Draw.fi

Project Proposal

Team 8
Alec Fong
Edward Han
Flavio Medrano
Katherine Ritchie
Tanuja Mohan

Table of Contents

High Level Requirements

This section of the document contains details and requirements for our project and an overall description of the behavior of our program, the Draw.fi applcation for Android.

Login

When the user taps on our application, she will be taken to a load view screen that displays our application name (Draw.fi) along with our logo (a blue cartoon dragon). Once the load screen clears, the user will see a login screen. The login screen will prompt the user for her username and password. If the user does not already have an account she can select the “New Account” option that allows her to make a new account. Once the user logs in he will be taken to the lobby screen where we will pair him with another player.

Estimated Time: 8 hours

Lobby

After the login screen, we will immediately begin the game with us first finding the user an opponent. The Lobby portion of the game is where the user will be paired up with another user that is also searching for another player.

If no other players are available at the time, a message will pop up to inform the user that no other players are currently searching for another player, and the user will need to choose to click one of two options: “Retry”, which allows the user to look for another player (this put the user as the highest priority out of all searching users) or “No thanks”, which will return the user to the login screen. The second option is equivalent to logging out of the game.

Canvas

The Canvas will be the main gameplay platform where the user draws her “Masterpiece”. The Canvas can only be accessed through Sandbox or through logging into the game and finding an opponent. The canvas consists of a bottom palette or a right menu that holds various drawing tools. The drawing tools that we will implement are a paintbrush (thin, medium, thick), an eraser (thin, medium, thick), and a color picker (red, orange, yellow, light green, light blue, blue, black, brown, green, purple, pink). At the top of the canvas will be a word describing what the teams have to draw. We will also have a timer in the canvas screen that limits the drawing time of each user to 30 seconds.

Estimated Time: 10 hours

Sand Box

If the user wishes not to log in or not create a new account, then the user can use our Sandbox. The Sandbox is a blank canvas where the user can doodle freely with the provided painting tools and options. In Studio mode, the user cannot save doodles. This mode is a means for a user to become familiar with the artistic tools and functionality provided in the drawing mode.

Estimated Time: 8 Hours

Points Total Screen

The final screen will display the number of points awarded to each player, and the total number of points accumulated for that user, with the option to play another game.

Estimated Time: 4 hours

Technical Specifications

activity_login_gui.xml

The login screen will be a vertical layout, consisting of our application's logo. At the top will be an TextView widget widget, containing the Drawfi text logo, with the Drawfi dragon logo below, 2 text fields or EditText widgets on top of each other below (first for username, second for password), 2 buttons laid out horizontally below: a Login button and a new account button, and a Sandbox button as the bottom-most widget. The Login button will authenticate users. When the player clicks the Login button, its listener will prompt them to the Lobby screen. This screen will have an animation of three moving dots to represent that the the game is loading and currently pairing the user to another random player online. Below the Login button will be a Sandbox button, for users who do not have an account. The logo and the buttons will all be aligned vertically.
Estimated Hours: 2 hours

- *Buttons: 5 min*
- *Graphics: 25 min*

LoginGUI.java

We will use Firebase for user authentication. If the user exists when they attempt to login, the username and password entered will be checked to see if exists on the Firebase database. Otherwise, they must create a new account. Users who do not have an account can use the app's Sandbox feature. When the player presses the Sandbox button, the screen transitions to a blank canvas with all tools at their disposal, but no topic or opponent.
Estimated Hours: 2 Hours

activity_sandbox.xml

The Sandbox is where any users can practice drawing, since it is nearly identical to the Main Canvas screen. In the center of the screen is the Canvas, a blank white rectangular area that will take up the majority of the screen. On the right of the Canvas will be a toolbar. The toolbar contains 3 vertically aligned buttons: three for differing widths of pencils for drawing (small, medium, large). At the very top of the screen will be a horizontal toolbar with a button on the left to return to the Login screen, and a label in the center that says “Sandbox”. *(This toolbar will be present in every following screen).* The bottom toolbar will be a color chooser, a grid 6 buttons wide by 2 buttons tall, displaying 11 different colors: red, orange, yellow, lime green, dark green, dark blue, cyan blue, purple, pink, brown, black.
Estimated Hours: 8 hours

SandboxCanvas.java

This will have the action listeners for all the drawing tools and draw methods to appear on the canvas. We will keep track of where the user is currently interacting on the Canvas screen using x and y coordinates and the program will accordingly show the pixels drawn in that location. The default drawing tool will be a pencil of medium thickness. When the user touches the screen, we will override what is already drawn at that given pixel on the Canvas screen.
Estimated Time: 8 hours
While the Canvas uses touch location, the toolbars use button listeners to distinguish which tool the user is selecting. The erase button will set the current pixel back to white, which is the color of the blank/unedited Canvas. The color and pencil thickness buttons will call different action listeners specific to each button, defined by the respective color or pencil thickness.
Thickness options: thin = 2 by 2 pixel, medium = 5 by 5 pixels, thick = 10 by 10 pixels.
Estimated Time: 8 hours

