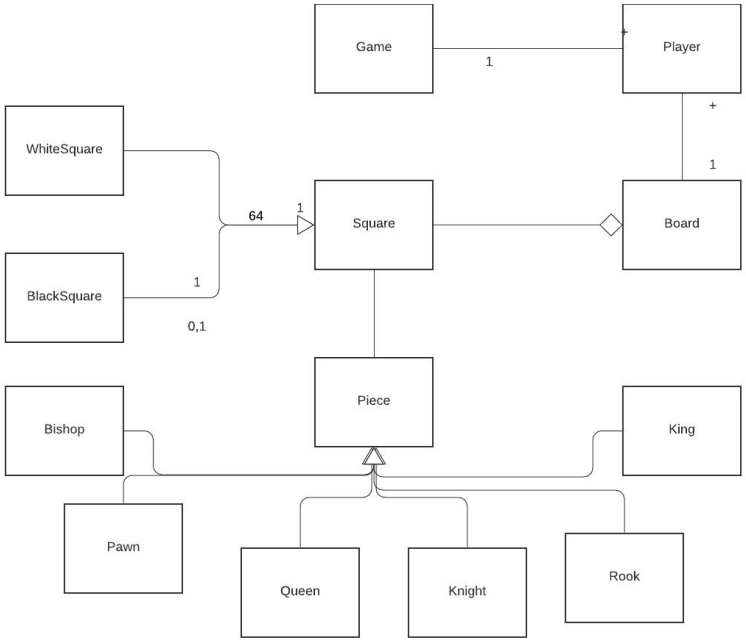
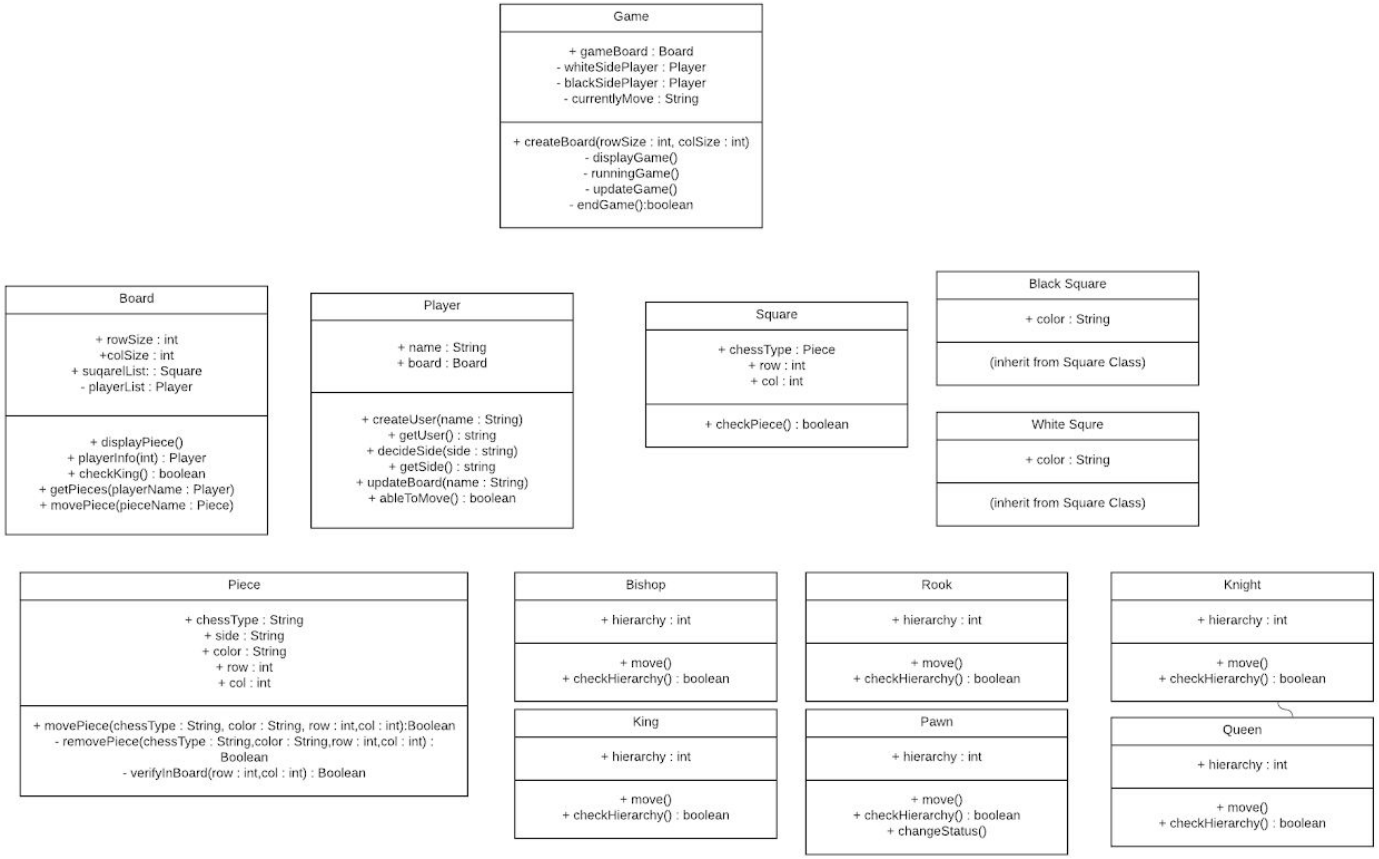
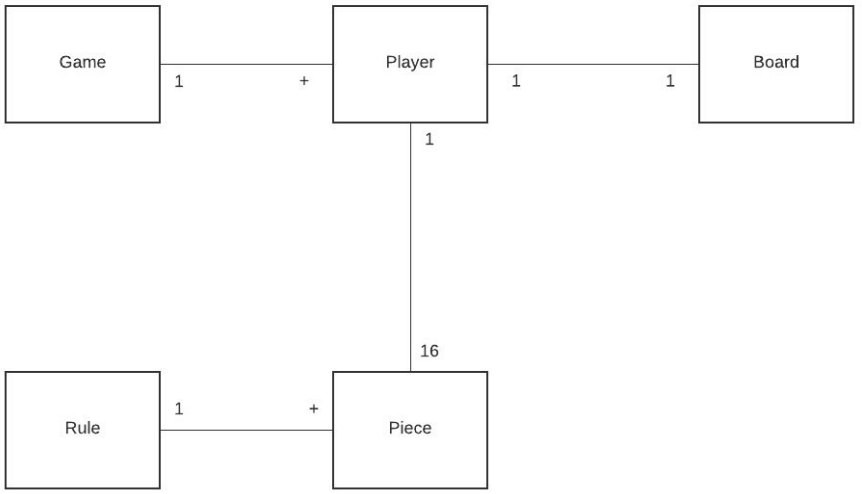
RCS IDs: wangy56, wangy62, zhangx33







