UNIVERSITY OF ALASKA ANCHORAGE

CSCE A470

CAPSTONE PROJECT

Interface Design for Carpal Tunnel Exercise Program

Author:

Trent Matthias

Supervisor:

Prof. Diana Steer, O.T./L

Anchorage AK, April 2016

tematthias@alaska.edu

Abstract

The goal of this capstone is to explore Android development and make a health application that can be useful to people afflicted by carpal tunnel syndrome. This document is structured to detail every stage of development, from concept to finished product. Starting with the idea behind the capstone, the Patient Exercise Guide, or PEG, will help carpal tunnel syndrome patients recover from their surgery in a guided manner. The goal of the PEG is to help people with carpal tunnel with professional level advice and care that a therapist would bring.

Chapter two of this capstone is about system integration and technologies being used. The PEG is being developed for Android which is very similar to Java. Android allows the PEG to reach a broad audience with minimal cost to the developers. The PEG is being designed with Android principles in mind, this includes the idea of everything is clickable and to use the action bar. The PEG is being developed with an Agile like methodology, this allows for rapid iterations and the ability to adapt to design changes later in the development cycle.

Chapter three is about the user interface and testing methodology being used. The PEG is targeting an older audience; the user interface is important, it needs to be clean and professional to hold interest. If the traversing the app is a challenge then people will simply not use it, the PEG needs to have a smartly designed user interface. To do this the PEG will use the action bar as a globally available navigation tool within the PEG. The PEG has a limited amount of screens so can afford to use the action bar as a static navigation tool.

Chapter four is simply the user manual and explains how to use the PEG. The PEG is small and easy to use as such the manual explains what the purpose of each screen is and the different ways to reach them.

Chapter five is the conclusion to this capstone. The conclusion goes over what was learned and how it applies to Computer Science and how those skills will be useful. Future features and regrets for the PEG are discussed and what went right and wrong during the development.

Acknowledgements

I would like to thank my Mom for supporting me throughout my college career. Without her I would not be where I am today.

Contents

List of Figures

Chapter 1

Introduction

1.1 Introduction

More than 500,000 people have carpel tunnel surgery every year [2]. Carpel tunnel is the most common nerve issue in adults but is quite rare in children [3]. Carpel tunnel syndrome is caused by repetitive stress to the wrist; this stress causes inflammation that pinches the nerve leading to the hand [4]. Carpel tunnel syndrome causes extreme pain and discomfort in the hands and can vary in intensity. Sometimes surgery is the only option to treat carpel tunnel syndrome, this leads to a lengthy recovery time.

Occupational therapy is usually required post-surgery of carpel tunnel syndrome. Occupational therapy helps people recover or maintain their work or living skills impeded by something like carpel tunnel syndrome or a disability. Occupational therapy works by removing environmental barriers and changing how the task being affected is performed. Occupational therapy is important, by reducing the barrier of entry hopefully more people with less severe issues can also reap the benefits.

This capstone is about making an Android app for people going through occupational therapy for carpel tunnel syndrome. Developing for Android is cheap and relatively easy to do, meaning one or two people can make a feature complete app, Android is ideal for small teams. Android is widespread, many people own an Android device and can bring it with them to their therapy sessions. By having an Android app designed to help treat carpel tunnel syndrome, people will be able to do easy exercises with their device in a guided manner. Patients do not have to worry about doing the exercises wrong and will have access to professional information right from their device.

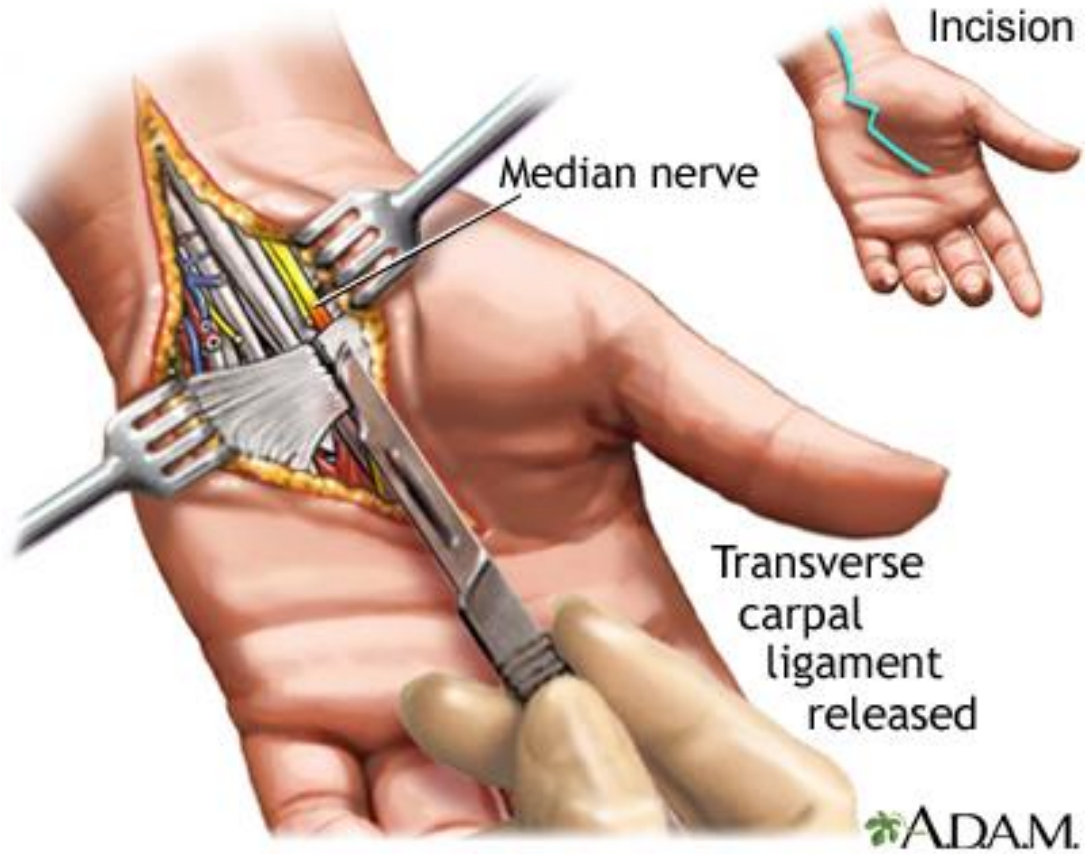


Figure 1: How carpal tunnel is treated

Source: Carpal tunnel syndrome. (n.d.). Retrieved February 01, 2016, from https://umm.edu/health/medical/reports/articles/carpal-tunnel-syndrome

1.2 Application

The application being developed is called the Patient Exercise Guide, or PEG, for short. When users first open the PEG they can either start the recovery process or resume their current treatment. The PEG is all about health care at the fingertips in a light easy to understand package. Once a user starts the process of recovering they will receive tips and basic exercises to help strengthen the wrist and fingertips. As the treatment continues, patients will have more complex exercises to perform such as rotating the phone in specific directions. By utilizing the gyroscope of the Android device the PEG will be able to accurately tell how the patient's range of motion compares to previous days. This data can be stored locally in the device's file system for easy access on the go. As treatment continues the gyroscope exercises will gradually become more complex, slowly growing the range of motion of the patient. Other potential features

include the ability to schedule when to take medicine or when the next therapy session is to take place. Appointments can be scheduled directly to the user's calendar on their device.

