Team: Tyler Tafoya, Mitch Zinser, Elanor Hoak, Anna Yudina

Title: Chess Xpress

Project Summary: This project will attempt to create a graphical implementation of the classic game of Chess. This implementation will allow two players to play and take turns, ending in either a tie or one player winning. The board and pieces will be a 2d representation of the board, with players able to click to move pieces.

Project Requirements:

- System must allow two players to play the game
- Game must end in either checkmate or stalemate
- Players must be able to move pieces
- When a pawn reaches the opposite end, it must be able to promote
- (Stretch) Players must be able to change their name
- (Stretch) Show valid moves when selecting a piece
- The system must be able to track which pieces each player has lost
- Pieces can only move following defined move-based rules
- If a player captures a piece, the captured piece must be removed from play
- If a player exposes a “check” then the move must be undone
- The system must display a 2D representation of the chess board
- The system must display a 2D representation of the chess pieces
- Pieces must respect “capture logic”
    - Ex. a bishop cannot move from a1-h8 if there's a piece sitting on c3.
- The system must keep track of wins and losses
- Each player must begin the game with 16 pieces (8 pawns, a king, a queen, two rooks, two bishops, two knights)

Business Requirements
None.

User Requirements

ID	Requirement Description	Topic	User	Priority
UR-01	Must be a “New Game” option asking for names of Player1 (white) and Player 2 (black)	Initialize	Player	Medium
UR-02	A game is in progress with move options until endgame	Play	Player	High
UR-03	The game has ended with action to	Play	Player	Medium

	choose from (replay or quit)			
UR-Stretch01	Users can have their own account with personal statistics	Access	Player	Low

Functional Requirements

ID	Requirement	Topic	Priority
FR-01	Must be able to check if a move is valid or not (piece and destination)	Play	High
FR-02	Must be able to end the game when one player is checkmated by the other player	Play	High
FR-03	Must allow a pawn, if it reaches the opposite end of the board, to be switched out for another piece	Play	High
FR-04	Must be able to track which pieces the player has lost, thus can't use them	Play	Medium
FR-05	Must be able to remove a piece, when it has been captured	Play	High
FR-06	Must start a new game with 32 total pieces; 16 per each player.	Play	High
FR-Stretch01	Must be able to track the wins and losses of each player	Logistics	Medium
FR-Stretch02	System can present a player their possible move options for a chosen chess piece	Play	Low

Non-Functional Requirements

ID	Requirement	Topic	User	Priority
NR-01	Interface must be simple enough, so that every player can interact with it	Interface Design	Player	High
NR-02	Each piece must be placed, and it can't be moved until its turn, if it's selected	Reliability	Player	High
NR-03	System must display all of the pieces	Reliability	Player	Medium

	at the beginning, and as the game continues all pieces that are in play			
NR-Stretch01	Players can't change the number of games they won/loss, only the system administrator can do this	Security	Player	Low
NR-Stretch02	A player can't access the other team's pieces, and move them	Security	Player	Low

Users and Tasks:

Use Case Diagrams: How many different types of users will the system have? What tasks do they need to accomplish with the system? Document how the system will support each task via a use case. Try to think of problems that can occur and document them. Provide use case diagrams and use case documents.

Use Case ID:	UC-01
Use Case Name:	Enter Names
Description:	The users can start a new game or exit program from menu

Actors:	Player 1, Player 2		
Pre-Conditions:	No game is active; on main menu screen		
Post-Conditions:	Game initialized and ready for white's first move		
Frequency of Use:	Every time the users want to play		
Flow of Events:		**Actor Action**	**System Response**
	1	Player 1 enters name	Name saved for display for Player 1
	2	Player 2 enters name	Name saved for display for Player 2.
	3	User clicks "Play"	Initialize new game
Variations:	Will not need to have this step if we allow the "Replay" option at the end - names will be kept		

Notes and Issues:	Should the player be able to load a saved game? Login information required?

Use Case ID:	UC-02
Use Case Name:	Player's Move
Description:	The user takes a turn to move a chess piece, updating the board state, possibly changing the number of pieces, and finally turning the next move to the other player.