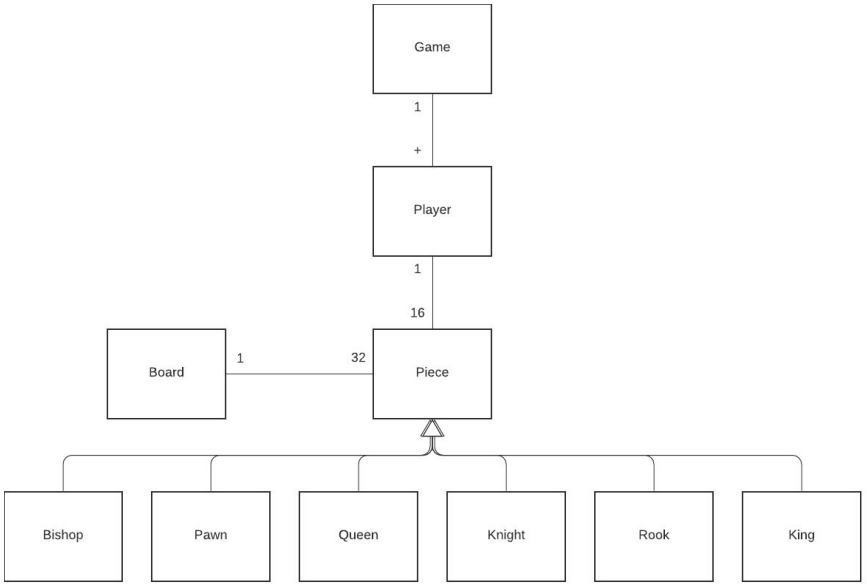
Game
+ gameBoard : Board - whiteSidePlayer : Player - blackSidePlayer : Player - currentlyMove : String
+ createBoard(rowSize : int, colSize : int) - displayGame() - runningGame() - updateGame() - endGame():boolean

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

Rule
+ chessPiece: Piece + moveDestination: coordinate + side: String + currentPlayer: Player +opponentPlayer: Player
+ checkMove(Piece) : boolean - checkKing(opponent Player) : boolean - checkMovePattern(Piece) : boolean - compareHierarchy(Piece, Piece) : boolean - checkDestination(moveDestination) : boolean - removePiece(opponentPlayer, Piece) - changeHiearchy(Piece)

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



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

Key differences:

The first class diagram is the most specific. We make one class to have multiple inheritance classes so that the specific details will be more organized and clear. For example, Piece class has 6 inherence classes which contain the property of the different chess. Different cheeses have their own pattern of movements.Instead of share the common movement, the individual classes for each of them will be easier to distinguish. Once we move to the test step, the code will be easier to modify and check if we face some bugs. In addition, class Square has two inheritance classes which is White Square and Black Square. As we all know Chess can only have two players. Therefore, we divide two players into two classes so that we can track each of them when they try to move the next step so that we will not mess up the information.

In the second class the player controls the piece directly and all the movements are constrained by rules. Rules regulate how games work and when it ends. Board saves the data of pieces. For piece class, the side decides the color of the piece and it decides whether the player can move the piece.

The third class contains games that inherit players and players control pieces on the chess board(different from the white square and black square first approach) that record all the moves. Pieces inherit all individual pieces. We need to create a specific kind piece class for a specific player.

Which we would prefer to use: Class diagram 1

I prefer to use the first Class diagram 1 because the logic of the first one is more organized and easier to understand. Since the information was not merged into the single class, we can fully handle each piece of them. If we face the error or bugs when we move to the test step, we may just need to edit one of the inheritance class. If we merge a whole bunch of information and data into one class, it is really hard for us to find mistakes. The worst situation may rewrite the entire class which is not efficient and also costs a lot of time.