SMATRT PHONE APPS DEVELOPMENT ASSIGNMENT 1

NAME : ZHAN Hui
UNIVERSITY NUMBER : 3035249227

0. README

APP Name:

Othello R&B

Brief:

Othello R&B is a strategy board game, required two players to take the red and blue pieces for each. It provides players the atmosphere of gaining fun from gaming with others through the sweet interface and sounds efforts.

Game Rules:

And to learn the specific game rules, please referring to WikiPedia Othello #Rules (https://en.wikipedia.org/wiki/Reversi#Rules).

Chapters:

Inside this readme file, I will introduce Othello R&B in the following aspects:

1.**Activities**: to introduce the activities of the game.
2.**Interface**: to introduce the android activities and the responding interface.
3.**Sounds**: to introduce the sounds design of the game.
4.**Function**: to introduce the functions Othello R&B can do.
5.**Design**: to introduce the overall design and class design.
6.**Implement**: to elaborate the implement of each class.

1. Activities

Othello R&B has two activities:

1.1 Welcome Activity

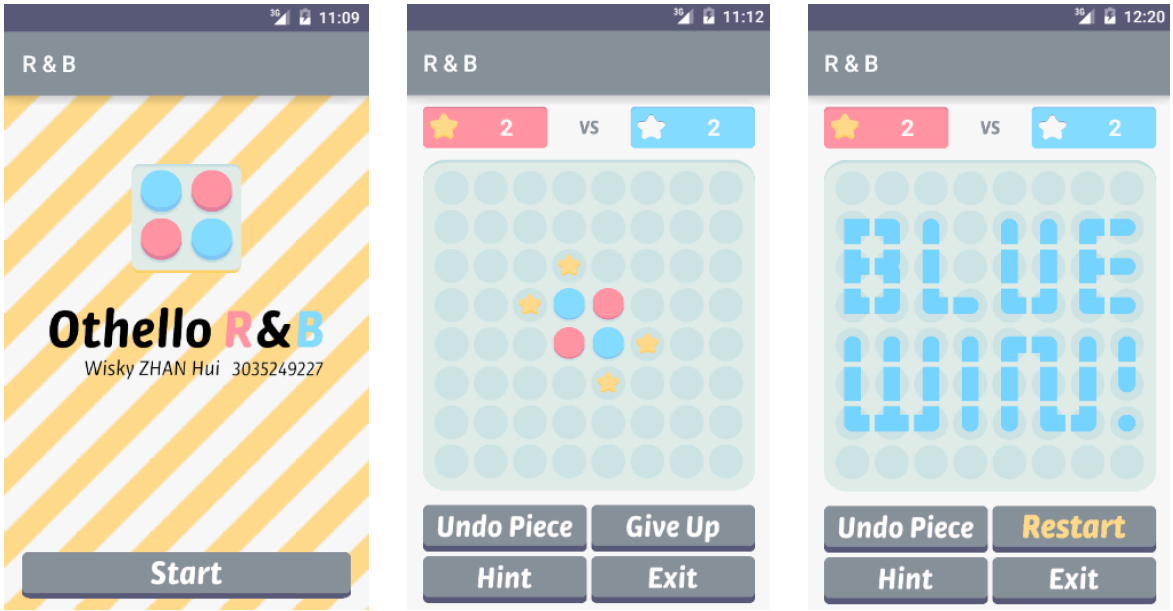
This activity is used to display the logo, name and author of the game, and it provides a button for user to jump into the game activity.

1.2 Game Activity

This activity is designed as the main gaming activity, users playing the game in this activity (playing chess, showing piece count, showing game results, etc.).

2. INTERFACE

In this part I will display the interface of the two activities and show the responding functions of the interface (buttons, display views, etc.).



2.1 Welcome Activity

The left picture above is the interface of Welcome Activity.
In this activity, it displays the logo, name and author of the game.

Start Button: the button at the bottom of the interface is game start button. By clicking this button, APP will jump to the Game Activity.

2.2 Game Activity

The right two pictures above are the interface of Game Activity.
In this activity, it displays the chessboard, chess pieces, scoreboards and function buttons.