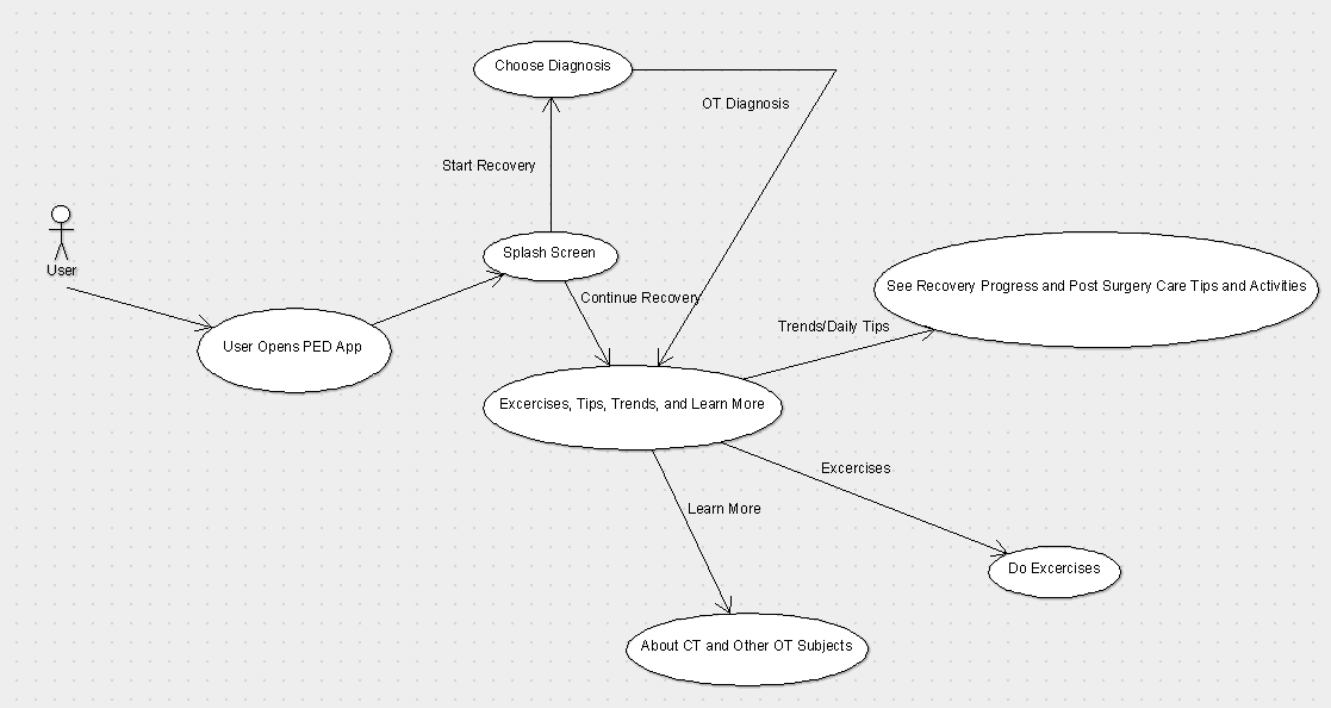


Figure 2: Use case diagram for the Patient Exercise Guide

A rough idea of how users will use the app can be seen in Figure 2, each bubble represents a separate screen. The first screen of the app will be a basic splash screen that will act as the hub of the app. From the splash screen users can start a new program or continue where they left off. As the app develops there may be time to implement more occupational therapy guides other than carpel tunnel syndrome. If there is enough time, then the choose diagnosis bubble is where a user would select from different guides and exercises. From either the splash screen or choose diagnosis screen the user can access the hub of the exercise portion of the app. Like the splash screen, the exercises screen acts like a menu for the other screens related to the topic selected in choose diagnosis. Users can see tips or recovery progress, start the actual exercises or learn more about carpel tunnel or other occupational therapy subjects from the respective screens.

Version	Codename	API	Distribution
2.2	Froyo	8	0.2%
2.3.3 - 2.3.7	Gingerbread	10	3.0%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	2.7%
4.1.x	Jelly Bean	16	9.0%
4.2.x		17	12.2%
4.3		18	3.5%
4.4	KitKat	19	36.1%
5.0	Lollipop	21	16.9%
5.1		22	15.7%
6.0	Marshmallow	23	0.7%

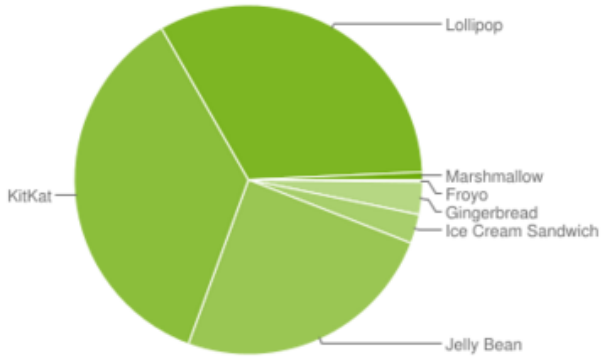


Figure 3: Android API distribution stats

Source: Dashboards. (n.d.). Retrieved February 01, 2016, from http://developer.android.com/about/dashboards/index.html

The most challenging part about developing an Android app is compatibility. Choosing the API is an important decision, the lower the API the more devices that are supported but the app has access to less features. This tradeoff is important and it must be decided which is more cruical, extra functionality or less compatibility. Another big issue with Android is screen size; with so many Android devices on the market it becomes difficult to support the various screens. By coding with relative layouts the PEG should be compatible on many different devices regardless of screen size.

1.3 Motivation

Android is the platform of choice for a few reasons, the main one being it is easy to develop for. As a student there is a limited amount of material covered for mobile platforms in classes. Android is covered the most, also Android development is essentially Java, the main language covered at UAA. As such it is easy to learn and develop for the Android API whereas IOS requires knowledge of the Swift programming language and requires specific hardware to develop on. There are over a billion Android devices on the market [7] the PEG could be beneficial to potentially millions of people suffering from varying degrees of carpel tunnel syndrome.

Figure 4: The Android robot logo

Source: Brand Guidelines. (n.d.). Retrieved February 01, 2016, from http://developer.android.com/distribute/tools/promote/brand.html

1.4 Recent Developments

According to HIMSS, 90 percent of adults own a smart phone [6]. Doctors are quickly adopting mobile technologies to enhance the patient experience through additional data and diagnostics [6]. The PEG could potentially be another tool in the doctor's tool belt to help treat patients in need of occupational therapy. Apps for occupational therapy already exist for kids, these apps manifest themselves as games to keep the kid engaged [8]. An app made specifically for occupational therapy has far reaching appeal for both kids and adults. The PEG will utilize professional language and exercises to simulate working with an actual therapist. Hopefully these features will distinguish the PEG from other health care apps, the goal of the PEG is to bring an occupational therapist to as many users as possible.

Chapter 2
System Integration and Modeling / Methodology

2.1 Programming Technology

Android apps use a version of Java, as such the technology is very similar. What makes Android different from Java is how the code structure works. Android devices have small screens and limited ways to interact with UI elements. As such Android needs a way to change screens while passing information along from the previous screen. Android does this by using activities and intents, activities are the screens and intents are the information and act of changing the screen. The other thing that differentiates Android from Java is the reliance on XML. Everything in the Android app is stored in XML files, such as UI elements. The Android manifest file, which stores all the information pertaining to the app, is an XML file.

Regular Java has different design patterns compared to Android even though both are essentially Java. Android can do the same thing as traditional Java but it is structured differently to make use of mobile devices. Fragments allow the developer to break the UI into small packages that can display differently based on the size of the screen. Android needs to be more flexible to fit all the screen sizes, some screens are tiny while others are quite big. Traditional java simply lacks the ability to have a dynamic UI and the ability to pass information between screens. Android also needs to be able to handle touch events, like swiping or pinching. Android has all the features of traditional Java programming while adding functionality that is crucial to phones and tablets.

Figure 5: A good representaion of Android programming

Source: Google v/s Oracle. (n.d.). Retrieved March 04, 2016, from http://thetechweekly.com/article/google-vs-oracle-gods-in-the-battlefield/

2.2 Application Design

The Patient Exercise Guide is being designed with a broad audience in mind. The UI needs to be intuitive and easy to use. Figure 6 shows that carpal tunnel syndrome is more common the older a person becomes. The PEG needs to be accommodating to an older audience, as such some design decisions need to be considered carefully. The most important is UI readability and navigation, older people may not be as familiar with Android principles, mainly that everything is clickable, and may need an extensive tutorial. On the other hand, if they are intimately familiar with Android app design quirks the PEG should allow the user to skip any tutorial and still be easy to understand. The PEG needs to avoid obfuscation because features are hard to understand or find. The app needs to make everything it can do clear without hurting the flow of the app. The user should be able to efficiently traverse to the screen they need to be on without getting

lost in the process. The goal of the PEG is to help carpal tunnel syndrome patients recover, not to frustrate them with poor app design and a cheap looking UI. The PEG must look and feel professional to appeal to an older audience.

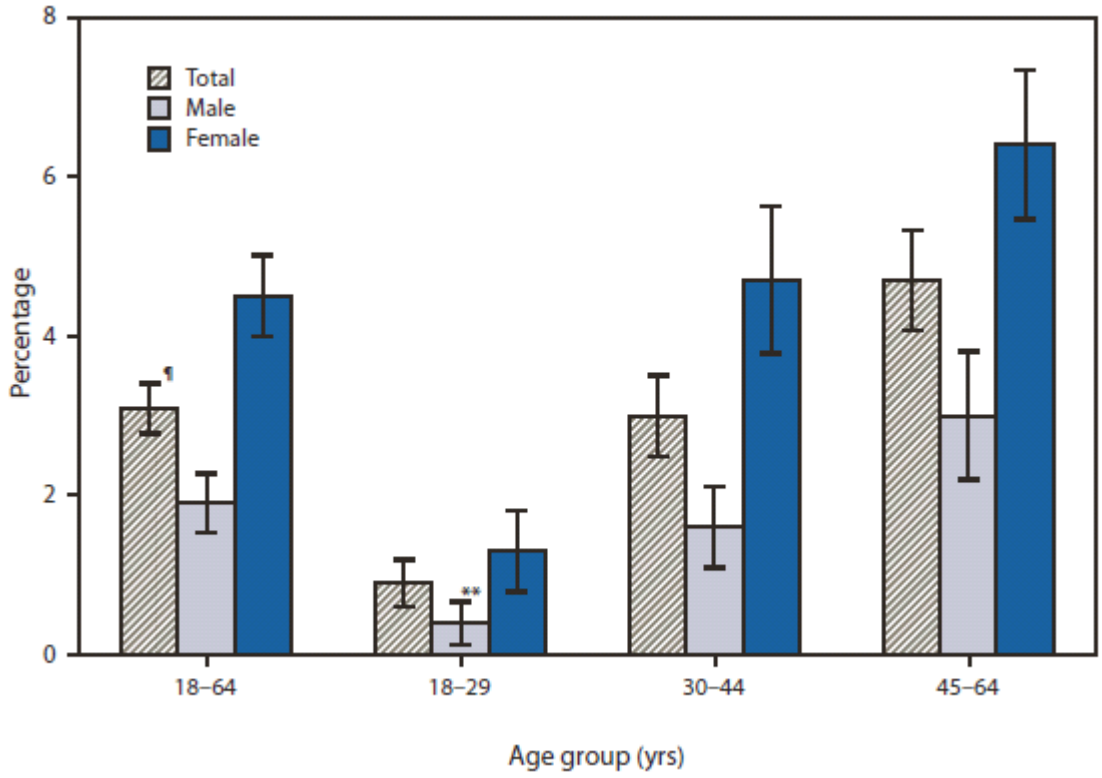


Figure 6: Statistics of people afflicted with carpal tunnel syndrome

Source: Percentage of Employed Adults Aged 18–64 Years Who Had Carpal Tunnel Syndrome in the Past 12 Months, by Sex and Age Group — National Health Interview Survey, 2010. (2011.). Retrieved March 04, 2016, from http://www.cdc.gov/mmwr/preview/mmwrhtml/mm6049a4.htm

The PEG can avoid most of the confusions by using clear buttons. This means avoiding vague symbols for button icons that do not represent what the button does. At most the PEG will only use Android specific symbols such as the three dots for overflow options. Extensive use of the action bar will make the UI have less clutter and give relevant options to the user. These options may be app universal or screen dependent. Any other option buttons can be deployed in a drawer on either the left or right side of the screen. By separating the buttons out and giving them enough space the PEG will have enough room to clearly label each button.

