Reversi Documentation

In the cond and Coordiate class, there are 2 class that store data type.

GamePlay class:

+ checkPossibleMove: this method will return all possible which player can move. This method also stores all possible moves arrPosibleMove which is need in another method.

+ countPlayerScore: count player’1 score, player’2 score.

+ checkEndGame: check if the game is over.

+ flipChesss and goFind: There are 2 methods to find the pieces are captured.

+ checkEndGame: check the board is full 64 pieces or not.

Minimax Class:

The main idea of this class is that I recursive in 6 depths. At each depth, I count the move that will give Bot the highest score, and the movement will give the player the lowest score.

+ **private int**[][] copyBoard(**int**[][] board, GamePlay gamePlay, **int** x, **int** y, **int** player: this a method to pass the current stage of the board to the next depth and also flip chess on the board.

+ **public int** decision(**int** depth, **int**[][] board, **int** curPlayer): this is a function find the best value (minimize value or maximize value). In this feature, on each player turn, this function will find the move can give minimize (if the player turns) value or maximize (if the BOT turns) value.

+ **public** cond decision(**int** x, **int** y, **int**[][] board) This is function have the same functionality as the above function. The differences are this function just solves the first depth and returns the best move that BOT will have the highest value.

+ **private int** score(**int**[][] arr, **int** turn): This is the function that calculates the value that the player gets. The more important positions the player has, the higher the score will be.

RegionScore class:

This is a class where I save default importance value for each position on the board.

PlayerVsPlayer class:

+ **protected void** computeBoard(): this method will check the state of each cell on board( 1: player’1 , 2 player’2, 3 this cell can move)

+ **public void** resetArray(): Initialize the chessboard in its initial state with four default cells.

+ **public void** press(): this method will wait until receive the cell that the player moved a valid move.

+ Boolean checkCanMove(**int** x, **int** y): this function will check (x, y) is the valid move or not.

+ **protected void** getRowColumn(**int** x, **int** y): this method is convert the value get on JPanel to the actual row and column.

+ **protected void** winner(): this method will show a pop up winner

+ **public void** actionGame(): this a method to show what my code do in each player step.

+ **public void** newGame(): this a method reset and pre-calculate everything in new game.

+ **protected** PlayerVsPlayer(): default constructor

+ **public void** running(): This is method help me to handle multithread. Because the actionGame() needs to loop all of time, I use timer and timerTask to call actionGame() after each 100 milliseconds.

+ **public static** PlayerVsPlayer getInstance(): method for Singleton Design Pattern.

PlayerVsBot extends PlayerVsPlayer class

+ **private void** botThink(): this method will call Minimax to find to optimal move for BOT.

+ **public void** press(): this method override the method on PlayerVsPlayer. When step is equal to 2(BOT Move), it will call the botThink() to find the bot move’s.

+ **public static** PlayervsBot getInstance(): method for Singleton Design Pattern.

RenderBoard class

+ **public void** paintBoard(Graphics g): this function will read the image board and paint it on the JPnanel.

+ **public static** RenderBoard instance(): method for Singleton Design Pattern.

RenderChess class

+ **public void** paintScore(Graphics g, **int** p1Score, **int** p2Score): This method will paint the player 1 score’s and the player 2 score’s.

+ **public void** paintCurrentMove(Graphics g, **int** step): painting the move which player is.

+ **public void** paintChess(Graphics g, ArrayList<Coordinate> board): this method receive an Array list board which is the list of changes on the board. After that, it will paint all these changes.

+ **public static** RenderChess instance(): method for Singleton Design Pattern.

Render class extends JPanel:

+ **public void** setBoard(ArrayList<Coordinate> board, **int** p1Score, **int** p2Score, **int** step): it have the same functionality with constructor.

+ **protected void** paintComponent(Graphics g): the method will paint everything on the panel.

+ **public void** winner(**int** id): show the Pop up winner

+ **public void** noMoves(**int** step): show the Pop up to warning that the current player cannot move.

Gui class:

+ **private void** startPlayerVsPlayer(): the private method to handle new Player vs Player.

+ **private void** startPlayerVsBot(): the private method to handle new Player vs Bot.

+ **private void** setNewGameJMenu(): add action Listener for newGameJMenu, when it is clicked, the new game is start

+ **private void** setBackToMenu(): (): add action Listener for backToMenu, when it is clicked, the main menu will be show.

+ **private void** setStopMusic(): pause music.

+ **private void** setPlayMusic(): resume music

+ **private void** setExitJMenu(): exit the programs

+ **private void** setMenu(): set all small component for menu bar

+ **public static void** newGamePlayerVsPlayer(): this method calls

startPlayerVsPlayer()

+ **public static void** newGamePlayerVsBot(): this method calls startPlayerVsBot()

Parameter class: in this class will store all the parameter which I need in another class like the size of the frame, image Paths, music path, player 1 name, player 2 name, etc.

Options class:

+ **private void** initComponents(): in this method, it will handle all the way to show jbutton, jlabel on jFrame.

+ **private void** brownJButtonActionPerformed(java.awt.event.ActionEvent evt): it will set the board which player chosen is brown.

+ **private void** grayJButtonActionPerformed(java.awt.event.ActionEvent evt): it will set the board which player chosen is gray.

+ **private void** greenJButtonActionPerformed(java.awt.event.ActionEvent evt): it will set the board which player chosen is green.

+ **private void** yellowJButtonActionPerformed(java.awt.event.ActionEvent evt): it will set the board which player chosen is yellow.

PlayerNames class:

+ **private void** initComponents(): in this method, it will handle all the way to show jbutton, jlabel, jTextField on jFrame.

+ **public void** saveJbuttonActionPerformed(java.awt.event.ActionEvent evt): this method will get the Player Name input from JtextField and set it in Parameter.

Rules class:

+ **private void** initComponents(): in this method, it will handle all the way to show jbutton, jlabel on jFrame.

drawRule class: this class extends from jpanel. It has a paintComponent method to paint the rules picture.

MenuView class:

+ **private void** initComponents(): in this method, it will handle all the way to show jbutton, jlabel, jTextField on jFrame.

+ **private void** playervsplayerJButtonActionPerformed(java.awt.event.ActionEvent evt): this method will call class Player Name for user input their name

+ **public void** actionGui(): this method check whenever the PlayerNames class close, it will start player vs player

+ **private void** optionsJButtonActionPerformed(java.awt.event.ActionEvent evt): call class options

+ **private void** playervspcJButtonActionPerformed(java.awt.event.ActionEvent evt): start player vs bot

+ **private void** guideJButtonActionPerformed(java.awt.event.ActionEvent evt): call guide class

+ **public static void** getInstance(): method for singleton design pattern

UML Diagram

