MODS Trek Final Report

Mobile Apps for Google Android

By: Group 3: Roshon Condell, Divyanshu Gandhi, Sebastian Montes, Dylan Vanstaden

MODS Trek Final Report

By: Dylan Vanstaden, Divyanshu Gandhi, Sebastian Montes, Roshon Condell

Florida Atlantic University

Professor Dr. Ravi Shankar

Abstract

The Museum of Discovery and Science (MODS) has decided to use a mobile app for android in order to give visitors a better museum experience. One of the problems they hope to solve with the app is having extra information, which cannot be found in exhibits. The app must also be user-friendly. This paper gives a detailed report of the *MODS Trek* mobile application for Google Android, made for the MODS. Topics that will be covered include the development and final product of the app, and future plans.

Keywords: MODS, app(s), Google, Android, Florida Atlantic University (FAU)

Background

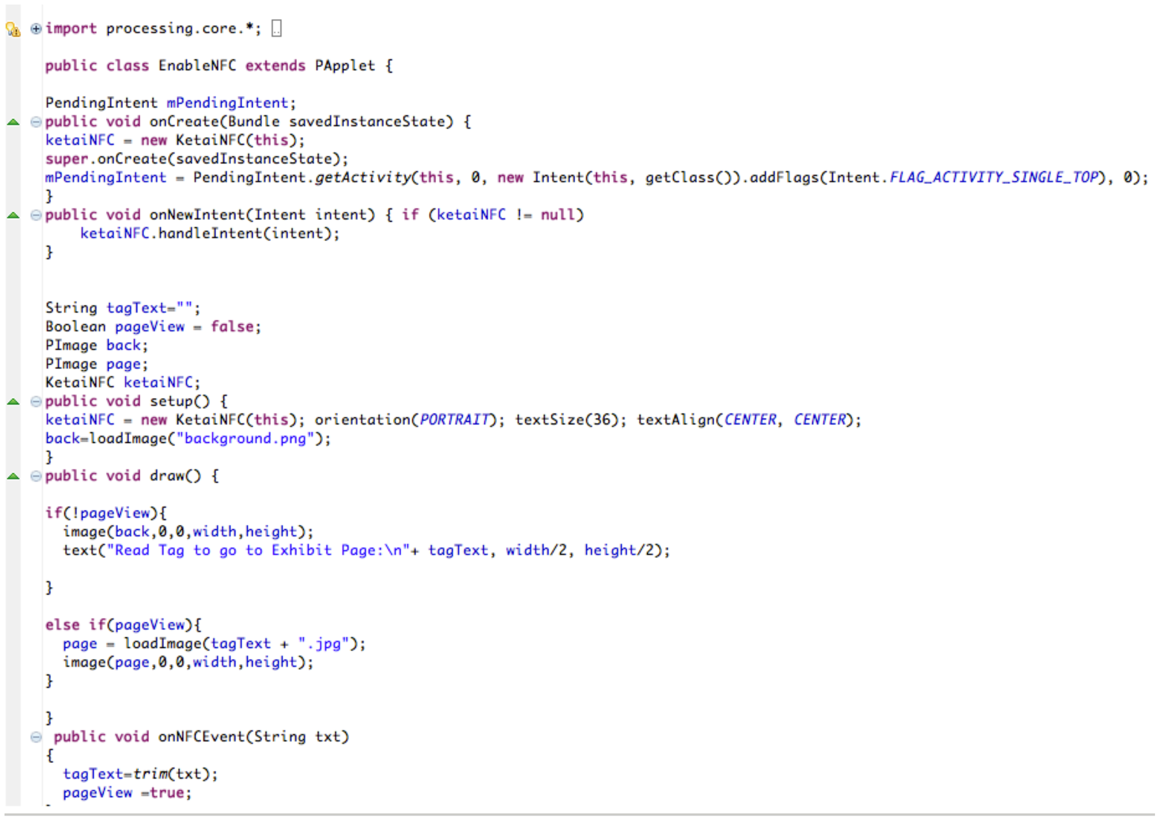
For years, the MODS has used paper maps, printed in black and white. Although it helps give visitors their bearings, it is outdated and unappealing to use. Not to mention all of the copies that litter the exhibits, or get thrown away. Finally, a solution; an app that provides users with a fully interactive color map, amongst other things, such as background information of exhibits, or a game for children. The MODS decided to have the students of FAU, in professor Dr. Ravi Shankar's *Mobile Apps for Google Android* class, create the app. How did a group of amateur programmers develop a fully functioning app in 3 weeks? Find out in this paper, written by Group 3, creators of the app MODS Trek.

