2012

Texas A&M University: "An Aggie does not lie, cheat or steal or tolerate those who do."

TEAM 6:
Phillip Ells,
William Guerra,
Daniel Tan

[PROJECT 3 REPORT]

CSCE-315: Programming Studio (Summer 2012) - Project 2: Android Reversi

Table of Contents:

1. Project 3 Design Document

Team 6 Members:
Phillip Ells
William Guerra
Daniel Tan

Section 1: Project Goals and Purpose

The project goal is to design and implement the Reversi game from Project 2 to be playable as an application on an Android device. All the functions accessible to the user in the Reversi game from Project 2 should be accessible through a graphical user interface (GUI). The GUI design must be intuitive and visually appealing.

Section 2: High-level Design Description

The following diagram describes the basic construction of the Android GUI design of this project.

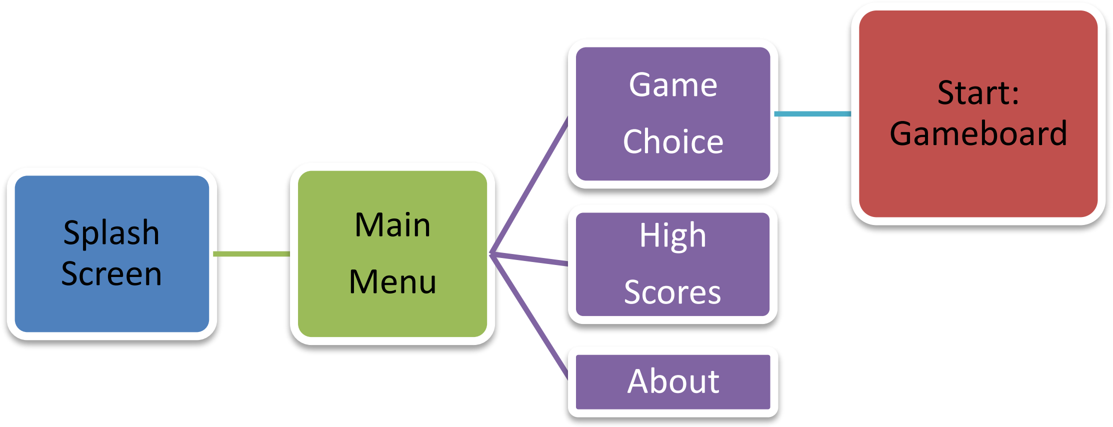


Figure 1 - Diagram of main components of the Android GUI

All the GUI components will be designed and coded in Java and in XML.

Level 1: Splash Screen

The Android Reversi application opens with a simple splash screen leading to the application's main menu.

Level 2: Main Menu

This screen connects the user to the 3 parts of the Reversi Android application: Game Choice, High Scores, and About.

Level 3A: Game Choice

This screen will allow the user to specify the game type: human vs. AI or 2 human players, the difficulty of the AI, and whether or not the human player plays as white or black. For a local 2-Player game, the color choice is irrelevant.

Level 3B: High Scores

This screen will contain a simple view of the number of wins/losses of against the different AI levels. The scores are obtained from an XML file on the SD card of the Android device. They are loaded upon the selection of the high scores menu, and are updated upon the completion of each game.

Level 3C: About

This screen will contain the app design credits, information about reversi, and will be displayed using a gesture-animated panel of images. Swipe left or right to see more.

Level 4: Gameboard

This is the heart of the Reversi Android application. This screen will contain the GUI for interacting with the Reversi engine designed for project 2.

Section 3: Low-Level Design Breakdown

Level 1 & 2 Screen Designs:

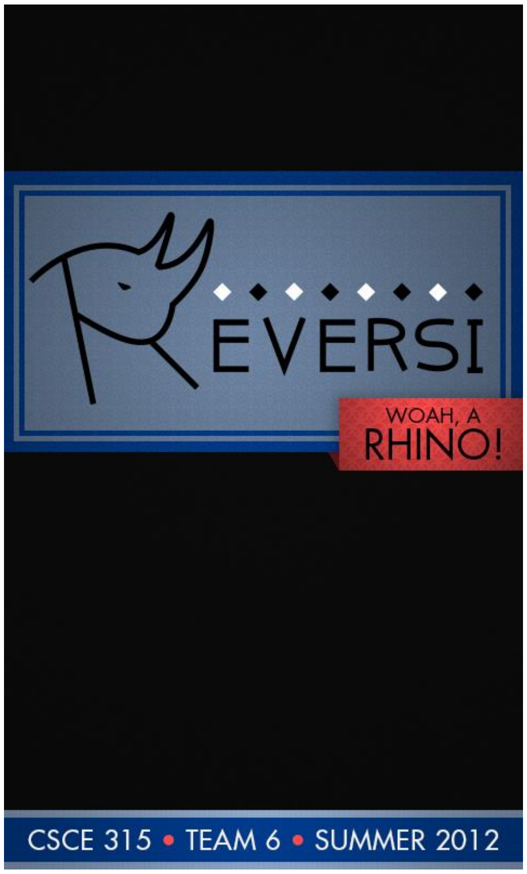


Figure 2 – Level 1 Screenshot

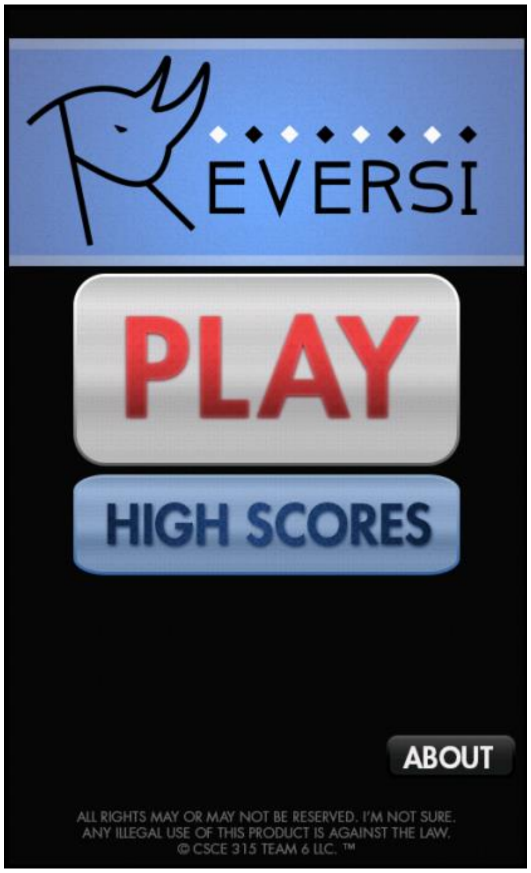
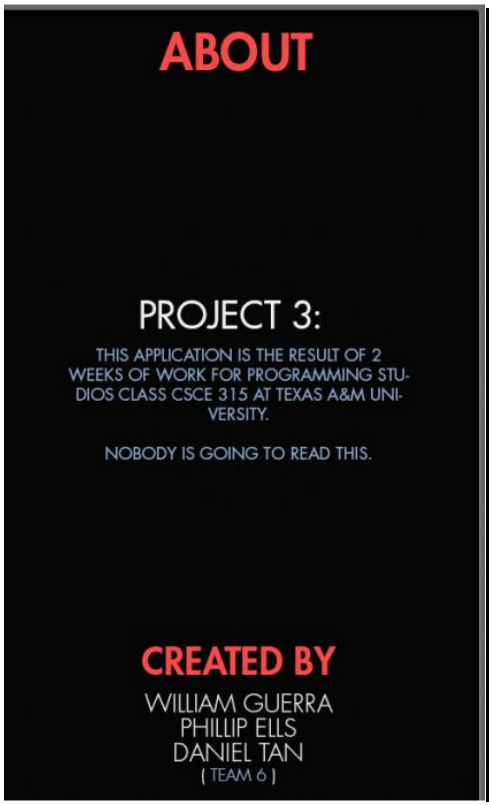

REVERSI
PLAY
HIGH SCORES
ABOUT
ALL RIGHTS MAY OR MAY NOT BE RESERVED. I'M NOT SURE.
ANY ILLEGAL USE OF THIS PRODUCT IS AGAINST THE LAW.
© CSCE 315 TEAM 6 LLC. ™

Figure 3 – Level 2 Screenshot

<u>Level 1-2 Design:</u>

The splash screen and all the menus are designed in the same basic fashion. All the screens are their own activities within the Android java code. The splash screen contains a single static background image, and the main menu has a static background overlaid with custom designed image buttons. For the sake of memory, the splash screen activity terminates as soon as the main menu activity is activated; thus, the splash screen will only appear when the application is opened rather than when the application is resumed. All the custom image buttons have their own individual xml files and images for the unclicked and pushed states. The user at any point when moving from the main menu to the other pages on level 3 can return to the main menu using the Android keypad back button.



ABOUT

PROJECT 3:

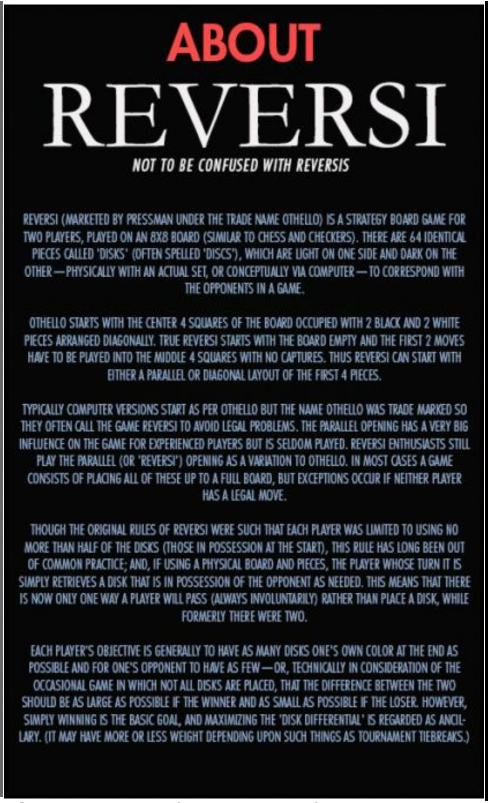
THIS APPLICATION IS THE RESULT OF 2 WEEKS OF WORK FOR PROGRAMMING STUDIOS CLASS CSCE 315 AT TEXAS A&M UNIVERSITY.

NOBODY IS GOING TO READ THIS.

CREATED BY

WILLIAM GUERRA
PHILLIP ELLS
DANIEL TAN
(TEAM 6)

Figure 4 – Level 3C Screenshot 1



ABOUT

REVERSI

NOT TO BE CONFUSED WITH REVERSIS

REVERSI (MARKETED BY PRESSMAN UNDER THE TRADE NAME OTHELLO) IS A STRATEGY BOARD GAME FOR TWO PLAYERS, PLAYED ON AN 8X8 BOARD (SIMILAR TO CHESS AND CHECKERS). THERE ARE 64 IDENTICAL PIECES CALLED 'DISKS' (OFTEN SPELLED 'DISCS'), WHICH ARE LIGHT ON ONE SIDE AND DARK ON THE OTHER—PHYSICALLY WITH AN ACTUAL SET, OR CONCEPTUALLY VIA COMPUTER—TO CORRESPOND WITH THE OPPONENTS IN A GAME.

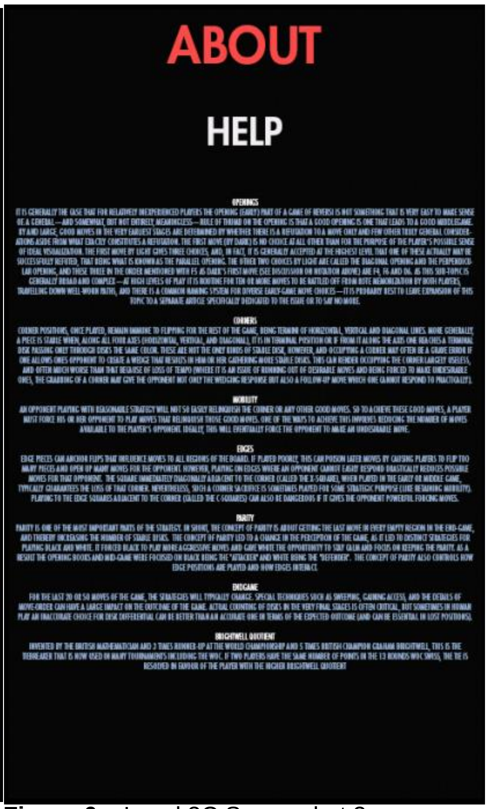
OTHELLO STARTS WITH THE CENTER 4 SQUARES OF THE BOARD OCCUPIED WITH 2 BLACK AND 2 WHITE PIECES ARRANGED DIAGONALLY. TRUE REVERSI STARTS WITH THE BOARD EMPTY AND THE FIRST 2 MOVES HAVE TO BE PLAYED INTO THE MIDDLE 4 SQUARES WITH NO CAPTURES. THUS REVERSI CAN START WITH EITHER A PARALLEL OR DIAGONAL LAYOUT OF THE FIRST 4 PIECES.

TYPICALLY COMPUTER VERSIONS START AS PER OTHELLO BUT THE NAME OTHELLO WAS TRADE MARKED SO THEY OFTEN CALL THE GAME REVERSI TO AVOID LEGAL PROBLEMS. THE PARALLEL OPENING HAS A VERY BIG INFLUENCE ON THE GAME FOR EXPERIENCED PLAYERS BUT IS SELDOM PLAYED. REVERSI ENTHUSIASTS STILL PLAY THE PARALLEL (OR 'REVERSI') OPENING AS A VARIATION TO OTHELLO. IN MOST CASES A GAME CONSISTS OF PLACING ALL OF THESE UP TO A FULL BOARD, BUT EXCEPTIONS OCCUR IF NEITHER PLAYER HAS A LEGAL MOVE.

THOUGH THE ORIGINAL RULES OF REVERSI WERE SUCH THAT EACH PLAYER WAS LIMITED TO USING NO MORE THAN HALF OF THE DISKS (THOSE IN POSSESSION AT THE START), THIS RULE HAS LONG BEEN OUT OF COMMON PRACTICE; AND, IF USING A PHYSICAL BOARD AND PIECES, THE PLAYER WHOSE TURN IT IS SIMPLY RETRIEVES A DISK THAT IS IN POSSESSION OF THE OPPONENT AS NEEDED. THIS MEANS THAT THERE IS NOW ONLY ONE WAY A PLAYER WILL PASS (ALWAYS INVOLUNTARILY) RATHER THAN PLACE A DISK, WHILE FORMERLY THERE WERE TWO.

EACH PLAYER'S OBJECTIVE IS GENERALLY TO HAVE AS MANY DISKS ONE'S OWN COLOR AT THE END AS POSSIBLE AND FOR ONE'S OPPONENT TO HAVE AS FEW—OR, TECHNICALLY IN CONSIDERATION OF THE OCCASIONAL GAME IN WHICH NOT ALL DISKS ARE PLACED, THAT THE DIFFERENCE BETWEEN THE TWO SHOULD BE AS LARGE AS POSSIBLE IF THE WINNER AND AS SMALL AS POSSIBLE IF THE LOSER. HOWEVER, SIMPLY WINNING IS THE BASIC GOAL, AND MAXIMIZING THE 'DISK DIFFERENTIAL' IS REGARDED AS ANCILLARY. (IT MAY HAVE MORE OR LESS WEIGHT DEPENDING UPON SUCH THINGS AS TOURNAMENT TIEBREAKS.)

Figure 5 – Level 3C Screenshot 2



ABOUT

HELP

OPENINGS

IT IS GENERALLY THE CASE THAT FOR RELATIVELY INEXPERIENCED PLAYERS THE OPENING (EARLY) PART OF A GAME OF REVERSI IS NOT SOMETHING THAT IS VERY EASY TO MAKE SENSE OF. A GENERAL—AND SOMEWHAT, BUT NOT ENTIRELY, MEANINGLESS—RULE OF THUMB ON THE OPENING IS THAT A GOOD OPENING IS ONE THAT LEADS TO A GOOD MIDDLEGAME. BY AND LARGE, GOOD MOVES IN THE VERY EARLIEST STAGES ARE DETERMINED BY WHETHER THERE IS A REFUTATION TO A MOVE ONLY AND FEW OTHER TRULY GENERAL CONSIDERATIONS ASIDE FROM WHAT EXACTLY CONSTITUTES A REFUTATION. THE FIRST MOVE (BY DARK) IS NO CHOICE AT ALL OTHER THAN FOR THE PURPOSE OF THE PLAYER'S POSSIBLE SENSE OF IDEAL VISUALIZATION. THE FIRST MOVE BY LIGHT GIVES THREE CHOICES, AND, IN FACT, IT IS GENERALLY ACCEPTED AT THE HIGHEST LEVEL THAT ONE OF THESE ACTUALLY MAY BE SUCCESSFULLY REFUTED, THAT BEING WHAT IS KNOWN AS THE PARALLEL OPENING. THE OTHER TWO CHOICES BY LIGHT ARE CALLED THE DIAGONAL OPENING AND THE PERPENDICULAR OPENING, AND THESE THREE IN THE ORDER MENTIONED WITH F5 AS DARK'S FIRST MOVE (SEE DISCUSSION ON NOTATION ABOVE) ARE F4, F6 AND D6. AS THIS SUB-TOPIC IS GENERALLY BROAD AND COMPLEX—AT HIGH LEVELS OF PLAY IT IS ROUTINE FOR TEN OR MORE MOVES TO BE RATTLED OFF FROM ROTE MEMORIZATION BY BOTH PLAYERS, TRAVELLING DOWN WELL-WORN PATHS, AND THERE IS A COMMON NAMING SYSTEM FOR DIVERSE EARLY-GAME MOVE CHOICES—IT IS PRIMARILY BEST TO LEAVE EXPANSION OF THIS TOPIC TO A SEPARATE ARTICLE SPECIFICALLY DEDICATED TO THE ISSUE OR TO SAY NO MORE.

CORNERS

CORNER POSITIONS, ONCE PLAYED, REMAIN IMMUNE TO FLIPPING FOR THE REST OF THE GAME, BEING TERMINI OF HORIZONTAL, VERTICAL AND DIAGONAL LINES. MORE GENERALLY, A PIECE IS STABLE WHEN, ALONG ALL FOUR AXES (HORIZONTAL, VERTICAL, AND DIAGONAL), IT IS IN TERMINAL POSITION OR IF FROM IT ALONG THE AXIS ONE REACHES A TERMINAL DISK PASSING ONLY THROUGH DISKS THE SAME COLOR. THESE ARE NOT THE ONLY KINDS OF STABLE DISK, HOWEVER, AND OCCUPYING A CORNER MAY OFTEN BE A GRAVE ERROR IF ONE ALLOWS ONES OPPONENT TO CREATE A WEDGE THAT RESULTS IN HIM OR HER GATHERING MORE STABLE DISKS. THIS CAN RENDER OCCUPYING THE CORNER LARGELY USELESS, AND OFTEN MUCH WORSE THAN THAT BECAUSE OF LOSS OF TEMPO (WHERE IT IS AN ISSUE OF RUNNING OUT OF DESIRABLE MOVES AND BEING FORCED TO MAKE UNDESIRABLE ONES, THE GRABBING OF A CORNER MAY GIVE THE OPPONENT NOT ONLY THE WEDGING RESPONSE BUT ALSO A FOLLOW-UP MOVE WHICH ONE CANNOT RESPOND TO PRACTICALLY).

MOBILITY

AN OPPONENT PLAYING WITH REASONABLE STRATEGY WILL NOT SO EASILY RELINQUISH THE CORNER OR ANY OTHER GOOD MOVES. SO TO ACHIEVE THESE GOOD MOVES, A PLAYER MUST FORCE HIS OR HER OPPONENT TO PLAY MOVES THAT RELINQUISH THOSE GOOD MOVES. ONE OF THE WAYS TO ACHIEVE THIS INVOLVES REDUCING THE NUMBER OF MOVES AVAILABLE TO THE PLAYER'S OPPONENT. IDEALLY, THIS WILL EVENTUALLY FORCE THE OPPONENT TO MAKE AN UNDESIRABLE MOVE.

EDGES

EDGE PIECES CAN ANCHOR FLIPS THAT INFLUENCE MOVES TO ALL REGIONS OF THE BOARD. IF PLAYED POORLY, THIS CAN POISON LATER MOVES BY CAUSING PLAYERS TO FLIP TOO MANY PIECES AND OPEN UP MANY MOVES FOR THE OPPONENT. HOWEVER, PLAYING ON EDGES WHERE AN OPPONENT CANNOT EASILY RESPOND DRASTICALLY REDUCES POSSIBLE MOVES FOR THAT OPPONENT. THE SQUARE IMMEDIATELY DIAGONALLY ADJACENT TO THE CORNER (CALLED THE X-SQUARE), WHEN PLAYED IN THE EARLY OR MIDDLE GAME, TYPICALLY GUARANTEES THE LOSS OF THAT CORNER. NEVERTHELESS, SUCH A CORNER SACRIFICE IS SOMETIMES PLAYED FOR SOME STRATEGIC PURPOSE (LIKE RETAINING MOBILITY). PLAYING TO THE EDGE SQUARES ADJACENT TO THE CORNER (CALLED THE C-SQUARES) CAN ALSO BE DANGEROUS IF IT GIVES THE OPPONENT POWERFUL FORCING MOVES.

PARITY

PARITY IS ONE OF THE MOST IMPORTANT PARTS OF THE STRATEGY. IN SHORT, THE CONCEPT OF PARITY IS ABOUT GETTING THE LAST MOVE IN EVERY EMPTY REGION IN THE END-GAME, AND THEREBY INCREASING THE NUMBER OF STABLE DISKS. THE CONCEPT OF PARITY LED TO A CHANGE IN THE PERCEPTION OF THE GAME, AS IT LED TO DISTINCT STRATEGIES FOR PLAYING BLACK AND WHITE. IT FORCED BLACK TO PLAY MORE AGGRESSIVE MOVES AND GAVE WHITE THE OPPORTUNITY TO STAY CALM AND FOCUS ON KEEPING THE PARITY. AS A RESULT THE OPENING BOOKS AND MID-GAME WERE FOCUSED ON BLACK BEING THE "ATTACKER" AND WHITE BEING THE "DEFENDER". THE CONCEPT OF PARITY ALSO CONTROLS HOW EDGE POSITIONS ARE PLAYED AND HOW EDGES INTERACT.

ENDGAME

FOR THE LAST 20 OR 30 MOVES OF THE GAME, THE STRATEGIES WILL TYPICALLY CHANGE. SPECIAL TECHNIQUES SUCH AS SWEEPING, GAINING ACCESS, AND THE DETAILS OF MOVE-ORDER CAN HAVE A LARGE IMPACT ON THE OUTCOME OF THE GAME. ACTUAL COUNTING OF DISKS IN THE VERY FINAL STAGES IS OFTEN CRITICAL, BUT SOMETIMES IN HUMAN PLAY AN INACCURATE CHOICE FOR DISK DIFFERENTIAL CAN BE BETTER THAN AN ACCURATE ONE IN TERMS OF THE EXPECTED OUTCOME (AND CAN BE ESSENTIAL IN LOST POSITIONS).

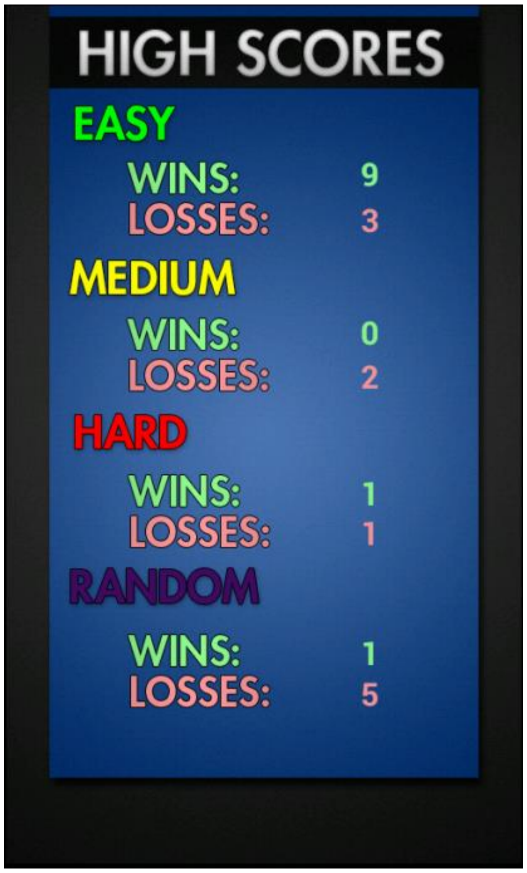
BRIGHTWELL QUOTIENT

INVENTED BY THE BRITISH MATHEMATICIAN AND 2 TIMES RUNNER-UP AT THE WORLD CHAMPIONSHIP AND 5 TIMES BRITISH CHAMPION GRAHAM BRIGHTWELL, THIS IS THE TIEBREAKER THAT IS NOW USED IN MANY TOURNAMENTS INCLUDING THE WOC. IF TWO PLAYERS HAVE THE SAME NUMBER OF POINTS IN THE 13 ROUNDS WOC SWISS, THE TIE IS RESOLVED IN FAVOUR OF THE PLAYER WITH THE HIGHER BRIGHTWELL QUOTIENT

Figure 6 – Level 3C Screenshot 3

<u>Level 3C Design:</u>

The simple About page provides credits and a brief written tutorial on playing the Reversi game (as provided by Wikipedia: http://en.wikipedia.org/wiki/Reversi). The page utilizes the Android gestures APIs to allow the user to intuitively swipe from one section of the about page to the next.


HIGH SCORES

EASY
WINS: 9
LOSSES: 3

MEDIUM
WINS: 0
LOSSES: 2

HARD
WINS: 1
LOSSES: 1

RANDOM
WINS: 1
LOSSES: 5

Figure 7 – Level 3B Screenshot

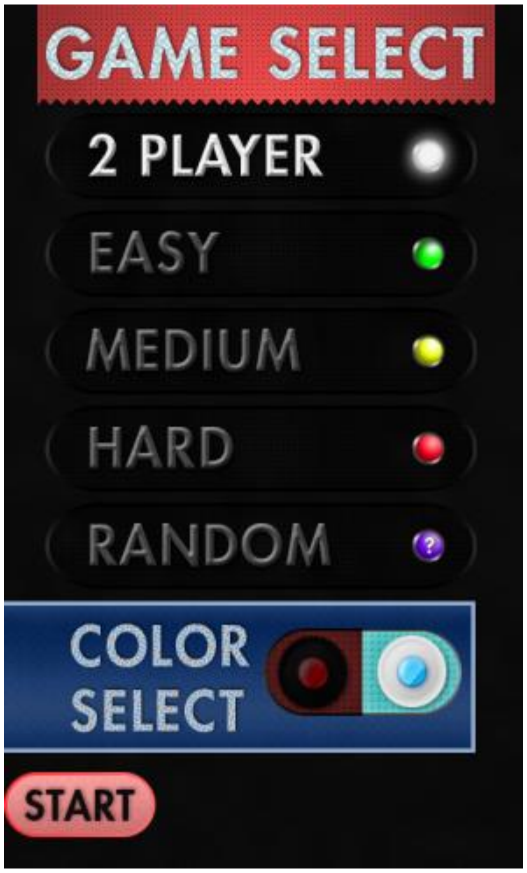


Figure 8 – Level 3A Screenshot

Level 3B Design:

The high scores page implements the use of the Android SD card. The card contains an xml file called thescores.xml which contains a simply period separated list of win/loss values for all the AI levels: easy, medium, hard, and random. When the user opens the high scores activity from the main menu, the XML file on the SD card is loaded and parsed and the appropriate values for the high scores are updated onto the Android TextView objects on the high scores XML file. This page is updated whenever the user completes a game against an AI opponent using the same load and parse process that opening the high scores activity does.

Level 3A Design:

The Game Choice menu contains Boolean variables in the java code that the Gameboard activity will access when determining how to set up the board. Here the XML and java code utilize a variety of button types: image buttons, radio buttons, and a toggle button. All buttons are custom buttons, again designed with their own individual XML files to allow different image states for unselected and selected. The radio buttons allow the user to choose to play against another human or an AI opponent. The toggle button allows the user to specify when playing against an AI opponent to play as black or white. This toggle button state is irrelevant if the user selects a 2 player game. The start button calls the Gameboard activity and switches the view to the Gameboard XML page. All the java AI code was implemented for this project; however, due to the heap and memory constraints of the Android emulators we tested with, the Hard AI takes too long to be a reasonable opponent to play against.

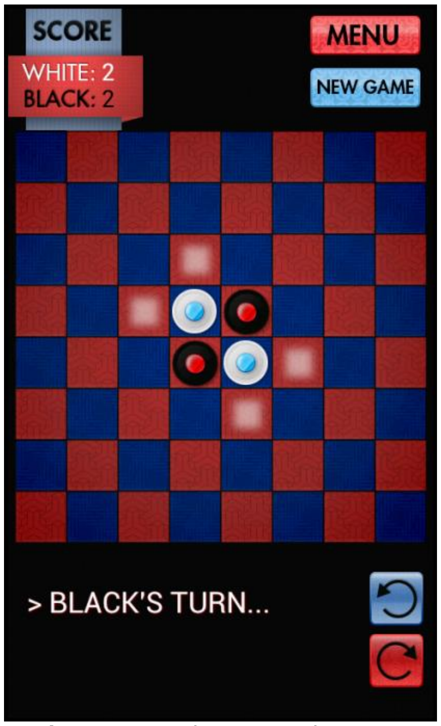


Figure 9–Level 4 Screenshot Start

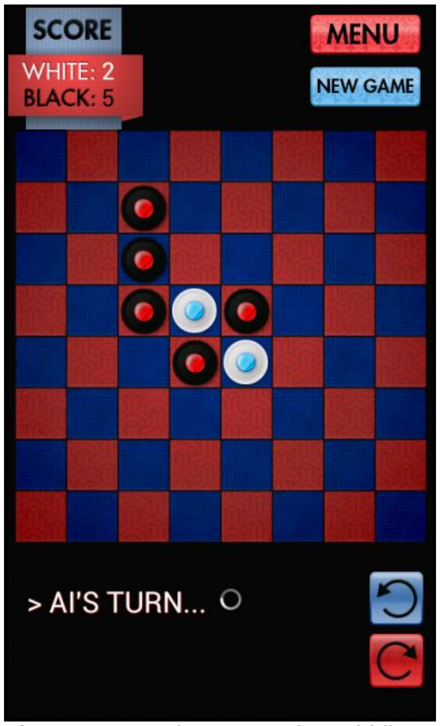


Figure 10–Level 4 Screenshot Middle

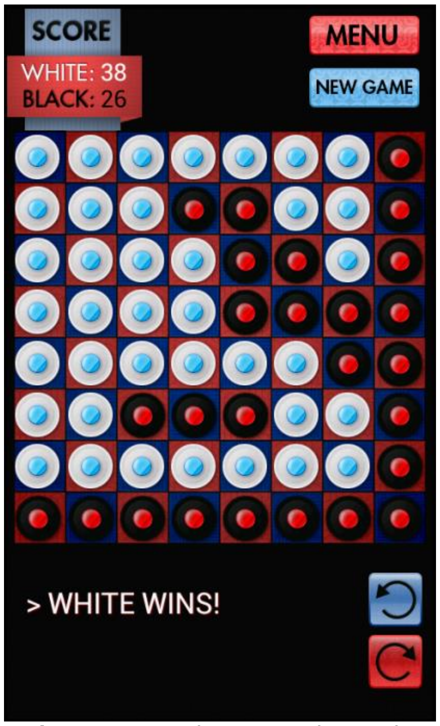
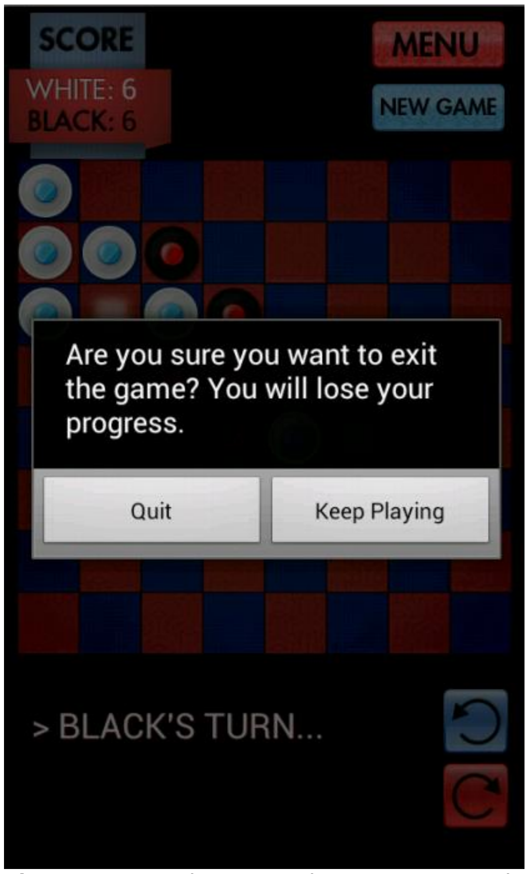
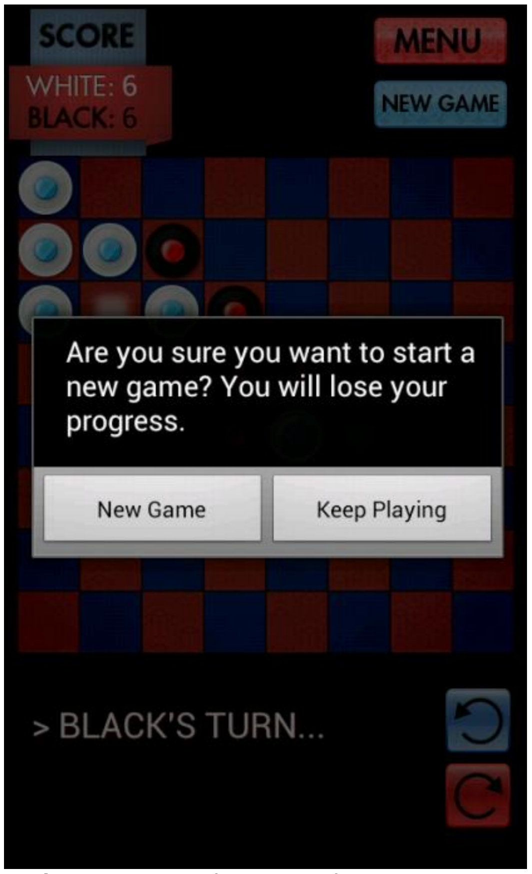


Figure 11–Level 4 Screenshot End



Are you sure you want to exit the game? You will lose your progress.

Figure 12 – Level 4 Screenshot: Menu Pressed



Are you sure you want to start a new game? You will lose your progress.

Figure 13 – Level 4 Screenshot: New Game Pressed

Level 4 Design:

Outside of the board and the pieces, the graphics present on the Gameboard screen include undo, redo, menu, new game, current score, and a game information text box. All the buttons are implemented as custom image buttons with special image states for clicked and unclicked. The texts are TextViews and are modified as necessary throughout the playing of the game to give the user information as to whose turn it is, when a player's turn is skipped, and what the current score is. The exact graphical representation of the Reversi board consisted of a static background image for the board, 64 invisible buttons on each square of the board, and 64 black piece images beneath the buttons. The resource image of the black pieces is changed to white as is appropriate given the state of the game. When a player presses on a particular valid location on the board, the black piece will be set to visible and its image resource set appropriately to black or white and all the bracketed pieces will be flipped over using an Android 3D flip animation. The user may at any time reset the board by clicking the new game button, or he or she may return to the main menu at any time by clicking the menu button. Both the menu and new game buttons provide safety prompts the check to ensure that the user intended to press either button. Also, undo and redo functions are available to the user at any point during the game except for when the game ends; which at this point the high scores XML file in the SD card is updated. The valid moves are displayed for the user during his turn or when he or she clicks on an invalid location on the board.

Project Implementation Functions:

Below is the entire list of functions implemented in the first build of the Android Reversi game.

Graphical User Interface:

SplashScreen.java
void onCreate(Bundle splashScreen) :
Displays the splash screen for 5 seconds before opening the Main Menu activity
void onPause()
Terminates the splash screen activity when the Main Menu activity is active.

AboutPage.java
void onCreate(Bundle gameChoice):
Displays the about XML file upon rendering the page.
void onTouchEvent (MotionEvent e):
Implements the swipe detection
void onFling (MotionEvent e1, MotionEvent e2, float velocityX, float velocityY):
Performs the swipe animations
void previousView():
Handles the case when the user swipes to the right
void nextView():
Handles the case when the user swipes to the left

HighScores.java
void onCreate(Bundle gameChoice):
Displays the high scores XML and loads and parses the SD card

MainMenu.java
void onCreate(Bundle mainMenu):
Displays the main menu and initializes all the interactive buttons

void onClick(View v):

Determines the actions taken when a specific button is pressed

boolean onCreateOptionsMenu(Menu menu):

Displays a pop-up menu and returns true when the user clicks the Android menu key

GameChoice.java

void onCreate(Bundle gameChoice):

Displays the game choice menu and initializes all the interactive buttons

void onClick(View v):

Determines the actions taken when a specific button is pressed; sets all the Booleans to determine game choice

boolean onCreateOptionsMenu(Menu menu):

Displays a pop-up menu and returns true when the user clicks the Android menu key

Gameboard.java

void onCreate(Bundle gameBoard):

Displays the Gameboard page and initializes all the buttons, images, and TextViews. The Gameboard also initializes the project 2 java game

void onClick(View v):

Determines the actions taken when a specific button is pressed and calls the appropriate ReversiRules.java functions

// Variable Access Functions/Utility

Gameboard getInstance():

Returns the Gameboard instance for other classes to call Gameboard functions

void madeValidMove(boolean trueWhenMoved):

Sets global Boolean to track whether or not the human player made a valid move

void setGameState():

Initializes the Gameboard Booleans, so that the code knows what options the user selected in the GameChoice menu

void switchPlayer():

A public function affecting the Boolean variable isWhite which tracks who the current player is (black or white)

boolean getHumanPlayer():

Returns the Boolean value of the human player's color (black vs. white)

void setPlayer(boolean color):

Switches to a specific player color (false for black, true for white)

void pauseFor(final int pauseLength):

Called when the Gameboard initializes to pause for a specified length of time before the user or the AI is allowed to move

// Dialog Buttons: Menu/New Game

void areYouSureDialog():

Generates an alert dialog to double check if the user wishes to go to the main menu

void newGameDialog():

Generates an alert dialog to double check if the user wishes to start a new game

// Gameboard UI manipulation

void turnOffButtons():

Renders the undo and redo buttons and any of the board buttons un-clickable

void turnOnButtons():

Renders the undo and redo buttons and any of the board buttons clickable

void addWhitePiece(int col, int row):
    Makes a white piece image visible in a specific column and row of the Gameboard
void addBlackPiece(int col, int row):
    Makes a black piece image visible in a specific column and row of the Gameboard
void addValidPiece(int col, int row):
    Makes a valid move marker image visible in a specific column and row of the Gameboard
void clearValidPieces():
    Sets all valid pieces on the board to invisible
void flipPieceAnimation(int col, int row):
    Performs the flip animation to a specific piece at a given column and row
void applyRotation(float start, float end, ImageView image1, ImageView image2, boolean isFirstImage) :
    Sets the needed attributes for a 3D flip animation to be performed, utilizes the following classes: DisplayNextView, Rotate3dAnimation, SwapViews
void initializeGameboard(ReversiRules.reversi newBoard):
    Initializes the Gameboard by placing the black and white piece images that correspond to the initial game state as defined in ReversiRules.java
void updateGameboard():
    Re-displays the images for the current state of the board contained in ReversiRules.reversi class
void updateCurrentScore(int white, int black):
    Replaces the TextViews containing the scores for both white and black with the values passed into this function
void updateStatusText(String statusText):
    Updates the TextView that prints with any string placed into the function as a String argument
void gameOver(String endStatus):
    Updates the TextView that prints with any string placed into the function as a String argument and renders all Gameboard buttons and the undo and redo buttons un-clickable
// AI related functions
void aiTurn():
    Is called after every user's move and based on the settings of the GameChoice Boolean variables, calls the appropriate AI functions
void randomMoveAi():
    Implements the random AI move functions from ReversiRules.java
void easyMoveAi():
    Implements the easy AI move functions from ReversiRules.java
void mediumMoveAi():
    Implements the medium AI move functions from ReversiRules.java
void hardMoveAi():
    Implements the hard AI move functions from ReversiRules.java

Animation Classes (from Android Open Source Project):

DisplayNextView.java
void onAnimationStart(Animation animation):
    An empty function that can be implemented if any actions is desired when an animation starts

void onAnimationEnd(Animation animation):

Handles the case when an image has been turned 90 degrees and needs to be flipped back but with a different image

void onAnimationRepeat(Animation animation):

An empty function that can be implemented if any actions is desired when an animation repeats

SwapViews.java

void run():

Runs the same rotation animation on the second of two images after the first has already been rotated

Rotate3dAnimation.java

void initialize(int width, int height, int parentWidth, int parentHeight):

Initializes the Rotate3dAnimation variables

void applyTransformation(float interpolatedTime, Transformation t):

Utilizes the Android camera class to rotate the camera angle around an image to simulate rotation

Game Engine Logic: from Project 2

ReversiRules.java

<u>*Board state functions:*</u>

void initializeBoard():

Initializes the board to the appropriate starting position of a Reversi game (2 blacks, 2 whites) and allocates memory to the entire 3D array for all possible 61 board states (from 4 pieces to a full board)

void fiftyprintlns():

A simple function printing 50 empty lines in between each board state to make the UI more visually appealing

void printBoard():

Prints the current state of the board

void printValidMoves(Piece player):

Places the marker ‘:’ in every board spot that is a valid move for the current player and prints the board and subsequently removes those markers from the board

void printMenu():

Prints the menu of instructions on how to play the game to the console

void copyState():

Copies the current state of the board and places that copy onto the next board state in memory prior to each user’s move

void copyState(Piece[][] fromState, Piece[][] toState):

Copies the current state of one 2D Piece array called fromState to another called toState

boolean checkNeighbors(int column, int row, Piece player, boolean next):

Returns true if the location that the user is attempting to place a piece on has next to it (on the north, south, east, and west side) an opponent’s piece or sequence of opponents pieces bounded at the end by another one of the current player’s pieces; Applies these checks to the global 3D board array at position currentMove

boolean checkDiagonals(int column, int row, Piece player, boolean next):

Returns true if the location that the user is attempting to place a piece on has next to it (on all diagonals: northwest, northeast, southwest, southeast side) an opponent’s piece or sequence of opponents pieces bounded at the end by another one of the current player’s pieces; Applies these checks to the global 3D board array at position currentMove boolean checkBranchNeighbors(int column, int row, Piece player, Piece[][] branch):

Returns true if the location that the user is attempting to place a piece on has next to it (on the north, south, east, and west side) an opponent's piece or sequence of opponents pieces bounded at the end by another one of the current player's pieces; Applies these checks to a specified 2D Piece array

boolean checkBranchDiagonals(int column, int row, Piece player, Piece[][] branch):

Returns true if the location that the user is attempting to place a piece on has next to it (on all diagonals: northwest, northeast, southwest, southeast side) an opponent's piece or sequence of opponents pieces bounded at the end by another one of the current player's pieces; Applies these checks to a specified 2D Piece array

boolean isValidMove(int column, int row, Piece player, boolean next):

Returns true if both checkNeighbors and checkDiagonals are true

boolean isBranchValidMove(int column, int row, Piece player, Piece[][] branch):

Returns true if both checkNeighbors and checkDiagonals are true on a board contained in Piece [][] branch

boolean isInBounds(int column, int row):

Returns true if the column and row specified are within the board

void flipPieces(int startCol, int endCol, int startRow, int endRow):

Flips all the pieces of the opponent of the current player in between one of the current player's pieces and the latest piece the current player has placed

void flipBranchPieces(int startCol, int endCol, int startRow, int endRow, Piece[][] branch):

Flips all the pieces of the opponent of the current player in between one of the current player's pieces and the latest piece the current player has placed on the Piece[][] branch board

void updateScore():

Counts the number of whites and black pieces on the board and updates the global variables storing those two values

void checkGameStatus(Piece player):

Checks at every turn if the current player has any valid moves and if the next player has any valid moves; if neither player has any valid moves, the function sets a global variable to indicate the end of the game

void checkWinner():

When the end of the game has been called, this function counts all the white and black pieces and prints out which player one and the final score

void updateScore():

Counts the number of whites and black pieces on the board and updates the global variables storing those two values

void checkGameStatus(Piece player):

Checks at every turn if the current player has any valid moves and if the next player has any valid moves; if neither player has any valid moves, the function sets a global variable to indicate the end of the game

void checkWinner():

When the end of the game has been called, this function counts all the white and black pieces and prints out which player one and the final score

<u>*User input:*</u>

void undo():

Allows the user to move back a board state to the previous board state void redo():

> *Allows the user to move forward a board state to the next board state provided that the user has performed an undo and hasn't moved*

void display(boolean on):

> *Turns the board display on and off*

void getInput(Piece player):

> *Prompts the user for an input and takes the input string from the user and handles all valid and invalid inputs*

void move(int column, int row, Piece player):

> *Places a piece on the board when it has been determined that the move the user is attempting is a valid one*

<u>*Program loop/AI Functions:*</u>

void gameLoop1():

> *A loop that takes turns getting user input from two players and making moves to the game board. It also checks for valid moves, checks to see if the game is over, and determines the winner.*

void gameLoop2():

> *A loop that takes turns getting user input and playing an easy-difficulty AI. It makes the corresponding moves to the game board. It also checks for valid moves, checks to see if the game is over, and determines the winner*

void gameLoop3():

> *A loop that takes turns getting user input and playing a medium-difficulty AI. It makes the corresponding moves to the game board. It also checks for valid moves, checks to see if the game is over, and determines the winner.*

void gameLoop4():

> *A loop that takes turns getting user input and playing a hard-difficulty AI. It makes the corresponding moves to the game board. It also checks for valid moves, checks to see if the game is over, and determines the winner.*

void gameLoop5():

> *A loop that takes turns getting user input and playing an AI that makes random moves. It makes the corresponding moves to the game board. It also checks for valid moves, checks to see if the game is over, and determines the winner.*

void gameStart():

> *Outputs the selection menu to the console and waits for the user to input a valid selection. The choices are 2 player local, 1 player easy AI, 1 player medium AI, 1 player hard AI, and 1 player random AI. After the user picks an option the user is prompted for the color of their piece.*

void aiRandomMove():

> *This AI randomly chooses one of the available moves on the board.*

void aiMostFlips():

> *The AI determines where to place the piece based on how many of the opponent's pieces will be flipped. The AI chooses the position where the largest number of opponent's pieces get flipped.*

void aiWorstMove():

> *The AI chooses the position where the smallest number of opponent's pieces get flipped.*

void aiHardMove(Piece player):

> *The AI chooses positions that limit the number of moves of the opponent.*

Tuple smartMove(Piece player):

> *A function for the hard-difficulty AI that returns the smartest move for it to make, using other functions to calculate game-winning factors; uses the recursive function to*

generate the linkedlist of branches that contain the best path on the second level of the valid move tree

void avoidCases(Piece player, int col, int row, int depth, Piece[][] currentBoard):

A function that checks for specific cases that the AI should avoid.

void recursive(Piece[][] Node, Piece player, int fullDepth, int currentDepth):

Performs the min/max depth-first search of a tree of boards beginning with Piece[][] Node; for each node within the tree valid moves are determined and pushed to a global linked list tracking the branch found with the fewest moves at every level; at every node it finds the moves that give the AI's opponent the fewest valid moves and pushes those values into another linked list

int numPiecesFlipped(int col, int row):

The number of pieces flipped is returned.

void playGame(int i):

playGame is used to determine which game loop to enter based on the input i.

void playAgain():

This function sends a prompt asking the player if they want to play again. If the user enters y the game returns to the main menu. If the user enters n the game exits, and the program terminates

void main():

Contains the main function which begins the program

<u>*High Scores Information Functions:*</u>

void updateScoresOnFile(boolean win):

Updates one of the eight values in the high scores XML file corresponding to the selected difficulty and if the player won or lost. Calls openScoresFile() and writeScoresFile() to edit the file on the SD card.

void openScoresFile():

Opens thescores.xml from the SD card then parses and stores all of the values into variables that can be used and changed by the program.

void writeScoresFile():

Saves the current state of the highScores variables and writes them to the file in thescores.xml on the SD card.

2. Post Production Notes

Individual Work Load Distribution:

Amongst our team members, Phillip Ells, William Guerra, and Daniel Tan, the division of labor of our group was very even overall: basically a 1/3 workload assigned to each individual (or 33%); not just for coding, but also for brainstorming, project designing, and project documentation

Design Thoughts and Considerations, Problems and Solutions, and Lessons Learned:

Difficulties:

One of the problems encountered in this project was the runtime on our hard-difficulty AI algorithm. It was simply too taxing for the android device and it takes a very long time to complete a move. We had minor difficulties as far as transferring the program to an Android application with a graphical user interface. Some of our code was just not built to handle that without modification. We had been prepared for problems like these, and all other problems. We were entirely aware that porting our application was going to require modification, and since none of our teammates had any Android experience, we were ready and willing to learn. They were simply resolved by learning more on the Android SDK and making adjustments to our code

What We Learned:

This was all of ours first ever experience programming an Android application, so I think it is safe to say we learned how to create everything in this app. From buttons to views to graphics, animations, and more; they were all learned through internet research and some help from the lab assignments.

3. Development Log

Team 6:
Phillip Ells
William Guerra
Daniel Tan

Development Log: "Each team should maintain a development log (Google doc) updated by the team members. Give access to the TA and the Instructor. This log will be graded. **There is no designated format.** We will check your daily progress."

6-25-12:

- Read the project 3 assignment
  - Establish goals for part 1
- SCRUM discussion
  - 1 sprint
- **SCRUM stories begun**

○ link play button to gameboard
○ add pop up menu
○ create basic gameboard layout
○ Create togglebutton
○ Project 3-Part 1: Design doc, SCRUM log, & XML design (for game statistics).

6-26-12:

- **SCRUM stories begun**

○ create splash screen
○ ReversiRules functions can affect gameboard activity
○ add buttons and pieces to gameboard xml
○ pieces can display a 3D flip animation
○ Complete Game selection art
○ complete Gameboard art
○ Complete Main menu art
○ Complete game mode menu
○ Complete main menu
○ Complete High scores art
○ Complete Win game graphics

- SCRUM **stories completed**

○ link play button to gameboard
○ add pop up menu
○ create basic gameboard layout
○ Create togglebutton

<u>6-27-12:</u>

- **SCRUM stories begun**

○ Gameboard tracks the current score of each player.
○ Implement redo
○ Implement undo
○ Implement AI
○ create options menu
○ link high scores page
○ create about page
○ Gameboard Menu Button will ask 'are you sure?'
○ New Game will ask 'are you sure'?
○ Gameboard can print out informative text at each game state

- **SCRUM stories completed**

○ Project 3-Part 1: Design doc, SCRUM log, & XML design (for game statistics).
○ create splash screen
○ ReversiRules functions can affect gameboard activity
○ add buttons and pieces to gameboard xml
○ pieces can display a 3D flip animation
○ Complete Game selection art
○ complete Gameboard art
○ Complete Main menu art
○ Complete game mode menu
○ Complete main menu

<u>Project 3-Part 1: Design doc, SCRUM log, & XML design (for game statistics) deliverables are submitted to eLearning *ahead of schedule*</u>

<u>6-28-12:</u>

- **SCRUM stories begun**

○ read/write test case to SD card

- **SCRUM stories completed**

○ Gameboard tracks the current score of each player.

○ Implement redo
○ Implement undo
○ Implement AI
○ create options menu
○ link high scores page
○ create about page

6-29-12:

- **SCRUM stories begun**

○ Player can start as black or white
○ Implement read and write to xml on SD

- **SCRUM stories completed**

○ read/write test case to SD card

6-30-12:

- **SCRUM stories begun**

○ finish about pages

- **SCRUM stories completed**

○ Complete High scores art
○ Player can start as black or white

7-01-12:

- **SCRUM stories begun**

○ Project 3 Final Deliverable

- **SCRUM stories completed**

○ Implement read and write to xml on SD
○ finish about pages

7-02-12:

- **SCRUM stories completed**

○ Complete Win game graphics
○ Gameboard Menu Button will ask 'are you sure?'
○ New Game will ask 'are you sure'?
○ Gameboard can print out informative text at each game state
○ Finish project 3 report
○ Project 3 Final Deliverable