Final Android Project – Documentation
EC 327 Introduction to Software Engineering, Fall 2015
Due December 11th, 2015

“BeNice”

by Eric Mooney, Dor Rubin, Philip Yuan

Table of Contents

I. Our Goal

BeNice is an app designed to create an exponential growth of good feelings throughout a user’s network of friends and family, inevitably spreading across the globe due to its inherently “viral” gameplay (and the 6-degrees of separation effect). In a world where we are constantly bombarded with negativity and superficiality, BeNice will bring back the warm glow of thoughtful humanity.

We also hope that BeNice reminds users of forgotten friends and acquaintances and guide in kickstarting conversations through the use of an entertaining platform.

II. Application Overview

Be Nice is an app made to let users easily spread friendly feelings and good vibes via text messages through fun gameplay.

BeNice is an Android Application developed on the Android Studio platform by Eric Mooney, Dor Rubin, and Philip Yuan as their final project for Boston University's EC327 Fall 2015 Course. The application consists of the several pages, which guide the user through the "BeNice Experience". In a simple and intuitive manner, the user moves beyond the Title Screen and onto the Game Page, where they are presented with 15 random "Compliments" and 3 randomly-selected members of their phone's Contacts.

The user can easily browse through the available Compliments using an intuitive carousel-style slider in the upper half of the screen, until they find an appropriate message for one of the pre-populated Contacts. Once a match is made by clicking on the yellow "BeNice" button, the compliment is delivered by text message sent from our app to the user's Contact.

In order to complete the game, they must send 3 Compliments during the session. If none of the Contacts are "worthy" of a Compliment, the user has 3 available "BeMean" selections by which they can reload a new random Contact. If 3 Compliments are sent, then the user gets redirected to a congratulations page which shows the history of the compliments sent to which contacts during that round. If, however, they had to use the "BeMean" option 3 times, they are redirected to a lose page.

III. Audience And Marketability

BeNice was designed for the vast majority of phone-users. With the rise of instant communication technology came a world filled with bad news; the bustle of modern life also makes little time for people to connect with the multitude of acquaintances, contacts, and friends amassed from life--both online and offline--which can sometimes total in the hundreds.

BeNice helps users from this demographic spread some fun and friendly vibes to a couple of contacts at a time through a simple-to-use and fun-to-play mini game.

Through the use of automatically generated compliments, BeNice can help break the ice to start conversations or act as a system to remind someone that you're thinking of them.

Because the game ultimately results in sending messages to multiple contacts per round, BeNice has an innate ‘viralness’ to it: if the app catches on, it easily spreads! For every round of the game, 3 more people get directed to our app. Assuming that every new user plays 1 round and sends to 3 unique contacts, then by the 10th wave of users (3^x users, with x being corresponding to each “wave” of users), our app will already have nearly 60,000 users. By the 20th wave, 3 billion users. Our map is self-marketing and will likely draw plenty of attention on both traditional and social media.

IV. Technical Documentation

The program is written primarily in Java files along with XML resources handling the user interface. There is extensive use of Twilio libraries on both the client and server side. There following pages and functions manage the user flow and gameplay:

A. Activity/Page Descriptions

- **Title Screen**
  - Goal: To welcome the user to the application and intuitively direct them to the GamePage
    - activity_title_screen.xml
      - Simple start menu with imageview to display logo and button widgets
      - Uses custom designed logo
    - TitleScreen.java
      - Button listeners
      - Initialization and use of custom font through Typeface class

- **Instructions**
  - Goal: To provide concise gameplay instructions in the simplest terms
    - activity_instructions.xml
      - Simple textview widget for instruction display
    - Instructions.java
      - Initialization and use of custom font through Typeface class

- **Game Page**
  - Goal: To allow user to create matches of Compliments to Contacts
    - activity_game_page.xml

        - Use of various widgets including textswitcher, buttons, and imageviews
        - Widgets positioned using linear, frame, and the default relative layouts
        - Uses custom graphics and icons

- GamePage.java
    - General
        - Initialization and declaration of arrays and Nice/Mean point counters
    - Important Classes
        - **Compliment.java**
            - Custom class with two member variables
                - String _id - stores a unique id for the compliment
                - String _message - stores the message component of the compliment (e.g. “You are a great friend!”)
            - Two member “getter” functions
                - public String get_id()
                - public String get_message()
            - static public ArrayList<Compliment> loadCompliments(ArrayList<Compliment> compliments_list)
                - takes in an arrayList and adds new arrayList<Compliment> class objects to the that arrayList
        - **Contacts.java**
            - Custom class with 3 member variables
                - String _id - stores a unique id for the contact
                - String _name - stores the contact’s name
                - String _phone - stores the contact’s phone number
            - Three member “getter” functions
                - public String get_id()
                - public String get_name()

        - public String get_phone()
- **Globals.java**
    - Custom class (built around a Singleton model) that acts as a way to store global variables.
        - Used to store the contacts/compliments clicked by the user on the GamePage, and then displayed again in the NiceGameOver page.
    - Three private variables
        - ArrayList<Contact> contacts_sent;
        - ArrayList<Compliment> messages_sent;
        - static Globals instance;
            - To insure that only one instance is created globally during the session.
    - One private constructor
        - Globals(){}
    - Public member functions to get/set contacts and compliments, as well as to clear these ArrayLists when the user starts a new game
        - void add_contacts_sent(Contact contact)
        - void add_messages_sent(Compliment compliment)
        - ArrayList<Compliment> get_messages_sent()
        - ArrayList<Contact> get_contacts_sent()
        - void clear_contacts()
        - void clear_messages()
    - A Public method of obtaining the instance of the Globals class, which has access to the Globals constructor, and makes sure that the constructor is only used when there isn't already an instance created.
        - static synchronized Globals getInstance()
- **Sms.java**
    - HttpHelper.java

            - Inherited from android-twilio sdk
                - Device
                - Connection
- Functions
    - **init()**
        - initialization of various buttons, viewers
        - sets custom font through Typeface class
    - **animation()**
        - initialization and use of ‘swiping’ animations (left-to-right and right-to-left animations) taken from assets and put to use for all text viewers
    - **Click()**
        - button listeners for on-click actions
        - increase of counters and detection of win/lose conditions leading to a new activity
        - changing of styles for various pictures and texts on-click to notify player of his actions
    - **set_Factory()**
        - used for textswitcher widgets to create the textview boxes
        - truncates strings with ellipses if contacts or compliments are too long to show
    - **onCreate()**
        - sets the view to the activity_game_page
        - loads the compliments from the Compliments.loadCompliments() function
        - runs processContacts() to obtain the phone’s contact list
        - uses a random number generator to shuffle all of the contacts and compliments, and then creates the necessary arrayLists for use in the view
    - **processContacts()**
        - utilizes the Android “ContentResolver” class to query all of the phone’s contacts.
        - The function iterates over each of the contacts and stores the id, phone number, and name of each contact that has at least 1 phone number
        - returns the completed contacts_all arrayList

- **Game Over**
    - Goal: To congratulate the user, show text message history on 'win' page, further reinforce the good feelings generated by the experience of the GamePage, and encourage the user to play again
        - activity_nice_game_over.xml, mean_game_over.xml,
            - Win/lose graphics, text, and 'play again' buttons
        - NiceGameOver.java, MeanGameOver.java
            - NiceGameOver.java displays user's match history for the round

B. Twilio Integration

- **Overview**
    - Due to security concerns, Twilio does not allow Android and iOS developers to implement straight up Http Requests from clients. The reason is that each client would contain a copy of the API Key and Secret
    - The work around is building a simple python server that handles requests from clients and forwards them to Twilio on the client's behalf
    - Use of the java-sdk and apache sdks were restricted to prevent developers from doing the above.
    - Instead android-twilio sdk is provided with unique methods and dependencies.

- **Step-By-Step Interactions**
    - Initialization
        - In GamePage.java, a Sms class instance is initialized and defined as 'phone'
        - Also initializes Device and Connection objects used to define the client's connection to Twilio
        - Client gets access token generated by an endpoint hosted by the server
        - Token defines the app's permission (Voice or SMS) as well as for the period
    - phone.message method
        - on click, the method .message is invoked on the phone instance
        - contact's name, phone number and compliment are passed in as arguments

        - Device.connect(parameters) hits the Heroku server with a request at the '/call' route where a call is mocked and a message is created and sent off to the number included in the parameters
        - This is a hack because the Twilio documentation did not specify how to send an SMS through Android.
        - All SMS's are sent from a phone number purchased and assigned on Twilio

- **What Works And Doesn't**
        - By default we have commented out the code that sends a message. This is because it is very buggy and causes stability issues with the app.
        - However, you can send a message to a hardcoded number (since the emulator does not contain real numbers)
        - Note that code to pull the number from the contact is there but commented out for testing
        - To test the SMS:
            - Uncomment line 404 (I swear this was completely coincidental) in GamePage.java.
            - Replace the number string in the first parameter with the number you would like to message
            - Build the app and deploy.
            - Once running select to be nice to the second contact. If the app does not crash, they will receive a message.
            - No other contacts you select to send a message to will receive one until the connection is closed and the app is reloaded

- **Additional Documentation**
    - server.py
        - Located at: vast-basin-8313.herokuapp.com
        - Code (excluding account ID and secret) is included in server-resources directory

C. Font/Graphic Licenses

"Bangers" Font by Vernon Adams: www.1001fonts.com

Licensed under the SIL Open Font License (OFL) which allows commercial and non-commercial use of the font.

Graphics and Logos: https://openclipart.org/

All project graphics designed and patched together using various clipart from above website.

All clipart are released into public domain and allows for unlimited commercial and non-commercial use.

V. Further Enhancements and Growth

BeNice acts as a solid base for many enhancements and can grow in a number of potential directions.

The first aspects we can improve upon would be the compliment variety. A system could be put in place which allows for users to select a category of compliments centered around themes, such as “family” or “love” or “friend” which would enhance the likelihood of sending more relevant, targeted compliments.

Also improvements could be made on contact detection to favor contacts who you haven’t contacted in a while (either via phone or text), allowing the player an option to turn this feature on/off. Of course since contacts on the user’s phones might not be actual people but business and restaurants, we can include some form of detection to detect and remove these as potential contacts in our game.

Gameplay wise, we can improve on the interactiveness through the use of more advanced integrated widgets to further take advantage of the touch screen such as a using slider (like a swiping action) in order to choose between messages. Additionally, we can include a more ‘punishing’ lose page but in a way that is thematically similar to the purpose of the app. For example, on a loss, we might force-send a random friendly message to a random contact on the person’s phone, or make the random send an option to “Redeem” themselves.

To increase marketability we could also embed a link to our app in the text messages sent, so that the recipient has an easier time finding our app.

Have fun and be nice!