Scoreboards: the rectangles below the activity bar, red left and blue right. They are used to show the chess piece count of each player and instruct who is the next one to go chess (PS. it is the one whose scoreboard's star is bright that take the turn to go chess next).
Chessboard: the square in light green below the scoreboards is chessboard. It is used to 1.display the current chessboard, 2.for user to put chess pieces (it will refresh itself after each player put pieces) and 3.to display the result of the game when the game meets it end.
Chess Box: the circle shapes in dark green are the chess boxes contained in the chessboard. They are used to place chess pieces, each user can put their chess piece on it if meeting the game rules.
Chess Piece Symbol: the flat wafers (red and blue circle shaped wafers, and yellow star shaped wafers) contained in the chessboard. The red and blue wafers are used to display the chess boxes taken by players, and the star wafers are used to instruct the chess boxes can put chess piece next turn (hint).
Undo Piece Button: the left button below the chessboard. By clicking this button, user can undo the chess piece he/she put last time.
Give Up & Restart Button: the right button below the chessboard. This button has two modes, give up mode and restart mode. By clicking the button in give up mode, the current player confirms to give up and the game finished; by clicking the button in restart mode, the app will start a new game.
Hint Button: the button below undo piece button. By clicking this button, app will switch on or off the hint display (star symbols) base on whether current hint switch is off or on
Exit Button: the button below give up and restart button. By clicking this button, app will quit current game and back to Welcome Activity.

3. SOUNDS

In this part I will point out the sounds efforts in the game and explain the reasons. Othello R&B uses overall 6 sounds in different situations, and the sound resources are downloaded from Japanese sound efforts designer Taira Komori's website (http://taira-komori.jpn.org/freesounden.html).

3.1 Game Start

Situation: When the game starts, the APP will play a sound to let players know.
Sound: A rhythm of guitar strumming.

3.2 Game Finish

Situation: When someone wins the game or the game draws, APP will play a congratulation rhythm.
Sound: A bright bell ring rhythm.

3.3 Piece Put

Situation: The sound played when player put his/her chess piece in a legal position, to note the user his/her action is accepted.
Sound: A sound imitates a wafer dropped on table.

3.4 Piece Cannot Put

Situation: The sound played when player touch a position where he/she cannot put chess piece on, to note the user his/her action is illegal and refused.
Sound: A sound imitates a wafer kicked back.

3.5 Player Pass

Situation: When one of the players has no chess box to put piece, the APP will play a sound to note the user his/her turn is passed and the user turn changes to his/her counter part.
Sound: A crisp sound.

3.6 Button Click

Situation: When the button clicked, the APP will play a sound.
Sound: Double crisp sounds.

4. FUNCTION

In this part, I will elaborate all of the functions the Othello R&B has (to show the objectives and their react on the interface).

4.1 Basic Function: the following functions are to satisfy the basic needs of Othello

4.1.1 Go Chess

This function ensures the basic gaming steps.
For each player clicks the chess boxes,

1.APP will check if this player can put his/her chess piece here, if can, go to step 2; if not, reject the action and play a sound let player know.
2.search the whole chessboard to find out all the rival's chess pieces that can be reversed to the current player's color,
3.reverse rival's chess pieces;

4.1.2 Show Player Turn

For each time one player does an accepted go chess action, the APP will check which player will take the next user turn, then APP will give the next player a bright star on his/her scoreboard.

4.1.3 Show Piece Count

For each time player does an accepted go chess action, APP will count the total all pieces both sides and then show the piece count on the scoreboards.

4.1.4 End Up Game

For each time player does an accepted go chess action, APP will check if the current turn satisfies one of the following situations:

1.all chess boxes are taken by chess pieces;
2.one of the players reverses all the chess pieces of his/her rival,

if yes, APP will finish the game; if no, the game will continue.

4.1.5 Show Result

When the game meets its end, APP will print the game result on the chessboard;

4.1.6 Restart Game

When the game finished, APP will switch Give Up & Restart button into restart mode. And then if user clicks on it, APP will initial the game and start a new game.

4.2 Additional Function:

4.2.1 Hint

This function allows users to turn on/off the hints, to show/off the positions he/she can put piece on. When the hint switch is turned on, APP will mark all the positions can place piece as bright stars.
When the current game is finished, hint button is disabled.

