

Documentation of Project Quantum Chess

by

Pongsatorn Kerdphol 5931040421
Phirasit Charoenchitseriwong 5931043321

This documentation is part of the final project of the course “Programming Methodology” (2110215) along with the actual game. In this document, the main discussions are about 3 things: 1) the brief manual of how to play this game, 2) how to use/play this game properly and 3) the description of the structure and all the methods inside the game’s code.

The whole game and this document can be found at:
<https://github.com/aqover/Quantum-Chess>

Quantum Chess

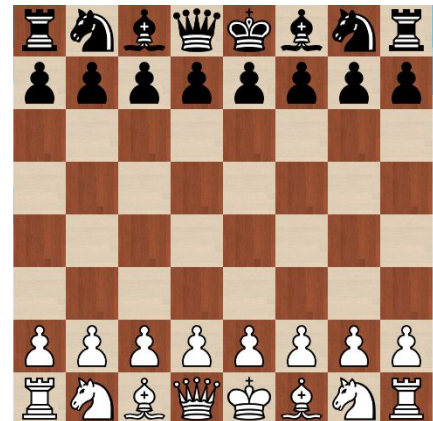
Chess is a game of strategy. Everyone is quite familiar with it since it has been played since the ancient time. In this game, a board of 8x8 squares with alternating color is used. Each player will have a equal number of troops. Each troop consists of 8 pawns, 2 rooks, 2 knights, 2 bishops, 1 queen and 1 king. Each chess piece has different ways to move.

Valid Moves

- A pawn can only move forward but can only attack diagonally.
- A rook can move to any cell within the same row or column.
- A knight moves like a shape of 'L'.
- A bishop can move to any cell that aligns in the same diagonal direction.
- A queen is a combination of a rook and a bishop.
- And, finally, a king can only move to any cell that adjacents to it. Despite being the weakness, king is the piece that determines the winner of the game.

General Rules

- No piece except knights can skip over any other piece that lies on the moving path.
- King can not commit a move that endangers itself. In such case that none of the moves exists the game ends with a draw. It is called a “stalemate.”
- Rules about “castling” and “en passant” are omitted due to the complication of Quantum Chess described later in this documentation.
- When a pawn reaches the end of the board, it can be upgraded into one of the following pieces:
 - bishop
 - knight
 - rook
 - queen
- To win the game is to create a checkmate. It is the case that player can threaten the opponent’s king without counter move that can prevent the king from being eaten.



Quantum Chess is a variation of a traditional chess originated by a mockumentary between Stephen Hawking and Paul Rudd (https://www.youtube.com/watch?v=Hi0BzqV_b44). The main idea of this game is to embed the power of quantum mechanics into a chess game. So, to make things nondeterministic, a move with possibility is used. Each move, as opposed to the traditional one, will have a chance to fail selected by the player. And, since nobody is certain about the outcome, the whole result will not be determined (yet) but will exist as a state of all possibility called "superposition." Being in a superposition state means that a piece can exist in one of the all possible cells. However, nobody knows that until two pieces try to occupy a same space. In this case, a "measurement" is used. A measurement is to look up what actually happen in that cell (and that cell only). So, everyone will know what happened in that cell. And, like a state of quantum information, the whole system will get affected from that measurement and turned into a new superposition state.

Game Rules

- Game is played similarly to normal chess game. However, each person can be able to pass (not to move) in each turn.
- To win the game is to capture the king (not to checkmate) since the actual position of king is unknown.
- Rules of en passant, castling are all omitted due to the simplicity of this game. Since, those rules involve moving more than one piece at the same time.
- The outcome of each measurement is what happened the moment a piece trying to step over another piece. Therefore, moving that same piece to the same one will not cause a measurement to happen.
- A measurement will determine whether a capture is occurred. If that is the case, all the possible places of the captured piece will collapse and disappear.
- When a pawn reaches the end of the board, a measurement will occur at that cell to confirm the upgrade.

The way to compute the possibility is to simulate all the possibilities of every move. So, when the move occurs, two new states are created from each of the current state. This might lead to an exponentially increasing number of states. But, since the measurement will happen when two pieces occupy a same cell, it will maintain that number to not exceed the limitation of the computer.

This game can help explain the basic concept of quantum mechanics to people who interest in it. It also serve as a variation of chess creating new techniques and strategies to other people that like challenges.

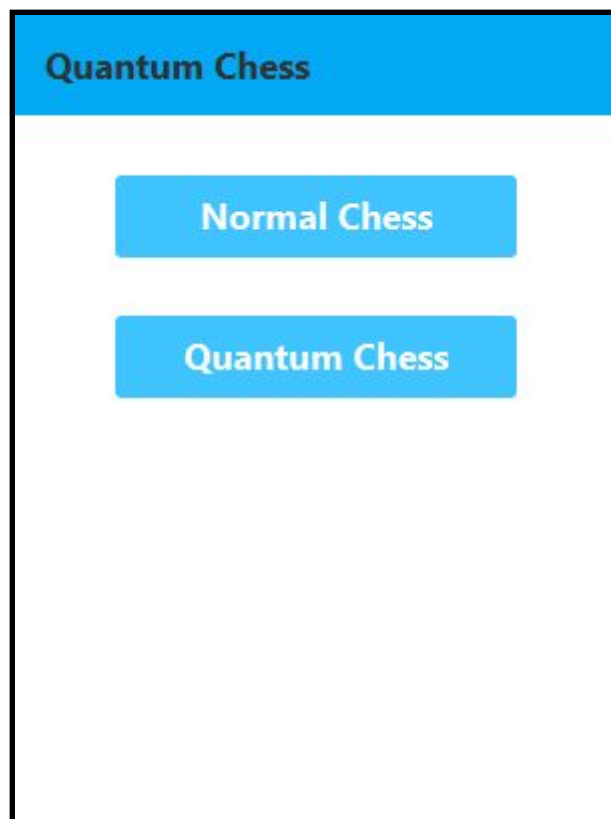
How To Play

This program is quite simple to use. There are 4 main features in this game:

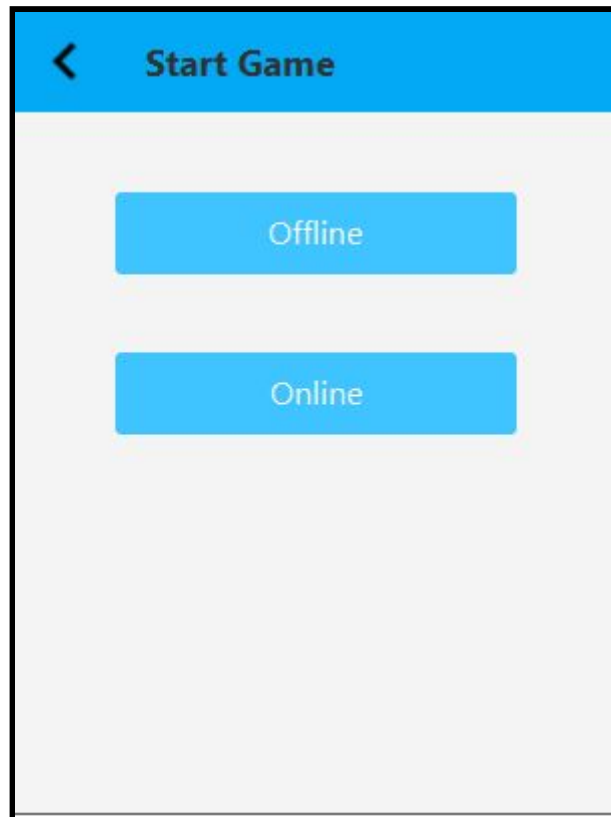
1. Normal Chess Game: Offline Mode
2. Normal Chess Game: Online Mode
3. Quantum Chess Game: Offline Mode
4. Quantum Chess Game: Online Mode

The following is the description of how to use the program.

First page: Main Menu



In this page, there are two buttons. Players can select which one they want to play. It contains all the information in the game.