Actors:	Player 1 or Player 2		
Pre-Conditions:	Game is initiated and ready for user input		
Post-Conditions:	Board updates, turn is changed to the opponent		
Frequency of Use:	Every time a player takes a turn		
Flow of Events:		**Actor Action**	**System Response**
	1	Player selects a piece	Check if piece is legal.
	2	Player selects destination	Check if destination is legal. If so, move selected piece to destination, update board and pieces list, turn over to next player.
Variations:	If either the piece or destination is not valid, then the move restarts.		
Notes and Issues:	Cannot make a move that puts player into check. Piece should check for special conditions (promotion for pawns, castling, en passant)		

Use Case ID:	UC-03
Use Case Name:	Game End
Description:	At the end of a game, the user is given an announcement and options

Actors:	Player 1 or Player 2		
Pre-Conditions:	Game has ended in checkmate or stalemate		
Post-Conditions:			
Frequency of Use:	Every time a game has ended		
Flow of Events:		**Actor Action**	**System Response**
	1	Player select “Replay”	Initialize new game with same Player names
	2	Player selects “Quit”	Program exits
Variations:	Upon endgame, if using UC-Stretch01, update W/L/D statistics for Player 1		
Notes and Issues:	No need to re-enter names for replay. If using UC-Stretch01, then the program should exit to the login screen instead of quitting the program		

Use Case ID:	UC-04
Use Case Name:	Piece is captured
Description:	When a piece is captured, remove piece from board

Pre-Conditions:	Player is in position to take opponent's piece		
Post-Conditions:	Captured piece is removed from board and attacking piece’s position is updated		
Frequency of Use:	Whenever a player takes a piece		
Flow of Events:		**Actor Action**	**System Response**
	1	Player moves piece on top of opponents piece	Remove captured piece from game board, remove captured piece from list of active pieces. Attacking piece’s position is updated. Board is updated.
Variations:			

Notes and Issues:	Opponent's piece must be in one of the valid move spaces for the selected piece; precheck for this

Use Case ID:	UC-05
Use Case Name:	Player is in check
Description:	When a player is in check, only allow them to make moves to get out of check

Pre-Conditions:	Player is in check		
Post-Conditions:	Player is no longer in check		
Frequency of Use:	Every time a player puts the other player in check		
Flow of Events:		**Actor Action**	**System Response**
	1	Player2 puts other Player1 in check	Check for Player1 in in check. Display that the player is in check.
	2	Player1 moves out of check	Check that move doesn't keep player in check, if so allow move. Update display.
Variations:	Player is in checkmate (see UC-06)		
Notes and Issues:			

Use Case ID:	UC-06
Use Case Name:	Player is in checkmate
Description:	When player is in checkmate, end game

Pre-Conditions:	Player is in checkmate

Post-Conditions:	Game end screen		
Frequency of Use:	End of game (once per game)		
Flow of Events:		**Actor Action**	**System Response**
	1	Player puts other player in checkmate	Check that move puts player in checkmate, if so display checkmate and end of game screen (UC-03)
Variations:			
Notes and Issues:	Stalemate?		

Use Case ID:	UC-07
Use Case Name:	Main menu button
Description:	When a player clicks the menu button, bring them back to the main menu

Pre-Conditions:	Game is initialized		
Post-Conditions:	Game ends and player is brought to main menu		
Frequency of Use:	Whenever player clicks button		
Flow of Events:		**Actor Action**	**System Response**
	1	Player clicks the menu button	Game is ended, main menu screen is displayed.
Variations:			
Notes and Issues:	There is no safety on the button, so when a user clicks it, the game is immediately ended		

Use Case ID:	UC-08
Use Case Name:	Quit button

Description:	When a user clicks the quit button, exit the game

Pre-Conditions:	Game is initialized		
Post-Conditions:	Game exits		
Frequency of Use:	Whenever player clicks button		
Flow of Events:		**Actor Action**	**System Response**
	1	Player clicks the exit button	Game is ended and window is closed, exiting game
Variations:			
Notes and Issues:	There is no safety on the button, so when a user clicks it, the game is immediately ended		

Use Case ID:	UC-09
Use Case Name:	Turn ends
Description:	When a player's turn is done, update the current player display

Pre-Conditions:	Player finishes taking a turn		
Post-Conditions:	Current player is updated, it is other player's turn		
Frequency of Use:	At the end of every turn		
Flow of Events:		**Actor Action**	**System Response**
	1	Player finishes their turn by making a valid move	Current player switches to other player, display is updated to show it is other player's turn
Variations:	Player puts other player into checkmate (See UC-06)		
Notes and Issues:			

Use Case ID:	UC-10
Use Case Name:	Pawn Promotion
Description:	When a pawn reaches the other side of the board, promote it.

Pre-Conditions:	Pawn reaches other side of board		
Post-Conditions:	Pawn is promoted to be another piece		
Frequency of Use:	Whenever a pawn reaches the opposite side of the board		
Flow of Events:		**Actor Action**	**System Response**
	1	Player moves pawn to opposite side of board	Check if move is valid. Display promotion screen
	2	Player selects piece to promote to	Remove pawn from board and piece list, add piece selection as new piece on the board in pawn's place. Update display.
Variations:			
Notes and Issues:			

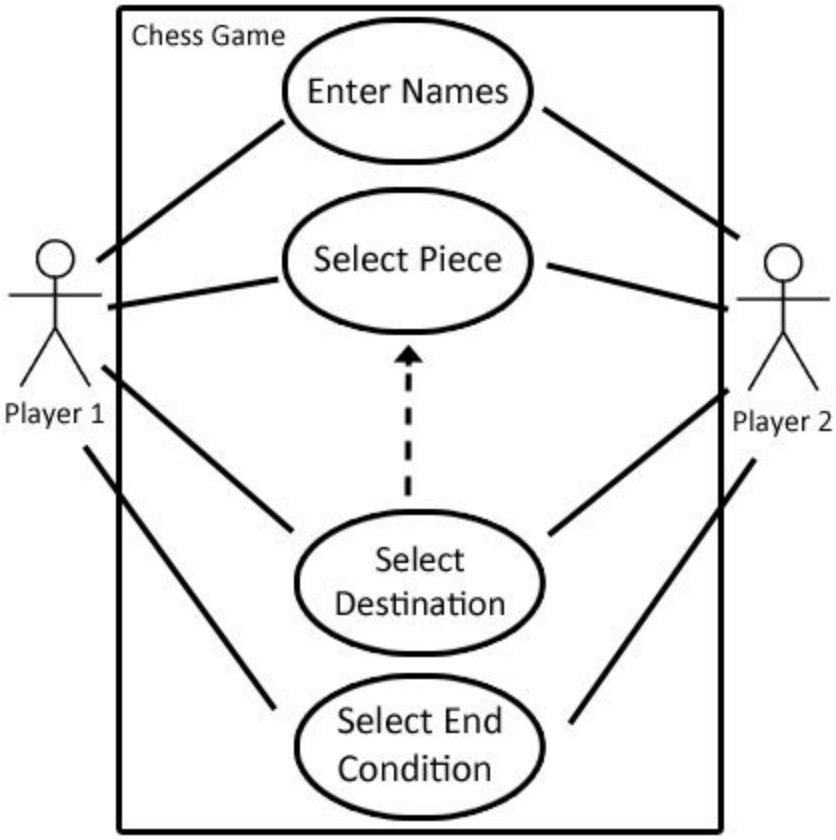
Use Case ID:	UC-Stretch01
Use Case Name:	Sign in to play
Description:	The user can sign in with a custom name and see their personal information or choose to play a new game

Actors:	User		
Pre-Conditions:	User is not signed in		
Post-Conditions:	User can see their scoreboard and choose to start a new game		
Frequency of Use:	Every time the user starts the program		
Flow of Events:		**Actor Action**	**System Response**

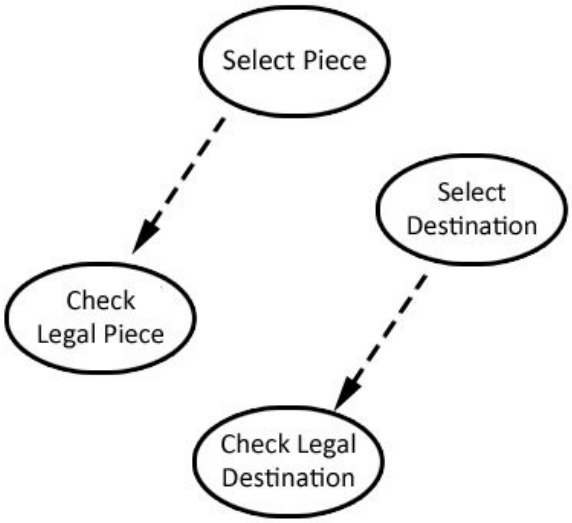
	1	Enters user information	Shows main menu and scoreboard information
	2	Selects New Game	Initialize game (UC-01)
Variations:	Possibly a different menu to show number of wins, loses, and draws. Also will need to alter UC-01 to not enter Player 1 name since they are logged in.		
Notes and Issues:	Requires a database and login interface		

Use Case ID:	UC-Stretch02
Use Case Name:	Show available moves
Description:	When a player clicks on one of the pieces, find all available moves for that piece and display them on the board

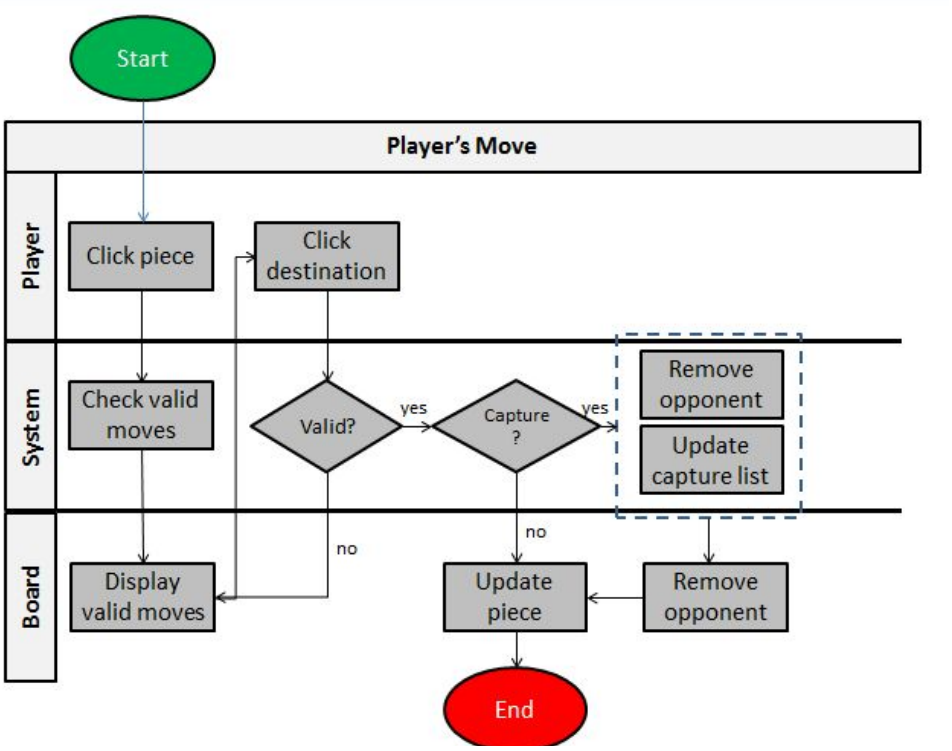
Actors:	User		
Pre-Conditions:	Board is initialized		
Post-Conditions:	User can see available moves for the selected piece		
Frequency of Use:	Every time the user clicks on a piece		
Flow of Events:		**Actor Action**	**System Response**
	1	Player selects a piece	Check if piece is legal. If so, display all possible moves for that piece.
	2	Player selects a destination	Check if destination is legal. If so, move selected piece to destination, update board and pieces list, turn over to next player.
Variations:	If either piece or destination are invalid, no move is made and it is still player's turn.		
Notes and Issues:	Requires calculating moves before destination is selected and displaying moves to user by updating board.		


