Name: Muhammed Ademola

Date: 4/25/2021

Project Title: Chess War

Summary of Project:

My project's aim is to create a Player vs Player game attack game based off of Chess pieces. The objective of the game will be to destroy all the other player's pieces. Each player will start off 6 pieces. Just like the normal game of chess, there will be 6 possible pieces, each player will get one of each. Each type of piece will have its own attributes (hit points (HP), attack points (AP), and mana points (MP)) and special properties. The pawn will retain the ability to upgrade to a non-pawn piece if they reach the other side of the board. The pieces will move across the chess board as they do in a normal chess game. When an enemy player's piece in the line of movement of the another, they can attack that piece. If the piece that is attacked is not destroyed, the attacking piece will not move from its position. The attacking piece will only move is the spot is requested is legal and empty or legal and a piece is being destroyed. If a piece loses all of its HP, it gets destroyed. A player gains mana by attacking other pieces. They can then use that mana for special moves.

My board will be a 6-by-6 board instead of a normal 8-by-8 chess board. Each player will have their pieces lined up on opposite ends of the chess board in a random order. One player will have black pieces, the other will have white pieces. One player will randomly start first. Each player will have 15 second to make their moves. If a player doesn't make a move, they forfeit their turn. The game continues until one player forfeits or has all their pieces destroyed. At the end of the game, the player will have the option of saving their game statistics to a text file for analyzing.

Classes:

1. Board: Creates, stores, and updates the board
2. Piece: Will provide a template for how each of the game pieces work.
3. Player: Stores information on player such as pieces left, time for move, total time, etc.
4. King: stores information about King piece such as legal moves, special move, etc.
5. Queen: stores information about Queen piece such as legal moves, special move, etc.
6. Rook: stores information about Rook piece such as legal moves, special move, etc.
7. Bishop: stores information about Bishop piece such as legal moves, special move, etc.
8. Knight: stores information about Knight piece such as legal moves, special move, etc.
9. Pawn: stores information about Pawn piece such as legal moves, special move, etc.

**Board Class**

**Abstract class: No**
**Subclass of: N/A**
**Composed of: Piece, Player, King, Queen, Rook, Bishop, Pawn, Knight**

**Data Members**

| Variable Name | Data Type | Static | Description |
|---|---|---|---|
| board | string[][] | no | Holds the string representation of board |
| pboard | Piece[][] | no | Holds the object representation of board |
| x, y, x_new, y_new | int | no | Stored indexes to be used for array |
| arr | const char*[] | no | Holds abbreviations for pieces |
| k1, k2 | King | no | Holds king pieces |
| q1, q2 | Queen | no | Holds queen pieces |
| r1, r2 | Rook | no | Holds rook pieces |
| b1, b2 | Bishop | no | Holds bishop pieces |
| n1, n2 | Knight | no | Holds knight pieces |
| p1, p2 | Pawn | no | Holds pawn pieces |

**Member Functions**

| Prototype | Static | Virtual | Overloading Operator | Friend of this Class | Description |
|---|---|---|---|---|---|
| void generate_board(); | no | no | no | no | Generates the board for the game |
| void show_board(); | no | no | no | no | Shows the black and white board for the game |

| | | | | | |
|---|---|---|---|---|---|
| void show_board_old(); | no | no | no | no | Shows string representation of board for the game |
| void show_board_init() | no | no | no | no | Shows the initial black and white board for the game |
| void instantiate(Player& player1, Player& player2) | no | no | no | no | Creates object representation of the board and add pieces to the players for them to manipulate |
| bool move_piece(int ind1, int ind2, char color) | no | no | no | no | Returns whether or not move is valid |
| bool update_board(Piece & curr, int x1, int y1, char c) | no | no | no | no | Updates the boards and returns whether or not move is valid, called by move_piece. |
| void update_player(int time_s, Player& player) | no | no | no | no | Updates the playes to write to files and check whether pieces has been destroyed. |
| void white_square(const char* d) | no | no | no | no | Prints white square in console |

| | | | | | |
|---|---|---|---|---|---|
| void black_square(const char* d) | no | no | no | no | Prints black square in console |
| void pawn_promotion(char c) | no | no | no | no | Governs whether pawn can be promoted or not |
| ~Board() | no | no | no | no | Stops ncurses instance. |

