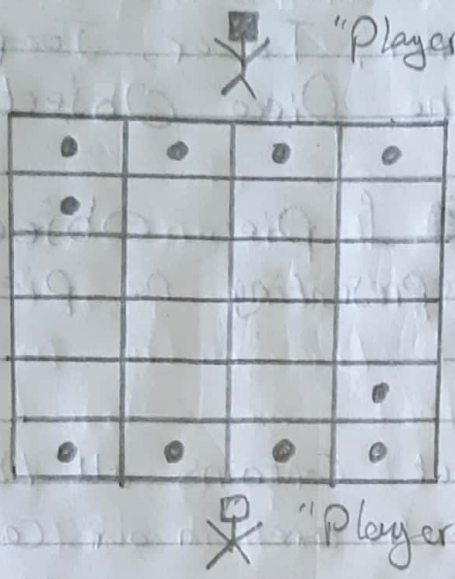
Chess Design



1] Objects of Chess :

⇒ The following are the Objects that can be identified from a Chess Game : ○

- Chess Piece, this can be a ~~(Pawn, Tower, Runner, Jumper, King)~~ (Pawn, Tower, Runner, Jumper, King)

- Spot, a place on the Chess Board that may have a piece on it or can be empty

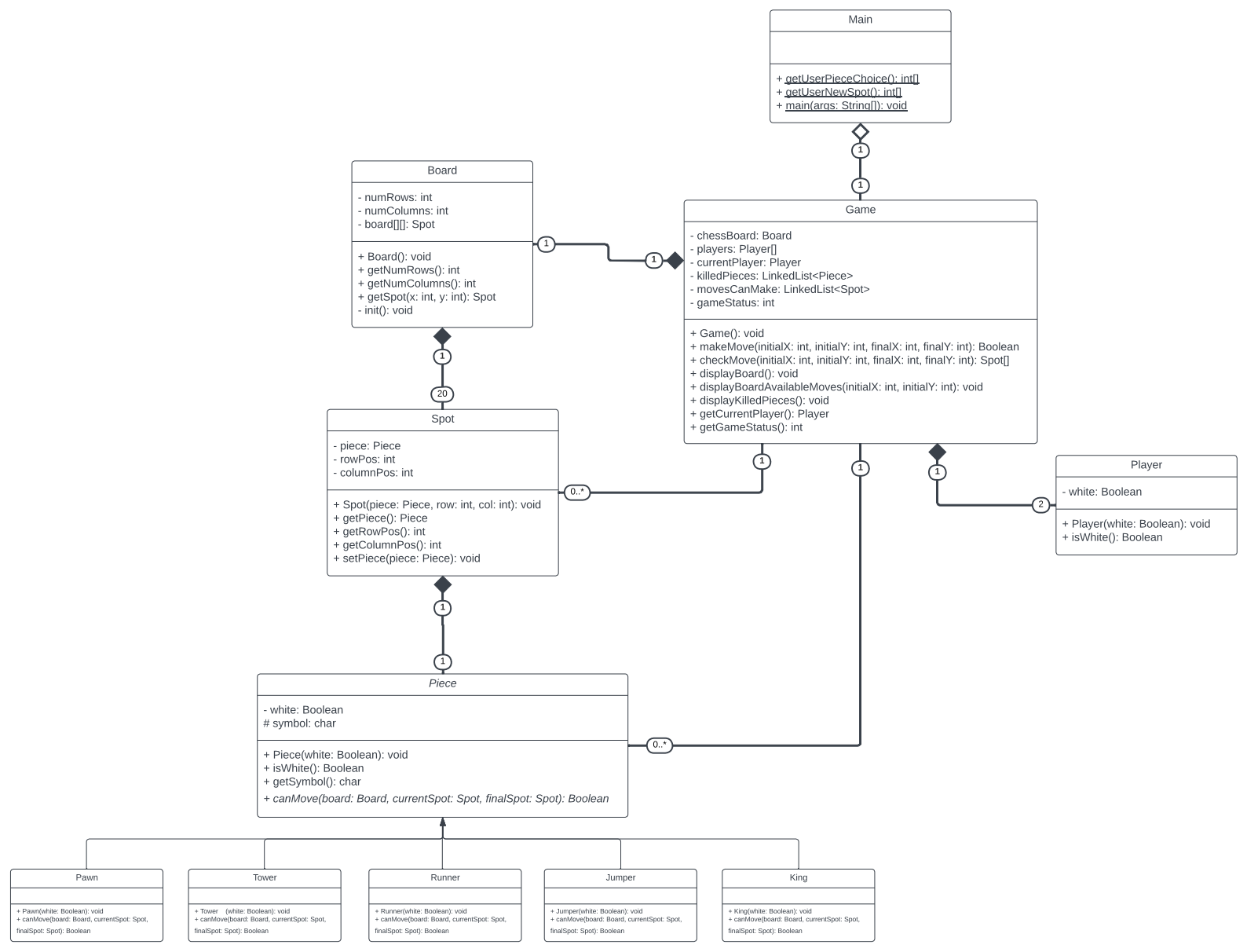
- player, a person who makes a move on the Chess pieces Based on the Color Side he Choose

- Board, an Object representing ~~the~~ all the Spots in a Chess Board

- Game, an Object Controling the Game flow

2) Relationships Between the Objects:

- The Pawn, Runner, Tower, Jumper Objects all inherit from the Piece Object.
- Every Spot holds 1 Piece Object or a null Object representing no pieces on the Spot.
- The Board Object contains all the Spots in Chess Game which are in our case $5 \times 4 = 20$ Spots.
- The Game holds 1 Board Object, and holds 2 Player Objects, and holds 0 or Many Piece Objects representing the Pieces that have died during the Game, and holds 0 or Many Spot Objects representing the Spots a Chess Piece, choosen by a player, can move to.


Main
+ getUserPieceChoice(): int[]
+ getUserNewSpot(): int[]
+ main(args: String[]): void
1
1
Board
- numRows: int
- numColumns: int
- board[][]: Spot
+ Board(): void
+ getNumRows(): int
+ getNumColumns(): int
+ getSpot(x: int, y: int): Spot
- init(): void
1
1
Game
- chessBoard: Board
- players: Player[]
- currentPlayer: Player
- killedPieces: LinkedList<Piece>
- movesCanMake: LinkedList<Spot>
- gameStatus: int
+ Game(): void
+ makeMove(initialX: int, initialY: int, finalX: int, finalY: int): Boolean
+ checkMove(initialX: int, initialY: int, finalX: int, finalY: int): Spot[]
+ displayBoard(): void
+ displayBoardAvailableMoves(initialX: int, initialY: int): void
+ displayKilledPieces(): void
+ getCurrentPlayer(): Player
+ getGameStatus(): int
1
20
Spot
- piece: Piece
- rowPos: int
- columnPos: int
+ Spot(piece: Piece, row: int, col: int): void
+ getPiece(): Piece
+ getRowPos(): int
+ getColumnPos(): int
+ setPiece(piece: Piece): void
0..*
1
1
Player
- white: Boolean
+ Player(white: Boolean): void
+ isWhite(): Boolean
1
2
1
1
Piece
- white: Boolean
symbol: char
+ Piece(white: Boolean): void
+ isWhite(): Boolean
+ getSymbol(): char
+ canMove(board: Board, currentSpot: Spot, finalSpot: Spot): Boolean
0..*
Pawn
+ Pawn(white: Boolean): void
+ canMove(board: Board, currentSpot: Spot, finalSpot: Spot): Boolean
Tower
+ Tower (white: Boolean): void
+ canMove(board: Board, currentSpot: Spot, finalSpot: Spot): Boolean
Runner
+ Runner(white: Boolean): void
+ canMove(board: Board, currentSpot: Spot, finalSpot: Spot): Boolean
Jumper
+ Jumper(white: Boolean): void
+ canMove(board: Board, currentSpot: Spot, finalSpot: Spot): Boolean
King
+ King(white: Boolean): void
+ canMove(board: Board, currentSpot: Spot, finalSpot: Spot): Boolean

4] Sequence Diagram:

a) The most important Use Case from the Requirements Analysis was Moving a Chess Piece

- Objects involved in Moving a Chess Piece

1) Main
2) Game
3) Board
4) Player
5) Spot
6) Piece (and all its subclasses)

- Events that Occur when Moving a Chess Piece

1) User Input is taken in the Main Object
2) The User Input are passed to the Game Object through the makeMove() method of the Game Object

3) The MakeMove() method first validates the User input by checking the boundraies of the numbers inputted by The User, and for this we use the Board Object to get the dimensions of the Chess Board using the getNumRows() & getNumColumns() methods, and if the User input is valid the getSpot() Method

4) Then the makeMove() method validates if the Choosen Spot is not empty of the Board Object is used to get the Spots at the Specified User Input

4) Then the makeMove() method validate if the choosen Spot Objects holds an empty Chess Piece, this is done using the ~~getChar~~ getPiece() method of the Spot Object

5) Then the makeMove() method validate if the Choosen Piece at the Spot has the Same Color as the Current Player Object using the isWhite() Method of the Piece Object, and & isWhite() method of the Player Object

6) Then the makeMove(), uses the canMove() Method of the Piece Object to check that the Specified Piece can move from the current Spot to the final Spot

7) If the canMove() Method of the Piece object returns True, then we check if the final Spot Contains another Chess Piece using the getPiece() Method of the Spot Object

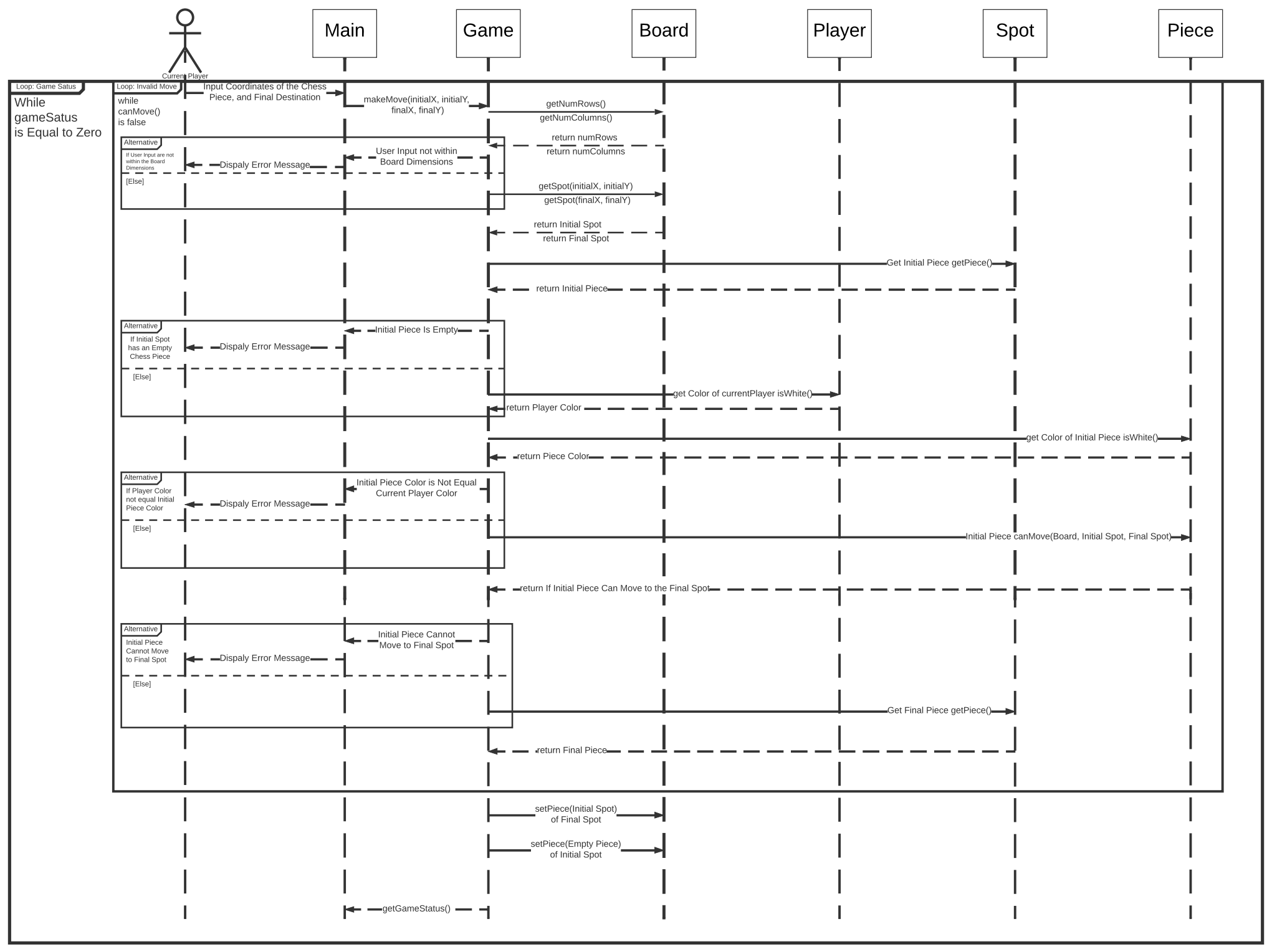
8) If the final Spot Contains a Chess Piece it is removed and added to the KilledPieces Property of the Game Object

10) We update the ~~Fro~~ Chess Piece in the final Spot to the Chess Piece in the Current Spot using the setPiece() & getPiece() methods of the Spot Object, and Set the Chess Piece in the Current Spot to an emptyPiece, using the setPiece() Method of the Spot Object

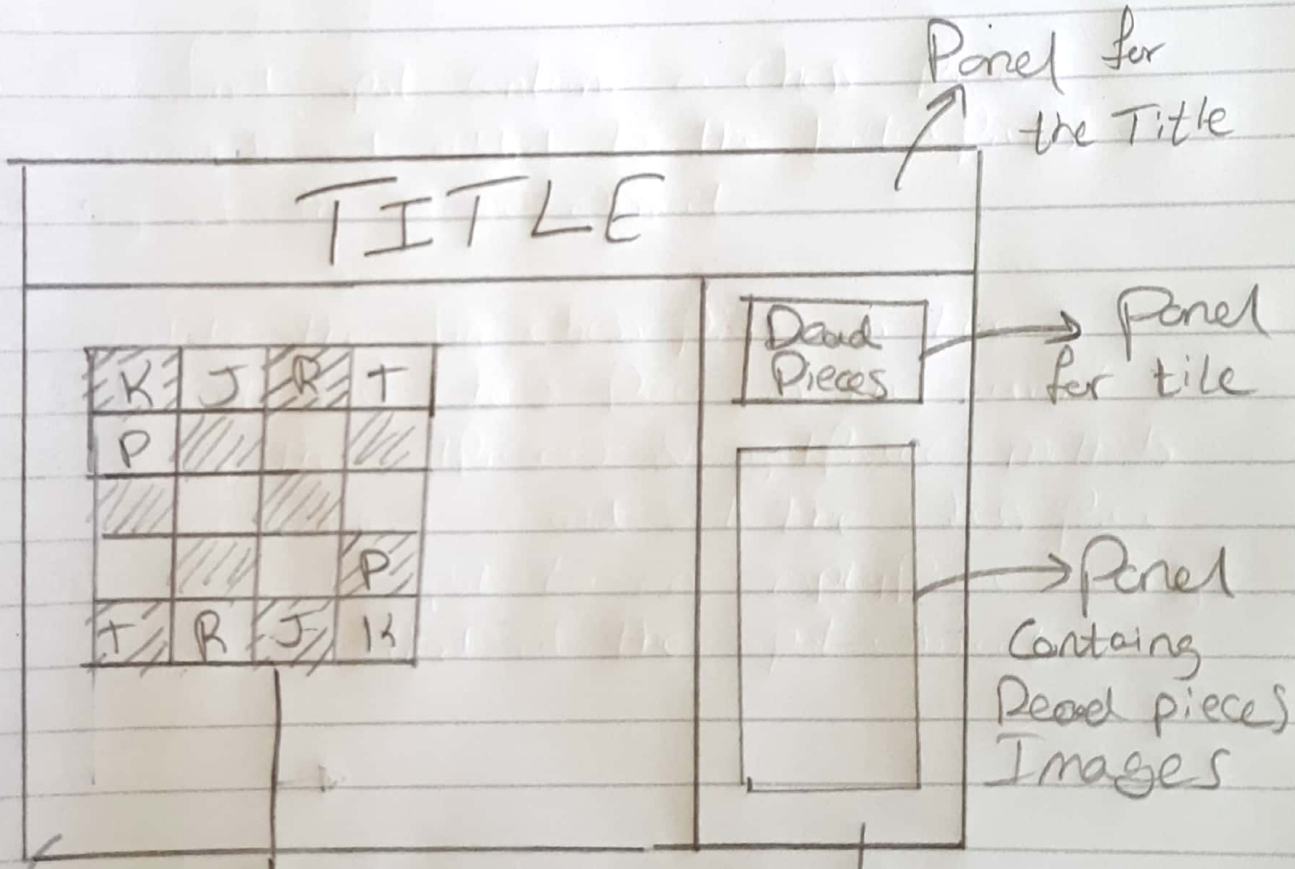
9) If the final Spot Contains a ~~Ches~~ King Chess Piece ~~the gameStatus of the Game Object~~ the Current Player Color is checked using the isWhite() method of the Player Object, if the Current Player was white gameStatus Property is Set to 1, else gameStatus Property of the Game Object is set to 2

11) The current player is set equal to the other player, and for this we use the players[] array property of the Game Object

12) The Main Object gets the gameStatus using the getGameStatus() Method of the Game Object ~~end if the Status is~~ after each move made, and if the gameStatus is not equal 0, the game ends and the player who won is printed


Current Player
Main
Game
Board
Player
Spot
Piece
Loop: Game Satus
While gameSatus is Equal to Zero
Loop: Invalid Move
while canMove() is false
Input Coordinates of the Chess Piece, and Final Destination
makeMove(initialX, initialY, finalX, finalY)
