Group Members:

(1) August Tan

(2) Elaine Ge

(3) Patricia Yuan

Introduction:

The primary task of this fifth and last assignment of CS246 - Object-Oriented Software Development course is to implement a game of chess using C++ with an emphasis of using the principle of object-oriented design techniques. This approach promotes modularity, reusability and maintainability during the development process of our fully functional chess game. In the Plan of Attack, we will include the introduction of the basic rules of chess, our anticipation of possible implementation for the project, and the answer to the questions given in the instructions.

How Chess is Played:

As highlighted in the project outline,

Chess is played on an 8x8 checkerboard, arranged so that there is a white square at the bottom right. Players take turns, making one move at a time. The player controlling the white pieces makes the first move.

There are six types of pieces:

- King (K) Moves one square in any direction
- Queen (Q) Moves in any of the eight possible directions, any distance, but cannot move past any piece that blocks its path.
- Bishop (B) Moves in any of the four diagonal directions, any distance, but cannot move past any piece that blocks its path.
- Rook (R) Moves in any of the four vertical/horizontal directions, any distance, but cannot move past any piece that blocks its path.
- Knight (N) If it sits on a square (x, y) it can move to square ($x \pm 2, y \pm 1$) or ($x \pm 1, y \pm 2$)
- Pawn (P) moves one square forward.
- Other features such as en passant, castling, stalemate, checkmate, etc are outlined in the project guidelines.

As the King (K) is checked when King can be possibly captured by the opponent, and there is no method to escape or make further movement, then the game is Checkmate, or the game is ended.

In our program, we will be utilizing the terminal to interact with the users using several commands to tell the program what users want to do at next steps.

High-level Overview:

Key Classes and Components

- *Board:*

The Board class represents the chessboard and contains the logic for setting up the pieces and managing the state of the game. The main responsibility, which will be used by other classes, is to track the position of all pieces, validating moves, and detecting winning/ special conditions such as stalemate/in check/ checkmate.

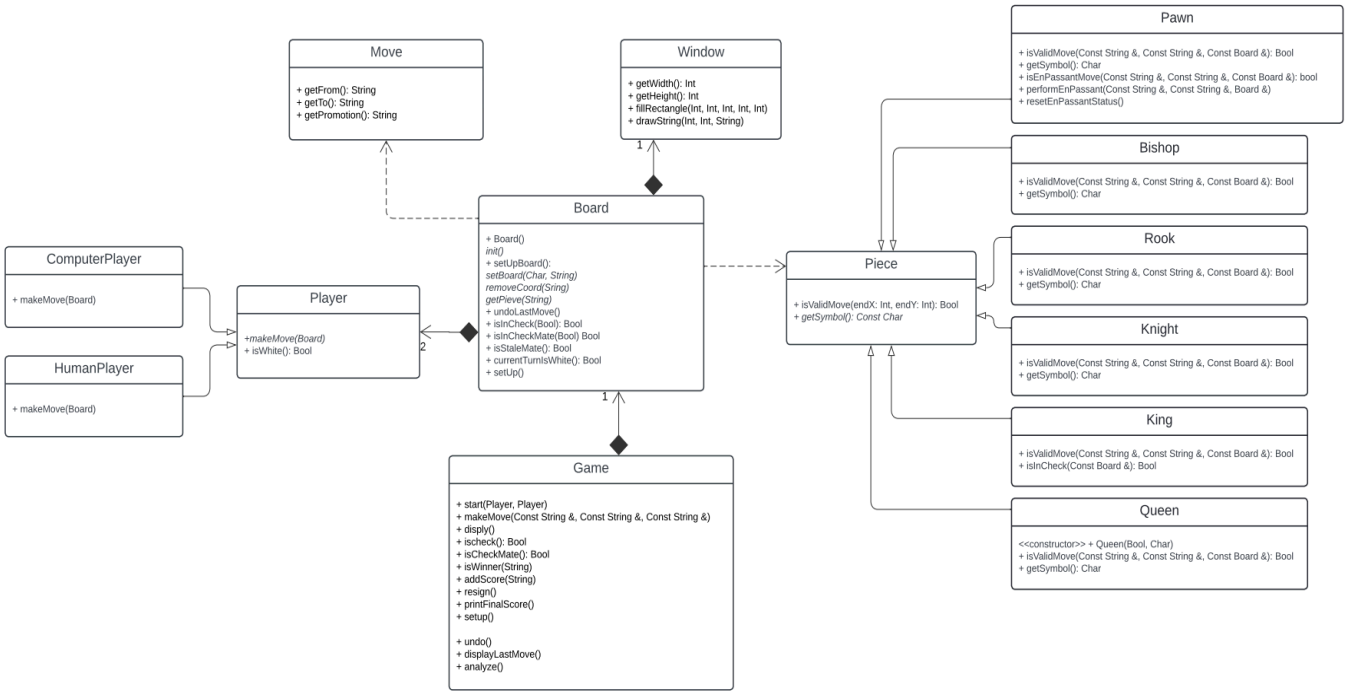
- *Game*

Game class manages the flow of the game, including features like setting up white and black players as either human or computer, checking if the move is valid, or ideally other additional features that will enhance the user's game experience.