**Piece Class**

**Abstract class: No**
**Subclass of: N/A**
**Composed of: N/A**

**Data Members**

| Variable Name | Data Type | Static | Description |
|---|---|---|---|
| name | string | no | Holds the name of the Piece |
| attack | int | no | Holds the attack power of the Piece |
| health | int | no | Holds the health of the Piece |
| mana | int | no | Holds the mana of the Piece |
| short_name | char | no | Holds the short name of the Piece |
| dest | bool | no | Holds whether piece has been destroyed or not |
| special | bool | no | Holds whether special move has been activated |
| move_num | int | no | Holds the amount of moves piece has made |

**Member Functions**

| Prototype | Static | Virtual | Overloading Operator | Friend of this Class | Description |
|---|---|---|---|---|---|
| Piece() | no | no | no | no | Constructor for class, sets variables to null |
| string get_name( ); | no | no | no | no | Returns data stored in variable name. |
| char get_short_ name(); | no | no | no | no | Returns an abbreviation of the piece name. |

| | | | | | |
|---|---|---|---|---|---|
| void set_short_name(); | no | no | no | no | sets an abbreviation of the piece name. |
| int get_attack(); | no | no | no | no | Returns attack power of piece |
| int manage_attack(); | no | no | no | no | manages attack power of piece |
| virtual bool special_move(); | no | yes | no | no | Implements the special move of the piece |
| int get_health(); | no | no | no | no | Get the health of the piece |
| int manage_health(); | no | no | no | no | Adds or subtract the health of the piece |
| int set_health(); | no | no | no | no | Set the health of the piece |
| int get_mana(); | no | no | no | no | Returns mana of piece |
| void manage_mana(); | no | no | no | no | Adds or subtract mana to piece |
| void move_num_increases() | no | no | no | no | Increases number of moves piece has made |
| void get_move_num() | no | no | no | no | returns the number of moves made by piece |

| | | | | | |
|---|---|---|---|---|---|
| void move_num_increases() | no | no | no | no | Increases number of moves piece has made |
| virtual bool legal_moves(std::string board[][6], int x, int y, int x1, int y1, char c) | no | yes | no | no | Governs moves that can legally be made by peice |
| void operator=(char _assign) | no | no | yes | no | Assigns the char in parameter list to short name of piece |
| bool operator==(char c) | no | no | yes | no | Compares c to piece shortname and returns the true or false |
| bool operator==(Piece& c) | no | no | yes | no | Compares the piece short name to the piece in parameter list and return true or false |
| void set_destroyed(bool a) | no | no | no | no | Sets the destroyed bool to true or false |
| Bool get_destroyed(bool a) | no | no | no | no | Returns the dest bool |
| friend std::ostream& operator<<(std::ostream& c, const Piece& _piece) | no | no | yes | yes | Extracts the piece short name to print. |

**Player Class**

**Abstract class: No**
**Subclass of: N/A**
**Composed of: Piece**

**Data Members**

| Variable Name | Data Type | Static | Description |
|---|---|---|---|
| name | string | no | Holds the name of the player |
| time | int | no | Holds the time player has used to make moves |
| move_i | int | no | Holds the number or individual player moves |
| pieces_added | int | yes | Holds the number of pieces added to class |
| num_pieces | int | no | Holds player number of pieces |
| Player_num | int | no | Holds the player number |
| Move_num | int | yes | Holds the number of moves made by both players combined |
| pieces | Int[] | no | Holds the pieces player has |

**Member Functions**

| Prototype | Static | Virtual | Overloading Operator | Friend of this Class | Description |
|---|---|---|---|---|---|
| int get_move( ) | no | no | no | no | Returns number of moves individual player has made |
| void move() | no | no | no | no | Adds one move to individual player move total |

| | | | | | |
|---|---|---|---|---|---|
| void player_stats(int time_s, Piece& piece_w) | no | no | no | no | Returns the move a player has made |
| void check_piece_stats(Piece& piece_w) | no | no | no | no | Returns time player took to make move |
| void operator+=(Piece& _piece) | no | no | yes | no | Adds pieces to the player's pieces array |
| static void increase_total_moves_count() | yes | no | no | no | Increases total move count for the class |
| static int get_total_moves() | yes | no | no | no | returns total moves for the class |
| static void increase_pieces_added | yes | no | no | no | Increase the static pieces added variable for when both players have pieces added |
| void decrease_num_pieces() | no | no | no | no | Decrease individual player number of pieces |

| | | | | | |
|---|---|---|---|---|---|
| int get_num_pieces() | No | no | no | no | Returns number of pieces an individual player has |
| Piece player_piece_array(int i) | no | no | no | no | Returns the piece at the index i |
| void set_player_number(int _num) | no | no | no | no | Sets the player number |
| friend std::istream& operator>>(std::istream& c, Player& _player) | No | no | yes | yes | Uses stream to input player names when requested |
| friend std::ostream& operator<<(std::ostream& c, const Player& _player) | No | no | yes | yes | Use stream to output player names when requested |

