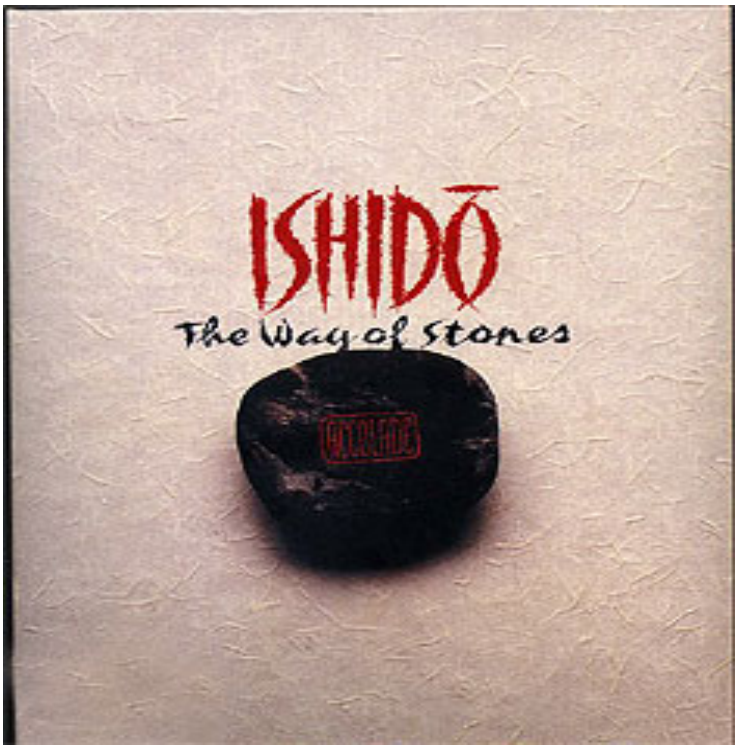
Ishido: The Way of Stones



A Senior Project Programmed for Android
By Austin Fouch
CMPS 450 Spring 2018
Ramapo College of New Jersey

Ishido: The Way of Stones
for Android

Table of Contents

Ishido: The Way of Stones
for Android

8. 1. Project Introduction

Ishido: The Way of Stones, is a puzzle video game initially released in 1990 by Accolade, a video game publisher. The game was developed by Publishing International and was designed by Michael Feinberg and programmed by Ian Gilman and Michael Sandige. The game was released on platforms such as Macintosh, MS-DOS, Sega Genesis, Atari Lynx and Game Boy. A physical version of Ishido was released by ASCII in Japan in 1991. Below is the "Legend of Ishido" that was released with the original copies of the game:

> "One misty spring morning in 1989, in the remote mountains of China's Han Shan province, a Mendicant monk of the Northern School of the White Crane branch of Taoism, walked silently out through the front gates of the Heavenly Peak Temple The monk carried a stone board, a set of seventy-two carved stone pieces, and an ancient scroll inscribed with brush and ink in elegant calligraphic script. He also carried with him a secret which had lain cloistered and hidden for thousands of years. "

The game itself is a puzzle board game consisting of 72 stones or tiles and a game board of 96 squares, setup as 8 rows squares in 12 columns. Each Ishido tile has two attributes: a color and symbol. There are 6 unique colors and 6 unique symbols in a set of Ishido tiles. Thus, there are 36 unique tile combinations. Each tile comes in pairs, so overall, a game of Ishido has 72 tiles. The primary objective of the game is place all 72 tiles on the game board. The challenge comes from the fact that a tile can only be placed on a square that is adjacent to a matching tile. A matching tile is any tile that has the same color or symbol as the tile being played. In this iteration of Ishido, placing a tile next to 1 matching tile awards 1 point, 2 matching tiles awards 2 points, 3 matching tiles awards 3 points, and 4

matching tiles awards double points resulting in 8 points. As the board continues to fill up, the importance of playing 4-way matches becomes paramount.

This iteration of Ishido comes with two game modes: Solitaire and Standard. Regardless of game mode, a game of Ishido starts the same way:

1. A deck of 72 tiles is created. Each unique combination of the 6 colors and 6 symbols results in 36 unique tiles. These 36 tiles are then duplicated, resulting in the deck of 72 tiles.
2. This deck is then shuffled randomly.
3. 6 tiles are then taken from the deck and placed in a setup deck. This setup deck must have a tile representing each color and symbol exactly once.
4. The tiles from the setup deck are then placed onto the game board in specific positions:
    i. Row 1, Column 1
    ii. Row 1, Column 12
    iii. Row 4, Column 6
    iv. Row 5, Column 7
    v. Row 8, Column 1
    vi. Row 8, Column 12
5. After the setup tiles are placed, the current tile being played is then drawn from the game's deck.
6. Now the game is ready to accept the user's first move, requiring the user to play the game's current tile on the game's initial board setup.

The solitaire game mode allows the user to play the game of Ishido alone, where the goal is simply to score as many points as possible given the board's initial setup of 6 tiles, the tile being played, and the 65 remaining tiles.

In the standard game mode, the user plays against the computer player. The user will always be awarded the first move of the game, and the computer player will play after each of the user's moves. Both the user and the computer are playing the game with the same game board, the same initial setup, and the same deck. The only difference between the two players' moves is the tile they are allowed to play.

In both game modes, the user is faced with only one restriction: A tile can only be placed on a square that is adjacent to a tile that matches in color or symbol. Essentially, the user attempts to play a tile that will award them 0 points, the move is deemed illegal and cannot be made. Following this simple rule, the game proceeds until one of two exit conditions is met:

1. The current tile cannot be played on the current game board in move that results points greater than 0.
2. The current tile was played and the deck is empty.

If either of these conditions are met, the game is over, the user is returned to the title activity, and the winner's score is announced. If playing solitaire, the user's score is announced.

A web version of Ishido: The Way of Stones was released by Andrew Birrell online at this curl: http://birrell.org/andrew/ishido/. Ian Gilman, one of the original programmers for the original 1990 version of Ishido released on Macintosh, MS-DOS, etc., released a user manual online at this URL: http://www.iangilman.com/software/ishido_manual.txt.

2. Installation Instructions

There are two ways install and use this project:

1. Android Emulator
2. Android Device

In both cases, the user must install Android Studio onto a Windows machine, preferably running a Windows 7 or newer operating system. Android Studio for Windows can be downloaded from this URL: https://developer.android.com/studio/install.

This version of Ishido can be downloaded from this URL: https://github.com/austinfouch/Ishido-Android. Once downloaded, open Android Studio and select File --> Open, browse to the location where Ishido-Android was downloaded, and select. Alternatively, the project can be imported entirely through GIT by selecting File --> New --> Project from Version Control --> GIT. Copy the GitHub link from above into the URL field, and select a directory to install the project to.

Once the project is imported into Android Studio it can be ran via the Android Emulator or an Android Device.

If the Android Emulator is chosen, the user's machine must be able to handle virtualization. For more information regarding the Android Emulator and virtualization, navigate to this URL: https://developer.android.com/studio/run/emulator. If the user's machine support virtualization, then click the "Run" button in Android Studio with this project open, elect the Android Emulator under "Available Virtual Devices", and click "OK".

If an Android Device is chosen, the user must connect their Android device to their Windows machine running Android Studio. This connection must be done via USB. With this project open in Android Studio, the user must click the "Run" button, select their Android Device under "Connected Devices", and click "OK".

3. User Manual

Once the user is able to successfully install Android Studio, import the project, and run it via the Android Emulator or on an Android Device, the game is ready to be played.

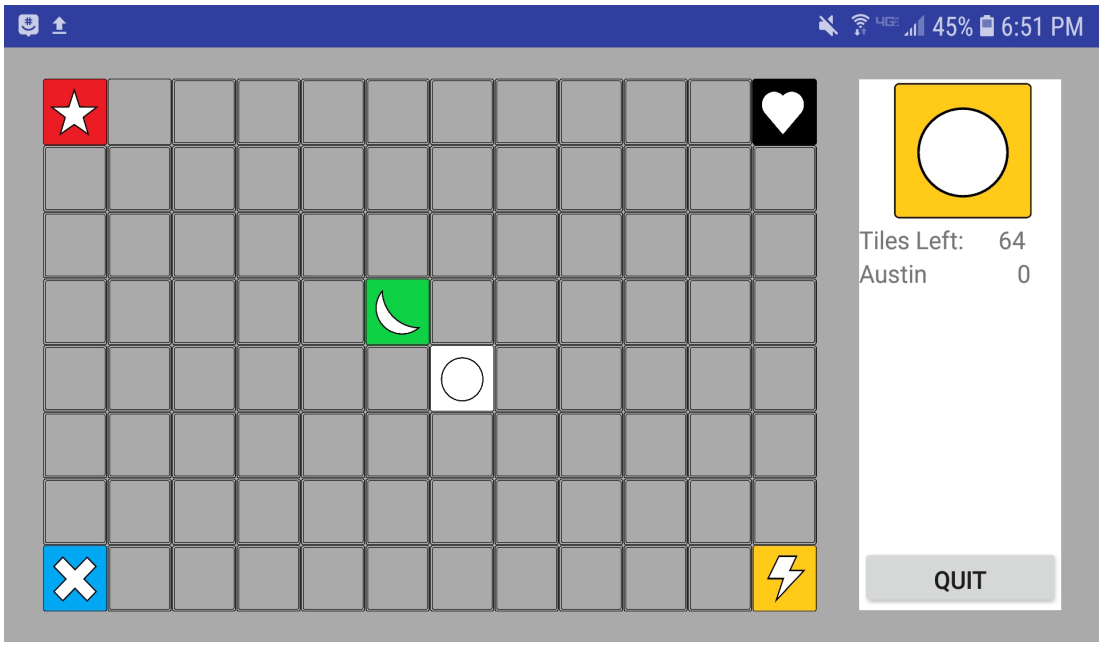
Beginning Play

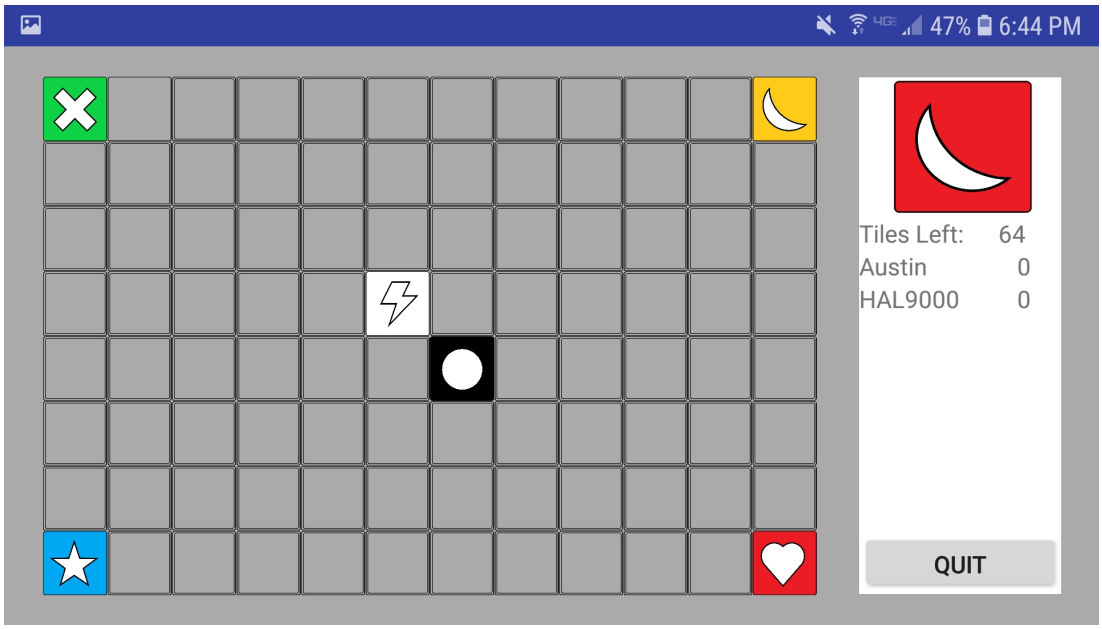
The first screen the user is brought to is called the Launcher Activity:

The Launcher Activity has two buttons: "Solitaire" and "Standard".

Clicking "Solitaire" will navigate the user to a new screen, the Game Activity, and start a new game of Solitaire Ishido:

Clicking "Standard" in the Launcher Activity will also navigate the user to the Game Activity, but will instead start a new game of Ishido where the user plays against the computer: HAL 9000. HAL is a formidable opponent, but his "points-now" mindset is no match for an advanced Ishido player who is capable of thinking multiple moves ahead.





Ishido: The Way of Stones

for Android

Rules of Play

From the first move this ancient game and beautiful puzzle will call upon your deepest powers of strategy and concentration as you match 72 stones on a board of 96 squares. At each turn, one tile from the deck is displayed as the current tile. Every tile has two attributes: a symbol and a background color. The user and HAL will try to place each stone on the board so that its color/pattern or symbol matches a tile next to it. The user and HAL then continue to place tiles until no more legal matches are possible or until the deck is empty.

Placing Tiles

As a user, the only way to attempt to play the current tile onto the game board is to click an open square on the board. Once an open board square is clicked, the user will notice a prompt open in the bottom right corner of their Android device/emulator screen. This prompt will detail to the user the row and column of the board that they are attempting to play the current tile on:

If the user clicks "Yes" on this prompt, the current tile will be played and their score will be updated to reflect the points gained by placing the current tile on the position they clicked. If the play the user is making would result in 0 points gained, the play is rendered illegal and the user will be prompted to make a different play.

If the user click "No" on this prompt, the user will be returned to the Game Activity and given another opportunity to make a legal play.

There is no way to move a tile once it has been place. There is no need to try and click or drag the current tile onto the board; the current tile will always be the tile the user is attempting to play, so clicking an open square on the board will always attempt to play the current tile.

Matching Stones

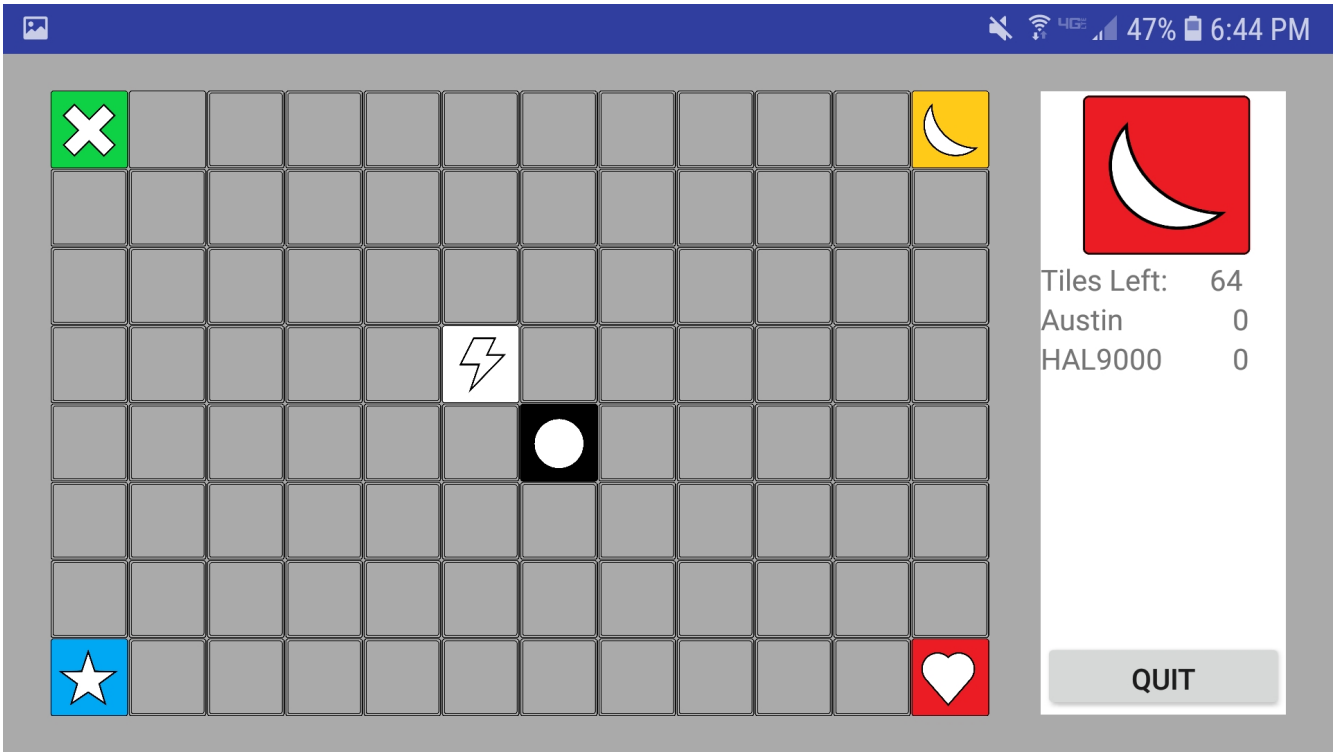
In order for a tile to be placed, it must match with an adjacent tile. An adjacent tile constitutes any tile on the game board square that is above, below, to the left, or to the right of the game board square the user is attempting to play on. A match constitutes of a single tile matching with two or more adjacent tiles in color or symbol. A legal tile play is any play where the current tile matches with at least one adjacent tile.

Gaining Points

Points are awarded to the user or HAL when either plays a tile that matches with at least one adjacent tile on the board. For one match, one point is awarded. For two matches, two points are awarded. For three matches, three points are awarded. For four matches, the player is awarded eight points.

Understanding the Game Activity

Below is a picture of a Standard Ishido game at setup:



The "grid" which takes up most of the screen is the game board. Outside of one button, this game board is the only interaction the user makes on this screen. Any open square, i.e. a game board square with no tile placed on it, can be interacted with by clicking once. Upon clicking, the user will be prompted to confirm their tile placement, as described in the "Placing Tiles" section of this manual.

The other portion of the screen is the "panel" to the right. This panel, from top to bottom, details the following:

1. The current tile available to the user for playing. This is simply the tile that was just drawn from the game's deck and is ready to be played.
2. The tiles left in the game's deck.

3. The user's name and their score. The user's name is shown on the left, and their score, the sum of all of the points gained from their plays, is shown to the right of their name.
4. The computer's name and score (if playing in standard mode). The computer's name is shown on the left, and their score, the sum of all of the points gained from their plays, is shown to the right of their name.
5. The user's last move made. Details the tile the user played, the location they played it, and the points gained from the play.
6. The computer's last move made (if playing in standard mode). Details the tile the computer played, the location the computer played it, and the points the computer gained from the play.
7. The "Quit" button. Will prompt the user to confirm they wish to exit the game. If they click "Yes" the game will exit and return to the Launcher Activity. If they click "No" the game will return normal.

4. Design of Project

Design Summary

The project was programmed in a Windows 10 environment, using Android Studio. Both the Launcher and Game Activities were designed using Android Studio and XML. All external resources used, such as the images for the tiles, were designed and implemented by this project's author, Austin Fouch. The programming language used to write all logic and interactivity source code within this project was Java Android. The Android Platform API targeted while programming this project was Android API 27. The device which both the Launcher and Game Activities were designed around was Google and LG's Nexus 4. All testing of this project was done using a Samsung Galaxy S7 running Android 8.0 OS.

