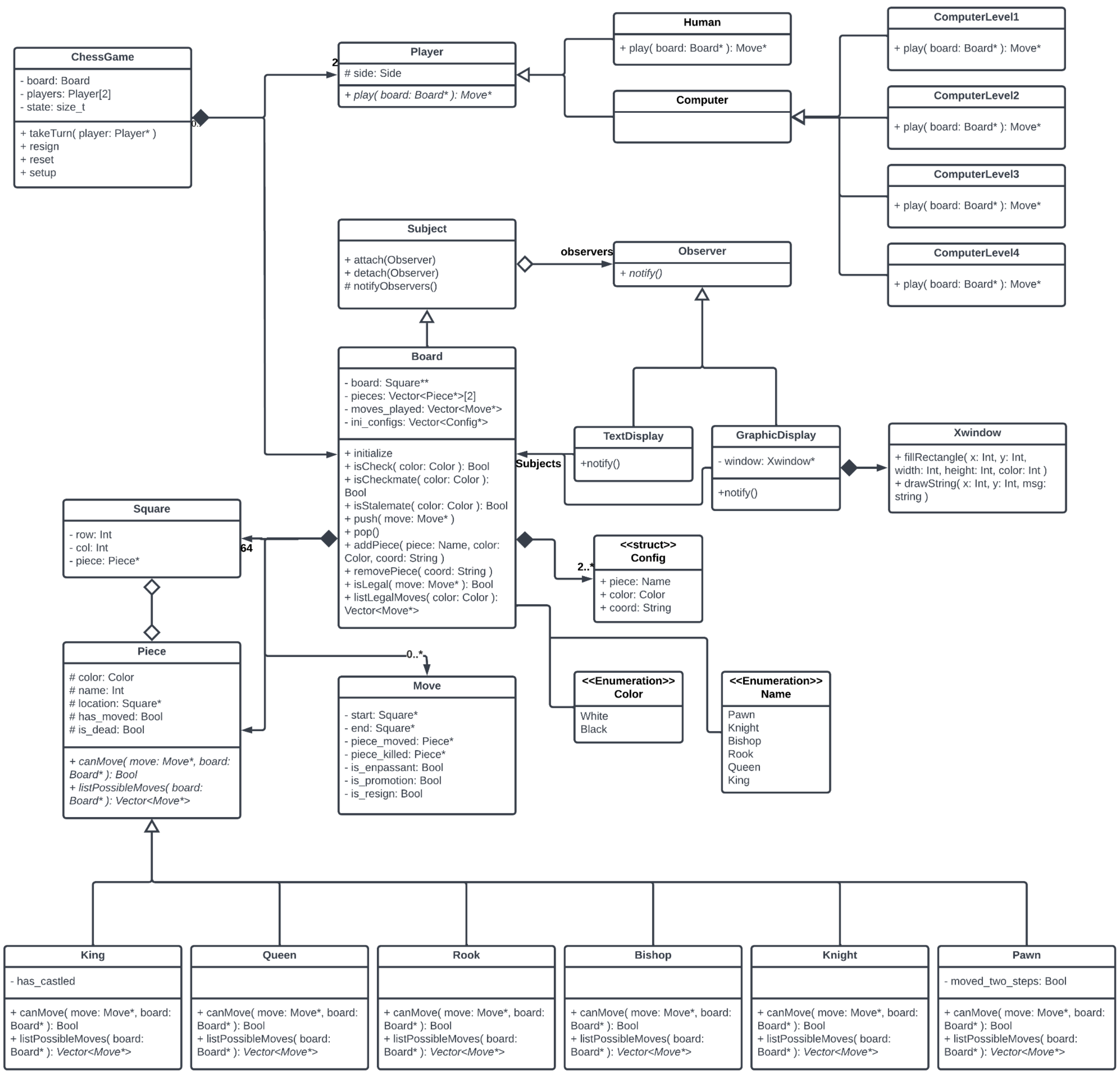
CS246 Fall 2022 Project – Chess

By Yejoon Jung and Taegwon Kim

I. Overview

Upon the release of the final project, we have looked into the project specifications and devised our initial design of the chess program, where we made our best attempt to reflect the notion of object-oriented programming and accommodate the specification of requirements in the assignment.

