Folder engine

11 printable files

(file list disabled)

engine\Board.java

```java
package engine;

import chess.PlayerColor;
import engine.piece.*;

public class Board {

    public interface AddListener {
        void action(Piece piece, int x, int y);
    }

    public interface CaptureListener {
        void action(int x, int y);
    }

    public interface PromoteListener {
        void action(Piece piece, int x, int y);
    }

    public interface CastleListener {
        void action(int kingX, int rookX, int y);
    }

    private AddListener onAdd;
    private CaptureListener onCapture;
    private PromoteListener onPromotion;
    private CastleListener onCastle;

    private Piece[][] pieces;

    private Piece lastMoved;

    private PlayerColor playerTurn;

    public Board() {
        pieces = new Piece[8][8];
        lastMoved = null;
    }

    public Piece[][] getPieces() {
        return pieces;
    }

    public void setPieces(Piece[][] pieces) {
        this.pieces = pieces;
    }

    public Piece getLastMoved() {
        return lastMoved;
    }

    public PlayerColor getPlayerTurn() {
        return playerTurn;
    }

    public void setPlayerTurn(PlayerColor playerTurn) {
        this.playerTurn = playerTurn;
    }
```

```java
public void setAddListener(AddListener onAdd) {
    this.onAdd = onAdd;
}

public void setCaptureListener(CaptureListener onCapture) {
    this.onCapture = onCapture;
}

public void setPromotionListener(PromoteListener onPromotion) {
    this.onPromotion = onPromotion;
}

public void setCastleListener(CastleListener onCastle) {
    this.onCastle = onCastle;
}

/**
 * Set a position for a piece.
 *
 * @param piece Piece to be set.
 * @param x     x coordinate.
 * @param y     y coordinate.
 */
public void setPiece(Piece piece, int x, int y) {
    pieces[x][y] = piece;

    if (onAdd != null) {
        onAdd.action(piece, x, y);
    }
}

/**
 * Remove a piece from a certain position.
 *
 * @param x x coordinate.
 * @param y y coordinate.
 */
public void removePiece(int x, int y) {
    pieces[x][y] = null;

    if (onCapture != null) {
        onCapture.action(x, y);
    }
}

/**
 * Check's whether the piece in the starting square can legally move to the destination square.
 *
 * @param fromX Starting x coordinate.
 * @param fromY Starting y coordinate.
 * @param toX   Desired x coordinate.
 * @param toY   Desired y coordinate.
 * @return Valid move or not.
 */
public boolean canMove(int fromX, int fromY, int toX, int toY) {
    Piece piece = pieces[fromX][fromY];

    // Check piece not moving
    if (piece == null || fromX == toX && fromY == toY) {
        return false;
    }

    // Check correct colour is playing
    if (playerTurn != piece.getColor()) {
        return false;
    }

    // Check if there is a piece on the destination square
    boolean capture = pieces[toX][toY] != null;

    // Check not capturing comrades unless castle
```

```java
        if (capture && pieces[toX][toY].getColor() == piece.getColor() && !(piece instanceof King &&
Math.max(Math.abs(toX - fromX), Math.abs(toY - fromY)) > 1 && piece.validMove(fromX, fromY, toX, toY, this, true))) {
            return false;
        }

        // Check for valid move
        if (piece.validMove(fromX, fromY, toX, toY, this, capture)) {
            // Check the move does not put the king in check
            return kingSafe(fromX, fromY, toX, toY, capture);
        }
        return false;
    }

    /**
     * Moves the piece from the starting position to the desired square.
     *
     * @param fromX Starting x coordinate.
     * @param fromY Starting y coordinate.
     * @param toX   Desired x coordinate.
     * @param toY   Desired y coordinate.
     */
    public void move(int fromX, int fromY, int toX, int toY) {

        Piece piece = pieces[fromX][fromY];

        boolean capture = pieces[toX][toY] != null;

        // Check if castle
        if (piece instanceof King && Math.abs(fromX - toX) > 1) {
            if (onCastle != null) {
                onCastle.action(fromX, fromX - toX > 0 ? 0 : 7, fromY);
                setPiece(piece, fromX - toX > 0 ? 2 : 6, fromY);
            }
        } else {