Chess Game
Enter Names
Select Piece
Select Destination
Select End Condition
Player 1
Player 2

Sub-Diagram


Select Piece
Select Destination
Check Legal Piece
Check Legal Destination

Activity Diagram:



Data Storage:

Data will persist through the use of a text file that will contain the number of wins, losses, and ties for each player. Two classes will access this data at runtime, one class to read the data when the game is starting to get previous player info, and another class to write updated data once the game is finished. Data will be store in a subfolder of the directory that the game is in.

UI Mockups:

Upon launching Chess Xpress, an image similar to Figure 1 below will appear. This GUI prompts each user (on the same system) to enter their names which will be stored and used by the system to create a new game. This screen gives users the option of starting a new game (Depart) or quitting (Leave the station) which will kill the system.

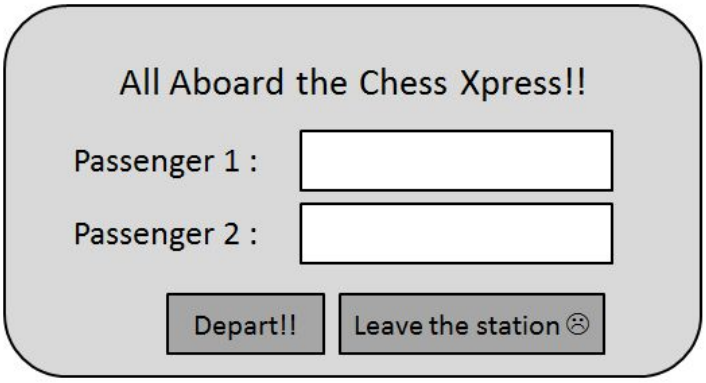


Figure 1

After the users enter names and press “Depart” on the previous screen, a new chess game will be generated, similar to Figure 2 shown below. Each player’s name will appear next to their respective color pieces. From this screen, users may begin playing by clicking on pieces they wish to interact with and clicking the corresponding destination location. Users may also return to the main menu to change names and start a new match, or quit which will kill the system.

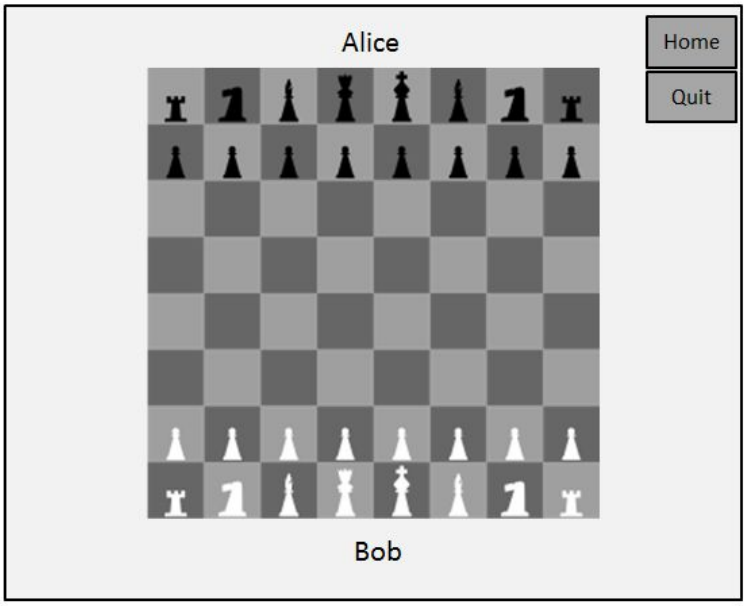


Figure 2

When a game concludes, a window similar to the one shown below in Figure 3 will appear. This window allows the users to click “Rematch” which will generate a new game using the same names as before, “New Game” which brings them back to the main screen (Figure 1), or “Quit” which will kill the system.