Classes

Tile Class

Member Variables

- IshidoColor m_color: enumerated value representing the color of the tile.
- IshidoSymbol m_symbol: enumerated value representing the symbol of the tile.

Member Functions

- Tile::Tile(): default constructor for the Tile class.
- Tile::Tile(IshidoColor, IshidoSymbol): copy constructor for the Tile class.
- void Tile::setColor(IshidoColor): setter for the m_color member variable.
- void Tile::setSymbol(IshidoSymbol): setter for the m_symbol member variable
- IshidoColor Tile::getColor(): getter for the m_color member variable.

- IshidoSymbol Tile::getSymbol(): getter for the m_symbol member variable.
- boolean Tile::isMatch(Tile): determines if the passed tile matches this tile.
- String Tile::getColorResourceStr(): returns the resource string associated with the color of this tile.
- String Tile::getSymbolResourceStr(): returns the resource string associated with the symbol of this tile.

Deck Class

Member Variables

- Vector<Tile> m_tiles: vector of Tile objects representing the deck for an Ishido game.

Member Functions

- Deck::Deck(): default constructor.
- Deck::Deck(Vector<Tile>): copy constructor.
- Vector<Tile> Deck::getTiles(): getter for the m_tiles member variable.
- void Deck::setTiles(Vector<Tile>): setter for the m_tiles member variable.
- void Deck::pop(): removes the last Tile from m_tiles.
- Tile Deck::top(): returns the last Tile from m_tiles.
- Tile Deck::push(Tile): adds a Tile to the end of m_tiles.
- void Deck::setup(): creates a deck ready for the start of an Ishido game.
- Vector<Tile> getSetupTiles(): returns a vector of 6 tiles that will be placed on the start of a game of Ishido.

Board Class

Member Variables

- Vector<Vector<Tile>> m_tiles: vector of vectors of Tile objects, representing the game board of Ishido.

Member Functions

- Board::Board(): default constructor. Initializes the board to be filled with blank tiles.
- Board::Board(Vector<Vector<Tile>>): copy constructor.
- Vector<Vector<Tile>> Board::getTiles(): returns the m_tiles member variable.
- void Board::setTiles(Vector<Vector<Tile>>): sets m_tiles to the given vector of vectors of Tile objects.
- Tile Board::getTile(Integer, Integer): given integer values for row and column, returns the Tile object from that position.
- void Board::setTile(Integer, Integer, Tile): given integer values for row and column, and a Tile object, sets the Tile object at the given position to the given Tile object.

Player Class

Member Variables

- Integer m_score: integer value representing the player's current score.
- String m_name: string value representing the player's name;

Member Functions

- Player::Player(): default constructor.
- Player::Player(String, Integer): copy constructor.
- void Player::setScore(Integer): setter for the m_score member variable.

- Integer Player::getScore(): getter for the m_score member variable.
- void Player::setName(): setter for the m_name member variable.
- String Player::getName(): getter for the m_name member variable.
- Integer isLegalPlay(Tile, Board, int, int): given a Tile object, calculates the points scored when placing the Tile on the given Board object at the given row and column values. If the return value is greater than 0, it is a legal play.

Human Class extends Player Class

Member Variables

- Inherits Player.m_name and Player.m_score member variables.

Member Functions

- Inherits default and copy constructors from Player.
- Inherits member variable getters and setters from Player.
- Inherits Integer Player::isLegalPlay(Tile, Board, int, int) from Player.

Computer Class extends Player Class

Member Variables

- Inherits Player.m_name and Player.m_score member variables.

Member Functions

- Inherits default and copy constructors from Player.
- Inherits member variable getters and setters from Player.
- Inherits Integer Player::isLegalPlay(Tile, Board, int, int) from Player.

- Turn Player::play(Tile, Board): given a Tile and Board object, determine the most valuable play currently on the board and return the Turn associated with that play.

ActivityLog Class

Member Variables

- String m_playerOneTurn: string value representing the Human user's turn.
- String m_playerTwoTurn: string value representing the Computer's turn.

Member Functions

- String ActivityLog::getPlayerOneTurn(): getter for the m_playerOneTurn member variable.
- void ActivityLog::setPlayerOneTurn(String): setter for the m_playerOneTurn member variable.
- String ActivityLog::getPlayerTwoTurn(): getter for the m_playerTwoTurn member variable.
- void ActivityLog::setPlayerTwoTurn(Turn): setter for the m_playerTwoTurn member variable.

Turn

Member Variables

- Tile m_tilePlayed: Tile object representing the Tile played on the turn.
- Integer m_rowPlayed: value representing the row where the Tile was played.
- Integer m_colPlayed: value representing the column where the Tile was played.
- String m_playerName: value representing the name of the Player making the play.

- Integer m_pointsScored: value representing the score gained on the play.

Member Functions

- Turn::Turn(): default constructor for the Turn class.
- Turn::Turn(Tile, Int, Int, String, Int): copy constructor for the Turn class.
- Int Turn::getPointsScored(): getter for the m_pointsScored member variable.
- Void Turn::setPointsScored(Int): setter for the m_pointsScored member variable.
- Tile Turn::getTilePlayed():: getter for the m_tilePlayed member variable.
- Void Turn::setTilePlayed(Tile): setter for the m_tilePlayed member variable.
- Int Turn::getColPlayed(): getter for the m_colPlayed member variable.
- Void Turn::setColPlayed(Int): setter for the m_colPlayed member variable.
- Int Turn::getRowPlayed(): getter for the m_rowPlayed member variable.
- Void Turn::setRowPlayed(Int): setter for the m_rowPlayed member variable.
- String Turn::getPlayerName(): getter for the m_playerName member variable.
- Void Turn::setPlayerName(String): setter for the m_playerName member variable.
- Int Turn::getPointsScored(): getter for the m_pointsScored member variable.
- Void Turn::setPointsScored(): setter for the m_pointsScored member variable.

IshidoColor Class

Member Variables

- enum IshidoColor: public member variable of enumerated values representing the possible color options for a Tile object's color member variable.

IshidoSymbol Class

Member Variables

- enum IshidoSymbol: public member variable of enumerated values representing the possible symbol options for a Tile object's symbol member variable.

IshidoConstants Class

Member Variables

- int DECK_SIZE: public final static integer value which represents the starting deck size for a game of Ishido, i.e. 72.
- int NUM_BOARD_ROWS: public final static integer value which represents the number of rows on a board for a game of Ishido, i.e. 8.
- int NUM_BOARD_COLS: public final static integer value which represents the number of rows on a board for a game of Ishido, i.e. 12.
- int UNQ_TILE_COUNT: public final static integer value which represents the number of duplicates for each unique tile, i.e. 2.

Activities

Launcher Activity

Buttons/Layouts/Views

- solitaireButton (Button): onClick calls Launcher::PlaySolitaire().
- standardButton (Button): onClick calls Launcher::Play()

Functions

- void Launcher::onCreate(Bundle): creates the activity and initializes layout.

- void Launcher::Play(View): launches Game Activity as a standard Ishido game with one Human player, the user, and one Computer player.
- void Launcher::PlaySolitaire(View): launches Game Activity as a solitaire game with one Human player, the user.

Game Activity

Member Variables

- Game m_game:
- TableLayout m_boardLayout: TableLayout with 8 TableRow layouts which contain 12 ImageViews.
- TextView m_tileCountLayout: TextView which has its text field set to the Tile objects left in m_game.getDeck()
- TextView m_player1NameLayout: TextView which has its text field set to the name of the Human player.
- TextView m_player1ScoreLayout: TextView which has its text field set to the score of the Human player.
- TextView m_player2NameLayout :TextView which has its text field set to the name of the Computer player.
- TextView m_player2ScoreLayout: TextView which has its text field set to the score of the Computer player.
- Boolean m_solitaireFlag: true if the game is solitaire, false otherwise.

Buttons/Layouts/Views

- boardLayout (TableLayout) (TableRow) (ImageView)
- currentTileView (ImageView)
- quitButton (Button)
- playerLabel (TextView)
- scoreView (TextView)
- playerLabel2 (TextView)
- scoreView2 (TextView)
- Tile Count (TextView)
- player1Turn (TextView)
- player2Turn (TextView)

Functions

- void GameActivity::quitGame(): exits the game and returns to Launcher Activity.
- Game GameActivity::getGame(): getter for the m_game member variable.
- Void GameActivity::setGame(Game): setter for the m_game member variable.
- TableLayout GameActivity::getBoardLayout(): getter for the m_boardLayout member variable.
- Void GameActivity::setBoardLayout(TableLayout): setter for the m_boardLayout member variable.
- ImageView GameActivity::getCurrTileLayout(): getter for the m_currTileLayout member variable.

- Void GameActivity::setCurrTileLayout(ImageView): setter for the m_currTileLayout member variable.
- TextView GameActivity::getTileCountLayout(): getter for the m_tileCountLayout member variable.
- Void GameActivity::setTileCountLayout(): setter for the m_tileCountLayout member variable.
- TextView GameActivity::getPlayer1NameLayout(): getter for the m_player1NameLayout member variable.
- Void GameActivity::setPlayer1NameLayout(TextView): setter for the m_player1NameLayout member variable.
- TextView GameActivity::getPlayer2NameLayout(): getter for the m_player2NameLayout member variable.
- Void GameActivity::setPlayer2NameLayout(TextView): setter for the m_player2NameLayout member variable.
- TextView GameActivity::getPlayer1ScoreLayout(): getter for the m_player1ScoreLayout member variable.
- Void GameActivity::setPlayer1ScoreLayout(TextView): setter for the m_player1ScoreLayout member variable.
- TextView GameActivity::getPlayer2ScoreLayout(): getter for the m_player2ScoreLayout member variable.

- Void GameActivity::setPlayer2ScoreLayout(TextView): setter for the m_player2ScoreLayout member variable.

Data Structures

The only standard data structures used in this project were Java Vectors. Each vector object was used to store Tile objects, such as Deck.m_tiles and Deck.getSetupTiles(). The Board class uses a vector of vectors of Tile objects, Board.m_tiles, to store the game board as rows and columns.

Programmer Defined Data Structures

Tile

Has variables for storing the color and symbol of a tile.

Deck

Has one member variable, a vector of Tile objects, and the class acts as a stack with functions such as Deck.pop(), Deck.push(), and Deck.top().

Board

Acts as a 2D vector of Tile objects. Tiles are accessed by row and column, where each row is a vector of tiles, and each column is a position in those vectors.

Game

Acts as the only data structure used in the Game Activity to set the user interface and draw the game. Has member variables for each aspect of a game of Ishido, such as Game.m_board, Game.m_deck, etc.

Player

Is an abstract data structure that is only used when inherited by the Human or Computer classes. Inherited attributes include Player.m_name and Player.m_score as well as Player::isLegalPlay().

Human

Inherits from the Player structure.

Computer

Inherits from the Player structure and adds a new function, Computer::Play(), which helps the computer decide the most optimal move on the board.

Turn

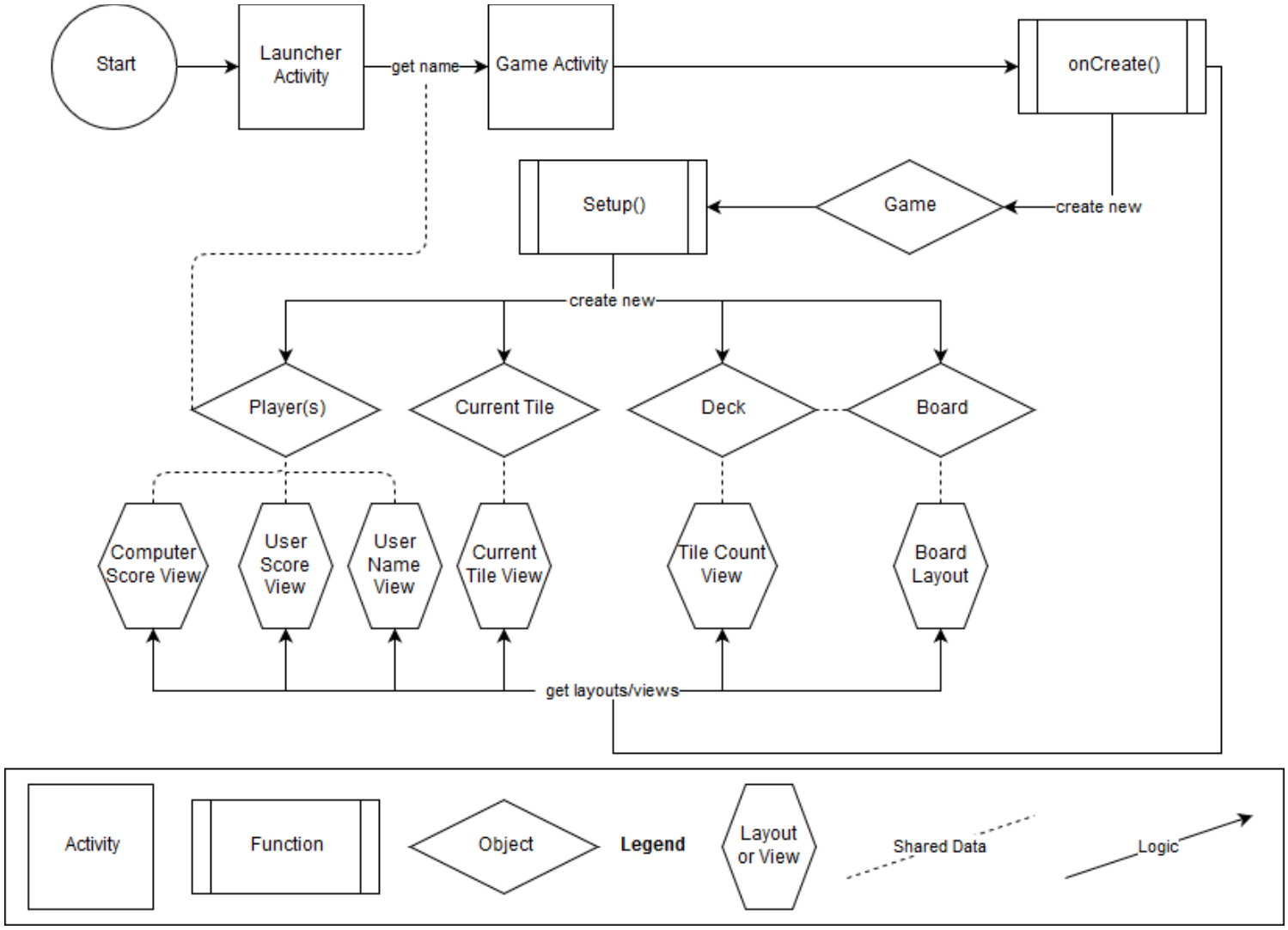
Basic data structure holding data representing a completed Ishido turn. This data includes the tile played, the row and column it was played, the player's name who played it, and the points score by the play.

Ishido: The Way of Stones

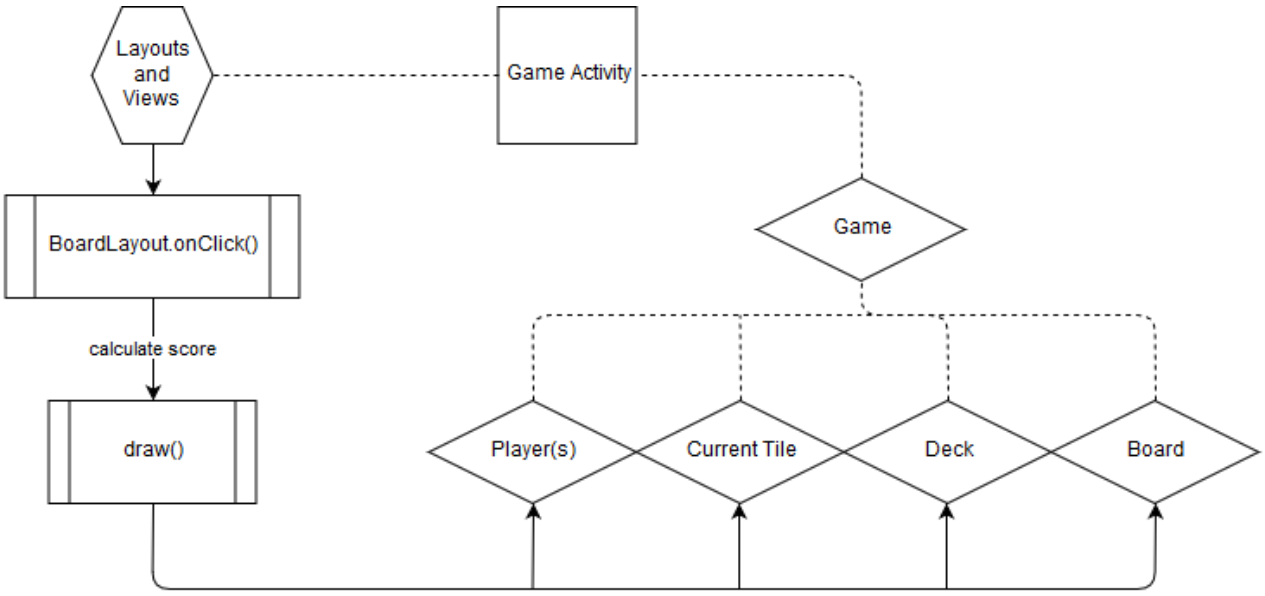
for Android

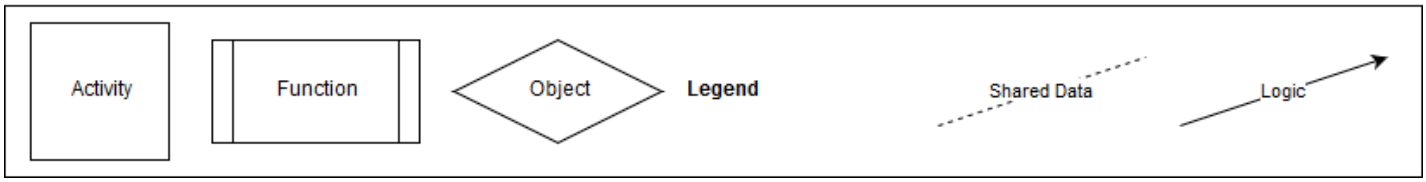
Flow Charts

Game Activity Setup Flow Chart



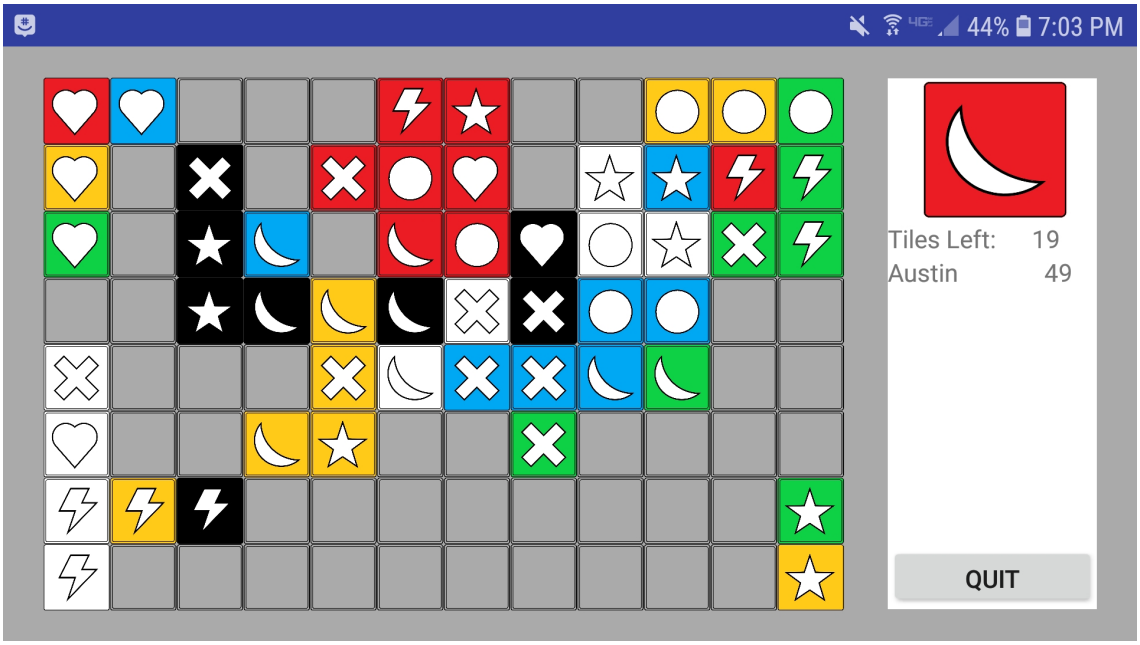
Game Activity UI Flow Chart



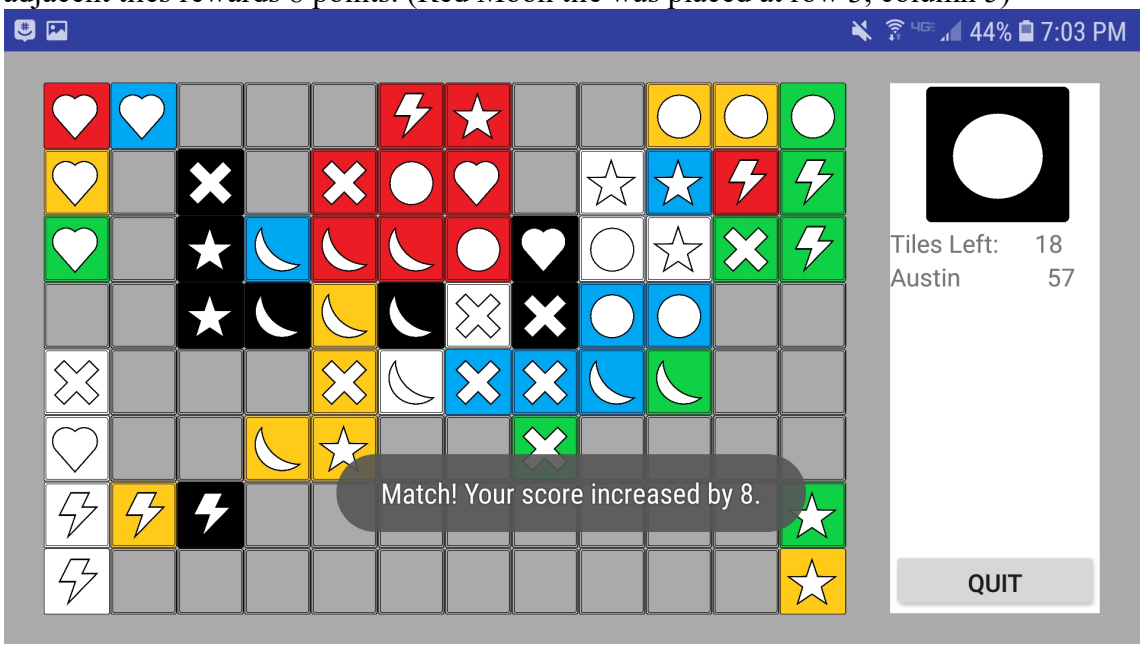


5. Testing

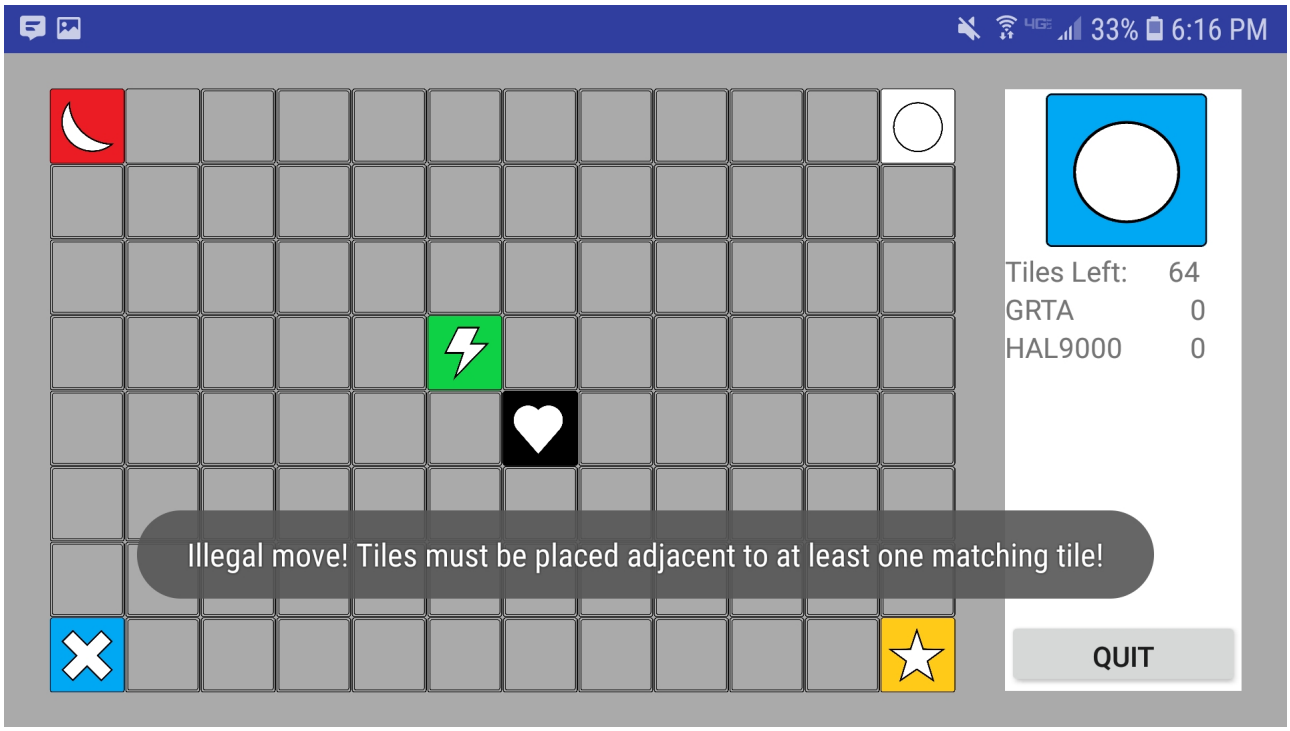
1. **4-way matching**: in order to test for a 4-way match to generate on the correct conditions and result in an 8 points play, the solitaire game mode was used:



The result worked as intended; playing a tile that matches in color or symbol with 4 adjacent tiles rewards 8 points. (Red Moon tile was placed at row 3, column 5)

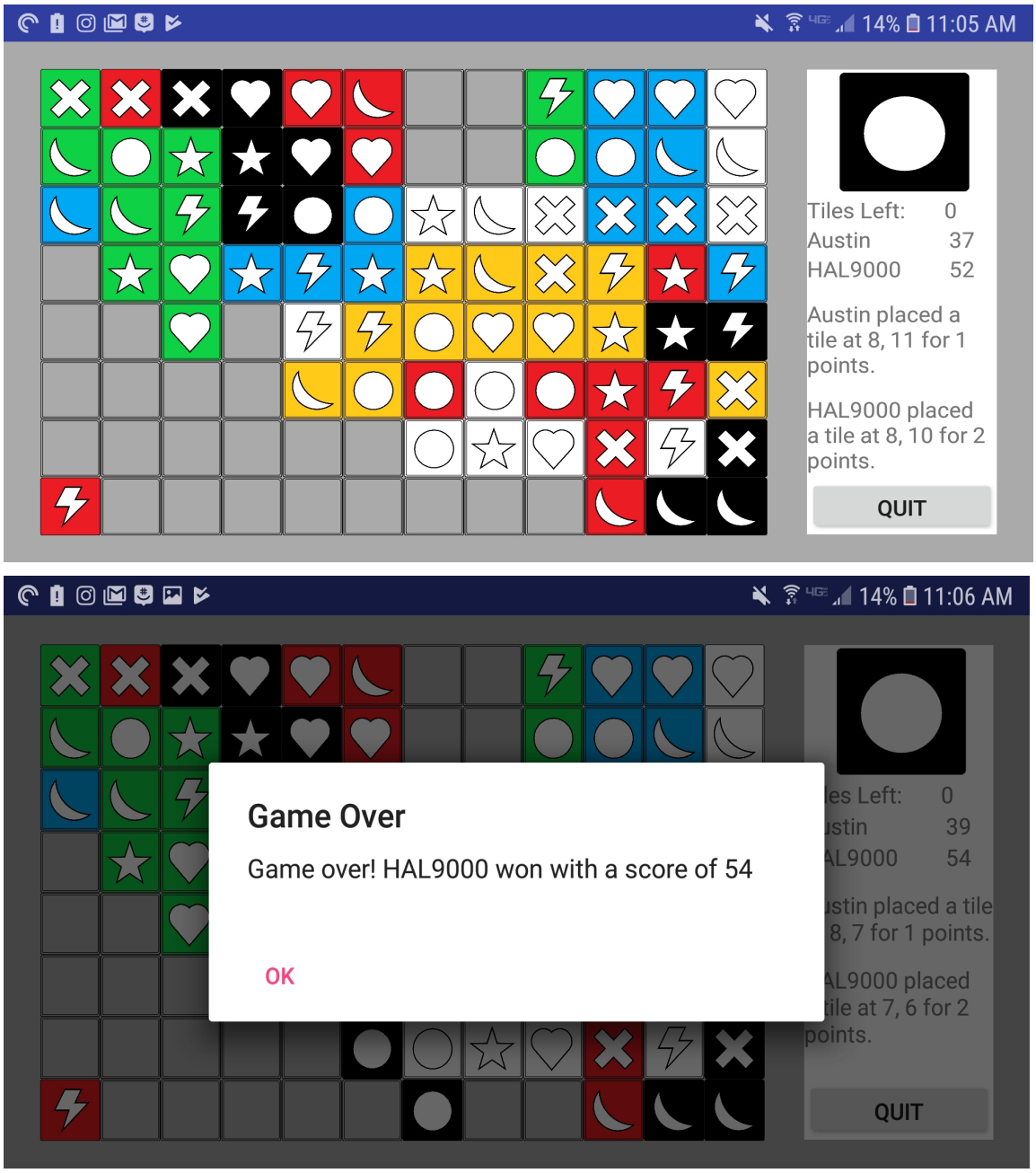


2. **Illegal mov**e: in order to stop the user from making an illegal move, the Tile being played was compared for legality with the tiles adjacent to it. The result worked as intended. In the screenshot below, the user attempted to play the Blue Circle tile at row 1, column 2 (adjacent to the non-matching, Red Moon tile)



3. **Exit condition:** once the deck is empty and the current tile is played, the game must exit and announce the winner:

Ishido: The Way of Stones
for Android



The screenshots above shows a game that is on its last move and the results of the test: the winning player announced and the game exits.

6. Summary and Conclusion

Summary

This project implements the seemingly-simple puzzle board game Ishido: The Way of Stones into an Android application, capable of being played through an Android emulator or device. The application offers two game modes: Solitaire and Standard. In Solitaire, the user plays against his or her self, having the ability to set up many 4-way matches if played correctly. In Standard, the user plays against the Ishido computer, HAL9000. HAL utilizes a brute force algorithm in order to make his moves. Although HAL can be a formidable opponent, any Human user can best him easily by focusing on multi-point plays while remembering to avoid setting up HAL for his own multi-point moves.

The user is presented with the same game setup regardless of game mode. There is the game board and the information panel. The game board contains all 96 of the squares of a traditional Ishido game board. Each square can hold exactly one tile. A tile is simply an object that has a color and a symbol. The user places the current tile, shown in the information panel, on any legal square on the game board. A legal square is a square where at least one adjacent tile matches with tile being played. A match is when two tiles adjacent to one another share a symbol or color. The goal of Solitaire is to score as many points as possible given the game board of 96 squares and a deck of 72 tiles. The goal of Standard is to try and beat HAL9000.

If a user attempts to play a tile that will result in an illegal play, they will be notified and the play will not go through. Before the user completes their turn, they will be prompted to confirm their tile placement.

The only exit condition for this iteration of Ishido is when the deck is empty and the current tile is played, as shown in the testing section of this manual.

The project was programmed in Java Android using the Android Studio IDE to both write the source code and design the XML layout files. The project was programmed in a Windows 10 environment, targeting the Android 27 API on a LG/Google Nexus 4. All testing and demonstrations were done on a Samsung Galaxy S9 running Android 8.0.

Conclusion

As the author of every aspect of this project, I can say that I am satisfied with the outcome. I took on a Java Android project to prepare me for designing large applications. By choosing to develop a game such as Ishido, I was able to experience developing an application that was exciting as the programmer as well as the user. Although this was only the second project I have implemented in Android, I was still able to accomplish most of what I set out to implement.

Some features that were on the initial wish-list, such as network play and serialization, were not implemented in this iteration of Ishido-Android. Another aspect of the project that can be improved upon is the computer's decision making when placing a tile. Currently, the algorithm is purely brute force. Another feature that could benefit the user would be a hint button, detailing legal plays on the board.

Overall, I am still satisfied with the project's outcome, even with the possibility for improvement. I will definitely come back to this project to optimize, add features, and more importantly, learn from the design choices I made here going forward in my career.

7. Biliography and Resources

Bibliography

1. "Ishido: The Way of Stones." *Wikipedia*, Wikimedia Foundation, 10 Sept. 2018, en.wikipedia.org/wiki/Ishido:_The_Way_of_Stones.
2. Gilman, Ian. "Ishido Manual." *Ian Gilman*, iangilman.com/software/ishido_manual.txt.

Resources

Flowchart software:

https://www.draw.io/

Android Studio Installation Guide:

https://developer.android.com/studio/install

Project GIT Repository:

https://github.com/austinfouch/Ishido-Android

Using the Android Emulator:

https://developer.android.com/studio/run/emulator.

Ishido: The Way of Stones web application by Andrew Birrell:

http://birrell.org/andrew/ishido/

Ishido: The Way of Stones 1990 User Manual by Ian Gilman:

http://www.iangilman.com/software/ishido_manual.txt

8. Source Code

```java
package austinfouch.com.ishido;

import java.util.Vector;

/**/
/*
    ActivityLog.java

    AUTHOR

        Austin Fouch

    DESCRIPTION

        ActivityLog class. Holds data representing the last Turn's taken by both the Human and
        Computer players. If Solitaire, the Computer's data will always be "".

    DATE

        01/30/2018

*/
/**/
public class ActivityLog
{
    private String m_playerOneTurn;
    private String m_playerTwoTurn;

    /**/
    /*
    ActivityLog::ActivityLog()

    NAME

            ActivityLog::ActivityLog - constructor for the ActivityLog class.

    SYNOPSIS

            public ActivityLog::ActivityLog();

    DESCRIPTION

            This function will construct an ActivityLog object. The member variables of the object
            will represent the last turn's taken by each player.

    RETURNS

            No return value.

    AUTHOR

            Austin Fouch

    DATE

            1/30/2018

    */
    /**/
    public ActivityLog()
    {
        m_playerOneTurn = new String();
        m_playerTwoTurn = new String();
    }

    /**/
    /*
```

```
68      ActivityLog::ActivityLog()
69
70      NAME
71
72              ActivityLog::ActivityLog - copy constructor for the ActivityLog class.
73
74      SYNOPSIS
75
76              public ActivityLog::ActivityLog(String a_playerOneTurn, String
                a_playerTwoTurn);
77                  a_playerOneTurn --> value to set m_playerOneTurn to.
78                  a_playerTwoTurn --> value to set m_playerTwoTurn to.
79
80      DESCRIPTION
81
82              This function will construct an ActivityLog object given data representing
                the last
83              turns taken by the two players.
84
85      RETURNS
86
87              No return value.
88
89      AUTHOR
90
91              Austin Fouch
92
93      DATE
94
95              1/30/2018
96
97      */
98      /**/
99      public ActivityLog(String a_playerOneTurn, String a_playerTwoTurn)
100     {
101         m_playerOneTurn = a_playerOneTurn;
102         m_playerTwoTurn = a_playerTwoTurn;
103     }
104
105     /**/
106     /*
107     ActivityLog::getPlayerOneTurn()
108
109     NAME
110
111             ActivityLog::getPlayerOneTurn - getter for the m_playerOneTurn member
                variable.
112
113     SYNOPSIS
114
115             public String ActivityLog::getPlayerOneTurn();
116
117     DESCRIPTION
118
119             This function will return the m_playerOneTurn member variable.
120
121     RETURNS
122
123             String.
124
125     AUTHOR
126
127             Austin Fouch
128
129     DATE
130
131             1/30/2018
132
133     */
```

```
134         /**/
135         public String getPlayerOneTurn()
136         {
137             return m_playerOneTurn;
138         }
139 
140         /**/
141         /*
142         ActivityLog::setPlayerOneTurn()
143 
144         NAME
145 
146                 ActivityLog::setPlayerOneTurn - setter for the m_playerOneTurn member
                    variable.
147 
148         SYNOPSIS
149 
150                 public void ActivityLog::setPlayerOneTurn(String a_playerOneTurn);
151                     a_playerOneTurn --> value to set m_playerOneTurn to.
152 
153         DESCRIPTION
154 
155                 This function will set the m_playerOneTurn member variable to the given
                    String value.
156 
157         RETURNS
158 
159                 Void.
160 
161         AUTHOR
162 
163                 Austin Fouch
164 
165         DATE
166 
167                 1/30/2018
168 
169         */
170         /**/
171         public void setPlayerOneTurn(String a_playerOneTurn)
172         {
173             this.m_playerOneTurn = a_playerOneTurn;
174         }
175 
176         /**/
177         /*
178         ActivityLog::getPlayerTwoTurn()
179 
180         NAME
181 
182                 ActivityLog::getPlayerTwoTurn - getter for the m_playerTwoTurn member
                    variable.
183 
184         SYNOPSIS
185 
186                 public String ActivityLog::getPlayerTwoTurn();
187 
188         DESCRIPTION
189 
190                 This function will return the m_playerTwoTurn member variable.
191 
192         RETURNS
193 
194                 String.
195 
196         AUTHOR
197 
198                 Austin Fouch
199 
```

```
200       DATE
201
202               1/30/2018
203
204       */
205       /**/
206       public String getPlayerTwoTurn()
207       {
208           return m_playerTwoTurn;
209       }
210
211
212       /**/
213       /*
214       ActivityLog::setPlayerTwoTurn()
215
216       NAME
217
218               ActivityLog::setPlayerTwoTurn - setter for the m_playerTwoTurn member
                  variable.
219
220       SYNOPSIS
221
222               public void ActivityLog::setPlayerTwoTurn(String a_playerTwoTurn);
223                   a_playerTwoTurn --> value to set m_playerTwoTurn to.
224
225       DESCRIPTION
226
227               This function will set the m_playerTwoTurn member variable to the given
                  String value.
228
229       RETURNS
230
231               Void.
232
233       AUTHOR
234
235               Austin Fouch
236
237       DATE
238
239               1/30/2018
240
241       */
242       /**/
243       public void setPlayerTwoTurn(String a_playerTwoTurn)
244       {
245           this.m_playerTwoTurn = a_playerTwoTurn;
246       }
247   }
248
```

```
1   package austinfouch.com.ishido;
2
3   import java.util.Vector;
4
5   /**/
6   /*
7       Board.java
8
9       AUTHOR
10
11          Austin Fouch
12
13      DESCRIPTION
14
15          Board class. Hold information related to the board used in a standard game of
            Ishido.
16              Vector<Vector<Tile>> m_tiles --> 2D Vector of Tiles, simulating a game board.
17
18      DATE
19
20          01/30/2018
21
22  */
23  /**/
24  public class Board
25  {
26      private Vector<Vector<Tile>> m_tiles;
27
28      /**/
29      /*
30      Board::Board()
31
32      NAME
33
34              Board::Board - constructor for the Board class.
35
36      SYNOPSIS
37
38              public Board::Board();
39
40      DESCRIPTION
41
42              This function will construct a Board object. The only member variable of
                the constructed
43              Board object, m_tiles, is set to be a 2D Vector of Tiles filled with blank
                tiles.
44
45      RETURNS
46
47              No return value.
48
49      AUTHOR
50
51              Austin Fouch
52
53      DATE
54
55              1/30/2018
56
57      */
58      /**/
59      public Board()
60      {
61          m_tiles = new Vector<>();
62          for(int row = 0; row < IshidoConstants.NUM_BOARD_ROWS; row++)
63          {
64              Vector<Tile> tempTiles = new Vector<>();
65              for(int col = 0; col < IshidoConstants.NUM_BOARD_COLS; col++)
66              {
```

```
                Tile tempTile = new Tile(IshidoColor.BLANK, IshidoSymbol.BLANK);
                tempTiles.add(tempTile);
            }

            m_tiles.add(tempTiles);
        }
    }

    /**/
    /*
    Board::Board()

    NAME

            Board::Board - copy constructor for the Board class.

    SYNOPSIS

            public Board::Board(Vector<Vector<Tile>> a_tiles);
                a_tiles --> 2D Vector of tiles to set m_tiles to.

    DESCRIPTION

            This function will construct a Board object. The only member variable of
            the constructed
            Board object, m_tiles, is set to be a copy of a_tiles, a2D Vector of Tiles.

    RETURNS

            No return value.

    AUTHOR

            Austin Fouch

    DATE

            1/30/2018

    */
    /**/
    public Board(Vector<Vector<Tile>> a_tiles)
    {
        this.m_tiles = a_tiles;
    }

    /**/
    /*
    Board::getTiles()

    NAME

            Board::getTiles - getter for the Board's tiles.

    SYNOPSIS

            public Board::getTiles();

    DESCRIPTION

            This function returns the Board's member variable, m_tiles.

    RETURNS

            Vector<Vector<Tile>>.

    AUTHOR

            Austin Fouch
```

```
135
136         DATE
137
138                 1/30/2018
139
140         */
141         /**/
142         public Vector<Vector<Tile>> getTiles()
143         {
144             return m_tiles;
145         }
146
147         /**/
148         /*
149         Board::getTile()
150
151         NAME
152
153                 Board::getTile - returns a Tile object from the specified row and column.
154
155         SYNOPSIS
156
157                 public Board::getTile(int a_row, int a_col);
158                     a_row --> row index of the tile
159                     a_col --> col index of the tile
160
161         DESCRIPTION
162
163                 This function returns a Tile object from the specified col and row.
164
165         RETURNS
166
167                 Tile.
168
169         AUTHOR
170
171                 Austin Fouch
172
173         DATE
174
175                 1/30/2018
176
177         */
178         /**/
179         public Tile getTile(int a_row, int a_col)
180         {
181             return m_tiles.get(a_row).get(a_col);
182         }
183
184         /**/
185         /*
186         Board::setTile()
187
188         NAME
189
190                 Board::setTile - sets the Tile object from the specified row and column to
                    the passed
191                                 Tile.
192
193         SYNOPSIS
194
195                 public Board::setTile(int a_row, int a_col, Tile tile);
196                     a_row   --> row index of the tile
197                     a_col   --> col index of the tile
198                     a_tile  --> Tile which the specified Tile will be set to.
199
200         DESCRIPTION
201
202                 This function sets the Tile at a_row, a_col in m_tiles to a_tile.
```

```
203
204         RETURNS
205
206                 Void.
207
208         AUTHOR
209
210                 Austin Fouch
211
212         DATE
213
214                 1/30/2018
215
216         */
217         /**/
218         public void setTile(int a_row, int a_col, Tile a_tile)
219         {
220             getTiles().get(a_row).set(a_col, a_tile);
221         }
222
223         /**/
224         /*
225         Board::setTiles()
226
227         NAME
228
229                 Board::setTiles - setter for the m_tiles member variable.
230
231         SYNOPSIS
232
233                 public Board::setTiles(Vector<Vector<Tile>> a_tiles);
234                     a_tiles  --> Tiles which m_tiles will be set to.
235
236         DESCRIPTION
237
238                 This function sets the m_tiles to a_tiles.
239
240         RETURNS
241
242                 Void.
243
244         AUTHOR
245
246                 Austin Fouch
247
248         DATE
249
250                 1/30/2018
251
252         */
253         /**/
254         public void setTiles(Vector<Vector<Tile>> a_tiles)
255         {
256             this.m_tiles = a_tiles;
257         }
258     }
259
```

```java
package austinfouch.com.ishido;

/**/
/*
    Comptuer.java

    AUTHOR

        Austin Fouch

    DESCRIPTION

        Computer class. Hold information related to a Computer player in a standard game of Ishido.
        Extends the Player class.

    DATE

        01/30/2018

*/
/**/
public class Computer extends Player
{
    public Computer()
    {
        super();
    }

    /**/
    /*
    Computer::Computer()

    NAME

            Computer::Computer - constructor for the Computer class.

    SYNOPSIS

            public Computer::Computer();

    DESCRIPTION

            This function will construct a Computer object. This is done by calling the Player
            constructor through the super() function.

    RETURNS

            No return value.

    AUTHOR

            Austin Fouch

    DATE

            1/30/2018

    */
    /**/
    public Computer(String a_name, Integer a_score)
    {
        super(a_name, a_score);
    }

    /**/
    /*
    Computer::Play()
```

```
68
69      NAME
70
71              Computer::Play() - returns the most optimal turn given tile and board
                objects.
72
73      SYNOPSIS
74
75              public Turn Computer::Play(Tile a_currTile, Board a_board);
76                  a_currTile  --> Tile object that is being compared to the board object.
77                  a_board     --> Board object that is being searched for optimal moves
                    using a Tile.
78
79      DESCRIPTION
80
81              This function will place the given Tile object on each position on the
                given Board
82              object, saving the score of each play. The play that nets the computer the
                most points
83              is returned as a Turn object.
84
85              This is done by first looping over each vector of vectors in the given Board
86              object and then looping over each of these vectors and calculating the
                score if a user
87              was to play the given Tile object at this position. If this play is the
                play that
88              gains the computer the most points, then this play is saved.
89
90              Once the entire Board object is iterated over, the optimal Turn object is
                created and
91              returned.
92
93      RETURNS
94
95              Turn object.
96
97      AUTHOR
98
99              Austin Fouch
100
101     DATE
102
103             1/30/2018
104
105     */
106     /**/
107     public Turn play(Tile a_currTile, Board a_board)
108     {
109         Integer currValue = 0;
110         Integer bestValue = 0;
111         Integer bestRow = -1;
112         Integer bestCol = -1;
113         // Loop over game board
114         for (int row = 0; row < IshidoConstants.NUM_BOARD_ROWS; row++)
115         {
116             for (int col = 0; col < IshidoConstants.NUM_BOARD_COLS; col++)
117             {
118                 currValue = isLegalPlay(a_currTile, a_board, row, col);
119                 if(currValue > bestValue)
120                 {
121                     bestValue = currValue;
122                     bestRow = row;
123                     bestCol = col;
124                 }
125             }
126         }
127
128         Turn turn = new Turn(a_currTile, bestRow, bestCol, getName(), bestValue);
129
```

```
130             return turn;
131         }
132     }
133
```

```java
package austinfouch.com.ishido;

import java.lang.reflect.Array;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Collections;
import java.util.List;
import java.util.Vector;

/**/
/*
    Deck.java

    AUTHOR

        Austin Fouch

    DESCRIPTION

        Deck class. Holds information related to the deck used in a standard game of Ishido.
            Vector<Tile> m_tiles --> Vector of Tiles, simulating a game deck.

    DATE

        01/30/2018

*/
/**/
public class Deck
{

    private Vector<Tile> m_tiles;

    /**/
    /*
    Deck::Deck()

    NAME

            Deck::Deck - constructor for the Deck class.

    SYNOPSIS

            public Deck::Deck();

    DESCRIPTION

            This function will construct a Deck object. The only member variable of the constructed
            Deck object, m_tiles, is set to be a Vector of Tiles.

    RETURNS

            No return value.

    AUTHOR

            Austin Fouch

    DATE

            1/30/2018

    */
    /**/
    public Deck()
    {
        this.m_tiles = new Vector<>();
```

```
68      }
69
70      /**/
71      /*
72      Deck::getTiles()
73
74      NAME
75
76              Deck::getTiles - getter for the Deck class's m_tiles member variable.
77
78      SYNOPSIS
79
80              public Vector<Tile> Deck::getTiles();
81
82      DESCRIPTION
83
84              This function will return the m_tiles member variable.
85
86      RETURNS
87
88              Vector<Tile>.
89
90      AUTHOR
91
92              Austin Fouch
93
94      DATE
95
96              1/30/2018
97
98      */
99      /**/
100     public Vector<Tile> getTiles()
101     {
102         return this.m_tiles;
103     }
104
105     /**/
106     /*
107     Deck::Deck()
108
109     NAME
110
111             Deck::Deck - copy constructor for the Deck class.
112
113     SYNOPSIS
114
115             public Deck::Deck(Vector<Tile> a_tiles);
116                 a_tiles --> Vector<Tile> to set m_tiles to.
117
118     DESCRIPTION
119
120             This function will set m_tiles to a_tiles.
121
122     RETURNS
123
124             No return value.
125
126     AUTHOR
127
128             Austin Fouch
129
130     DATE
131
132             1/30/2018
133
134     */
135     /**/
136     public Deck(Vector<Tile> a_tiles)
```

```
137         {
138             this.m_tiles = a_tiles;
139         }
140 
141         /**/
142         /*
143         Deck::setTiles()
144 
145         NAME
146 
147                 Deck::setTiles - setter for the m_tiles member variable.
148 
149         SYNOPSIS
150 
151                 public void Deck::setTiles(Vector<Tile> a_tiles);
152                     a_tiles --> Vector<Tile> to set m_tiles to.
153 
154         DESCRIPTION
155 
156                 This function will set m_tiles to a_tiles.
157 
158         RETURNS
159 
160                 Void.
161 
162         AUTHOR
163 
164                 Austin Fouch
165 
166         DATE
167 
168                 1/30/2018
169         */
170         /**/
171         public void setTiles(Vector<Tile> a_tiles)
172         {
173             this.m_tiles = a_tiles;
174         }
175 
176         /**/
177         /*
178         Deck::pop()
179 
180         NAME
181 
182                 Deck::pop - removes the last element in m_tiles.
183 
184         SYNOPSIS
185 
186                 public void Deck::pop();
187 
188         DESCRIPTION
189 
190                 Removes the last Tile in the member variable m_tiles.
191 
192         RETURNS
193 
194                 Void.
195 
196         AUTHOR
197 
198                 Austin Fouch
199 
200         DATE
201 
202                 1/30/2018
203         */
204         /**/
205         public void pop()
```

```
206     {
207         this.m_tiles.removeElementAt(this.m_tiles.size() - 1);
208     }
209 
210     /**/
211     /*
212     Deck::top()
213 
214     NAME
215 
216             Deck::top - returns the last element in m_tiles.
217 
218     SYNOPSIS
219 
220             public Tile Deck::top();
221 
222     DESCRIPTION
223 
224             Returns the last Tile in the member variable m_tiles.
225 
226     RETURNS
227 
228             Tile.
229 
230     AUTHOR
231 
232             Austin Fouch
233 
234     DATE
235 
236             1/30/2018
237     */
238     /**/
239     public Tile top()
240     {
241         return this.m_tiles.elementAt(this.m_tiles.size() - 1);
242     }
243 
244     /**/
245     /*
246     Deck::push()
247 
248     NAME
249 
250             Deck::push - adds a new tile to m_tiles.
251 
252     SYNOPSIS
253 
254             public void Deck::top(Tile a_tile);
255                 a_tile --> Tile added to m_tiles
256 
257     DESCRIPTION
258 
259            Pushes a_tile to the end of m_tiles.
260 
261     RETURNS
262 
263             Void.
264 
265     AUTHOR
266 
267             Austin Fouch
268 
269     DATE
270 
271             1/30/2018
272     */
273     /**/
274     public void push(Tile a_tile)
```

```
275         {
276             this.getTiles().add(a_tile);
277         }
278 
279         /**/
280         /*
281         Deck::setup()
282 
283         NAME
284 
285                 Deck::setup - initializes the deck for a game of Ishido.
286 
287         SYNOPSIS
288 
289                 public void Deck::setup();
290 
291         DESCRIPTION
292 
293                 Initializes the Ishido Deck. Creates 72 Tiles, 2 of each unique color +
                    symbol combo and
294                 shuffles.
295 
296         RETURNS
297 
298                 Void.
299 
300         AUTHOR
301 
302                 Austin Fouch
303 
304         DATE
305 
306                 1/30/2018
307         */
308         /**/
309         public void setup()
310         {
311             // Create 2 combos of every IshidoColor and IshidoSymbol as a tile; push to
312             for (int i = 0; i < IshidoConstants.UNQ_TILE_COUNT; i++)
313             {
314                 for (IshidoColor color : IshidoColor.values())
315                 {
316                     if ((color != IshidoColor.BLANK) && (color != IshidoColor.HELP))
317                     {
318                         for (IshidoSymbol symbol : IshidoSymbol.values())
319                         {
320                             if((symbol != IshidoSymbol.BLANK) && (symbol !=
                                IshidoSymbol.HELP))
321                             {
322                                 Tile t = new Tile(color, symbol);
323                                 this.push(t);
324                             }
325                         }
326                     }
327                 }
328             }
329 
330             Collections.shuffle(this.getTiles());
331         }
332 
333         /**/
334         /*
335         Deck::getSetupTiles()
336 
337         NAME
338 
339                 Deck::getSetupTiles - creates a list of 6 Tiles that contains only one of
                    each symbol
340                                       and color.
```

```
341
342         SYNOPSIS
343
344                 public Vector<Tile> Deck::getSetupTiles();
345
346         DESCRIPTION
347
348                 Creates a new Vector of Tiles that holds 1 tile for each color and symbol
                    for a total of
349                 6 tiles, removing these tiles from the deck as they are added to this new
                    Vector.
350
351         RETURNS
352
353                 Vector<Tile>.
354
355         AUTHOR
356
357                 Austin Fouch
358
359         DATE
360
361                 1/30/2018
362         */
363         /**/
364         public Vector<Tile> getSetupTiles()
365         {
366             Vector<Tile> setupTiles = new Vector<>();
367             List<IshidoColor> colorList = new ArrayList<>();
368             List<IshidoSymbol> symbolList = new ArrayList<>();
369
370             //  1. check to see if tile color and symbol has been seen before
371             //  2. if it has not, add tile to setupTiles, add color/symbol to lists, and
                    remove tile
372             //     from deck.
373             //  3. go to next tile in deck, repeat
374             for(int i = 0; i < m_tiles.size(); i++)
375             {
376                 if(!colorList.contains(m_tiles.get(i).getColor()) &&
377                         !symbolList.contains(m_tiles.get(i).getSymbol()))
378                 {
379                     setupTiles.add(m_tiles.get(i));
380                     colorList.add(m_tiles.get(i).getColor());
381                     symbolList.add(m_tiles.get(i).getSymbol());
382                     m_tiles.removeElementAt(i);
383                 }
384             }
385
386             return setupTiles;
387         }
388     }
389
```

```
1   package austinfouch.com.ishido;
2
3   import android.app.Activity;
4   import android.content.Context;
5   import android.widget.ImageView;
6   import android.widget.LinearLayout;
7   import android.widget.TableLayout;
8   import android.widget.TextView;
9
10  import java.util.Collections;
11  import java.util.Vector;
12
13  /**/
14  /*
15      Game.java
16
17      AUTHOR
18
19          Austin Fouch
20
21      DESCRIPTION
22
23          Game class. This data structure acts the the Game model for the Game Activity
            class. All
24          dynamic aspects of the screen are drawn using this class.
25
26      DATE
27
28          01/30/2018
29
30  */
31  /**/
32  public class Game
33  {
34      private Tile m_currTile;
35      private Deck m_deck;
36      private Board m_board;
37      private ActivityLog m_log;
38      private Player m_playerOne;
39      private Player m_playerTwo;
40
41      /**/
42      /*
43      Game::Game()
44
45      NAME
46
47              Game::Game - constructor for the Game class.
48
49      SYNOPSIS
50
51              public Game::Game();
52
53      DESCRIPTION
54
55              This function will construct a Game object. The member variables
                initialized in this
56              function are m_currTile, m_deck, m_board, m_log, m_playerOne and m_playerTwo.
57
58      RETURNS
59
60              No return value.
61
62      AUTHOR
63
64              Austin Fouch
65
66      DATE
67
```

```
68              1/30/2018
69
70      */
71      /**/
72      public Game()
73      {
74          this.m_currTile = new Tile();
75          this.m_deck = new Deck();
76          this.m_board = new Board();
77          this.m_log = new ActivityLog();
78          this.m_playerOne = new Human();
79          this.m_playerTwo = new Computer();
80      }
81
82      /**/
83      /*
84      Game::Game()
85
86      NAME
87
88              Game::Game - copy constructor for the Game class.
89
90      SYNOPSIS
91
92              public Game::Game(Tile a_currTile, Deck a_deck, Board a_board, ActivityLog
                a_log,
93                          Player a_playerOne, Player a_playerTwo);
94                  a_currTile  --> Tile object to set m_currTile to.
95                  a_deck      --> Deck object to set m_deck to.
96                  a_board     --> Board object to set m_board to.
97                  a_log       --> ActivityLog object to set m_log to.
98                  a_playerOne --> Player object to set m_playerOne to.
99                  a_playerTwo --> Player object ot set m_playerTwo to.
100
101     DESCRIPTION
102
103             This function will construct a Game object. The member variables
                initialized in this
104             function are m_currTile, m_deck, m_board, m_log, m_playerOne and m_playerTwo.
105
106     RETURNS
107
108             No return value.
109
110     AUTHOR
111
112             Austin Fouch
113
114     DATE
115
116             1/30/2018
117
118     */
119     /**/
120     public Game(Tile a_currTile, Deck a_deck, Board a_board, ActivityLog a_log, Player
        a_playerOne,
121                 Player a_playerTwo)
122     {
123         this.m_currTile = a_currTile;
124         this.m_deck = a_deck;
125         this.m_board = a_board;
126         this.m_log = a_log;
127         this.m_playerOne = a_playerOne;
128         this.m_playerTwo = a_playerTwo;
129     }
130
131     /**/
132     /*
133     Game::calculateScore()
```

```
134
135         NAME
136
137                 Game::calculateScore - calculate the score of the given Ishido play.
138
139         SYNOPSIS
140
141                 public Game::calculateScore(Tile a_currTile, Board a_board, int a_row, int
                    a_col);
142                     a_currTile  --> Tile object used to determine score.
143                     a_board     --> Board object used to place Tile object on and determine
                        score.
144                     a_row       --> int representing the row where the Tile object is being
                        played on
145                                     the Board object.
146                     a_col       --> int representing the column where the Tile object is
                        being played on
147                                     the Board object.
148
149         DESCRIPTION
150
151                 This function will calculate the score of an attempted play in an Ishido
                    game. This is
152                 accomplished by using the given Tile and Board objects, placing the Tile on
                    the Board
153                 at the given row and column position, and calculating the score produced.
                    This score
154                 is then returned as an Integer value.
155
156         RETURNS
157
158                 Integer.
159
160         AUTHOR
161
162                 Austin Fouch
163
164         DATE
165
166                 1/30/2018
167
168         */
169         /**/
170         public Integer calculateScore(Tile a_currTile, Board a_board, int a_row, int a_col)
171         {
172             Integer value = 0;
173             // check tile match left
174             if( a_col != 0)
175             {
176                 if( a_currTile.isMatch(a_board.getTile(a_row, a_col - 1)))
177                 {
178                     value++;
179                 }
180             }
181             // check tile match right