II. UML


ChessGame
- board: Board
- players: Player[2]
- state: size_t
+ takeTurn(player: Player*)
+ resign
+ reset
+ setup
Player
# side: Side
+ play(board: Board*): Move*
2
Human
+ play(board: Board*): Move*
Computer
ComputerLevel1
+ play(board: Board*): Move*
ComputerLevel2
+ play(board: Board*): Move*
ComputerLevel3
+ play(board: Board*): Move*
ComputerLevel4
+ play(board: Board*): Move*
Subject
+ attach(Observer)
+ detach(Observer)
# notifyObservers()
observers
Observer
+ notify()
Board
- board: Square**
- pieces: Vector<Piece*>[2]
- moves_played: Vector<Move*>
- ini_configs: Vector<Config*>
+ initialize
+ isCheck(color: Color): Bool
+ isCheckmate(color: Color): Bool
+ isStalemate(color: Color): Bool
+ push(move: Move*)
+ pop()
+ addPiece(piece: Name, color: Color, coord: String)
+ removePiece(coord: String)
+ isLegal(move: Move*): Bool
+ listLegalMoves(color: Color): Vector<Move*>
Subjects
TextDisplay
+notify()
GraphicDisplay
- window: Xwindow*
+notify()
Xwindow
+ fillRectangle(x: Int, y: Int, width: Int, height: Int, color: Int)
+ drawString(x: Int, y: Int, msg: string)
Square
- row: Int
- col: Int
- piece: Piece*
64
<<struct>>
Config
2..*
+ piece: Name
+ color: Color
+ coord: String
Piece
# color: Color
# name: Int
# location: Square*
# has_moved: Bool
# is_dead: Bool
+ canMove(move: Move*, board: Board*): Bool
+ listPossibleMoves(board: Board*): Vector<Move*>
0..*
Move
- start: Square*
- end: Square*
- piece_moved: Piece*
- piece_killed: Piece*
- is_enpassant: Bool
- is_promotion: Bool
- is_resign: Bool
<<Enumeration>>
Color
White
Black
<<Enumeration>>
Name
Pawn
Knight
Bishop
Rook
Queen
King
King
- has_castled
+ canMove(move: Move*, board: Board*): Bool
+ listPossibleMoves(board: Board*): Vector<Move*>
Queen
+ canMove(move: Move*, board: Board*): Bool
+ listPossibleMoves(board: Board*): Vector<Move*>
Rook
+ canMove(move: Move*, board: Board*): Bool
+ listPossibleMoves(board: Board*): Vector<Move*>
Bishop
+ canMove(move: Move*, board: Board*): Bool
+ listPossibleMoves(board: Board*): Vector<Move*>
Knight
+ canMove(move: Move*, board: Board*): Bool
+ listPossibleMoves(board: Board*): Vector<Move*>
Pawn
- moved_two_steps: Bool
+ canMove(move: Move*, board: Board*): Bool
+ listPossibleMoves(board: Board*): Vector<Move*>

III. Plan

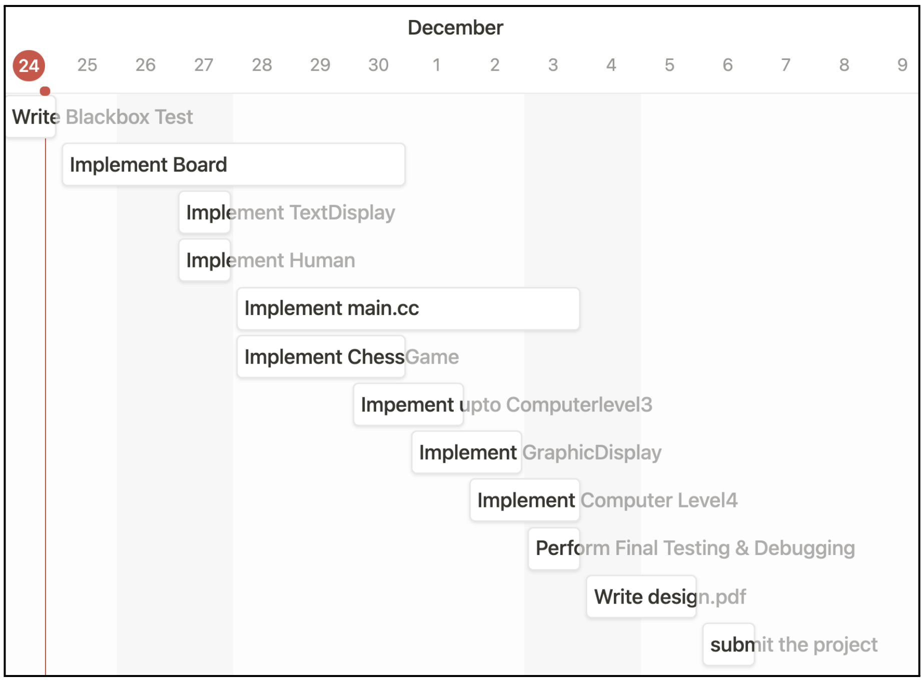
- Flow of tasks (project breakdown)

  Our plan for the chess project is as follows:

  1. Write blackbox test cases for the project before start implementing the project
  2. Implement rudimentary features of Board(initialize, push, pop, ...etc), and other classes required to operate Board, including Square, Move, and Piece and its subclasses.
  3. Implement TextDisplay, and Player and its subclass Human
  4. Implement ChessGame, the client main.cc, and additional features of Board(allow to mutate initial configuration of the board)
  5. Implement Computer and its subclasses up to ComputerLevel3, and GraphicDisplay
  6. Implement Computer Level4
  7. Perform final writebox testing and write design.pdf & revise UML
  8. Submit the final project safely into the marmoset

  This approach was taken since it is more efficient to implement the core part of the program prior to other parts. For example, board is the primary feature of chess where many other parts of the program depend upon, and thus we need to implement it from the very first step. With this approach, we will be able to compile and test the basic features of the project as soon as possible.

- Estimated completion date



  - Implementing the classes is expected to take 9 days, and writing the final document is expected to take 2 days. Summing up, we expect the final project to be completed within 11 days. Starting the project from November Thu. 24, the estimated completion date for the project is Dec. 5.

- Task Distribution

  Since working on the same piece of code at the same time may cause confusion, we decided to divide our tasks by file and classes.

  - Yejoon will be responsible for implementing the class Board, Square, Move, Observer, Subject, Player, and GraphicDisplay.
  - Taegwon will work on the class Piece and its subclasses, TextDisplay, Human, ChessGame, and main.cc.
  - Computer and its subclasses will be distributed appropriately depending on individuals' workload and time left as we progress our project (may drop ComputerLevel4 if time runs out).

  We distributed our tasks in a way that we can both follow the flow of the tasks listed above at the same time without both working on the same piece of code. We expect this will improve our working experience during the project.

IV. Answer to Problems

1. *Question: 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. Although you are not required to support this, discuss how you would implement a book of standard openings if required.*

   Given a set of standard opening move sequences, we implement a class representing a book of standard opening, which reads and translates the sequences into a single tree structure where all the opening moves are arranged in a way that a player can navigate through in a constant time. We can use this class to build computer players that prioritizing the opening moves whenever possible or to help human players to find better moves.

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

   The class Move, representing each move played in the game of chess, contains the information of its corresponding move including the start position, end position, piece moved, piece killed, and whether the move is any of castling, promotion, or en passant. Whenever a player makes a move, Board records the move by creating an instance of class Move representing that move and pushing it into a vector named moves_played. This allows players to undo their moves unlimited times (until there are no more moves to undo) by reverse operating each move and popping it out of the vector moves_played one by one from the back of the vector moves_played. Specifically, when undoing the last move, the Board may go through the following steps (with special moves such as castling must be handled appropriately with additional straps): get the last element of moves_played, retreat the piece moved by the last Move backward to its start position, if there is a piece killed by the Move then revoke the piece to the end position, and finally pops the last move out of the vector moves_played.

3. *Question: 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*

   In order to accommodate the four-handed chess game, we first need to increase the number of players and colors from 2 to 4. It can be done by editing constants and enumerations defined in ChessGame and Board, which automatically updates the array of Players to length 4. Next, we need to make changes on the magnitude and initial configuration of the board so that it matches the rule of four-handed chess. Also, we need to make changes to the command interpreter in client main.cc, so that it can support the commands for four-handed chess games. Finally, we need to make changes on the TextDisplay and GraphicDisplay classes so that it can correctly display the newly configured board with expanded board size and pieces of two newly added colors. Other than these, there will be no other major changes required since most of the control flows, arrays, and functions in the program will depend on the constants and enumerations.