**King Class**

**Abstract class: N/A**
**Subclass of: Piece**
**Composed of: N/A**

**Data Members**

| Variable Name | Data Type | Static | Description |
|---|---|---|---|

**Member Functions**

| Prototype | Static | Virtual | Overloading Operator | Friend of this Class | Description |
|---|---|---|---|---|---|
| King() | no | no | no | no | Initializes attributes of class |
| bool legal_moves(std::string board[][6], int x, int y, int x1, int y1, char c) | no | no | no | no | Tells whether a move is legal or not |
| bool special_move() | no | no | no | no | Implements the special move of the piece; |

**Queen Class**

**Abstract class: N/A**
**Subclass of: Piece**
**Composed of: N/A**

**Data Members**

| Variable Name | Data Type | Static | Description |
|---|---|---|---|

**Member Functions**

| Prototype | Static | Virtual | Overloading Operator | Friend of this Class | Description |
|---|---|---|---|---|---|
| Queen() | no | no | no | no | Initializes attributes of class |
| bool legal_moves(std::string board[][6], int x, int y, int x1, int y1, char c) | no | no | no | no | Tells whether a move is legal or not |
| bool special_move() | no | no | no | no | Implements the special move of the piece; |

**Rook Class**

**Abstract class: N/A**
**Subclass of: Piece**
**Composed of: N/A**

**Data Members**

| Variable Name | Data Type | Static | Description |
|---|---|---|---|

**Member Functions**

| Prototype | Static | Virtual | Overloading Operator | Friend of this Class | Description |
|---|---|---|---|---|---|
| Rook() | no | no | no | no | Initializes attributes of class |
| bool legal_moves(std::string board[][6], int x, int y, int x1, int y1, char c) | no | no | no | no | Tells whether a move is legal or not |
| bool special_move() | no | no | no | no | Implements the special move of the piece; |

**Bishop Class**

**Abstract class: N/A**
**Subclass of: Piece**
**Composed of: N/A**

**Data Members**

| Variable Name | Data Type | Static | Description |
|---|---|---|---|

**Member Functions**

| Prototype | Static | Virtual | Overloading Operator | Friend of this Class | Description |
|---|---|---|---|---|---|
| Bishop() | no | no | no | no | Initializes attributes of class |
| bool legal_moves(std::string board[][6], int x, int y, int x1, int y1, char c) | no | no | no | no | Tells whether a move is legal or not |
| bool special_move() | no | no | no | no | Implements the special move of the piece; |

**Knight Class**

**Abstract class: N/A**
**Subclass of: Piece**
**Composed of: N/A**

**Data Members**

| Variable Name | Data Type | Static | Description |
|---|---|---|---|

**Member Functions**

| Prototype | Static | Virtual | Overloading Operator | Friend of this Class | Description |
|---|---|---|---|---|---|
| Knight() | no | no | no | no | Initializes attributes of class |
| bool legal_moves(std::string board[][6], int x, int y, int x1, int y1, char c) | no | no | no | no | Tells whether a move is legal or not |
| bool special_move() | no | no | no | no | Implements the special move of the piece; |

**Pawn Class**

**Abstract class: N/A**
**Subclass of: Piece**
**Composed of: N/A**

**Data Members**

| Variable Name | Data Type | Static | Description |
|---|---|---|---|
| Promotion_p | bool | no | True or false on whether piece has been promoted |

**Member Functions**

| Prototype | Static | Virtual | Overloading Operator | Friend of this Class | Description |
|---|---|---|---|---|---|
| Pawn() | no | no | no | no | Initializes attributes of class |
| bool promotion( int x, int y, int x1, int y1, char c) | no | no | no | no | Tells whether a move is eligible for promotion |
| bool special_move() | no | no | no | no | Implements the special move of the piece; |
| bool is_promoted() | no | no | no | no | Returntrue or false on whether a piece has been promoted |

**Demonstration of OOP Concepts**

1. Encapsulation:

   Multiple private data members in my Piece, Player, and Board class. These members are all private because they represent data that shouldn't be able to be tempered by outside classes or functions.