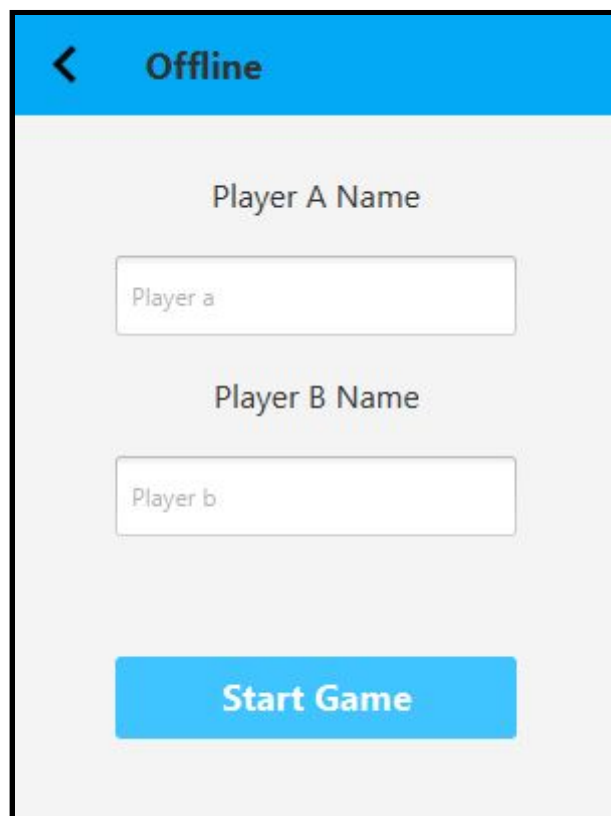
A mobile app screen titled "Start Game" with a blue header bar containing a back arrow. The main area is light gray and contains two blue buttons: "Offline" and "Online".

Start Game

Offline

Online

The next page, is about which mode the player want to play (offline or online).



A mobile app screen titled "Offline" with a blue header bar containing a back arrow. The main area is light gray and contains two text input fields for "Player A Name" and "Player B Name", each with a placeholder text "Player a" and "Player b" respectively. At the bottom is a blue button labeled "Start Game".

Offline

Player A Name

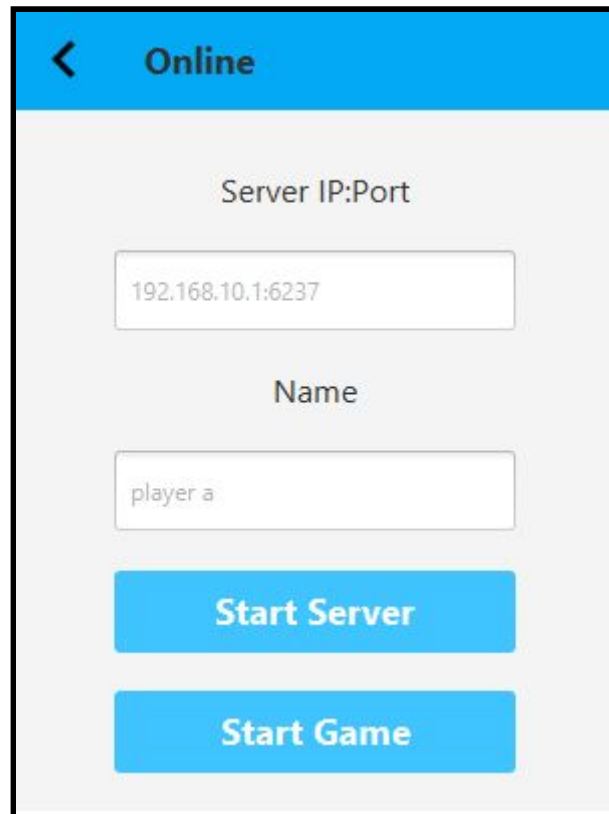
Player a

Player B Name

Player b

Start Game

If the user chooses offline mode the next page will be this page. The user can add the name of two players and then click to start the game.



Online

Server IP:Port

192.168.10.1:6237

Name

player a

Start Server

Start Game

However, If the user chooses online mode. They have to put in hostname and port of the server they want to run. They also have to wait for someone else to join server before start, or they can join someone else's server.

After someone connect the server, the game will redirect to that game online mode

Normal Chess Game: Offline Mode



In normal chess game offline mode, the board will look like the figure above. Each person will have a timer denotes the total play time. There is also a radio box showing who is playing.

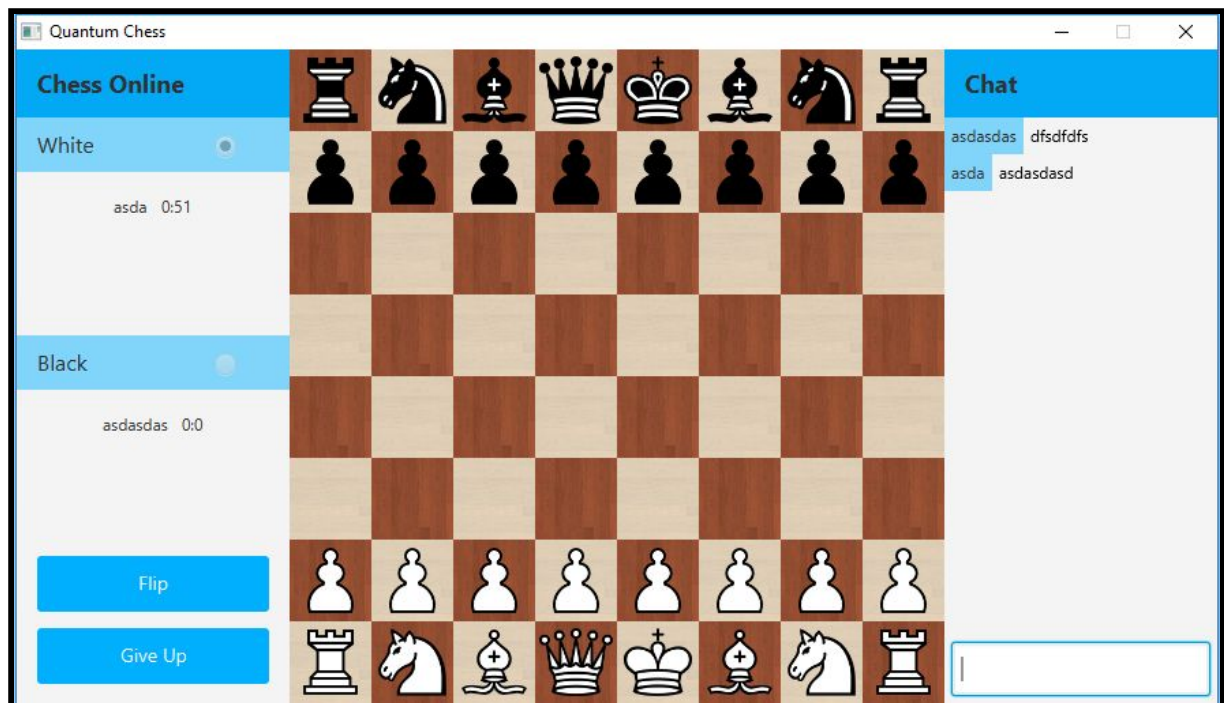
There are also 4 buttons that can be played.

- Undo button: undo the last move. if there is no last move, do nothing.
- flip button: flip the board.
- Redo button: redo the last undo. if there is no last undo, do nothing.
- Quit: finish playing go back to main menu.

The board consists of 2 type of pieces: black and white ones. To move each piece the user has to click the piece they want. There will be options display to the user to inform which cell this piece can move to. And, after the user click on the valid cell, the piece will move to that cell. The board will got flipped. The turn pass on to the next person.

If the game ends after some moves, the dialog will show up saying the result. And the scene got set back to main menu.

Normal Chess Game: Online Mode

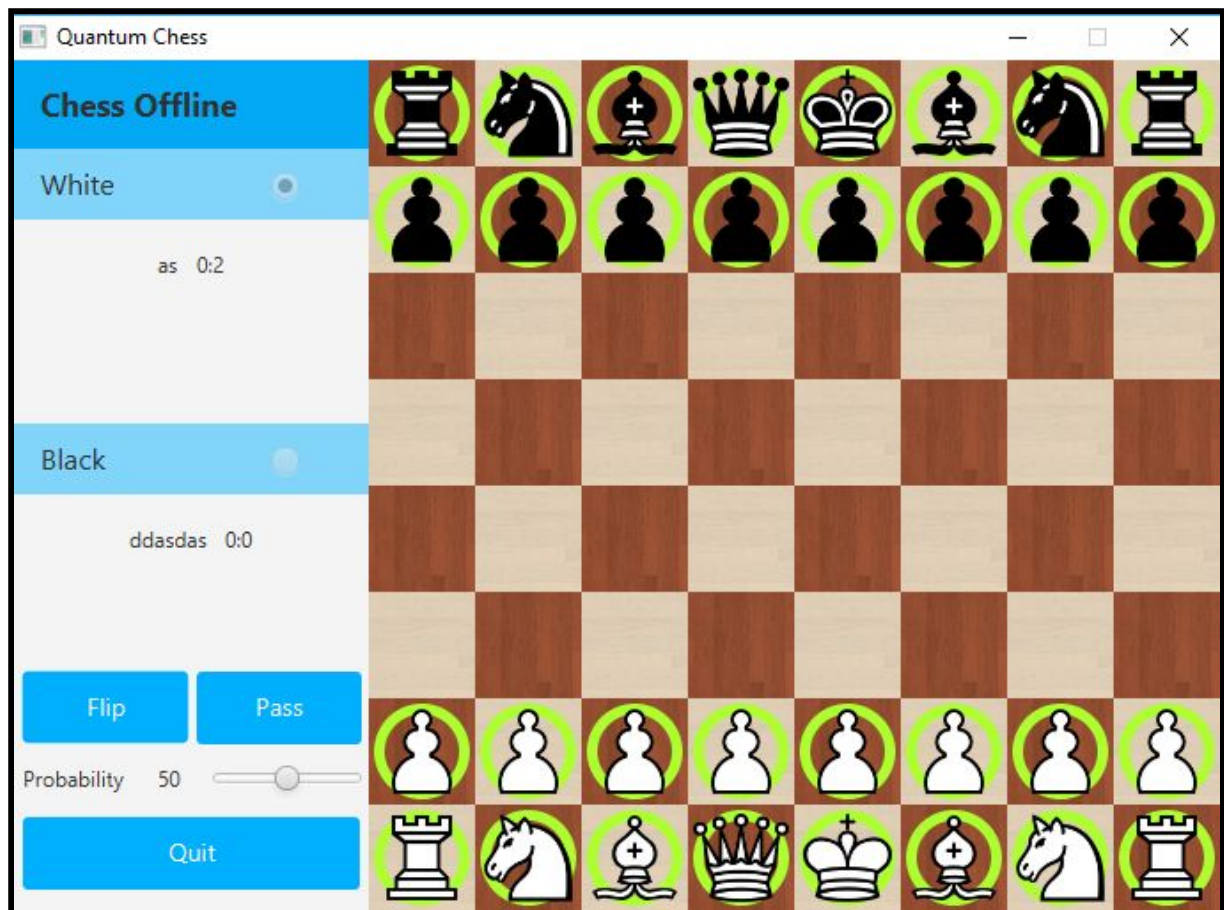


In online mode, most of how to play the game is pretty like the offline mode. However, each side of the user can only control one side of the piece. There is a new feature which is a chat mode. Players can talk to each other through this feature.

There is also no undo and redo button because those move can not be reversed when playing with stranger online although the board can be flipped normally.

The give up button will also ask for confirmation before leaving the game, the server will send surrender message to the opponent and close itself seconds later.

Quantum Chess Game: Offline Mode



In quantum game, however, moves are a little more complicated. In the figure, there will be probability bar inside each piece. It denotes the probability of each piece being in that cell.

Like the normal one, the buttons on the left side are like the previous ones with the addition of a pass button. It means that there is no move in this turn and switch to the opponent immediately. There is also a slider inside to change the success of the move. Changing this value will result in a different outcome of the following moves.



When a piece is move the result will be like this figure. This is the result of two moves by white and black each which results in two pawns simultaneously exist in different place. However, if one of the piece tries to move to a cell that already be occupied. The result board is a result of a measurement of that event.

Quantum Chess Game: Online Mode



Online mode is literally a combination between a quantum chess and an online mode from normal chess. All the description of this mode has been described in the previous pages.

Class Description

2.1 Package application

2.1.1 class Main

2.1.1.1 Method

+ void start(Stage)	add listener and set stage
+ void main(String[])	launch application

2.2 Package controller

2.2.1 class BoardGameOnlineController

2.2.1.1 Field

- Team PLAYER_TURN	keeps the player who will do the first moves
# Chat chat	a chat object uses for chatting to each other
# ChessOnlineDetail detail	a detail object which keeps all the buttons on the left side of the screen
- TCPsocket socket	a socket uses for online communication

2.2.1.2 Method

+ Chat getChat()	chat getter
+ ChessOnlineDetail getOnlineDetail()	detail getter
+ BoardGameOnlineController(String, TCPsocket)	Constructor. Create the game with the string and connect it with the given socket
+ boolean isCurrentTurn()	return whether the local player is the current player of the game
# void initialPane()	create all the required panes
+ void endGame()	surrender. Send message to the opponent and then leave the game
# void surrender()	Opponent surrenders. Show a message and then leave the game
+ void endTurn()	function that is called after the turn ended. Check the status of the game
+ void update()	pane update function. Rerender all the objects
# void checkUpgradePawn(Runnable)	check for upgradeable pawn and execute the given program latter
+ boolean movePiece(ChessPiece, Tuple<Integer, Integer>)	move the selected piece to the selected target
+ void OnReceived(TCPCommand, String)	when the opponent send message to us. Decipher and call the requested function
+ void OnSended(String)	function executed after a message is send
+ void OnClosed()	function executed when the connection is closed
+ void OnConnected()	function executed when a connection is established

2.2.2 class ChessController

2.2.2.1 Field

# HBox pane	the detail of the board
# ChessBoard board	current chess board
# ChessDetail detail	chess board control panel: control all the buttons, display and time counter
# NormalChessGame normalChessGame	model to control the current state of the game
# AnimationTimer animationTimer	timer uses for animation render
# Tuple<Integer, Integer> mouse	position of the mouse
# long timePrevious	time counter uses with animationTimer
# ChessPiece selectedPiece	the piece that is being selected by mouse
# ChessPiece lastMovedPiece	the last piece that was moved
+ boolean disable	use for diable the whole board

2.2.2.2 Method

+ HBox getPane()	pane getter
+ ChessBoard getBoard()	board getter
+ NormalChessGame getNormalChessGame()	normalChessGame getter
+ ChessDetail getDetail()	detail getter
+ ChessController()	Constructor. reset the game and init the pane
- void resetGame()	set all values to the default ones
+ void startGame()	start animationTimer with a reset value of timePrevious
+ void endTurn()	check the game state and flip the board if the next person's side is incorrect
+ void pass()	nothing
- void changeLastMovedPiece(ChessPiece)	lastMovedPiece setter
+ void update()	update the game, polling value from MouseUtility, check moving and update the board
+ void endGame()	resetGame, stop animationTimer and get back to main menu scene
# void checkEndGame()	check whether the game is finished. If it does, show who's the winner and end the game
# void upgradePawn(ChessPiece)	upgrade the pawn with the selected piece
# void checkUpgradePawn(Runnable)	check is there an ungradeable pawn and upgrade it. Also execute the command after the method is finished
+ void flipBoard()	flip the board
+ void undo()	undo the last move. If there is no last move, do nothing
+ void redo()	redo the next move. If there is no next move, do

	nothing. (After the new move, all history after it will be erased)
+ Team getTurn()	return who supports to play this turn
# void select(ChessPiece)	select the piece, call after the mouse is pressed
+ boolean movePiece(ChessPiece, Tuple<Integer, Integer>)	move the piece from the selected piece to the desired place
# void initialPane()	init all panes construct them like in the figure

2.2.3 class GFController

2.2.3.1 Field

# TextField playerA	name of player A (white)
# TextField playerB	name of player B (black)
# ChessController chessControl	chessControl model

2.2.3.2 Method

+ GFController()	Constructor. Init chessControl
+ void clear()	clear both player's names
+ void handlerBack(MouseEvent)	set handler for back button (go to main menu)
+ void startGame(MouseEvent)	load the game when the start button is pressed. Reject when atleast one of the names is missing
+ void showAlert(AlertType, String)	show the message
# void createGameController()	create the normalChessGameControll and set the scene with it

