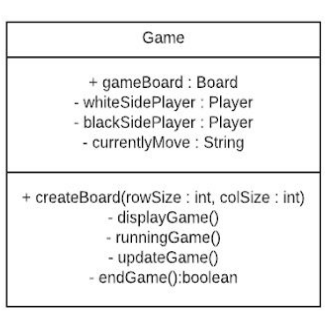
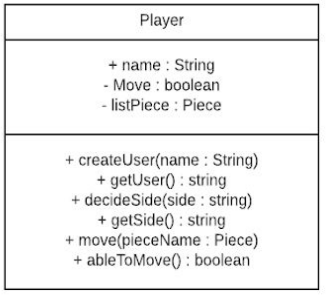
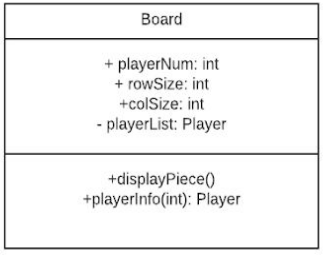
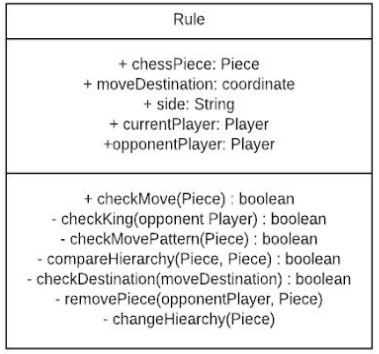
UML class diagrams

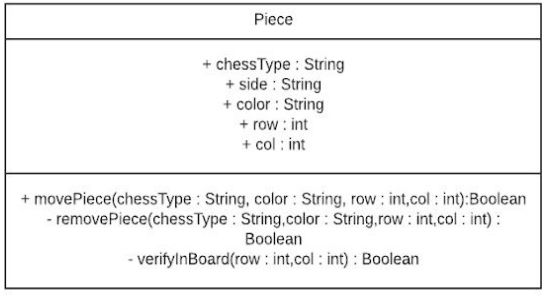
RCS IDs: jianr, liaoy3

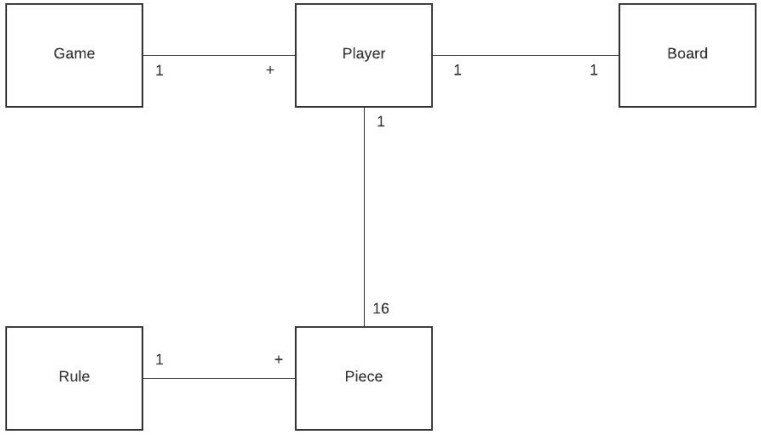


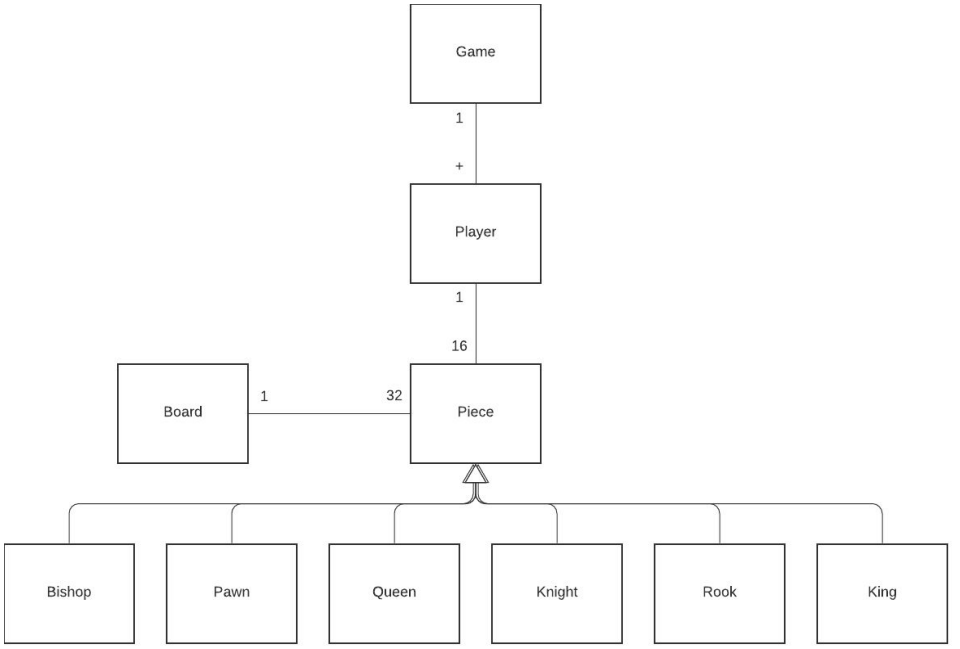












Game
+ gameBoard : Board - whiteSidePlayer : Player - blackSidePlayer : Player - currentlyMove : String
+ createBoard(rowSize : int, colSize : int) - displayGame() - runningGame() - updateGame() - endGame():boolean

Player
+ name : String - Move : boolean - listPiece : Piece
+ createUser(name : String) + getUser() : string + decideSide(side : string) + getSide() : string + move(pieceName : Piece) + ableToMove() : boolean

Board
+ playerNum : int + rowSize : int +colSize : int - playerList : Player
+displayPiece() +playerInfo(int) : Player +checkKing() : boolean

Piece
+ chessType : String + side : String + color : String + row : int + col : int
+ movePiece(chessType : String, color : String, row : int,col : int):Boolean - removePiece(chessType : String,color : String,row : int,col : int) : Boolean - verifyInBoard(row : int,col : int) : Boolean

Bishop
+ hierarchy : int
+ move() + checkHierarchy() : boolean

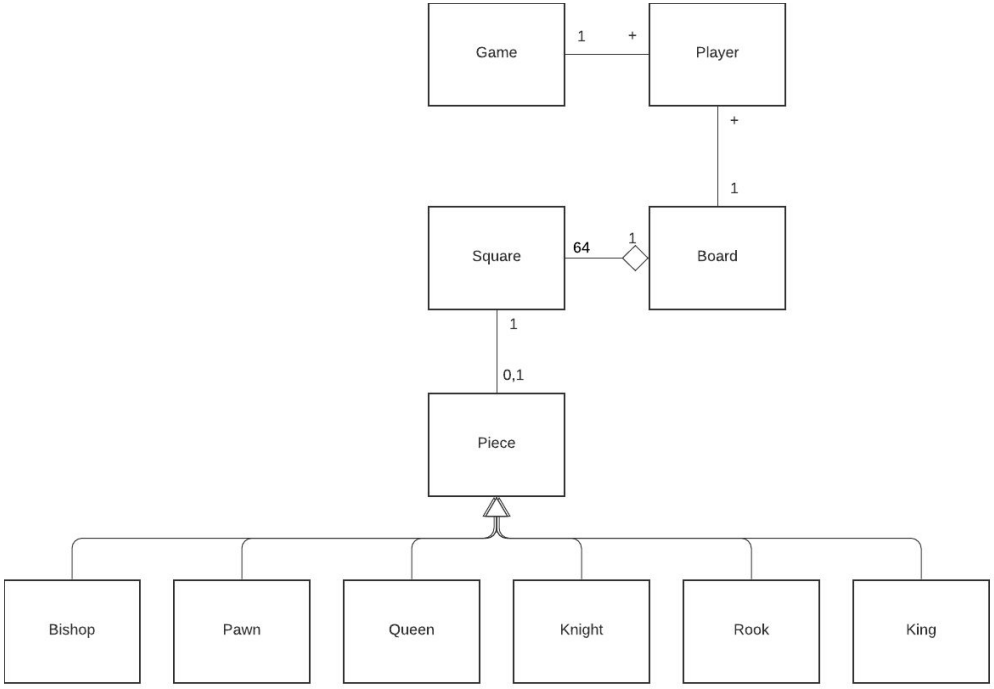
King
+ hierarchy : int
+ move() + checkHierarchy() : boolean

Pawn
+ hierarchy : int
+ move() + checkHierarchy() : boolean + changeStatus()

Rook
+ hierarchy : int
+ move() + checkHierarchy() : boolean

Knight
+ hierarchy : int
+ move() + checkHierarchy() : boolean

Queen
+ hierarchy : int
+ move() + checkHierarchy() : boolean



Game
+ gameBoard : Board - whiteSidePlayer : Player - blackSidePlayer : Player - currentlyMove : String
+ createBoard(rowSize : int, colSize : int) - displayGame() - runningGame() - updateGame() - endGame():boolean

Player
+ name : String + board : Board
+ createUser(name : String) + getUser() : string + decideSide(side : string) + getSide() : string + updateBoard(name : String) + ableToMove() : boolean

Board
+ rowSize : int +colSize : int + suqarelList: : Square - playerList : Player
+ displayPiece() + playerInfo(int) : Player + checkKing() : boolean + getPieces(playerName : Player) + movePiece(pieceName : Piece)

Square
+ chessType : Piece + color : String + row : int + col : int
+ checkPiece() : boolean

Piece
+ chessType : String + side : String + color : String + row : int + col : int
+ movePiece(chessType : String, color : String, row : int,col : int):Boolean - removePiece(chessType : String,color : String,row : int,col : int) : Boolean - verifyInBoard(row : int,col : int) : Boolean

Bishop
+ hierarchy : int
+ move() + checkHierarchy() : boolean

King
+ hierarchy : int
+ move() + checkHierarchy() : boolean

Pawn
+ hierarchy : int
+ move() + checkHierarchy() : boolean + changeStatus()

Rook
+ hierarchy : int
+ move() + checkHierarchy() : boolean

Knight
+ hierarchy : int
+ move() + checkHierarchy() : boolean

Queen
+ hierarchy : int
+ move() + checkHierarchy() : boolean

Key difference between the 3 approaches:

These approaches all have a similar basic structure but they are different in their inheritance, abstraction and use of attributes. Approach 1 uses the attribute “side” under the Piece class, which distinguishes the player side and avoid to create unnecessary Side classes. The Board class serves as a recorder for the game and only interact with the Player class. Rule class is created for the simplicity of each chess, there is no need to create classes for every different type of chess. Approach 2 uses inheritance by adding classes for every different type of chess, which makes the rule inside of each piece type class, it has better visualization and it is more convenient to debug rules under separated classes. Approach 3 uses Square class which supports the implementation of Board class, every element inside of the Board class’s data structure will be a type of Square, so the Board data representation could be simpler.

The optimal UML class:

- Class diagram 1, it provides the best abstraction compare to the other two, and it avoids to create multiple redundant classes for each chess.