Nate Smith, Jerry Salas, Joshua Dowd, Jake Puebla
CPSC 231
Dr. E.E.L
14 December 2025

Write Up CPSC 231

Description

Our project is a functional game that allows a person to play chess against themselves or another person. We utilized third-party GUI libraries, incorporating logic learned in class, to accomplish this.

We chose this project because we all like playing chess. We all understood the challenges that might be faced, but had a good general grasp of the logic/formatting that would be crucial in creating Chess. The move logic and Abstract class structuring were an easily agreed-upon course of action that pushed us forward in this project.

Classes & Structure

* chessBoard.java and BorderLayoutExample.java are used exclusively for the GUI (Graphic User Interface) for the game.

Board:

This abstract class represents a chessboard, and also contains the logic behind castling, en passant, and the player's turn.

Member Variables:

- Piece[][] board: represents the board itself. The square A1 is represented by board[0][0], and the square H8 is represented by board[7][7].
- boolean whiteTurn: represents which player's turn it is. If it is White's turn, whiteTurn = true, but if it is Black's turn, whiteTurn = false.
- int enPassantRow, int enPassantCol, and boolean enPassantIsWhite: contains the logic behind en passant.
- boolean whiteShortCastle, boolean whiteLongCastle, boolean blackShortCastle, and boolean blackLongCastle: contains the logic behind castling.

Methods:

- Board(Board other)
    - Creates a deep copy of an existing Board object and returns it.

- King getKing(boolean isWhite)
    - Returns the King object associated with the color isWhite.

- boolean move(int startRow, int startCol, int endRow, int endCol)
    - Attempts to move the Piece object located at position (startRow, startCol) on the board to position (endRow, endCol).
    - If the move is valid, this function performs the desired move and then returns true.
    - If the move is invalid, nothing happens, and the function returns false.

- void uncheckedMove(Piece currPiece, int startRow, int startCol, int endRow, int endCol)
    - Helper method, used by the move function to move the pieces across the board.

- boolean movePassant(int startRow, int startCol, int endRow, int endCol)
    - Attempts to perform en passant by moving the Pawn object at position (startRow, startCol) to the position (endRow, endCol). Returns true if the move is possible, false otherwise.
    - If the move is valid, this function performs the desired move and then returns true.
    - If the move is invalid, nothing happens, and the function returns false.

- boolean castle(int endCol)
    - Attempts to castle by moving the King to the column endCol. Returns true if the move is possible, false otherwise.
    - If the move is valid, this function performs the desired move and then returns true.
    - If the move is invalid, nothing happens, and the function returns false.

- boolean isCheckmate()
    - Returns true if the current player is in checkmate.

- void changeTurn()
    - Changes the player’s turn.

Piece (abstract class):

This abstract class represents a generic chess piece on the board.

Member Variables:

- boolean isWhite: represents the piece's color.
- int row: represents the piece's row on the chessboard.
- int col: represents the piece's column on the chessboard.

Methods:

- boolean canMove(Board gameBoard, int endRow, int endCol)
    - Returns true if the current Piece object moving to location [endRow, endCol] on the Board gameBoard would be a valid move, according to the rules of chess. If the move is invalid, returns false.
    - gameBoard is the Board object currently being used to run the game.
    - endRow is the desired row position of the move that is being checked for validity
    - endCol is the desired column position of that move.

- Piece copy()
    - Creates a new deep copy of the current Piece object and returns it.

- int getRow()
    - Accessor method: returns row

- int getCol()
    - Accessor method: returns col

- boolean isWhite()
    - Accessor method: returns isWhite

Knight (extends Piece):

This class represents a Knight piece on a chessboard. Inherits all its methods and member variables from the abstract Piece superclass. The canMove function is overridden to determine whether or not a specific move will be valid for a Knight.

Bishop (extends Piece):

This class represents a Bishop piece on a chessboard. Inherits all its methods and member variables from the abstract Piece superclass. The canMove function is overridden to determine whether or not a specific move will be valid for a Bishop.

Rook (extends Piece):
This class represents a Rook piece on a chessboard. Inherits all its methods and member variables from the abstract Piece superclass. The canMove function is overridden to determine whether or not a specific move will be valid for a Rook.

Queen (extends Piece):
This class represents a Queen piece on a chessboard. Inherits all its methods and member variables from the abstract Piece superclass. The canMove function is overridden to determine whether or not a specific move will be valid for a Queen.

Pawn (extends Piece):
This class represents a Pawn piece on a chessboard. Inherits most of its methods and member variables from the abstract Piece superclass. The canMove function is overridden to determine whether or not a specific move will be valid for a Pawn.
Non-Inherited Member Variables:

- boolean hasMoved: is set to true if the Pawn piece has already been moved this game, false otherwise. This is important because only pawns which have not moved are allowed to move two spaces forward in a turn; otherwise they can only be moved one space forward (excluding special circumstances like en passant).

Non-Inherited Methods:

- boolean isValid(Board gameBoard, int endCol, int endRow, int direction)
    - Helper function for canMove method. Used to determine move validity.

- Piece getTarget(Piece[][] board, int endCol, int endRow)
    - Helper function for isValid method. Used to check which piece the player is attempting to capture with a diagonal movement from the pawn.

King (extends Piece):
This class represents a King piece on a chessboard. Inherits most of its methods and member variables from the abstract Piece superclass. The canMove function is overridden to determine whether or not a specific move will be valid for a King.
Non-Inherited Methods:

- boolean isInCheck(Board gameBoard, int kingRow, int kingCol)
    - Determines if the King is currently under attack at the specified position
    - Iterates through all opponent pieces and checks if any can attack the King's position
    - Returns true if in check, false if safe

- boolean canPieceAttack(Piece piece, Board gameBoard, int fromRow, int fromCol, int toRow, int toCol)
    - Private helper method that checks if a given piece can attack a target square
    - Implements basic movement rules for each piece type without recursively checking for check (prevents infinite recursion)
    - Returns true if the piece can attack the target square

- boolean isPathClear(Board gameBoard, int fromRow, int fromCol, int toRow, int toCol)
    - Private helper method that verifies no pieces block the path between two squares
    - Used for sliding pieces (Bishops, Rooks, Queens) when checking attacks
    - Returns true if the path is clear

- boolean canCastle(Board gameBoard, boolean isKingSide)
    - Validates whether castling is legal on the specified side
    - Checks: castling rights from Board, empty squares between King and Rook, King not in check, King doesn't pass through or land in check
    - Returns true if castling is legal

- char getSymbol()
    - Returns 'K' for white King, 'k' for black King