Method

To develop the app, Group 3 used Processing and Eclipse, as well as Google's Android Development Tools (ADT) for coding, and used Adobe Photoshop to make the app logo/icon, backgrounds, and map icons. The app was tested on Asus Nexus 7 tablet (2013), with Android version 4.4.3.

Code

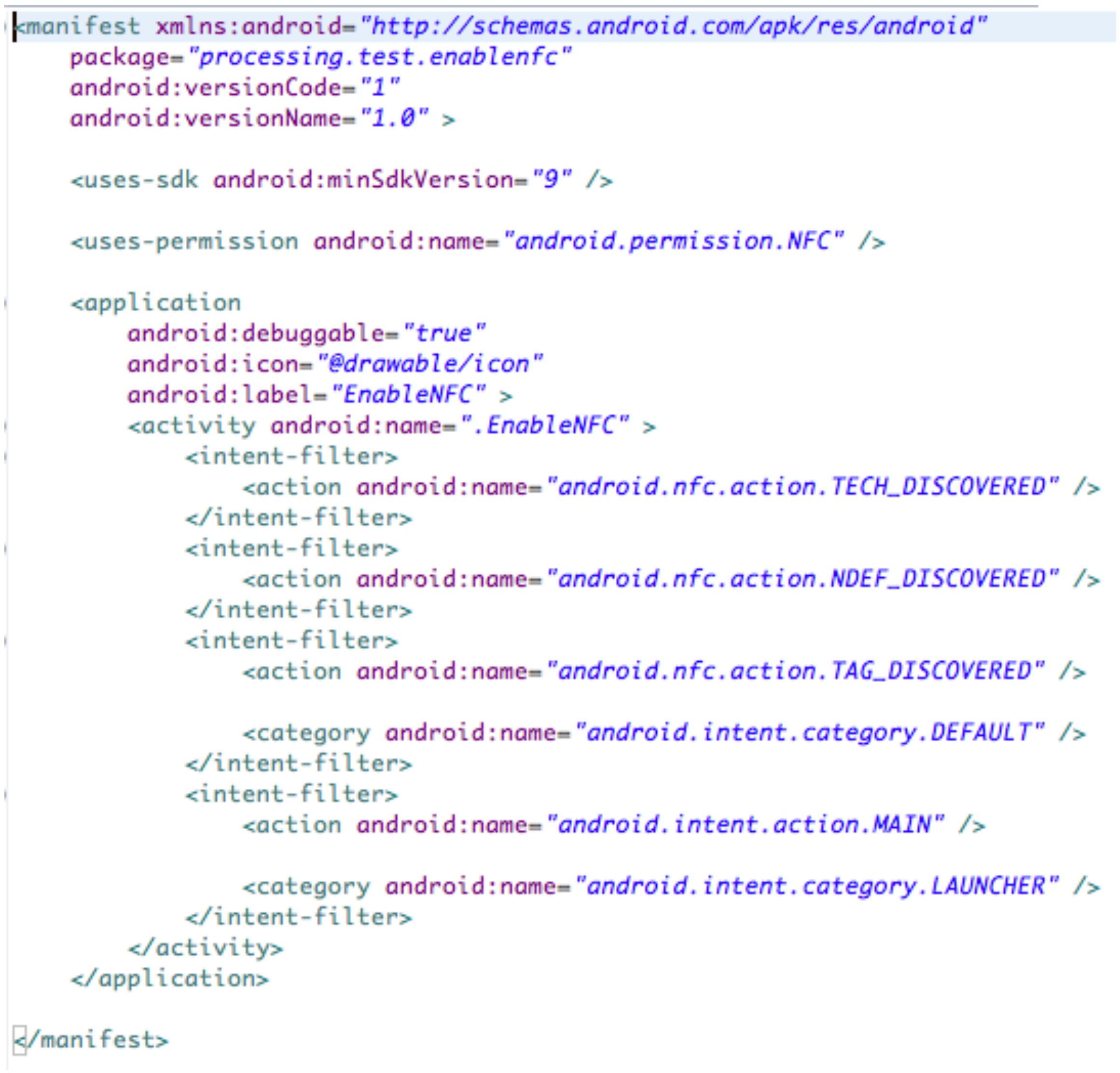
Near Field Communication (NFC)



```
import processing.core.*;

public class EnableNFC extends PApplet {

PendingIntent mPendingIntent;
public void onCreate(Bundle savedInstanceState) {
ketaiNFC = new KetaiNFC(this);
super.onCreate(savedInstanceState);
mPendingIntent = PendingIntent.getActivity(this, 0, new Intent(this, getClass()).addFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP), 0);
}
public void onNewIntent(Intent intent) { if (ketaiNFC != null)
    ketaiNFC.handleIntent(intent);
}


String tagText="";
Boolean pageView = false;
PImage back;
PImage page;
KetaiNFC ketaiNFC;
public void setup() {
ketaiNFC = new KetaiNFC(this); orientation(PORTRAIT); textSize(36); textAlign(CENTER, CENTER);
back=loadImage("background.png");
}
public void draw() {

if(!pageView){
  image(back,0,0,width,height);
  text("Read Tag to go to Exhibit Page:\n"+ tagText, width/2, height/2);

}

else if(pageView){
  page = loadImage(tagText + ".jpg");
  image(page,0,0,width,height);
}

}
 public void onNFCEvent(String txt)
{
  tagText=trim(txt);
  pageView =true;
```