Tools:
Pencil - *Estimated Time: 1 hour*
Eraser - *Estimated Time: 1 hour*

Colors - Estimated Time: 1 hour

activity_lobby.xml

Like all screens in the game, the Lobby will have a toolbar at the very top with an icon in the far left for going “back” and returning to the Login screen. Under that toolbar will be a label notifying the user that the game is looking for another player to join the game, an an animation of 3 loading dots under it. When another player is found, that label will change to notify the user that another player has been found, and two buttons will appear in the center of the screen under the label. The buttons will be to join a game with this player, or to reject this player and wait for another. The Join button will bring the user to the Main Canvas screen. The Retry button will reset the label to notify the user that the program is searching for a player. As described below, this label will change after a certain amount of time to notify the user there are no other players in the lobby who can start a game. At this point, the two buttons will reappear but they will prompt the user to either keep waiting or logout.

Activity Fragments:

- *“Finding Player” fragment 1.5 hour*
- *“Accept Player” fragment 1.5 hour*
- *“No Players” fragment 1.5 hour*

Estimated Time: 6.5 hours (3 fragments plus Lobby layout)

LobbyActivity.java

The Lobby is linked to our database, which keeps track of who has already logged in and is currently “waiting” at the Lobby screen. The program will communicate to the server to receive a connection, and the person who is logged in will be the client (Player A). Now, the server will look for another logged-in player who is also ready to play (Player B). Player B’s information will be pulled from the database of available players and Player A will be notified “User found! Play against Player B?” and Player A can now choose to accept or wait to find another Player. If after 30 seconds of waiting, there is no confirmation from the other user, a message will come up to say “Player has not

responded. Find another player?" and the player matching search will begin again. The database will look for other users who log in and become available in the database. If Player B exits the lobby, this player's account information will be removed from the database and Player A will be shown a message "No users online. Try again?". By removing Player B from the database, the number of available players will be updated and Player A will be notified that there is no available user to play against, since only users at the Lobby screen searching for another player will have their account information in the database. The lobby database will be implemented using Firebase.
Estimated Time: 8 hours

activity_main.xml

The Main Canvas will be identical to the aforementioned Sandbox, except for two new widgets at the top of the canvas aligned in the middle. The 2nd top-most widget will be a timer counting down the remaining seconds the player has left to draw on the canvas. The top-most widget will be the label containing the game's keyword.
Estimated Time: 2 hour (for the features not present in the Sandbox: timer and keyword)

GameCanvas.java

This screen has the exact same functionality as the Sandbox, but with one additional feature: a timer widget which limits the drawing time for the user, as part of the game's functionality. When the timer is up, the user will not be able to edit the canvas or select any of the buttons from the toolbars. Along with the timer, a randomly selected keyword will be chosen for the players to draw. For example, the keyword could be "dolphin" or "pizza". We will hard code a set of keywords and randomly select one keyword out of this set to be displayed in the label at the top of the Canvas.
Estimated Time: 2 hours (this is for the features not present in Sandbox)

activity_final.xml

The player will be prompted to this screen after they have finished a game. The screen consists of a the user's points for the game, the total number of points linked to their account, and a button with the button text, "Play Again". When the player presses the play again button, its listener will trigger an event that leads them back to the LobbyGUI. If the player decides to press the back button instead, they will be led to the login screen.
Estimated Hours: 30 mins

Final.java

The number of points for a single game round as well as the total number of points that the user has accumulated will be retrieved from the database and displayed on the corresponding GUI. This class will implement a button listener for the Play Again Button, so the user may return to the lobby screen.

Point.java

Point is the most basic thing that the user can put on the screen. Points will get drawn when the user touches the screen. PointConnection inherits from this class.

PointConnection.java

This class is to fill in the points on the touchscreen so that the drawing doesn't look like MSPaint spraypaint. Without this class, the user would be drawing individual, unconnected points rather than lines.

PointGenerator.java

This class generates a random integer between 2 and 10 as the possible number of points that can be awarded to each player for a single game.