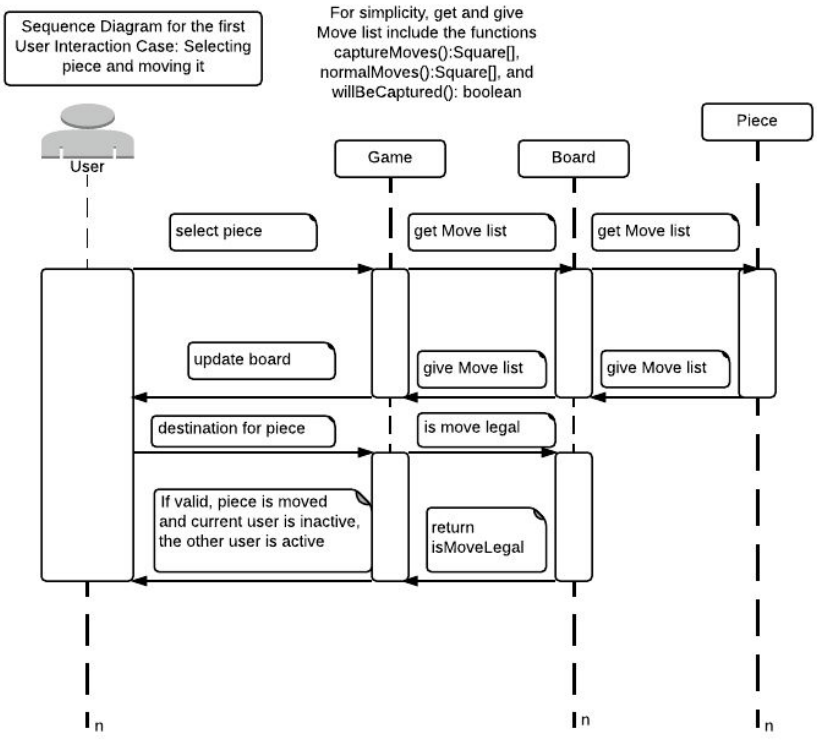
Figure 3

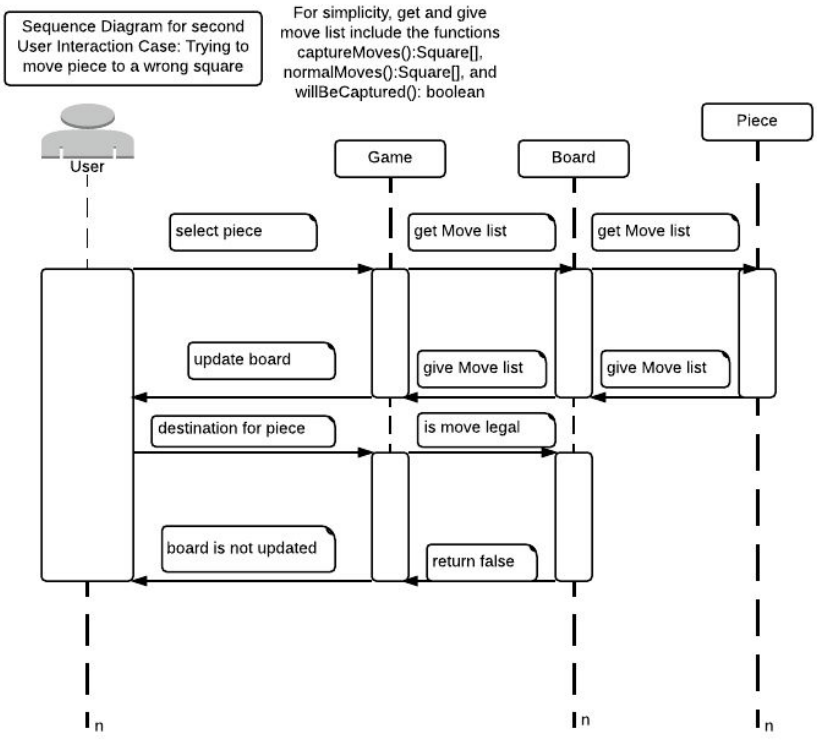
User Interactions:

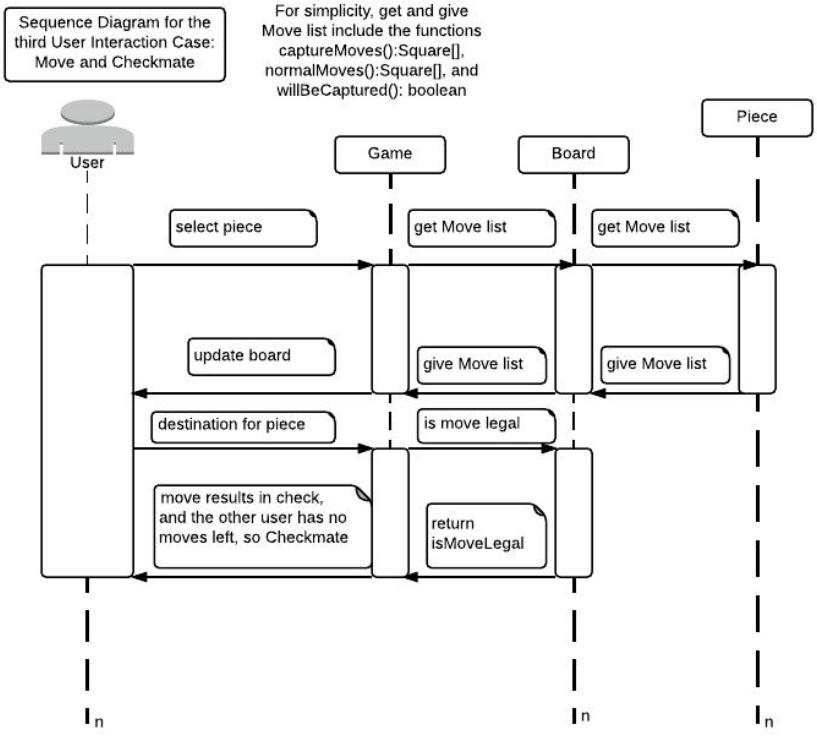
1) The first primary user experience of this project will be selecting a chess piece and moving it to an appropriate square on the chessboard. When the user selects a piece, for example a pawn, they will have the ability to choose any square that has been specifically calculated for that piece from the functions in the Move class. When the user has decided on a square, the board will be updated with the chosen chess piece in it's new position. The user will go inactive meaning that they can't move any other pieces, until the second player has gone.

2) The second user experience which may occur frequently is when the user tries to move their chess piece to an invalid square. The system, in our case the game, will have already calculated which squares the piece can go to from the Move class, so it will recognize that if the particular square doesn't match any of the options, then it will not allow the player to move there. Instead, it will return a false signal and the board will not update until the player makes a correct move with either the same or new chess piece.

3) The third user experience, which will occur towards the end of the game, is moving and achieving a checkmate. The user will select any of their remaining pieces and move the piece to a square that has been verified by the program, by using the functions from the Move class. The board will be updated, and if the other user doesn't have any pieces that can be moved, then the system will call the isCheckmate() function, and this will lead to an end of the game.


Sequence Diagram for the first User Interaction Case: Selecting piece and moving it
For simplicity, get and give Move list include the functions captureMoves():Square[], normalMoves():Square[], and willBeCaptured(): boolean
User
Game
Board
Piece
select piece
get Move list
get Move list
update board
give Move list
give Move list
destination for piece
is move legal
If valid, piece is moved and current user is inactive, the other user is active
return isMoveLegal
n
n
n


Sequence Diagram for second User Interaction Case: Trying to move piece to a wrong square
For simplicity, get and give move list include the functions captureMoves():Square[], normalMoves():Square[], and willBeCaptured(): boolean
User
Game
Board
Piece
select piece
get Move list
get Move list
update board
give Move list
give Move list
destination for piece
is move legal
board is not updated
return false
n
n
n


Sequence Diagram for the third User Interaction Case: Move and Checkmate
For simplicity, get and give Move list include the functions captureMoves():Square[], normalMoves():Square[], and willBeCaptured(): boolean
User
Game
Board
Piece
select piece
get Move list
get Move list
update board
give Move list
give Move list
destination for piece
is move legal
move results in check, and the other user has no moves left, so Checkmate
return isMoveLegal
n
n
n

Class Diagram: