): Sets up the scene and shows the board on the stage.
- void createBoard(Pane root): Creates and draws the board.
- void customPiecesToBoard(Pane root, ArrayList<Tile> tileCoords): Loads a custom piece position to the board.
- void initText(Pane root, String playerA, String playerB): Initiates and writes text to window.
- void addDot(Pane root, int x, int y): adds dot to a specified location
- int convertCoordsX(int x): Converts a piece X coordinate into an X coordinate for drawing to a pane.
- int convertCoordsY(int y): Converts a piece Y coordinate into an Y coordinate for drawing to a pane.
- void removeCheckSquare(Pane root): Changes a square back to its original color when no longer in check.
- void removeSelectedSquare(int x, int y): Removes the selected square from the board.
- void removeDots(Pane root): Removes all the dots from the screen using a variable.
- void selectedPiece(Pane root, int x, int y): executes a move based on the inputs, updates the GUI, and checks for check and checkmate.

- void turnUpdates(Pane root, int lastX, int lastY, int tileX, int tileY, int mouseX, int mouseY): Executes a move and updates the board.
- void checkAndCheckmate(): Checks if there is a check or checkmate, and ends the game if there is.
- void drawAllDots(Pane root, int x, int y): Draws all possible moves for a piece on the board.
- void checkSquare(): Draws a check square on the board.
- void addButtons(Pane root): Adds the three buttons to the board: draw, resign, and exit.
- updatePiece(Pane root, int x, int y): Updates the piece on the board.
- void updateTurn(Pane root, String text): Updates the turn text.

Private methods:

- void removePieces(Pane root): Removes pieces from the board and draws them in their new positions.
- void drawPiece(Pane root, String imageName, int x, int y): Draws a specified piece on a specified square.

4.12 Check interface specification

Type: public
Extends: nothing
Public methods:

- void checkCheck(Tile originalTile, Board originalBoard): This method checks all the check condition on a king, and where said check is a checkmate or not.

4.13 PlayerMenu class interface specification

Type: public
Extends: Application

- This is required by JavaFX

Public methods:

- void start(Stage primaryStage): This method draws the player menu and sets it up to handle user input.

Fields:

- TextField player1TextField: The text field where the user enters the name of Player 1.
- TextField player2TextField: The text field where the user enters the name of Player 2.
- CheckBox blackBox: The checkbox that Player 1 can select to play as Black.
- CheckBox whiteBox: The checkbox that Player 1 can select to play as White.

4.14 LoadMenu class interface specification

Type: public
Extends: Application

- This is required by JavaFX

Public methods:

- void start(Stage primaryStage): This method draws the load menu for both Load and Replay functions and sets it up to handle user input.

Fields:

- File selectedFile: The file that was selected by the user.
- TextField fenStringTextField: The text field where the user enters the FEN string of the game.
- String title: The title of the menu.
- boolean whichMenu: A Boolean value that indicates whether the load menu is for loading a game or a replay.

4.15 ExitMenu class interface specification

Type: public
Extends: Application
- This is required by JavaFX

Public methods:
- void start(Stage primaryStage): This method draws the exit menu and sets it up to handle user input.

Fields:
- String message: The message that is displayed to the user.

Additional Notes:
- The ExitMenu class is used to handle the window prompt that is displayed to the user when they select either "Resign", "Draw", or "Exit" during their turn in the middle of the game.