   Line Number and files:
   Board.h : lines 24-48
   Piece.h : lines 9-48
   Player.h : lines 22 - 45

2. Inheritance:

   All of the Chess pieces that inherit from the Piece class so they could use the schematic of that class. This provides most of the code for how I will build the specific classes for pieces. It also stops me from repeating the same code in multiple classes.

   Line Number and files:
   Pawn.h : line 7
   Bishop.h : line 7
   Queen.h : line 7
   King.h : line 7
   Rook.h : line 7
   Knight.h : line 7

3. Polymorphism:

   All of my Chess piece classes are polymorphic because they are all instances of the piece class. They all use multiple functions from the piece abstract class and repurpose them as their own.

   Line Number and files:
   Board.cpp: line 4 (used at lines 5, 7, 8)
   Board.cpp: line 220 (used at lines 232, 249, 267)
   Player.cpp: Line 28 (used at lines 38-42)
   Player.cpp: Line 74-75
   Player.cpp: Line 78-79
   Piece.cpp: Line 103-14
   Piece.cpp: Line 119-120

4. Static Members/Functions:

I used static functions in my player class to make sure that I could keep track of the number of total moves made in the game and the number of pieces each player had when the object array board is initialized.

Line Number and files:
Board.cpp: Line 6
Board.cpp: Line 88
Main.cpp: Line 279

Player.h: Lines 215, 28, 37, 38, 39

5. Friend functions:

   I'm using a friend function in my game class to be able to manipulate names values in classes such player with user input and to be able to out put them to the screen

   Line Number and files:
   Main.cpp: line 148, 151, 152, 155, 206 (defined at Player.cpp 74 and 78)

6. Overloaded operators:

   I overloaded the << and >> operators in the player class to be able to display the player name and input the player name. I overloaded == twice in the piece class to be able to compare short name of pieces and full names of pieces when appropriate. I also overloaded = and << in piece to make it easy to assign short names to pieces and output piece's short name.

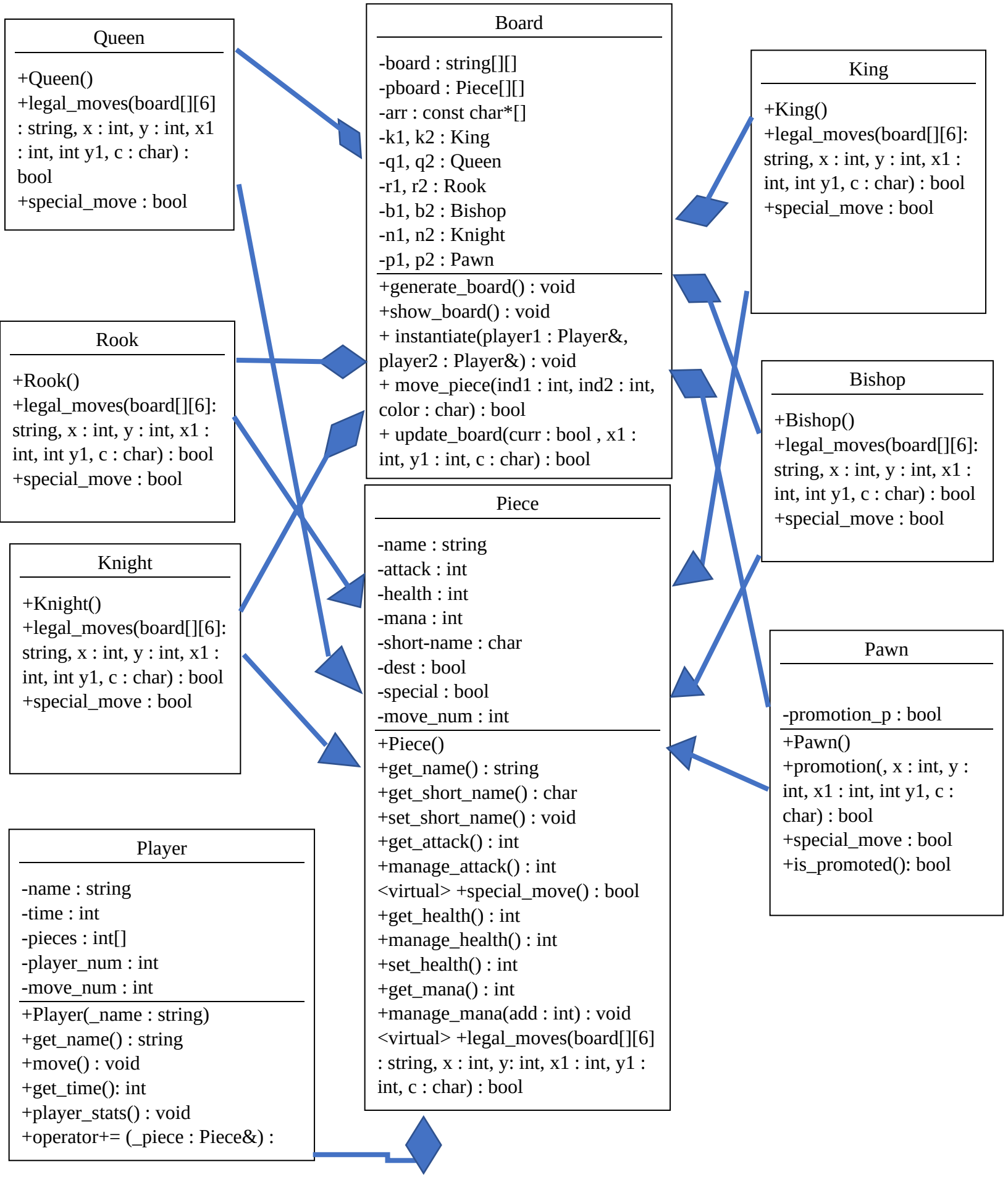
   Line Number and files:
   Definitions:
   Piece.cpp: lines 97, 100, 103, 119 (used: Board.cpp 190, 192, 304)
   Player.cpp: lines 71, 74, 78 (used: Board.cpp line 84-85 and Main.cpp: line 148, 151, 152, 155, 206)

7. Text file(s):

   I plan to write game statistics out to a text file so players can know how they played. Doing this can help a player analyze their moves and improve.

   Line Number and files:
   Player.cpp: line 33 (Used Board.cpp line 7)

**UML Diagram**


Queen
+Queen()
+legal_moves(board[][6] : string, x : int, y : int, x1 : int, int y1, c : char) : bool
+special_move : bool
Board
-board : string[][]
-pboard : Piece[][]
-arr : const char*[]
-k1, k2 : King
-q1, q2 : Queen
-r1, r2 : Rook
-b1, b2 : Bishop
-n1, n2 : Knight
-p1, p2 : Pawn
+generate_board() : void
+show_board() : void
+ instantiate(player1 : Player&, player2 : Player&) : void
+ move_piece(ind1 : int, ind2 : int, color : char) : bool
+ update_board(curr : bool , x1 : int, y1 : int, c : char) : bool
King
+King()
+legal_moves(board[][6]: string, x : int, y : int, x1 : int, int y1, c : char) : bool
+special_move : bool
Rook
+Rook()
+legal_moves(board[][6]: string, x : int, y : int, x1 : int, int y1, c : char) : bool
+special_move : bool
Bishop
+Bishop()
+legal_moves(board[][6]: string, x : int, y : int, x1 : int, int y1, c : char) : bool
+special_move : bool
Knight
+Knight()
+legal_moves(board[][6]: string, x : int, y : int, x1 : int, int y1, c : char) : bool
+special_move : bool
Piece
-name : string
-attack : int
-health : int
-mana : int
-short-name : char
-dest : bool
-special : bool
-move_num : int
+Piece()
+get_name() : string
+get_short_name() : char
+set_short_name() : void
+get_attack() : int
+manage_attack() : int
<virtual> +special_move() : bool
+get_health() : int
+manage_health() : int
+set_health() : int
+get_mana() : int
+manage_mana(add : int) : void
<virtual> +legal_moves(board[][6] : string, x : int, y: int, x1 : int, y1 : int, c : char) : bool
Pawn
-promotion_p : bool
+Pawn()
+promotion(, x : int, y : int, x1 : int, int y1, c : char) : bool
+special_move : bool
+is_promoted(): bool
Player
-name : string
-time : int
-pieces : int[]
-player_num : int
-move_num : int
+Player(_name : string)
+get_name() : string
+move() : void
+get_time(): int
+player_stats() : void
+operator+= (_piece : Piece&) :