2.2.4 class GOController

2.2.4.1 Field

<u>- GOController INSTANCE</u>	singleton to the whole class
# TextField ip	the ip of the server
# TextField name	name of the player
- Pane modal	message panel
- Pane main	main pane
- Label time	connection timeout timer
# long TIMEOUT	30 sec: connection time limit
# TCPSocket socket	connection socket
# AcceptClient waiting	connection waiting thread
<u>- BoardGameOnlineController chessControl</u>	game controller
<u>- String MY_GAME_TYPE</u>	"NORMAL_CHESS": game type
# String nameWhite	name of white player

# String nameBlack	name of black player
# String gameType	type of the game

2.2.4.2 Method

+ GOController()	Constructor. load game online fxml
+ void clear()	reset ip and name to empty string. reset every pointer to null
+ void handlerBack(MouseEvent)	handle back button being pressed. move back to main menu
+ void startGame(MouseEvent)	join the server. start the game. Create the whole chess board
+ void startServer(MouseEvent)	start the server. wait until someone joins the game
+ void stopWaiting(MouseEvent)	destroy the server thread after no one connects to the server within time limit
+ void showAlert(AlertType, String)	show error dialog message
- void setShowModal(boolean)	show the waiting message before connect to the server
- void linkReady(Thread)	Listener. create the game when someone joins the server
# void createTCPClient(String, String, TCPListener)	create TCP client with the given host and port
# void createTCPServer(String, TCPListener)	create TCP server with the given host and port
# void createGameController()	create the chess game with a connection socket
+ void OnReceived(TCPCommand, String)	execute the received commands
+ void OnSended(String)	when command is send. (do no thing)
+ void OnClosed()	when connection is closed. (donothing)
+ void OnConnected()	when connection is established. (do nothing)

2.2.5 class AcceptClient

2.2.5.1 Field

- long startTimeout	store the start time. use to calculate the total time
- boolean isSuccess	Is someone connected to this thread
- Runnable onDone	execute commands after thread is finished

2.2.5.2 Method

+ boolean isSuccess()	isSuccess getter
+ AcceptClient()	Constructor. construct the whole thread by assign all the values with the given ones
+ void addListener(Runnable)	onDone setter

+ void run()	run the thread wait until someone connects the thread.
+ void destroy()	stop the thread by changing the value of startTimeout

2.2.6 class GQFController

2.2.6.1 Method

GQFController()	Constructor. call super().
createGameController()	create quantum game instead of normal chess game

2.2.7 class GQOController

2.2.7.1 Field

- <u>GQOController INSTANCE</u>	singleton object
- <u>QuantumChessOnlineController chessControl</u>	quantum chess control model
- <u>String MY_GAME_TYPE</u>	"QUANTUM_CHESS": type of the game

2.2.7.2 Method

+ GQOController()	Constructor. call super()
+ void startGame(MouseEvent)	join the server and start the game.
+ void startServer(MouseEvent)	start server. wait until someone to join the server and then start the game
# void createGameController()	create the quantum chess game
+ void OnReceived(TCPCommand, String)	execute the received commands

2.2.8 class MainMenuController

2.2.8.1 Field

- TextField playerA	name of player A
- TextField playerB	name of player B
- Button normalChess	choose normalChess
- Button quantumChess	choose quantumChess

2.2.8.2 Method

+ MainMenuController()	Constructor. load MainMenu.fxml
+ void handlerSelectNormal(MouseEvent)	select normal chess game. set type of game and load the next page
+ void handlerSelectQuantum(MouseEvent)	select quantum chess game. set type of game and load the next page
+ void handlerAbout(MouseEvent)	show the about page
+ void showAlert(AlertType, String)	show error string

2.2.9 class QuantumChessController

2.2.9.1 Field

# QuantumChessGame quantumChessGame	quantum chess game model
# QuantumChessDetail detail	detail pane (buttons and selections)
# double moveProb	probability for a move to success
# double[][] possibility	possibility of a move to any other cell

2.2.9.2 Method

+ getQuantumChessGame()	quantumChessGame getter
+ getQuantumDetail()	detail setter
+ QuantumChessController()	Constructor. reset game and init Pane
+ void setMoveProb(double)	moveProb setter
+ double getMoveProb()	moveProb getter
- void resetGame()	initialize all variables
+ void endTurn()	check end game and flip the board if necessary
+ void pass()	pass the turn
# void checkEndGame()	if game is ended, show the result and set scene to main menu
# void upgradePawn(ChessPiece)	upgrade pawn if it reaches the end line
# void checkUpgradePawn(Runnable)	check whether there is an upgradeable pawn and upgrade it
+ Team getTurn()	return who is playing
# void select(ChessPiece)	select the choosen piece
+ void update()	upgdade the game. render all piece again and move if commanded
+ boolean movePiece(ChessPiece, Tuple<Integer, Integer>)	move the game according to the given parameters
+ boolean movePiece(ChessPiece, Tuple<Integer, Integer>, boolean)	move the game with specific override boolean. (use for online mode only)
# void initialPane()	create all necessary panes

2.2.10 class QuantumChessOnlineController

2.2.10.1 Field

- Team PLAYER_TURN	keep turn who play first
# Chat chat	chat pane

# QuantumChessOnlineDetail detail	detail of all buttons and selections
- TCPSocket socket	socket connecting to other players

2.2.10.2 Method

+ Chat getChat()	chat getter
+ QuantumChessOnlineDetail getOnlineDetail()	detail getter
+ QuantumChessOnlineController QuantumChessOnlineController(String, TCPSocket)	constructor. call super and set value of socket
+ boolean isCurrentTurn()	return whether the local player is the player of that turn
# void initialPane()	create all necessary panes
+ void endGame()	surrender option. send command to tell another player that you give up
# void surrender()	opponent has surrendered. show victory message and set scene to main menu
+ void endTurn()	end the turn. check game result. if game is done, show result and set scene to main menu
+ void update()	update the game (mouse and all the click)
# void checkUpgradePawn(Runnable)	check there is an upgradeable pawn. If so, upgrade that pawn. Also execute the runnable command after
+ Tuple<Integer, Integer> movePiece(ChessPiece, Tuple<Integer, Integer>)	move the piece. send the move to the opponent
+ void OnReceived(TCPCommand, String)	execute the command send from opponent
+ void OnSended(String)	do nothing
+ void OnClosed()	do nothing
+ void OnConnected()	do nothing

2.2.11 final class SceneManager

2.2.11.1 Field

- <u>Stage primaryStage</u>	application stage
- <u>Scene mainMenu</u>	mainMenu scene
- <u>Scene selectGame</u>	selectGame scene
- <u>Scene gameOffline</u>	game offline scene
- <u>Scene gameOnline</u>	game online scene
- <u>Scene quantumGameOffline</u>	qauntum chess offline scene
- <u>Scene quantumGameOnline</u>	qauntum chess online scene
- <u>boolean disable</u>	protect showMessage run only one times

- <u>boolean isQuantumChess</u>	keep truth that user click quantum chess
---------------------------------	--

2.2.11.2 Method

+ <u>void initialize(Stage)</u>	set title and set stage to main menu and show
+ <u>void setSceneMainMenu()</u>	set stage to mainMenu scene
+ <u>void setSceneSelectGame(boolean)</u>	set stage to selectGame scene
+ <u>void setSceneGameOnline()</u>	set stage to gameOnline scene
+ <u>void setSceneGameOffline()</u>	set stage to gameOffline scene
+ <u>void setScene(Pane)</u>	set stage to scene that given pane
+ <u>boolean showMessage(String, onFinish)</u>	show alert box
+ <u>boolean showMessage(String, Collection<? extends ButtonType>, onFinish)</u>	show alert box with button type given

2.2.12 Interface onFinish

2.2.12.1 Method

+ <u>void run()</u>	nothing
+ <u>void run(ButtonType)</u>	nothing

2.2.13 class SelectGameController

2.2.13.1 Method

+ <u>SelectGameController()</u>	load pane
+ <u>void handlerOnline(MouseEvent)</u>	call setSceneGameOnline
+ <u>void handlerOffline(MouseEvent)</u>	call setSceneGameOffline
+ <u>void handlerBack(MouseEvent)</u>	call setSceneMainMenu

2.3 Package helper

2.3.1 class InputUtility

2.3.1.1 Field

<u>+ double mouseX</u>	the x-coordinate of the mouse
<u>+ double mouseY</u>	the y-coordinate of the mouse
<u>- boolean isMouseLeftClicked</u>	status of whether the left mouse is being clicked
<u>- boolean isMouseRightClicked</u>	status of whether the right mouse is being clicked

2.3.1.2 Method

<u>+ void mouseLeftDown()</u>	set the value of isMouseLeftClicked to true
<u>+ void mouseRightDown()</u>	set the value of isMouseRightClicked to true
<u>+ void mouseLeftRelease()</u>	set the value of isMouseLeftClicked to false
<u>+ void mouseRightRelease()</u>	set the value of isMouseRightClicked to false
<u>+ boolean isMouseLeftClicked()</u>	isMouseLeftClicked getter
<u>+ boolean isMouseRightClicked()</u>	isMouseRightClicked getter
<u>+ void update()</u>	reset isMouseLeftClicked, isMouseRightClicked
<u>+ Tuple<Integer, Integer> getMousePosition()</u>	get mouse position in the board
<u>+ Tuple<Integer, Integer> getMousePosition(double, double)</u>	convert mouse position into table cell

2.3.2 enum Team

2.3.2.1 Field

<u>+ NONE</u>	belongs to no team
<u>+ PLAYER_WHITE</u>	belongs to white
<u>+ PLAYER_BLACK</u>	belongs to black

2.3.3 class Tuple<X, Y>

2.3.3.1 Field

<u>- X I</u>	store i on variable X type
<u>- Y J</u>	store j on variable Y type

2.3.3.2 Method

<u>+ Tuple(X, Y)</u>	new tuple from x and y
<u>+ Tuple(Tuple<X, Y>)</u>	new tuple from tuple
<u>+ boolean equals(Object)</u>	check tuple are same thing
<u>+ X getI()</u>	return i
<u>+ Y getJ()</u>	return j

2.3.4 class Utility

2.3.4.1 Method

<u>+ Color rgbFade(Color, Color, double)</u>	calculate r g b value
--	-----------------------

2.4 Package library.Socket

2.4.1 class TCPClient

2.4.1.1 Field

- long <u>KEEP_ALIVE_INTERVAL</u>	time to send keep alive command
- <u>Socket socket</u>	instance of socket
- String host	keep hostname
- int port	keep port
- int totalPacket	keep number of packet that received
- int lossPacket	keep number of receive package which not complete
- String lastPacket	keep string of command that last send
- ArrayList<TCPLListener> listeners	list of listener

2.4.1.2 Method

+ void addListener(TCPLListener)	add listener to list
+ TCPClient(String, int)	set host and port
+ void connect()	connect to server
+ boolean isConnected()	check connection that connect to server
+ void run()	connect socket and receive msg
+ String read()	read string from socket
+ int write(String, boolean)	write message to socket
+ void close(boolean, boolean)	close socket and run listener
+ void destroy()	call close function
- void OnConnected()	call OnConnected for each listener
- void OnSended(String)	call OnConnected for each listener
- void OnReceived(TCPCommand, String)	call OnReceived for each listener
- void OnClosed()	call OnClose for each listener
+ int getTotalPacket()	get total packet
+ int getLossPacket()	get loss packet

2.4.2 enum TCPCommand

2.4.2.1 Field

+ <u>MOVE</u>	move command
+ <u>SET_BOARD_VERSION</u>	set board version command
+ <u>GET_BOARD_VERSION</u>	get board version command
+ <u>SET_TIME_PLAYER_WHITE</u>	set time player white command
+ <u>GET_TIME_PLAYER_WHITE</u>	get time player white command
+ <u>SET_TIME_PLAYER_BLACK</u>	set time player black command

+ <u>GET_TIME_PLAYER_BLACK</u>	get time player black command
+ <u>BLACK_DRAW</u>	player black want to draw command
+ <u>BLACK_SURRENDER</u>	player black want to surrender command
+ <u>BLACK_END_TURN</u>	player black end his turn command
+ <u>WHITE_DRAW</u>	player white want to draw command
+ <u>WHITE_SURRENDER</u>	player white want to surrender command
+ <u>WHITE_END_TURN</u>	player white end his turn command
+ <u>GAME_RESULT</u>	game result command
+ <u>NAME_PLAYER</u>	name player command
+ <u>SET_UPGRADE_PAWN</u>	set upgrade pawn command
+ <u>SEND_TEXT</u>	send chat command
+ <u>SET_GAME_TYPE</u>	set game type command
+ <u>GET_GAME_TYPE</u>	get game type command
+ <u>SET_NAME_PLAYER_WHITE</u>	set name player white command
+ <u>GET_NAME_PLAYER_WHITE</u>	get name player white command
+ <u>SET_NAME_PLAYER_BLACK</u>	set name player black command
+ <u>GET_NAME_PLAYER_BLACK</u>	get name player black command
+ <u>TCP_KEEPALIVE</u>	keep alive socket command
+ <u>TCP_FAIL</u>	loss package command
- int ID	command index

2.4.2.2 Method

- TCPCommand(int)	set index
+ String toString()	return index string in format "%02d"
+ int getValue()	get index
+ <u>TCPCommand valueOf(int)</u>	get tcpcommand from index

2.4.3 interface TCPListener

2.4.3.1 Method

+ void OnReceived(TCPCommand, String)	nothing
+ void OnSended(String)	nothing
+ void OnClosed()	nothing
+ void OnConnected()	nothing

2.4.4 class TCPServer

2.4.4.1 Field

- <u>long KEEP_ALIVE_INTERVAL</u>	time to send keep alive command
- <u>ServerSocketserver</u>	server socket
- <u>Socket client</u>	client socket
- int port	number port
- int totalPacket	number of package received
- int lossPacket	number of package loss
- String lastPacket	string of cmd which the last write
- ArrayList<TCPLListener> listeners	list of listener

2.4.4.2 Method

+ void addListener(TCPLListener)	add listener to list
+ TCPServer(int)	set port
+ boolean isConnected()	check connection that wait connection from client
+ void run()	create connection and receive msg
+ String read()	read string message from socket
+ int write(String, boolean)	write string message to socket
+ void close(boolean, boolean)	close socket
+ void destroy()	set socket and server to null
- void OnConnected()	call OnConnected for each listener
- void OnSended(String)	call OnConnected for each listener
- void OnReceived(TCPCommand, String)	call OnReceived for each listener
- void OnClosed()	call OnClose for each listener
+ int getTotalPacket()	return total package
+ int getLossPacket()	return loss package

2.4.5 interface TCPSocket

2.4.5.1 Method

+ int write(String, boolean)	nothing
+ int write(TCPCommand, String)	encode command and value then write
+ String read()	nothing
+ boolean isConnected()	nothing
+ void sendLossPacket()	write tcp loss command
+ void destroy()	nothing

2.5 Package Model

2.5.1 class ChessBoard

2.5.1.1 Field

- String[] board	stores the board represents as a 2-dimentional array of characters
- int NUMBER_OF_ROWS	number of rows in the board (a.k.a height of the board)
- int NUMBER_OF_COLUMNS	number of columns in the board (a.k.a width of the board)

2.5.1.2 Method

+ ChessBoard(String, int, int)	Constructor methods takes a string of length width * height and create a board from it
+ ChessBoard(ChessBoard)	Copy Constructor
+ void move(Move, char)	Move the piece according to the move and replace the former cell with an emptySpace
+ void moveDuplicate(Move, char)	Instead of moving, return a new board with the result of that moving
+ void fill(char)	assigns every cell with the given character
+ int getRows()	return NUMBER_OF_ROWS
+ int getColumns()	return NUMBER_OF_COLUMNS
+ char getAt(int, int)	return the piece at the given position
+ void setValue(int, int, char)	set the board cell with the given character
+ String toString(String)	encode the board back to one long string with a delimiter
+ String toString()	return toString("\n")
+ String[] getBoard()	board getter

2.5.2 static class Move

2.5.2.1 Field

- String message	description of the move
+ <u>int row1</u>	row of the cell to move from
+ <u>int col1</u>	column of the cell to move from
+ <u>int row2</u>	row of the cell to move to
+ <u>int col2</u>	column of the cell to move to

2.5.2.2 Method

+ Move(int, int, int, int)	Constructor. taking 4 numbers and fill it according to the order in the class. Also generate the message from those values.
+ Move(String)	Constructor, extracting information from the string. Use that string as a message.
+ boolean isMoving()	return true if and only if the move involves two

	distinct cells.
+ boolean isInBound(int, int, int, int)	return true if and only if all the values are in the given bound.
+ String toString()	message getter

2.5.3 interface ChessGameInfo

2.5.3.1 Field

+ int <u>GAME_RESULT_DRAW</u>	0: Status of a draw
+ int <u>GAME_RESULT_WHITE_WINS</u>	1: Status of white is the winner
+ int <u>GAME_RESULT_BLACK_WINS</u>	2: Status of black is the winner
+ int <u>GAME_RESULT_ONGOING</u>	3: Status of the game has not concluded yet.
+ int <u>BOARD_SIZE</u>	8: size of the traditional chess game's board

2.5.4 interface Piece

2.5.4.1 Field

+ char <u>WHITE_ROOK</u>	'r': the character that represents white rook
+ char <u>BLACK_ROOK</u>	'R': the character that represents black rook
+ char <u>WHITE_KNIGHT</u>	'k': the character that represents white knight
+ char <u>BLACK_KNIGHT</u>	'K': the character that represents black knight
+ char <u>WHITE_BISHOP</u>	'b': the character that represents white bishop
+ char <u>BLACK_BISHOP</u>	'B': the character that represents black bishop
+ char <u>WHITE_KING</u>	'k': the character that represents white king
+ char <u>BLACK_KING</u>	'K': the character that represents black King
+ char <u>WHITE_QUEEN</u>	'q': the character that represents white queen
+ char <u>BLACK_QUEEN</u>	'Q': the character that represents black queen
+ char <u>WHITE_PAWN</u>	'p': the character that represents white pawn
+ char <u>BLACK_PAWN</u>	'P': the character that represents black pawn
+ char <u>EMPTY_SPACE</u>	':': the character that represents empty space
+ char <u>WHITE_PAWN_LIST</u>	'12345678': a list of characters that represent white pawns
+ char <u>BLACK_PAWN_LIST</u>	'!@#\$\$%^&*': a list of characters that represent black pawns

2.5.4.2 Method

+ boolean <u>isWhitePawn(char)</u>	check whether a character is a white pawn (Is it in the WHITE_PAWN_LIST?)
+ boolean <u>isBlackPawn(char)</u>	check whether a character is a black pawn (Is it in the BLACK_PAWN_LIST?)

+ boolean isPawn(char)	check whether a character is a pawn
+ boolean isWhite(char)	check whether a character is a white piece
+ boolean isBlack(char)	check whether a character is a black piece

2.5.5 class NormalChessGame

2.5.5.1 Field

+ Team FIRST_TURN	final variable: keeps the status of the first player to move.
- int versionIndex	keeps index of the version of the game that is currently being played.
- ArrayList<ChessBoard> versions	list that keeps all the versions of the board at any moment of the game.
- ArrayList<Move> moves	list that keeps all the moves that being played by both players.

2.5.5.2 Method

+ NormalChessGame(ChessBoard)	Constructor. Construct the game. Set the start state of the game to given board. Set the starting player to white.
+ NormalChessGame(String)	Constructor. Construct the game by using ChessBoard Constructor.
+ NormalChessGame()	Constructor. Construct a default normal chess game.
+ NormalChessGame(NormalChessGame)	Copy Constructor.
+ void setVersion(int)	change version of the game to the given one.
+ void moveVersion(int)	move the version of the game by the given one.
+ boolean undo()	commit moveVersion(-1)
+ boolean redo()	commit moveVersion(1)
+ boolean isMoveValid(Move)	return whether a move is a valid chess move
+ boolean isMoveValidNoKing(ChessBoard, Move)	return whether the move is valid in the given board without checking "king checking".
+ boolean isMoveValid(ChessBoard, Move)	return whether the move is valid in the given board
+ boolean isUpgradePawnAvailable()	return whether a pawn reaches the other end of the board
# boolean setPosition(int, int, char)	set the given cell with the given character
+ void upgradePawn(char, char)	upgrade pawn when it reaches the end with the given (black/white) char.
+ boolean move(Move)	move the game with the given move
+ int getGameResult()	return the status of the game (GAME_RESULT_DRAW,

	GAME_RESULT_WHITE_WINS, GAME_RESULT_BLACK_WINS, GAME_RESULT_ONGOING)
+ String getResultMessage(int)	return the message from the given game status.
+ Team getSide(char)	return the side of the given char
+ List<ChessBoard.Move> getPossibleMoves(Team)	return all possible moves that can be used this turn
+ boolean[][] getValidMoves(int, int)	return all possible moves that starts from the given cell
+ ChessBoard getBoard(int)	return the version of the game according to the given number
+ ChessBoard getBoard()	return the current board
+ ChessBoard.Move getMove(int)	return the move that creates the version of the game according to the given number
+ ChessBoard.Move getMove()	return the last move
+ Team getTurn()	return who is playing
+ String toString(String)	return the current board with the given delimiter
+ String toString()	return the current board with delimiter = '\n'

2.5.6 class QuantumChessGame

2.5.6.1 Field

+ Team FIRST_TURN	final variable: keeps the status of the first player to move
+ String FIN	final variable: keeps the starting board
- List<QuantumBoard> possibleBoards	list of all possibility dictates by all the moves and measurements
+ List<Character> deadPieces	list of the pieces that are confirmed dead
# ChessBoard currentGame	a secret chess board that keeps the actual (non quantum) state of the game
# ChessBoard displayBoard	a display board that shows which cells have a chance to have one of the pieces
# double[][] piecePossibility	a array of possibility of a piece ending there
# boolean lastMoveHiddenStatus	a status of whether the last move was sucessful
- List<QuantumMove> moves	a list of all moves that have been played by both players

2.5.6.2 Method

+ QuantumChessGame(String)	Constructor. Create a game with the board from the given string
- void resetGame()	reset the game to the start

+ QuantumChessGame()	Constructor. Create a game with the default board (normal chess game board)
+ double successProb(Move)	return the possibility of the given move being successful
+ double[][] getPossibilityMoves(int, int)	return the array of possibility of the move from the given cell to any other cell
# void setDisplayBoard()	create the display board and keep it at displayBoard variable
+ void measure(ArrayList<Integer>, ArrayList<Integer>)	measure all the cells given at the same time. eliminate all the improper states from possibleBoards
# char checkBoard(int, int)	return the char result from a measurement at the given cell
# void setPosition(int, int, char)	set the given cell with the given character in every possible states
+ void upgradePawn(char, char)	upgrade pawn with the given (black/white) char
+ boolean isUpgradePawnAvailable()	return whether there is an upgradeable pawn in the board
+ void move(QuantumMove, boolean)	move the game with the given move with a force command to the move the hidden board
- boolean randomSuccess(double)	return true with the probability of the given value
+ void move(QuantumMove)	move the game with the given move, and move the hidden board with the given probability
+ void pass()	skip a turn
+ boolean lastMoveStatus()	lastMoveHiddenStatus getter
+ boolean isDead(char)	return whether the given character is already captured
+ Team getTurn()	return who is playing
+ ChessBoard getDisplayBoard()	displayBoard getter
+ double[][] getPiecePossibility()	piecePossibility getter
+ int getGameResult()	return the result message from the given status
+ <u>String getResultMessage(int)</u>	return the status of the game (GAME_RESULT_DRAW, GAME_RESULT_WHITE_WINS, GAME_RESULT_BLACK_WINS, GAME_RESULT_ONGOING)

2.5.7 static class QuantumBoard

2.5.7.1 Field

- double prob	probability of the board is the actual board
+ ChessBoard BOARD	detail of the board

2.5.7.2 Method

+ QuantumBoard(double, ChessBoard)	Constructor. Assign all variables with the given ones
+ double getProb()	prob getter
+ void setProb(double)	prob setter

2.5.8 static class QuantumMove

2.5.8.1 Field

+ double PROB	probability of the move being successful
+ ChessBoard.Move MOVE	detail of the move

2.5.8.2 Method

+ QuantumMove(double, Move)	Constructor. Assign all variables with the given ones
+ QuantumMove(String)	Constructor. create a move from extracting information from the string
+ String encode()	return the move encoded as a string (use for online communication)
+ String toString()	return encode()