182             if( a_col != IshidoConstants.NUM_BOARD_COLS - 1)
183             {
184                 if( a_currTile.isMatch(a_board.getTile(a_row, a_col + 1)))
185                 {
186                     value++;
187                 }
188             }
189             // check tile match above
190             if( a_row != 0)
191             {
192                 if( a_currTile.isMatch(a_board.getTile(a_row - 1, a_col)))
193                 {
194                     value++;
195                 }
```

```
196         }
197         // check tile match below
198         if( a_row != IshidoConstants.NUM_BOARD_ROWS - 1)
199         {
200             if( a_currTile.isMatch(a_board.getTile(a_row + 1, a_col)))
201             {
202                 value++;
203             }
204         }
205         // double 4-way match value
206         if( value > 3)
207         {
208             value = value * 2;
209         }
210 
211         return value;
212     }
213 
214     /**/
215     /*
216     Game::getCurrTile()
217 
218     NAME
219 
220             Game::getCurrTile - getter for the Game object's m_currTile member variable.
221 
222     SYNOPSIS
223 
224             public Game::getCurrTile();
225 
226     DESCRIPTION
227 
228             This function returns the Game's member variable, m_currTile.
229 
230     RETURNS
231 
232             Tile.
233 
234     AUTHOR
235 
236             Austin Fouch
237 
238     DATE
239 
240             1/30/2018
241 
242     */
243     /**/
244     public Tile getCurrTile()
245     {
246         return m_currTile;
247     }
248 
249     /**/
250     /*
251     Game::setCurrTile()
252 
253     NAME
254 
255             Game::setCurrTile - setter for the Game object's m_currTile member variable.
256 
257     SYNOPSIS
258 
259             public Game::setCurrTile(Tile a_currTile);
260                 a_currTile --> Tile object to set m_currTile to.
261 
262     DESCRIPTION
263 
264             This function sets the Game's member variable, m_currTile, to the given Tile.
```

```
265
266         RETURNS
267
268                 Void.
269
270         AUTHOR
271
272                 Austin Fouch
273
274         DATE
275
276                 1/30/2018
277
278         */
279         /**/
280         public void setCurrTile(Tile a_currTile)
281         {
282             this.m_currTile = a_currTile;
283         }
284
285         /**/
286         /*
287         Game::getDeck()
288
289         NAME
290
291                 Game::getDeck - getter for the Game object's m_deck member variable.
292
293         SYNOPSIS
294
295                 public Game::getDeck();
296
297         DESCRIPTION
298
299                 This function returns the Game's member variable, m_deck.
300
301         RETURNS
302
303                 Deck.
304
305         AUTHOR
306
307                 Austin Fouch
308
309         DATE
310
311                 1/30/2018
312
313         */
314         /**/
315         public Deck getDeck()
316         {
317             return m_deck;
318         }
319
320         /**/
321         /*
322         Game::setDeck()
323
324         NAME
325
326                 Game::setDeck - setter for the Game object's m_deck member variable.
327
328         SYNOPSIS
329
330                 public Game::setDeck(Deck a_deck);
331                     a_deck --> Deck object to set m_deck to.
332
333         DESCRIPTION
```

```
334
335             This function sets the Game's member variable, m_deck, to the given Deck
                variable.
336
337     RETURNS
338
339             Void.
340
341     AUTHOR
342
343             Austin Fouch
344
345     DATE
346
347             1/30/2018
348
349     */
350     /**/
351     public void setDeck(Deck a_deck)
352     {
353         this.m_deck = a_deck;
354     }
355
356     /**/
357     /*
358     Game::getBoard()
359
360     NAME
361
362             Game::getBoard - getter for the Game object's m_board member variable.
363
364     SYNOPSIS
365
366             public Game::getBoard();
367
368     DESCRIPTION
369
370             This function returns the Game's member variable, m_board.
371
372     RETURNS
373
374             Board.
375
376     AUTHOR
377
378             Austin Fouch
379
380     DATE
381
382             1/30/2018
383
384     */
385     /**/
386     public Board getBoard()
387     {
388         return m_board;
389     }
390
391     /**/
392     /*
393     Game::setBoard()
394
395     NAME
396
397             Game::setBoard - setter for the Game object's m_board member variable.
398
399     SYNOPSIS
400
401             public Game::setBoard(Board a_board);
```

```
402                 a_board --> Board object to set m_board to.
403
404         DESCRIPTION
405
406                 This function sets the Game's member variable, m_board, to the given Board
                    variable.
407
408         RETURNS
409
410                 Void.
411
412         AUTHOR
413
414                 Austin Fouch
415
416         DATE
417
418                 1/30/2018
419
420         */
421         /**/
422         public void setBoard(Board a_board)
423         {
424             this.m_board = a_board;
425         }
426
427         /**/
428         /*
429         Game::getLog()
430
431         NAME
432
433                 Game::getLog - getter for the Game object's m_log member variable.
434
435         SYNOPSIS
436
437                 public Game::getLog();
438
439         DESCRIPTION
440
441                 This function returns the Game's member variable, m_log.
442
443         RETURNS
444
445                 ActivityLog.
446
447         AUTHOR
448
449                 Austin Fouch
450
451         DATE
452
453                 1/30/2018
454
455         */
456         /**/
457         public ActivityLog getLog()
458         {
459             return m_log;
460         }
461
462         /**/
463         /*
464         Game::setLog()
465
466         NAME
467
468                 Game::setLog - setter for the Game object's m_log member variable.
469
```

```
470       SYNOPSIS
471
472               public Game::setLog(ActivityLog a_log);
473                   a_log --> ActivityLog object to set m_log to.
474
475       DESCRIPTION
476
477               This function sets the Game's member variable, m_log, to the given
                  ActivityLog
478               variable.
479
480       RETURNS
481
482               Void.
483
484       AUTHOR
485
486               Austin Fouch
487
488       DATE
489
490               1/30/2018
491
492       */
493       /**/
494       public void setLog(ActivityLog a_log)
495       {
496           this.m_log = a_log;
497       }
498
499       /**/
500       /*
501       Game::getPlayerOne()
502
503       NAME
504
505               Game::getPlayerOne - getter for the Game object's m_playerOne member
                  variable.
506
507       SYNOPSIS
508
509               public Game::getPlayerOne();
510
511       DESCRIPTION
512
513               This function returns the Game's member variable, m_playerOne.
514
515       RETURNS
516
517               Player.
518
519       AUTHOR
520
521               Austin Fouch
522
523       DATE
524
525               1/30/2018
526
527       */
528       /**/
529       public Player getPlayerOne()
530       {
531           return this.m_playerOne;
532       }
533
534       /**/
535       /*
536       Game::setPlayerOne()
```

```
537
538         NAME
539
540                 Game::setPlayerOne - setter for the Game object's m_playerOne member
                    variable.
541
542         SYNOPSIS
543
544                 public Game::setPlayerOne(Player a_player);
545                     a_player --> Player object to set m_playerOne to.
546
547         DESCRIPTION
548
549                 This function sets the Game's member variable, m_playerOne, to the given
                    Player
550                 variable.
551
552         RETURNS
553
554                 Void.
555
556         AUTHOR
557
558                 Austin Fouch
559
560         DATE
561
562                 1/30/2018
563
564         */
565         /**/
566         public void setPlayerOne(Player a_player)
567         {
568             this.m_playerOne = a_player;
569         }
570
571         /**/
572         /*
573         Game::getPlayerTwo()
574
575         NAME
576
577                 Game::getPlayerTwo - getter for the Game object's m_playerTwo member
                    variable.
578
579         SYNOPSIS
580
581                 public Game::getPlayerTwo();
582
583         DESCRIPTION
584
585                 This function returns the Game's member variable, m_playerTwo.
586
587         RETURNS
588
589                 Player.
590
591         AUTHOR
592
593                 Austin Fouch
594
595         DATE
596
597                 1/30/2018
598
599         */
600         /**/
601         public Player getPlayerTwo()
602         {
```

```
603             return this.m_playerTwo;
604         }
605 
606         /**/
607         /*
608         Game::setPlayerTwo()
609 
610         NAME
611 
612                 Game::setPlayerTwo - setter for the Game object's m_playerTwo member
                    variable.
613 
614         SYNOPSIS
615 
616                 public Game::setPlayerTwo(Player a_player);
617                     a_player --> Player object to set m_playerTwo to.
618 
619         DESCRIPTION
620 
621                 This function sets the Game's member variable, m_playerTwo, to the given
                    Player
622                 variable.
623 
624         RETURNS
625 
626                 Void.
627 
628         AUTHOR
629 
630                 Austin Fouch
631 
632         DATE
633 
634                 1/30/2018
635 
636         */
637         /**/
638         public void setPlayerTwo(Player a_player)
639         {
640             this.m_playerTwo = a_player;
641         }
642 
643         /**/
644         /*
645         Game::setup()
646 
647         NAME
648 
649                 Game::setup - sets up the Game object of the start of an Ishido game.
650 
651         SYNOPSIS
652 
653                 public Game::setup();
654 
655         DESCRIPTION
656 
657                 This function will setup the m_deck, m_board, m_currTile, and Player member
                    variables
658                 for the start of a game of Ishido.
659 
660         RETURNS
661 
662                 Void.
663 
664         AUTHOR
665 
666                 Austin Fouch
667 
668         DATE
```

```
669
670                 1/30/2018
671
672         */
673         /**/
674         public void setup()
675         {
676             // Initialize deck
677             // setupTiles are the 6 tiles that start on the Ishido board
678             setDeck(new Deck());
679             getDeck().setup();
680             Vector<Tile> setupTiles = getDeck().getSetupTiles();
681             Collections.shuffle(setupTiles);
682
683             // initialize board
684             // set the board so these setupTiles take up specific positions:
685             // (0,0), (0,11), (3, 5), (4, 6), (7, 0), (7, 11)
686             setBoard(new Board());
687             getBoard().setTile(0, 0, setupTiles.firstElement());
688             setupTiles.removeElementAt(0);
689             getBoard().setTile(0, 11, setupTiles.firstElement());
690             setupTiles.removeElementAt(0);
691             getBoard().setTile(3, 5, setupTiles.firstElement());
692             setupTiles.removeElementAt(0);
693             getBoard().setTile(4, 6, setupTiles.firstElement());
694             setupTiles.removeElementAt(0);
695             getBoard().setTile(7, 0, setupTiles.firstElement());
696             setupTiles.removeElementAt(0);
697             getBoard().setTile(7, 11, setupTiles.firstElement());
698             setupTiles.removeElementAt(0);
699
700             // Initialize current tile with top tile of deck
701             getCurrTile().setColor(getDeck().top().getColor());
702             getCurrTile().setSymbol(getDeck().top().getSymbol());
703             getDeck().pop();
704
705             // Initialize Activity Log
706             setLog(new ActivityLog());
707             getLog().setPlayerOneTurn("The Game has begun!");
708         }
709     }
710
```

```
package austinfouch.com.ishido;

import android.app.AlertDialog;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.res.Resources;
import android.graphics.drawable.Drawable;
import android.media.Image;
import android.support.v7.app.ActionBar;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.text.Layout;
import android.view.View;
import android.view.Window;
import android.view.WindowManager;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.LinearLayout;
import android.widget.TableLayout;
import android.widget.TableRow;
import android.widget.TextView;
import android.widget.Toast;

import java.util.Vector;

/**/
/*
    GameActivity.java

    AUTHOR

        Austin Fouch

    DESCRIPTION

        Game Activity class. This Activity is the same layout for both Solitaire and
        Standard

        The m_game member variable is the model for each Ishido game. Every other
        member variable is
        a layout or view within the Game Activity that visualizes the m_game model to
        the user.

        The only interactive views are the individual ImageViews that make up the game
        board.

    DATE

        01/30/2018

*/
/**/
public class GameActivity extends AppCompatActivity implements View.OnClickListener
{
    private Game m_game;
    private TableLayout m_boardLayout;
    private ImageView m_currTileLayout;
    private TextView m_tileCountLayout;
    private TextView m_player1NameLayout;
    private TextView m_player1ScoreLayout;
    private TextView m_player2NameLayout;
    private TextView m_player2ScoreLayout;
    private TextView m_player1LogView;
    private TextView m_player2LogView;
    private Boolean m_solitaireFlag;

    public TextView getPlayer1LogView()
    {
        return this.m_player1LogView;
```

```
66      }
67
68      public void setPlayer1LogView(TextView a_player1LogView)
69      {
70          this.m_player1LogView = a_player1LogView;
71      }
72
73      public TextView getPlayer2LogView()
74      {
75          return this.m_player2LogView;
76      }
77
78      public void setPlayer2LogView(TextView a_player2LogView)
79      {
80          this.m_player2LogView = a_player2LogView;
81      }
82
83      /**/
84      /*
85      GameActivity::getGame()
86
87      NAME
88
89              GameActivity::getGame - getter for the m_game member variable.
90
91      SYNOPSIS
92
93              public Game GameActivity::getGame();
94
95      DESCRIPTION
96
97              This function returns the m_game member variable.
98
99      RETURNS
100
101             Game.
102
103     AUTHOR
104
105             Austin Fouch
106
107     DATE
108
109             1/30/2018
110     */
111     /**/
112     public Game getGame()
113     {
114         return m_game;
115     }
116
117     /**/
118     /*
119     GameActivity::setGame()
120
121     NAME
122
123             GameActivity::setGame - setter for the m_game member variable.
124
125     SYNOPSIS
126
127             public void GameActivity::getGame(Game a_game);
128                 a_game --> value to set m_game to.
129
130     DESCRIPTION
131
132             This function sets the m_game member variable.
133
134     RETURNS
```

```
135
136             Void.
137
138         AUTHOR
139
140             Austin Fouch
141
142         DATE
143
144             1/30/2018
145         */
146         /**/
147         public void setGame(Game a_game) {
148             this.m_game = a_game;
149         }
150
151         /**/
152         /*
153         GameActivity::getBoardLayout()
154
155         NAME
156
157             GameActivity::getBoardLayout - getter for the m_boardLayout member variable.
158
159         SYNOPSIS
160
161             public TableLayout GameActivity::getBoardLayout();
162
163         DESCRIPTION
164
165             This function returns the m_boardLayout member variable.
166
167         RETURNS
168
169             TableLayout.
170
171         AUTHOR
172
173             Austin Fouch
174
175         DATE
176
177             1/30/2018
178         */
179         /**/
180         public TableLayout getBoardLayout()
181         {
182             return m_boardLayout;
183         }
184
185         /**/
186         /*
187         GameActivity::setBoardLayout()
188
189         NAME
190
191             GameActivity::setBoardLayout - setter for the m_boardLayout member variable.
192
193         SYNOPSIS
194
195             public void GameActivity::setBoardLayout(TableLayout a_boardLayout);
196                 a_boardLayout --> value to set m_boardLayout to.
197
198         DESCRIPTION
199
200             This function sets the m_game member variable.
201
202         RETURNS
203
```

```
204             Void.
205
206     AUTHOR
207
208             Austin Fouch
209
210     DATE
211
212             1/30/2018
213     */
214     /**/
215     public void setBoardLayout(TableLayout a_boardLayout)
216     {
217         this.m_boardLayout = a_boardLayout;
218     }
219
220     /**/
221     /*
222     GameActivity::getCurrTileLayout()
223
224     NAME
225
226             GameActivity::getCurrTileLayout - getter for the m_currTileLayout member
                variable.
227
228     SYNOPSIS
229
230             public ImageView GameActivity::getCurrTileLayout();
231
232     DESCRIPTION
233
234             This function returns the m_currTileLayout member variable.
235
236     RETURNS
237
238             ImageView.
239
240     AUTHOR
241
242             Austin Fouch
243
244     DATE
245
246             1/30/2018
247     */
248     /**/
249     public ImageView getCurrTileLayout()
250     {
251         return m_currTileLayout;
252     }
253
254     /**/
255     /*
256     GameActivity::setCurrTileLayout()
257
258     NAME
259
260             GameActivity::setCurrTileLayout - setter for the m_currTileLayout member
                variable.
261
262     SYNOPSIS
263
264             public void GameActivity::setCurrTileLayout(ImageView a_currTileLayout);
265                 a_currTileLayout --> value to set m_currTileLayout to.
266
267     DESCRIPTION
268
269             This function sets the m_currTileLayout member variable.
270
```

```
271         RETURNS
272
273                 Void.
274
275         AUTHOR
276
277                 Austin Fouch
278
279         DATE
280
281                 1/30/2018
282         */
283         /**/
284         public void setCurrTileLayout(ImageView a_currTileLayout)
285         {
286             this.m_currTileLayout = a_currTileLayout;
287         }
288
289         /**/
290         /*
291         GameActivity::getTileCountLayout()
292
293         NAME
294
295                 GameActivity::getTileCountLayout - getter for the m_tileCountLayout member
                    variable.
296
297         SYNOPSIS
298
299                 public ImageView GameActivity::getTileCountLayout();
300
301         DESCRIPTION
302
303                 This function returns the m_tileCountLayout member variable.
304
305         RETURNS
306
307                 TextView.
308
309         AUTHOR
310
311                 Austin Fouch
312
313         DATE
314
315                 1/30/2018
316         */
317         /**/
318         public TextView getTileCountLayout()
319         {
320             return m_tileCountLayout;
321         }
322
323         /**/
324         /*
325         GameActivity::setTileCountLayout()
326
327         NAME
328
329                 GameActivity::setTileCountLayout - setter for the m_tileCountLayout member
                    variable.
330
331         SYNOPSIS
332
333                 public void GameActivity::setTileCountLayout(TextView a_tileCountLayout);
334                     a_tileCountLayout --> value to set m_tileCountLayout to.
335
336         DESCRIPTION
337
```

```
338             This function sets the m_tileCountLayout member variable.
339
340     RETURNS
341
342             Void.
343
344     AUTHOR
345
346             Austin Fouch
347
348     DATE
349
350             1/30/2018
351     */
352     /**/
353     public void setTileCountLayout(TextView a_tileCountLayout)
354     {
355         this.m_tileCountLayout = a_tileCountLayout;
356     }
357
358     /**/
359     /*
360     GameActivity::getPlayer1NameLayout()
361
362     NAME
363
364             GameActivity::getPlayer1NameLayout - getter for the m_player1NameLayout
                member variable.
365
366     SYNOPSIS
367
368             public ImageView GameActivity::getPlayer1NameLayout();
369
370     DESCRIPTION
371
372             This function returns the m_player1NameLayout member variable.
373
374     RETURNS
375
376             TextView.
377
378     AUTHOR
379
380             Austin Fouch
381
382     DATE
383
384             1/30/2018
385     */
386     /**/
387     public TextView getPlayer1NameLayout()
388     {
389         return m_player1NameLayout;
390     }
391
392     /**/
393     /*
394     GameActivity::setPlayer1NameLayout()
395
396     NAME
397
398             GameActivity::setPlayer1NameLayout - setter for the m_player1NameLayout
                member variable.
399
400     SYNOPSIS
401
402             public void GameActivity::setPlayer1NameLayout(TextView a_player1NameLayout);
403                 a_player1NameLayout --> value to set m_player1NameLayout to.
404
```

```
405         DESCRIPTION
406
407                 This function sets the m_player1NameLayout member variable.
408
409         RETURNS
410
411                 Void.
412
413         AUTHOR
414
415                 Austin Fouch
416
417         DATE
418
419                 1/30/2018
420         */
421         /**/
422         public void setPlayer1NameLayout(TextView a_player1NameLayout)
423         {
424             this.m_player1NameLayout = a_player1NameLayout;
425         }
426
427         /**/
428         /*
429         GameActivity::getPlayer1ScoreLayout()
430
431         NAME
432
433                 GameActivity::getPlayer1ScoreLayout - getter for the m_player1ScoreLayout
                    member variable.
434
435         SYNOPSIS
436
437                 public ImageView GameActivity::getPlayer1ScoreLayout();
438
439         DESCRIPTION
440
441                 This function returns the m_player1ScoreLayout member variable.
442
443         RETURNS
444
445                 TextView.
446
447         AUTHOR
448
449                 Austin Fouch
450
451         DATE
452
453                 1/30/2018
454         */
455         /**/
456         public TextView getPlayer1ScoreLayout()
457         {
458             return m_player1ScoreLayout;
459         }
460
461         /**/
462         /*
463         GameActivity::setPlayer1ScoreLayout()
464
465         NAME
466
467                 GameActivity::setPlayer1ScoreLayout - setter for the m_player1ScoreLayout
                    member variable.
468
469         SYNOPSIS
470
471                 public void GameActivity::setPlayer1ScoreLayout(TextView
```

```
            a_player1ScoreLayout);
472             a_player1ScoreLayout --> value to set m_player1ScoreLayout to.
473
474     DESCRIPTION
475
476             This function sets the m_player1ScoreLayout member variable.
477
478     RETURNS
479
480             Void.
481
482     AUTHOR
483
484             Austin Fouch
485
486     DATE
487
488             1/30/2018
489     */
490     /**/
491     public void setPlayer1ScoreLayout(TextView a_player1ScoreLayout)
492     {
493         this.m_player1ScoreLayout = a_player1ScoreLayout;
494     }
495
496     /**/
497     /*
498     GameActivity::getPlayer2NameLayout()
499
500     NAME
501
502             GameActivity::getPlayer2NameLayout - getter for the m_player2NameLayout
                member variable.
503
504     SYNOPSIS
505
506             public ImageView GameActivity::getPlayer2NameLayout();
507
508     DESCRIPTION
509
510             This function returns the m_player2NameLayout member variable.
511
512     RETURNS
513
514             TextView.
515
516     AUTHOR
517
518             Austin Fouch
519
520     DATE
521
522             1/30/2018
523     */
524     /**/
525     public TextView getPlayer2NameLayout()
526     {
527         return m_player2NameLayout;
528     }
529
530     /**/
531     /*
532     GameActivity::setPlayer2NameLayout()
533
534     NAME
535
536             GameActivity::setPlayer2NameLayout - setter for the m_player2NameLayout
                member variable.
537
```

```
538         SYNOPSIS
539
540                 public void GameActivity::setPlayer2NameLayout(TextView a_player2NameLayout);
541                     a_player2NameLayout --> value to set m_player2NameLayout to.
542
543         DESCRIPTION
544
545                 This function sets the m_player2NameLayout member variable.
546
547         RETURNS
548
549                 Void.
550
551         AUTHOR
552
553                 Austin Fouch
554
555         DATE
556
557                 1/30/2018
558         */
559         /**/
560         public void setPlayer2NameLayout(TextView a_player2NameLayout)
561         {
562             this.m_player2NameLayout = a_player2NameLayout;
563         }
564
565         /**/
566         /*
567         GameActivity::getPlayer2ScoreLayout()
568
569         NAME
570
571                 GameActivity::getPlayer2ScoreLayout - getter for the m_player2ScoreLayout
                    member variable.
572
573         SYNOPSIS
574
575                 public ImageView GameActivity::getPlayer2ScoreLayout();
576
577         DESCRIPTION
578
579                 This function returns the m_player2ScoreLayout member variable.
580
581         RETURNS
582
583                 TextView.
584
585         AUTHOR
586
587                 Austin Fouch
588
589         DATE
590
591                 1/30/2018
592         */
593         /**/
594         public TextView getPlayer2ScoreLayout()
595         {
596             return m_player2ScoreLayout;
597         }
598
599         /**/
600         /*
601         GameActivity::setPlayer2ScoreLayout()
602
603         NAME
604
605                 GameActivity::setPlayer2ScoreLayout - setter for the m_player2ScoreLayout
```

```
606             member variable.
607
608     SYNOPSIS
609
610             public void GameActivity::setPlayer2ScoreLayout(TextView
                a_player2ScoreLayout);
611                 a_player2ScoreLayout --> value to set m_player2ScoreLayout to.
612
613     DESCRIPTION
614
615             This function sets the m_player2ScoreLayout member variable.
616
617     RETURNS
618
619             Void.
620
621     AUTHOR
622
623             Austin Fouch
624
625     DATE
626
627             1/30/2018
628     */
629     /**/
630     public void setPlayer2ScoreLayout(TextView a_player2ScoreLayout)
631     {
632         this.m_player2ScoreLayout = a_player2ScoreLayout;
633     }
634
635     /**/
636     /*
637     GameActivity::onCreate()
638
639     NAME
640
641             GameActivity::onCreate - called when the Game Activity is created.
642
643     SYNOPSIS
644
645             public void GameActivity::onCreate();
646
647     DESCRIPTION
648
649             This function will set the layout of the Game Activity to the XML file
                located
650             at R.layout.activity_launcher. Then each of the layout/view member
                variables are set to
651             the specified layout/view IDs in the XML file. The game model is setup using
652             game.setup(). Every ImageView within the TableLayout has an OnClick
                listener enabled,
653             which is handled the Game.onClick() function. The initial m_game model is
                draw to the
654             screen through the Game.draw() functions. The m_solitaireFlag is set to
                true if the
655             Intent object's Extra variable solitaireFlag has a value of "true" or false
                if the
656             solitaireFlag Extra variable has a value of "false".
657
658     RETURNS
659
660             Void.
661
662     AUTHOR
663
664             Austin Fouch
665
666     DATE
```

```
667             1/30/2018
668
669     */
670     /**/
671     @Override
672     protected void onCreate(Bundle savedInstanceState)
673     {
674         super.onCreate(savedInstanceState);
675         setContentView(R.layout.activity_game);
676
677         // hides big action bar on app
678         getSupportActionBar().hide();
679
680         // initialize layouts
681         setBoardLayout((TableLayout) findViewById(R.id.boardLayout));
682         setCurrTileLayout((ImageView) findViewById(R.id.currentTileView));
683         setTileCountLayout((TextView) findViewById(R.id.tileCountView));
684         setPlayer1NameLayout((TextView) findViewById(R.id.playerLabel));
685         setPlayer1ScoreLayout((TextView) findViewById(R.id.scoreView));
686         setPlayer2NameLayout((TextView) findViewById(R.id.playerLabel2));
687         setPlayer2ScoreLayout((TextView) findViewById(R.id.scoreView2));
688         setPlayer1LogView((TextView) findViewById(R.id.player1Turn));
689         setPlayer2LogView((TextView) findViewById(R.id.player2Turn));
690
691         // create new game
692         setGame(new Game());
693         getGame().setup();
694
695         // establish row tags, column tags, and onclick listeners for every tileView
696         setupBoardListeners(getBoardLayout());
697
698         // draw current tile, board, and tile count
699         drawTile(getGame().getCurrTile(), getCurrTileLayout());
700         drawBoard(getGame().getBoard(), getBoardLayout());
701         getGame().getDeck().pop();
702         drawTileCount(getGame().getDeck(), getTileCountLayout());
703
704         // get intent, set user name
705         Intent intent = getIntent();
706         getGame().getPlayerOne().setName(intent.getStringExtra("playerName"));
707         getGame().getPlayerOne().setScore(0);
708         String temp = intent.getStringExtra("solitaireFlag");
709         if(temp.equals("false"))
710         {
711             m_solitaireFlag = false;
712         } else {
713             m_solitaireFlag = true;
714         }
715
716         if(m_solitaireFlag == false)
717         {
718             getGame().getPlayerTwo().setName("HAL9000");
719             getGame().getPlayerTwo().setScore(0);
720         }
721
722         drawPlayers(getGame(), getPlayer1NameLayout(), getPlayer1ScoreLayout(),
723                 getPlayer2NameLayout(), getPlayer2ScoreLayout());
724     }
725
726     /**/
727     /*
728     GameActivity::drawPlayers()
729
730     NAME
731
732             GameActivity::drawPlayers - draw Player data to screen.
733
734     SYNOPSIS
735
```

```
736             public void GameActivity::drawPlayers(Game a_game, TextView
                a_player1NameView,
737                             TextView a_player1ScoreView, TextView a_player2NameView,
738                             TextView a_player2ScoreView);
739                 a_game              --> Game variable used to determine Players' name
                    and score.
740                 a_player1NameView   --> TextView which has its text value set
                    Game.m_playerOne.name.
741                 a_player1ScoreView  --> TextView which has its text value set
                    Game.m_playerOne.score.
742                 a_player2NameView   --> TextView which has its text value set
                    Game.m_playerTwo.name.
743                 a_player2ScoreView  --> TextView which has its text value set
                    Game.m_playerTwo.score.
744
745         DESCRIPTION
746
747             This function will use the given Game object to set the text values for the
                given
748             TextView objects.
749
750         RETURNS
751
752             Void.
753
754         AUTHOR
755
756             Austin Fouch
757
758         DATE
759
760             1/30/2018
761
762         */
763         /**/
764         public void drawPlayers(Game a_game, TextView a_player1NameView, TextView
        a_player1ScoreView,
765                                 TextView a_player2NameView, TextView a_player2ScoreView)
766         {
767             a_player1NameView.setText(a_game.getPlayerOne().getName());
768             a_player1ScoreView.setText(a_game.getPlayerOne().getScore().toString());
769             if(m_solitaireFlag == false)
770             {
771                 a_player2NameView.setText(a_game.getPlayerTwo().getName());
772                 a_player2ScoreView.setText(a_game.getPlayerTwo().getScore().toString());
773             }
774         }
775
776         /**/
777         /*
778         GameActivity::drawTile()
779
780         NAME
781
782             GameActivity::drawTile - draw tile to the screen.
783
784         SYNOPSIS
785
786             public void GameActivity::drawTile(Tile a_currTile, ImageView
                a_currTileView);
787                 a_currTile      --> Tile object to draw.
788                 a_currTileView  --> ImageView where a_currTile is drawn to.
789
790         DESCRIPTION
791
792             This function will use the given Tile object to set the foreground and
                background
793             resources of the given ImageView.
794
```

```
795         RETURNS
796
797                 Void.
798
799         AUTHOR
800
801                 Austin Fouch
802
803         DATE
804
805                 1/30/2018
806
807         */
808         /**/
809         public void drawTile(Tile a_currTile, ImageView a_currTileView)
810         {
811             Resources resources = this.getResources();
812
813             // draw foreground
814             String fgResStr = a_currTile.getSymbolResourceStr();
815             final int fgResId = resources.getIdentifier(fgResStr, "drawable",
                getPackageName());
816             Drawable fg = getDrawable(fgResId);
817             a_currTileView.setForeground(fg);
818
819             // draw background
820             String bgResStr = a_currTile.getColorResourceStr();
821             final int bgResId = resources.getIdentifier(bgResStr, "drawable",
                getPackageName());
822             Drawable bg = getDrawable(bgResId);
823             a_currTileView.setBackground(bg);
824
825             // disable onclick unless blank or help tile
826             if(a_currTile.getSymbol() != IshidoSymbol.BLANK && a_currTile.getSymbol() !=
                IshidoSymbol.HELP)
827             {
828                 a_currTileView.setOnClickListener(null);
829             }
830         }
831
832         /**/
833         /*
834         GameActivity::drawTileCount()
835
836         NAME
837
838                 GameActivity::drawTileCount - draw tile to the screen.
839
840         SYNOPSIS
841
842                 public void GameActivity::drawTileCount(Deck a_deck, TextView
                    a_tileCountView);
843                     a_deck          --> Deck object used to draw deck size to screen.
844                     a_tileCountView --> TextView where a_deck.size() is drawn to.
845
846         DESCRIPTION
847
848                 This function will use the given Deck object to draw the object's size to the
849                 the deck size to screen.
850
851         RETURNS
852
853                 Void.
854
855         AUTHOR
856
857                 Austin Fouch
858
859         DATE
```

```
860
861             1/30/2018
862
863     */
864     /**/
865     public void drawTileCount(Deck a_deck, TextView a_tileCountView)
866     {
867         // set tile count to current deck size
868         Integer deckSize = a_deck.getTiles().size();
869         a_tileCountView.setText(deckSize.toString());
870     }
871
872     /**/
873     /*
874     GameActivity::drawBoard()
875
876     NAME
877
878             GameActivity::drawBoard - draw game board to the screen.
879
880     SYNOPSIS
881
882             public void GameActivity::drawBoard(Board a_board, TableLayout a_boardView);
883                 a_board     --> Board object used to draw the game board to screen.
884                 a_boardView --> TextView where a_deck.size() is drawn to.
885
886     DESCRIPTION
887
888             This function will use the given Board object to draw game board to the
                screen.
889
890             This is accomplished by iterating over the children of the given
                TableLayout, which
891             return TableRow layouts. Then iterating over the children of a TableRow
                layout,
892             ImageViews are returned.
893
894             Each ImageView in these TableRows represents an individual square on the
                game board, and
895             is given two tags, row and col, to save positional data relevant to the
                game board on
896             each ImageView. This aspect of the game board layout is done in the
897             GameActivity.enableListeners() function.
898
899             Every ImageView is then drawn using the GameActivity.drawTile() function,
                passing
900             the current ImageView and the associated Tile object in the
                GameActivity.game.board
901             object.
902
903     RETURNS
904
905             Void.
906
907     AUTHOR
908
909             Austin Fouch
910
911     DATE
912
913             1/30/2018
914
915     */
916     /**/
917     public void drawBoard(Board a_board, TableLayout a_boardView)
918     {
919         // loop over children (TableRow) of a_boardView (TableLayout)
920         for( Integer i = 0; i < a_boardView.getChildCount(); i++)
921         {
```

```
            View rowView = a_boardView.getChildAt(i);
            if( rowView instanceof TableRow)
            {
                // loop over children (ImageView) of rowView (TableRow)
                for( Integer j = 0; j < ((TableRow) rowView).getChildCount(); j++)
                {
                    View tileView = ((TableRow) rowView).getChildAt(j);
                    if ( tileView instanceof ImageView)
                    {
                        // get the tile from board model at i, j and draw the tile at
                        that position
                        Tile currTile = a_board.getTile(i, j);
                        drawTile(currTile, (ImageView) tileView);
                    }
                }
            }
        }
    }

    /**/
    /*
    GameActivity::setupBoardListeners()

    NAME

            GameActivity::setupBoardListeners - setup onClick() listeners on the game
            board.

    SYNOPSIS

            public void GameActivity::setupBoardListeners(TableLayout a_boardLayout);
                a_boardLayout --> TableLayout to set onClick() listeners on.

    DESCRIPTION

            This function will enable OnClick() listeners on every ImageView in every
            TableRow in
            the given TableLayout.

    RETURNS

            Void.

    AUTHOR

            Austin Fouch

    DATE

            1/30/2018

    */
    /**/
    public void setupBoardListeners(TableLayout a_boardLayout)
    {
        // loop over children (TableRow) of a_boardView (TableLayout)
        for( Integer i = 0; i < a_boardLayout.getChildCount(); i++)
        {
            View rowView = a_boardLayout.getChildAt(i);
            if( rowView instanceof TableRow)
            {
                // loop over children (ImageView) of rowView (TableRow)
                for( Integer j = 0; j < ((TableRow) rowView).getChildCount(); j++)
                {
                    View tileView = ((TableRow) rowView).getChildAt(j);
                    if ( tileView instanceof ImageView)
                    {
                        // set tags --> 0:row 1:col, enable OnClickListener
                        tileView.setTag(R.string.row, i);
```

```
                        tileView.setTag(R.string.col, j);
                        tileView.setOnClickListener(this);
                    }
                }
            }
        }
    }

    /**/
    /*
    GameActivity::onClick()

    NAME

            GameActivity::onClick - onClick handler for ImageView and Button objects.

    SYNOPSIS

            public void GameActivity::onClick(View v);
                v --> View/Button being clicked.

    DESCRIPTION

            This function will handle the onClick functionality of every ImageView and
            Button in the
            Game Activity.

            ImageViews:
                Every clickable ImageView is found on the TableLayout which represents
                the game
                board. These game board squares are ImageViews, and are only clickable
                when they
                have no Tile object placed on them. If they are indeed Blank squares,
                then when
                clicked, this function interprets that click as the user attempting to
                play the
                current tile to clicked ImageView, or game board square. The user is
                prompted to
                confirm their attempted play, given the row and column they are playing
                at. If the
                play is confirmed, the score of the play is then calculated. If the
                result is
                greater than 0, then the play is legal and made. The score is added to
                the human
                player's score and their log is updated. The current tile is changed to
                the tile at
                the top of the deck and the deck is popped.

                If the game is in Solitaire, the above process repeats until the deck
                is empty,
                redrawing the current tile, tile count, board, score, and log after
                each play by
                the user.

                If the game is in Standard, HAL9000 makes his move after each of the
                human player's
                move. HAL9000's move is determined by placing the current tile on each
                blank game
                board square, calculating the score gained from the play. The highest
                scoring play
                is made. After HAL9000's move is made, the current tile, tile count,
                game board,
                scores, and logs are redrawn.

            Buttons:
                There is only one button in this Activity, the quit button. Once
                clicked, the user
                will be prompted to confirm they want to exit the game.
```

```
                If Yes is clicked, then the application is returned to the Launcher
                Activity.

                If No is clicked, the the application is returned to the current state
                of the Game
                Activity.

    RETURNS

            Void.

    AUTHOR

            Austin Fouch

    DATE

            1/30/2018

    */
    /**/
    @Override
    public void onClick(final View v) {
        if(v instanceof Button)
        {
            quitGame();
        }

        // get currentTile and make sure the onClick doesn't trigger for it
        final ImageView currTileView = (ImageView) findViewById(R.id.currentTileView);
        //final int confirmTileId =
        this.getResources().getIdentifier("confirm_tile.png", "drawable",
        this.getPackageName());
        int currTileId = currTileView.getId();
        if(v instanceof ImageView && v.getId() != currTileId) // is ImageView, but not
        currentTileView
        {
            // build alert dialog, telling user of the tile position clicked
            AlertDialog.Builder builder = new  AlertDialog.Builder(this)
                    .setIcon(android.R.drawable.ic_dialog_alert)
                    .setTitle("")
                    .setMessage("Place tile here?\n\nrow : " + ((int)
                    v.getTag(R.string.row) + 1) + "\ncol  : " + ((int)
                    v.getTag(R.string.col) + 1) )
                    .setPositiveButton("Yes", new DialogInterface.OnClickListener()
                    {
                        @Override
                        public void onClick(DialogInterface dialog, int which) {
                            Intent intent = getIntent();
                            int row = (int) v.getTag(R.string.row);
                            int col = (int) v.getTag(R.string.col);
                            Integer score =
                            getGame().calculateScore(getGame().getCurrTile(),
                            getGame().getBoard(), row, col);
                            if (score > 0) // legal move
                            {
                                // draw tile being played on ImageView that was clicked
                                drawTile(getGame().getCurrTile(), (ImageView) v);

                                // set tile in board model, set current tile to
                                deck.top(), then pop deck
                                getGame().getBoard().setTile(row, col,
                                getGame().getCurrTile());

                                getGame().getPlayerOne().setScore(getGame().getPlayerOne(
                                ).getScore() + score);
                                try {
                                    getGame().setCurrTile(getGame().getDeck().top());
                                    getGame().getDeck().pop();
```

```
1096                        } catch (ArrayIndexOutOfBoundsException e)
1097                        {
1098                            String exitStr = "Game over! ";
1099
                                getGame().getPlayerOne().setScore(getGame().getPlayer
                                One().getScore() + (int) score);
1100                            if (getGame().getPlayerOne().getScore() >
                                getGame().getPlayerTwo().getScore()) {
1101                                exitStr += getGame().getPlayerOne().getName() +
                                    " won with a score of " +
                                    (getGame().getPlayerOne().getScore());
1102                            } else {
1103                                exitStr += getGame().getPlayerTwo().getName() +
                                    " won with a score of " +
                                    (getGame().getPlayerTwo().getScore());
1104                            }
1105
1106                            dialog.dismiss();
1107                            gameOver(exitStr);
1108                        }
1109
1110                        String playerTurnStr =
                            getGame().getPlayerOne().getName() + " placed a tile at
                            ";
1111                        playerTurnStr += (row + 1) + ", " + (col + 1);
1112                        playerTurnStr += " for " + score + " points.";
1113                        getGame().getLog().setPlayerOneTurn(playerTurnStr);
1114
                            getPlayer1LogView().setText(getGame().getLog().getPlayerO
                            neTurn());
1115
1116                        // draw current tile, draw board, draw tile count, draw
                            score(s)
1117                        drawTile(getGame().getCurrTile(), getCurrTileLayout());
1118                        //drawBoard(getGame().getBoard(), getBoardLayout());
1119                        drawTileCount(getGame().getDeck(), getTileCountLayout());
1120                        drawPlayers(getGame(), getPlayer1NameLayout(),
                            getPlayer1ScoreLayout(), getPlayer2NameLayout(),
                            getPlayer2ScoreLayout());
1121
1122                        // if a standard game, computer's turn
1123                        if(m_solitaireFlag == false)
1124                        {
1125                            int test  = getGame().getDeck().getTiles().size();
1126                            if(test == 0)
1127                            {
1128                                String exitStr = "Game over! ";
1129
1130                                if (getGame().getPlayerOne().getScore() >
                                    getGame().getPlayerTwo().getScore()) {
1131                                    exitStr +=
                                        getGame().getPlayerOne().getName() + " won
                                        with a score of " +
                                        (getGame().getPlayerOne().getScore());
1132                                } else {
1133                                    exitStr +=
                                        getGame().getPlayerTwo().getName() + " won
                                        with a score of " +
                                        (getGame().getPlayerTwo().getScore());
1134                                }
1135
1136                                dialog.dismiss();
1137                                gameOver(exitStr);
1138                            }
1139
1140                            Turn computerTurn = new Turn();
1141                            Computer computer = new Computer();
1142
1143                            computerTurn =
```

```
                    computer.play(getGame().getCurrTile(),
                    getGame().getBoard());
1144                // set tile in board model, set current tile to
                    deck.top(), then pop deck
1145
                    getGame().getBoard().setTile(computerTurn.getRowPlaye
                    d(),
1146
                                                 computerTurn.getColPlayed
                                                 (),
1147                                             getGame().getCurrTile());
1148                View rowPlayedView =
                    getBoardLayout().getChildAt(computerTurn.getRowPlayed
                    ());
1149                View tilePlayedView = ((TableRow)
                    rowPlayedView).getChildAt(computerTurn.getColPlayed()
                    );
1150                drawTile(getGame().getCurrTile(), (ImageView)
                    tilePlayedView);
1151
                    getGame().getPlayerTwo().setScore(getGame().getPlayer
                    Two().getScore() + computerTurn.getPointsScored());
1152
1153                try {
1154                    getGame().setCurrTile(getGame().getDeck().top());
1155                    getGame().getDeck().pop();
1156                } catch (ArrayIndexOutOfBoundsException e)
1157                {
1158                    String exitStr = "Game over! ";
1159
1160                    if (getGame().getPlayerOne().getScore() >
                    getGame().getPlayerTwo().getScore()) {
1161                        exitStr +=
                        getGame().getPlayerOne().getName() + " won
                        with a score of " +
                        (getGame().getPlayerOne().getScore());
1162                    } else {
1163                        exitStr +=
                        getGame().getPlayerTwo().getName() + " won
                        with a score of " +
                        (getGame().getPlayerTwo().getScore());
1164                    }
1165
1166                    dialog.dismiss();
1167                    gameOver(exitStr);
1168                }
1169
1170                // draw current tile, draw board, draw tile count,
                    draw score(s)
1171                drawTile(getGame().getCurrTile(),
                    getCurrTileLayout());
1172                //drawBoard(getGame().getBoard(), getBoardLayout());
1173                drawTileCount(getGame().getDeck(),
                    getTileCountLayout());
1174                drawPlayers(getGame(), getPlayer1NameLayout(),
                    getPlayer1ScoreLayout(), getPlayer2NameLayout(),
                    getPlayer2ScoreLayout());
1175
1176                String computerTurnStr =
                    getGame().getPlayerTwo().getName() + " placed a
                    tile at ";
1177                computerTurnStr += (computerTurn.getRowPlayed() +
                    1) + ", " + (computerTurn.getColPlayed() + 1);
1178                computerTurnStr += " for " +
                    computerTurn.getPointsScored() + " points.";
1179                getGame().getLog().setPlayerTwoTurn(computerTurnStr);
1180
                    getPlayer2LogView().setText(getGame().getLog().getPla
                    yerTwoTurn());
```

```
                                }
                            } else {
                                // illegal move, do nothing but tell user
                                Toast.makeText(getApplicationContext(), "Illegal move!
                                Tiles must be placed adjacent to at least one matching
                                tile!", Toast.LENGTH_LONG).show();
                            }
                            dialog.dismiss();
                        }

                    })
                    .setNegativeButton("No", null);
            // resize AlertDialog, place it below current tile
            AlertDialog alertDialog = builder.create();
            alertDialog.show();
            WindowManager.LayoutParams lp = new WindowManager.LayoutParams();
            lp.copyFrom(alertDialog.getWindow().getAttributes());
            lp.width = 450;
            lp.height = 750;
            lp.x=720;
            lp.y=600;
            alertDialog.getWindow().setAttributes(lp);
        }
    }

    /**/
    /*
    GameActivity::quitGame()

    NAME

            GameActivity::quitGame - quits the Game Activity.

    SYNOPSIS

            public void GameActivity::quitGame();

    DESCRIPTION

            This function will exit the Game Activity and return to the Launcher
            Activity.

    RETURNS

            Void.

    AUTHOR

            Austin Fouch

    DATE

            1/30/2018

    */
    /**/
    public void quitGame()
    {
        AlertDialog alertDialog = new AlertDialog.Builder(this).create();
        alertDialog.setTitle("Quit");
        alertDialog.setMessage("Quit game?");

        alertDialog.setButton(AlertDialog.BUTTON_POSITIVE, "YES",
                new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int which) {
                        dialog.dismiss();
                        finish();
                    }
                });
```

```
1247
1248            alertDialog.setButton(AlertDialog.BUTTON_NEGATIVE, "NO",
1249                    new DialogInterface.OnClickListener() {
1250                        public void onClick(DialogInterface dialog, int which) {
1251                            dialog.dismiss();
1252                        }
1253                    });
1254
1255            alertDialog.show();
1256        }
1257
1258        /**/
1259        /*
1260        GameActivity::gameOver()
1261
1262        NAME
1263
1264                GameActivity::gameOver - displays the winner and their score, exits the
                    Game Activity.
1265
1266        SYNOPSIS
1267
1268                public void GameActivity::gameOver();
1269
1270        DESCRIPTION
1271
1272                This function will display an AlertDialog object to the screen, detailing
                    the winner
1273                of the game and their score. The function then exits the Game Activity,
                    returning to
1274                the Launcher Activity.
1275
1276        RETURNS
1277
1278                Void.
1279
1280        AUTHOR
1281
1282                Austin Fouch
1283
1284        DATE
1285
1286                1/30/2018
1287
1288        */
1289        /**/
1290        public void gameOver(String a_exitStr)
1291        {
1292            AlertDialog alertDialog = new AlertDialog.Builder(this).create();
1293            alertDialog.setTitle("Game Over");
1294            alertDialog.setMessage(a_exitStr);
1295
1296            alertDialog.setButton(AlertDialog.BUTTON_NEUTRAL, "OK",
1297                    new DialogInterface.OnClickListener() {
1298                        public void onClick(DialogInterface dialog, int which) {
1299                            dialog.dismiss();
1300                            finish();
1301                        }
1302                    });
1303
1304            alertDialog.show();
1305        }
1306    }
1307
```

```java
package austinfouch.com.ishido;

import java.util.List;
import java.util.Vector;

/**/
/*
    Human.java

    AUTHOR

        Austin Fouch

    DESCRIPTION

        Human class. Extends the Player class.

    DATE

        01/30/2018

*/
/**/
public class Human extends Player
{
    public Human()
    {
        super();
    }

    /**/
    /*
    Human::Human()

    NAME

            Human::Human - constructor for the Board class.

    SYNOPSIS

            public Human::Human();

    DESCRIPTION

            This function will construct a Human object. This is done by calling the
            Player
            constructor through the super() function.

    RETURNS

            No return value.

    AUTHOR

            Austin Fouch

    DATE

            1/30/2018

    */
    /**/
    public Human(String a_name, Integer a_score)
    {
        super(a_name, a_score);
    }
}

