Project 3: Key

CS 165: Object Oriented Software Development

Problem Definition

Chess with objects is an extension of Project 1 where the rules of chess are enforced and the score of a given position is computed for extra credit.

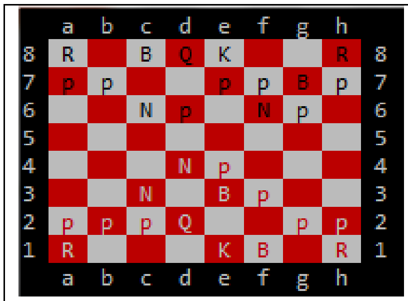
Design Overview

A base class will represent a piece and each type of piece will be its own derived class. Each piece will know is value (force), how to draw itself, and how it moves.

The board will be an array of pointer to pieces. With each turn, a list of all the possible moves will be computed. When the user attempts to move, his attempt will be compared against the list of possible moves to determine the legality of the attempt.

Interface design

Output

<table>
<tr><td>
<pre>
   a b c d e f g h
8 R   B Q K   R 8
7 p p   p p B p 7
6   N p   N p   6
5   5
4   N p   4
3   N   B p   3
2 p p p Q   p p 2
1 R   K B   R 1
   a b c d e f g h
</pre></td><td>Board</td></tr>
<tr><td><pre>
(White):
</pre></td><td>The prompt will alternate between white and black according to the move:</td></tr>
<tr><td><pre>
Options:
 quit   Leave the game. You will be prompted to save
 ?   Display these options
 r   Read a saved game from a file
 help   Display all the possible moves for a given piece
 b2b4   Specify a move using the Smith Notation
</pre></td><td>The menu of options</td></tr>
<tr><td><pre>
To save a game, please specify the filename.
    To quit without saving a file, just press <enter>
</pre></td><td>When the user indicated the game is over, the “prompt for file name” message is given</td></tr>
<tr><td><pre>
Which piece would you like to find the moves for? d7
Possible moves are:
        d7d8
        d7c8b
        d7d8Q
        d7d8R
        d7d8B
        d7d8N
</pre></td><td>When the user asks for help moving a piece, this prompt and display is given</td></tr>
</table>

Input

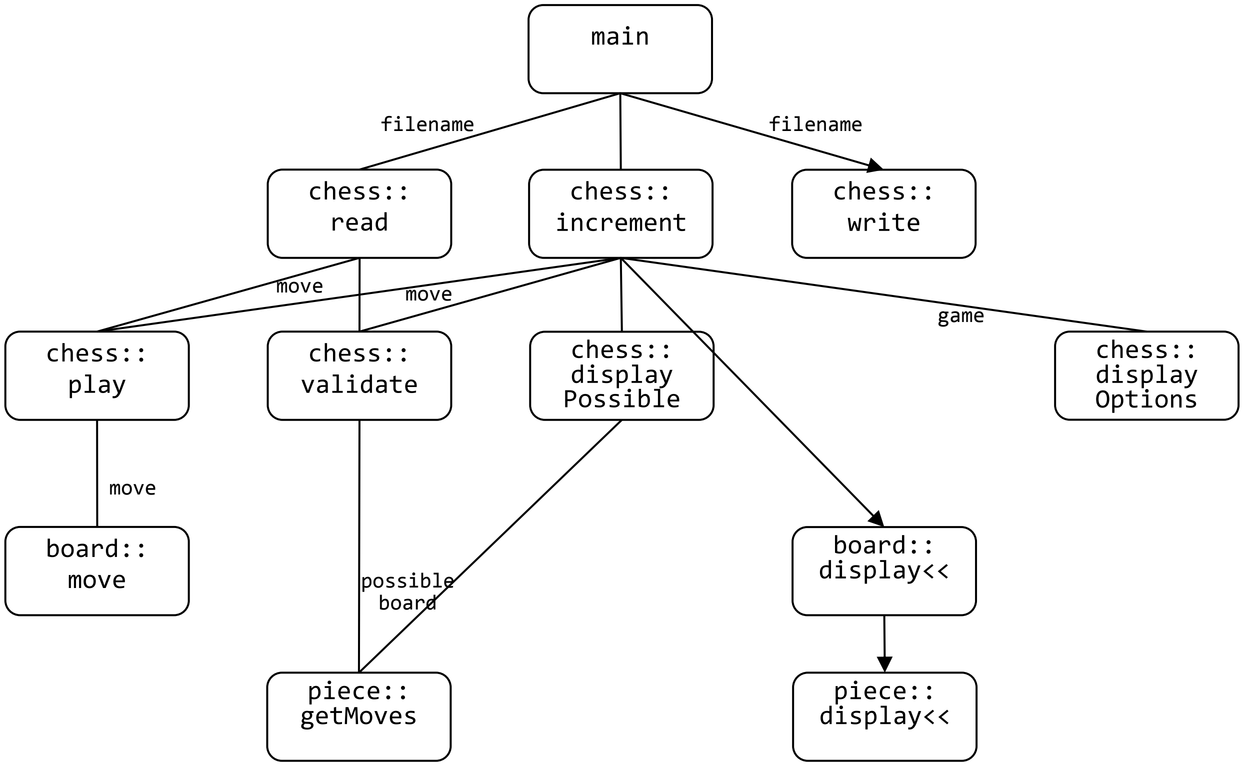
`B2b4`	Any chess algebra is interpreted as a move
`?`	Question mark launches the menu of options
`r`	Launches a prompt for a filename to read
`quit`	Leaves the program with first prompting for save filename
`help`	Prompt the user for the square to probe
everything else	Re-prompt the user for an option and display an error message.

Errors

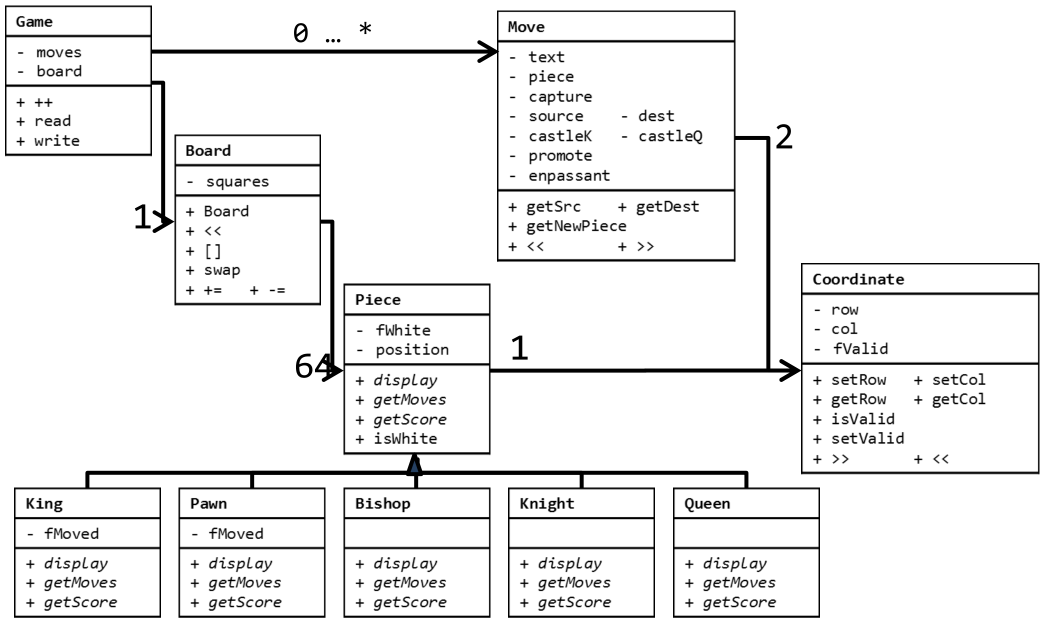
The following message may be encountered by the user:

`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
`Illegal move. The <piece name> cannot move to this location`	When a piece is moved to a location that is not valid

Structure chart



Data Structures



Algorithms

```
Board::display
   PUT header
   FOR pos.row = 7 .. pos.isValid by -1
      FOR pos.col = 0 .. pos.isValid by 1
         PUT board[pos]
      pos = valid
```

```
Rook::getMoves(moves)
   delta = { {1, 0}, {0, 1}, {-1, 0}, {0, -1} };
   FOR iDelta = 0 .. 3
      pos = position + delta
      WHILE position.isValid() && board[pos] == ' '
         moves.push_back(pos)
         pos += delta[iDelta]
```

```
Rook::getMoves(moves)
   delta = { {1, 0}, {0, 1}, {-1, 0}, {0, -1} }
   genMoveList(moves, board, delta, true /*slide*/)
```

```
Knight::getMoves(moves)
   delta = { {-1,  2}, {1,  2}, {-2,  1}, {2,  1},
             {-1, -2}, {1, -2}, {-2, -1}, {2, -1}}
   genMoveList(moves, board, delta, false /*slide*/)
```

```
King::getMoves(moves)
   delta = { {-1, -1}, {1,  -1}, {1,  1}, {-1,  1},
             {1, 0}, {0, 1}, {-1, 0}, {0, -1}}
   genMoveList(moves, board, delta, false /*slide*/)

   IF not moved and board[row, 5] = space and
        board[row, 6] = space and board[row, 7] = rook
        and board[row, 7] not moved
      moves += castle(row, 6)
```

```
Piece::genMoveList(moves, board, delta, slide?)
    FOR i = 0 … delta.num
        posMove = position + delta[i]
        IF slide?
            WHILE posMove is valid and board[posMove] = space
                moves += posMove
                posMove += delta[i]
        IF posMove is valid and
           (board[posMove] is opposite color or board[posMove] = space)
                moves += posMove
```

```
Pawn::getMoves(moves)
   posMove = position.r + (1 or -1), position.c                     move forward
   IF posMove.isValid and board[posMove] = empty
      moves += posMove

   posMove = position.r + (2 or -2), position.c                     move two
   IF posMove.isValid and board[posMove] = empty and notMoved
      moves += posMove

   posMove = position.r + (1 or -1), position.c + (1 or -1)       attack
   IF posMove.isValid and board[posMove] is opposite color
      moves += posMove and attack

   posMove = posMove = position.r + (1 or -1), position.c         en-passant
   IF posMove.isValid and board[posMove.r, position.c] is pawn and he just moved
      moves += posMove and enpassant

   IF front rank                                                    promote
      moves += posMove and promote to {QRBN}
```

```
Board::move(move)
   IF move.king-side castle
      swap(move.src, move.des)
      move.src.row = 5
      move.des.row = 7
      swap(move.src, move.des)
   IF move.queen-side castle
      swap(move.src, move.des)
      move.src.row = 3
      move.des.row = 0
      swap(move.src, move.des)
   IF move.en-passant
      swap(src, des)
      posKill = src.row, des.col
      board -= posKill
   IF move.capture
      board -= des
      swap(src, des)
   IF move.promotion
      board -= src
      SWITCH move.promotion
         ‘Q’ board = new Queen
         ‘R’ board = new Rook
         …
   ELSE
      swap(src, des)
```

File formats

The game stores the list of moves in the `moves` array. That array is written to the file in Smith Notation with the white, black moves paired:

```
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 entered the moves in by hand.

Error Handling

User Errors

Error	Condition	Handling
Invalid option	User specifies input that does not correspond to the known options	Re-prompt for option
Invalid move	An error has occurred in the coordinates or any special designation in the Smith Notation	Re-prompt for move

File Errors

Error	Condition	Handling
Invalid file name	Requested file is not present or not available for any reason	Display error and continue
Invalid move	An error has occurred in the coordinates or any special designation in the Smith Notation	Display error message and stop reading the file at that point.

Internal Errors

Each class will have error checking:

Error	Source	Handling
`Board` : The position on the board and the piece's self-reported position should be the same.	Piece is incorrectly moved	Assert is thrown
`Move` : Illegal move because piece missing or whatever	Parse move allowed a move that is not legal or the board and the move are out of sync.	Assert is thrown
`Position` : Invalid moves are checked	Invalid bit not properly checked	Assert is thrown