4.16 Board interface specification

Type: public
Extends: nothing
Public methods:
- public Board(): constructor for the chess board class. Creates a new board and fills it with empty tiles.
- void addPiece(int x, int y, Piece piece): Add individual pieces to the board for testing
- void initializeBoard():Fills the board's tiles with pieces in the starting position for a game of chess.
- Tile getTile(int x, int y): Returns a tile from the 2d array of tiles, based on its index.
- void setTile(int y, int x, Tile newTile): Sets a tile to a specified place in the 2d array of tiles.
- Tile getActualTile(int x, int y): Returns a tile from the 2d array of tiles, based on its stored coordinates.
- void displayBoard(): Outputs a textual representation of the board.
- Board cloneBoard(): creates an exact copy of the board so that we can perform checks for various methods.
- void flipBoard(): iterates over the entire board to flip all coordinates, then sets the flipped variable.
- void generateMostMoves(): Generates all possible moves for all pieces on the board.
- void simpleMove(int firstX, int firstY, int destX, int destY): Moves the piece at the specified coordinates to the specified destination.
- void movePiece(int firstX, int firstY, int destX, int destY): Moves the piece at the specified coordinates to the specified destination, handling castling and en passant.
- void specificPieceUpdate(Tile currentPiece, int move): This method updates the current status of certain pieces like Pawn Rook and King.
- void executeCastling(Tile king, Tile rook): Executes a castling move.
- void executeEnPassant(int x, int y): Executes an en passant move.
- void generateAllMoves(): generates all the possible moves a piece can have at player's current turn.
- void filterAllMoves(): filters all the moves of the pieces across the board based on whether a move puts your own king into check.
- ArrayList<Tile> getAllColorMoves(boolean color): Gets a list of all possible moves for all pieces of the specified color.
- Tile findKing(boolean color): Gets the tile containing the king of the specified color.
- void setFlipped(boolean flipped): Sets the flipped flag.
- boolean isFlipped() : check whether the board is flipped or not.

4.17 Game interface specification

Type: public
Extends: nothing
Public methods:
- EmptyPiece(): The constructor initializes an empty piece.

- Game(String Player1, String Player2) : constructor for the game class. Creates a Board and Check object.
- Game(String directory, String Player1, String Player2) : constructor for the game class. Creates a Board and Check object. Also gets the location of the fen string file as a parameter.
- Game(String directory): constructor for the game class. Gets the location of the fen string file as a parameter.
- void initiateGame(): Initializes the game and displays the board.
- void loadGame(String gameFEN): Loads a game from a FEN string.
- void updateColorPawns(boolean color): Iterates over every pawn of the color opposite to the turn and updates their "move status".
- Tile checkCheck(): Checks to see whether a check is active and returns the corresponding tile containing the king in check.
- int checkMove(int firstX, int firstY, int destX, int destY): Checks if move entered on GUI is valid.
- ArrayList<Tile> pieceTiles(): Returns all the tiles of the board in an ArrayList.
- Tile replayExecution(int index): The replay method for updating the board based on an index to obtain a relevant fenstring.
- void turnExecution(): The sequence which occurs when the turn is going to be switched.
- boolean isTurn(): Returns the current turn.
- void changeTurn(): Switches the turn to the opposite color.
- Board getBoard(): Gets the board and returns it.
- ArrayList<String> getFenList(): Returns the list of FEN strings for the game.

4.18 WinPopUp class interface specification

Type: public

Extends: Application

- This is required by JavaFX

Public methods:

- WinPopUp(Stage parentStage, String message): Constructor of the object WinPopUp.
- WinPopUp(Stage parentStage, String message, boolean draw, boolean secondMenu): Constructor of the object WinPopUp with the Draw menu.
- void start(Stage stage): This methods draws out the window pop-up prompt when called.

Fields:

- private String message: The message to be shown on the pop-up prompt.
- private boolean draw: Whether the pop-up is for a draw offer.
- private boolean secondMenu: Whether the pop-up is for a second menu.

Additional Notes:

- The WinPopUp class handles all the pop-up prompt when a player select either "Draw", "Resign", or "Exit" during their turn in the middle of the chess game.

4.19 PromotionPopp class interface specification

Type: public

Extends: Application

- This is required by JavaFX

Public methods:

- PromotionPopup(Stage parentStage, boolean playerColor): Constructor of the object PromotionPopup.
- void start(Stage stage): This method draws out the window pop-up prompt when called.
- Button addButton(String name, int x, int y): Adds a button onto the screen using the parameters.

Fields:

- private String message: The message to be shown on the pop-up prompt.

- private boolean draw: Whether the pop-up is for a draw offer.
- private boolean secondMenu: Whether the pop-up is for a second menu.

Additional Notes:

- The PromotionPopup class draws a pop-up window with four buttons: Queen, Rook, Bishop, and Knight. The player can click on one of these buttons to promote their pawn to that piece.