- *Piece*

An abstract base class that encapsulates the common fields and methods - essentially the attributes - of all chess pieces. Derived classes include: King, Queen, Rook, Bishop, Knight, Pawn. The specific movement rules and interactions of each piece are managed and implemented by the derived classes.

The UML class diagram below shows the relationship and the design for this project:



Distribution of work

We contemplated over the problem of working on the same components of code together, which may invite confusion of implementations as well as the possibility of merge conflicts, so we decided to take on a more modular approach to work on this project, which means working on different components of the project.

- Patricia will be working on setting up the **player.h** and the two derived classes **computerplayer.h** and **humanplayer.h**, which includes the code for computer level 1-4.
- August will work on all the moves and special moves of all the pieces, which essentially means **piece.h** and **(derived classes).h**. Once done, he will work together with Patricia on computerplayer.h.
- Elaine's part entails setting up the board, game flow control, scoreboard and making moves, and the undo features, which essentially means **board.h**, **game.h** and **window.h.**

Estimated Completion Dates

- July 10th: Initial meeting, choosing project, setting up repository
- July 12th - 19th: Creating interfaces, making UML diagram, plan of attack
- July 20th - 24th: Implementation
- July 24th/25th: Debugging, ensuring all files and implementation sync
- July 25th/26th: Update final UML diagram, design overview
- July 26th: Final submission

Answer To Given Questions:

1. **Chess programs usually come with a book of standard opening move sequences, which list accepted opening moves and responses to opponents' moves, for the first dozen or so moves of the game. See for example https://www.chess. com/explorer which lists starting moves, possible responses, and historic win/draw/loss percentages. Although you are not required to support this, discuss how you would implement a book of standard openings if required.**

To implement a program that displays recommended moves for the first dozen moves of games, we can augment move.h class, making it contain more fields. Other than the existing fields for recording the user's move, we will add the response of the opponent after the move, an int indicating the number of times that player used this move, three int that represent the number of win, draw, and loss after the game is ended.

After the above process is done, we can create an augmented Move object that records the player's move and store the data into the vector. As the game is played multiple times, we will be able to analyse this vector and display possible responses, historic win/draw/loss percentages.

2. **How would you implement a feature that would allow a player to undo their last move? What about an unlimited number of undos?**

We create a move.h that creates a Move object. In the class, we create an object that records the piece that the user uses in each move, an initial coordinate that indicates a piece's original coordinates before the move, and an ending coordinate after the move. We have a bool that checks if there is a promotion. If the promotion is true, then the promoted piece would also be recorded. In addition, if one piece captures another piece, this will also be recorded in the string representing the captured piece.

By doing so, we could create a vector of Move objects that records each move from the users. If the user wants to undo their last move, then the last move would be erased. The board would recover based on the move that is erased. If the user keeps undoing, the process will be repeated until there are no further moves and can be deleted. Then, the user will be forced to stop the game but move instead. This could be done in our board.h and game.h, so we can directly use them in main.cc.

3. **Variations on chess abound. For example, four-handed chess is a variant that is played by four players (search for it!). Outline the changes that would be necessary to make your program into a four-handed chess game. (If it's important to your answer, state whether you're assuming free-for-all or team rules and then answer the question. You don't need to get too specific into the rule set changes in answering the question though; your focus should be more on what would need to be altered at the high level of the design?)**

At a **high-level**, the board.h needs to be altered slightly to accommodate a 14x14 grid instead of 8x8 for a 4-hands chess game. [https://www.chess.com/terms/4-player-chess] Handling this new layout includes initialising the board with the correct dimensions and updating the methods that manage piece placement and movement.

For player.h, the class should be able to handle multiple players instead of just 2. The rotation logic has to be changed to accommodate four players as well, since we are currently using boolean flags to determine the turn of two players.

For pieces.h and game.h, most underlying logic remains the same, and the only changes that should be made is for the movement rules for interaction between 4 players to be accommodate, such as check, checkmate and capturing work.