4.2.2 Undo Piece (retract)

This function allows users undo his/her go chess action last time, and the whole chessboard, player turn and scoreboard will retract to the last time.
When the current game is finished, undo piece button is disabled.

4.2.3 Giving Up

This function allows one of the players who find himself/herself is almost lose or do not want to continue current game, and then he/she could click give up button. When the give up button is clicked on, APP will judge the player who clicks as the lose part and the rest one as the winner. After give up a game, APP will set give up and restart button on restart mode.

4.2.4 Audio

As elaborated at "3. SOUNDS".

5. DESIGN

In this part, I will elaborate the overall design of the game (to show all of the function shown at "4. FUNCTION" are responded by witch classes separately) and the specific classes design

5.1 Overall Design

According to the function needs of the game, I mainly design following 4 core function blocks:

5.1.1.Chessboard: this block provides the place to game, and this class takes the responsibility to remember the current status (all chess pieces positions od each turn, all hints positions of each turn, the user turn of each turn, the piece count of both side, the hint switch status, the game status) of whole game and the game logic (initial game, end game, go chess, undo piece, count pieces, check hint, check reverse, refresh chessboard, switch hint).

5.1.2.ChessBox: this block provides the place to put piece, and takes the responsibility to remember the status of the chess piece contained inside it (the chess piece symbol, and the position of the box) and the chess piece logic (change the symbol of the chess piece).

5.1.3.Scoreboard: this block takes responsibility to remember the status of player (the piece count, user turn) and display the player status.

5.1.4.Othello Button: this block takes responsibility to remember button status (button name, face image, double switch status) and the click logic.

5.2 Classes

According to the overall design, here comes the specific class design:

5.2.1. Chessboard:

Type	Member variables	Only the core variables
Private Int [64][64]	chessPosition	To record all chess positions of each turn
Private Int [66][64]	canPtPosition	To record all hint positions of each turn
Private Int [62]	userRound	To record user turn of each turn
Private Int	redCount	To record red piece count
Private Int	blueCount	To record blue piece count
Private Boolean	hintSwitch	To record hint switch status (on/off)
Public Boolean	gameFinished	To record game status (finished yes/no)
Type	Member methods	Only the core methods
Private Void	initGame(Context)	To initial the whole game
Private Void	finishGame(int)	To end the game up
Public Boolean	goChess(int, int)	Core method to go chess
Public Void	undo(int)	Core method to undo a piece
Private Void	coungChessPiece(int)	To count the pieces of both sides

Type	Member methods	Only the core methods
Private Void	judgeCanPtPosition(int, int)	To search all positions can put piece
Private Void	ifCanPt(int, int, int)	To judge whether a position can put piece
Private Void	recolorRivalPieces(int, int)	To search all rival pieces can be reversed
Private Void	refreshChessboard(int)	To fresh all chess boxes status
Private Void	setNextUserRound(int, int)	To set which player will take next turn

5.2.2.ChessBoxAdapter

Type	Member variables	Only the core variables
Private ArrayList	chessBoxList	To record all chess boxes in the chessboard
Private Int [64]	chessOrder	To record initial chess positions
Type	Member methods	Only the core methods
Public Int	getCount()	To get the chessboard size
Public ChessBox	getItem(int)	To get the item in a certain position
Public View	getView (int, View, ViewGroup)	To get view

5.2.3.ChessBox

Type	Member variables	Only the core variables
Private Int []	chessTypes	To record all chess piece symbols
Private static Int	count	To record count of all ChessBoxes
Public Int	identifier	To record the ChessBox ID
Type	Member methods	Only the core methods
Public Int	getChessType(int)	To get the chess box type
Public Void	setChessType(int)	To set chess piece symbol in box

5.2.4.Scoreboard

Type	Member variables	Only the core variables
Private Int	userColor	To record the owner's user color
Private Int	chessCount	To record the owner's piece count
Type	Member methods	Only the core methods
Public Void	setScoreboard(int,int)	To set user turn and chess piece count

5.2.5.OthelloButton

Type	Member variables	Only the core variables
Private Int	buttonFace	To record the button face image
Public boolean	doubleSwitch	To record whether has double modes
Type	Member methods	Only the core methods
Public Boolean	onTouch (View,motionEvent)	To respond when the button click

6. IMPLEMENT

In this part I will show the implement of several extremely significant core methods.