4.20 ReplayBoard class interface specification

Type: public

Extends: Application

- This is required by JavaFX

Public methods:

- ReplayBoard(Stage parentStage, String fenSave, String nameOne, String nameTwo, boolean turn): Constructor of the object ReplayBoard.
- void start(Stage stage): Sets up the scene and shows the board on the stage.
- void createBoard(Pane root): Creates and draws the board.
- void customPiecesToBoard(Pane root, ArrayList<Tile> tileCoords): Loads a custom piece position to the board.
- void initText(Pane root, String playerA, String playerB): Initiates and writes text to window.
- void removeCheckSquare(Pane root): Changes a square back to its original colour when no longer in check.
- void checkSquare(Pane root): Changes a square to red when the king is in check.
- void addButtons(Pane root): Adds buttons to the window.
- void updateTurn(Pane root, String text): Updates the turn text.

Private methods:

- void removePieces(Pane root): Removes pieces from the board and draws them in their new positions.
- void drawPiece(Pane root, String imageName, int x, int y): Draws a specified piece on a specified square.

4.21 FenConverter class interface specification

Type: public

Extends: nothing

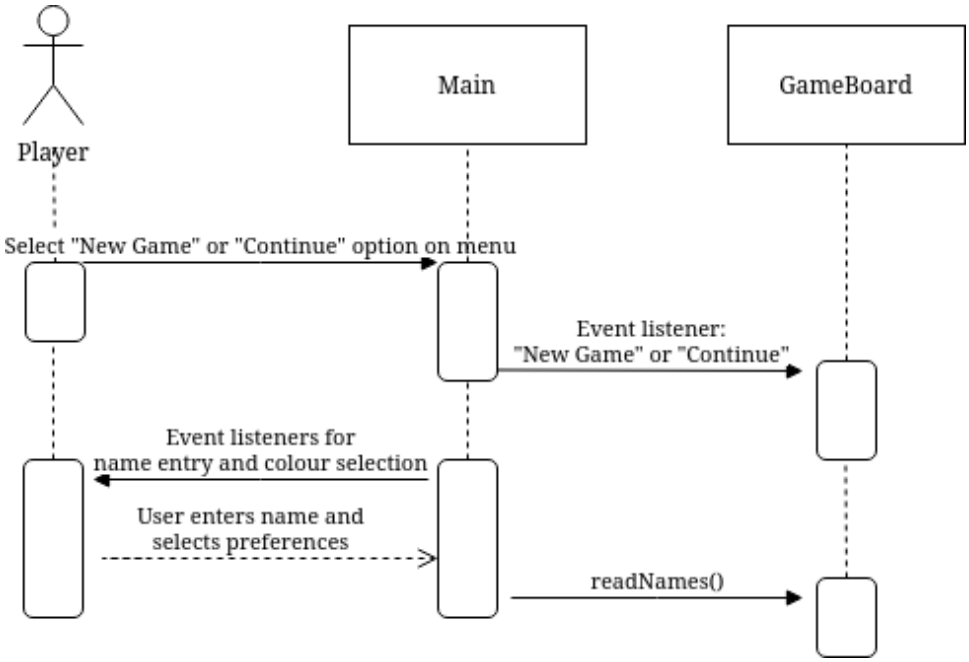
Public methods:

- loadFENFromFile(String fileName): Loads a single FEN string from a file.
- loadGameFromFile(String fileName): Fills an arrayList with all FEN strings in a file.
- loadBoardFromFen(String fen): Converts a FEN string into a board with pieces.
- boardToFen(Board board, boolean turn): Writes a chess board to a FEN string.
- getAllFenStrings(String fileName): Fills an arrayList with all FEN strings in a file.
- topFenString(String filename): Gets the top FEN string from a file.
- writeLineToFile(String filePath, String line): Writes a line to a file.
- writeSaveOnly(Board board, boolean turn): Writes a chess board to a file in the save only format.
- isTurnColor(): Returns the current turn color.

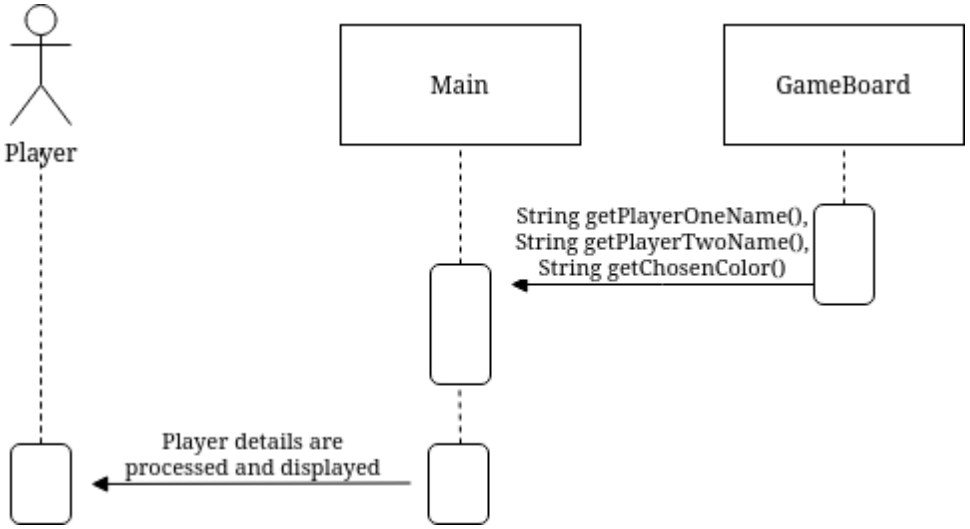
5. DETAILED DESIGN

5.1 Sequence Diagrams

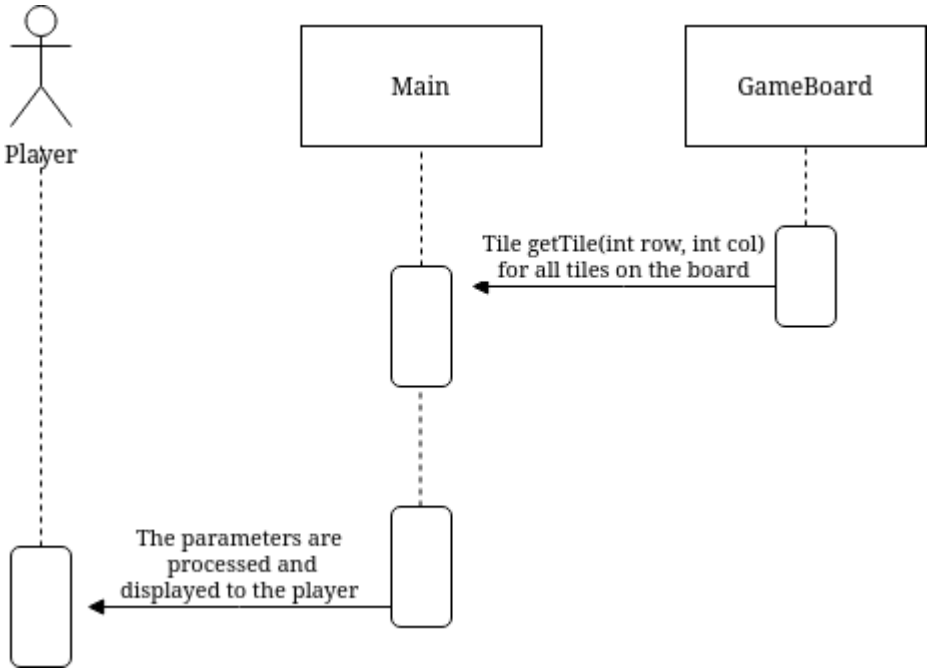
FR1: Player Setup



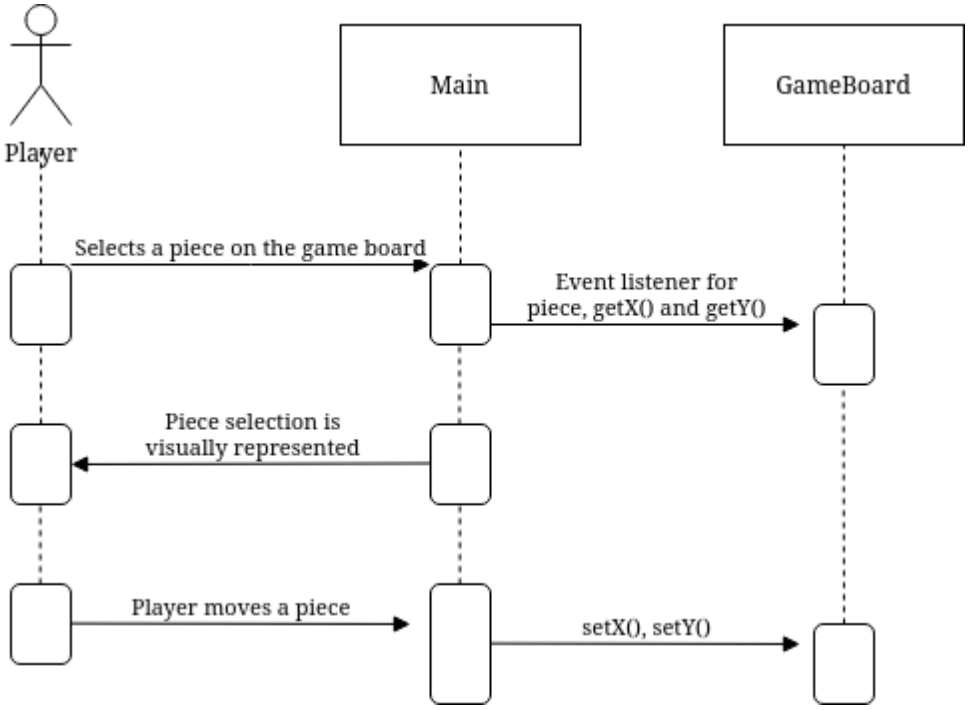
FR2: Player Management


Player
Main
GameBoard
String getPlayerOneName(),
String getPlayerTwoName(),