2.6 Package Model.piece

2.6.1 class Bishop

2.6.1.1 Field

- <i>Bishop</i> instance	instance of this class
--------------------------	------------------------

2.6.1.2 Method

+ Bishop getInstance()	return instance
+ Bishop(Integer, Integer, Team)	inistial row ,col and piece picture
+ char getWhitePiece()	return character of white bishop
+ char getBlackPiece()	return character of black bishop
+ char getChar()	return 'b'
+ boolean isValidMove(ChessBoard, Move)	check move is valid

2.6.2 abstract class ChessPiece

2.6.2.1 Field

# Image DISPLAY_IMAGE	instance of image
# Team TEAM	team side
# int row	position x on board
# int col	position y on board
# boolean isdead	piece is dead
# double prob	probability
# boolean isSelected	piece was select
# boolean isLastMoved	this piece is the last move

2.6.2.2 Method

+ <i>boolean</i> isValidMove(ChessBoard, Move)	nothing
+ <u>ChessPiece</u> getInstance(char)	return instance of parameter char
+ <i>char</i> getWhitePiece()	nothing
+ <i>char</i> getBlackPiece()	nothing
+ void draw(GraphicsContext)	draw piece, prob, last move and hover
+ void drawLastMoved(GraphicsContext)	fill rect
+ void drawHover(GraphicsContext)	fill rect
+ Team getTeam()	return team
+ int getRow()	return row
+ int getI()	return row
+ int getColumn()	return col

+ int getJ()	return col
+ boolean isDead()	return isDead
+ boolean isSelected()	return isSelected
+ void setSelected(boolean)	set isSelected
+ boolean isLastMoved()	return isLastMove
+ void setLastMoved(boolean)	set isLastMove
+ void setOnlyPosition(int, int)	set row and col
+ void setPositionOnScreen(int, int)	set position on screen
+ void setPosition(int, int)	set position value and position on screen
+ void setPossibility(double)	set prob
+ ChessPiece(int, int, Team, Image)	initial value and set position

2.6.3 static class NoPieceException

2.6.3.1 Field

- <u>long serialVersionUID</u>	set value to 2122415123622064555L
- <u>String msg</u>	msg exception

2.6.3.2 Method

+ String getMessage()	return msg
+ NoPieceException(char)	set msg is "This chess piece is unknown : " + parameter

2.6.4 class King

2.6.4.1 Field

- King instance	instance of this class
-----------------	------------------------

2.6.4.2 Method

+ King getInstance()	return instance
+ King(Integer, Integer, Team)	initial row ,col and piece picture
+ char getWhitePiece()	return character of white king
+ char getBlackPiece()	return character of black king
+ boolean isValidMove(ChessBoard, Move)	check move is valid
+ boolean isKingThreaten(ChessBoard, char)	check king is threaten

2.6.5 class Knight

2.6.5.1 Field

- Knight instance	instance of this class
-------------------	------------------------

2.6.5.2 Method

+ Knight getInstance()	return instance
+ char getWhitePiece()	return character of white knight
+ char getBlackPiece()	return character of black knight
+ char getChar()	return 'n'
+ Knight(Integer, Integer, Team)	inistial row ,col and piece picture
+ boolean isValidMove(ChessBoard, Move)	check move is valid

2.6.6 class Pawn

2.6.6.1 Field

- Pawn instance	instance of this class
-----------------	------------------------

2.6.6.2 Method

+ Pawn getInstance()	return instance
+ Pawn(Integer, Integer, Team)	inistial row ,col and piece picture
+ char getWhitePiece()	return character of white pawn
+ char getBlackPiece()	return character of black pawn
+ boolean isValidMove(ChessBoard, Move)	check move is valid

2.6.6 class Queen

2.6.6.1 Field

- Queen instance	instance of this class
------------------	------------------------

2.6.6.2 Method

+ Queen getInstance()	return instance
+ Queen(Integer, Integer, Team)	inistial row ,col and piece picture
+ char getWhitePiece()	return character of white queen
+ char getBlackPiece()	return character of black queen
+ char getChar()	return 'q'
+ boolean isValidMove(ChessBoard, Move)	check move is valid

2.6.6 class Rook

2.6.6.1 Field

- Rookinstance	instance of this class
----------------	------------------------

2.6.6.2 Method

+ Rook getInstance()	return instance
+ Rook(Integer, Integer, Team)	inistial row ,col and piece picture
+ char getWhitePiece()	return character of white rook
+ char getBlackPiece()	return character of black rook
+ char getChar()	return 'r'
+ boolean isValidMove(ChessBoard, Move)	check move is valid

2.7 Package scene.gameBoard

2.7.1 class Chat

2.7.1.1 Field

- int <u>CHAT_LIMIT</u>	Number of chat that show on interface
# List<ChatField> chatFieldData	List of chat field which show on interface
# String username	player name
# TCP Socket socket	chat socket, get from main socket
- Button clear	clear button
- Button send	send button
- TextField inputMessage	message text field
- VBox messageField	box of all message which show on interface

2.7.1.2 Method

+ Chat()	load pane and set event handler to inputMessage
+ void insert(ChatField)	insert chat field to list
+ void update()	show chat field that not to exceed chat_limit
+ String getUsername()	return username
+ void setUsername(String)	set username
+ TCP Socket getSocket()	return socket
+ void setSocket(TCP Socket)	set socket
+ void requestFocus()	request focus to input ,essage
# void handleClear(MouseEvent)	clear message input
# void handleSend(MouseEvent)	call sendMessage
- void sendMessage()	valid message create new chat field insert chat field send message to socket clear message input

2.7.2 class ChatField

2.7.2.1 Field

# String USER	player name
# String MESSAGE	message
# long TIME	send time
# Label text	instance of text label which contain player name
# TextFlow textflow	instance of text flow which contain message
# Pane flowpane	instance of flow pane which contain label and textflow

2.7.2.2 Method

+ ChatField(String)	split name, message and time. create new pane
+ ChatField(String, String, long)	set user, message and time create new pane
# void createPane()	create pane follow by picture
+ String encode()	encode user message and time to string
+ String toString()	return encode
+ String getUser()	return user
+ String getMessage()	return message
+ long getTime()	return time
+ Pane getPane()	return flowpane
+ long compare(ChatField, ChatField)	compare time

2.7.3 class ChessBoard

2.7.3.1 Field

- boolean flipBoard	board was flipped
---------------------	-------------------

2.7.3.2 Method

+ ChessBoard()	add event handler set width and height set flipboard to false
+ void flipBoard()	change position all of entity from top to bottom and bottom to top.
+ boolean isBoardFlipped()	return flipboard
# void addListenEvents()	set position on input utility
+ void paintComponent()	draw background and entity
+ void paintValidMoves(boolean[][])	draw entity is valid
+ void paintPossibilityMoves(double[][], double)	draw possibility move
+ void updatePawn(Constructor<? extends ChessPiece>)	update pawn
+ void setBoard(NormalChessGame)	set piece on board
+ void setBoard(QuantumChessGame)	set piece on board

2.7.4 class ChessDetail

2.7.4.1 Field

- long nanoSecond	1 second on nano second
# long timePlayerA	player a time
# long timePlayerB	player b time

# ChessControllergame Control	chess control
- Label labelNameA	instance of player a name
- Label labelTimeA	instance of player a time
- Label labelNameB	instance of player b name
- Label labelTimeB	instance of player b time
- RadioButton radioA	instance of player a radio
- RadioButton radioB	instance of player b radio

2.7.4.2 Method

+ long getTimePlayerW()	return time player a
+ long getTimePlayerB()	return time player b
+ void setTimePlayerW(long)	set time player a
+ void setTimePlayerB(long)	set time player b
+ String getNameBlack()	return player a name
+ void setName(String, String)	set player a name
+ void increseTime(long)	increase time play
# void handleFlipBoard(MouseEvent)	call flipboard in game control
# void handleUndo(MouseEvent)	call undo in game control
# void handleRedo(MouseEvent)	call redo in game control
# void handleQuit(MouseEvent)	show message for comfirm to quit game
+ ChessDetail(ChessController)	load pane
+ void update()	check turn and swtich radio to those team update time