6.1 Chessboard - goChess(int turn, int position)

```
// judge user color ( next chess piece color: r/b )
int userColor = userRound[turn];
// load chessboard of last turn into the chessboard of this turn
chessPosition[turn] = new int[64];
for(int i = 0; i < 64; i++) chessPosition[turn][i] = chessPosition[turn-1][i];
// judge if could put chess piece two situations cannot put chess :
// 1. box has been occupied; 2. can not eat counterpart's chess piece
if(chessPosition[turn][position]!=0 || canPtPosition[turn][position]==0) return false;
// re-colored rival chess pieces
recolorRivalPieces(turn, position, userColor);
// judge the can put chess pieces of next time
judgeCanPtPosition(turn, (3 - userColor));
// then refresh the whole chessboard
refreshChessboard(turn);
// then count the chess pieces
countChessPieces(turn);
// then set the next user round
setNextUserRound(turn, userColor);
```

6.2 Chessboard - undo(int turn)

```
// check if the turn has chess piece to undo
if(turn <= 0) return false;
// refresh the whole chessboard
refreshChessboard(turn-1);
// count the chess pieces
countChessPieces(turn-1);
```

6.3 Chessboard - ifCanPt(int turn, int position, int userColor)

```
// check if position is null
if(chessPosition[turn][position]!=0) return false;
// transfer position to vector (row, col)
int row = getRow(position); int col = getCol(position);
// get edges
int r = 7-col; int dn = 7-row;
// up
for(int i = 1; i <= row; i++){
    // transfer vector(row, col) to position
     int tmpP = getPosition(row-i, col);
```

```
        // get chess position
        int tmpC = chessPosition[turn][tmpP];
        // to check if the rival's chess piece
        if(tmpC == (3-userColor))continue;
        // to check if meet the tail
        if(tmpC == userColor){if(i>1) return true; break;}
        // to check if over flow
        if(tmpC == 0 || i == row)break;
    }
    // up-right
    for(int i = 1; i <= Math.min(r, row); i++){...}
    // all other directions
    {...}
    // left-up
    for(int i = 1; i <= Math.min(col, row); i++){...}
```

6.4 Chessboard - recolorRivalPieces(int turn, int position, int userColor)

```
    int row = getRow(position); int col = getCol(position);
    int r = 7-col; int dn = 7-row;
    // stack to store the rival pieces can be recolored
    java.util.Stack candidate = new java.util.Stack();
    // up
    // push the backtracking stop node into the stack
    candidate.push(position);
    for(int i = 1; i <= row; i++){
        int tmpP = getPosition(row-i, col);
        int tmpC = chessPosition[turn][tmpP];
        // if meet rival pieces, push into stack
        if(tmpC == (3-userColor))candidate.push(tmpP);
        // meet self piece, break out
        if(tmpC == userColor)      break;
        // no rival pieces can be recolored, pop all pieces out till stop node
        if(tmpC == 0 || i == row){while((int)candidate.pop()!=position){} break;}
    }
    // up-right
    candidate.push(position);
    for(int i = 1; i <= Math.min(r, row); i++){...}
    // all other directions
    {...}
    // left-up
    candidate.push(position);
    for(int i = 1; i <= Math.min(col, row); i++){...}
   // recolor
    while(!candidate.isEmpty()) chessPosition[turn][(int)candidate.pop()] = userColor;
```

6.5 Chessboard – refreshChessboard(int)

```
int length = chessPosition[turn].length;
// refresh the pieces
for(int i = 0; i < length; i++)
    ((ChessBox)this.getItemAtPosition(i))
    .setImageResource(ChessBox.getChesstype(chessPosition[turn][i]));
// refresh the hints, if hint switch is on
if (hintSwitch)
    for(int i = 0; i < length; i++)
        if(canPtPosition[turn+1][i] == 1)
            ((ChessBox)this.getItemAtPosition(i))
            .setImageResource(R.drawable.star_box);
```

---------== END ==---------