Detailed Design

Server Package:

Server.java:

Member Variables:

- private Vector<ServerThreads> serverThreads
- private int port
- GameCanvas gc
- int playerCounter;
- public static final String *TAG* = “xyz”
- int isReadyCount = 0;

Methods:

- run(): will be in charge of listening for messages from the Clients that are connected to the server
- sendDataToAllClients(); will be in charge of sending the Server’s messages to all of its clients. Such messages include:
    - If they are waiting to be matched up
    - Who their opponent will be
    - What the topic will be
- checkToSendMessages(): checks to see if other clients are available and the game is ready to play, and sends a message to all other clients

ServerThread.java:

Member Variables:

- private Socket s;
- private Server server;
- private OutputStream os;
- private InputStream is;
- private boolean isReady;

Methods:

- run(): checks to see if the server and all clients are ready and sends a message accordingly
- getStringFromInputStream(InputStream is): returns a string from the input stream
- sendMessage(byte b): writes a byte to the output stream

Client Package:

Client.java:

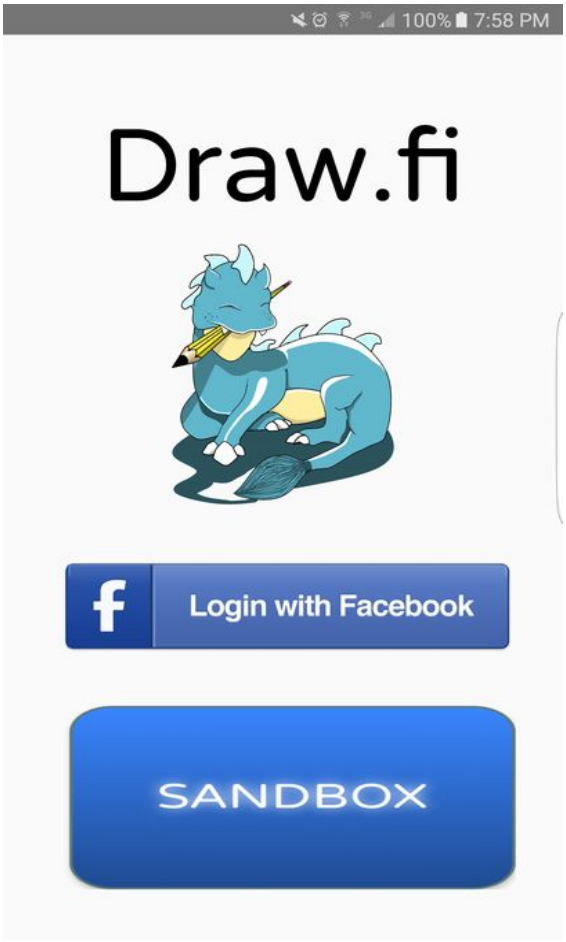
Takes in the client and socket

Methods:

- getReady(): returns a boolean that signals the activity that it can switch
- sendToServer(byte b): in charge of sending the message to the Server
    - The “message” includes if we have successfully logged in
        - If we are waiting for an opponent
        - If we have found an opponent
        - If we have started to play the game (will also send the keyword so that both Users will have the same keyword)
        - If we have reached TimesUp (see below)
        - If we have hit logged out button (see below)
- Run method: depending on the message we receive from the Server will use the information accordingly
- If the message we read in is the username then we will update the GameCanvas.java to display this username
- If we read the message that we have found an opponent then we will move to the GameCanvas.java screen
- Also, if the message we read in the the message that the TimesUp then we will store the Client’s drawing
- If we have logged out then we will will erase the data in our database that is associated with the user

Member Variables:

- String dstAddress;
- int dstPort;
- OutputStream os;
- InputStream is;
- private Socket socket;
- boolean readyToSwitch;
- public static final String *TAG* = “xyz”



activity_login_gui.xml

Parent Layout: RelativeLayout with white background

Widgets:

- All widgets are separated by some margin or spacing
- All widgets that allow for text will be a Varela Round font

  TextView

  - Aesthetics:
    - Black text
    - Size enlarged
    - Centered horizontally in the parent layout
  - Text: “Draw.fi”
  - Layout Location: Topmost widget

  ImageView

  - Aesthetics:

        - Image: a cartoon-style baby blue dragon holding a pencil in its mouth, and a paint brush for a tail
        - Details: Image was drawn using an image editing software called paint.net and saved as a .png file type, so that it can be imported via a Batch Drawable Import
        - Centered horizontally in the parent layout
    - Image source: A .png image file, whose source path is saved locally to the project in the drawable directory nested in the resources directory.
    - Layout Location: relatively below the Draw.fi TextView.

EditText (username)

- Hint: “Enter username”
- Located below Draw.fi dragon

EditText (password)

- Hint: “Enter password”
- Located below the username EditText

Inner RelativeLayout (nested)

- Layout specified to display, contain, and align buttons
- All widgets that are inside are centered within the layout as well
- Centered horizontally in the parent layout
- Layout Location: entire layout is located below the password EditText

  Button (Login)
    - Aesthetics:
        - Rectangular with rounded corners
        - Bold textStyle
        - White textColor
        - Size enlarged
        - Blue gradient background specified by mybutton.xml
        - Shadow effect (for text)
    - Text: “Login”

  Button (New Account)
    - Aesthetics:
        - Same as login button
    - Text: “New Account”
    - Located to the right of the login button

Button (Sandbox)

- Aesthetics
    - Rectangular with rounded corners
    - Bold textStyle
    - White textColor
    - Size enlarged
    - Blue gradient background specified by mybutton.xml
    - Shadow effect (for text)
- Text: “Sandbox”
- Layout Location: below Login and New Account buttons

LoginGUI.java

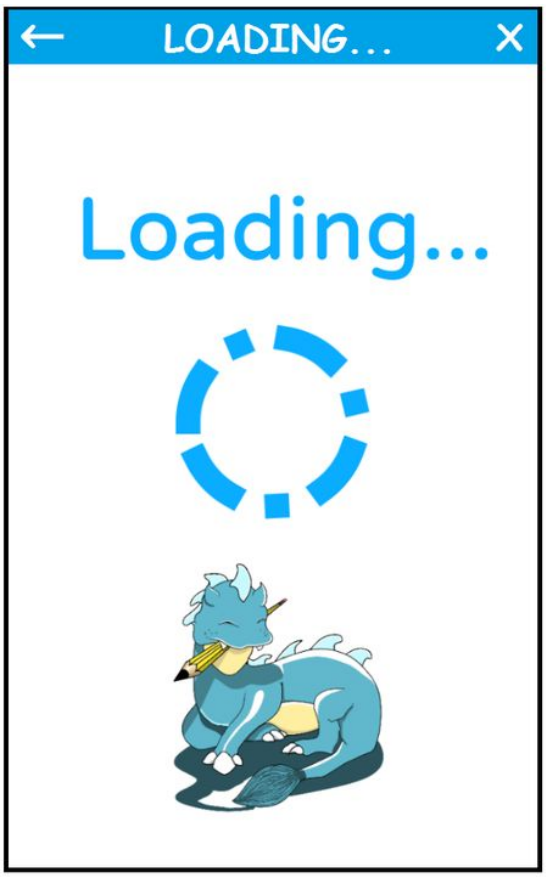
Methods:

- The onEditorAction Listeners will be assigned to the EditText fields and store the inputted text as strings
- The setOnClick Listeners will be assigned to the Login button, New Account button and Sandbox button and change the screen/activity state when clicked on.

Member Variables:

- *private Typeface custom_font*
    - we will be using “Varela Round” as our specific custom font for the theme of our logo and all other text throughout the project
- *private EditText usernameEdit*
    - a text field that allows the user to type their username in
- *private EditText passwordEdit*
    - a text field that allows the user to type their password in
- *private Button loginButton*
    - When the user clicks on loginButton, its listener will authenticate the user credentials from the text entered in *usernameEdit* and *passwordEdit*.
- *private Button createButton*
    - When the user clicks on createButton, its listener will check if an account exists from the text entered in *usernameEdit* and *passwordEdit*.
- *private Button sandboxbutton*
    - When the user clicks on sandboxButton, its listener will fire and trigger an intent to the canvas screen.
- *private Firebase database*
    - Stores an instance of a Firebase database

- *private DatabaseReference mReference*
    - Stores a reference to a Firebase database
- *private ImageView dragonView*
    - Stores the Draw.fi dragon into this Image View



activity_lobby.xml

- Implements Toolbar at top of this screen
- *TextView*
    - Aesthetics:
        - Light blue textColor
        - Size enlarged
        - Centered within parent layout
    - Text: “Welcome ” + (username) + “Finding opponent…”
    - Layout Location: Topmost Widget
- *Animation (Three Blue Loading Dots)*

        - Aesthetics:
            - First dot moves up and moves down at the same time, the second moves up. Second dot moves down at the same time third one moves up. When the third dot comes back down, animaiton repeats.
            - Centered within parent layout
        - Layout Location: below TextView
- *ImageView (Dragon)*
    - Aesthetics:
        - Image: light blue baby dragon (Draw.fi mascot)
        - Centered within parent layout
    - Layout Location: below Animation

LobbyActivity.java

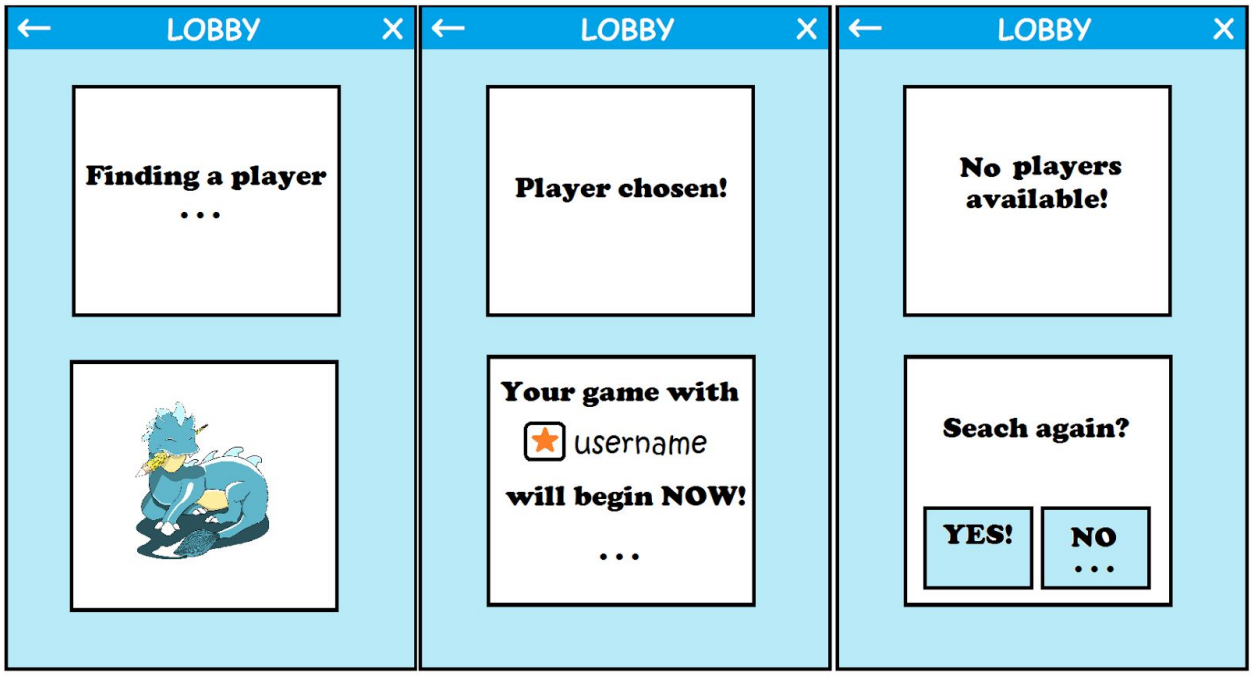
Data Members:

- public Button logoutButton;
- public TextView welcomePrompt;
- public TextView searchingPrompt;
- public FirebaseDatabase mDatabase;
- public DatabaseReference mReference;
- public Button retryButton;
- public Button noButton;
- String id;
- String mKey;
- String name;
- String surname;
- boolean isSecondUser = false;
- String player1 = null;
- String player2 = null;
- private Dialog settingGameDialog;
- private int counter = 0;
- View settingView;
- private AVLoadingIndicatorView avi;

Methods

- setKeyWord(): sets the keyword for each player

- nextActivity(): launches an intent to the game screen when the user has found an opponent and all other conditions have been met
- onOptionsItemSelected(): logs the user out when they click on the back button on the toolbar
- createTimer(): creates a timer that times out the user searching for an opponent after some time



Data Structure:

- Double Ended Queue to hold Users
- We will pull two players at a time from the queue and put them as opponents
    - If there is an odd number in the queue or no one other than the current player in the queue then we will wait
    - If after waiting 15 seconds another player gets pushed to the queue then we will give them an option to “Retry?”
        - Retry: puts the current user back into highest priority (places them in the beginning of the queue)

Methods of Double Ended Queue:

- Void addToLobby(User user): this pushes users who logged in, to the back of the Dequeue

- User findMatch(): if the Dequeue has more than two players, it will pop off the first two users from the front
- Void match(User user1, User user2): will connect user1 and user2 to server to play against each other in the game.
- Void retry(User user): This will push a user to the front of the dequeue. Since they have been waiting longer to start the game than a user who just entered the lobby, they will have priority to start a game first, hence being pushed back to the front of the deq


Draw.fi
DOG
0:53


Draw.fi
DOG
0:53


SANDBOX
hello

activity_main.xml

Widgets

- canvas - a DrawingView object depicting the user's drawing
- Tool buttons: largeWidthIcon, mediumWidthIcon, smallWidthIcon, eraserIcon - each tool button is a clickable ImageView depicting either the eraser or one of three pencil widths
- Color buttons (redButton, pinkButton, etc) - Each color button is a clickable ImageView aligned next to one another in a 2x6 grid
- hideIcon - an ImageView showing the color and tool (pencil width or eraser) that the user has currently selected

GameCanvas.java

Data Members:

- Server server;
- Client client;
- public static final String *TAG* = "xyz"
- private DrawView canvas;
- private Toolbar mToolbar;
- private Toolbar mToolbar1;
- private TextView keyword;
- private TextView timerTextView;
- private TextView clearButton;
- private ImageView smallBrush, mediumBrush, largeBrush;
- public int orangeColor, purpleColor, darkGreenColor, pinkColor, brownColor;
- public FirebaseDatabase mDatabase;
- public DatabaseReference mReference;
- String playerOpponent;
- String playerMe;
- long startTime = 0;
- int counter = 0;
- public String ipAddress;
- public String opponentAddress;

Methods:

- onCreate() - connects all widgets to local member variable instances, generates random keyword and sets the label accordingly, starts timer.
- onClick listeners for each color button: sets mColor, mSize, mType
- onTouchListener for canvas - This method shows lines of specific color and width where the user has touched the screen.
- Overriden methods from DrawingView: onSizeChanged(), onDraw(), onTouchEvent(). These methods change the pixels on the canvas in conjunction with the listener.
- hideIconListener - hides or shows all tool and color ImageViews

  Child Class: SandboxCanvas.java
  (identical to Canvas.XML without timer and keywordLabel)

  Child Class: activity_sandbox.XML
  Identical to Canvas.java utilizing all methods but timeUp() and generateKeyword()

  Child Class: GameCanvas.java
  Member Variables:
  - keywordTextView - Shows the randomized keyword
  - timer - Timer that counts down how much longer users have to draw

  Methods
  - generateKeyword(): from our database of words, we will randomly select one word for both the Users to use. Each word will have a primary key ID ranging from 1 to n where n is the number of words in our table. We will randomly generate a value between 1 and n and grab the word whose primary key ID is this number
  - timeUp() - When the timer runs out to 0 seconds, all ImageViews become unclickable, and Toast comes up that the user has run out of time. When the Toast disappears, the Loading activity launches, which will be followed by the ViewImage activity.

DrawView.java

Member Variables:

- List<Point> allPoints = new ArrayList<Point>();
- public int orangeColor, pinkColor, purpleColor, brownColor, darkGreenColor;
- private Paint paint = new Paint();
- private int paintColor;
- private int width;

Methods:

- private void init();
- public void forceRedraw();
- public void changeBrushSize(int size);
- public void clearCanvas();
- public void onDraw(Canvas canvas);
- public void changeColor(int c);
- public boolean onTouch(View view, MotionEvent event);

EndGameActivity.java

Member Variables:

- private String myScore;
- private String opponentScore;
- private DatabaseReference mReference;
- private String myName;
- private String opponentName;
- private TextView myPoints;
- private TextView theirPoints;
- private TextView win;
- public FirebaseDatabase mDatabase;

Methods:

- onCreate();

Message.java

Variables:

- private String name;
- private Boolean isDone;
- private int points;
- private int order;

Methods:

- public String getName();
- public boolean isDone();
- public int getPoints();

Player.java

Variables:

- private String uid;
- private String firstName;
- private String lastName;
- private String opponentKey;
- private String mKeyword;
- private String matched;
- private Boolean isSecond;
- private int totalScore;
- private String ipAddress;
- String key;

Methods:

- Getters and setters implemented for each variable.

Point.java

Variables:

- public float x, y;
- public int color, width;

Methods:

- draw(final Canvas canvas, final Paint paint);
- isSamePoint(Point p);

PointConnection.java

Variables:

- private final Point neighbor;

Methods:

- draw(final Canvas canvas, final Paint paint);

PointGenerator.java

Variables:

- private int pointVal = 0;

Methods:

- getPoint();

activity_end_game.xml

- TextView: “Game Over!”
    - Topmost widget
- TextView: “You Won!” or “You Lost!”
    - Below game over text view
- Textview: “Your score: ” + user’s score
    - Below win/lose text view
- Textview: “Opponent score: ” + opponent’s score
    - Below user’s score
- TextView: “Your Current Stash: ” + user’s current stash of points
    - Below opponent’s score:
- ImageView: Draw.fi dragon
    - Below stash text view

Testing Document

This document lists our application's test cases based on type: front-end test cases pertaining to GUI screens are categorized by Android Activity, while back-end test cases pertaining to login authentication and image database management issues are categorized under those respective headers.

Number of Tests: 55

Toolbar (all screens)

- The Toolbar is present at the top of every screen of the app (except the Login screen).
- Clicking the Back button (arrow image) quits the current activity and changes the screen to the Login Activity screen.
- The label in the middle of Toolbar accurately describes the current screen.
- The label in the middle of the Toolbar is centered horizontally.
- The label in the middle of Toolbar updates immediately when the screen changes ("Lobby", "Loading", "Draw.fi", etc.).

Login Screen

- All widgets/items fit within the margins of the login screen and are not cut-off in any way.
- All widgets/items that use text display a coherent font and are respectively aligned and centered.
- The Draw.fi logo is correctly displayed as the top item, with the correct font and black text color.
- The Draw.fi dragon image is correctly displaying below the Draw.fi logo and centered both vertically and horizontally.
- The text fields are correctly loaded below the Draw.fi dragon.
- The Login and New Account buttons are next to each separated by a space in between, and both buttons are correctly displayed below the text fields.

- The sandbox button has a blue background with white text correctly displaying, titled: “Sandbox”, and is the bottom-most widget.

Canvas (either Sandbox or Game screen)

- Upon screen creation, the colors palette grid loads each color swatch image (red, orange, yellow, etc. + the eraser).
- Upon screen creation, the tools grid loads each tool icon (large width, medium width, small width).
- Upon screen creation, the default draw settings are color: black, width: small and this can be confirmed when the user begins to draw.
- Upon screen creation, the clear button is correctly displayed towards the left hand margin as an X icon.
- The selected color is the correct color produced as user draws.
- The selected pencil width is the width produced as the user draws.
- Any area on the canvas touched by the user changes to the most recently selected color in the most recently selected width, even if the area has been drawn on before.
- If the eraser is selected and the user draws, the pixels on the screen touched by the user change to the color white.
- The clear button correctly clears the canvas when clicked on.
- For the sandbox mode, the keyword and timer should both not appear at the top of the canvas.

Game Screen

- A randomized keyword appears in a label at the top of the canvas.
- A timer appears at the top of the canvas, below the keyword.
- The timer counts down from thirty until zero.
- When the timer reaches 0, the screen displays a message labeled “Time’s up!” and changes to the final game screen.

Lobby Screen

- Upon creation, the top fragment loads the text “Welcome “ + (player’s username) + “Finding a player…” into its label.
- Upon creation, the animation of 3 blue loading dots should be in motion and appear correctly below the label and be centered horizontally within the screen.
- Upon creation, the bottom fragment loads a graphic into its ImageView.
- After 30 seconds of waiting, both the top and bottom fragments change to show the current state of finding a player (as described below). The screen MUST change after 30 seconds.
- If a player is not found: (Below are individual test cases)
    - the top label changes text to “No players found!”
    - The bottom fragment should now show two buttons side by side centered horizontally within the layout, both below the top label.
    - The left button should correctly display: “Retry”
    - The right button should correctly display: “No thanks”
    - Clicking the “No thanks” button returns the user to the Login Activity screen.
    - Clicking the Retry resets the text in the top fragment label to be “Finding a player…” and resets the bottom fragment to load the dragon mascot image into the ImageView.
- If a player is found: (Below are individual test cases)
    - A dialog box appears in the center of the screen laid on top of the underlying screen.
    - The dialog box displays the text: “Opponent found”
    - The dialog box displays its own animation below the text of alternating motions between a blue triangle revolving horizontally and flipping vertically along the x-axis.
    - After a player and opponent have matched, the app launches the Lobby Screen in transition to the Game Screen Activity

========== BACKEND STARTS HERE ==========

Login

- Check if a user can successfully login or create a new account
    - Login Case:
        - If a user does not input both a username and password, or inputs a username but not a password and vice-versa, and attempts to login, a Toast pops up indicating that this login attempt was invalid.
        - If a user inputs both a username and password and an account has been created for the inputted credentials, then the user should successfully be transferred to the lobby.
        - If a user inputs both a username and password and an account has not been created for the inputted credentials when logging in, then a Toast pops indicating that an account does not exist for the inputted credentials.
    - New Account Case:
        - If a user does not input both a username and password, or inputs a username but not a password and vice-versa, and attempts to create a new account, a Toast pops indicating that this account creation attempt was invalid.
        - If a user inputs both a username and password to create a new account and an existing account does not exist for the inputted credentials, then the user is logged in and transferred to the lobby screen.
        - If a user inputs both a username and password to create a new account and an existing account already exists for the inputted credentials, then a Toast pops up indicating that an account already exists for the inputted credentials.

Server (Lobby)

- For 15 seconds, keep checking the dequeue for another player if there is only one player in the queue

        - Case 1: If there are no other users in the dequeue
            - This user get add to the empty dequeue and waits for another user to enter the dequeue
        - Case 2: If there is a single other user already in the dequeue
            - The incoming user should be paired with this single user and move on the GameCanvas screen. Both users should be removed from the dequeue resulting in a now empty dequeue
        - Case 3: If there is more than one user already in the dequeue
            - The new user should be placed at the end of the dequeue and wait until it becomes either the first or second user in the dequeue (this then moves back to Case 1 and Case 2)
    - If Case 1 occurs for full wait time of 15 seconds
        - Case 1a: If the user presses “Try Again”
            - The user should be placed again into the dequeue this time in the very front giving it highest priority. The wait time should reset
        - Case 1b: If the user decides to click the “Back” button in the toolbar
            - The user should be taken back to the Login Activity screen. The user should be disconnected from the Server and should no longer be in the dequeue.
        - Case 1c: If the user decides to click the “Exit” button in the toolbar
            - The application should completely quit. The user should be disconnected from the Server and should no longer be in the dequeue. The next time the application starts it will take the user to the login screen.

Database

- Adding a user to a database results in an insertion with correct information
- Retrieving user information is correct
- Retrieving keyword results in random keyword

Deployment Document

This document details step-by-step instructions to deploy our application in a different environment (Android Studio).

Step 1: Download Android Studio

1. Download and install Java 8 at http://java.com
2. Download Android Studio for the correct operating system (Windows, Mac OS X, Linux) at https://developer.android.com/studio/index.html
3. Install Android Studio
    a. Windows:
        i. Launch the .exe file just downloaded and follow the setup wizard to install any necessary SDK tools. **(Important: It will show the location where it will install the SDK—take note of this since you will need it later)**
        ii. On some Windows systems, the launcher script does not find where Java is installed. If you encounter this problem, you need to set an environment variable indicating the correct location.
            1. **Start menu > Computer > System Properties > Advanced System Properties.**
            2. Open up **Advanced tab > Environment Variables** and add a new system variable JAVA_HOME that points to your JDK folder. For example: C:\Program Files\Java\jdk1.7.0_21
    b. Mac:
        i. Launch the .dmg file just downloaded
        ii. Drag and drop Android Studio into the Applications folder.
        iii. Open Android Studio and follow the setup wizard to install any necessary SDK tools.
        iv. Depending on your security settings, when you attempt to open Android Studio, you might see a warning that says the package is damaged and should be moved to the trash. If this happens, go to **System Preferences > Security & Privacy** and under **Allow applications downloaded from**, select **Anywhere**. Then open Android Studio again.
4. Install SDKs for each API version of Android you want to use, as well as additional tools that will be helpful.
    a. Start a new project (Need to open/access Android Studio's AVD manager, so settings do not matter)

b. Once Android Studio is open, go to **Tools > Android > SDK Manager**
c. Under the **SDK Platform** tab, make sure the following are checked:
    i. Android 6.0 (Marshmallow)
    ii. Android 5.1 (Lollipop)
d. Under the **SDK Tools** tab, make sure the following are checked:
    i. Android SDK Built Tools
    ii. Android SDK Platform-Tools X.X.X (*version number*)
    iii. Android SDK Tools X.X.X (*version number*)
    iv. Documentation for Android SDK
    v. Google Play Services, rev X (*version number*)
    vi. Google Repository, rev X (*version number*)
    vii. Google USB Driver, rev X (*version number*)
    viii. Intel x86 Emulator Accelerator (HAXM installer), rev X (*version number*)
e. Click **OK** once done

5. Configure Application Testing Platforms (2 options)
    a. Real Android Device:
        i. On your android device, go to **Settings > About phone** and tap **Build number** seven times.
        ii. Return to the previous screen to find that you have enabled **Developer options**.
    b. AVD Emulators (Android virtual devices)
        i. Start a new project (Need to open/access Android Studio AVD manager, so settings do not matter)
        ii. Once Android Studio is open, go to **Tools > Android > AVD Manager**
        iii. On the next window, choose **Create Virtual Device**
        iv. Choose **Phone > Nexus 6P (any 5.7" screen)**
        v. Choose **Release Name: Marshmallow** and **ABI:x86_64**
        vi. Choose **Finish**
6. Enable Hardware Acceleration
    a. After downloading and installing all the SDKs and tools, install the Intel Hardware Acceleration Execution Manager (HAXM), located within the SDK folder path from part 3):
        i. Windows: **SDK_LOCATION**\extras\intel\Hardware_Accelerated_Execution_Manager/intelhaxm-android.exe
        ii. Mac: **SDK_Location**/extras/intel/Hardware_Accelerated_Execution_Manager/intelhaxm-android.dmg
    b. Run the program and follow the on-screen instructions (Mac users may first have to double click on **IntelHAXM.mpkg** icon to begin installation).

Step 2: Open project

1. Download/pull and clone the project/code from the source (GitHub repository) into the desired local directory.
2. If the folder is zipped in a compressed folder, extract the folder and unzip it.
3. Open Android Studio and under **File > Open**, navigate to the directory containing the cloned repository/project folder
4. Look for a green circle icon and your project name next to it. Click on it and press **OK**, or double click on it. (Do not click on any of the nested folders/components)
    a. Note: An error may pop up saying the project path is incorrect. Simply press **OK**, since it will change the source path to your local path directory.
5. Wait for Android Studio to load all the files. (May take some time)
6. Once the project has finished loading, you will see a message the bottom left-hand corner: "Gradle build finished in _s ___ms (__ minutes ago)"

Step 3: Run Program

Once you are ready to run the application, you can test it on either a real Android device, or through the AVDs provided by Android Studio.

Real Android Device

- Required items:
    - Android device (with **Developer options** enabled)
    - Complementary charging USB cable

Run your Program

1. Go to Android Studio and look for a green triangle button (under the **VCS** tab) at the top and click on it
    a. Alternatively, under the **Run** tab, click on **Run 'app'**
2. The deployment window will pop up along with a list of connected devices and any emulators you have created. Select the deployment target you wish to use.
    a. If you are running the program on a connected device, your Android device and computer must be connected with the USB cable.
3. Wait for the project to build all necessary components and finish compiling. (First time building usually takes a couple of minutes)
4. Once Android Studio has finished compiling, your application will automatically be installed and launched.
5. Your application is ready for use and testing.