            setPiece(piece, toX, toY);
        }

        removePiece(fromX, fromY);

        // Pawn move
        if (piece instanceof Pawn p) {
            ((Pawn) piece).setLastMoveDist(Math.abs(fromY - toY));

            // Promotion
            if (toY == 7 || toY == 0) {
                promotePawn(p, toX, toY);
            }

            // If valid move and diagonal not capture -> en passant
            if (!capture && fromX != toX) {
                removePiece(toX, toY - (piece.getColor() == PlayerColor.WHITE ? 1 : -1));
            }
        }

        if (piece instanceof SpecialPiece) {
            ((SpecialPiece) piece).moved();
        }

        // Switch which colour is playing
        playerTurn = playerTurn == PlayerColor.WHITE ? PlayerColor.BLACK : PlayerColor.WHITE;

        lastMoved = piece;
    }

    /**
     * Promotes a pawn to a piece chosen by the user.
     *
     * @param pawn Pawn to promote.
     * @param toX  x coordinate of the pawn.
     * @param toY  y coordinate of the pawn.
     */
```

```java
private void promotePawn(Pawn pawn, int toX, int toY) {
    if (onPromotion != null) {
        onPromotion.action(pawn, toX, toY);
    }
}

/**
 * Find the current player's King's position
 *
 * @param color Color of the current player.
 * @return x and y coordinates for the king.
 */
private int[] findKing(PlayerColor color) {
    int[] position = {-1, -1};
    for (int i = 0; i < pieces.length; ++i) {
        for (int j = 0; j < pieces.length; ++j) {
            if (pieces[i][j] != null && pieces[i][j] instanceof King && pieces[i][j].getColor() == color) {
                position[0] = i;
                position[1] = j;
                return position;
            }
        }
    }
    return position;
}

/**
 * Check's whether the king would be put in check with a specific move.
 *
 * @param fromX   Starting x coordinate.
 * @param fromY   Starting y coordinate.
 * @param toX     Desired x coordinate.
 * @param toY     Desired y coordinate.
 * @param capture Piece on destination square.
 * @return King would be in check.
 */
public boolean kingSafe(int fromX, int fromY, int toX, int toY, boolean capture) {
    // To check whether the king is in danger, we simulate the move being made
    Piece piece = pieces[fromX][fromY];
    Piece victim = null;
    // Check piece not moving to same square
    if (!(fromX == toX && fromY == toY)) {
        if (capture) {
            victim = pieces[toX][toY];
        }
        pieces[toX][toY] = piece;
        pieces[fromX][fromY] = null;
    }

    // Find the king
    int[] kingPos = findKing(piece.getColor());
    if (kingPos[0] != -1 && kingPos[1] != -1) {
        // Check that the King is not in check
        boolean check = isInCheck(piece.getColor());
        // Put back the pieces
        pieces[toX][toY] = victim;
        pieces[fromX][fromY] = piece;
        return !check;
    }
    return true;
}

/**
 * Check's whether the king is currently in check.
 *
 * @param color Color of the king to check.
 * @return King in check.
 */
public boolean isInCheck(PlayerColor color) {
    int[] kingPos = findKing(color);
    for (int i = 0; i < pieces.length; ++i) {
            for (int j = 0; j < pieces.length; ++j) {
                if (pieces[i][j] != null && pieces[i][j].getColor() != color && pieces[i][j].validMove(i, j,
kingPos[0], kingPos[1], this, true)) {
                    return true;
                }
            }
        }
        return false;
    }

    /**
     * Checks whether check mate has been reached.
     *
     * @return Check mate.
     */
    public boolean isCheckMate() {
        for (int i = 0; i < pieces.length; ++i) {
            for (int j = 0; j < pieces.length; ++j) {
                if (pieces[i][j] != null && pieces[i][j].getColor() == playerTurn) {
                    for (int k = 0; k < pieces.length; ++k) {
                        for (int l = 0; l < pieces.length; ++l) {
                            if (canMove(i, j, k, l)) {
                                return false;
                            }
                        }
                    }
                }
            }
        }
        return true;
    }
}
```