getNumRows()
getNumColumns()
return numRows
return numColumns
Alternative
If User Input are not within the Board Dimensions
Dispaly Error Message
User Input not within Board Dimensions
[Else]
getSpot(initialX, initialY)
getSpot(finalX, finalY)
return Initial Spot
return Final Spot
Get Initial Piece getPiece()
return Initial Piece
Alternative
If Initial Spot has an Empty Chess Piece
Dispaly Error Message
Initial Piece Is Empty
[Else]
get Color of currentPlayer isWhite()
return Player Color
get Color of Initial Piece isWhite()
return Piece Color
Alternative
If Player Color not equal Initial Piece Color
Dispaly Error Message
Initial Piece Color is Not Equal Current Player Color
[Else]
Initial Piece canMove(Board, Initial Spot, Final Spot)
return If Initial Piece Can Move to the Final Spot
Alternative
Initial Piece Cannot Move to Final Spot
Dispaly Error Message
Initial Piece Cannot Move to Final Spot
[Else]
Get Final Piece getPiece()
return Final Piece
setPiece(Initial Spot) of Final Spot
setPiece(Empty Piece) of Initial Spot
getGameStatus()

5] How will the Graphical User Interface of the Chess Game Look like:



Resources Used : Geeks for Geeks

URL : https://www.geeksforgeeks.org/design-a-chess-game/