The last important design decision is how the UI will look on different screen sizes. The PEG will not support tablets due to how the hand exercises work. This gives the PEG the ability to not use fragments and instead focus on layout design. Relative layouts allow UI components to dynamically fit different screens sizes by specifying where an object is relative to other objects

or the screen. This will allow the UI to fit on most Android screens and avoid the complexity of fragments.

2.3 Hardware

Figure 7: The Samsung Galexy Tab 4

Source: Samsung Galaxy Tab 4 8.0 (2015) Coming with Snapdragon 410 Chipset. (2015.). Retrieved March 06, 2016, from http://www.wlivenews.com/samsung-galaxy-tab-4-8-0-2015-coming-with-snapdragon-410-chipset.html

The PEG is being tested on local hardware, in this case a Samsung Galaxy Tab 4 seen in Figure 7. An advantage to using a tablet is knowing how the UI will look on a bigger screen. The PEG is also being tested on smaller Android phones which is the optimal and supported way of using the app.

2.4 Agile Coding

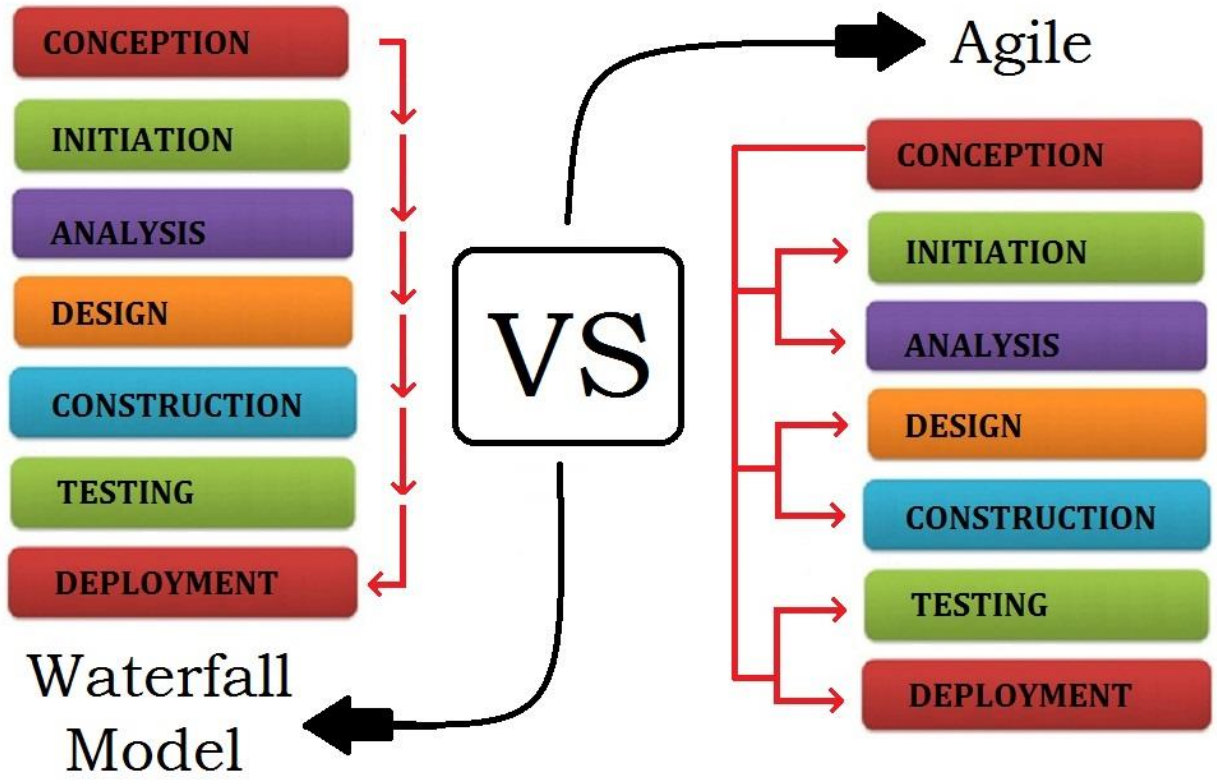


Figure 8: Agile vs waterfall

Source: Agile Vs Waterfall. (2012.). Retrieved March 06, 2016, from https://www.sdlc.ws/agile-vs-waterfall/

The PEG is being developed with an Agile approach due to it being a relatively small project and small team. Agile gives the team a few advantages, mainly the ability to adapt to changes. Agile is all about being able to adapt to changes in design later in development [9]. Waterfall is planned ahead, if change occurs it has a big time cost associated with it [9]. Waterfall also has much more documentation where Agile uses test cases to document the code. Agile is better for small teams and projects and is why we chose to use it for the PEG.

Another aspect to Agile is the coding methodology or clean code. Clean code is code that is easy to understand and change [13]. The PEG is being developed with clean code in mind, this means clear variable and method names and easy to understand code flow and formatting. This is a challenging in Android because of the XML and how separated the logic and UI portion of the code is.

		Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11	Week 12
2	Project Outline												
3	Requirements Specification												
4	Research												
5	Interface Design												
6	Exercise Design												
7	Writing the Care Manual												
8	Program Implementation												
9	Testing												
10	Maintenance												

Figure 9: Gantt chart for the PEG application

Figure 9 shows a rough schedule of development for the PEG application. A good chunk of the project will be writing the care manual for carpal tunnel syndrome and testing the application. Since the PEG is being developed with Agile, testing occurs throughout the whole process.

Chapter 3

Design and Testing / User Interface

3.1 Application User Interface

The PEG will have a simple user interface to let the user easily access information related to carpal tunnel syndrome. The app is focused towards an older audience, as such the user interface needs to look clean and professional. Another important aspect is screen traversal; the PEG has a limited amount of screens to avoid long traverse time. Long travel time can be devastating to apps; frustrated users are more likely to stop using the app all together. The PEG will use the action bar of Android to act as a global menu. From this menu, users can get to all the relevant screens they would want to access. This lets the user quickly jump around the app and avoid long traversal chains to get to their destination. This has the added benefit of having a relevant page only one hop away, Android's back button goes back one hop so the user can quickly go back from where they came.

The PEG will adhere to Android app standards, this includes the idea that everything is clickable and to use the action bar [15]. On IOS the design philosophy is different, not everything is clickable. As such app designers have to make clickable things look clickable. This has the benefit of having no ambiguity on what is and is not clickable. On Android because everything is clickable (or touchable) UI elements can blend together. Some people can view this as a negative because it makes Android more confusing. If an Android app has an element that is not clickable, then the inconsistency will confuse people. If the PEG were to be developed for IOS, then a different user interface would be necessary to adhere to IOS standards.

Figure 10: Android vs IOS

Source: iOS vs Android. Retrieved March 08, 2016, from http://thinkapps.com/blog/development/platform-build-first-ios-vs-android/

3.2 Testing Methodology

Testing is important for all computer applications, if the program is unstable due to bugs those bugs need to be dealt with before launch of the product. Testing allows application programmers to catch and rectify these bugs before production. The PEG will go through multiple stages of testing: design testing, code stability testing, and hardware testing.

Chapter 3 – Design and Testing / User Interface

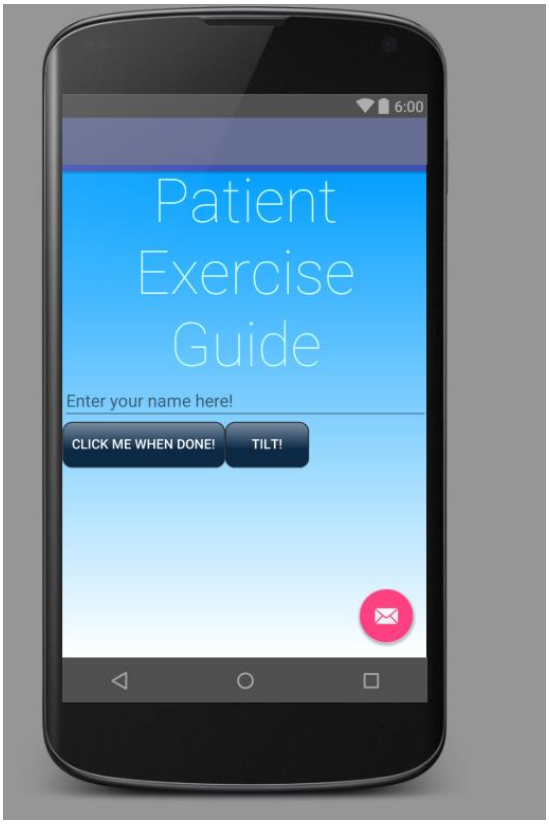


Figure 11: An early prototype for the PEG

Design testing consist on how the app is laid out, this type of testing will show design flaws that need to be fixed for the final version. These flaws could range from a traversal issue to fundamental design issues such as the exercises do not work as intended. The best way to do this type of testing is to get an early prototype of the application and show it to people. Gather feedback about the app and make meaningful changes that better the design. Much of design is subjective so the larger the sample pool the better the testing will be. If some aspect of the design is flawed, then a good chunk of the participants will note it to be an issue. Design testing also helps streamline the user interface, if a function of the app is being heavily used over some other function then make it easier to get to the heavily used part of the app. Rearranging how icons or how text is displayed can make the app easier to read. Design issues are hard to catch and may need an outside perspective to find.

Code stability testing consist of making sure the app does not crash because the user did something unexpected. This can range from a series of inputs that when made in a specific order crashes the app to simply clicking a button crashes the app. Basic user testing will catch most of the simple crashes, for more complicated crashes being able to let more people use the app will help bring those to light. The PEG is a small app so things like unit testing are less useful. The PEG is being built by a small team and has limited functionality so code stability testing should be relatively easy.

Hardware testing is important for the user interface; the PEG could potentially be used on multiple types of devices. These devices could all have different screen sizes, as such the PEG needs to have a user interface that can adapt to changes in screen size. This type of testing is limiting for the PEG because the team only has access to a limited amount of devices. In an ideal world the PEG would be tested at all screen sizes with lots of devices but unfortunately this is not the case. The PEG is being tested and optimized for phone sized screens as such large tablet screens may be unsupported.

Figure 12: Three stages of Android testing

Source: Best Automation Tools for Testing Android Applications. Retrieved March 09, 2016, from http://qabok.com/best-automation-tools-for-testing-android-applications/

3.3 Agile Methodology VS Agile Coding Methodology

The PEG is being developed with agile clean code in mind but this is different than agile as a project management tool. Agile clean coding is self-documenting code meaning that method and variable names describe what the function or variable is for. This means less comments in the code, the code should be easy to understand structurally and easy to follow. This can be a challenge for large projects, luckily the PEG is relatively small. Agile as a project management tool is a way to attack a problem and organize people. Agile is about rapid development that can change to new client request [14]. Agile is built around week long sprints and weekly meetings with the client to show progress [14]. Agile has less focus on code documentation than a waterfall method, the documentation comes from the test cases. The PEG is being developed with agile in mind but there is no client the PEG is being made for. There is not a large team working on the PEG that needs to be managed to meet deadlines. As such the PEG is being developed in an Agile like way but not a full agile method. However, the PEG is being developed with the idea of clean code. Clean code will result in less bugs and easier trouble

shooting for bugs found later in development. Agile is important for the development of the PEG and fits well with the team structure.

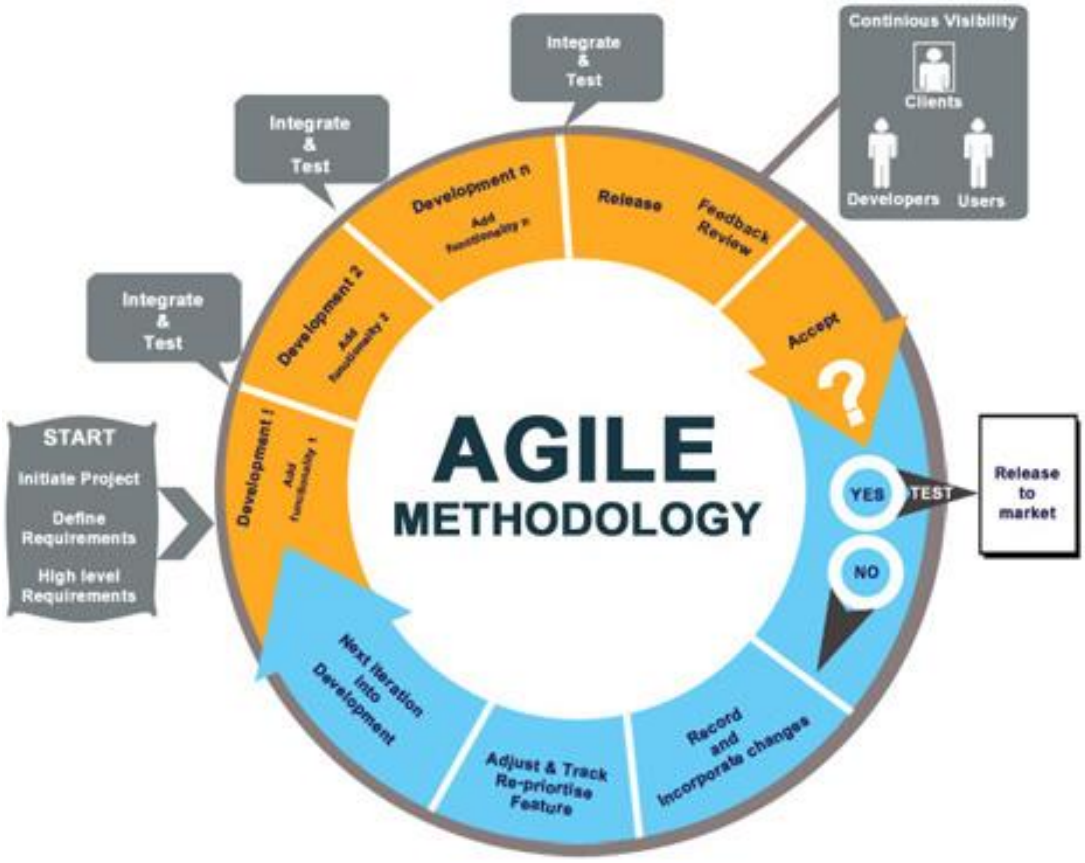


Figure 13: Agile for project management

Source: Best Automation Tools for Testing Android Applications. Retrieved March 09, 2016, from http://america.pink/software-development-process_4084983.html

Chapter 4

User Manual

4.1 Introduction

The Patient Exercise Guide is designed to help people afflicted with carpel tunnel syndrome. To help, the Patient Exercise Guide guides the user through hand exercises to help strengthen the wrist. The Patient Exercise Guide can also be used as a preventive measure for carpal tunnel syndrome due to the hand stretches.

Figure 14: The Patient Exercise Guide icon

The Patient Exercise Guide is mainly for people recovering from surgery for carpal tunnel syndrome. The Patient Exercise Guide is a personal occupational therapist that fits right in the user's pocket. The Patient Exercise Guide will help keep track of the user's progress and guide the user through the exercises that real occupational therapists recommend. The Patient Exercise

Guide also comes with a comprehensive care guide to help people learn more about carpal tunnel syndrome.

The Patient Exercise Guide is designed for smaller Android devices that can be held by one hand. Tablets are not supported and would not work for the exercises due to the larger size. Because the Patient Exercise Guide is under active development, this user guide may be out of date for later versions. This user manual is relative to the current prerelease version of the Patient Exercise Guide.

4.2 Starting the App

The Patient Exercise can be found on the Google Play Store for Android devices. The PEG is only available for Android at this time. Once the app is downloaded tapping the icon brings the user to the screen depicted in Figure 15. From the landing page the user can start a recovery program, go straight to the hand exercises or simply read about carpal tunnel syndrome. The PEG uses the action bar to access the care guide from all the screens except on the actual exercises.

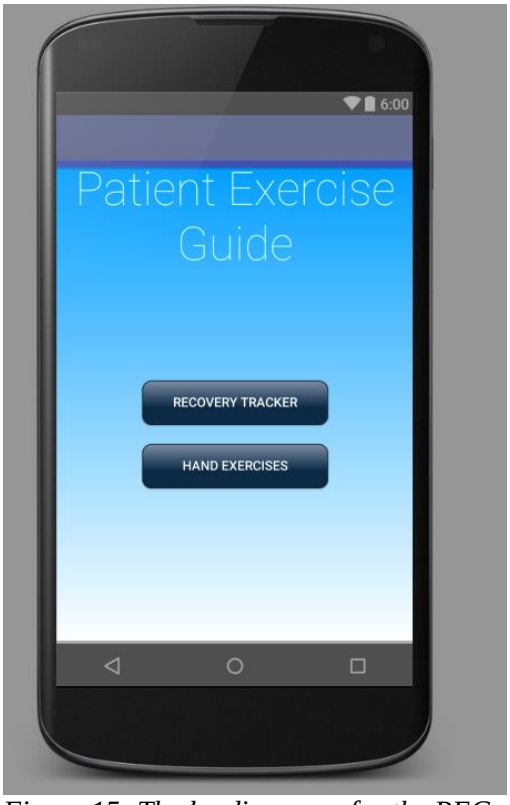


Figure 15: The landing page for the PEG

The recovery tracker is how the user keeps track of their program progress. When first entering the recovery tracker the user can start a new program or, if they already have started one, can see where they are in the program. The recovery tracker in conjunction with the care guide keeps the user on track with their recovery through the currently six-week program.

4.3 Choosing an Exercise

Tapping on hand exercises bring the user to the hand exercise selection screen depicted in Figure 2. The hand exercise screen consists of a spinner with all three exercises and pictures that describe how to perform the exercise. For more information on the three exercises consult the care guide for more in depth information. From the exercise selection screen the user can go back to the previous screen with the built in Android back button, can access the care guide, or reach the recovery page. Tapping the start exercise button brings the user to the meat of the app, the exercise screen.

Figure 16: The exercise selection screen

4.4 Performing the Exercise

The exercise screen is designed to use either the accelerometer or magnetometer depending on the exercise. To properly use the PEG, the user's Android device must have these two sensors or they will not be able to perform the exercises.

Once on the screen depicted in Figure 17 tapping the screen will reset and calibrate the sensor for the resting position of the hand. The user then performs the motions for the exercise selected and the green line will move up or down towards the red portion of the circle. The red portion shows where overextension occurs when performing the exercise. The Android device will vibrate to warn the user they are overextending their wrist and to back off. The max values show the user their range of motion and is stored. These values can be viewed from the recovery page.

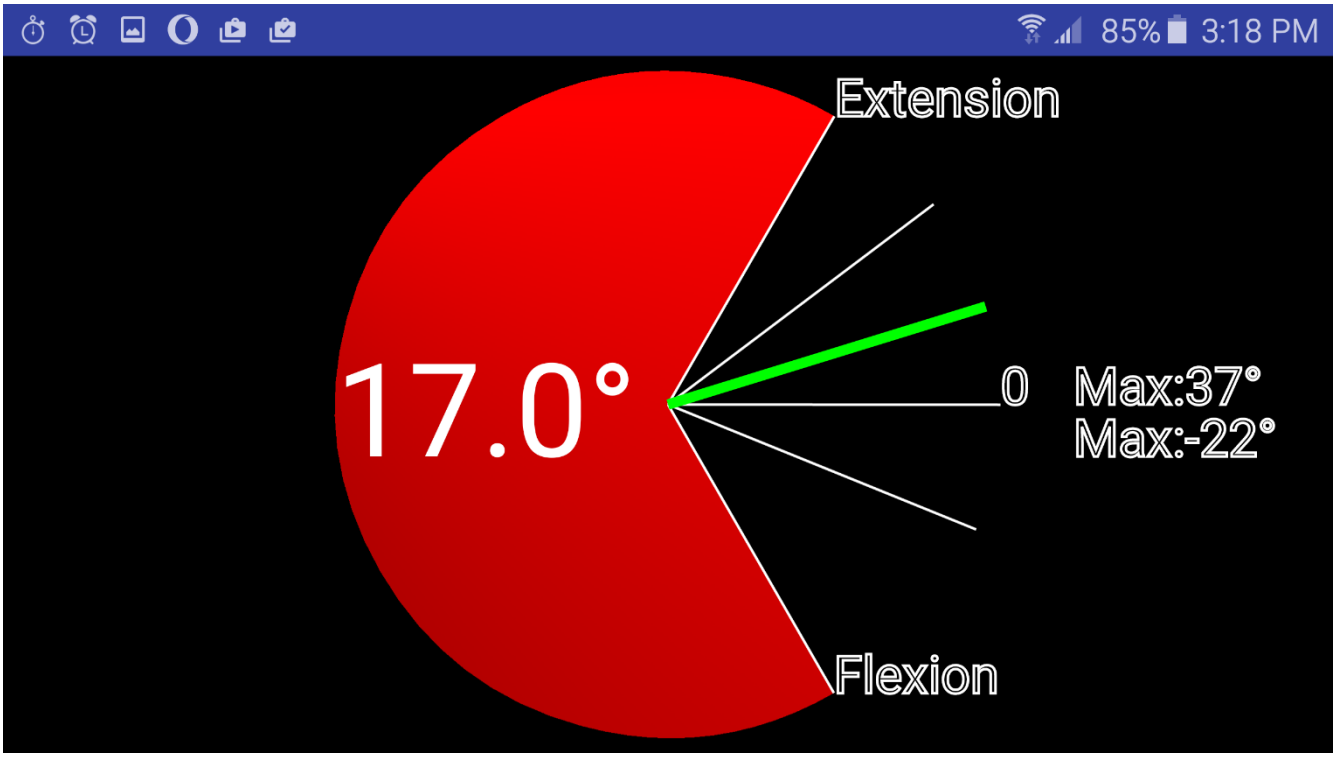


Figure 17: The UI for performing the exercise

The optimal way to use the PEG is with the external phone strap that holds the device to the user's wrist. This allows the user to keep their hands free while still getting feedback from their device.

Chapter 5
Summary and Conclusion

5.1 What was Learned

Throughout this capstone there were many challenges that had to be overcome. Version control and the use of GitHub was another layer of complexity on top of the project. Android is relatively complex compared to standard Java due to how the Java code integrates with the XML code of the screens. This intricacy causes issues with version control because merging can run into conflicts. There were many times during this capstone where the code would pull incorrectly from GitHub and would need to be fixed manually. As the semester progressed version control became easier due to learning the command line commands but was always scary to perform merges because everything could break.

Android Studio was another tool that needed to be mastered to finish the PEG. Eclipse was the prior version before Android Studio so learning Android Studio was critical to the project as Eclipse is no longer the sanctioned IDE for Android development. This was simple due to how similar Android Studio and Eclipse are but there are subtle differences that need to be understood to take advantage of Android Studio. Android Studio is a better less buggy version of Eclipse and has far less issues.

One specific feature that needed to be learned was saving and loading to an Android device. To do this the team set up a small activity to test what are known as shared preferences. Shared preferences are small files that contain information on variables on a per app basis. If the team could not find a suitable method to save and load, then the PEG would lose much of its functionality as a personal care guide. Luckily shared preference works out perfectly for the PEG.

Figure 18: Test screen for saving and loading

The most important thing that was learned was how to better utilize Android resources to design competent user interfaces. Using the action bar and proper placement of buttons and pictures is vital to Android app flow and can make or break any app. The user interface, while not the most complicated code wise, is vital because everyone that uses the app interacts with the user interface. Great care must be exerted when designing a user interface, if it is too complicated or too spread out then people will just delete the app and find a better one. There are too many apps on the app store, it is almost guaranteed that another app exists that tries to do what the app being developed does. The key is making a better app through a better user interface and competent underlying system. In the case of the PEG the underlying system to the user interface is the actual exercises. Without the exercises the PEG would be a generic health information app about carpal tunnel syndrome. Without the user interface then no one would be able to use the exercise system. The user interface and underlying systems merge to complete an app and both are important to the success of any given app.

5.2 Implications

Hopefully when the PEG is completed it will reach a wide audience and be useful to people. The PEG was originally thought up to be a personal Occupational Therapist that can be brought on the go. This is still the goal and purpose of the PEG.

From a developer perspective knowing some form of mobile development is important for the future. More and more applications are being ported or developed natively for Android and IOS. The future is on these devices for general purpose application that do not need a physical keyboard to function properly. Apps related to all fields are being developed daily and improve

the efficiency of the everyday person working in those fields. Mobile computing is important and being able to develop for it means that there will be plenty of opportunities ahead.

5.3 Recommendations for Future Development

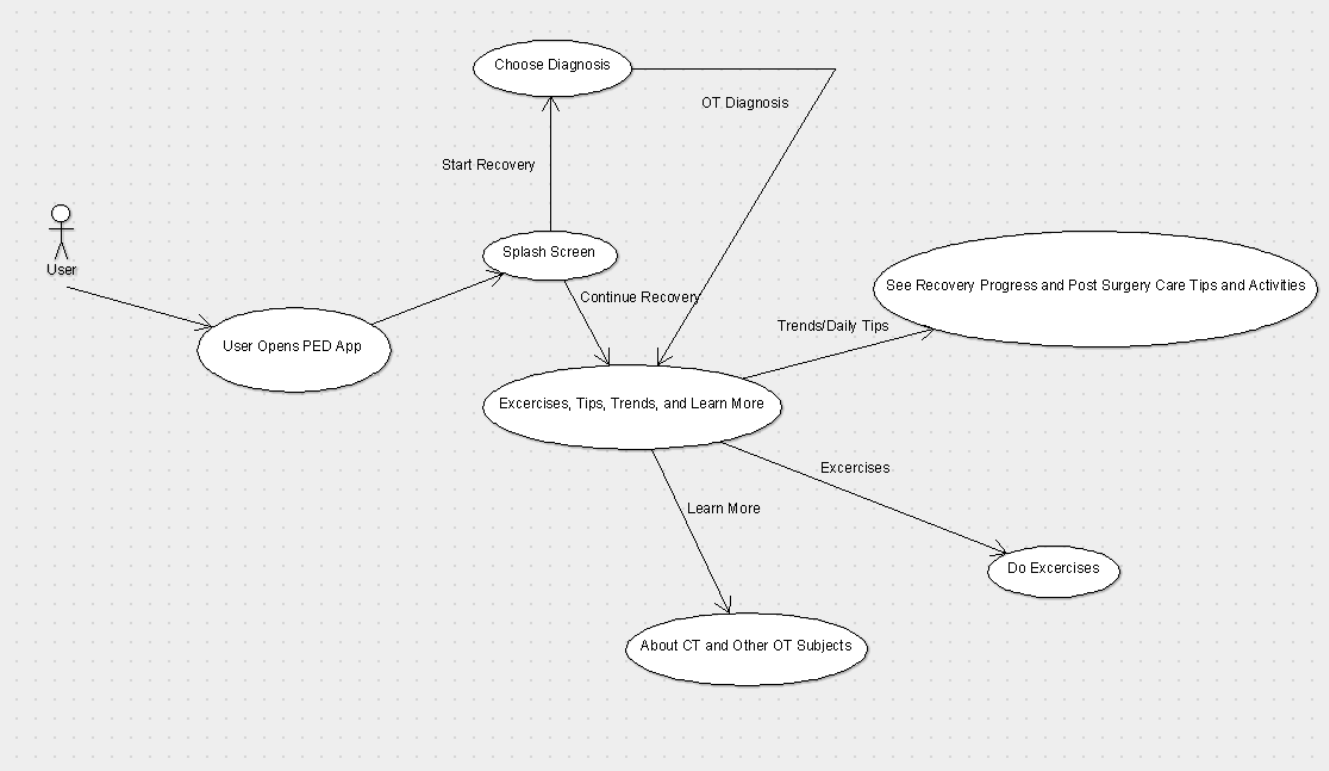
Unfortunately, the team was not able to implement all the ideas that were originally thought of due to time constraints. The recovery page needs to be fleshed out, adding the ability to log into different care programs would greatly extend the usefulness of the app. Implementing more information for the care guide and having a more structured exercise program would bring the PEG to the next level. Adding a website to track stats related to each person using the PEG would be a cool and useful feature but would be challenging with a small team. Having more than just carpal tunnel syndrome and making a general purpose care app with multiple exercises and guide lines would expand the possible user base. There are always improvements to be made on both the underlying system and user interface making each better is important to the future of the PEG.

