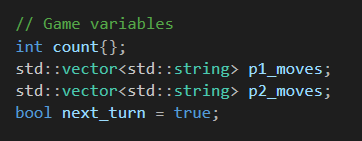
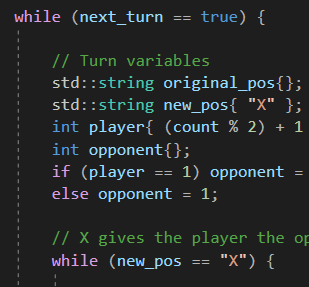
The individual game code is contained within a while loop that operates while the string “repeat” is “Y”, this is referred to as the game loop. The user can then set the value of repeat to “Y” or “N” at the end of the game depending on whether they want to play another game. The code for an individual game starts with the construction of a “board” called “game\_board” using the default constructor. Pieces are then placed on the board by assigning shared pointers in “squares” to the correct piece type with correct “player” integer, this is shown in figure (6).

*Figure 6 – The creation of the board and filling in of the pieces.*

Next, variables whose scope lasts an entire game are declared, this is shown in figure (7). Two vectors containing strings are used to store the moves that the two players have completed. “count” is initialised to 0 and is increased by 1 each time a turn in completed. A bool called “next\_turn” is also initialised to true, this is changed to false at the end of a turn if a stalemate or checkmate occurs.

*Figure 7 – Variables that last an entire game*

Turns are contained within a while loop, referred to as the turn loop, that operates while “next\_turn” is true. Inside this, variables whose scope lasts an entire turn, shown in figure (8) are declared. These are strings which describe the coordinates of the original and new positions of the piece being moved and the interger “Player” which describes which player’s turn it is. This is determined by:

which alternates between 1 and 2 each turn. Opponent is then set to the other number. “new\_pos” is initialised to “X” so that the programme enters the next while loop, referred to as the selection loop which operates until the players selection of piece and move has verified to be legal and confirmed by the player.

*Figure 8 – Variables that last an entire turn*

Two vectors of integers “allowed” and “allowed\_checked” are then declared. These are used to hold the vector position in “squares” that a given piece can move to. A representation of the board is then outputted to the player using the member function of board, “print\_board”The programme then enter the next while loop which operates until the size of “allowed\_checked” is non-zero. The player (either 1 or 2) is then instructed to enter the coordinate position of the piece they wish to move. The programme will ask the user to enter the position again if either of Boolean functions: “check\_input\_form\_orig” which checks that the format of the position is valid, or “check\_piece” which checks that the player has a piece at the position, return false.

The allowed moves for the piece are assigned to allowed by the member function “allowed\_moves”. This is a virtual function in “piece”, but is fully described in all derived classes. This function does not take into account whether or not the player is playing themselves into check, which is illegal.

The programme then uses the function “test\_all\_for\_check” which iterates over the moves in allowed, performs them, and adds them to allowed\_checked if the player is not in check. The move is then undone. The board is then displayed to the player again, squares that can be moved to are displayed with an asterisk in the corner. The player is then asked to choose a square to move to. If the entry is not of the correct format or the move is not in “allowed\_checked”, then the player will be asked again, if the player enters “X”, the player will be able to make another selection. If the player enters a move which is in “allowed\_moves” and of the correct format then the code will exit the selection loop.

The first character of the piece type, and the move is then stored in a string, which is then added to either “p1\_moves” or “p2\_moves” depending on which player is playing. The piece is then moved using the member function of board “move\_piece”. If the move results in a pawn reaching eighth rank then the user is asked for a promotion choice which is then checked and performed.

A for loop is then used to iterate over squares and find the position of the opponents pieces. The allowed moves are determined for each piece. All allowed moves are appended to “allowed\_opponent\_all”. This is used to determine the bool “legal\_moves” which is only true if “allowed\_opponent\_all” is non zero.

The bool “check” which is only true if the opponent is in check is then determined by the function “check\_check”. What happens next is determined by the value of the “legal\_move”.

|  |  |  |  |
| --- | --- | --- | --- |
|  | | check\_check | |
| True | False |
| legal\_moves | True | Opponent has been checked | As normal |
| std::cout << "Player " << opponent << " is in check!" << std::endl; |  |
| False | Opponent has been checkmated | Stalemate |
| std::cout << "Checkmate! Player " << player << " wins!" << std::endl;  next\_turn = false; | std::cout << "Stalemate! Game over!" << std::endl;  next\_turn = false; |