engine\Game.java

```
package engine;

import chess.ChessController;
import chess.ChessView;
import chess.PieceType;
import chess.PlayerColor;
import engine.piece.*;

public class Game implements ChessController {
    private static final int BOARD_SIZE = 8;
    private ChessView view;
    private final Board board;

    public Game() {
        board = new Board();
    }

    @Override
    public void start(ChessView view) {
        this.view = view;
        this.view.startView();
        board.setPlayerTurn(PlayerColor.WHITE);

        // Event listeners
        board.setAddListener((piece, x, y) -> view.putPiece(piece.getType(), piece.getColor(), x, y));

        board.setCaptureListener(view::removePiece);

        board.setPromotionListener((piece, x, y) -> {
            PlayerColor color = piece.getColor();
            Piece[] choices = {
                    new Queen(color, board),
                    new Knight(color, board),
                    new Rook(color, board),
```

```java
                new Bishop(color, board)
        };

        Piece userChoice;
        do {
            userChoice = view.askUser("Promotion", "Choose a piece to promote to", choices);
        } while (userChoice == null);


        board.removePiece(x, y);
        board.setPiece(userChoice, x, y);
    });

    board.setCastleListener(((kingX, rookX, y) -> {
        King king = (King) board.getPieces()[kingX][y];
        Rook rook = (Rook) board.getPieces()[rookX][y];
        int kingTo = kingX - rookX > 0 ? 2 : 6;
        int rookTo = kingTo == 2 ? 3 : 5;
        board.removePiece(rookX, y);
        board.removePiece(kingX, y);
        board.setPiece(king, kingTo, y);
        board.setPiece(rook, rookTo, y);
    }));
}

@Override
public boolean move(int fromX, int fromY, int toX, int toY) {
    boolean move = false;

    if (board.canMove(fromX, fromY, toX, toY)) {
        board.move(fromX, fromY, toX, toY);
        move = true;
    }

    String message = "";
    boolean checkMate = false;

    if (board.isInCheck(board.getPlayerTurn())) {
        message = "Check! ";
        if (board.isCheckMate()) {
            message = "Checkmate! " + (board.getPlayerTurn() == PlayerColor.WHITE ? PlayerColor.BLACK :
PlayerColor.WHITE) + " wins!";
            checkMate = true;
        }
    }

    if (!checkMate) {
        message += board.getPlayerTurn() + " to play";
    }

    view.displayMessage(message);

    return move;
}

/**
 * Initialize the board with the pieces at their starting positions.
 */
@Override
public void newGame() {
    board.setPieces(new Piece[8][8]);

    // Pawns
    for (int i = 0; i < BOARD_SIZE; i++) {
        board.getPieces()[i][1] = new Pawn(PlayerColor.WHITE, board);
        board.getPieces()[i][6] = new Pawn(PlayerColor.BLACK, board);
        view.putPiece(PieceType.PAWN, PlayerColor.WHITE, i, 1);
        view.putPiece(PieceType.PAWN, PlayerColor.BLACK, i, 6);
    }

    // Rooks
```

```
        board.getPieces()[0][0] = new Rook(PlayerColor.WHITE, board);
        board.getPieces()[7][0] = new Rook(PlayerColor.WHITE, board);
        board.getPieces()[0][7] = new Rook(PlayerColor.BLACK, board);
        board.getPieces()[7][7] = new Rook(PlayerColor.BLACK, board);
        view.putPiece(PieceType.ROOK, PlayerColor.WHITE, 0, 0);
        view.putPiece(PieceType.ROOK, PlayerColor.WHITE, 7, 0);
        view.putPiece(PieceType.ROOK, PlayerColor.BLACK, 0, 7);
        view.putPiece(PieceType.ROOK, PlayerColor.BLACK, 7, 7);

        // Knights
        board.getPieces()[1][0] = new Knight(PlayerColor.WHITE, board);
        board.getPieces()[6][0] = new Knight(PlayerColor.WHITE, board);
        board.getPieces()[1][7] = new Knight(PlayerColor.BLACK, board);
        board.getPieces()[6][7] = new Knight(PlayerColor.BLACK, board);
        view.putPiece(PieceType.KNIGHT, PlayerColor.WHITE, 1, 0);
        view.putPiece(PieceType.KNIGHT, PlayerColor.WHITE, 6, 0);
        view.putPiece(PieceType.KNIGHT, PlayerColor.BLACK, 1, 7);
        view.putPiece(PieceType.KNIGHT, PlayerColor.BLACK, 6, 7);

        // Bishops
        board.getPieces()[2][0] = new Bishop(PlayerColor.WHITE, board);
        board.getPieces()[5][0] = new Bishop(PlayerColor.WHITE, board);
        board.getPieces()[2][7] = new Bishop(PlayerColor.BLACK, board);
        board.getPieces()[5][7] = new Bishop(PlayerColor.BLACK, board);
        view.putPiece(PieceType.BISHOP, PlayerColor.WHITE, 2, 0);
        view.putPiece(PieceType.BISHOP, PlayerColor.WHITE, 5, 0);
        view.putPiece(PieceType.BISHOP, PlayerColor.BLACK, 2, 7);
        view.putPiece(PieceType.BISHOP, PlayerColor.BLACK, 5, 7);

        // Queens
        board.getPieces()[3][0] = new Queen(PlayerColor.WHITE, board);
        board.getPieces()[3][7] = new Queen(PlayerColor.BLACK, board);
        view.putPiece(PieceType.QUEEN, PlayerColor.WHITE, 3, 0);
        view.putPiece(PieceType.QUEEN, PlayerColor.BLACK, 3, 7);

        // Kings
        board.getPieces()[4][0] = new King(PlayerColor.WHITE, board);
        board.getPieces()[4][7] = new King(PlayerColor.BLACK, board);
        view.putPiece(PieceType.KING, PlayerColor.WHITE, 4, 0);
        view.putPiece(PieceType.KING, PlayerColor.BLACK, 4, 7);

        board.setPlayerTurn(PlayerColor.WHITE);
        view.displayMessage(board.getPlayerTurn() + " to play");
    }
}
```