2.7.5 class ChessOnlineDetail

2.7.5.1 Field

- long nanoSecond	1 second in nano second
- Label labelNameA	instance of player a name
- Label labelNameB	instance of player b name
- Label labelTimeA	instance of player a time
- Label labelTimeB	instance of player b time
- RadioButton radioA	instalcen of player a radio
- RadioButton radioB	instance of player b radio
# long timePlayerA	player a time
# long timePlayerB	player b time
# BoardGameOnlineController gameControl	game control instance

2.7.5.2 Method

+ long getTimePlayerW()	return player a time
+ long getTimePlayerB()	return player b time
+ void setTimePlayerW(long)	set player a time
+ void setTimePlayerB(long)	set player b time
+ String getNameWhite()	return player a name from label
+ String getNameBlack()	return player b name from label
+ void setName(String, String)	set player name
+ void increaseTime(long)	increase time
+ void handleGiveUp(MouseEvent)	show message confirm and call end game
+ void handleFlip(MouseEvent)	call flip board
+ ChessOnlineDetail(BoardGameOnlineController)	load pane
+ void update()	update radio and time

2.7.6 class QuantumChessDetail

2.7.6.1 Field

- long nanoSecond	1 second in nano second
# long timePlayerA	player a time
# long timePlayerB	player b time
# QuantumChessController gameControl	game control
- Label labelNameA	instance of player a name
- Label labelTimeA	instance of player a time
- Label labelNameB	instance of player b name
- Label labelTimeB	instance of player b time
- RadioButton radioA	instalcen of player a radio
- RadioButton radioB	instalcen of player b radio
- Slider moveProb	instance of prob slider
- Label labelProb	instance of prob label

2.7.6.2 Method

+ long getTimePlayerW()	return player a time
+ long getTimePlayerB()	return player b time
+ void setTimePlayerW(long)	set player a time
+ void setTimePlayerB(long)	set player b time
+ String getNameWhite()	retun player a name from label
+ String getNameBlack()	return player b name from label
+ void setName(String, String)	set player name
+ void increaseTime(long)	increase time
# void handleFlipBoard(MouseEvent)	call flip board

# void handlePass(MouseEvent)	call pass
# void handleMoveProbChange(Number, Number)	change prob value on game control
# void handleQuit(MouseEvent)	show message confirm and call end game
+ void QuantumChessDetail(QuantumChessController)	load pane and add listener

2.7.7 class QuantumChessOnlineDetail

2.7.7.1 Field

- long nanoSecond	1 second in nano second
# long timePlayerA	player a time
# long timePlayerB	player b time
# QuantumChessController gameControl	game control
- Label labelNameA	instance of player a name
- Label labelTimeA	instance of player a time
- Label labelNameB	instance of player b name
- Label labelTimeB	instance of player b time
- RadioButton radioA	instalcen of player a radio
- RadioButton radioB	instalcen of player b radio
- Slider moveProb	instance of prob slider
- Label labelProb	instance of prob label

2.7.7.2 Method

+ long getTimePlayerW()	return player a time
+ long getTimePlayerB()	return player b time
+ void setTimePlayerW(long)	set player a time
+ void setTimePlayerB(long)	set player b time
+ String getNameWhite()	retun player a name from label
+ StringgetNameBlack()	return player b name from label
+ void setName(String, String)	set player name
+ void increseTime(long)	increase time
# void handleFlipBoard(MouseEvent)	call flip board
# void handlePass(MouseEvent)	call pass
# void handleMoveProbChange(Number, Number)	change prob value on game control
# void handleQuit(MouseEvent)	show message confirm and call end game
+ QuantumChessOnlineDetail(QuantumChessControlle r)	load pane and add listener

2.8 Package scene.gameboard.shareObject

2.8.1 class Animation

2.8.1.1 Field

- Animation INSTANCE	instance
- Runnable onFinish	runnable function which it will run on animation finish
- ChessPiece source	piece will move somewhere
- Tuple<Integer, Integer> sink	piece will move here
- boolean isAnimating	animation is animating, right?
- int animationTimeSpeed	animation will end in this time
- long startNanoTime	start time in nano seconds
- long stopNanoTime	stop time in nano seconds
- double tmpTime	temp time
- double tmpX	temp x
- double tmpY	temp y
- double tmpDistance	temp distance

2.8.1.2 Method

+ Animation getInstance()	return instance
+ Animation()	initial variable
+ Animation(ChessPiece, Tuple<Integer, Integer>, Runnable)	initial variable and start animation
+ boolean isAnimating()	return isAnimation
+ void startAnimate(ChessPiece, Tuple<Integer, Integer>, Runnable)	set source and sink; set isAnimation to true; get distance to tmpDistance set startTime and stopTime set function on finished play move sound
+ void update(long)	check if isAnimation is not true then return set tmpTime to remain time and calculate position x and y set piece to x and y check animation end
+ void stopAnimate()	run onFinish function set isAnimation to false set source and sink to null stop move sound
- double calculateDistance()	petagorat distance
+ void handle(long)	run update function run update function in game holder

2.8.2 class GameHolder

2.8.2.1 Field

- <u>GameHolder INSTANCE</u>	instance of gameHolder
+ <u>Image bb</u>	instance of bb image
+ <u>Image bk</u>	instance of bk image
+ <u>Image bn</u>	instance of bn image
+ <u>Image bp</u>	instance of bp image
+ <u>Image bq</u>	instance of bq image
+ <u>Image br</u>	instance of br image
+ <u>Image wb</u>	instance of wb image
+ <u>Image wk</u>	instance of wk image
+ <u>Image wn</u>	instance of wn image
+ <u>Image wp</u>	instance of wp image
+ <u>Image wq</u>	instance of wq image
+ <u>Image wr</u>	instance of wr image
+ <u>Image bgDark</u>	instance of bgDark image
+ <u>Image bgLight</u>	instance of bgLight image
+ <u>AudioClip pieceMove</u>	instance of move sound
+ <u>AudioClip pieceDead</u>	instance of dead sound
+ <u>double size</u>	size for each cell
+ <u>boolean isTurnEnded</u>	isTurnEnded
- <u>List<IRenderable> entity</u>	list of piece of both side

2.8.2.2 Method

+ <u>GameHolder getInstance()</u>	return instance
+ <u>List<IRenderable> getEntity()</u>	return list of entity
+ <u>ChessPiece getPiece(Tuple<Integer, Integer>)</u>	find piece which i and j are same
+ <u>ChessPiece getPieceFromMouse(Tuple<Integer, Integer>)</u>	find piece which i and j are same
+ <u>void addEntity(IRenderable)</u>	add entity to list
+ <u>void addEntity(ArrayList<IRenderable>)</u>	add entity to list
+ <u>void setEntity(ArrayList<IRenderable>)</u>	copy entity parament to list
+ <u>GameHolder()</u>	set size to 60
+ <u>void update()</u>	check entity if entity is destroyed the remove and ply dead sound
+ <u>void loadResource()</u>	load all image and audioClip

2.8.3 interface IRenderable

2.8.3.1 Method

+ int getZ()	nothing
+ void draw(GraphicsContext)	nothing
+ boolean isDestroyed()	nothing
+ boolean isVisible()	nothing

2.9 Package scene.gameBoard.view

2.9.1 class ChessBackGround

2.9.1.1 Method

public ChessBackGround()	call construct parent and set z to 0.
public draw(GraphicsContext)	draw background on board game

2.9.2 class ChessValidMoves

2.9.2.1 Field

- <u>double h</u>	Default entity height - set to GameHolder.size
- int row	the row of entity
- int col	the col of entity

2.9.2.2 Method

public ChessValidMoves(int, int)	- call construct parent - initial row to x and col to y. - set position on screen
public getI()	- return row
public getJ()	- return col
public draw(GraphicsContext)	- fill oval on center of cell and color red, alpha 0.5
public drawProbability(GraphicsContext, double)	- fill probability text on left bottom corner cell

2.9.3 class Entity

2.9.3.1 Field

# double x	- vertical position on canvas
# double y	- horizontal position on canvas
# int z	- level off entity
# boolean visible	- this entity can visible
# boolean destroyed	- this entity have been destroyed

2.9.3.2 Method

Entity()	initial visible to true and destroyed to false
getX()	return x
getY()	return y
getZ()	return z
isDestroyed()	return destroyed
isVisible()	return visible
setZ(int)	set z
setPositionOnScreen(double, double)	set x and y

draw(GraphicsContext)	nothing
Destroy()	set destroyed to true