5.4 Conclusion

The PEG, while limited, is a powerful little app that gives users the ability to treat or prevent carpal tunnel syndrome. This is complimented with a slick user interface to help guide the user through the app without causing them confusion or frustration. Hopefully the PEG will help lots of people treat their carpal tunnel syndrome or give them relevant information pertaining to carpal tunnel syndrome. The PEG is designed as a personal occupational therapist for Android devices and mostly succeeds but needs to be expanded to realize its full potential

Appendix A:

The flowing is the initial UML diagram for the PEG application.



Appendix B:

The following is the current Java source code for the PEG application. This does not include the layout XML files or Android manifest file.

```
package com.example.strykermclane.peg;

import android.content.Intent;
import android.os.Bundle;
import android.support.v4.widget.DrawerLayout;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.webkit.WebView;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;

public class careguide extends AppCompatActivity {

    private DrawerLayout mDrawerLayout;
    private ListView mDrawerList;
    private ArrayAdapter<String> mAdapter;
    int currentSection = 0;
    String htmlAsString;

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

        Intent myIntent = getIntent();

        getContent();

        mDrawerLayout = (DrawerLayout) findViewById(R.id.drawer_layout);
        mDrawerList = (ListView)findViewById(R.id.navList);
        String[] sectionArray = {"Intro","Week 1","Week 2","Week 3","Week 4"};
        mAdapter = new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1,
sectionArray);
        mDrawerList.setAdapter(mAdapter);

        mDrawerList.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View view, int position,
long id) {
                displayView(position);
                mDrawerLayout.closeDrawer(mDrawerList);
            }
        });
    }

    private void displayView(int position){
        switch(position){
            case 0:
                currentSection = 0;
                break;
            case 1:
                currentSection = 1;
                break;
```

Appendix B

```
            case 2:
                currentSection = 2;
                break;
            case 3:
                currentSection = 3;
                break;
            case 4:
                currentSection = 4;
                break;
            default:
                currentSection = 0;
                break;
        }
        getContent();

    }

    private void getContent(){

        // get our html content
        switch(currentSection) {
            case 0:
                htmlAsString = getString(R.string.careGuide0);      // used by WebView
                break;
            case 1:
                htmlAsString = getString(R.string.careGuide1);
                break;
            case 2:
                htmlAsString = getString(R.string.careGuide2);
                break;
            case 3:
                htmlAsString = getString(R.string.careGuide3);
                break;
            case 4:
                htmlAsString = getString(R.string.careGuide4);
                break;
        }
        // set the html content on a Webview;
        WebView webView = (WebView) findViewById(R.id.webView);
        webView.loadDataWithBaseURL(null, htmlAsString, "text/html", "utf-8", null);

    }


}
```

```
package com.example.strykermclane.peg;

import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.Bundle;
import android.os.Vibrator;
import android.util.Log;
import android.view.WindowManager;
import android.widget.Toast;


public class exerciseActivity extends Activity implements SensorEventListener{
```

Appendix B

```
    private static SensorManager sensorManager;
    private MyCompassView compassView;
    private Sensor sensor;
    int exerSwitchCompass;
    private static Vibrator vibrateManager;
    private Sensor accelerometer;
    private Sensor magnetometer;
    private float[] accelVals;
    private float[] compassVals;
    static final float ALPHA = 0.15f;
    public float orientationGesture;

    /** Called when the activity is first created. */

    protected float[] lowPass( float[] input, float[] output ) {
        if ( output == null ) return input;

        for ( int i=0; i<input.length; i++ ) {
            output[i] = output[i] + ALPHA * (input[i] - output[i]);
        }
        return output;
    }
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        compassView = new MyCompassView(this);
        setContentView(compassView);

        getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);

        Intent myIntent = getIntent();
        exerSwitchCompass = (myIntent.getIntExtra("exerSwitch",3));
        switch(exerSwitchCompass){
            case 0:
                compassView.exerSwitchCompass = 0;
                break;
            case 1:
                compassView.exerSwitchCompass = 1;
                break;
            case 2:
                compassView.exerSwitchCompass = 2;
                break;
        }

        // Setup the sensors
        sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        accelerometer = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
        if (accelerometer != null) {
            sensorManager.registerListener(this, accelerometer,
SensorManager.SENSOR_DELAY_UI);
            Log.i("Compass MainActivity", "Registerered for ORIENTATION Sensor");

        }
        else {
            Log.d("Compass MainActivity", "accelerometer is null");
            Toast.makeText(this, "accelerometer is null",
                    Toast.LENGTH_LONG).show();
            finish();
        }

        magnetometer = sensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD);
        if (magnetometer != null) {
```

Appendix B

```
            sensorManager.registerListener(this, magnetometer,
SensorManager.SENSOR_DELAY_UI);
            Log.i("Compass MainActivity", "Registerered for ORIENTATION Sensor");
        }
        else{
                Log.d("Compass MainActivity", "magnetometer is null");
                Toast.makeText(this, "magnetometer is null, BIATCH!",
                        Toast.LENGTH_LONG).show();
                finish();
        }

    }



        @Override
        public void onAccuracyChanged(Sensor sensor, int accuracy) {
        }

        @Override
        public void onSensorChanged(SensorEvent event) {

            vibrateManager = (Vibrator) getSystemService(Context.VIBRATOR_SERVICE);

            int sensorType = event.sensor.getType();
            switch (sensorType) {
                case Sensor.TYPE_ACCELEROMETER:
                    accelVals = lowPass( event.values.clone(), accelVals );
                    break;
                case Sensor.TYPE_MAGNETIC_FIELD:
                    compassVals = lowPass( event.values.clone(), compassVals );
                    break;
                default:
                    Log.w("Compass MainActivity", "Unknown sensor type " +
sensorType);
                    return;

            } if (accelVals != null && compassVals != null) {
                float R[] = new float[9];
                float I[] = new float[9];
                boolean success = SensorManager.getRotationMatrix(R, I, accelVals,
compassVals);
                if (success) {
                    float orientation[] = new float[3];
                    SensorManager.getOrientation(R, orientation);

                    switch (exerSwitchCompass) {
                        case 0:
                            float azimuth = orientation[0];
                            int azimuthDeg = (int)
Math.round(Math.toDegrees(azimuth));
                            if (((azimuthDeg - compassView.defaultDeviation > 40) ||
(azimuthDeg - compassView.defaultDeviation < -25)) && compassView.vibeSwitch) {
                                vibrateManager.vibrate(100);
                            }
                            orientationGesture = azimuthDeg + 270;
                            break;
                        case 1:
                            float pitch = orientation[1];
                            int pitchDeg = (int) Math.round(Math.toDegrees(pitch));
                            if ((pitchDeg > 60) || (pitchDeg < -60)) {
                                vibrateManager.vibrate(100);
```

Appendix B

```
                            }
                            if((pitchDeg > orientationGesture - 270) && (pitchDeg > 0)
&& (orientationGesture - 270 > 0) && (pitchDeg > compassView.maxUpper)){
                                compassView.maxUpper = pitchDeg;
                            }
                            else if((pitchDeg < orientationGesture - 270) && (pitchDeg
< 0) && (orientationGesture - 270 < 0) && (pitchDeg < compassView.maxLower)){
                                compassView.maxLower = pitchDeg;
                            }
                            orientationGesture = pitchDeg + 270;
                            break;
                        case 2:
                            float roll = orientation[2];
                            int rollDeg = (int) Math.round(Math.toDegrees(roll));
                            if ((rollDeg > 60) || (rollDeg < -60)) {
                                vibrateManager.vibrate(100);
                            }
                            if((rollDeg > orientationGesture - 270) && (rollDeg > 0)
&& (orientationGesture - 270 > 0) && (rollDeg > compassView.maxUpper)){
                                compassView.maxUpper = rollDeg;
                            }
                            else if((rollDeg < orientationGesture - 270) && (rollDeg <
0) && (orientationGesture - 270 < 0) && (rollDeg < compassView.maxLower)){
                                compassView.maxLower = rollDeg;
                            }
                            orientationGesture = rollDeg + 270;
                            break;
                    }
                    compassView.updateData(orientationGesture);
                }
            }


        }


    @Override
    protected void onResume() {
        super.onResume();
        sensorManager.registerListener(this, accelerometer,
SensorManager.SENSOR_DELAY_UI);
        sensorManager.registerListener(this, magnetometer,
SensorManager.SENSOR_DELAY_UI);
    }

    @Override
    protected void onPause() {
        super.onPause();
        sensorManager.unregisterListener(this);
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        if (sensor != null) {
            sensorManager.unregisterListener(this);
        }
    }


}
```