String getChosenColor()
Player details are
processed and displayed

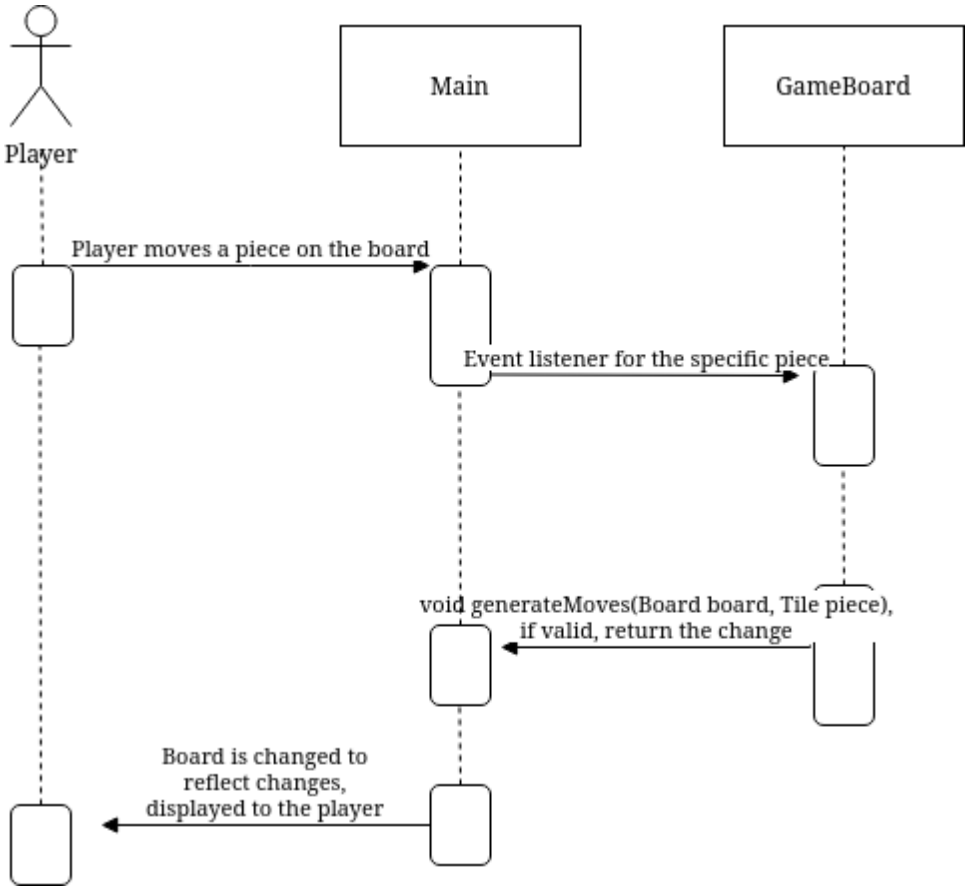
FR3: Board Management



FR4: Piece selection



FR5: Movement



5.2 Significant Algorithms

5.2.1 lineMovement (Tile currentPiece, Board board, int n, int m)

ArrayList moves = new ArrayList

int x = get x from tile

int y = get y from tile

```
        for (i 1 -> 8 i+1) {
                Tile temp = get tile from (x+(i*n), y+(i*m))
                if (if there's an enemy piece)
                        add temp to moves
                else if (if there's a friendly)
                        add temp to moves
                        break from loop
                else
                        break from loop
        }
        return moves
```

Explanation

The line movement method generates all the valid tile moves a specified piece can do. The piece is put into the method through “current tile”, the board is passed through into the method as board and the integers ‘n’ and ‘m’ define the direction you want to check the moves in. If n is 1 it will check every tile up from the starting one to the end of the board, then return that list. This method is created with polymorphism in mind allowing it to be used for straight and diagonal lines, reducing the number of methods needed for checking valid moves.

5.2.2 Castling (Tile king, Board board)

```
ArrayList moves = new ArrayList
King king = get piece from tile
int kingX = get x from king
int kingY = get y from king
Rook leftRook = new rook(!kingColour)
Rook rightRook = new rook(!kingColour)
if (if piece char at (0, kingY) == 'r' or 'R')
        leftRook = get piece from (0, kingY)
if (if piece char at (0, kingY) == 'r' or 'R' )
        rightRook = get piece from (7, kingY)
leftSquareOne = get piece char from (kingX+1, kingY)
leftSquareTwo = get piece char from (kingX+2, kingY)
rightSquareOne = get piece char from (kingX-1, kingY)
rightSquareTwo = get piece char from (kingX-2, kingY)
rightSquareThree = get piece char from (kingX-3, kingY)

if (king hasn’t moved)
        if (left rook hasn’t moved and left rook color == king color)
                if (leftSquareOne and leftSquareTwo are empty)
                        add to moves Tile (kingX+2, kingY)
        if (right rook hasn’t moved and right rook color == king color)
                if (rightSquareOne and rightSquareTwo and rightSquareThree are empty)
                        add to moves (kingX-3, kingY)
return moves;
```

Explanation

Castling algorithm is quite straight forward but makes use of a lot of variables due to the nature of castling involving the checking of many tiles to complete one move. Castling requires the king to have not moved, the rook on the side chosen to not have moved and the tiles the king passed through when castling to be not “controlled” by an enemy piece. The pseudocode goes through all the necessary checks for castling specified and then returns a list of the possible up to 2 tiles the king can move to through castling.

5.2.3 getColorMoves (boolean color, Board board)

```
ArrayList moves = new ArrayList
```

```
for (i 1 -> 8 i+1)
        for (j 1 -> 8 j+1)
                currentTile = get tile at board (i, j)
                if (color of piece on tile == color)
                        add all current pieces moves to moves
return moves
```

Explanation

This algorithm returns all the moves for a specified colour, this is required for the checkCheck method. This algorithm will iterate over the entire board and retrieve the moves of every piece of the colour passed through the parameters. This is the simplest solution to get all of the moves of a specified colour.

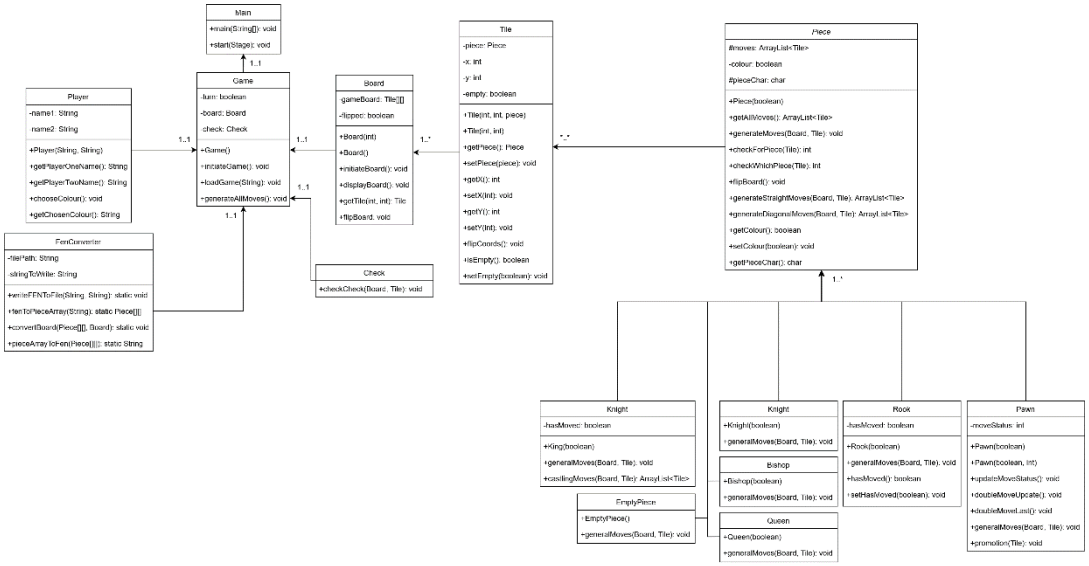
5.2.4 checkCheck (Tile king, Board board)

```
ArrayList moves1 = ArrayList of moves from pieces that touch the king
ArrayList moves2 = ArrayList of moves the king can make to break check
If (moves1 size == 0)
        checkmate is possible
For (Each move in moves2)
If one of the moves would allow a piece to touch the king, then it is check
If (check and checkmate are possible)
Return 2 (Game over)
Else if (the king is in check)
Return 1 (The king is in check)
Return 0 (The king is safe)
```

Explanation

checkCheck is an algorithm to identify whether a player's king is currently in check or checkmate, returning an integer representing the state of the king. This algorithm is essential to the game and is used within the checkmate method. It makes use of a single for loop as this for loop will iterate over all of the moves. Then it checks if any of the moves overlap with the king and if any do then the king is in check. A king is in check if it is being attacked by any piece. This is a very important algorithm as it crucial to the functionality of chess. It's results are passed to the game class check method.

5.3 UML Class Diagrams



5.4 Significant Data Structures

Our program will have a 2D array that is to be created by the Board class. This will be used to store tiles, which contain all information about pieces. This 2d array is a virtual representation of a chessboard. Our program has several methods to accommodate this array. Such as a method that flips and mirrors the coordinates, allowing chess moves to be calculated for both sides of the board.

Our program will also use text files to store games and their configurations. Each file will have a list of FEN strings which represent the state of the board at each turn. These files will be used for resuming games as well as the replay function. Example FEN String: “rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1”.

REFERENCES

[1] Software Engineering Group Projects – Design Specification Standards / 2.3 (Release)
[2] Use Case Document/1.0(Release)

DOCUMENT HISTORY

Version	*Issue No.*	*Date*	*Changes made to document*	*Changed by*
0.1	N/A	08/03/23	N/A - original version	Jac127
0.2	N/A	13/03/23	Added some sequence diagrams	Arm36
0.3	N/A	14/03/23	Added interface description	Mrm19
0.4	N/A	14/03/23	Added component diagram	Jas160
0.5	N/A	15/03/23	Added more sequence diagrams and updated existing ones to better match the interface description	Arm36
0.6	N/A	15/03/23	Added UML Class Diagram	Wip24
0.7	N/A	16/03/23	Added class mapping to requirements and significant data structures	Jac127
0.8	#32	21/03/23	Modified FENFiles to add loading method	Jac127
0.9	#33	21/03/23	Updated sequence diagrams	Arm36
0.10	#36	21/03/23	Added Significant Algorithms	Ktd1
0.11	#37	22/03/23	UML Diagram & Interface description update	Wip24
0.12	#24	23/03/23	Added updated component diagram	Jac127
1.0	N/A	23/03/23	Release	Jac127
1.1	N/A	26/04/23	Added ktd1's pseudocode descriptions and an example FEN string	Arm36
1.2	N/A	10/05/23	Updated significant classes, mapping classes to functional requirements and significant algorithms. Also fixed table format.	Jac127
1.3	N/A	10/05/23	Added new UML class diagram (made by wip24)	Jac127
1.4	N/A	10/05/23	Interface description Updated	mrm19
1.5	N/A	11/05/23	Updated Component diagram	Jas160