Brian Queen’s Chess: Design Manual

Ryan Bailis, Jim Campbell, Ethan Dunne, Jake Schaeffer

Introduction:

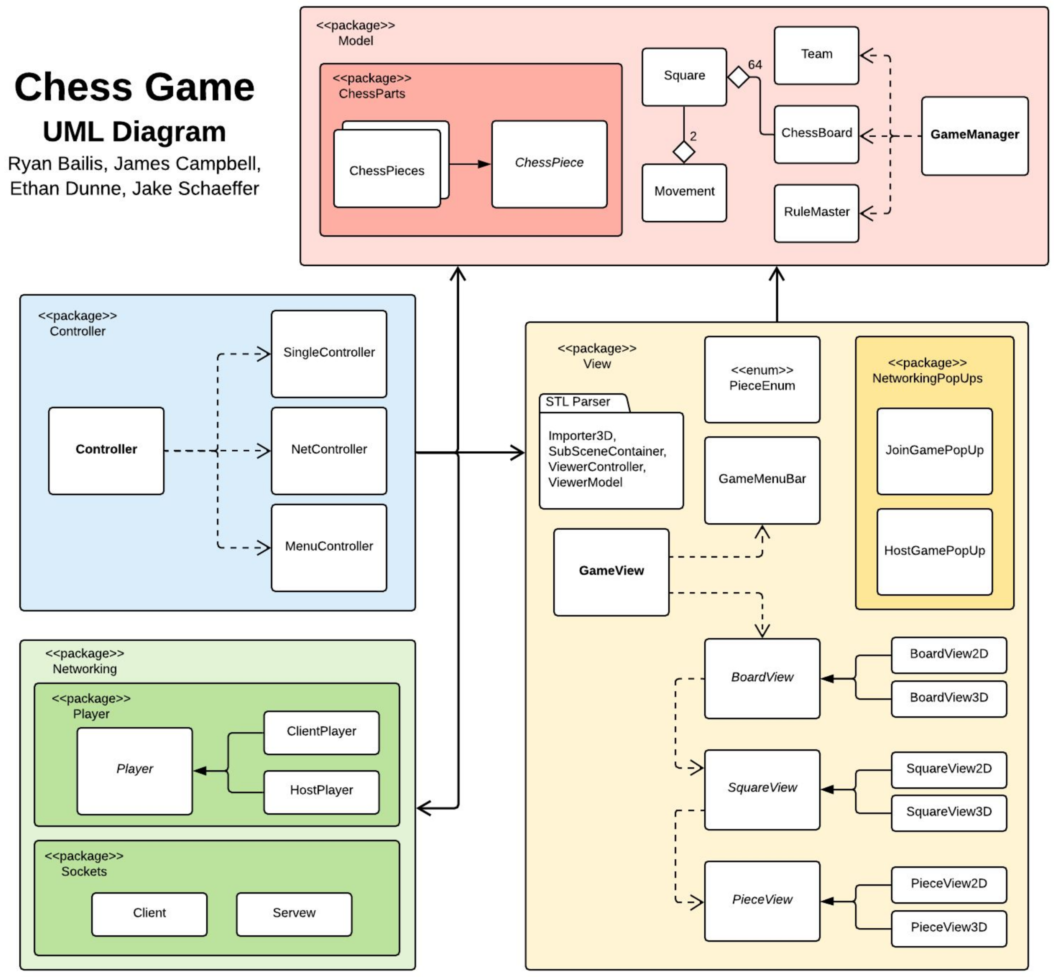
Our final project is a chess simulation that was designed using the Model View Controller design pattern. We also added Networking capabilities that exist outside of the MVC but is implemented inside of it. The Model creates the game itself, holding all of the data for the current chess game. It can be told to move pieces on the board. We have a console representation of the game that represents the states of the board as a string so that we were sure that the model was completely separate from the Controller or View. The View has two main packages, 2D view, and 3D view, these packages have the javaFX visual representations of all of the parts of the model. The Controller has access to both the Model and the View, and using events can change both the Model and the View based off input from the user. The Controller also handles the customization of colors using the View and the Model. The Controller also implements the networking, and following commands from the user will establish itself as a host or client and connect to another system running the same program. It then also handles receiving updates from the other system and sending updates to the other system. All of these parts come together to create a color customizable, 2D/3D, network supporting, chess simulation.

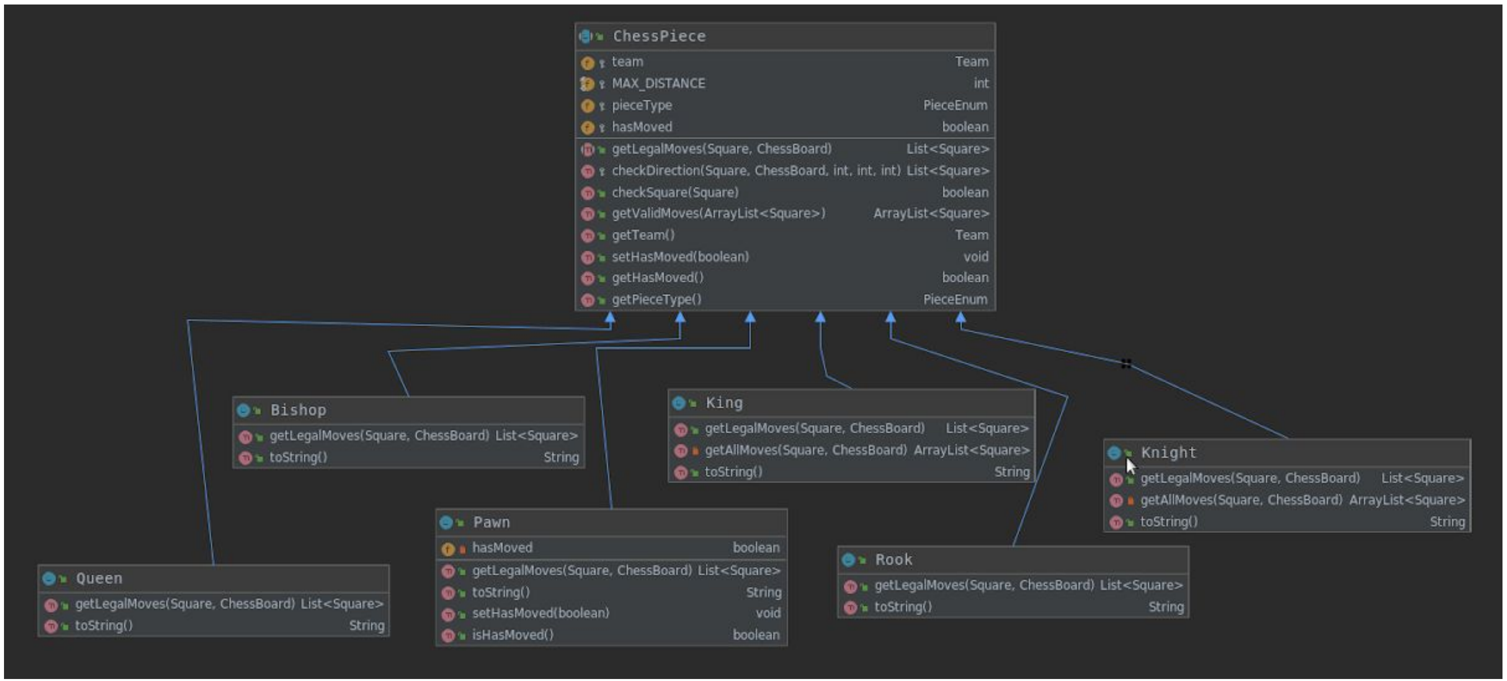
User Stories:

- Be able to not only know the movement pattern of the piece when deciding to make moves, but take into account that you can not move onto a square that a piece on your team occupies.
- Make it so that only one team is able to move at a time, and that the team switches after every move.
- If a piece from one team is moved to a square containing a piece from another team, remove the piece that did not move.
- Design a system for checkmate, and winning the game.
- Be able to restart the game.
- Be able to play against a player on another computer.
- Be able to exit the game

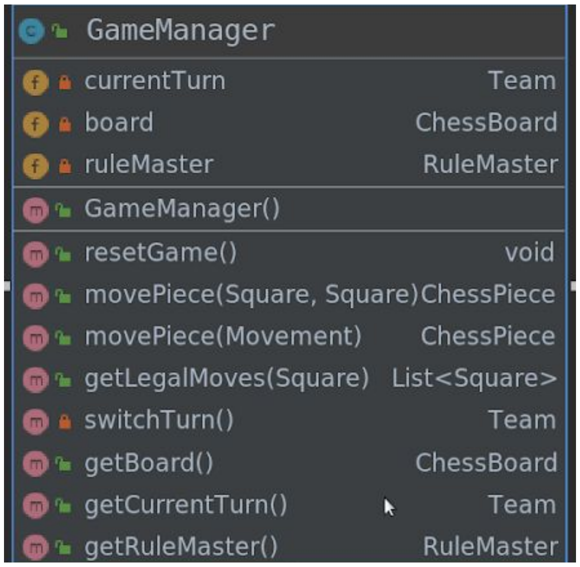
OOD:

The class that runs The game is called ChessGameMain. This class puts together our model, view and controller. The model is GameManager. The view is GameView, and the controller is controller. Each piece is able to move based on a different set of rules. These rules have been created by each piece's getLegalMoves method. Every pieces inherits from ChessPiece, which contains methods like checkDirection and



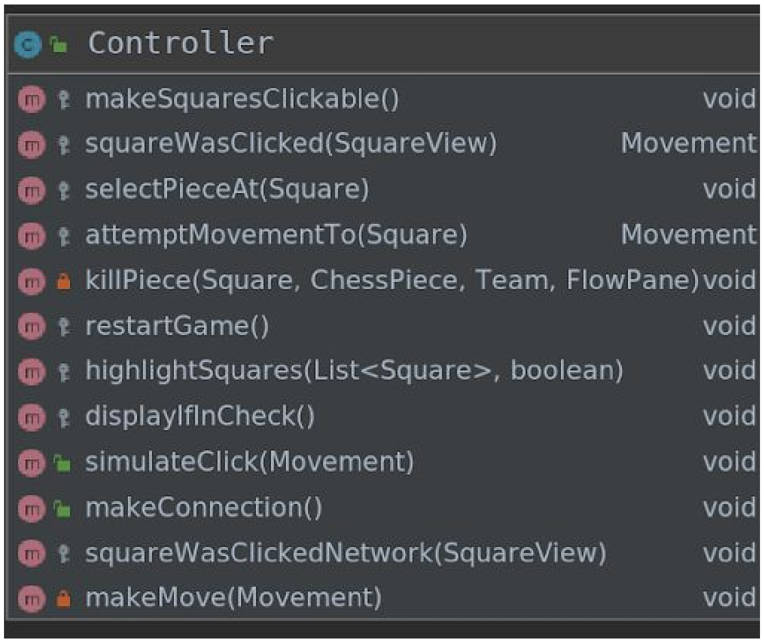

ChessPiece
team Team
MAX_DISTANCE int
pieceType PieceEnum
hasMoved boolean
getLegalMoves(Square, ChessBoard) List<Square>
checkDirection(Square, ChessBoard, int, int, int) List<Square>
checkSquare(Square) boolean
getValidMoves(ArrayList<Square>) ArrayList<Square>
getTeam() Team
setHasMoved(boolean) void
getHasMoved() boolean
getPieceType() PieceEnum
Bishop
getLegalMoves(Square, ChessBoard) List<Square>
toString() String
King
getLegalMoves(Square, ChessBoard) List<Square>
getAllMoves(Square, ChessBoard) ArrayList<Square>
toString() String
Knight
getLegalMoves(Square, ChessBoard) List<Square>
getAllMoves(Square, ChessBoard) ArrayList<Square>
toString() String
Pawn
hasMoved boolean
getLegalMoves(Square, ChessBoard) List<Square>
toString() String
setHasMoved(boolean) void
isHasMoved() boolean
Rook
getLegalMoves(Square, ChessBoard) List<Square>
toString() String
Queen
getLegalMoves(Square, ChessBoard) List<Square>
toString() String

getValidMoves that help to determine where the piece can move on the current board. Each piece also has a team that helps determine when it can be moved or captured, as well as which piece can capture which.



```
GameManager
currentTurn                        Team
board                        ChessBoard
ruleMaster                   RuleMaster
GameManager()
resetGame()                        void
movePiece(Square, Square)ChessPiece
movePiece(Movement)          ChessPiece
getLegalMoves(Square)      List<Square>
switchTurn()                       Team
getBoard()                   ChessBoard
getCurrentTurn()                   Team
getRuleMaster()              RuleMaster
```

The Game Manager is what provides the functionality that enables us to actually run a game. The movePiece method is the most important method because it is what moves pieces around the board. The GameManager also provides an option to reset the game, which is used when restarting.



```
Controller
makeSquaresClickable()                               void
squareWasClicked(SquareView)                     Movement
selectPieceAt(Square)                                void
attemptMovementTo(Square)                        Movement
killPiece(Square, ChessPiece, Team, FlowPane)void
restartGame()                                        void
highlightSquares(List<Square>, boolean)              void
displayIfInCheck()                                   void
simulateClick(Movement)                              void
makeConnection()                                     void
squareWasClickedNetwork(SquareView)                  void
makeMove(Movement)                                   void
```

The Controller allows us to have a game that is fully integrated with the GUI. Some important methods in the include makeSquaresClickable, which allows a user to click on a square. there are helper methods also used in order to actually move the piece using the mouse, like squareWasClicked. highlightSquares is used to indicate possible moves to a user. displayIfInCheck allows the user to see if they are in check, and of course restartGame allows us to completely reset the view and the model. In order to do networking, the last four methods are used to simulate the other user’s actions on the current user’s board in the model and GUI.