Project 3: Object Oriented Chess

Problem Definition:

This program will let two players play chess. A very basic interface will allow the user to load a chess board from a file using the Smith Notation, display a board on the screen using a text user interface, change the position of the different pieces, and finally write the status of the game to a file using Smith Notation.

Design Overview:

The board is an 8x8 array of characters. Each character refers to a single piece. Uppercase letters are black pieces, and lowercase letters represent white pieces. Additionally there is an array of moves which keeps track of the game.

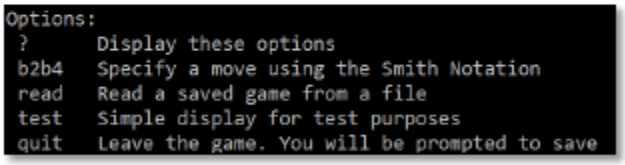
When the board is read from a file, moves will be automatically loaded into the game, parsed for any problems, and then adjusted to display on the board. It works exactly as if the user were to enter the moves one move at a time. The user is given an opportunity repeatedly to enter moves until they specify to quick.

Interface Design:

Output

The board with the pieces on it, the test boardt

Prompt for user input



The menu of options

Save prompt at the end of the game

Input

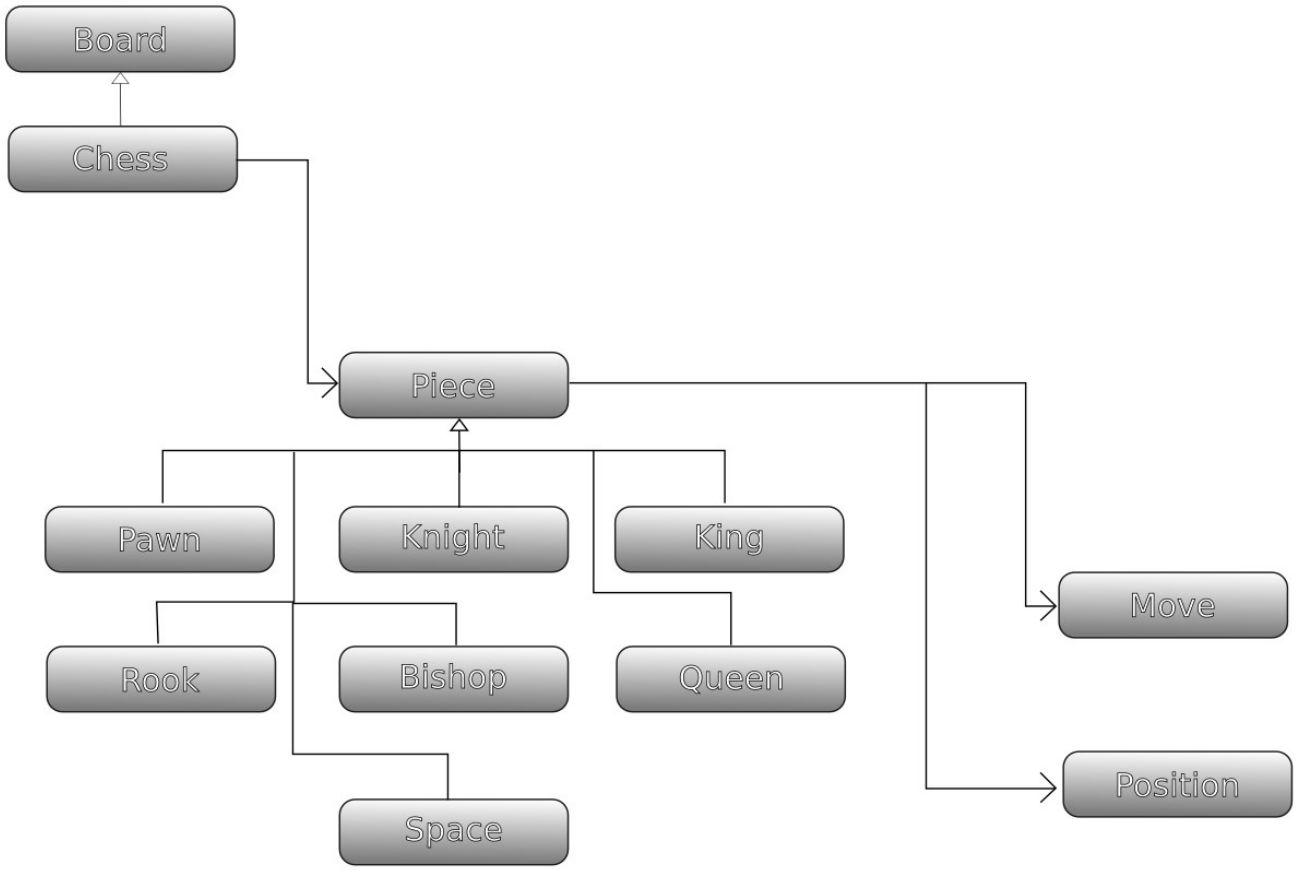
`b2b4`	Any chess algebra is interpreted as a move
`?`	Question mark launches the menu of options
`r`	Launches a prompt for a filename to read
`quit`	Leaves the program with first prompting for save filename
everything else	Re-prompt the user for an option and display an error message.

Errors

`Invalid specification of source coordinates`	The first letter must be 'a'-'h' and the second number must be 1-8. Anything else will generate this message.
`Piece not present in source coordinate`	Trying to move a piece from a source location when there is no piece there.
`Invalid specification of destination coordinates`	Same as first message except for destination coordinates.
`Illegal to move onto your own piece`	Trying to move onto your own piece, something which is never legal.
`Unknown promotion piece specification`	Specifying a promotion (NBRQ), capture (pnbrqk), castling (cC) or en-passant (E) with an invalid letter

1.

Structure Chart:



Data Structures:

Board
- test - allocated -square
+ Board + getPiece + swap + resetBoard + freeBoard + getTest + setTest + [] + <<

Chess
-board -move -moves -numMoves -fileName -fQuit
+Chess +interact +displayOptions +displaySmith +getFileName

Move
- source - dest - piece - capture - enpassant - castleK - castleQ -isWhite -error
+ Move + Move(copy) + getDes + getSrc + getText + getPromotion + getCapture + getEnPassant + getCastleK + getCastleQ + getWhiteMove + == + != + = + << + >> + setCapture + setWhiteMove + setSrc + setDes + setEnPassant + setPromote + setCastle

Position
- row - col
+ Position + Position(row, col) + getRow + getCol + setRow + setCol + isValid + = + == + >> + <<

Data Structures Continued:

Piece
-+ Position -+ isWhiteSquare -+ letter -+ fWhite
+ Place() + Place(white, row, col, letter) + display + getLetter + getScore + getColor + isWhite +setSquareColor + <<

Pawn : Piece
- fMoved
+ Pawn(white, row, col) + getMoves + getScore

Rook : Piece
-fMoved
+Rook (white, row, col) + getMoves + getScore

Knigth : Piece
-
+ Knight (white, row, col) + getMoves + getScore

Bishop : Piece
-
+ Bishop (white, row, col) + getMoves + getScore

Queen : Piece
-
+ getLetter + getMoves + getScore

King : Piece
- fMoved
+ King (white, row, col) + getMoves + getScore

Space : Piece
-
+ Space() + Space(white, row, col) + getMoves + getScores

Algorithms:

```
FUNCTION freeBoard
        IF (allocated is true)
                FOR ( i=0, i<8, i++)
                        FOR (p=0, p<8, p++)
                                Delete square[i][p]
                        ENDFOR
                ENDFOR
        ENDIF
ENDFUNCTION
```

```
FUNCTION displayOptions
        PUT: Options: <NEWLINE>
                ?
                B2b4
                Read
                Help
                Test
                Rank
                Quit
                Smith
ENDFUNCTION
```

```
FUNCTION getText
        IF (error with the length)
                RETURN error
        ENDIF

        PUT source and destination

        IF (enpassant)
                PUT “E”
        ENDIF

        IF (castleK)
                PUT "c"
        ENDIF

        IF (castleQ)
                PUT "C"
        ENDIF
        IF (space not empty)
                PUT  original piece
        ENDIF

        IF (capture and not empty)
                PUT lower case letter or upper case letter in place
        ENDIF

        RETURN results

ENDFUNCTION
```

File Formats:

The game store every move in an array which is written to a file at the end of the game in Smith Notation.

```
e2e4 c7c5
g1f3 d7d6
d2d4 c5d4p
f3d4 g8f6p
b1c3 g7g6
c1e3 f8g7
f2f3 b8c6
d1d2 e8g8c
f1c4
```

When the file is read, each move is "played" exactly as if the user was to enter each move by hand.

Error Handling:

User Errors

Error	Condition	Handling
Invalid move	`sourceRow > h or sourceRow < a or sourceColumn > 8 or sourceColumn < 1 or …`	Display error and re-prompt
Invalid Filename	`fin.fail()`	Re-prompt for filename

File Errors

Error	Condition	Handling
Invalid move	`sourceRow > h or sourceRow < a or sourceColumn > 8 or sourceColumn < 1 or …`	Display error message and stop reading the file at that point.

Internal Errors

Error	Source	Handling
Illegal coordinate	`move.source.r > 7 or move.source.r < 0 or move.source.c > 7 or move.source.c < 0 or …`	Assert is thrown
Illegal piece	`board[r][c] != { space, p, P, r, R, … }`	Assert is thrown