Appendix B

```
package com.example.strykermclane.peg;

import android.content.Intent;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.Spinner;

public class HandExercises extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_hand_exercises);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        setTitle("Exercises");

        Spinner spinner = (Spinner) findViewById(R.id.spinner);
// Create an ArrayAdapter using the string array and a default spinner layout
        ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(this,
                R.array.choices, R.layout.spinner);
// Specify the layout to use when the list of choices appears
        adapter.setDropDownViewResource(R.layout.spinner_drop_down);
// Apply the adapter to the spinner
        spinner.setAdapter(adapter);


    }
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.menu_exercise_selection, menu);
        return true;
    }
    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();

        //noinspection SimplifiableIfStatement
        if (id == R.id.bar_CareGuide2) {
            Intent myIntent = new Intent(this, careguide.class);
            startActivity(myIntent);
        }
        else if(id == R.id.bar_recover){
            Intent myIntent = new Intent(this, Recover.class);
            startActivity(myIntent);
        }
        return super.onOptionsItemSelected(item);
    }

    public void goToExercise(View view){
        Spinner spinner = (Spinner) findViewById(R.id.spinner);
        if (spinner.getSelectedItem().toString().equals("Flexion Extension")){
            /**
             * This method bring the user to the next screen.
             *
             *show the usage of various javadoc Tags.
```

Appendix B

```
             * @param view
             * @return Nothing
             */
            int exerSwitch = 3;
            Intent intentFlexionExtension = new Intent(this, exerciseActivity.class);
            intentFlexionExtension.putExtra("exerSwitch",2);
            startActivity(intentFlexionExtension);
        }
        else if(spinner.getSelectedItem().toString().equals("Pronation Supination")){
            /**
             * This method bring the user to the next screen.
             *
             *show the usage of various javadoc Tags.
             * @param view
             * @return Nothing
             */
            int exerSwitch = 3;
            Intent intentPronationSupination = new Intent(this,
exerciseActivity.class);
            intentPronationSupination.putExtra("exerSwitch",1);
            startActivity(intentPronationSupination);
        }
        else if(spinner.getSelectedItem().toString().equals("Radial Ulnar
Deviation")){
            /**
             * This method bring the user to the next screen.
             *
             *show the usage of various javadoc Tags.
             * @param view
             * @return Nothing
             */
            int exerSwitch = 3;
            Intent intentRadialUlnarDeviation = new Intent(this,
exerciseActivity.class);
            intentRadialUlnarDeviation.putExtra("exerSwitch",0);
            startActivity(intentRadialUlnarDeviation);
        }
    }
}

package com.example.strykermclane.peg;

import android.content.Intent;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;

/**
 * <h1>Landing Page</h1>
 *
 *
 *
 * @author  Stryker McLane
 * @version 1.0
 * @since   1/2016
 */

public class LandingPage extends AppCompatActivity {

    public final static String EXTRA_MESSAGE = "com.mycompany.myfirstapp.MESSAGE";
```

Appendix B

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_landing_page);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        setTitle("PEG");
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.menu_landing_page, 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();

        //noinspection SimplifiableIfStatement
        if (id == R.id.bar_CareGuide) {
            Intent myIntent = new Intent(this, careguide.class);
            startActivity(myIntent);
        }

        return super.onOptionsItemSelected(item);
    }

    /** Called when the user clicks the Send button */

    public void goToHand(View view){
        Intent myIntent = new Intent(this, HandExercises.class);
        startActivity(myIntent);
    }
    public void goToRecovery(View view){
        Intent myIntent = new Intent(this, Recover.class);
        startActivity(myIntent);
    }
}

package com.example.strykermclane.peg;

import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Path;
import android.graphics.RectF;
import android.graphics.SweepGradient;
import android.util.Log;
import android.view.MotionEvent;
import android.view.View;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
```

Appendix B

```
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;

public class MyCompassView extends View {

    // max unlar deviation line: 30 - 39 deg
    // max radial deviation line: 20 - 25 deg
    // max flexion line: 60 - 80 deg
    // max extension line: 60 - 75 deg
    private int maxFlexion = -60;
    private int maxExtension = 60;
    private int maxPronation = 60;
    private int maxSupination = -60;
    private int maxUlnarDeviation = 40;
    private int maxRadialDeviation = -25;

    String newString = "blank";

    private int regulateValue(int currentPosition)
    {
        int regulatedValue = 0;
        regulatedValue = 90 - currentPosition;
        return regulatedValue;
    }

    private Paint paintGreen;
    private Paint paintRed;
    private Paint paintWhite;
    private Paint paintGradient;
    private float position = 0;
    public int defaultDeviation = 0;
    public int exerSwitchCompass;
    public boolean vibeSwitch = false;
    Bitmap calibrateButtonBMP;
    private String currentRotation;
    public int maxUpper = 0;
    public int maxLower = 0;
    int[] userMaximumArray = new int[2];

    public MyCompassView(Context context) {
        super(context);
        init();
    }

    private void init() {

        //paintGreen: used for pointer
        paintGreen = new Paint();
        paintGreen.setAntiAlias(true);
        paintGreen.setStrokeWidth(20);
        paintGreen.setTextSize(100);
        paintGreen.setStyle(Paint.Style.STROKE);
        paintGreen.setColor(Color.GREEN);

        //paintRed:
        paintRed = new Paint();
        paintRed.setAntiAlias(true);
        paintRed.setStrokeWidth(5);
        paintRed.setTextSize(100);
        paintRed.setStyle(Paint.Style.STROKE);
        paintRed.setColor(Color.WHITE);
```

Appendix B

```
        //paintWhite: used for current rotation reading
        paintWhite = new Paint();
        paintWhite.setAntiAlias(true);
        paintWhite.setStrokeWidth(5);
        paintWhite.setTextSize(250);
        paintWhite.setStyle(Paint.Style.FILL);
        paintWhite.setColor(Color.WHITE);

        //paintGradient: used for body of compass
        paintGradient = new Paint();
        paintGradient.setStrokeWidth(1);
        paintGradient.setStrokeCap(Paint.Cap.SQUARE);
        paintGradient.setStyle(Paint.Style.FILL);

        getUserMaximum();
    }

    @Override
    protected void onDraw(Canvas canvas) {

        int xPoint = getMeasuredWidth() / 2;
        int yPoint = getMeasuredHeight() / 2;

        //Draw Background
        canvas.drawColor(Color.BLACK);

        //Draw Calibrate BMP if exerSwictch = 0 (Radial / Ulnar Deviation Exercise
Selected)
        calibrateButtonBMP =
BitmapFactory.decodeResource(getResources(),R.drawable.calibratebutton);
        if(exerSwitchCompass == 0) {
            canvas.drawBitmap(calibrateButtonBMP, 0, 0, null);
        }

        //Calculate radius of circle based on screen size
        float radius = (float) (Math.max(xPoint, yPoint) * 0.5);

        //rect1: rectangle used to create compass arc
        RectF rect1 = new RectF(xPoint - radius, yPoint - radius, xPoint + radius,
yPoint + radius);

        int[] colors = {Color.RED, Color.BLACK, Color.RED};
        float[] positions = {0f, 0.5f, 1f};
        SweepGradient gradient = new SweepGradient(100, 100, colors, positions);
        paintGradient.setShader(gradient);;

        //

        //tempX, tempY: coordinates for endpoint of neutral position line
        float tempX = (float) (xPoint + radius * Math.sin((double) (regulateValue(0))
/ 180 * 3.143));
        float tempY = (float) (yPoint - radius * Math.cos((double) (regulateValue(0))
/ 180 * 3.143));

        Path p = new Path();
        p.moveTo(xPoint, yPoint);
        p.lineTo(tempX, tempY); // neutral position line
        //canvas.drawText(String.valueOf(defaultDeviation), tempX, tempY, paintRed);
        canvas.drawText(String.valueOf(newString), tempX, tempY, paintRed);
        p.moveTo(xPoint, yPoint);
```

Appendix B

```
        switch(exerSwitchCompass) {
            case 0:
                //30 degrees
                tempX = (float) (xPoint + radius * Math.sin((double)
(regulateValue(maxUlnarDeviation))  / 180 * 3.143));
                tempY = (float) (yPoint - radius * Math.cos((double)
(regulateValue(maxUlnarDeviation)) / 180 * 3.143));
                p.lineTo(tempX, tempY); // top line
                canvas.drawText(("Ulnar"), tempX, tempY - 100, paintRed);
                canvas.drawText(("Deviation"), tempX, tempY, paintRed);
                p.moveTo(xPoint, yPoint);
                //-25 degrees
                tempX = (float) (xPoint + radius * Math.sin((double)
(regulateValue(maxRadialDeviation))  / 180 * 3.143));
                tempY = (float) (yPoint - radius * Math.cos((double)
(regulateValue(maxRadialDeviation)) / 180 * 3.143));
                p.lineTo(tempX, tempY); // bottom line
                canvas.drawText(("Radial"), tempX, tempY + 100, paintRed);
                canvas.drawText(("Deviation"), tempX, tempY + 200, paintRed);
                canvas.drawArc(rect1, 25, 295, true, paintGradient);
                break;
            case 1:
                tempX = (float) (xPoint + radius * Math.sin((double)
(regulateValue(maxPronation)) / 180 * 3.143));
                tempY = (float) (yPoint - radius * Math.cos((double)
(regulateValue(maxPronation)) / 180 * 3.143));
                p.lineTo(tempX, tempY); // top line
                canvas.drawText(("Pronation"), tempX, tempY, paintRed);
                p.moveTo(xPoint, yPoint);
                tempX = (float) (xPoint + radius * Math.sin((double)
(regulateValue(maxSupination)) / 180 * 3.143));
                tempY = (float) (yPoint - radius * Math.cos((double)
(regulateValue(maxSupination)) / 180 * 3.143));
                p.lineTo(tempX, tempY); // bottom line
                canvas.drawText(("Supination"), tempX, tempY, paintRed);
                canvas.drawArc(rect1, 60, 240, true, paintGradient);
                break;
            case 2:
                tempX = (float) (xPoint + radius * Math.sin((double)
(regulateValue(maxExtension)) / 180 * 3.143));
                tempY = (float) (yPoint - radius * Math.cos((double)
(regulateValue(maxExtension)) / 180 * 3.143));
                p.lineTo(tempX, tempY); // top line
                canvas.drawText(("Extension"), tempX, tempY, paintRed);
                p.moveTo(xPoint, yPoint);
                tempX = (float) (xPoint + radius * Math.sin((double)
(regulateValue(maxFlexion)) / 180 * 3.143));
                tempY = (float) (yPoint - radius * Math.cos((double)
(regulateValue(maxFlexion)) / 180 * 3.143));
                p.lineTo(tempX, tempY); // bottom line
                canvas.drawText(("Flexion"), tempX, tempY, paintRed);
                canvas.drawArc(rect1, 60, 240, true, paintGradient);
                break;
        }
        if((exerSwitchCompass == 1) || (exerSwitchCompass == 2)) {
            p.moveTo(xPoint, yPoint);
            tempX = (float) (xPoint + radius * Math.sin((double)
(regulateValue(maxUpper)) / 180 * 3.143));
            tempY = (float) (yPoint - radius * Math.cos((double)
(regulateValue(maxUpper)) / 180 * 3.143));
            p.lineTo(tempX, tempY); // top line
            canvas.drawText(("Max:" + maxUpper + "\u00B0"), getMeasuredWidth() - 500,
getMeasuredHeight() / 2, paintRed);
```

Appendix B

```
            p.moveTo(xPoint, yPoint);
            tempX = (float) (xPoint + radius * Math.sin((double)
(regulateValue(maxLower)) / 180 * 3.143));
            tempY = (float) (yPoint - radius * Math.cos((double)
(regulateValue(maxLower)) / 180 * 3.143));
            p.lineTo(tempX, tempY); // bottom line
            canvas.drawText(("Max:" + maxLower + "\u00B0"), getMeasuredWidth() - 500,
(getMeasuredHeight() / 2)+ 100, paintRed);
            canvas.drawArc(rect1, 60, 240, true, paintGradient);
        }


        canvas.drawPath(p, paintRed);



        if(exerSwitchCompass == 0) {
            position -= defaultDeviation;
        }


        //Draw Pointer
        canvas.drawLine(xPoint,
                yPoint,
                (float) (xPoint + radius
                        * Math.sin((double) (-position) / 180 * 3.143)),
                (float) (yPoint - radius
                        * Math.cos((double) (-position) / 180 * 3.143)), paintGreen);


        currentRotation = String.valueOf(position - 270) + "\u00B0";
        //Draw current Rotation Value
        canvas.drawText(currentRotation, xPoint - (radius), yPoint + 100, paintWhite);
    }

    public void updateData(float position) {
        this.position = position;
        invalidate();
    }

    @Override
    public boolean onTouchEvent(MotionEvent ev) {

        switch (ev.getAction()) {

            case MotionEvent.ACTION_DOWN: {
                calibrateButtonClicked();
                syncUserMaximum();

                break;
            }
            case MotionEvent.ACTION_MOVE: {
                break;
            }
            case MotionEvent.ACTION_UP:
                break;
        }
        return true;
    }

    void calibrateButtonClicked(){
        defaultDeviation = (int) position - 270;
        if(exerSwitchCompass == 0){
```

Appendix B

```
            vibeSwitch = true;
        }
        invalidate();
    }

    void syncUserMaximum(){
        Context context = this.getContext();
        String maxUpperString = Integer.toString(maxUpper);
        String maxLowerString = Integer.toString(maxLower);
        String userMax = maxUpperString +","+ maxLowerString;
        try {
            OutputStreamWriter outputStreamWriter = new
OutputStreamWriter(context.openFileOutput("recovery.txt", Context.MODE_PRIVATE));
            outputStreamWriter.write(userMax);
            outputStreamWriter.close();
        }
        catch (IOException e) {
            Log.e("Exception", "File write failed: " + e.toString());
        }
    }private String readFromFile() {
        Context context = this.getContext();

        String ret = "";

        try {
            InputStream inputStream = context.openFileInput("recovery.txt");

            if ( inputStream != null ) {
                InputStreamReader inputStreamReader = new
InputStreamReader(inputStream);
                BufferedReader bufferedReader = new BufferedReader(inputStreamReader);
                String receiveString = "";
                StringBuilder stringBuilder = new StringBuilder();

                while ( (receiveString = bufferedReader.readLine()) != null ) {
                    stringBuilder.append(receiveString);
                }

                inputStream.close();
                ret = stringBuilder.toString();
            }
        }
        catch (FileNotFoundException e) {
            Log.e("login activity", "File not found: " + e.toString());
        } catch (IOException e) {
            Log.e("login activity", "Can not read file: " + e.toString());
        }

        return ret;
    }

    void getUserMaximum(){
        newString = readFromFile();
        int[] userMaxInput = new int[2];
        String[] userMaxInputString;
        userMaxInputString = newString.split(",",2);
        for(int i = 0;i<2;i++){
            userMaxInput[i] = Integer.valueOf(userMaxInputString[i]);
        }
        maxUpper = userMaxInput[0];
        maxLower = userMaxInput[1];
    }
```

Appendix B

```
}

package com.example.strykermclane.peg;

import android.content.Intent;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.TextView;

public class Recover extends AppCompatActivity {
    private SharedPreferences preferenceSettings;
    private SharedPreferences.Editor preferenceEditor;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_recover);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

    }

    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.recover_menu, menu);
        return true;
    }

    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();

        //noinspection SimplifiableIfStatement
        if (id == R.id.bar_CareGuide) {
            Intent myIntent = new Intent(this, careguide.class);
            startActivity(myIntent);
        }
        else if(id == R.id.bar_Exercise){
            Intent myIntent = new Intent(this, HandExercises.class);
            startActivity(myIntent);
        }

        return super.onOptionsItemSelected(item);
    }
    public void save(View view){
        TextView test = (TextView) findViewById(R.id.editText);
        String testSave = test.getText().toString();
        preferenceSettings = getPreferences(0);
        preferenceEditor = preferenceSettings.edit();
        preferenceEditor.putString("testing",testSave);
        preferenceEditor.commit();
        test.setText("");
    }
    public void load(View view){
        preferenceSettings = getPreferences(0);
        String toLoad = preferenceSettings.getString("testing", "");
```

Appendix B

```
        TextView loading = (TextView) findViewById(R.id.editText);
        loading.setText(toLoad);
    }
}
```

GitHub: https://github.com/slmclane/P

References

[1] Brand Guidelines. (n.d.). Retrieved February 01, 2016, from http://developer.android.com/distribute/tools/promote/brand.html

[2] Carpal Tunnel Syndrome. (2013, July 14). Retrieved February 01, 2016, from http://www.nytimes.com/health/guides/disease/carpal-tunnel-syndrome/surgery.html

[3] Carpal Tunnel Syndrome Fact Sheet. (n.d.). Retrieved February 01, 2016, from http://www.ninds.nih.gov/disorders/carpal_tunnel/detail_carpal_tunnel.htm

[4] Carpal tunnel syndrome. (n.d.). Retrieved February 01, 2016, from https://umm.edu/health/medical/reports/articles/carpal-tunnel-syndrome

[5] Dashboards. (n.d.). Retrieved February 01, 2016, from http://developer.android.com/about/dashboards/index.html

[6] Mobile Technology Upgrades Continue to Impact Healthcare Engagement. (n.d.). Retrieved February 01, 2016, from http://www.himss.org/news/newsdetail.aspx?itemnumber=41520

[7] Price, R. (2015). Android just achieved something it will take Apple years to do. Retrieved February 01, 2016, from http://www.businessinsider.com/android-1-billion-shipments-2014-strategy-analytics-2015-2?r=UK

[8] IPad and Tablet Apps for a Pediatric Occupational Therapist. (2013). Retrieved February 01, 2016, from https://www.webpt.com/blog/post/ipad-and-tablet-apps-pediatric-occupational-therapist

[9] Agile Vs Waterfall. (n.d.). Retrieved March 06, 2016, from https://www.sdlc.ws/agile-vs-waterfall/

[10] Google v/s Oracle : Gods in the Battlefield. (n.d.). Retrieved March 06, 2016, from http://thetechweekly.com/article/google-vs-oracle-gods-in-the-battlefield/

[11] Repetitive motion results in longest work absences : The Economics Daily : U.S. Bureau of Labor Statistics. (n.d.). Retrieved March 06, 2016, from http://www.bls.gov/opub/ted/2004/mar/wk5/art02.htm

[12] Software Testing Automation Company | Best Automation Tools for Testing Android Applications. (n.d.). Retrieved March 09, 2016, from http://qabok.com/best-automation-tools-for-testing-android-applications/

[13] What is Clean Code and why should you care? (2014). Retrieved March 06, 2016, from http://cvuorinen.net/2014/04/what-is-clean-code-and-why-should-you-care/

References

[14] Agile Methodology Understanding Agile Methodology. (n.d.). Retrieved March 09, 2016, from http://agilemethodology.org/

[15] Android Design Principles. (n.d.). Retrieved March 09, 2016, from http://developer.android.com/design/get-started/principles.html

[16] Software development process. (n.d.). Retrieved March 09, 2016, from http://america.pink/software-development-process_4084983.html

[17] Samsung Galaxy Tab 4 8.0 (2015) Coming with Snapdragon 410 Chipset. (2015). Retrieved March 06, 2016, from http://www.wlivenews.com/samsung-galaxy-tab-4-8-0-2015-coming-with-snapdragon-410-chipset.html

[18] IOS vs Android Development. (2014). Retrieved March 09, 2016, from http://thinkapps.com/blog/development/platform-build-first-ios-vs-android/