```

```java
package austinfouch.com.ishido;

/**/
/*
    IshidoColor.java

    AUTHOR

        Austin Fouch

    DESCRIPTION

        IshidoColor class used for enumerating the possible tile colors.

    DATE

        01/29/2018

*/
/**/
public enum IshidoColor
{
    WHITE, BLACK, BLUE, GREEN, RED, YELLOW, BLANK, HELP
}
```

```java
package austinfouch.com.ishido;

import java.lang.reflect.Field;

/**/
/*
    IshidoConstants.java

    AUTHOR

        Austin Fouch

    DESCRIPTION

        Constants class. Holds immutable constants for setting up an Ishido game.

    DATE

        01/30/2018

*/
/**/
public class IshidoConstants
{
    public final static int DECK_SIZE = 72;
    public final static int NUM_BOARD_ROWS = 8;
    public final static int NUM_BOARD_COLS = 12;
    public final static int UNQ_TILE_COUNT = 2;
}

```

```java
package austinfouch.com.ishido;

/**/
/*
    IshidoSymbol.java

    AUTHOR

        Austin Fouch

    DESCRIPTION

        IshidoColor class used for enumerating the possible tile symbols.

    DATE

        01/29/2018

*/
/**/
public enum IshidoSymbol
{
    BOLT, CIRCLE, CROSS, HEART, MOON, STAR, BLANK, HELP
}
```

```java
package austinfouch.com.ishido;

import android.app.AlertDialog;
import android.content.DialogInterface;
import android.content.Intent;
import android.graphics.drawable.GradientDrawable;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.text.InputType;
import android.view.View;
import android.widget.EditText;
import android.widget.ImageButton;
import android.widget.LinearLayout;

/**/
/*
    Launcher.java

    AUTHOR

        Austin Fouch

    DESCRIPTION

        Launcher Activity class. This Activity is the first screen the user sees. It has two
        buttons: Solitaire and Standard.

        Solitaire.onClick():
            The user is prompted to enter their name. The Launcher then attaches the name
            to as well as a solitaire flag, marked as true, to an Intent object. Then the launcher
            launches the Game Activity with the Intent object attached.

        Standard.onClick():
            The user is prompted to enter their name. The Launcher then attaches the name
            to as well as a solitaire flag, marked as false, to an Intent object. Then the launcher
            launches the Game Activity with the Intent object attached.

    DATE

        01/30/2018

*/
/**/
public class Launcher extends AppCompatActivity
{
    private String m_playerName = "";

    /**/
    /*
    Launcher::onCreate()

    NAME

            Launcher::onCreate - called when the Launcher Activity is created.

    SYNOPSIS

            public void Launcher::onCreate();

    DESCRIPTION

            This function will set the layout of the Launcher Activity to the XML file located
            at R.layout.activity_launcher.

    RETURNS
```

```
66
67              Void.
68
69      AUTHOR
70
71              Austin Fouch
72
73      DATE
74
75              1/30/2018
76
77      */
78      /**/
79      @Override
80      protected void onCreate(Bundle savedInstanceState)
81      {
82          super.onCreate(savedInstanceState);
83          setContentView(R.layout.activity_launcher);
84      }
85
86      /**/
87      /*
88      Launcher::Play()
89
90      NAME
91
92              Launcher::Play - launches the Game Activity as a Standard game.
93
94      SYNOPSIS
95
96              public void Launcher::Play();
97
98      DESCRIPTION
99
100             This function will prompt the user to enter their name. The Launcher then
                attaches the name
101             to as well as a solitaire flag, marked as false, to an Intent object. Then
                the launcher
102             launches the Game Activity with the Intent object attached.
103
104     RETURNS
105
106             Void.
107
108     AUTHOR
109
110             Austin Fouch
111
112     DATE
113
114             1/30/2018
115
116     */
117     /**/
118     public void Play(View view)
119     {
120         AlertDialog.Builder builder = new AlertDialog.Builder(this);
121         builder.setTitle("Standard: Enter Name");
122
123         // Set up the input
124         final EditText input = new EditText(this);
125         LinearLayout.LayoutParams lp = new LinearLayout.LayoutParams(
126                 LinearLayout.LayoutParams.MATCH_PARENT,
127                 LinearLayout.LayoutParams.MATCH_PARENT);
128         input.setLayoutParams(lp);
129         builder.setView(input); // uncomment this line
130
131         // Set up the buttons
132         builder.setPositiveButton("OK", new DialogInterface.OnClickListener()
```

```
133             {
134                 @Override
135                 public void onClick(DialogInterface dialog, int which) {
136                     m_playerName = input.getText().toString();
137                     Intent intent = new Intent(getApplicationContext(), GameActivity.class);
138                     intent.putExtra("playerName", m_playerName);
139                     intent.putExtra("solitaireFlag", "false");
140                     startActivity(intent);
141                 }
142             });
143 
144             builder.setNegativeButton("Cancel", new DialogInterface.OnClickListener()
145             {
146                 @Override
147                 public void onClick(DialogInterface dialog, int which) {
148                     dialog.cancel();
149                 }
150             });
151 
152             builder.show();
153         }
154 
155         /**/
156         /*
157         Launcher::PlaySolitaire()
158 
159         NAME
160 
161                 Launcher::Play - launches the Game Activity as a Solitaire game.
162 
163         SYNOPSIS
164 
165                 public void Launcher::Play();
166 
167         DESCRIPTION
168 
169                 This function will prompt the user to enter their name. The Launcher then
                    attaches the name
170                 to as well as a solitaire flag, marked as true, to an Intent object. Then
                    the launcher
171                 launches the Game Activity with the Intent object attached.
172 
173         RETURNS
174 
175                 Void.
176 
177         AUTHOR
178 
179                 Austin Fouch
180 
181         DATE
182 
183                 1/30/2018
184 
185         */
186         /**/
187         public void PlaySolitaire(View view)
188         {
189             AlertDialog.Builder builder = new AlertDialog.Builder(this);
190             builder.setTitle("Solitaire: Enter Name");
191 
192             // Set up the input
193             final EditText input = new EditText(this);
194             LinearLayout.LayoutParams lp = new LinearLayout.LayoutParams(
195                     LinearLayout.LayoutParams.MATCH_PARENT,
196                     LinearLayout.LayoutParams.MATCH_PARENT);
197             input.setLayoutParams(lp);
198             builder.setView(input); // uncomment this line
199 
```

```
200         // Set up the buttons
201         builder.setPositiveButton("OK", new DialogInterface.OnClickListener()
202         {
203             @Override
204             public void onClick(DialogInterface dialog, int which) {
205                 m_playerName = input.getText().toString();
206                 Intent intent = new Intent(getApplicationContext(), GameActivity.class);
207                 intent.putExtra("playerName", m_playerName);
208                 intent.putExtra("solitaireFlag", "true");
209                 startActivity(intent);
210             }
211         });
212 
213         builder.setNegativeButton("Cancel", new DialogInterface.OnClickListener()
214         {
215             @Override
216             public void onClick(DialogInterface dialog, int which) {
217                 dialog.cancel();
218             }
219         });
220 
221         builder.show();
222     }
223 }
224 
```

```java
package austinfouch.com.ishido;

/**/
/*
    Player.java

    AUTHOR

        Austin Fouch

    DESCRIPTION

        Player class. Holds name and score data and a function for determining a legal
        play.

    DATE

        01/30/2018

*/
/**/
public class Player
{
    private Integer m_score;
    private String m_name;

    /**/
    /*
    Player::isLegalPlay()

    NAME

            Player::isLegalPlay - given a Tile, Board.

    SYNOPSIS

            public Player::setScore(Integer a_score);
                a_score --> Integer value to set m_score to.

    DESCRIPTION

            This function will return an Integer value which represents the value of
            playing
            the given Tile object on the given Board object at the given position.

            If the value is greater than 0, the play can be interpreted as legal.

            If the value is 0, the play is illegal.

    RETURNS

            Integer.

    AUTHOR

            Austin Fouch

    DATE

            1/30/2018

    */
    /**/
    public Integer isLegalPlay(Tile a_currTile, Board a_board, int a_row, int a_col)
    {
        // if tile is not blank
        if(a_board.getTile(a_row,  a_col).getSymbol() != IshidoSymbol.BLANK)
        {
            return 0;
```

```
68          }
69
70          Integer value = 0;
71          // check tile match left
72          if( a_col != 0)
73          {
74              if( a_currTile.isMatch(a_board.getTile(a_row, a_col - 1)))
75              {
76                  value++;
77              }
78          }
79          // check tile match right
80          if( a_col != IshidoConstants.NUM_BOARD_COLS - 1)
81          {
82              if( a_currTile.isMatch(a_board.getTile(a_row, a_col + 1)))
83              {
84                  value++;
85              }
86          }
87          // check tile match above
88          if( a_row != 0)
89          {
90              if( a_currTile.isMatch(a_board.getTile(a_row - 1, a_col)))
91              {
92                  value++;
93              }
94          }
95          // check tile match below
96          if( a_row != IshidoConstants.NUM_BOARD_ROWS - 1)
97          {
98              if( a_currTile.isMatch(a_board.getTile(a_row + 1, a_col)))
99              {
100                 value++;
101             }
102         }
103         // double 4-way match value
104         if( value > 3)
105         {
106             value = value * 2;
107         }
108
109         return value;
110     }
111
112     /**/
113     /*
114     Player::getName()
115
116     NAME
117
118             Player::getName - getter for the m_name member variable.
119
120     SYNOPSIS
121
122             public Player::getName();
123
124     DESCRIPTION
125
126             This function will return the m_name member variable.
127
128     RETURNS
129
130             String.
131
132     AUTHOR
133
134             Austin Fouch
135
136     DATE
```

```
137
138             1/30/2018
139
140     */
141     /**/
142     public String getName()
143     {
144         return this.m_name;
145     }
146
147     /**/
148     /*
149     Player::setName()
150
151     NAME
152
153             Player::setName - setter for the m_name member variable.
154
155     SYNOPSIS
156
157             public Player::setName(String a_name);
158                 a_name --> String value to set m_name to.
159
160     DESCRIPTION
161
162             This function will set the n_name member variable to the value of a_name.
163
164     RETURNS
165
166             No return value.
167
168     AUTHOR
169
170             Austin Fouch
171
172     DATE
173
174             1/30/2018
175
176     */
177     /**/
178     public void setName(String a_name)
179     {
180         this.m_name = a_name;
181     }
182
183     /**/
184     /*
185     Player::getScore()
186
187     NAME
188
189             Player::getScore - getter for the m_score member variable.
190
191     SYNOPSIS
192
193             public Player::getScore();
194
195     DESCRIPTION
196
197             This function will return the m_score member variable.
198
199     RETURNS
200
201             Integer.
202
203     AUTHOR
204
205             Austin Fouch
```

```
206
207         DATE
208
209                 1/30/2018
210
211         */
212         /**/
213         public Integer getScore()
214         {
215             return this.m_score;
216         }
217
218         /**/
219         /*
220         Player::setScore()
221
222         NAME
223
224                 Player::setScore - setter for the m_score member variable.
225
226         SYNOPSIS
227
228                 public Player::setScore(Integer a_score);
229                     a_score --> Integer value to set m_score to.
230
231         DESCRIPTION
232
233                 This function will set the m_score member variable to the value of a_score.
234
235         RETURNS
236
237                 No return value.
238
239         AUTHOR
240
241                 Austin Fouch
242
243         DATE
244
245                 1/30/2018
246
247         */
248         /**/
249         public void setScore(Integer a_score)
250         {
251             this.m_score = a_score;
252         }
253
254         /**/
255         /*
256         Player::Player()
257
258         NAME
259
260                 Player::Player - copy constructor for the Player class.
261
262         SYNOPSIS
263
264                 public Player::Player(String a_name, Integer a_score);
265                     a_name  --> String value to set m_name to.
266                     a_score --> Integer value to set m_score to.
267
268         DESCRIPTION
269
270                 This function will construct a Player object given String and Integer values.
271
272         RETURNS
273
274                 No return value.
```

```
275
276         AUTHOR
277
278                 Austin Fouch
279
280         DATE
281
282                 1/30/2018
283
284         */
285         /**/
286         public Player(String a_name, Integer a_score)
287         {
288             this.m_name = a_name;
289             this.m_score = a_score;
290         }
291
292
293         /**/
294         /*
295         Player::Player()
296
297         NAME
298
299                 Player::Player - constructor for the Player class.
300
301         SYNOPSIS
302
303                 public Player::Player();
304
305         DESCRIPTION
306
307                 This function will construct a Player object.
308
309         RETURNS
310
311                 No return value.
312
313         AUTHOR
314
315                 Austin Fouch
316
317         DATE
318
319                 1/30/2018
320
321         */
322         /**/
323         public Player()
324         {
325             this.m_name = new String();
326             this.m_score = 0;
327         }
328     }
329
```

```java
package austinfouch.com.ishido;

/**/
/*
    Tile.java

    AUTHOR

        Austin Fouch

    DESCRIPTION

        Tile class. Hold information related to each Tile used in a standard game of
        Ishido.
            IshidoColor m_color     --> enumerated 'color' of the tile
            IshidoSymbol m_symbol   --> enumerated 'symbol' of the tile

    DATE

        01/29/2018

*/
/**/
public class Tile
{
    private IshidoColor m_color;
    private IshidoSymbol m_symbol;

    /**/
    /*
    Tile::Tile()

    NAME

            Tile::Tile - constructor for the Tile class.

    SYNOPSIS

            public Tile::Tile();

    DESCRIPTION

            This function will construct a Tile object.

    RETURNS

            No return value.

    AUTHOR

            Austin Fouch

    DATE

            10:26pm 1/29/2018

    */
    /**/
    public Tile()
    {
    }

    /**/
    /*
    Tile::Tile()

    NAME

            Tile::Tile - copy constructor for the Tile class.
```

```
69
70      SYNOPSIS
71
72              public Tile::Tile( IshidoColor a_color, IshidoSymbol a_symbol );
73                  a_color  --> the color of the Tile to be constructed.
74                  a_symbol --> the symbol of the Tile to be constructed.
75
76      DESCRIPTION
77
78              This function will construct a Tile object. The member variables of the
                constructed Tile
79              object, m_color and m_symbol, are set to the values of the respective
                parameters,
80              a_color and a_symbol.
81
82              The values are enumerated and defined in the IshidoColor and IshidoSymbol
                classes. The
83              values, combined as "color_symbol", correspond to PNG files in the
                res/drawable
84              directory.
85
86      RETURNS
87
88              No return value.
89
90      AUTHOR
91
92              Austin Fouch
93
94      DATE
95
96              10:26pm 1/29/2018
97
98      */
99      /**/
100     public Tile(IshidoColor a_color, IshidoSymbol a_symbol)
101     {
102         this.m_color = a_color;
103         this.m_symbol = a_symbol;
104     }
105
106     /**/
107     /*
108     Tile::setColor()
109
110     NAME
111
112             Tile::setColor() - setter for the m_color member variable of the Tile class.
113
114     SYNOPSIS
115
116             public void Tile::setColor( IshidoColor a_color);
117                 a_symbol --> the symbol to set m_color to.
118
119     DESCRIPTION
120
121             This function will assign the value of the member variable m_color to the
                value of the
122             parameter a_color.
123
124     RETURNS
125
126             Void.
127
128     AUTHOR
129
130             Austin Fouch
131
132     DATE
```

```
133
134                 10:26pm 1/29/2018
135
136         */
137         /**/
138         public void setColor(IshidoColor a_color)
139         {
140             this.m_color = a_color;
141         }
142
143         /**/
144         /*
145             Tile::setSymbol()
146
147             NAME
148
149                     Tile::setSymbol() - setter for the m_symbol member variable of the Tile
                        class.
150
151             SYNOPSIS
152
153                     public void Tile::setSymbol( IshidoSymbol a_symbol);
154                         a_symbol --> the symbol to set m_symbol to.
155
156             DESCRIPTION
157
158                     This function will assign the value of the member variable m_symbol to
                        the value of
159                     the parameter a_symbol.
160
161             RETURNS
162
163                     Void.
164
165             AUTHOR
166
167                     Austin Fouch
168
169             DATE
170
171                     10:26pm 1/29/2018
172
173         */
174         /**/
175         public void setSymbol(IshidoSymbol a_symbol)
176         {
177             this.m_symbol = a_symbol;
178         }
179
180         /**/
181         /*
182             Tile::getColor()
183
184             NAME
185
186                     Tile::getColor() - getter for the m_color member variable of the Tile
                        class.
187
188             SYNOPSIS
189
190                     public void Tile::getColor();
191
192             DESCRIPTION
193
194                     This function will return the value of the member variable m_color.
195
196             RETURNS
197
198                     Void.
```

```
199
200             AUTHOR
201
202                     Austin Fouch
203
204             DATE
205
206                     10:26pm 1/29/2018
207
208         */
209         /**/
210         public IshidoColor getColor()
211         {
212             return this.m_color;
213         }
214
215         /**/
216         /*
217             Tile::getSymbol()
218
219             NAME
220
221                     Tile::getSymbol() - getter for the m_symbol member variable of the Tile
                        class.
222
223             SYNOPSIS
224
225                     public void Tile::getSymbol();
226
227             DESCRIPTION
228
229                     This function will return the value of the member variable m_symbol.
230
231             RETURNS
232
233                     Void.
234
235             AUTHOR
236
237                     Austin Fouch
238
239             DATE
240
241                     10:26pm 1/29/2018
242
243         */
244         /**/
245         public IshidoSymbol getSymbol()
246         {
247             return this.m_symbol;
248         }
249
250         /**/
251         /*
252             Tile::isMatch()
253
254             NAME
255
256                     Tile::isMatch() - boolean function for determining if a tile matches
                        with this tile.
257
258             SYNOPSIS
259
260                     public boolean Tile::isMatch(Tile a_tile);
261                         a_tile --> the tile being compared to this tile.
262
263             DESCRIPTION
264
265                     This function returns True if the passed tile matches this tile, False
```

```
                otherwise.
266
267         RETURNS
268
269                 Boolean.
270
271         AUTHOR
272
273                 Austin Fouch
274
275         DATE
276
277                 10:26pm 1/29/2018
278
279     */
280     /**/
281     public boolean isMatch(Tile a_tile)
282     {
283         if(this.m_color == a_tile.getColor())
284         {
285             return true;
286         }
287         else if(this.m_symbol == a_tile.getSymbol())
288         {
289             return true;
290         }
291         else
292         {
293             return false;
294         }
295     }
296
297     /**/
298     /*
299         Tile::getColorResourceStr()
300
301         NAME
302
303                 Tile::getColorResourceStr() - used to determine the resource name
                    associated with
304                                               this tile's color
305
306         SYNOPSIS
307
308                 public String Tile::getColorResourceStr();
309
310         DESCRIPTION
311
312                 This function returns a String value that is the name of a resource
                    associated
313                 with this tile's color.
314
315         RETURNS
316
317                 String.
318
319         AUTHOR
320
321                 Austin Fouch
322
323         DATE
324
325                 10:26pm 1/29/2018
326
327     */
328     /**/
329     public String getColorResourceStr()
330     {
331         String resIDStr = "";
```

```
332             switch(this.m_color)
333             {
334                 case WHITE:
335                     resIDStr = "white_";
336                     break;
337                 case BLACK:
338                     resIDStr = "black_";
339                     break;
340                 case BLUE:
341                     resIDStr = "blue_";
342                     break;
343                 case GREEN:
344                     resIDStr = "green_";
345                     break;
346                 case RED:
347                     resIDStr = "red_";
348                     break;
349                 case YELLOW:
350                     resIDStr = "yellow_";
351                     break;
352                 default:
353                     break;
354             }
355 
356             resIDStr += "blank_tile";
357             return resIDStr;
358         }
359 
360         /**/
361         /*
362             Tile::getSymbolResourceStr()
363 
364             NAME
365 
366                     Tile::getSymbolResourceStr() - used to determine the resource name
                        associated with
367                                                    this tile's symbol
368 
369             SYNOPSIS
370 
371                     public String Tile::getSymbolResourceStr();
372 
373             DESCRIPTION
374 
375                     This function returns a String value that is the name of a resource
                        associated
376                     with this tile's symbol.
377 
378             RETURNS
379 
380                     String.
381 
382             AUTHOR
383 
384                     Austin Fouch
385 
386             DATE
387 
388                     10:26pm 1/29/2018
389 
390         */
391         /**/
392         public String getSymbolResourceStr()
393         {
394             String resIDStr = "blank_";
395             switch(this.m_symbol)
396             {
397                 case BOLT:
398                     resIDStr += "bolt_";
```

```
                break;
            case CIRCLE:
                resIDStr += "circle_";
                break;
            case CROSS:
                resIDStr += "cross_";
                break;
            case HEART:
                resIDStr += "heart_";
                break;
            case MOON:
                resIDStr += "moon_";
                break;
            case STAR:
                resIDStr += "star_";
                break;
            case HELP:
                resIDStr = "help_";
                break;
            default:
                break;
        }

        resIDStr += "tile";
        return resIDStr;
    }
}

```

```java
package austinfouch.com.ishido;

/**/
/*
    Turn.java

    AUTHOR

        Austin Fouch

    DESCRIPTION

        Turn class. Hold information related to each Turn taken in a standard game of
        Ishido.
            Tile m_tilePlayed       --> the tile played.
            Integer m_rowPlayed     --> the row the tile was played on.
            Integer m_colPlayed     --> the column the tile was played on.
            String m_playerName     --> the Player name who played the tile.
            Integer m_pointsScored  --> the points gained by the play.

    DATE

        01/29/2018

*/
/**/
public class Turn
{
    private Tile m_tilePlayed;
    private Integer m_rowPlayed;
    private Integer m_colPlayed;
    private String m_playerName;
    private Integer m_pointsScored;

    /**/
    /*
        Turn::Turn()

        NAME

                Turn::Turn - default constructor for the Turn class.

        SYNOPSIS

                public Turn::Turn();

        DESCRIPTION

                This function creates a Turn object.

        RETURNS

                No return value.

        AUTHOR

                Austin Fouch

        DATE

                10:26pm 1/29/2018

    */
    /**/
    public Turn()
    {
    }

    /**/
```

```
69      /*
70          Turn::Turn()
71
72          NAME
73
74                  Turn::Turn - copy constructor for the Turn class.
75
76          SYNOPSIS
77
78                  public Turn::Turn(Tile a_tilePlayed, Integer a_rowPlayed, Integer
                    a_colPlayed,
79                                    String a_playerName, Integer a_pointsScored);
80                      Tile m_tilePlayed       --> the tile played.
81                      Integer m_rowPlayed     --> the row the tile was played on.
82                      Integer m_colPlayed     --> the column the tile was played on.
83                      String m_playerName     --> the Player name who played the tile.
84                      Integer m_pointsScored  --> the points gained by the play.
85
86          DESCRIPTION
87
88                  This function creates a Turn object given values for each of a Turn
                    object's member
89                  variables.
90
91          RETURNS
92
93                  No return value.
94
95          AUTHOR
96
97                  Austin Fouch
98
99          DATE
100
101                 10:26pm 1/29/2018
102
103     */
104     /**/
105     public Turn(Tile a_tilePlayed, Integer a_rowPlayed, Integer a_colPlayed, String
        a_playerName,
106                 Integer a_pointsScored)
107     {
108         this.m_tilePlayed = a_tilePlayed;
109         this.m_rowPlayed = a_rowPlayed;
110         this.m_colPlayed = a_colPlayed;
111         this.m_playerName = a_playerName;
112         this.m_pointsScored = a_pointsScored;
113     }
114
115     /**/
116     /*
117         Turn::getPointsScored()
118
119         NAME
120
121                 Turn::getPointsScored - getter for the m_pointsScored member variable.
122
123         SYNOPSIS
124
125                 public Integer Turn::getPointsScored();
126
127         DESCRIPTION
128
129                 This function returns the m_pointsScored member variable.
130
131         RETURNS
132
133                 Integer.
134
```

```
135             AUTHOR
136
137                     Austin Fouch
138
139             DATE
140
141                     10:26pm 1/29/2018
142
143         */
144         /**/
145         public Integer getPointsScored()
146         {
147             return m_pointsScored;
148         }
149
150         /**/
151         /*
152             Turn::getTilePlayed()
153
154             NAME
155
156                     Turn::getTilePlayed - getter for the m_tilePlayed member variable.
157
158             SYNOPSIS
159
160                     public Integer Turn::getTilePlayed();
161
162             DESCRIPTION
163
164                     This function returns the m_tilePlayed member variable.
165
166             RETURNS
167
168                     Tile.
169
170             AUTHOR
171
172                     Austin Fouch
173
174             DATE
175
176                     10:26pm 1/29/2018
177
178         */
179         /**/
180         public Tile getTilePlayed()
181         {
182             return m_tilePlayed;
183         }
184
185         /**/
186         /*
187             Turn::getRowPlayed()
188
189             NAME
190
191                     Turn::getRowPlayed - getter for the m_rowPlayed member variable.
192
193             SYNOPSIS
194
195                     public Integer Turn::getRowPlayed();
196
197             DESCRIPTION
198
199                     This function returns the m_rowPlayed member variable.
200
201             RETURNS
202
203                     Integer.
```

```
204
205             AUTHOR
206
207                     Austin Fouch
208
209             DATE
210
211                     10:26pm 1/29/2018
212
213         */
214         /**/
215         public Integer getRowPlayed()
216         {
217             return m_rowPlayed;
218         }
219
220         /**/
221         /*
222             Turn::getColPlayed()
223
224             NAME
225
226                     Turn::getColPlayed - getter for the m_colPlayed member variable.
227
228             SYNOPSIS
229
230                     public Integer Turn::getColPlayed();
231
232             DESCRIPTION
233
234                     This function returns the m_colPlayed member variable.
235
236             RETURNS
237
238                     Integer.
239
240             AUTHOR
241
242                     Austin Fouch
243
244             DATE
245
246                     10:26pm 1/29/2018
247
248         */
249         /**/
250         public Integer getColPlayed()
251         {
252             return m_colPlayed;
253         }
254
255         /**/
256         /*
257             Turn::getPlayerName()
258
259             NAME
260
261                     Turn::getPlayerName - getter for the m_playerName member variable.
262
263             SYNOPSIS
264
265                     public String Turn::getPlayerName();
266
267             DESCRIPTION
268
269                     This function returns the m_playerName member variable.
270
271             RETURNS
272
```

```
273                 String.
274
275         AUTHOR
276
277                 Austin Fouch
278
279         DATE
280
281                 10:26pm 1/29/2018
282
283     */
284     /**/
285     public String getPlayerName()
286     {
287         return m_playerName;
288     }
289 }
290
```