engine\Main.java

```
package engine;

import chess.ChessController;
import chess.ChessView;
import chess.views.gui.GUIView;

public class Main {
    public static void main(String[] args) {
        // 1. Création du contrôleur pour gérer le jeu d'échecs
        ChessController controller = new Game();

        // 2. Création de la vue désirée
        ChessView view = new GUIView(controller);
        //ChessView view = new ConsoleView(controller); MODE CONSOLE

        // 3. Lancement du programme
        controller.start(view);
    }
}
```

engine\piece\Bishop.java

```java
package engine.piece;

import chess.PieceType;
import chess.PlayerColor;
import engine.Board;

public class Bishop extends Piece {
    public Bishop(PlayerColor color, Board board) {
        super(color, PieceType.BISHOP, board);
    }

    @Override
    public boolean validMove(int fromX, int fromY, int toX, int toY, Board board, boolean capture) {
        // Check if diagonal move
        if (Math.abs(fromX - toX) == Math.abs(fromY - toY)) {
            int xSign = toX - fromX >= 0 ? 1 : -1;
            int ySign = toY - fromY >= 0 ? 1 : -1;
            // Check no pieces in between
            for (int i = 1; i < Math.abs(fromX - toX); ++i) {
                if (board.getPieces()[fromX + i * xSign][fromY + i * ySign] != null) {
                    return false;
                }
            }
            return true;
        }
        return false;
    }

    @Override
    public String textValue() {
        return "Bishop";
    }
}
```

engine\piece\King.java

```java
package engine.piece;

import chess.PieceType;
import chess.PlayerColor;
import engine.Board;

public class King extends SpecialPiece {
    public King(PlayerColor color, Board board) {
        super(color, PieceType.KING, board);
    }

    @Override
    public boolean validMove(int fromX, int fromY, int toX, int toY, Board board, boolean capture) {

        int xDiff = Math.abs(fromX - toX);
        int yDiff = Math.abs(fromY - toY);
        int xCorner = fromX - toX < 0 ? 7 : 0;
        // 1 square
        if (xDiff == 1 && yDiff == 0 || xDiff == 0 && yDiff == 1 || xDiff == 1 && yDiff == 1) {
            super.moved = true;
            return true;
        }

        // Castle
        Piece rook = board.getPieces()[xCorner][fromY];
        return (xDiff == 2 || (capture && xCorner == toX)) && yDiff == 0 && canCastle(rook, xCorner, fromX, fromY);
    }

    private boolean canCastle(Piece rook, int xCorner, int fromX, int fromY) {
        // Check nor the king nor the rook have moved before
        if (this.hasNotMoved() && rook instanceof Rook && ((Rook) rook).hasNotMoved()) {
            // Check there are no pieces in between the king and the rook
            for (int i = 1; i < Math.abs(fromX - xCorner); ++i) {
                if (board.getPieces()[xCorner + (xCorner == 0 ? i : -i)][fromY] != null) {
                    return false;
                }
            }
            // Check the king does not move over any spaces in which he would be checked
            return board.kingSafe(fromX, fromY, fromX - (xCorner == 0 ? 1 : -1), fromY, false);
        }
        return false;
    }

    @Override
    public String textValue() {
        return "King";
    }
}
```

engine\piece\Knight.java

```
package engine.piece;

import chess.PieceType;
import chess.PlayerColor;
import engine.Board;

public class Knight extends Piece {
    public Knight(PlayerColor color, Board board) {
        super(color, PieceType.KNIGHT, board);
    }

    @Override
    public boolean validMove(int fromX, int fromY, int toX, int toY, Board board, boolean capture) {
        return Math.abs(fromX - toX) == 1 && Math.abs(fromY - toY) == 2 || Math.abs(fromX - toX) == 2 &&
Math.abs(fromY - toY) == 1;
    }

    @Override
    public String textValue() {
        return "Knight";
    }
}
```

engine\piece\Pawn.java

```
package engine.piece;

import chess.PieceType;
import chess.PlayerColor;
import engine.Board;

public class Pawn extends SpecialPiece {
    private int lastMoveDist;

    public Pawn(PlayerColor color, Board board) {
        super(color, PieceType.PAWN, board);
        lastMoveDist = 0;
    }

    public void setLastMoveDist(int dist) {
        lastMoveDist = dist;
    }

    @Override
    public boolean validMove(int fromX, int fromY, int toX, int toY, Board board, boolean capture) {
        // Factorisation between white and black pawns
        int whiteBlack = 1;
        if (color == PlayerColor.BLACK) {
            whiteBlack = -1;
        }
        // Straight line
        if (!capture && fromX == toX) {
            // 1 square
            if (fromY == toY - whiteBlack) {
                return true;
            }
            // 2 square, check that the pawn is on the correct row and no piece in front
            else {
                return fromY == toY - 2 * whiteBlack && board.getPieces()[fromX][toY - whiteBlack] == null &&
(whiteBlack == 1 && toY == 3 || whiteBlack == -1 && toY == 4);
            }
        }
        // Capture
        else if (capture && Math.abs(fromX - toX) == 1 && fromY == toY - whiteBlack) {
            return true;
        }
        // En passant
        else {
            // Check correct move format
            if (Math.abs(toX - fromX) == 1 && fromY == toY - whiteBlack) {
                Piece otherPawn = board.getPieces()[toX][fromY];
                // Check if nothing on destination square and neighboring piece is a pawn which just moved 2 squares
                return !capture && otherPawn instanceof Pawn && ((Pawn) otherPawn).lastMoveDist == 2 &&
board.getLastMoved() == otherPawn;
            }
        }
        return false;
    }

    @Override
    public String textValue() {
        return "Pawn";
    }
}
```

engine\piece\Piece.java

```
package engine.piece;

import chess.ChessView;
import chess.PieceType;
import chess.PlayerColor;
import engine.Board;

public abstract class Piece implements ChessView.UserChoice {
    protected PlayerColor color;

    protected PieceType type;

    protected Board board;

    public Piece(PlayerColor color, PieceType type, Board board) {
        this.color = color;
        this.type = type;
        this.board = board;
    }

    public PlayerColor getColor() {
        return color;
    }

    public PieceType getType() {
        return type;
    }
```

```java
    /**
     * Check's whether a specific move is valid for the piece currently on the departure square.
     *
     * @param fromX   Starting x coordinate.
     * @param fromY   Starting y coordinate.
     * @param toX     Desired x coordinate.
     * @param toY     Desired y coordinate.
     * @param board   Game board to analyse.
     * @param capture Piece on the destination square.
     * @return Valid move.
     */
    public abstract boolean validMove(int fromX, int fromY, int toX, int toY, Board board, boolean capture);

    /**
     * Provides a name for each type of piece for the graphic interface.
     *
     * @return Piece type name.
     */
    public abstract String textValue();
}
```

engine\piece\Queen.java

```java
package engine.piece;

import chess.PieceType;
import chess.PlayerColor;
import engine.Board;

public class Queen extends Piece {
    public Queen(PlayerColor color, Board board) {
        super(color, PieceType.QUEEN, board);
    }

    @Override
    public boolean validMove(int fromX, int fromY, int toX, int toY, Board board, boolean capture) {
        // Check if valid bishop or rook move
        Bishop bishop = new Bishop(color, board);
        Rook rook = new Rook(color, board);
        return bishop.validMove(fromX, fromY, toX, toY, board, capture) || rook.validMove(fromX, fromY, toX, toY,
board, capture);
    }

    @Override
    public String textValue() {
        return "Queen";
    }
}
```

engine\piece\Rook.java

```java
package engine.piece;

import chess.PieceType;
import chess.PlayerColor;
import engine.Board;

public class Rook extends SpecialPiece {
    public Rook(PlayerColor color, Board board) {
        super(color, PieceType.ROOK, board);
    }

    @Override
    public boolean validMove(int fromX, int fromY, int toX, int toY, Board board, boolean capture) {
        // Straight/normal
        int xDiff = toX - fromX;
        int yDiff = toY - fromY;
        if (Math.abs(xDiff) == 0 || Math.abs(yDiff) == 0) {
            int xSign = xDiff >= 0 ? xDiff == 0 ? 0 : 1 : -1;
            int ySign = yDiff >= 0 ? yDiff == 0 ? 0 : 1 : -1;
            // Check no pieces in between
            for (int i = 1; i < Math.max(Math.abs(xDiff), Math.abs(yDiff)); ++i) {
                if (board.getPieces()[fromX + i * xSign][fromY + i * ySign] != null) {
                    return false;
                }
            }
            return true;
        }
        return false;
    }

    @Override
    public String textValue() {
        return "Rook";
    }
}
```

engine\piece\SpecialPiece.java

```java
package engine.piece;

import chess.PieceType;
import chess.PlayerColor;
import engine.Board;

public abstract class SpecialPiece extends Piece {

    protected boolean moved;

    public SpecialPiece(PlayerColor color, PieceType type, Board board) {
        super(color, type, board);
        moved = false;
    }

    /**
     * Check's whether a piece has been moved before.
     *
     * @return Piece has not been moved.
     */
    public boolean hasNotMoved() {
        return !moved;
    }

    /**
     * Informs that the piece has been moved.
     */
    public void moved() {
        moved = true;
    }
}
```