**COMP7506 Smart Phone Apps Development Assignment 1**

November 9, 2015

**ClarkOthello**

**Introduction**

ClarkOthello is a smartphone application for Android-based phones which simulates the board game Othello. It was developed using Android Studio with a minimum API 21 and tested on the “Nexus S Api 22” Android Virtual Device.

**Files Included**

ClarkOthello (folder) - Contains all program files (including xml and java)

Images (folder) - Contains all images used in the app

ClarkOthello.apk - APK file of the app

ClarkOthello Readme.doc - Readme file for the app

**Gameplay**

The app opens with a start page that contains a “Start Game” button. After the user clicks on the button, the game begins. The game page contains an 8x8 board that initially contains two black pieces and two white pieces arranged in the 4 middle squares of the board. As outlined in the assignment guidelines, players take turns placing their black and white pieces on the board according to rules of the strategy game Othello. To place a piece, the player clicks on one of the game squares, and that player’s color piece is played on the square if the move is valid. To be a valid move, the placement needs to result in the flipping of one or more of the opponent’s pieces, which occurs when there are pieces of the opponent's color in a straight line and bounded by the piece just placed and another piece of the current player's color. The game ends when all 64 squares have been filled or if neither player can play a valid move. The color with the most pieces on the board when the game ends is the winner.

The players can start a new game at any time by clicking the “New Game” button, which will reset the board to the starting position. The players can also toggle hints on or off by pushing the toggle button next to “Show Hints:”. With the button turned to “On”, translucent pieces are used to show the player all the possible piece placements for the current player’s color.

**Design**

XML Files

|  |  |
| --- | --- |
| AndroidManifest.xml | Sets the package (“com.mattclark.clarkothello”), launches the Main\_Activity class when the app loads, assigns the icon image file ic\_launcher, which is a copy of *othello\_icon.png* (self-designed using Mac Paintbrush and the *black\_chess.png* and *white\_chess.png* files given with the project assignment), holds the second Activity: GamePage class |
| content\_main.xml | First layout that displays when app is opened. The overall container is a RelativeLayout, which contains:   * ImageView with src: *othello\_icon.png* * Button “buttonStart” that starts the game by calling a new Intent that loads the Gamepage Activity and corresponding xml file * Two TextViews (contained in a LinearLayout) that displays my name and UID number |
| activity\_game\_page.xml | Main game play layout that loads when the “Start Game” button in the Main Activity is clicked. The overall container is a RelativeLayout, which contains:   * TableView container that has 8 rows and 8 columns. Each view in the TableView is an ImageButton with a blue background (#0022ff) and initial src as *transparent.png* (given with the project assignment files). Each of the 64 ImageButtons has an OnClick function set to call the method “clickSquare” when the button is clicked * LinearLayout just below the table containing an TextView with text “Turn:” and an ImageView that holds either *black\_chess.png* or *white\_chess.png* depending on which color’s turn it is. Initially starts with the black chess image * Two LinearLayouts at the bottom corners of the screen, which both hold a ImageView (displaying *black\_chess.png* and *white\_chess.png*) and a TextView displaying the score for black and white, respectively * TextView at the bottom of the screen, centered, with text “Score” * Button named “buttonNewGame” at the top-left of the screen that starts a new game when clicked * LinearLayout at the top-right of the screen, holding a TextView with text “Show Hints:” and a ToggleButton that alternates between “On” and “Off” and displays or erases the hint images |
| winner.xml | RelativeLayout that forms the display for the Dialog alert box that is shown when the game is over. Consists of:   * LinearLayout with an ImageView (will contain either *black\_chess.png, white\_chess.png*, or *transparent.png* depending on the winner, or if there is a draw) and a TextView that displays “WINS!” or “IT’S A DRAW!” depending on the outcome   Button marked “Ok” that closes the Dialog box |
| strings.xml | Holds the string files used in the program |

Java Files

|  |  |
| --- | --- |
| MainActivity.java | Main class, extends AppCompatActivity. Loads the activity\_main.xml layout when the app loads. Holds the OnClickListener for the “Start Game” button, which calls GamePage.class when clicked |
| GamePage.java | Extends AppCompatActivity and is the class handling the main gameplay. Contains methods that control the game, which are detailed in the following sections |

.

GamePage.java Major Variables

|  |  |
| --- | --- |
| board[][] | An 8x8 array of ImageButtons. Holds the 64 ImageButton square pieces; used to reference and change the button source images, depending on which color piece is played on the square |
| boardColors[][] | An 8x8 array of type int. Holds the color of the pieces played; 0 is an “empty” square, 1 is a white square, and 2 is a black square |
| nextValidMove | ArrayLIst of type Integer. Dynamically contains the ImageButton id values of which buttons of the board can be a valid moved. Updated on each turn. |
| squaresToFlip | ArrayLIst of type Integer. Dynamically contains the ImageButton id values of which squares of the board need to be flipped on the current turn. Updated each turn. |
| blackScore, whiteScore | Type ints that keep track of the black and white scores. |
| currentTurnIsBlack | Boolean variable that keeps track of the current turn (‘true’ for black turn and ‘false’ for white turn). |
| displayHints | Boolean variable that keeps track of whether the hints need to be displayed (‘true’) or not (‘false’). |

GamePage.java Methods

|  |  |
| --- | --- |
| onCreate(Bundle savedInstanceState) | Protected void method. Called when the Activity is started from MainActivity. Loads the layout (activity\_game\_page.xml). Calls the **initializeBoard()** method. Sets the listener for the ‘toggleHints’ ToggleButton, which changes displayHints to true/false and calls the **drawHints()** or **eraseHints()** method. Sets the listener for the ‘buttonNewGame’ Button, which recalls **initializeBoard()** to restart the game. |
| initializeBoard() | Public void method. Initializes the board setup to begin the game play by doing the following:   * Sets *board[][]* to refer to the 64 ImageButtons of the board * Sets all 64 elements of *boardColors[][]* to 0 (“empty square”) * Draws the starting formation of 2 black and 2 white squares in the middle of the board * Sets *blackScore* and *whiteScore* to equal 2 each and calls the **updateScore()** method * Initializes the *nextValidMove* ArrayList to contain the ID’s of the squares of the 4 valid moves for the starting turn * Calls **drawHints()** if *displayHints* is true * Initializes *currentTurnIsBlack* to be true and calls the **setImageViewTurn()** method |
| clickSquare(View v) | Public void method. Is called when player clicks on one of the 64 ImageButtons on the game board and handles the placing of the game piece and subsequent game logic. Gets the ImageButtons ID from the View v, and if the ID is in the *nextValidMoves* ArrayList, the following occurs:   * The *row* and *col* are parsed from the square’s ID name * If *currentTurnIsBlack* is true, the ImageButton’s src file changes to *black\_chess.png*, the *boardColors[row][col]* element is updated to be 2 (for ‘black’), and the *blackScore* is incremented by 1. If *currentTurnIsBlack* is false, the ImageButton’s src file changes to *white\_chess.png*, the *boardColors[row][col]* element is updated to be 1 (for ‘white), and the *whiteScore* is incremented by 1. * The **flipSquares(row,col)** method is called to flip the bounded square pieces. * The **updateScores()** method is called to update the score TextViews in the layout * The current square ID is removed from *nextValidMove* * *currentTurnIsBlack*is changed to it’s negation * The hints are removed by calling **eraseHints** if *displayHints* is true * The **fillNextValidMove()** method is called to reset the *nextValidMove* ArrayList * If *nextValidMove* comes back empty, then there was no valid move for the color. The turn is changed and **fillNextValidMove()** is called again. If *nextValidMove* is empty again, then neither color could play, and the game is over. The **gameOver()** method is called. * The hints are drawn by calling **drawHints** if *displayHints* is true * The **setImageViewTurn()** method is called   If the ID of the View v is not in *nextValidMoves*, the click is ignored. |
| isValid(ImageButton sqr) | Public boolean method. Checks to see if *sqr* is in the *nextValidMove* ArrayList. Returns true or false if so. |
| updateScores() | Public void method. Updates the score TextViews (‘textBlackScore’ and ‘textWhiteScore’) in the layout |
| fillNextValidMove() | Public void. Checks all 64 squares to see if the current turn’s color can be played on that square. To be valid, all 8 directions (N,S,E,W,NE,NW,SE,SW) are checked to see if the piece encapsulates the opposite color with aonther of the current color. If the move is valid, the square’s ID is added to the *nextValidMove* ArrayList. |
| eraseHints() | Public void. For all the ID’s in *nextValidMove*, the src file is changed to *transparent.png*. |
| drawHints() | Public void. For all the ID’s in *nextValidMove*, the src file is changed to *black\_chess\_t.png* or *white\_chess\_t.png* depending on which turn it is. |
| gameOver() | Public void. Dynamically sets layout of *winner.xml* and builds a Dialog box to display the result of the game. If there is a winner, the corresponding ImageView is updated with *black\_chess.png* or *white\_chess.png*. If the scores are tied, the ImageView is changed to be 0x0 and the text changes to “IT’S A DRAW!”. The Ok button listener is created which closes the Dialog box when clicked. |
| flipSquares(int myRow, int myCol) | Public void. Calls all 8 of the **check<direction>()** methods with the boolean parameter set to ‘true’ so the 8 methods will update the *squaresToFlip* ArrayList. Once all 8 methods have been checked, for each ImageButton ID in *squaresToFlip:*   * *ro*w and *col* are parsed * The src of the ImageButton is set to either *black\_chess.png* or *white\_chess.png* depending on the color of the turn * *boardColors[row][col]* is changed to 1 (“white”) or 2 (“black”) depending on the color * *blackScore* and *whiteScore* are updated accordingly (current turn’s score is incremented, opposite turn’s score is decremented) |
| checkN (int myRow, int myCol, boolean addToFlip)  Also: checkS, checkE, checkW, checkNE, checkNW, checkSE, checkSW | Public Boolean. Checks whether square is a valid move by checking in the North direction. The immediate square to the North has to already be filled with the opposite color of the turn, or the move is not valid (returns ‘false’). If this condition is satisfies, continues checking north until it finds a square with the same color as the current turn (the “bounding square”). If it encounters and empty square or the boundary of the board (row < 0), then it stops checking and returns ‘false’. If another piece of the current turn is found, returns ‘true’. If *addToFlip* is true and the method returns ‘true’, the squares between the current square and the bounding square are added to the *squaresToFlip* ArrayList.  The same logic as above applies in the remaining 7 **check<direction>()** methods; the loop in each function adjusts to check in the direction indicated. |

**Limitations**

A few limitations exist:

1. The app was designed using a minimum API of 21. Earlier API’s may not handle the layout or gameplay accurately.
2. The starting position of the squares (2 black, 2 white) is hard-coded and can’t easily be changed.
3. The first turn of the game has to be black. This is hardcoded and can’t easily be changed. Correspondingly, the *nextValidMoves* ArrayList is initialized according to black having the first turn.

**Extra Features**

To satisfy the requirements of the assignment, I chose to code the “Show Hints” toggle, which displays or hides the translucent hint pieces that show the valid moves. The design of this feature is covered in the methods above.

Additionally, I designed a *winner.xml* layout to display the winner of the game. This Dialog box pops up when the game is over showing the winning color (or if it’s a draw). The user can close the dialog box by clicking “Ok” which returns to the finished game. To begin a new game, the user clicks the “New Game” button.

I designed the *othello\_icon.png* image, which is used as the app icon image and the image on the starting page. I used *black\_chess.png* and *white\_chess.png,* which were provided with the assignment, and created the image using Mac Paintbrush.

**References**

All of my code is original; however, I did a lot of research on syntax and proper use of Views, Layouts, ArrayLists, etc. and based some of my code on examples provided. For the Dialog box, I borrowed heavily from the example that was given, so I referenced that URL in the comments of my code. Other websites I went to for help and coding examples are:

Building a UI Interface:

<http://developer.android.com/training/basics/firstapp/building-ui.html>

Android UI Layouts:

<http://www.tutorialspoint.com/android/android_user_interface_layouts.htm>

Handling Events:

<http://examples.javacodegeeks.com/android/core/ui/events/android-event-handling-example/>

Changing source for ImageButton:

<http://stackoverflow.com/questions/17129589/how-do-i-programmatically-change-the-imagebutton-src-target-when-a-condition-is>

Using variable for R.id:

<http://stackoverflow.com/questions/12227310/how-can-we-use-a-variable-in-r-id>

Getting name of ImageButton in onClick:

<http://stackoverflow.com/questions/5015691/how-can-i-get-the-name-of-an-image-button-in-android>

Using Toggle:

<http://developer.android.com/guide/topics/ui/controls/togglebutton.html>

Custom Dialog (referenced in the comments in my code):

<http://www.mkyong.com/android/android-custom-dialog-example/>

Changing Icon image:

<https://romannurik.github.io/AndroidAssetStudio/icons-launcher.html>