MOBILE ASSISTANT FOR PHYSICAL THERAPY EXERCISES FOR BENIGN POSITIONAL PAROXYSMAL VERTIGO

Design and Implementation of an Android Application Module "Workout Buddy"

MS Project Report

Advisor: Professor Ugo Buy

SUBMITTED BY: AMIR SHADAAB MOHAMMED

GRADUATION SEMESTER: SPRING 2014

UIN: 657929026

TABLE OF CONTENTS

SECTION 1: ACKNOWLEDGEMENT

I am very thankful to Professor Ugo Buy for his support and valuable advice during the project. I highly appreciate his guidance at every step and constant support and for providing relevant information related to the project. I would like to thank Bhavana Singh for completing the first module of the project that dealt with the base of the application and for documenting it in great detail which made it easier for me to understand the base of the Workout Buddy module.

I would like to thank Bucky Roberts for his video logs on YouTube explaining the Android Framework where I learnt how to code in an Android environment.

I would also like to thank Faisal Faroz of Business School at University of Illinois at Chicago for helping me with capturing the sensor readings by being the test subject. It was because of his support and enthusiasm for the project, I was able to capture the sensor readings at different exercise positions with great precision.

SECTION 2: INTRODUCTION

As mentioned in Bhavana’s documentation of the application [1], BPPV (Benign Paroxysmal Positional Vertigo) is a medical condition and a cause of peripheral vertigo in patients. Physical therapy is a useful remedy to alleviate this condition and helps train the brain to resist vertigo. Patients are prescribed to perform Brandt Daroff maneuvers that trigger vertigo. Over time this therapy helps reduce the BPPV incidents. For more information on BPPV, please refer to this article.

Bhavana’s application helped the users in tracking incidents and workouts and setting up reminders for workouts. The extension of this application will be a “Workout Buddy” module that will help users keep track of their exercises in real time. The user places the phone on his head and performs the Brandt Daroff maneuvers and the workout buddy application module tracks the exercise movements and gives real time audio feedback on how the user is performing the maneuvers.

The application keeps track of the accelerometer, magnetometer and orientation readings and checks for head movements and gives audio feedback based on the exercise movements. The user can select the number of cycles in the exercise based on how serious his condition is and the workout buddy will make the user perform the exercises that many times. At the end of the exercise the user is asked to enter the number of times he experienced vertigo and this is stored in the database as an exercise log.

For brevity, Brandt Daroff exercises are referred to as simply exercise and 'Mobile Assistant for exercises for BPPV' as application in the rest of this document.

SECTION 3: APPLICATION REQUIREMENTS AND ASSUMPTIONS

This chapter lists the assumptions, functional and non-functional requirements of the application that have been implemented as part of this project. It also identifies requirements planned as future development. A list of stakeholders with their respective role descriptions in this project is as follows.

3.1 STAKEHOLDERS

1. Dr. Matthew Kircher – Client of the application who gave Bhavana valuable inputs on what the application should and should not do.

2. Professor Ugo Buy – Academic Advisor who guided with design and implementation of the application.

3. Bhavana Singh – executed design, development and testing of the base application except the “Workout Buddy”.

4. Amir Shadaab Mohammed – executed design, development and testing of “Workout Buddy” module of the application.

5. Users of the application – The users who will install the application from Google play store and use it.

3.2 FUNCTIONAL REQUIREMENTS

1. A button to start the exercise should be displayed on the screen in the Workout Buddy section.

2. The button to start should be large enough for user to be able to press when phone is attached to his head.

3. The exercise should start after 5 seconds to give the user enough time to get in the proper starting position.

4. The Workout Buddy section should have a number picker which allows the user to select the number of cycles he wants to perform in the exercise.

5. The maximum value for the cycles of exercise should be thirty as this is a large enough number for one set in Brandt Daroff exercises.

6. The minimum number of cycles of exercise should be one.

7. The actual exercise page should show the Accelerometer and Orientation readings for all three axis in real time. (Refer Section 4 for more information)

8. If the user is not in the sensor range during an exercise movement, that particular cycle should be repeated from the beginning.

9. Upon completion of each exercise step a temporary pop-up message should be displayed on the screen showing which exercise movement was completed.

10. Upon failure of any step, a temporary pop-up message should be displayed showing which exercise movement failed.

11. Any instructions playing should be stopped when the user moves away from the screen.

12. At the end of the exercise, the user should be given an option to enter the number of vertigos experienced

13. The user should be able to log all his exercises that were performed using the Workout Buddy.

3.3 NON-FUNCTIONAL REQUIREMENTS

1. The user interface should be intuitive and easy to use.

2. The instructions must be in simple English.

3. Response time of any kind of UI interaction should not exceed 3 seconds.

4. Exercises done with Workout Buddy should be marked specifically.

5. The application should not crash when the user gets a call or when the user moves away from the application.

6. The application should not crash upon changing the orientation of the phone in any angle or direction.

7. User should be able to download and install the application from web using provided instructions.

8. Tutorial video should be streaming without hitches provided an Internet connection is available.

9. The voice instructions of the application should be in clear voice and spoken in simple English.

10. User should not need to look or hold the phone on which the application is installed while exercising.

3.4 ASSUMPTIONS

1. The application has been developed with a minimum requirement of Android version 4.0 (Ice Cream Sandwich) and API level 14. The application will not function on lower android versions.

2. The application has been developed and tested on Android version 4.4 (Kit Kat) API level 19.

3. The environment used for development is eclipse IDE included in android ADT (Android Development Tools) running on Ubuntu release 12.04 (CPU – Intel I7, 8GB RAM).

4. The user’s phone has accelerometer and magnetometer sensors in them.

5. The exercise starts assuming that the user has the phone placed on his forehead in the portrait mode with the screen facing outwards. This is the point when the sensors are calibrated and this is considered as reference point for all exercises.

6. The user has a band that he can use to attach the phone on his head.

7. The user is not moving rapidly and is moving slowly during the exercise.

8. The user does not turn their heads more than 180 degrees during the exercises that requires them to turn their head.

SECTION 4: THINGS TO KNOW

WHAT ARE ACCELEROMETER AND ORIENTATION SENSORS?

Accelerometer Sensors: "An acceleration sensor measures the acceleration applied to the device, including the force of gravity.

Conceptually, an acceleration sensor determines the acceleration that is applied to a device by measuring the forces that are applied to the sensor itself." [6].

Orientation Sensors: "The orientation sensor lets you monitor the position of a device relative to the earth's frame of reference (specifically, magnetic north). The orientation sensor derives its data by using a device's geomagnetic field sensor in combination with a device's accelerometer." [6].

The axis along which these sensor readings are derived are shown in the figure below.

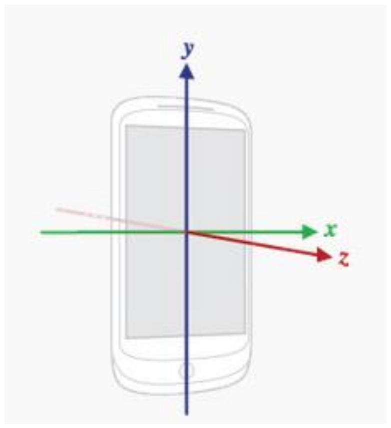


**Figure 1: Co-ordinates with respect to the phone.
Image from google Android developer's page [6].**

You can read more about Sensors in detail in Section 6.4.

WHAT IS AN ASYNCTASK?

In Android, the code runs on a thread known as (User Interface) UI thread. This thread is responsible for updating the UI, example: setting text fields, repainting the screen, etc. If we do heavy processing on this thread, our application hangs and the UI freezes. To avoid this, we need to do heavy processing in the background. We do this with the help of AsyncTask in Android. This helps us push all the heavy tasks in the background. Since this application module has heavy tasks like reading sensor values while another thread is simultaneously reading them, using sleep thread to stop tracking while instruction is being played, etc, we use AsyncTask. Refer [8] for more details.

WHAT IS AN ACTIVITY?

“In Android, an activity is a single and focused thing a user can do. All activities in some way interact with the user using the UI so the Activity class takes care of creating a window for you in which you can place your UI.” [7]

WHAT IS AN INTENT?

In Android, an intent is when you want to perform some kind of an action. In this application we will only be starting one activity from another activity using intents. To get from one activity to another we have to create intents and then start the intended activity using the startActivity(Intent i) method. We can also add extra values that we want to send to send to the next activity. We can then extract the values from the intent and use those values in the second activity.

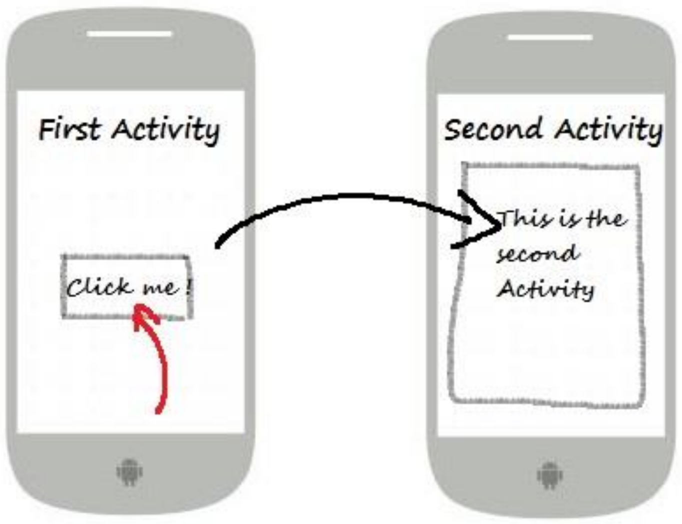


**Figure 2: Intent example.
Image from IT&C Solutions.**

SECTION 5: APPLICATION DESIGN

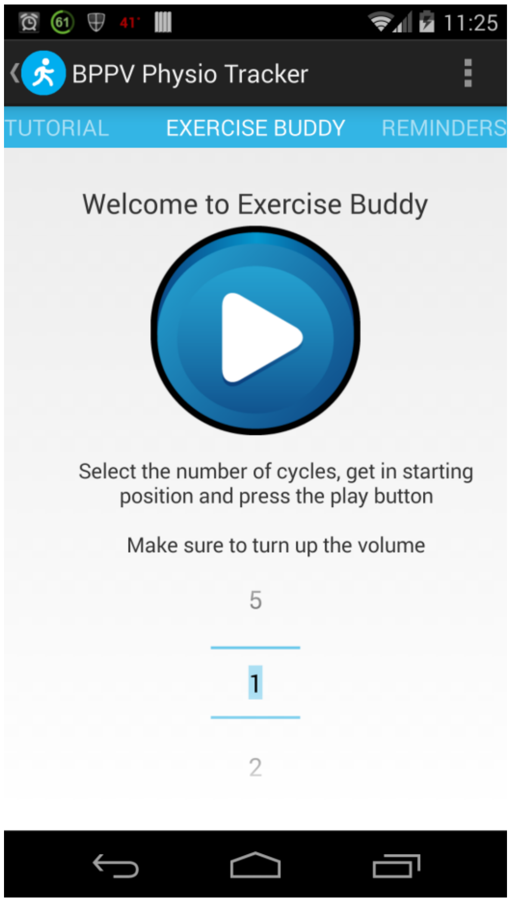
The application module "Workout Buddy" is designed with two main UI elements, the "Workout Buddy Fragment" and the "Workout Buddy activity". Everything else in the application runs in the background. The UI of the application is described below in detail. The application module works mostly with audio commands as most of the time during the exercise, the application interface will not be visible to the user performing the exercise as the phone will be attached to his head.

The application implements the Sensor Events and the AsyncTask classes. Both these classes are used to perform background processes of getting sensor values and tracking exercise movements in the background.

5.1 UI DESIGN

WORKOUT BUDDY VIEW FRAGMENT

The Workout Buddy View Fragment is designed with an image button, some text with instructions and a number picker as shown below. The user can select the number of cycles he wants to perform and clicks the play button. When clicking the image button, make sure the volume is turned up so that you can listen to the audio instructions that will be played on the next activity.



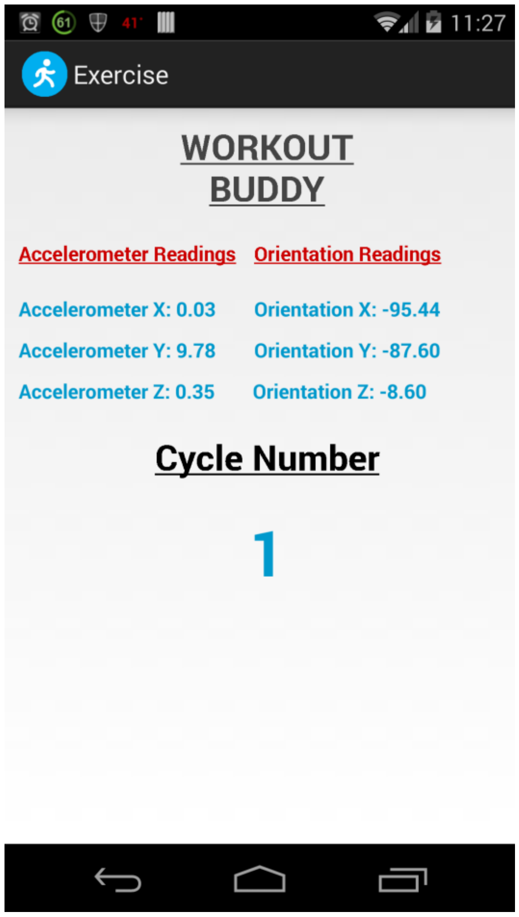
Select the number of cycles, get in starting position and press the play button

Make sure to turn up the volume

Figure 3: Workout Fragment with number picker and button

WORKOUT BUDDY ACTIVITY

This activity displays the sensors readings with respect to all three axis. This activity also displays the cycle number you are currently performing. This is the activity where all the audio instructions are given to the user based on the sensor readings being displayed on the screen.



WORKOUT BUDDY

Accelerometer Readings	Orientation Readings
Accelerometer X: 0.03	Orientation X: -95.44
Accelerometer Y: 9.78	Orientation Y: -87.60
Accelerometer Z: 0.35	Orientation Z: -8.60

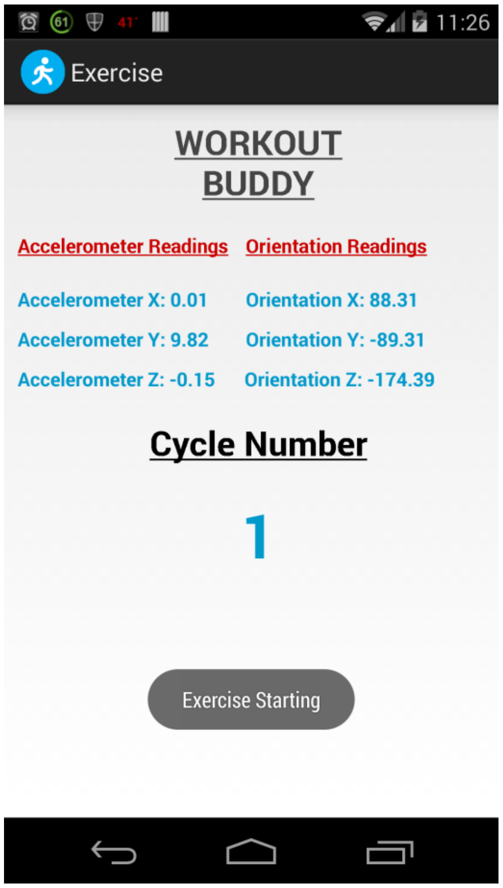
Cycle Number

1

Figure 4: Workout Activity showing sensor values and number of cycles

WORKOUT BUDDY ACTIVITY (MEDIA PLAYER AND TOAST MESSAGES)

Apart from displaying sensor readings, this activity also displays the toast messages based on what the user is doing in the exercise as shown below.

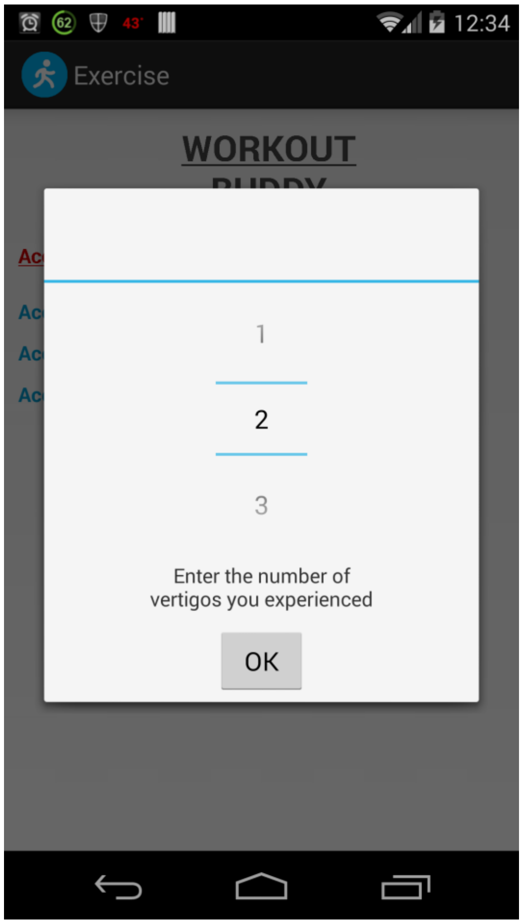


Accelerometer X: 0.01 Orientation X: 88.31
Accelerometer Y: 9.82 Orientation Y: -89.31
Accelerometer Z: -0.15 Orientation Z: -174.39

Figure 5: Workout activity showing a toast message

NUMBER OF VERTIGOS DIALOG FRAGMENT

At the end of each exercise a dialog fragment is displayed asking users the number of vertigos he experienced. When the user enters the number and clicks OK, the workout log is stored in the database.

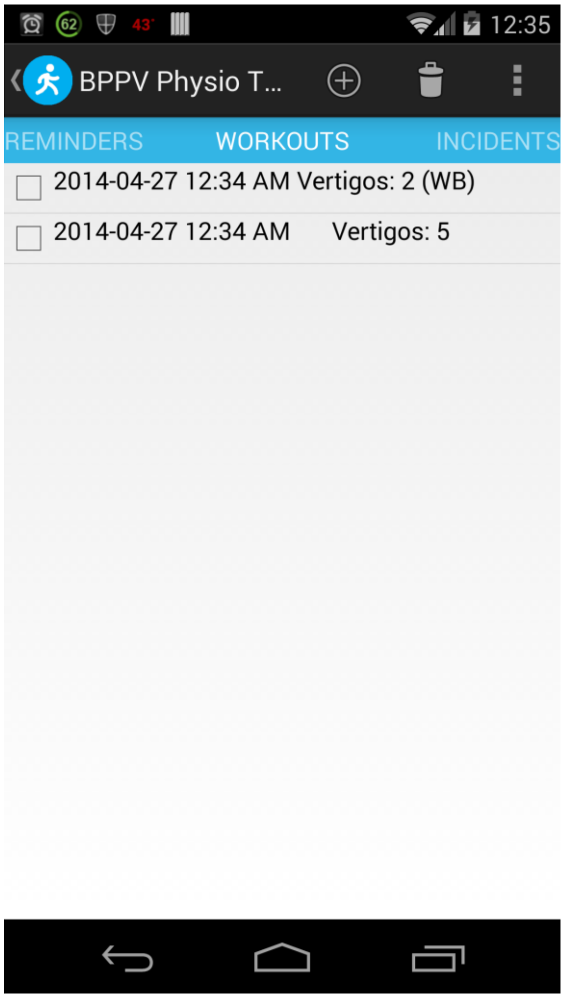


Enter the number of vertigos you experienced

Figure 6: Number picker to enter the number of vertigos experienced

WORKOUT LOG AFTER WORKOUT BUDDY EXERCISE

When the user swipes to the Workouts fragment in the main activity, the user can see his workout logged in the database with "(WB)" marked next to it indicating that the exercise was performed using the Workout Buddy



2014-04-27 12:34 AM Vertigos: 2 (WB)

2014-04-27 12:34 AM Vertigos: 5

Figure 7: Workout log showing workout exercises done using Workout Buddy

NEW INTERFACE FOR MAIN MENU

The main menu was redesigned with better and cleaner icons in the home screen as shown below. The application has a newer icon. Clicking each option would take you to the respective screen. Clicking the icon on top will get you back to the home screen. Icon images of different resolutions were used to be able to be clear and crisp on any kind of phone.

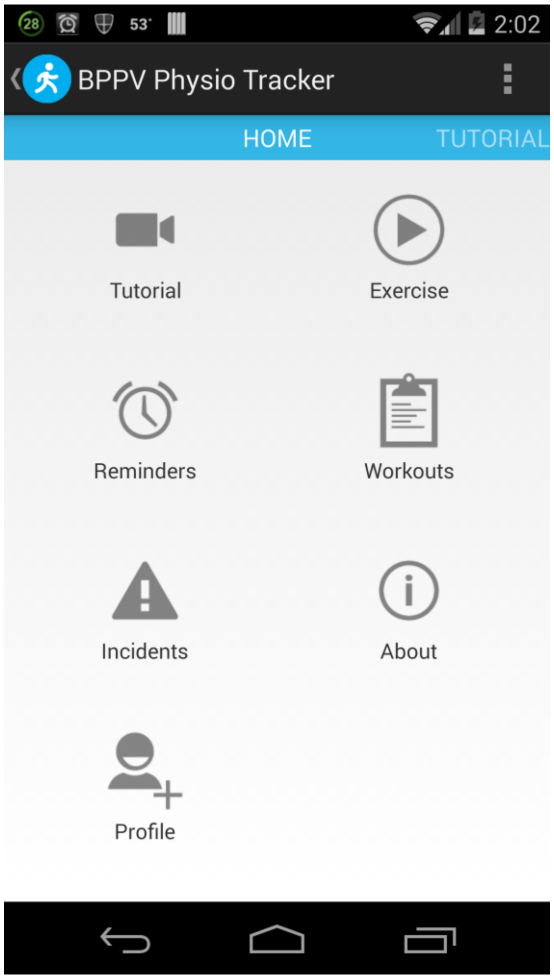


Figure 8: Newer and cleaner home screen icons

5.2 HOW TO USE THE WORKOUT BUDDY MODULE

The exercise needs to be done as described for a workout session to successfully complete. To use the “Workout Buddy” assistant properly please follow the guidelines as shown below.

STEP 1:

Swipe to the “Workout Buddy” fragment from the main menu as shown in figure 1 and select the number of cycles you want in your exercise set with the help of the number picker.

STEP 2:

Increase the volume of the device so the instructions can be heard clearly while performing the exercises.

STEP 3:

Find a flat surface like a couch or a bed and sit on it in the upright position and wear the device on your head in the portrait position with the screen facing outward as shown in the picture below:

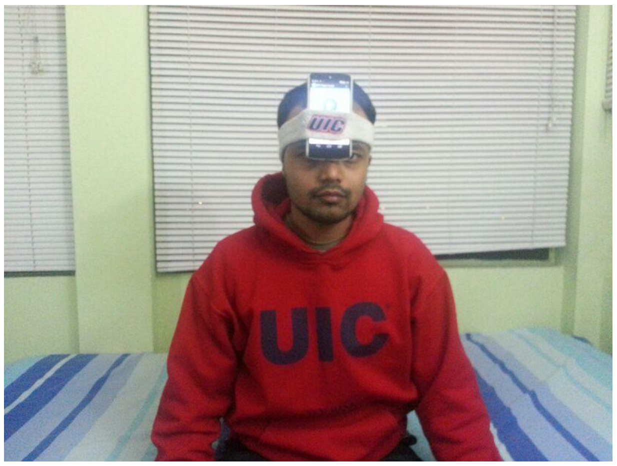


Figure 9: Starting Position

STEP 4:

Click the play button whenever you are ready to perform the exercise.

STEP 5:

The audio commands will now start. For the first few seconds, please stay still as this is the time when the sensors are capturing the starting position of your exercise and all the exercise movements that follow are recorded and checked with respect to the starting position.

STEP 6:

As the instruction to turn your head to the left is played, please turn your head 45 degrees to the left as shown in the picture below and remain in this position until the next instruction is played.

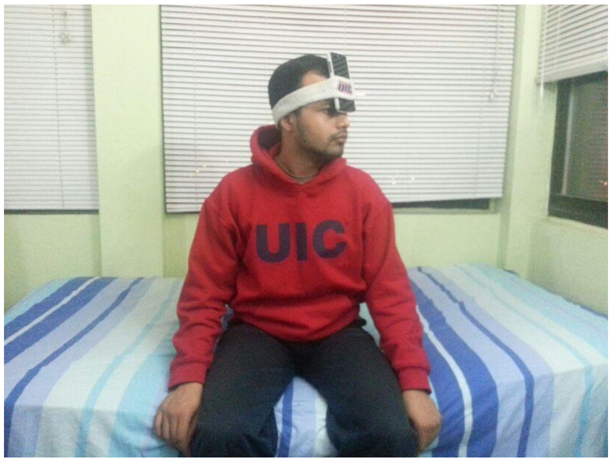


Figure 10: Left Head

STEP 7:

When the next instruction is played to lie down on your right shoulder with your head being at the same angle, please do so quickly and remain in this position until next instruction is played as shown in the picture below.

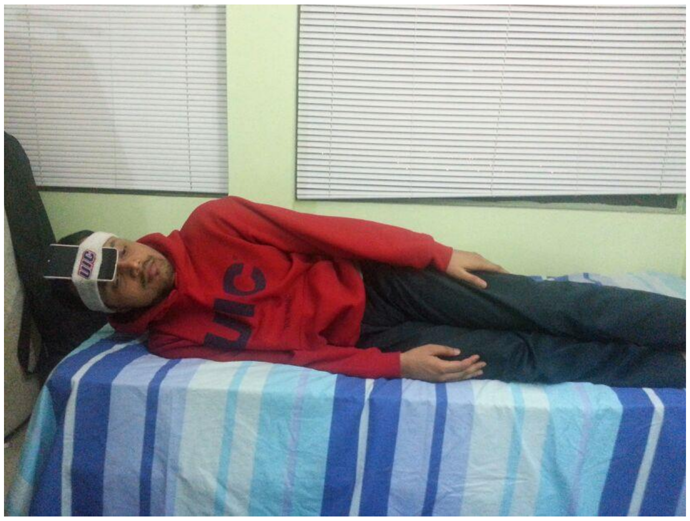


Figure 11: Right Shoulder

STEP 8:

You might feel dizzy if your BPPV condition persists. Stay in the same position until next instruction.

STEP 9:

Get back to the starting position as shown in step 3 and remain in this position until next instruction is played. Please stay still in this position as the sensors will calibrate again at this point.

STEP 10:

When the instruction is played please turn your head 45 degrees to the right as shown in the picture below and stay in the same position until next audio instruction is played.

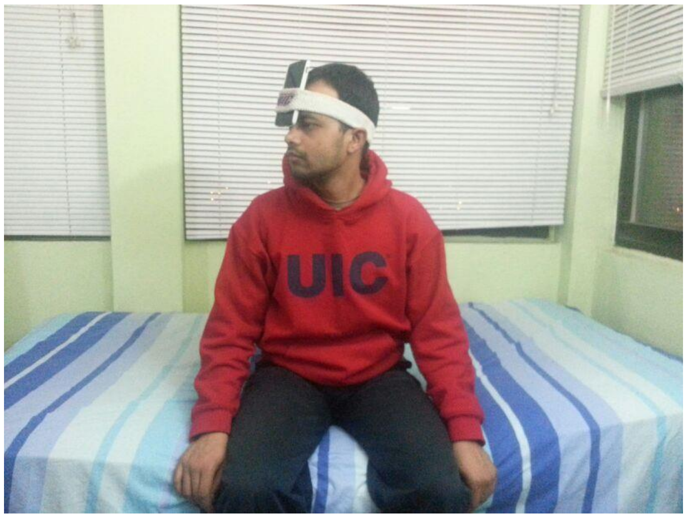


Figure 12: Right Head

STEP 11:

When the next instruction is played, please lie down on your left shoulder keeping your head at the same angle as shown in the picture below and remain in this position until next audio instruction is played.

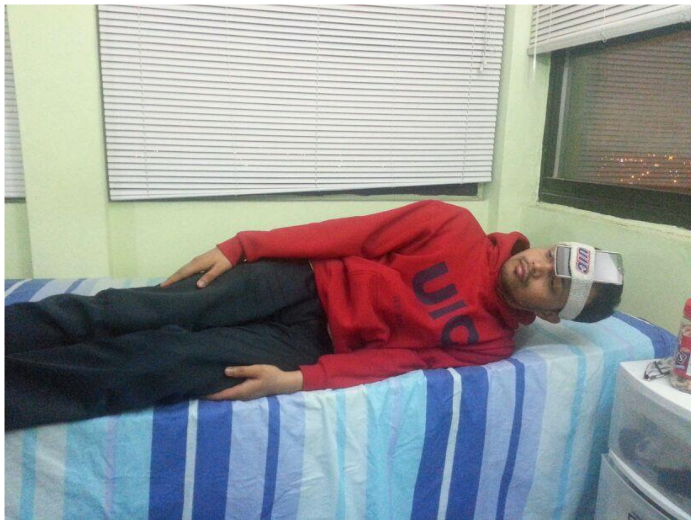


Figure 13: Left Shoulder

STEP 12:

When the next audio instruction is played, get back to the starting position as shown in step 3.

STEP 13:

Wait for audio that says that the exercise was completed successfully. You can cow remove the device from your head.

STEP 14:

Enter the number of vertigos you experienced using the number picker on the screen and click OK

SECTION 6: LOW LEVEL DETAILS

6.1 CLASS DIAGRAM

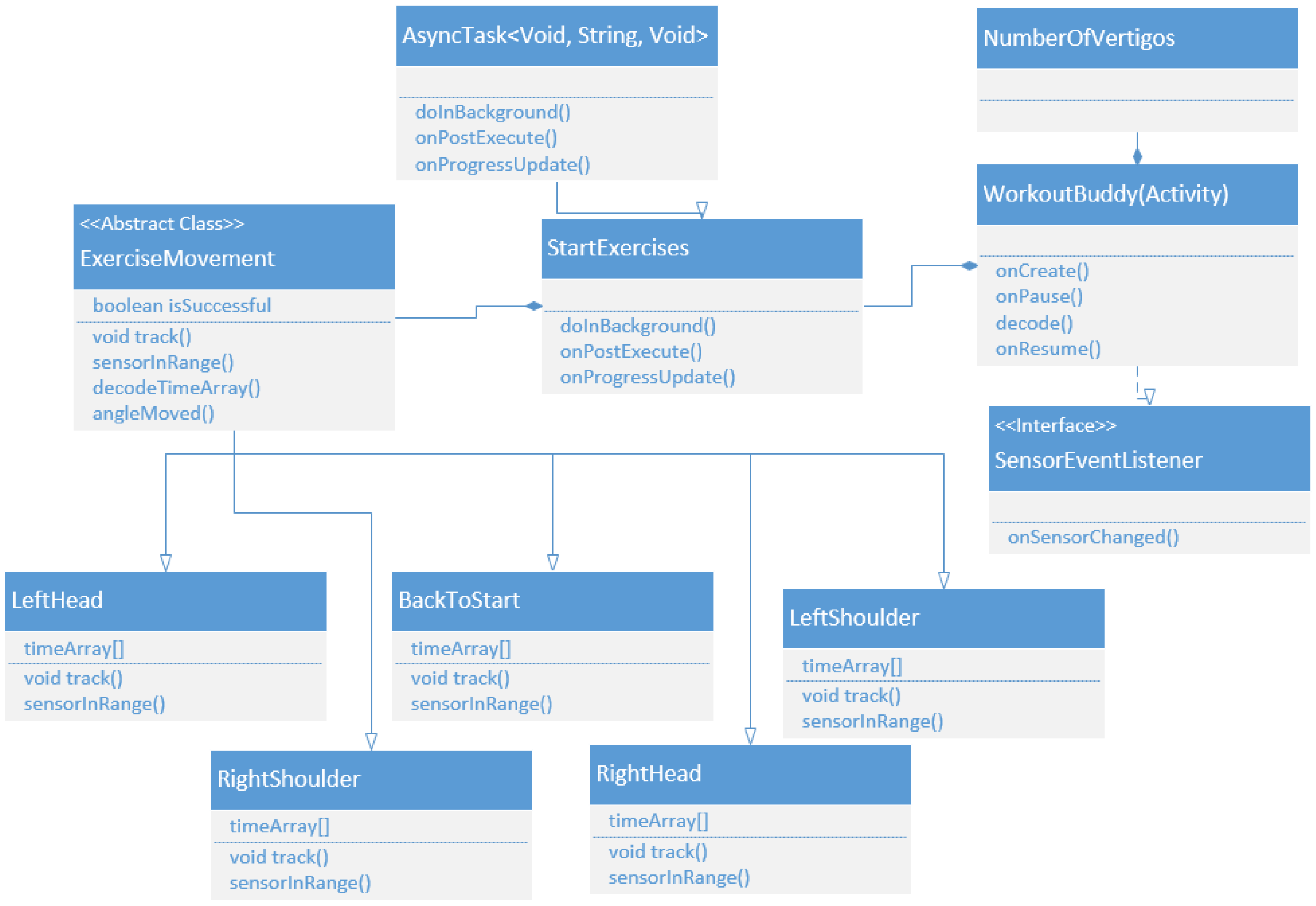


Figure 14: Class diagram of the Workout Buddy module of the application

Some key points from the class diagram:

1) The main class here in the WorkoutBuddy class as this is the activity that has the UI that displays the sensor readings and cycle count and the sensor readings and tracking the values is done in the background AyncTask contained in the WorkoutBuddy class.

2) StartExercises is the class that handles the background execution of the exercise movement tracking. It is nested within the WorkoutBuddy class and contains the actual exercise movement objects.

3) ExerciseMovements: This is the abstract class that is the super class of all the concrete exercise movements. This class makes sure that any class extending it should implement track() and sensorInRange() methods that are used to monitor the exercises. This uses strategy design pattern.

4) LeftHead: It extends the ExerciseMovement class and it is the class that has the method to track the exercise movement in which the head is rotated 45 degrees to the left. This method analyzes the head movement for 3 seconds and gives an audio feedback based on how the exercise was done.

5) RightHead: It extends the ExerciseMovement class and it is the class that has the method to track the exercise movement in which the head is rotated 45 degrees to the right. This method analyzes the head movement for 3 seconds and gives an audio feedback based on how the exercise was done.

6) RightShoulder: It extends the ExerciseMovement class and it is the class that has the method to track the exercise movement in which the user lies down on his right shoulder keeping the head tilted slightly up. This method analyzes the head movement for 15 seconds and gives an audio feedback based on how the exercise was done.

7) LeftShoulder: It extends the ExerciseMovement class and it is the class that has the method to track the exercise movement in which the user lies down on his left shoulder keeping the head tilted slightly up. This method analyzes the head movement for 15 seconds and gives an audio feedback based on how the exercise was done.

8) BackToStart: It extends the ExerciseMovement class and it is the class that has the method to track the exercise movement in which the user gets back from an exercise to his starting position. This method analyzes the head movement for 5 seconds and gives an audio feedback

based on how the exercise was done.

9) NumberOfVertigos: This is the class that manages the pop up that appears after the exercise is completed successfully to ask the user how many vertigos he experienced. Upon entering this number, the exercise time along with the number of vertigos are stored in the workout log in the database.

10) MainActivity: Although already developed by Bhavana, some slight changes were made to this class to be able to identify if the main activity was invoked from the Workout Buddy fragment. If it was, it means that the user successfully completed an exercise and a workout log needs to be added.

11) WorkoutBuddyViewFragment: This was a place holder in Bhavana's code. This was replaced by a UI that displays a button and a number picker and a few instructions for the user before he clicks the play button. Upon clicking the play button, the exercise is started and the WorkoutBuddy activity is started.

6.2 SEQUENENCE DIAGRAM

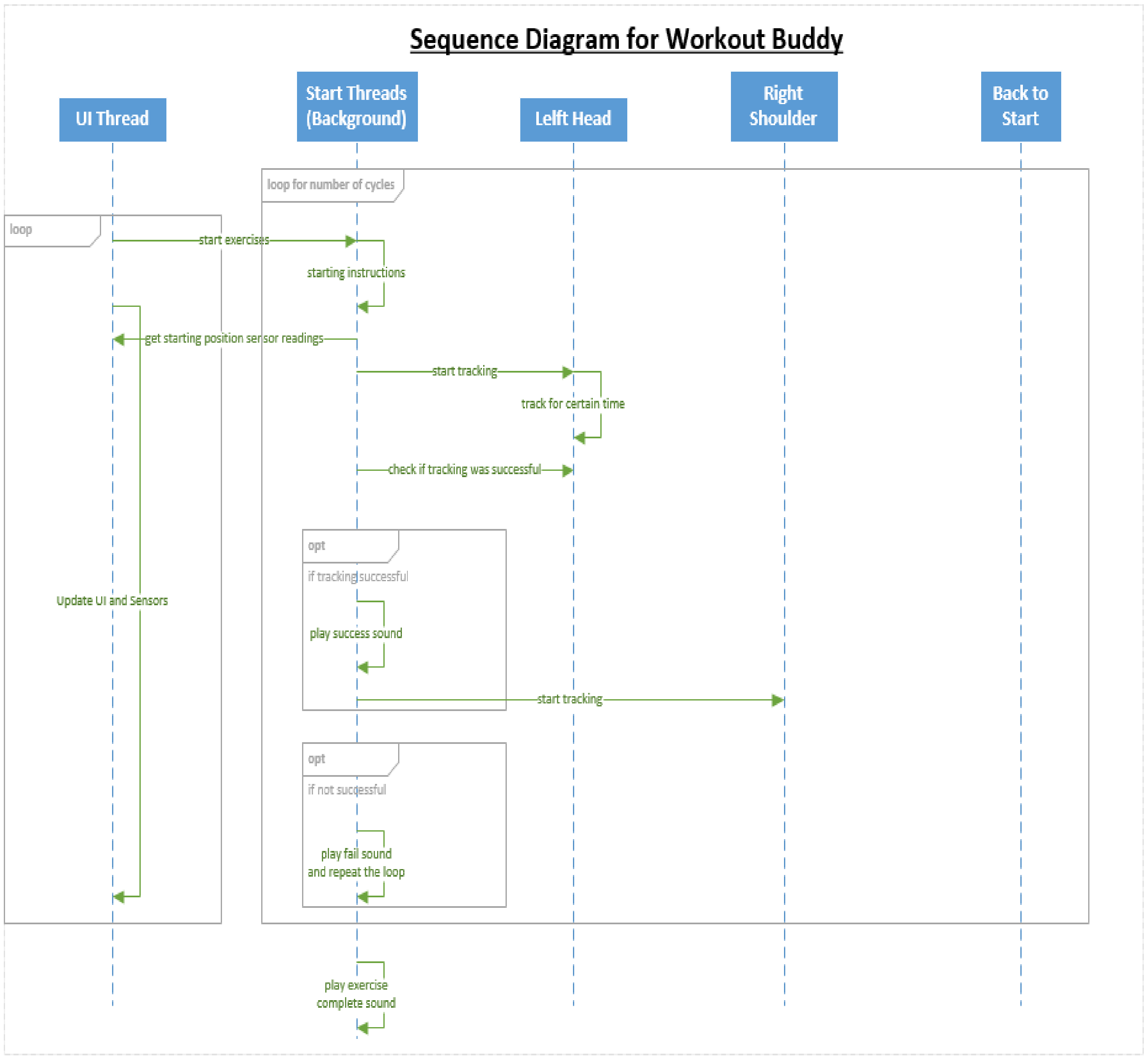


Figure 15: Sequence diagram showing how different elements in Workout Buddy module interact

The sequence diagram of the application shows that the Workout Buddy module has two very important threads running in parallel and need to communicate with each other regularly. The first thread is the UI thread which is responsible for updating UI elements like text views, toasting messages, displaying number pickers, updating sensor value etc. The second thread is the AsyncTask thread that runs in the background and keeps checking the sensor values constantly being updated by the UI thread. This thread invokes the main ExerciseMovement objects track() method which checks if the user is doing that particular exercise correctly.

The AsyncTask thread is also responsible for playing media player sounds with instructions and the feedback to give when the user performs the exercises. The figure above only shows interaction of one exercise object with the UI thread and the background task. The rest of the objects interact in the same way in a linear sequence.

The entire interaction of the UI thread and the background process loops through the number of times based on the number selected by the user before starting the exercise.

6.3 AUDIO INSTRUCTIONS FLOWCHART

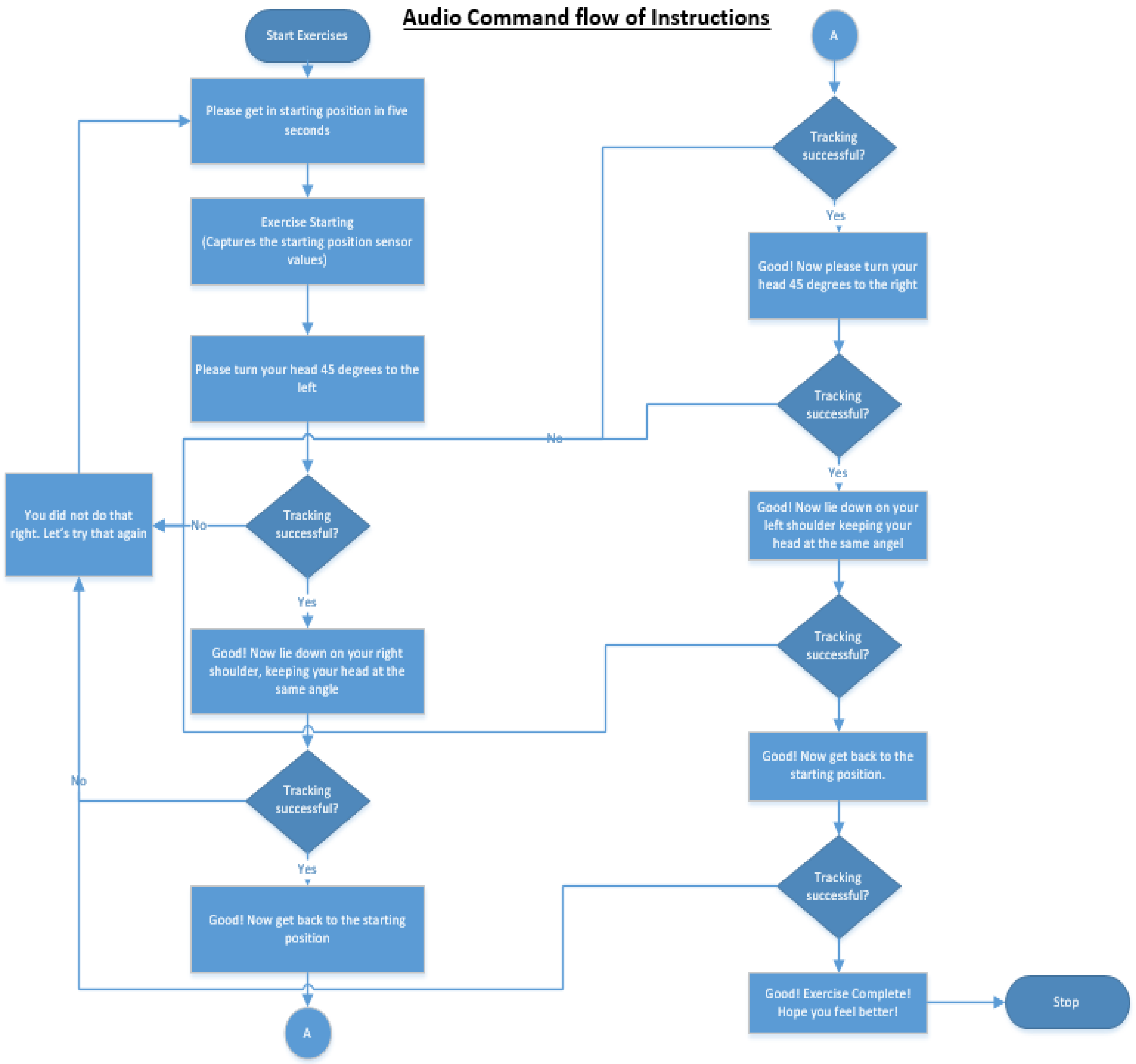

Audio Command flow of Instructions
Start Exercises
Please get in starting position in five seconds
Exercise Starting (Captures the starting position sensor values)
Please turn your head 45 degrees to the left
Tracking successful?
No
You did not do that right. Let's try that again
Yes
Good! Now lie down on your right shoulder, keeping your head at the same angle
Tracking successful?
No
Yes
Good! Now get back to the starting position
A
A
Tracking successful?
Yes
Good! Now please turn your head 45 degrees to the right
Tracking successful?
Yes
Good! Now lie down on your left shoulder keeping your head at the same angel
Tracking successful?
Good! Now get back to the starting position.
Tracking successful?
Good! Exercise Complete! Hope you feel better!
Stop

Figure 16: Audio Instructions flow during the workout

The figure above shows the flow in which the audio commands are played during the exercise. Workout Buddy runs in a loop and upon failure to complete the loop, it restarts it until the user does the exercises as prescribed.

The whole loop shown above runs for n number of times where n is the number selected in the Workout Buddy View fragment. This will complete the set for the user and the workout will be logged with the number of vertigos in the database.

6.4 SENSORS OVERVIEW

The application senses how the device is moving by keeping track of the accelerometer and orientation readings. The values displayed on the screen are along the three imaginary axis drawn over the phone are displayed in the picture below

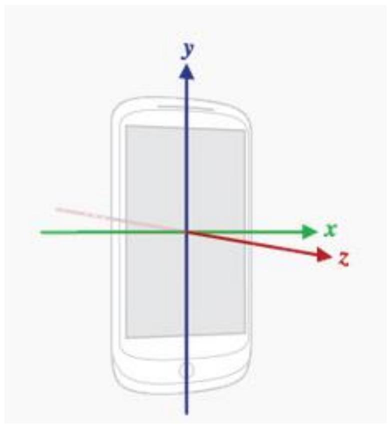


Figure 17: Imaginary axis with respect to the phone.
Image from Google development website [4]

- Android's sensors are controlled by external services and only send events when they choose to.
- An app must register a callback to be notified of a sensor event.
- Each sensor has a related Listener interface that your callback must implement.

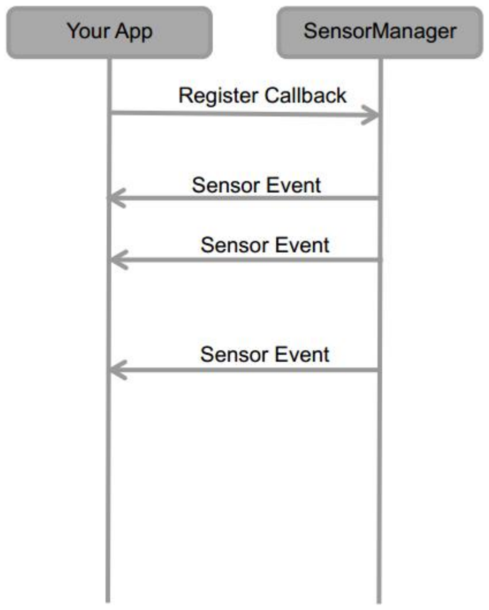


Figure 18: How Sensor Manager works.
Image from Tamer Nadeem from Old Dominion University [4]

ACCELEROMETER SENSOR

"Acceleration is defined as the rate of change of velocity. Since we are dealing with slow movements, we will only consider gravitational force along the axis. For example, in the portrait mode the accelerometer value on the Y – axis will be around 9.8 as that is the value of the gravitational pull. Similarly, an upside down phone in the portrait mode will give a value of -9.8 on the Y – axis." [6].

ORIENTATION SENSOR

"The orientation on the other hand is the angle made by the same imaginary axis with a line pointing towards the magnetic north. The values in all three axis are as follows:

1) x-axis (azimuth) 0/360 degrees is north, 90 east, 180 south, and 270 west
2) y-axis (pitch) 0 flat on its back, -90 standing upright.
3) z-axis (roll) 0 flat on its back, -90 is the screen facing left" [6].

“The orientation sensor derives its data by processing the raw sensor data from the accelerometer and the geomagnetic field sensor. Because of the heavy processing that is involved, the accuracy and precision of the orientation sensor is diminished (specifically, this sensor is only reliable when the roll component is 0). As a result, the orientation sensor was deprecated in Android 2.2 (API level 8). Instead of using raw data from the orientation sensor, we recommend that you use the `getRotationMatrix()` method in conjunction with the `getOrientation()` method to compute orientation values. You can also use the `remapCoordinateSystem()` method to translate the orientation values to your application's frame of reference.”[6].

```
boolean success = SensorManager.getRotationMatrix(R, I, valuesAccelerometer,
valuesMagnetic);

if (success)
       {
       valuesOrientation = new float[3];
       SensorManager.getOrientation(R, valuesOrientation);
       }
```

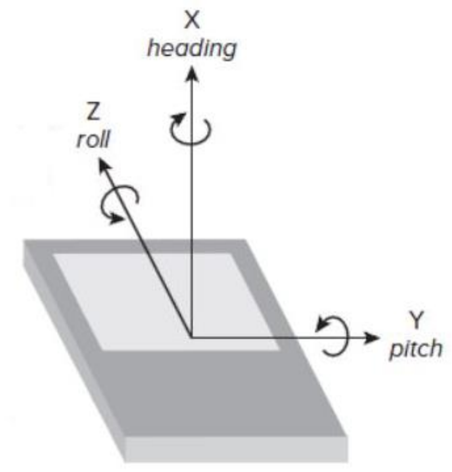


Figure 19: Orientation changes with respect to the axis.
Image from Tamer Nadeem from Old Dominion University [4].

6.5 CALCULATING SHOULDER MOVEMENTS

Shoulder movements are calculated using the value for force of gravity on the phone. The exercise tracks the accelerometer value on the Y-axis to be able check if the exercise was done properly or not. The average values for the Accelerometer along the Y axis in the two shoulder movements are given below:

Right Shoulder: 6 m/(s*s)

Left Shoulder: -6 m/(s*s)

This is slightly less than the gravitational pull to ensure the tilt in the head when the user is lying down on his shoulder.

The track() method inside the Shoulder exercises track the accelerometer readings for 15 seconds and make sure that out of those 15 seconds, you are in the right position for at least 10 seconds.

6.6 CALCULATING ANGLE MOVED BY THE HEAD

The head movements in the exercises are calculated using the orientation. Orientation is derived to get the angle made between the magnetic north and the three axis on the phone as shown in figure 9.

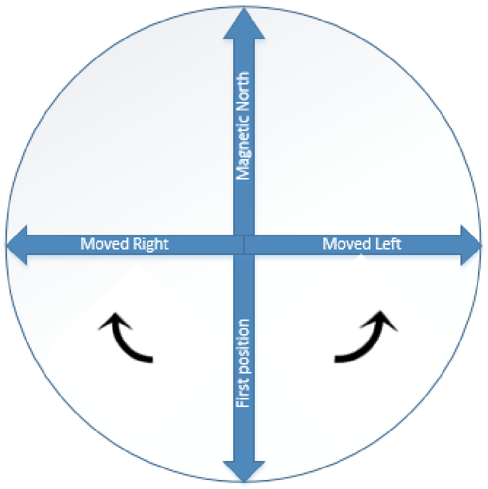


Figure 20: Changes in the rotation angles

The angle is derived and it is negative if it moves to the left with respect to the starting position and it is positive if it moved right with respect to the starting position.

The average movement in the angle for the head movements are shown below:

Left Head: -45 degree radians

Right Head: 45 degree radians

The orientation sensor that was derived from the accelerometer and magnetic field sensor is used to find the angle made by the phone's x axis with the direction of magnetic north.

CODE TO CALCULATE DEGREE MOVED:

First and second and the orientation readings for the starting and ending position of the exercise which the application is tracking.

The method returns the angle moved (negative if moved to left, positive if moved to right).

```
float angleMoved(float first, float second){
            float angle;

            angle = second - first;

            if(Math.abs(angle)>180)
                  {
                  if(angle < 0)
                        angle = (360 - Math.abs(angle));
                  else
                        angle = (360 - Math.abs(angle))*-1;

                  }

            return angle;
}
```

SECTION 7: USAGE

This section describes how to setup the development environment and use the application.

8.1 SETUP

a) Download and setup ADT using instructions on this page.

b) Eclipse is included in the bundle and can be used as an IDE for development.

c) Download the application code from here. Unzip the code and import it inside the eclipse workspace. The code is setup for development and modifications.

d) Do not launch the application in the emulator as this will not give changing sensor values. Always use an Android phone that is connected with the USB and make sure your computer has the USB driver installed.

e) When you launch the application, it will run on your Android phone. You can wear it over your head with a band with the USB connection still active to be able to see the LogCat.

f) When you move your head, different audio commands are played based on your head movement.

5.2 CODE WALK-THROUGH

5.2.1 CODE ORGANIZATION:

Everything related to the Workout Buddy module is kept in a separate package called "com.example.msapp2.exercises" in the source folder. The .java files contained in this package, their description and the respective layout files are given below:

1) **Class:** BackToStart.java
   Description: This class contains code that tracks if the user has come back in the upright position again. It extends the abstract class ExerciseMovement.java. It checks for five seconds the user's position and sets the boolean array accordingly.
   Layout File: (No layout file. This object works in the background)

2) **Class:** ExerciseMovement.java
   Description: This is an abstract class that is the super class of all the exercise movements (LeftHead.java, RightHead.java, LeftShoulder.java, RightShoulder.java, BackToStart.java). It also implements some of the functionalities like measuring the angle moved by the head, measuring if the sensor values are in the range of some pre-defined values, etc.
   Layout File: (No layout file. This object works in the background)

3) **Class:** LeftHead.java
   Description: This class contains code that tracks if the user has turned his head to the left with respect to the orientation of the starting position. It extends the abstract class ExerciseMovement.java. It checks for five seconds the user's orientation change and sets the boolean array accordingly.
   Layout File: (No layout file. This object works in the background)

4) **Class:** LeftShoulder.java
   Description: This class contains code that tracks if the user is lying flat on his left shoulder. It extends the abstract class ExerciseMovement.java. It checks for thirty seconds the user's position and sets the boolean array accordingly. It also plays the media player sound in the middle of tracking to tell the user that he might experience dizziness.
   Layout File: (No layout file. This object works in the background)

5) **Class:** NumberOfVertigos.java
   Description: This is the class that implements the pop up dialog fragment after the exercise is completed.
   Layout: vertigo_count.xml

6) **Class:** RightShoulder.java
   Description: This class contains code that tracks if the user is lying flat on his right shoulder. It extends the abstract class ExerciseMovement.java. It checks for thirty seconds the user's position and sets the boolean array accordingly. It also plays the media player sound in the middle of tracking to tell the user that he might experience dizziness.
   Layout File: (No layout file. This object works in the background)

7) **Class:** RightHead.java
   Description: This class contains code that tracks if the user has turned his head to the right with respect to the orientation of the starting position. It extends the abstract class ExerciseMovement.java. It checks for five seconds the user's orientation change and sets the boolean array accordingly.
   Layout File: (No layout file. This object works in the background)

8) **Class:** SensorReadings.java
   Description: This class serves as an abstract data type that stores float array of accelerometer readings and orientation readings. This data type is used to store the current sensor values and it is used in methods to derive the orientation values.
   Layout File: (No layout file. This is an abstract data type for sensor values.

5.2.2 CODE SNIPPETS

Tracking the exercise:

The example below shows the method for tracking the exercise for four seconds and setting the timearray[] values appropriately. At the end of the method, decodeTimeArray() is called to find out if the exercise was successfully performed. The user can track the exercise for a particular amount of time and set the isSuccessful Boolean value at the end.

```
@Override
public void track(){

android.os.Process.setThreadPriority(android.os.Process.THREAD_PRIORITY_BAC
KGROUND);

try{Thread.sleep(500);}catch(InterruptedException e){e.printStackTrace();}

for(int i=0; i<4; i++){
      try{Thread.sleep(1000);}catch(InterruptedException
      e){e.printStackTrace();}

      SensorReadings sr = WorkoutBuddy.s;

      if(sr != null){
            if(sensorInRange(sr))
                  timeArray[i] = true;
                  else
                  timeArray[i] = false;
            }

      }

      isSuccessful = decodeTimeArray(timeArray, 2);

}
```

FIND IF SENSOR IS IN RANGE

In this example we are checking if the accelerometer sensor is in the range of 8.5 to 10. 5 along the Y - axis. Similarly, orientation can also be checked to see if it is in a particular range. We can edit this method in the respective exercise's concrete class to change the valid range for sensors for the exercise to be successful.

```
@Override
protected boolean sensorInRange(SensorReadings sense)
{
if(sense != null)
{
      if( sense.accelerometer[1] < 10.5 && sense.accelerometer[1] > 8.5)
            return true;
            else
            return false;
}
      else
      return false;

}
```

MANAGING DIFFERENT INPUTS IN ONPROGRESS() METHOD

Using the try – catch block to handle integer and string updates in the onProgress() method. onProgress() is used to update the cycle number when integer type argument is passed to it and to toast exercise progress message if a string type argument is passed to it. If you want to handle more types of inputs, you can create your own AsyncTask class. [3]

```
@Override
protected void onProgressUpdate(String... message) {
super.onProgressUpdate(message);
//If message is an integer type, update the text to indicate cycle number
      try{
            Integer i = Integer.parseInt(message[0]);
            cycleNumber.setText(String.valueOf(i));
            }

//If message is string type, toast it
      catch(NumberFormatException e){
            String msg = ((String[])message)[0];
            Toast toast = Toast.makeText(getApplicationContext() , msg,
            Toast.LENGTH_SHORT);
            toast.show();
            }


}
```

SECTION 7: TESTING

Testing was done on an actual phone and not on an emulator as emulator would not give changes in the sensor readings. Unit testing was done on each class and each method and considering it as one unit. Each screen was also considered as sub component and was tested as a unit. Integration testing was performed to make sure that all the components were working fine together.

The device used for testing was an LG Nexus 5 that runs on Android 4.4.2 Kit Kat operating system. System testing was performed to cover functional and non-functional requirements. Performance testing was also done to make sure the application did not lag or hang in the middle of using it. The tests made sure that the application was responsive at all times and the user could get into and out of the application without it hanging.

SECTION 8: ISSUES

This section describes main bugs that I found and fixed.

1. Nothing related to the UI should be done in the doInBackground() method of the activity as this will cause the application to crash. Instead, the publishProgress() method should be used to handle UI elements. All UI updates were moved out of background process and a way of handling both integer and string type parameters in the publishProgress() method was created using try – catch blocks.
2. The sensor readings even in the normal mode change too rapidly to be reliable to track exercises. To solve this a low pass filter function defined in the WorkoutBuddy class was created and the sensor values were passed through this function before being used. This made the sensor values smooth and manageable.
3. The sensor readings are also measured in the background but Android framework takes care of it. Make sure the onSensorChanged() method is not doing any heavy UI updates or heavy calculations as this will block the UI thread and the application may crash.

Media Player sounds were getting played over each other. One good way of handling this was to put all the media player objects in a HashMap and make sure to stop any media player that was already playing by iterating through the HashMap and stopping any media player that was playing sound. This also makes it easier to give each media player a meaningful name and it could be retrieved easily.

SECTION 9: FUTURE ENHANCEMENTS

This chapter lists the possible future enhancements for this application.

AUDIO FEEDBACK FROM USER FOR CHECKING VERTIGO

The user can give audio feedback in the middle of exercising to mention if he is feeling dizzy during the exercises.

MINING DATA IN DATABASE

The application can run in background from time to time and check the incidents and workouts in the database to automatically advice about workouts if more incidents are noticed with low number of workouts

VISUALIZATION OF THE HISTORY DATA

Charts can graphs can be populating showing statistics of the how the user is doing. The user can also set goals for himself and see how he is doing against his goals.

SETTING YOUR OWN PRECISION

The user can set his own precision values and calibrate it according to his height. This will make the detection of exercises more accurate.

SHARING EXERCISE DATA WITH FRIENDS

Users can share their data with their friends to be able to compare their progress with others and be motivated to perform exercises.

SECTION 10: REFERENCES

1. Bhavana's documentation for the Mobile Assistant for Physical Therapy Exercises for Benign Positional Paroxysmal Vertigo application
2. Applying low pass filters on Android sensors: http://www.raweng.com/blog/2013/05/28/applying-low-pass-filter-to-android-sensors-readings/
3. Managing different values in publishProgress() method of AsyncTask: http://stackoverflow.com/questions/23251315/cam-publishprogress-be-overloaded-in-asynctask/23293344#23293344
4. Android Sensors: http://www.cs.odu.edu/~cs495/materials/Lec-05_Android-Sensors.pdf
5. Used the code here to generate compass which made it easier to visualize orientation readings. http://sunil-android.blogspot.com/2013/02/create-our-android-compass.html
6. Sensor Overview http://developer.android.com/guide/topics/sensors/sensors_overview.html
7. Android Intents http://www.itcsolutions.eu/2011/08/31/android-tutorial-how-to-create-and-display-a-new-form-window-or-activity/
8. AsyncTask http://developer.android.com/reference/android/os/AsyncTask.html