Figure 1

Enable NFC file is written in processing and extends the PApplet class, which allows eclipse to run the file as a processing code. The program uses the Ketai library to access the NFC sensors. Once the NFC sensor is sensed, it will read the file and look for a string, which will then be the path to the picture of the exhibit page. As of right now, we haven’t programmed to check for errors, and this is a simple prototype program.

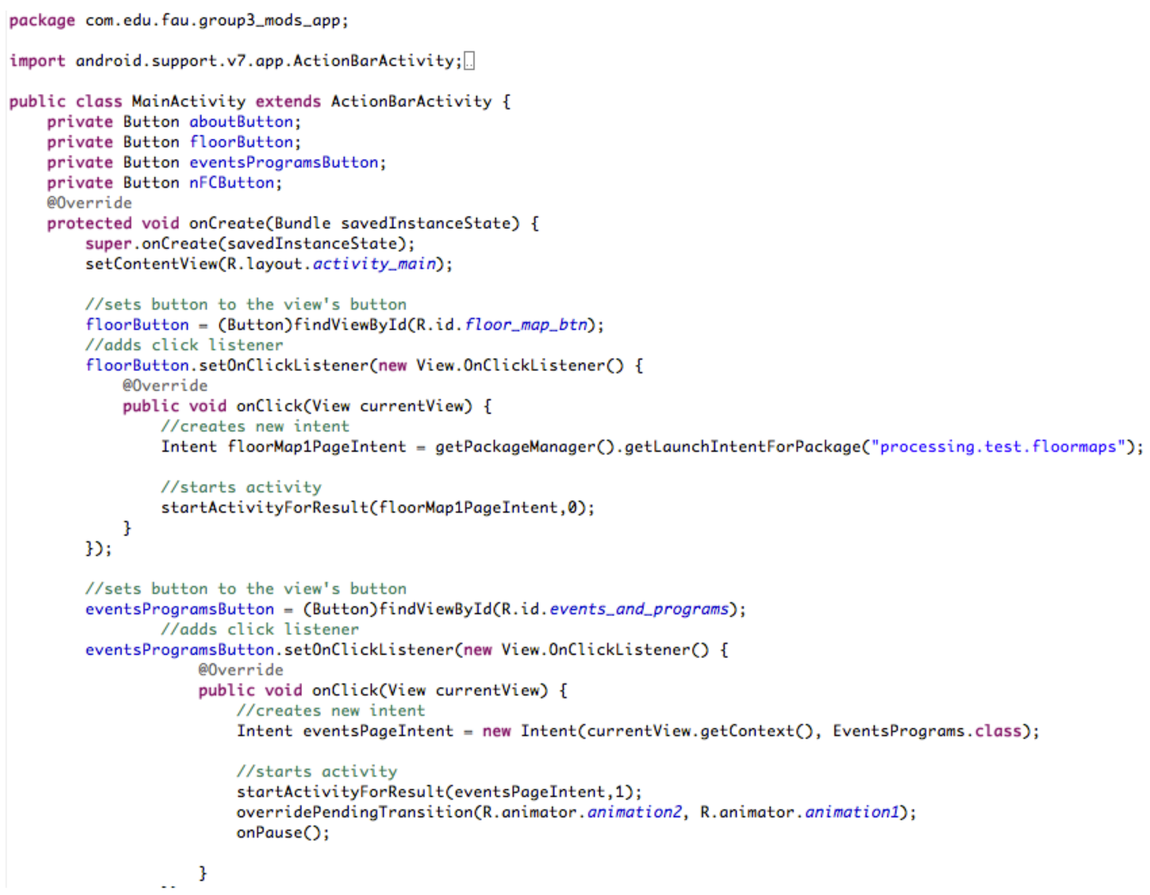


```xml
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="processing.test.enablenfc"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="9" />

    <uses-permission android:name="android.permission.NFC" />

    <application
        android:debuggable="true"
        android:icon="@drawable/icon"
        android:label="EnableNFC" >
        <activity android:name=".EnableNFC" >
            <intent-filter>
                <action android:name="android.nfc.action.TECH_DISCOVERED" />
            </intent-filter>
            <intent-filter>
                <action android:name="android.nfc.action.NDEF_DISCOVERED" />
            </intent-filter>
            <intent-filter>
                <action android:name="android.nfc.action.TAG_DISCOVERED" />

                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Figure 2

This is the manifest file of the enable NFC program. The first thing it does is give permission for the table to read the NFC tag then it declares actions for when the tag is discovered and it gives a launcher activity to the class.

Main Activity

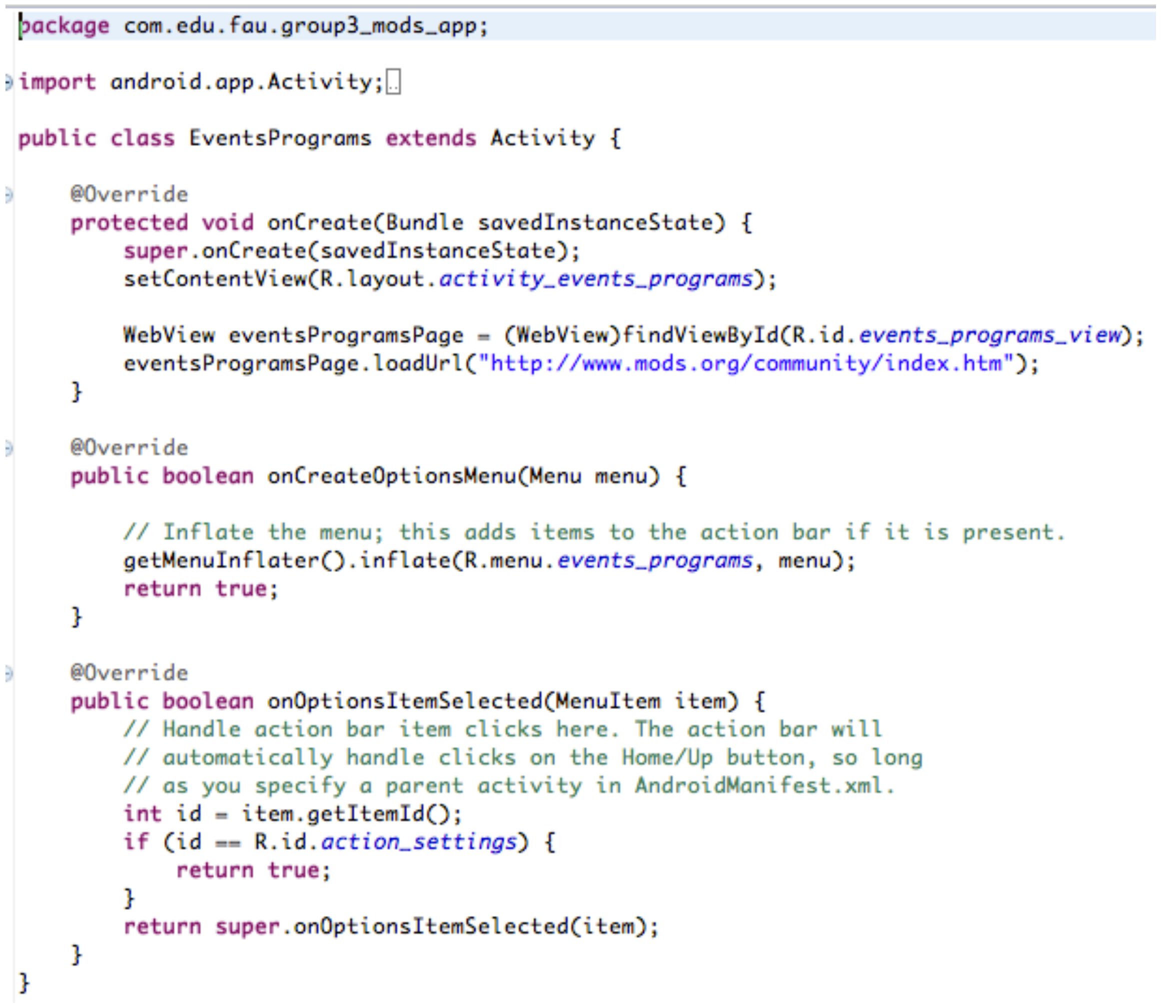


```
package com.edu.fau.group3_mods_app;

import android.support.v7.app.ActionBarActivity;

public class MainActivity extends ActionBarActivity {
    private Button aboutButton;
    private Button floorButton;
    private Button eventsProgramsButton;
    private Button nFCButton;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        //sets button to the view's button
        floorButton = (Button)findViewById(R.id.floor_map_btn);
        //adds click listener
        floorButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View currentView) {
                //creates new intent
                Intent floorMap1PageIntent = getPackageManager().getLaunchIntentForPackage("processing.test.floormaps");

                //starts activity
                startActivityForResult(floorMap1PageIntent,0);
            }
        });

        //sets button to the view's button
        eventsProgramsButton = (Button)findViewById(R.id.events_and_programs);
                //adds click listener
        eventsProgramsButton.setOnClickListener(new View.OnClickListener() {
                    @Override
                    public void onClick(View currentView) {
                        //creates new intent
                        Intent eventsPageIntent = new Intent(currentView.getContext(), EventsPrograms.class);

                        //starts activity
                        startActivityForResult(eventsPageIntent,1);
                        overridePendingTransition(R.animator.animation2, R.animator.animation1);
                        onPause();

                    }
```

Figure 3

The main activity class holds the code for the main menu, where the 4 buttons can direct you to different pages. As see in the code, there are 4 buttons and we declare them to be the corresponding button in the graphical layout. We then add onClickListeners, which tell whether the item has been clicked or not, and once clicked, the corresponding intent, or page, will be called and opened.

Events and Programs

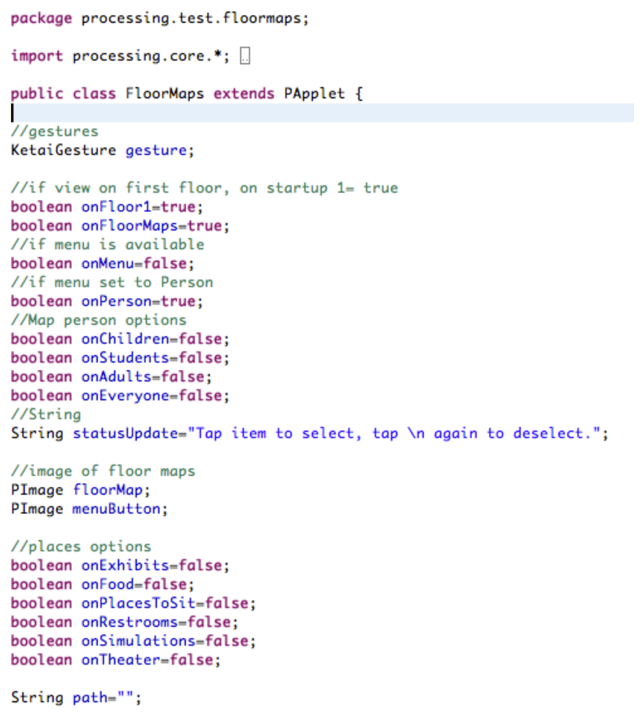


```java
package com.edu.fau.group3_mods_app;

import android.app.Activity;

public class EventsPrograms extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_events_programs);

        WebView eventsProgramsPage = (WebView)findViewById(R.id.events_programs_view);
        eventsProgramsPage.loadUrl("http://www.mods.org/community/index.htm");
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {

        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.events_programs, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();
        if (id == R.id.action_settings) {
            return true;
        }
        return super.onOptionsItemSelected(item);
    }
}
```

Figure 4

Events program page is called from the main menu page and then it uses a web view to call the page on the website. Since hardcoding the events and programs would become outdated, it is best to just use the information on the website. Once loaded, the user can view the website page where all the events are shown.

Floor Map(s)

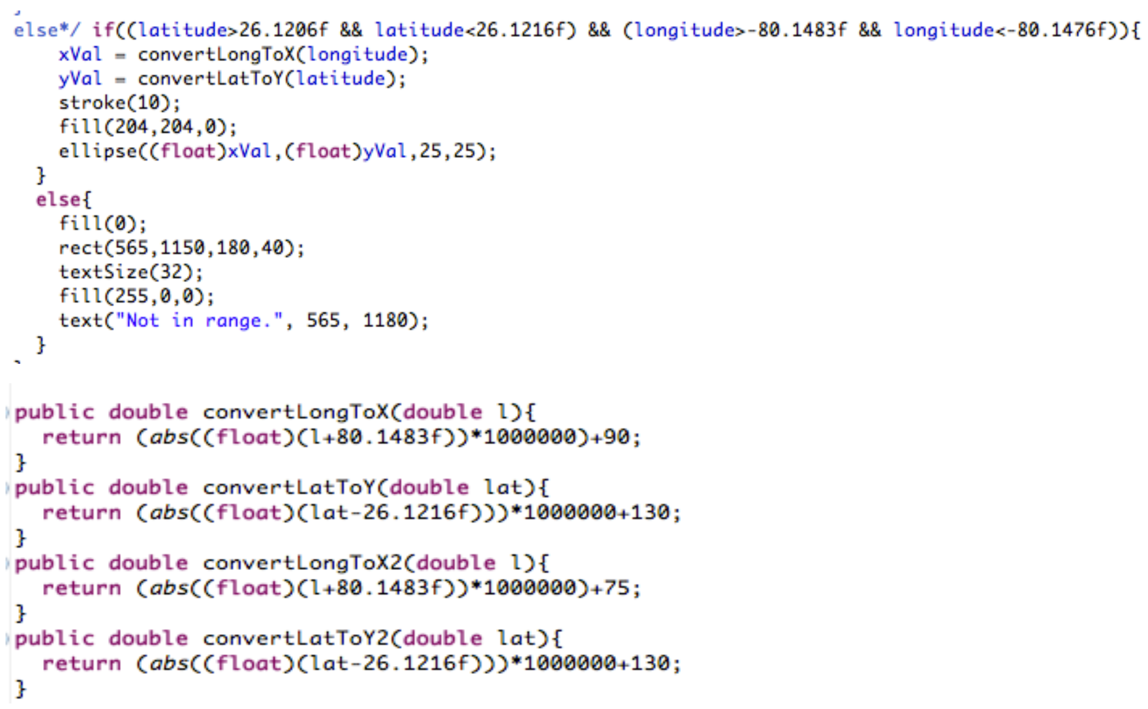


```
package processing.test.floormaps;

import processing.core.*; 

public class FloorMaps extends PApplet {

//gestures
KetaiGesture gesture;

//if view on first floor, on startup 1= true
boolean onFloor1=true;
boolean onFloorMaps=true;
//if menu is available
boolean onMenu=false;
//if menu set to Person
boolean onPerson=true;
//Map person options
boolean onChildren=false;
boolean onStudents=false;
boolean onAdults=false;
boolean onEveryone=false;
//String
String statusUpdate="Tap item to select, tap \n again to deselect.";

//image of floor maps
PImage floorMap;
PImage menuButton;

//places options
boolean onExhibits=false;
boolean onFood=false;
boolean onPlacesToSit=false;
boolean onRestrooms=false;
boolean onSimulations=false;
boolean onTheater=false;

String path="";
```

Figure 5

The code for the floor maps uses many Boolean checks in its draw method to be able to function as it is shown. The code is written in Processing. All these different Booleans are checked under the draw method and the interface is created based on what Booleans are true and what aren't. The Booleans check if the interface is currently under the map view, the floor 1 view, the menu overlay view and which of the current button selections are turned on. When a certain button selection is on, the button color changes and the icon is set on the screen.

GPS



```
else*/ if((latitude>26.1206f && latitude<26.1216f) && (longitude>-80.1483f && longitude<-80.1476f)){
    xVal = convertLongToX(longitude);
    yVal = convertLatToY(latitude);
    stroke(10);
    fill(204,204,0);
    ellipse((float)xVal,(float)yVal,25,25);
  }
  else{
    fill(0);
    rect(565,1150,180,40);
    textSize(32);
    fill(255,0,0);
    text("Not in range.", 565, 1180);
  }

public double convertLongToX(double l){
  return (abs((float)(l+80.1483f))*1000000)+90;
}
public double convertLatToY(double lat){
  return (abs((float)(lat-26.1216f)))*1000000+130;
}
public double convertLongToX2(double l){
  return (abs((float)(l+80.1483f))*1000000)+75;
}
public double convertLatToY2(double lat){
  return (abs((float)(lat-26.1216f)))*1000000+130;
}
```

Figure 6

The code above shows the code for the GPS feature for the first floor. This code is written under Booleans, which check if the interface is currently under the first floor map. It then checks to see if the users longitude and latitude is inside the museum, if it isn't, a "Not in range." message will show. If it is in range, then the longitudes and latitudes are converted to pixel values and an ellipse representing your relative location in the musem is shown. Although this is not as accurate as it can be, the code was tested and it has an extent of accuracy. The best way to improve this code would definitely be making a more accurate floor map for the museum. The GPS feature works the same way for the second floor but a little differently when converted to pixels as shown by the convertLongToX() and convertLatToY() methods.

Results

Interactive Map

An interactive map, that allows users to choose different audiences, such as children, students, and adults. Tabs for each floor were also utilized, making the app user friendly and easier to navigate. See Appendix A.

GPS

GPS was implemented, giving users the ability to view where they are in the museum. See Appendix B.

About

The about page gives users museum information, such as operating hours, phone number, and address. The about page also displays the names' developers of the application. See Appendix C.

NFC

Android devices have the ability to read NFC tags. By going to the NFC page, a user is able to scan an NFC tag in an exhibit, which would pull up relative information. See Appendix D.

Discussion

Future Plans

In the beginning, Group 3 had many ideas for the app, which would take advantage of android devices' hardware capabilities, as well as a user-friendly design. Although Group 3 presented the "final product", they hope to implement the following in later versions of the application:

- Voiceover
- Mini-game

Conclusion

The purpose of creating MODS Trek was to give visitors a helpful guide that they can carry in their pocket. Ultimately, Group 3 managed to do this, as well as create some other functions within the app. Users will be able to show/hide icons on the map, read NFC tags, and see social events occurring and coming up at the museum. Not to mention, users will also be able to find museum contact information, its address, and operating hours, all in one place. We had a great time making this app, and plan to add other functions, to make it the best MODS Trek it can be.

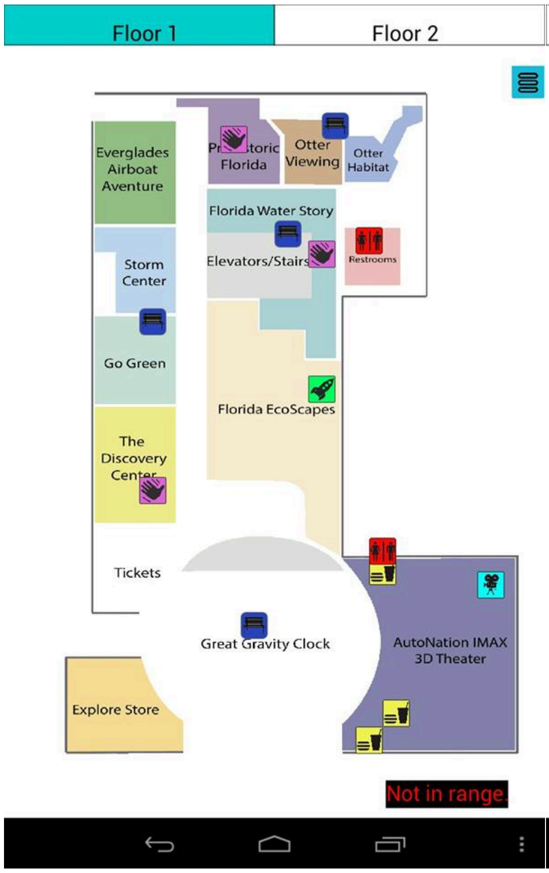
References

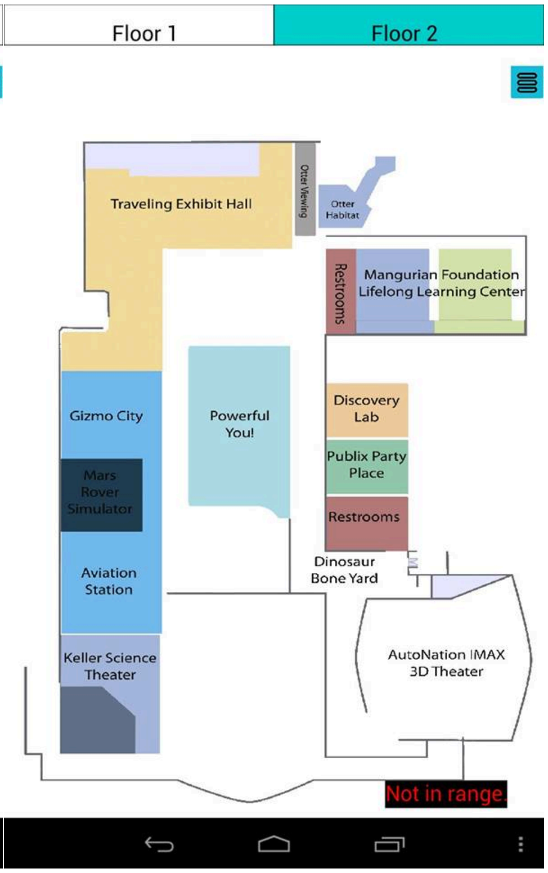
Google. (n.d.). *Near Field Communication*. Retrieved 6 24, 2014, from Android Developers: http://developer.android.com/guide/topics/connectivity/nfc/index.html

Sauter, D. (2012). *Rapid Android Development.* (J. Osborn, Ed.) Dallas, Texas, USA: The Pragmatic Bookshelf.

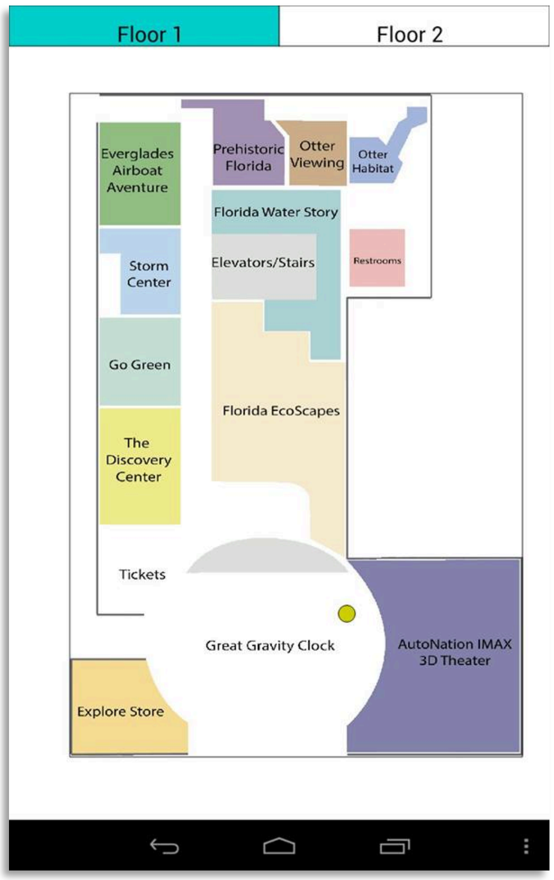
Appendices

Appendix A

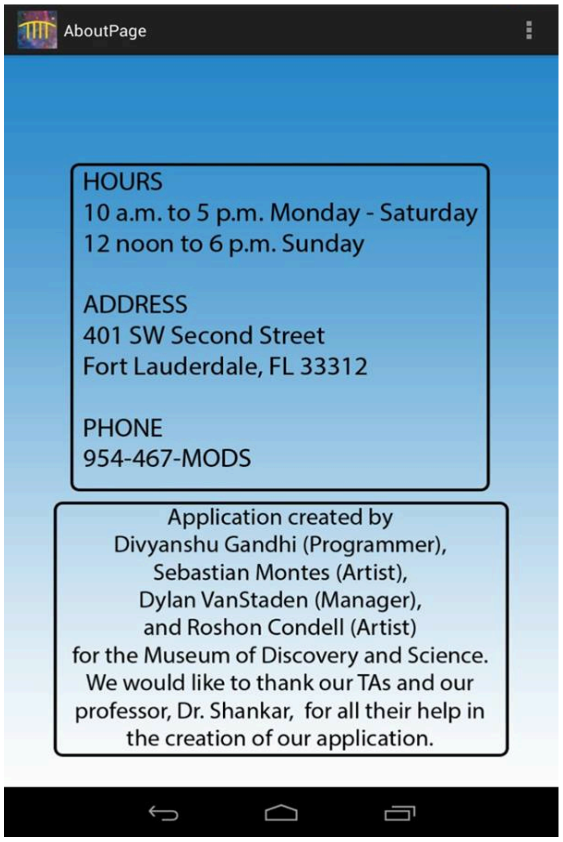




Appendix C



Appendix B



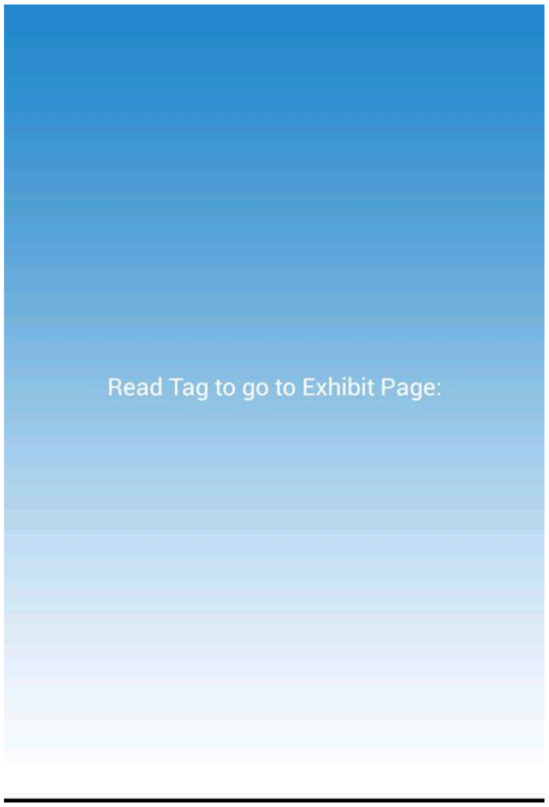
HOURS
10 a.m. to 5 p.m. Monday - Saturday
12 noon to 6 p.m. Sunday

ADDRESS
401 SW Second Street
Fort Lauderdale, FL 33312

PHONE
954-467-MODS

Application created by
Divyanshu Gandhi (Programmer),
Sebastian Montes (Artist),
Dylan VanStaden (Manager),
and Roshon Condell (Artist)
for the Museum of Discovery and Science.
We would like to thank our TAs and our
professor, Dr. Shankar, for all their help in
the creation of our application.

Appendix D



Read Tag to go to Exhibit Page: