Utilizing a Mobile Device to Implement a Dual-Recording Eye Exam

By

Joseph M. Costello

Submitted in Partial Fulfillment of the Requirements

for the Degree of

Master

of

Computing and Information Systems

YOUNGSTOWN STATE UNIVERSITY

DECEMBER, 2020

Utilizing a Mobile Device to Implement a Dual-Recording Eye Exam

Joseph M. Costello

I hereby release this thesis to the public. I understand that this thesis will be made available from the OhioLink ETD Center and the Maag Library Circulation Desk for public access. I also authorize the University or other individuals to make copies of this thesis as needed for scholarly research.

Signature:

_____

Joseph M. Costello, Student Date

Approvals:

_____

Dr. Salvatore A. Sanders, Dean of Graduate Studies Date

_____

Dr. Abdu Arslanyilmaz, Thesis Advisor Date

_____

Dr. Feng Yu, Committee Member Date

_____

Dr. John Sullins, Committee Member Date

ABSTRACT

Detecting a concussion early and accurately is critical to the recovery of the person sustaining the concussion. Being able to more accurately detect whether or not a person has been concussed can lead to people seeking treatments more often and have fewer permanent issues due to concussion. Being able to accurately diagnose a concussion via automated means first requires accurate and reliable recordings of a pupillary response eye examination. It is shown that utilizing a mobile device to administer an eye examination while simultaneously recording the pupillary response to that exam is accurate and reliable enough to show the difference between a person with a known pupillary response affecting health issue and a person with a healthy pupillary response.

Table of Contents

I. Introduction

A concussion is a type of injury to the brain caused by the brain bouncing or twisting in the skull [1]. Concussions can happen by a variety of ways including contact sports or activities that cause the body to change velocity quickly [2]. The momentum of the brain causes it to bump or twist in the skull during these quick velocity changes. Some concussions cause you to lose consciousness and some do not, but all are serious as they are damage to your brain. Concussions are classified as mild traumatic brain injuries and are akin to a small bruise to the brain [1]. These cannot be directly detected by using computerized tomography (CT) scans or Magnetic Resonance Imaging (MRI) scans, but these are often used to rule out more serious brain injury such as bleeding or structural injuries to the brain.

The current way concussions are detected is through neurological and cognitive testing [3], if that indicates a concern, then the CT and/or MRI scans are performed to determine if there is a serious injury to the brain itself such as bleeding and or swelling. [4]. While the scans help show more serious injury the concussion diagnosis is not dependent on them[5]. While these scans can tell us if a more serious injury has definitely occurred, they are not the easiest to administer on the spot and are expensive. It is imperative to detect a concussion immediately on the spot when it occurs during a game so as to prevent further damage to the brain [6].

A medical professional as part of the neurological and cognitive testing may administer an oculomotor exam, sometimes termed the follow-my-finger exam after the instructions and assess by watching the eyes looking for smooth pursuit of the finger or saccade eye movement. Smooth pursuit is the eye moving fluidly while tracking the finger. Saccade movement is the eye moving past the finger and then a corrective jump back movement that produces a flutter movement in the eye. Because the test is subjectively assessed by the medical professional, it is only used as one of many indicators of a possible concussion and not definitive.

The Ohio High School Athletic Association (OHSAA) has enacted regulations that state anyone removed from play due to a suspected concussion is unable to return to play the same day, even if determined by a medical professional afterwards to have not been concussed.[7] This regulation specifies behavioral conditions to determine if an athlete has been concussed such as appearing stunned or confused, forgetting plays, unsure of scores or opponents. This is highly subjective and is up to the person administering the test whether or not to classify an injury a concussion injury. This can possibly lead to bad decisions on the part of the person administering the exam, both for ruling a concussion when there isn't and vice versa. For example, as a result of misdiagnosis, players could sustain further injuries to their brains upon allowing them to continue to play even though they had undiagnosed concussions [7]. Having a more definitive testing method that can be used immediately and provides an objective and accurate assessment of the players' likelihood of being concussed is of great value.

There has been a movement towards getting more conveniently administered, less subjective, and more accurate concussion detection than a series of questions to help the injured better recover if they have a concussion and to eliminate doubt in the diagnosis of the concussion [8].

A smartphone app was being developed that used light to measure pupillary dilation & response was in development but didn't make its funding goals and appeared to be abandoned in 2017[9]. It was attempting to get development funding through IndieGoGo but fell short of its goal. According to the link on the fundraiser page, the website brightlamp.org was used for this project. The website currently has a mobile app that uses replaces a penlight for eye testing but does not look like it incorporated the concussion detection aspect that the fundraiser was mentioning [10].

In 2019 a new device called EyeBOX [11] was approved by the FDA for concussion detection that doesn't require a baseline test to establish non-concussed levels, but the device is not easily transported on the field, requires level placement on a raised platform to place the head in the holster, and external power to work.

Figure 1 - Oculogica's EyeBOX[11]

Aside from the cumbersome nature of the device for outdoor sports diagnoses, the device boasts a 5-minute test time with no need for prior baseline exams to diagnose a concussion.

II. Problem

How can a system be implemented to easily and effectively diagnose concussions that can be administered almost immediately at the site of the injury? This is important because the sooner a concussion is definitively diagnosed, steps can be taken to minimize the sustained injury or to prevent exacerbating the injury. Due to the delay in symptom onset, simple observable behavioral assessments might let undiagnosed concussions slip past detections[12]. If an athlete is allowed to continue to play with an undiagnosed concussion can lead to worse complications including fatal brain swelling[12]. Being able to flag a person accurately for a possible concussion will reduce undiagnosed concussions which will in turn reduce the occurrence of fatal complications due to undiagnosed concussions.

Beyond the benefit to patient health there are also other benefits that may be realized by this technology. In sports the team will benefit by being able to heal injured players and reduce exacerbating the injury. This will help the teams avoid liability and return to full performance quicker. The teams will be able to keep a player in the game if they are able to determine that they did not sustain a traumatic brain injury. This will help the team perform better right now and help with the progress and flow of the game. A mobile diagnostic tool that is easy to use, affordable or free to obtain, and fast to administer with a quick diagnosis would be able to be used by all recreational and professional players at all ages.

The medical sector will also be able to benefit. This will help misdiagnosis and incorrectly diagnosed concussions to avoid liability for potential patient injury and death.

III. Study Objectives

The purpose of this study is to exhibit development and implementation of a mobile application that provides an easy-to-use and effective eye exam tool to administer the exam at the point of injury without having to take the injured person to another location for an exam. Specifically, this study will:

- Showcase a mobile or tablet application that records eye movements and is able to be used with post processing software to determine with a consistent and accurate degree of certainty that a concussion has been sustained.
- Demonstrate the application for cross-platform development (iOS and android) and implementation.
- Exhibit computer vision analysis of the video on-device in determining concussion
- Showcase a technique of recording both the Exam UI & the Eye movements at the same time. It may be beneficial to determine movement delay and could possibly provide a tangible determination metric.

IV. App Design

A. Interface Design

The interface was designed as a dot moving to simulate a follow-my-finger exam as sometimes administered by a medical professional to determine oculomotor impairment. The dot will start and stop from the center of the screen and traverse the screen in the shape of an 'H' like the clinical exams [13].

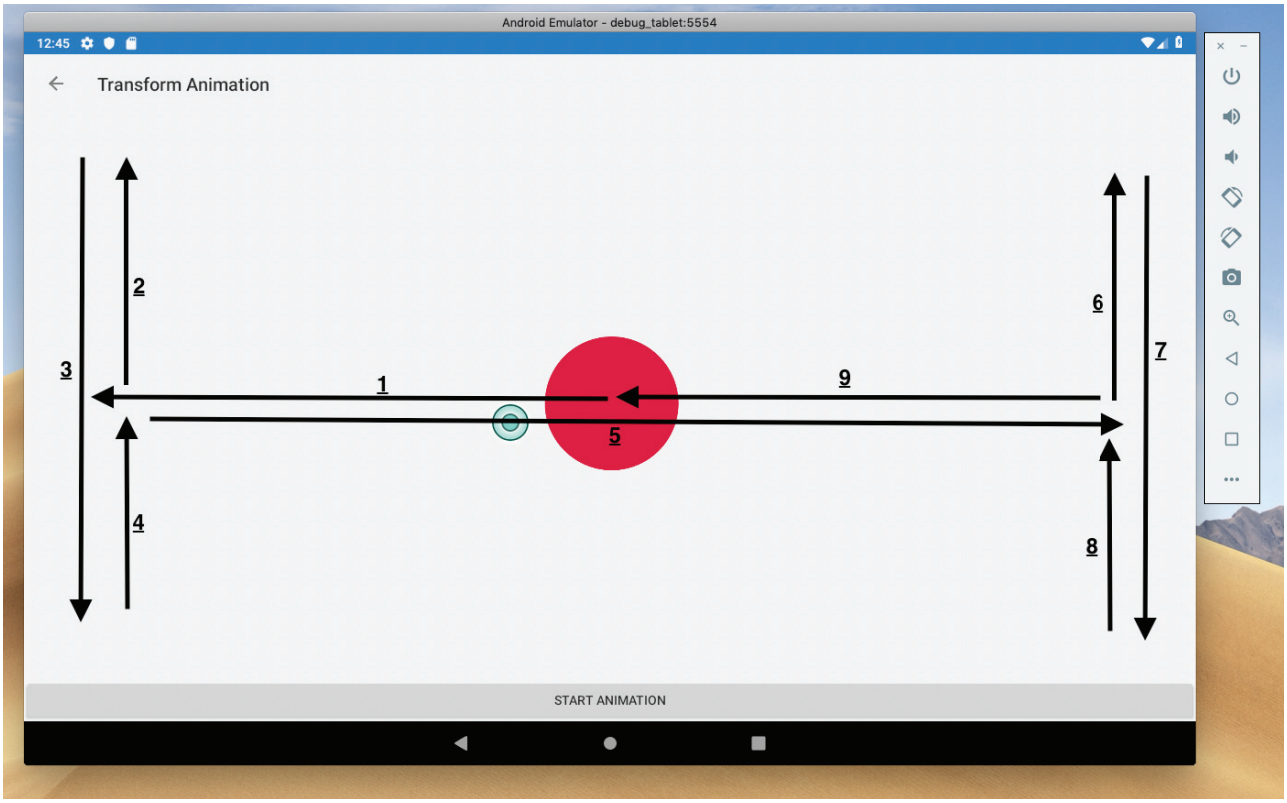


Figure 2 - Showing the path of the dot with the movement path segments labelled 1-9

Oculomotor latencies, the delay of the eye to begin movement from a stopped position, has shown to be indicative of other neurological issues [14] and may help in the accuracy of detection for concussions as well. To incorporate latency measurement into the exam the dot movement was modified to include a configurable pause at each direction change of the dot. The movements commands were already broken up into straight line segments and each command takes a delay modifier in milliseconds. A new setting value was added to the setting screens for transition delay and incorporated into the delay modifier for the movement command.

We've mapped out the exam dot movement as such to match the 'H 'shape used in clinical examinations of the CN-III cranial nerve to include the lateral up, lateral down, medial down, medial up movements [13]12/18/20 2:29:00 PM.

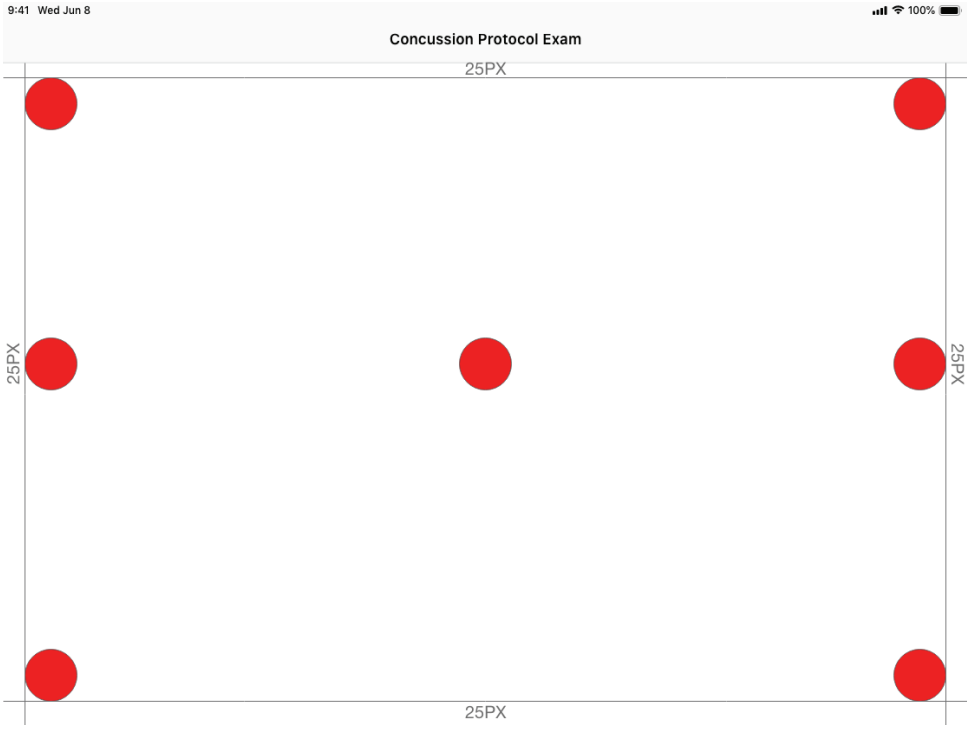


Figure 3 - Dot at each direction transition point

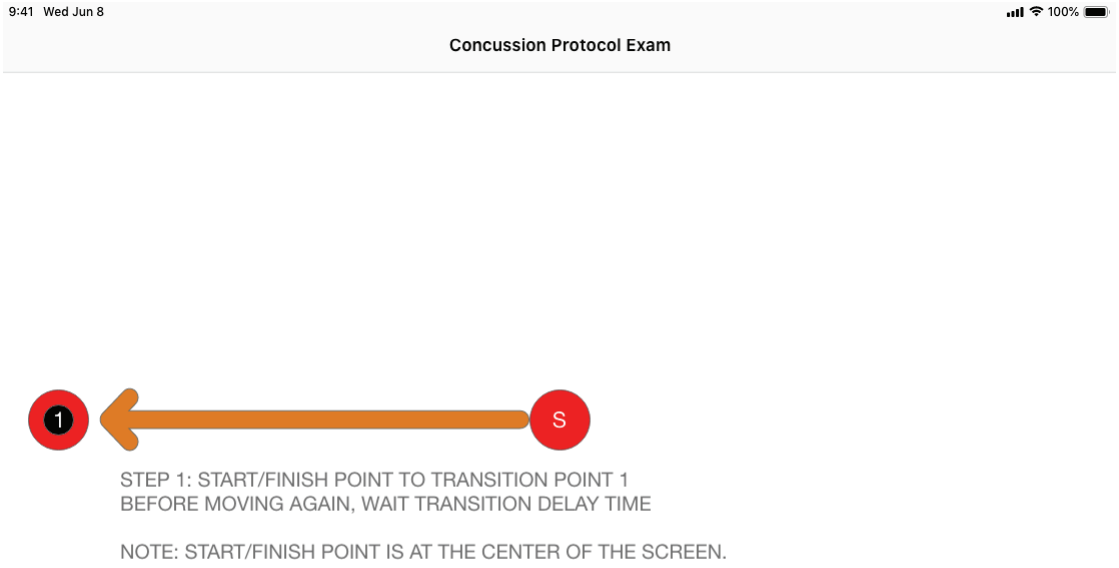


Figure 4 - Step 1 dot moves from start (S) point to transition point (TP) 1

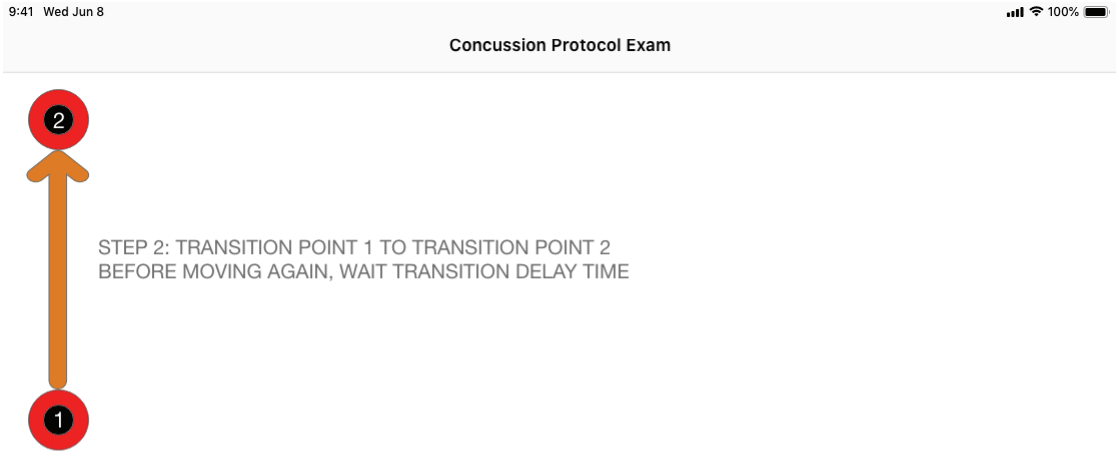


Figure 5 - Step 2 Dot moves from TP1 to TP2

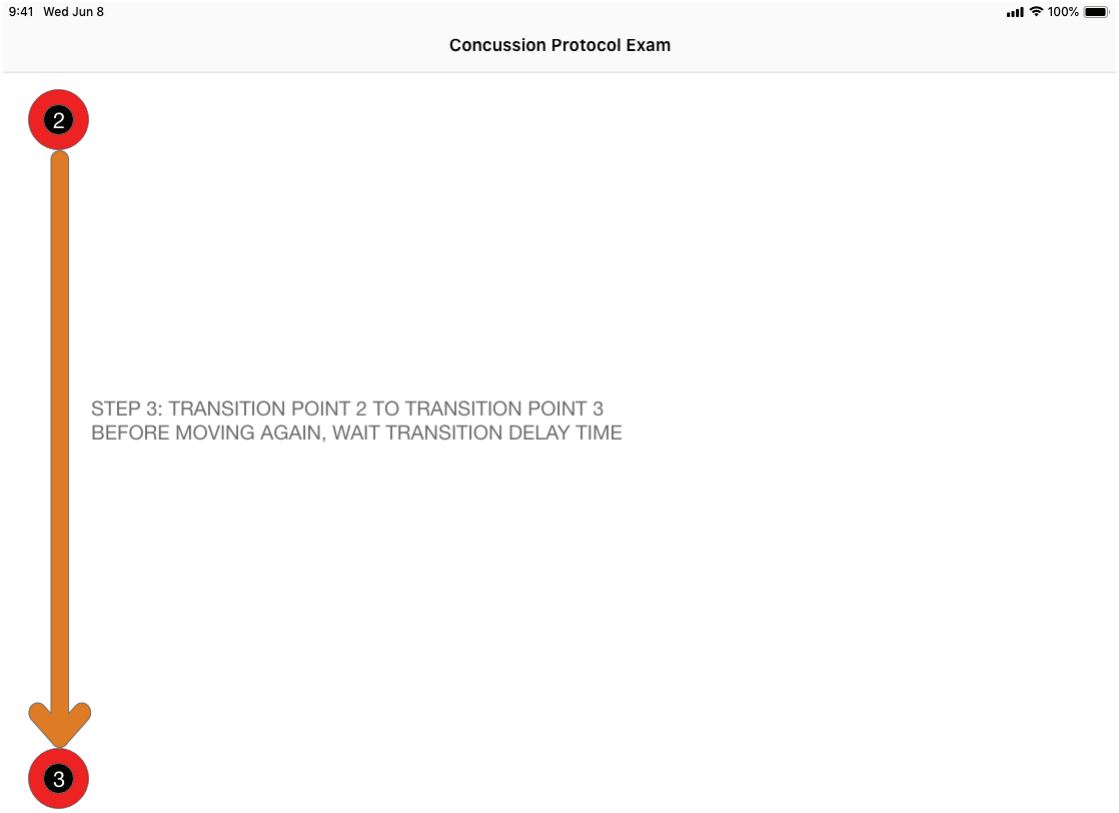


Figure 6 - Step 3 dot moves from TP2 to TP3

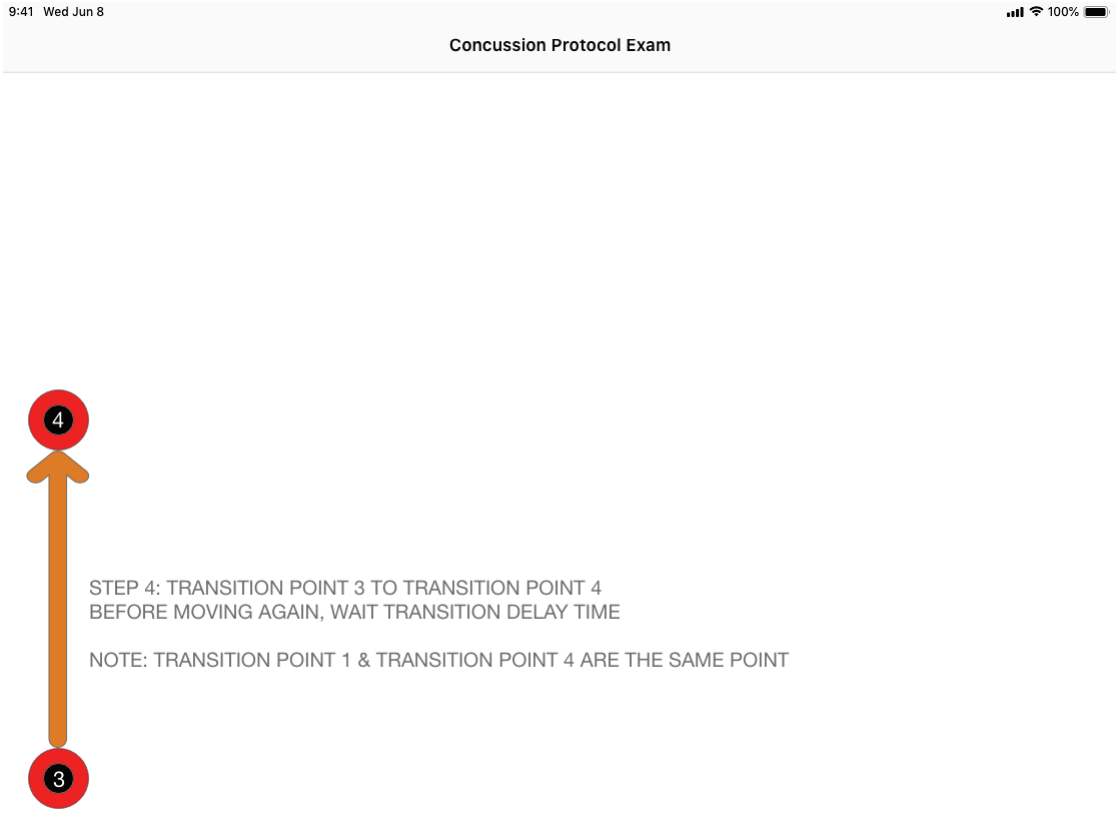


Figure 7 - Step 4 dot moves from TP3 to TP4

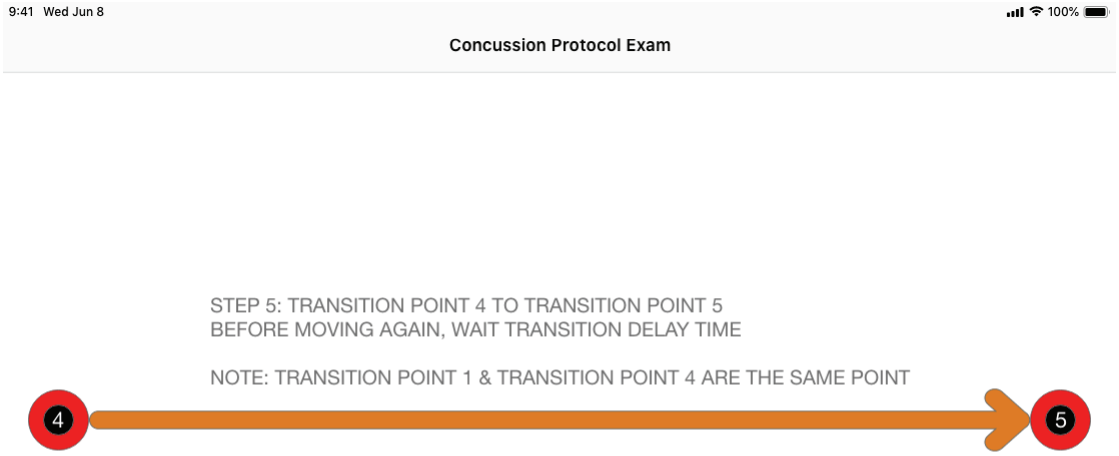


Figure 8 - Step 5 dot moves from TP4 to TP5

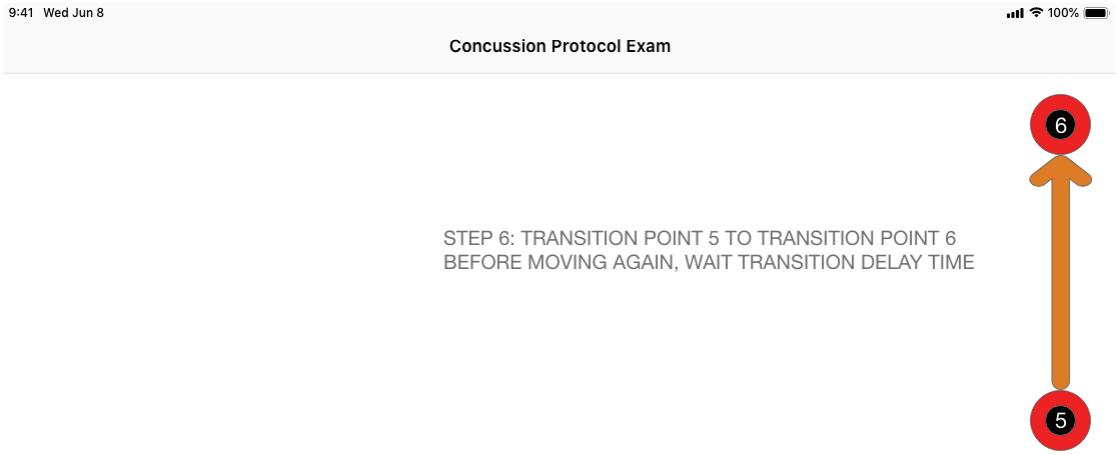


Figure 9 - Step 6 dot moves from TP5 to TP6

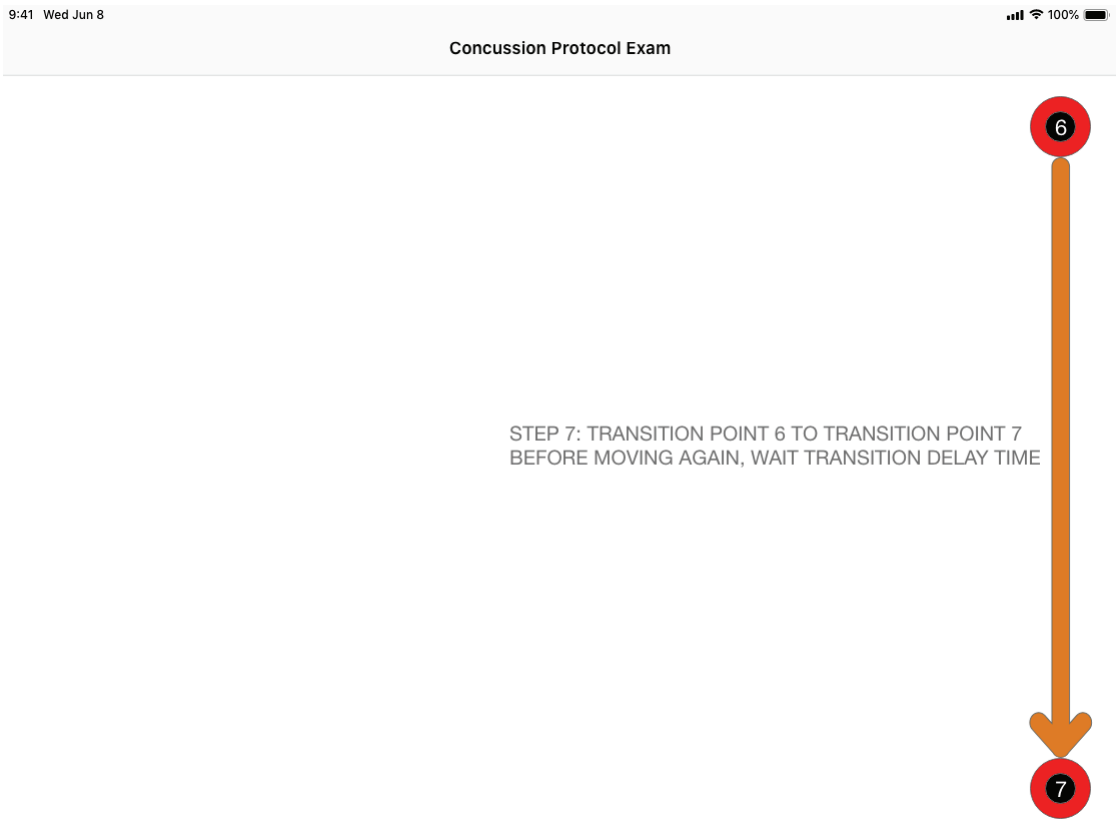


Figure 10 - Step 7 dot moves from TP6 to TP7

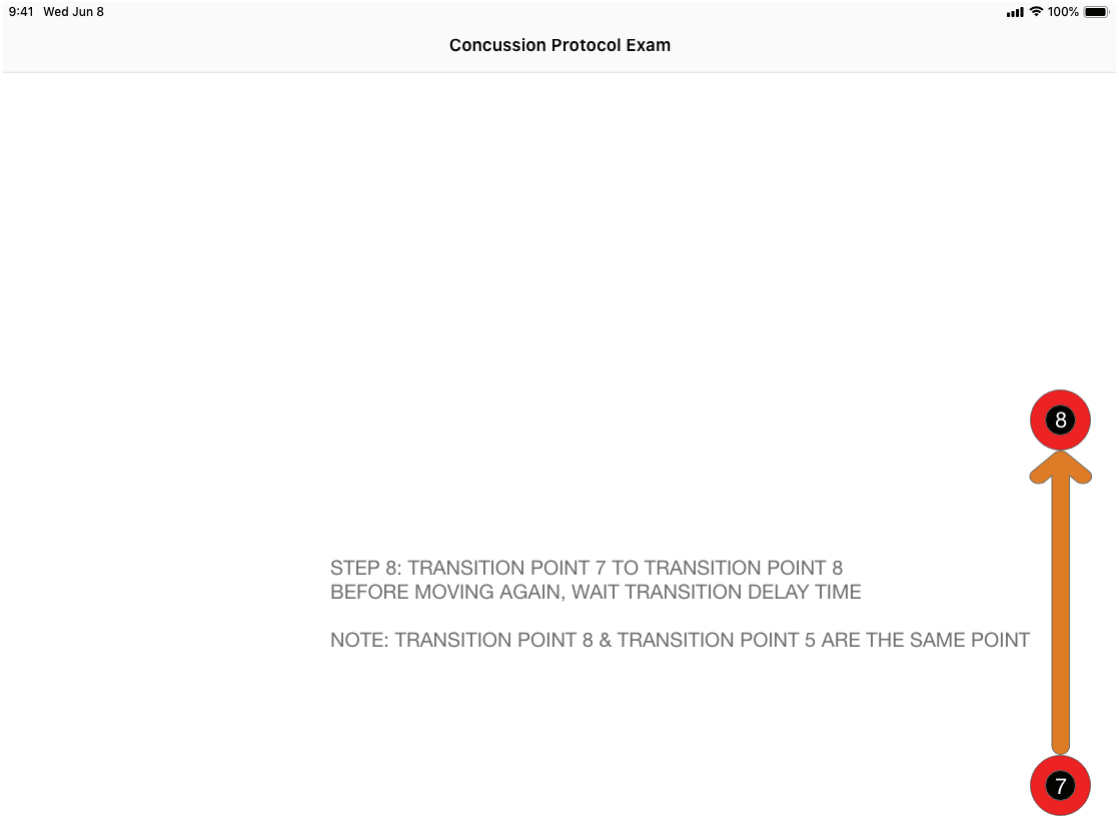


Figure 11 - Step 8 dot moves from TP7 to TP8

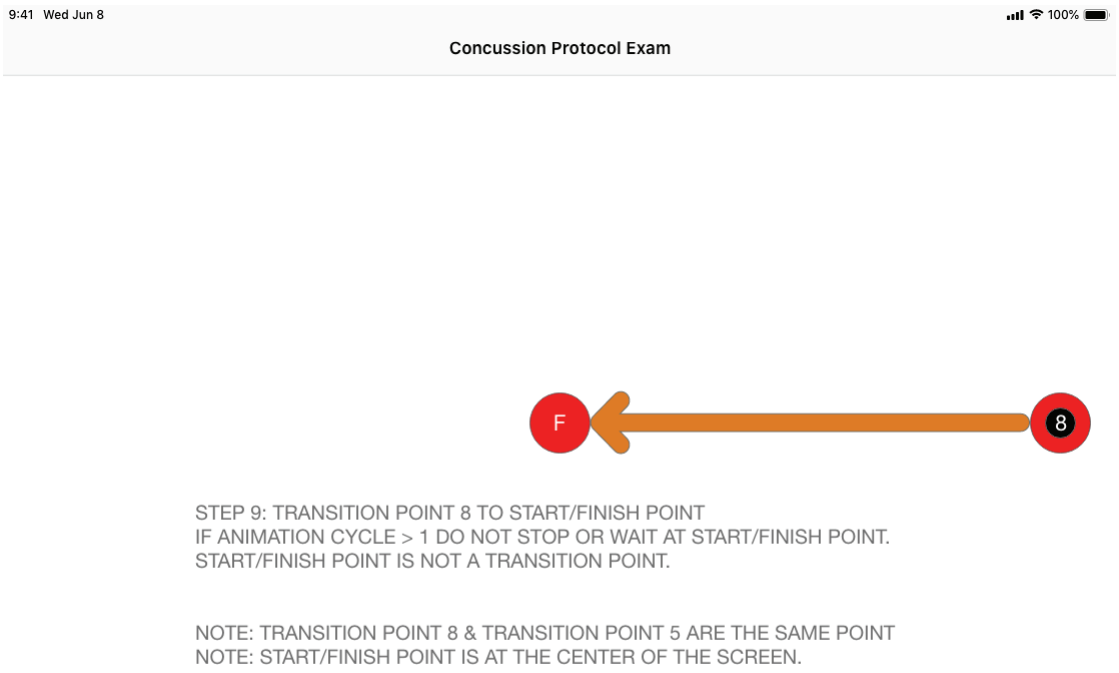


Figure 12 - Step 9 dot moves from TP8 to finish (F) point

The dot movement was accomplished by utilizing the animated methods of the view wrapped in its own function animate().

```
   animate(){

     Animated.sequence([
       Animated.loop(
         Animated.sequence([
           Animated.timing(this.translateValue,
             { toValue: { x: -(Dimensions.get('window').width/2-this.state.userData.size/2), y: 0 },
            duration: ((Dimensions.get('window').width/2-
this.state.userData.size/2)/this.state.userData.speed)*10,
              easing: Easing.linear }),
          // ------------------------------------------------------------------------


           Animated.timing(this.translateValue,
            { toValue: { x: -(Dimensions.get('window').width/2-this.state.userData.size/2), y: -
(Dimensions.get('window').height/2-this.state.userData.size/2) },
            duration: ((Dimensions.get('window').height/2-
this.state.userData.size/2)/this.state.userData.speed)*10,
             delay:this.state.userData.delay*1000,
              easing: Easing.linear }),

             //------------------------------------------------------------------
```

```
          Animated.timing(this.translateValue,
           { toValue: { x: -(Dimensions.get('window').width/2-this.state.userData.size/2), y: 0 },
            duration: ((Dimensions.get('window').height/2-
this.state.userData.size/2)/this.state.userData.speed)*10,
            delay:this.state.userData.delay*1000,
            easing: Easing.linear }),

         //    ----------------------------------------------------------------------

           Animated.timing(this.translateValue,
             { toValue: { x: -(Dimensions.get('window').width/2-this.state.userData.size/2), y:
(Dimensions.get('window').height/2-this.state.userData.size/2) },
              duration: ((Dimensions.get('window').height/2-
this.state.userData.size/2)/this.state.userData.speed)*10,
               easing: Easing.linear }),
         //----------------------------------------------------------------------

            Animated.timing(this.translateValue,
              { toValue: { x: -(Dimensions.get('window').width/2-this.state.userData.size/2), y: 0 },
               duration: ((Dimensions.get('window').height/2-
this.state.userData.size/2)/this.state.userData.speed)*10,
               delay:this.state.userData.delay*1000,
                easing: Easing.linear }),

         //----------------------------------------------------------------------

             Animated.timing(this.translateValue,
               { toValue: { x: 0, y: 0 },
               duration: ((Dimensions.get('window').width/2-
this.state.userData.size/2)/this.state.userData.speed)*10,
                delay:this.state.userData.delay*1000,
                 easing: Easing.linear }),

         //----------------------------------------------------------------------

             Animated.timing(this.translateValue,
               { toValue: { x: (Dimensions.get('window').width/2-this.state.userData.size/2), y: 0 },
                duration: ((Dimensions.get('window').width/2-
this.state.userData.size/2)/this.state.userData.speed)*10,
                 easing: Easing.linear }),

          //----------------------------------------------------------------------

              Animated.timing(this.translateValue,
                { toValue: { x: (Dimensions.get('window').width/2-this.state.userData.size/2), y: -
(Dimensions.get('window').height/2-this.state.userData.size/2) },
                 duration: ((Dimensions.get('window').height/2-
this.state.userData.size/2)/this.state.userData.speed)*10,
                  delay:this.state.userData.delay*1000,
                   easing: Easing.linear }),
         //    ----------------------------------------------------------------------

              Animated.timing(this.translateValue,
                { toValue: { x: (Dimensions.get('window').width/2-this.state.userData.size/2), y: 0 },
```

```
                duration: ((Dimensions.get('window').height/2-
this.state.userData.size/2)/this.state.userData.speed)*10,
                delay:this.state.userData.delay*1000,
                 easing: Easing.linear }),


          //-------------------------------------------------------------------------

                Animated.timing(this.translateValue,
                  { toValue: { x: (Dimensions.get('window').width/2-this.state.userData.size/2), y:
(Dimensions.get('window').height/2-this.state.userData.size/2)},
                   duration: ((Dimensions.get('window').height/2-
this.state.userData.size/2)/this.state.userData.speed)*10,
                    easing: Easing.linear }),

          //-------------------------------------------------------------------------

                Animated.timing(this.translateValue,
                  { toValue: { x: (Dimensions.get('window').width/2-this.state.userData.size/2), y: 0 },
                   duration: ((Dimensions.get('window').height/2-
this.state.userData.size/2)/this.state.userData.speed)*10,
                    delay:this.state.userData.delay*1000,
                     easing: Easing.linear }),

          //-------------------------------------------------------------------------

                Animated.timing(this.translateValue,
                  { toValue: { x: 0, y: 0 },
                   duration: ((Dimensions.get('window').width/2-
this.state.userData.size/2)/this.state.userData.speed)*10,
                    delay:this.state.userData.delay*1000,
                     easing: Easing.linear }),
        ]),
      { iterations: this.state.userData.cycle })
    ]).start(console.log('animation finish'))
  }
```

In anticipation of future studies, the post processing team requested to have the ability of adjusting certain criteria to see if it will help with the accuracy of the exam, we decided to add in an adjustable settings page to the app which the person administering the exam can adjust. This will facilitate studying if different settings effect the reliability or accuracy of the exam analysis.

We have identified the following settings as possibly beneficial settings to adjust

- Size – Adjust between 3 preset sizes for the dot during the exam
- Color – Adjust between Red/Black/Blue to be able to handle possible color blindness
- Speed – Adjust the speed that the dot in the exam takes.
- Transition Delay – Adjust the delay of the dot at transition points.
- Path Cycling – Cycle through the ‘H’ pattern multiple times

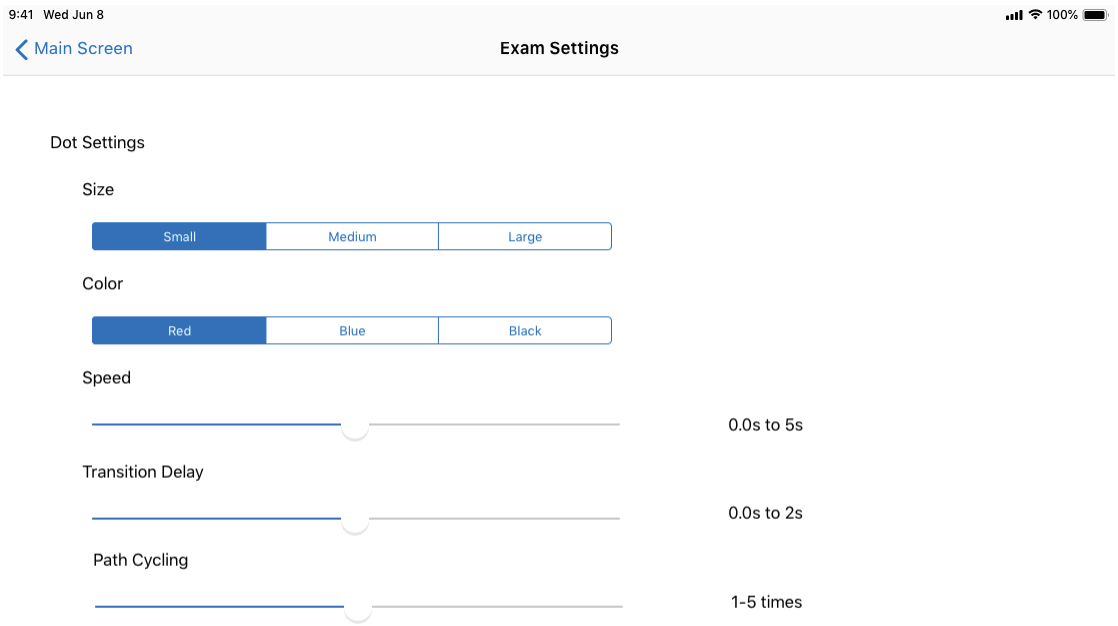


Figure 13 - Settings Page

The settings are all set on the settings page and saved in an internal SQLite DB created by the application. The Size & Color settings are incorporated into the declaration of the dot itself using the backgroundColor and height/width properties.

```
<View style={{ flex: 1 }}>
    <Animated.View style={{
      backgroundColor:this.state.userData.color,
      width:this.state.userData.size,
      height:this.state.userData.size,
      borderRadius:50,
      top:Dimensions.get('window').height/2-this.state.userData.size/2,
      left:Dimensions.get('window').width/2-this.state.userData.size/2,
      position: "absolute",
      transform: translateTransform
      }} />
    </View>
```

The speed and transition delay are incorporated by modifying the values of the duration and delay properties of the dot animation by the values from the app settings.

```
Animated.timing(this.translateValue,
  { toValue: { x: (Dimensions.get('window').width/2-this.state.userData.size/2), y: 0 },
  duration: ((Dimensions.get('window').height/2-this.state.userData.size/2)/this.state.userData.speed)*10,
  delay:this.state.userData.delay*1000,
   easing: Easing.linear }),
```

The path cycling setting is accomplished by wrapping all of the animation commands within a loop with the iterations set to the path cycling value.

```
Animated.loop(
  Animated.sequence([ ···
  ]),
{ iterations: this.state.userData.cycle })
```

B. Mobile Programming Frameworks

Mobile Applications can be written natively in the development environment for each device, or by using a cross-platform framework which helps abstract out some of the development. Natively iOS uses XCode IDE and Objective-C or Swift as the programming language. Android uses Eclipse as the IDE and Java as the programming Language. Using a cross-platform Framework allows at least some of the project to be developed for both iOS and Android at the same time, but some functionality is still distinct between the two operating systems & needs to be written separately for both systems. But you can still consolidate your code into one project and share a similar programming language.

There were two frameworks that were looked at as possible candidates:

- Xamarin
- React Native

Xamarin, developed by Microsoft, uses Visual Studio & C# while React Native was developed by Facebook and uses JavaScript as its programming language & Node.js to compile. Due to using JavaScript, React Native does not have a dedicated IDE. Ultimately, you can use any plain text editor to edit your project, however using an IDE that provides Autocomplete and syntax highlight is beneficial to the organization of your code and speed of development.

Both frameworks provide similar functionality, but ultimately, the ability of React Native to make changes to the JavaScript code without recompiling the native application was determined to be a faster development environment for what we were trying to accomplish.

C. Mobile Devices

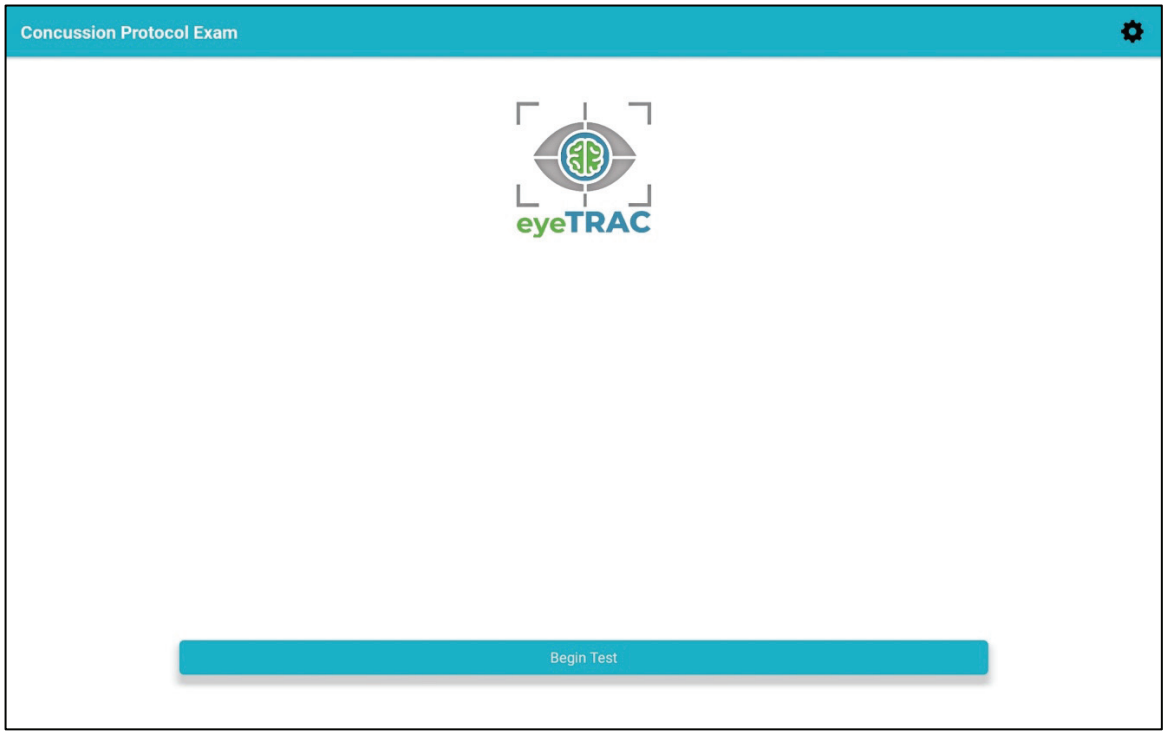
We've decided on a Samsung Galaxy Tab A 10.1 for development and testing. The device has a 10.1" screen and a 2MP front facing camera for eye capture. Due to this we will be focusing on Android functionality. That limits the range of possible devices that this can run on to android devices. The exam should work on any android device that has a front-facing camera.

V. App Screens

The application has a minimal design to streamline app development time and to minimize any compatibility issues when/if it is moved towards full cross platform capabilities with iOS. It consists of 4 screens.

- The Home Screen
- The Settings Screen
- The Front Camera Viewport Aiming Screen
- The Exam Screen

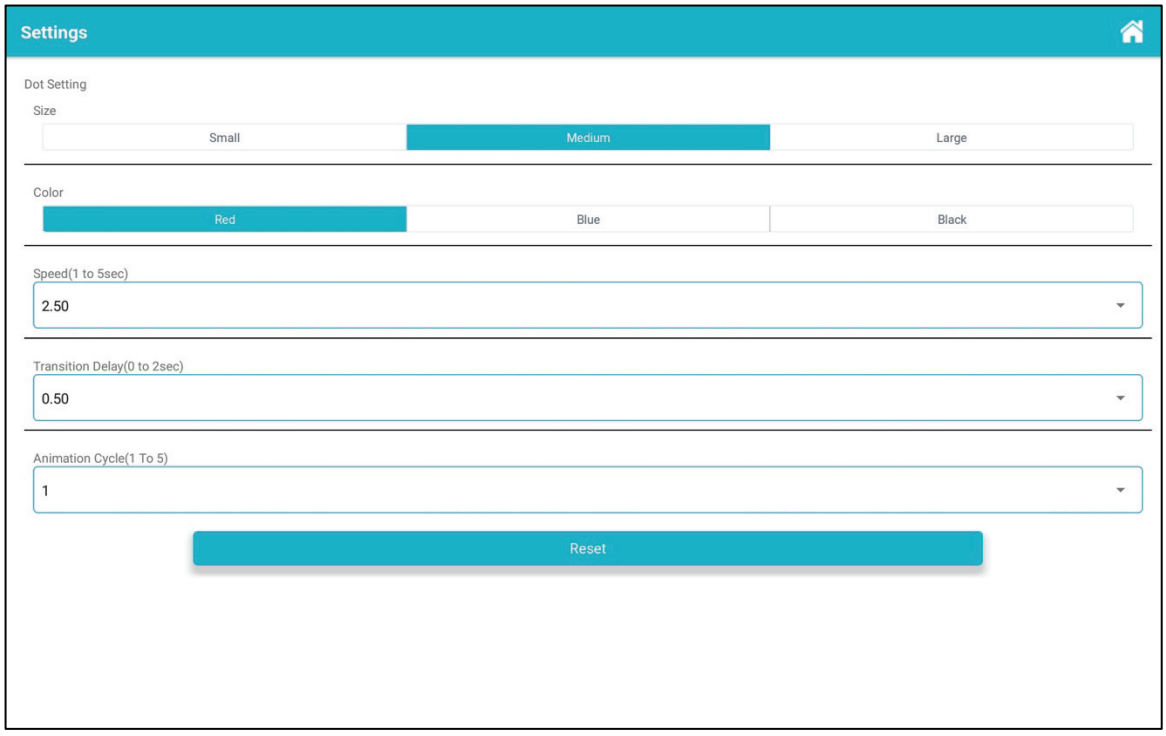
The Home Screen consists of the startup screen when the app is entered. There is a button in the top right for the settings, and a large button in the center to start the Eye Exam.



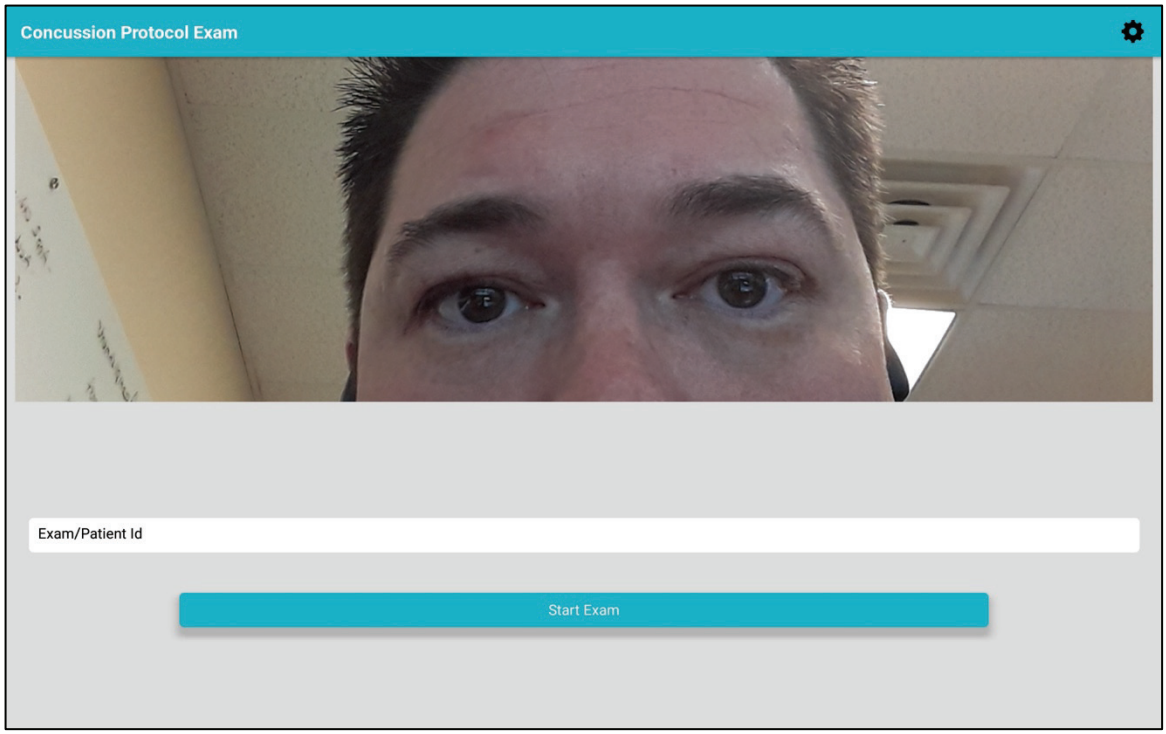
The Settings Screen consists of the options described in App Design section. Any adjustments to the settings are for evaluation in future studies and would be determined by the facilitators of those studies. The settings are saved in a local SQLite DB so they can be accessed and saved easily and persist through restarts.

The database settings are saved to the settings table. The settings table consists of 6 columns:

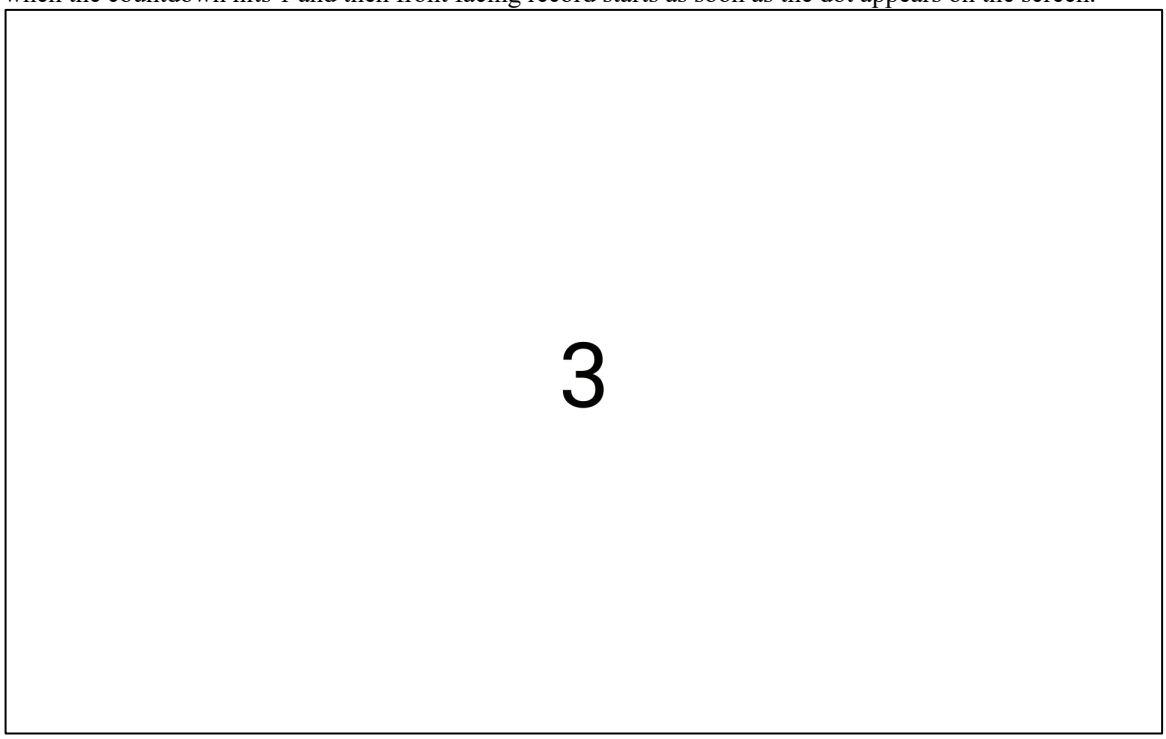
```
CREATE TABLE IF NOT EXISTS settings (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        size INTEGER,
        color TEXT,
        speed FLOAT,
        delay FLOAT,
        cycle INTEGER
);
```

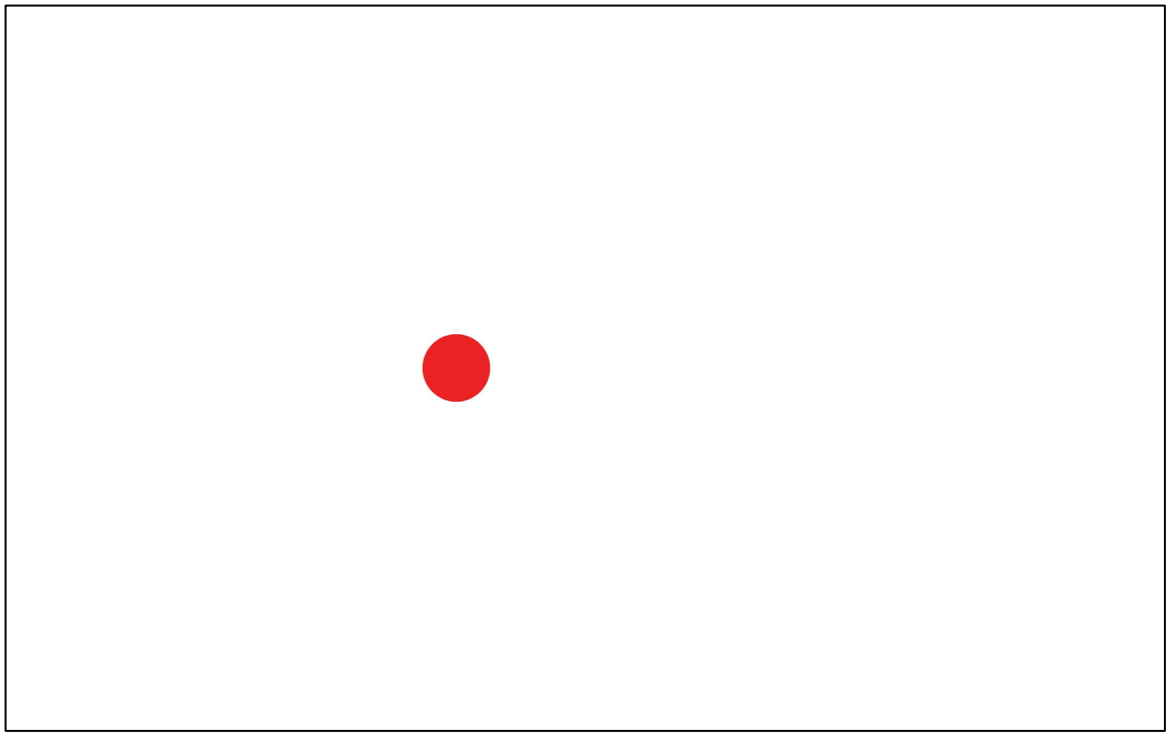


The Front Camera Viewport Aiming Screen allows you to position the device with a live view of the front facing camera. When satisfied, there is a button to start the exam. There is also a place to enter an optional patient id. This will append to the video file to help identify multiple exam files on the device.



The Exam Screen counts down from 5 to 1 and then administers the dot exam. The UI recording starts when the countdown hits 1 and then front facing record starts as soon as the dot appears on the screen.





VI. Programming issues

Most of the screens and functions of this app are pretty straight forward and are derived from standard React-Native boilerplate code that can be found online at the official React-Native tutorial [15]. The recording screen however, to be able to get the app to record the user interface had to use Native Modules had to be used as the React-Native framework does not provide for that functionality. The MediaProjection Native Module was used to get a stream of the screen capture & the MediaRecorder Native Module to record it to the internal storage.

Appendix A presents the key program files regarding the mobile app and the native UI recording module. The full source code can be found in the git repository located at https://github.com/saeohioalpha/eyetrak.

VII. Conclusion

The application successfully records the UI and the front-facing camera at the same time in the manner requested by the team working on processing the pupillary tracking software using computer vision. This application will allow further research into the pupillary tracking as a viable method for concussion detection that can be used in portable consumer grade devices that can be used immediately when a concussion is suspected to have occurred.

Based on self-experimentation, it was found that the application successfully provides a useful and quick method for recording pupillary response for post-analysis and diagnosis by a medical professional without the need for bulky equipment. It also provides a record of the eye examination that can be used to validate a diagnosis after the fact if one is given on the site of the injury that can be later used to back up that diagnosis.

Because this application runs on any Android device with a front-facing camera it is has multiple advantages over current field detection methods. These devices are highly portable, so it can be utilized immediately on the field without moving the player to a remote testing location. These devices are affordable (sub-$150) enabling an organization to have multiple devices on hand and it can be administered to multiple

people simultaneously versus large devices costing multiple thousands of dollars. Analysis on these recordings allow repeatable analysis to help determine reliability compared to a subjective pupillary response test administered by a professional and accuracy by comparing results to medically diagnosed concussions.

VIII. Future Study

Further study can also explore whether the different settings such as dot movement speed, transition delay timings, size of the dot, or cycling through the exam multiple times can have an improved chance at reliably detecting whether or not a concussion has occurred.

Additional study into the optimal device distance and placement from the subjects face to derive the best results from the post-processing functionality will also lead to a more repeatable and accurate method for administering the exam.

Integration of post-processing functionality directly into the mobile application will also provide for a better tool, but this requires certain computer vision tools to be able to be ported to the mobile operating systems, at present time only a subset of features have been. Currently OpenCV is a good candidate and has some features ported to mobile friendly development platforms, but not all of the required functionality is available yet to be able to integrate it into the mobile application itself.

Applying machine learning techniques to the collected videos in future study might also identify other metrics to help identify concussions or other diagnoses to help apply this application to other areas of research and detection.

IX. References

[1] "What Is a Concussion? | HEADS UP | CDC Injury Center," Feb. 12, 2019. https://www.cdc.gov/headsup/basics/concussion_whatis.html (accessed Oct. 31, 2019).

[2] "Concussion - Symptoms and causes," *Mayo Clinic*. https://www.mayoclinic.org/diseases-conditions/concussion/symptoms-causes/syc-20355594 (accessed Nov. 04, 2019).

[3] "Concussion Signs and Symptoms | HEADS UP | CDC Injury Center," Feb. 12, 2019. https://www.cdc.gov/headsup/basics/concussion_symptoms.html (accessed Nov. 04, 2019).

[4] "Concussion - Diagnosis and treatment - Mayo Clinic." https://www.mayoclinic.org/diseases-conditions/concussion/diagnosis-treatment/drc-20355600 (accessed Nov. 17, 2020).

[5] "Concussion Signs, Diagnosis, and Treatment | UPMC," *UPMC Sports Medicine*. https://www.upmc.com/services/sports-medicine/conditions/concussions (accessed Nov. 17, 2020).

[6] R. C. Cantu, "When to Disqualify an Athlete After a Concussion," *Curr. Sports Med. Rep.*, vol. 8, no. 1, pp. 6–7, Feb. 2009, doi: 10.1249/JSR.0b013e31819677db.

[7] W. P. Meehan, R. C. Mannix, M. J. O'Brien, and M. W. Collins, "The Prevalence of Undiagnosed Concussions in Athletes," *Clin. J. Sport Med. Off. J. Can. Acad. Sport Med.*, vol. 23, no. 5, pp. 339–342, Sep. 2013, doi: 10.1097/JSM.0b013e318291d3b3.

[8] B. Koh and A. Pearce, "How the latest technological advances in diagnosing concussion could influence sports policy," *LawInSport*, Apr. 2018.

[9] "Detect a Concussion in Seconds - App by brightlamp," *Indiegogo*. https://www.indiegogo.com/projects/2005969 (accessed Nov. 04, 2019).

[10] "Reflex - PLR Analyzer," *App Store*. https://apps.apple.com/us/app/reflex-plr-analyzer/id1412154869 (accessed Nov. 27, 2019).

[11] "Oculogica's EyeBOX: A Tech Solution to Identify Concussions," *Center for Health Technology Hunter College*, Feb. 09, 2019. https://nychealthtech.org/oculogicas-eyebox-a-tech-solution-to-identify-concussions/ (accessed Nov. 04, 2019).

[12] "Delayed concussion symptoms," May 16, 2018. https://qbi.uq.edu.au/brain/concussion/delayed-concussion-symptoms (accessed Nov. 27, 2019).

[13] "UCSD's Practical Guide to Clinical Medicine." https://meded.ucsd.edu/clinicalmed/eyes.htm (accessed Nov. 27, 2019).

[14] K. Wessel, C. Moschner, K.-P. Wandinger, D. Kömpf, and W. Heide, "Oculomotor Testing in the Differential Diagnosis of Degenerative Ataxic Disorders," *Arch. Neurol.*, vol. 55, no. 7, pp. 949–956, Jul. 1998, doi: 10.1001/archneur.55.7.949.

[15] "Learn the Basics · React Native." https://facebook.github.io/react-native/docs/tutorial (accessed Nov. 23, 2019).

A. APPENDIX

Native UI Recording Module

MainApplication.java:

```
package com.eyetest;

import android.app.Application;

import com.facebook.react.ReactApplication;
import org.pgsqlite.SQLitePluginPackage;
import com.RNFetchBlob.RNFetchBlobPackage;
import com.oblador.vectoricons.VectorIconsPackage;
import org.reactnative.camera.RNCameraPackage;
import com.swmansion.gesturehandler.react.RNGestureHandlerPackage;
import com.facebook.react.ReactNativeHost;
import com.facebook.react.ReactPackage;
import com.facebook.react.shell.MainReactPackage;
import com.facebook.soloader.SoLoader;
import org.pgsqlite.SQLitePluginPackage;


import java.util.Arrays;
import java.util.List;

public class MainApplication extends Application implements ReactApplication {

  private final ReactNativeHost mReactNativeHost = new ReactNativeHost(this) {
    @Override
    public boolean getUseDeveloperSupport() {
      return BuildConfig.DEBUG;
    }

    @Override
    protected List<ReactPackage> getPackages() {
      return Arrays.<ReactPackage>asList(
          new MainReactPackage(),
            new SQLitePluginPackage(),
            new RNFetchBlobPackage(),
            new VectorIconsPackage(),
            new RNCameraPackage(),
            new RNGestureHandlerPackage(),
            new RecorderReactPackage()
      );
    }

    @Override
    protected String getJSMainModuleName() {
      return "index";
    }
  };

  @Override
  public ReactNativeHost getReactNativeHost() {
    return mReactNativeHost;
  }
```

```
  @Override
  public void onCreate() {
    super.onCreate();
    SoLoader.init(this, /* native exopackage */ false);
  }
}
```

RecorderReactPackage.java:

```
package com.eyetest;

import com.facebook.react.ReactPackage;
import com.facebook.react.bridge.JavaScriptModule;
import com.facebook.react.bridge.NativeModule;
import com.facebook.react.bridge.ReactApplicationContext;
import com.facebook.react.uimanager.ViewManager;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class RecorderReactPackage implements ReactPackage {

  @Override
  public List<ViewManager> createViewManagers(ReactApplicationContext reactContext) {
    return Collections.emptyList();
  }

  @Override
  public List<NativeModule> createNativeModules(
                        ReactApplicationContext reactContext) {
    List<NativeModule> modules = new ArrayList<>();

    modules.add(new RecorderManager(reactContext));

    return modules;
  }

  // Backward compatibility
   public List<Class<? extends JavaScriptModule>> createJSModules() {
      return new ArrayList<>();
    }

}
```

RecorderManager.java:

```
package com.eyetest;

import android.widget.Toast;

import com.facebook.react.bridge.ReactApplicationContext;
import com.facebook.react.bridge.ReactContextBaseJavaModule;
import com.facebook.react.bridge.ReactMethod;
import com.facebook.react.modules.core.DeviceEventManagerModule;
import com.facebook.react.bridge.Callback;
```

```
import java.lang.ref.WeakReference;



public class RecorderManager extends ReactContextBaseJavaModule {

  private static WeakReference<MainActivity> mWeakActivity;

  public RecorderManager(ReactApplicationContext reactContext) {
    super(reactContext);
  }

  public static void updateActivity(MainActivity activity) {
    mWeakActivity = new WeakReference<MainActivity>(activity);
  }


  @Override
  public String getName() {
    return "RecorderManager";
  }

  @ReactMethod
  public void start(Callback successCallback, Callback errorCallback) {
   try{
      mWeakActivity.get().startRecording();
      successCallback.invoke();
   }catch(Exception e){
     errorCallback.invoke(e.getMessage());
   }
    // Toast.makeText(getReactApplicationContext(), "started", Toast.LENGTH_SHORT).show();
  }

  @ReactMethod
  public void stop(Callback successCallback, Callback errorCallback) {
    try{
      mWeakActivity.get().stopRecording();
    String filePath = mWeakActivity.get().getVideoPath();
    getReactApplicationContext()
            .getJSModule(DeviceEventManagerModule.RCTDeviceEventEmitter.class)
            .emit("updateFilePath", filePath);

    successCallback.invoke();

    }catch(Exception e){
       errorCallback.invoke(e.getMessage());
    }
    // Toast.makeText(getReactApplicationContext(), "stopped", Toast.LENGTH_SHORT).show();

  }
}
```

MainActivity.java:

```
package com.eyetest;
```

```
import com.facebook.react.ReactActivity;
import com.facebook.react.ReactActivityDelegate;
import com.facebook.react.ReactRootView;
import com.swmansion.gesturehandler.react.RNGestureHandlerEnabledRootView;
import android.view.View;
import android.annotation.SuppressLint;
import android.annotation.TargetApi;
import android.content.Context;
import android.content.Intent;
import android.hardware.display.DisplayManager;
import android.hardware.display.VirtualDisplay;
import android.media.MediaRecorder;
import android.media.projection.MediaProjection;
import android.media.projection.MediaProjectionManager;
import android.os.Build;
import android.os.Bundle;
import android.os.Environment;
import android.util.DisplayMetrics;
import android.util.Log;
import android.util.SparseIntArray;
import android.view.Surface;
import android.view.Window;
import android.view.WindowManager;
import com.rn.full.screen.FullScreenModule;
import java.util.Date;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import org.pgsqlite.SQLitePluginPackage;
import java.io.IOException;

public class MainActivity extends ReactActivity {

    private static final String TAG = "MainActivity";
    private static final int REQUEST_CODE = 1000;
    private int mScreenDensity;
    private MediaProjectionManager mProjectionManager;
    private static final int DISPLAY_WIDTH = 720;
    private static final int DISPLAY_HEIGHT = 1280;
    private MediaProjection mMediaProjection;
    private VirtualDisplay mVirtualDisplay;
    private MediaProjectionCallback mMediaProjectionCallback;
    private MediaRecorder mMediaRecorder;
    private static final SparseIntArray ORIENTATIONS = new SparseIntArray();
    private String videoPath;

    static {
        ORIENTATIONS.append(Surface.ROTATION_0, 90);
        ORIENTATIONS.append(Surface.ROTATION_90, 0);
        ORIENTATIONS.append(Surface.ROTATION_180, 270);
        ORIENTATIONS.append(Surface.ROTATION_270, 180);
    }
```

```
    @SuppressLint("NewApi")
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
         this.getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
                WindowManager.LayoutParams.FLAG_FULLSCREEN);
                hideNavigationBar();
        RecorderManager.updateActivity(this);


        DisplayMetrics metrics = new DisplayMetrics();
        getWindowManager().getDefaultDisplay().getMetrics(metrics);
        mScreenDensity = metrics.densityDpi;

        mMediaRecorder = null;

        mProjectionManager = (MediaProjectionManager)
getSystemService(Context.MEDIA_PROJECTION_SERVICE);
    }


    @Override
    public void onWindowFocusChanged(boolean hasFocus) {
        super.onWindowFocusChanged(hasFocus);
        if (hasFocus) {
            hideNavigationBar();
        }
    }

    private void hideNavigationBar() {

getWindow().getDecorView().setSystemUiVisibility(View.SYSTEM_UI_FLAG_HIDE_NAVIGATION);
    }


    @SuppressLint("NewApi")
    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (requestCode != REQUEST_CODE) {
            Log.e(TAG, "Unknown request code: " + requestCode);
            return;
        }
        if (resultCode != RESULT_OK) {
            mMediaRecorder = null;
            mMediaProjection = null;
            return;
        }

        try {
            mMediaProjectionCallback = new MediaProjectionCallback();
            mMediaProjection = mProjectionManager.getMediaProjection(resultCode, data);
            mMediaProjection.registerCallback(mMediaProjectionCallback, null);
            mVirtualDisplay = createVirtualDisplay();
            mMediaRecorder.start();
        } catch (Exception e) {
```

```
        e.printStackTrace();
    }
}


protected ReactActivityDelegate createReactActivityDelegate(){
    return new ReactActivityDelegate(this, getMainComponentName()) {
        @Override
        protected ReactRootView createRootView() {
            return new RNGestureHandlerEnabledRootView(MainActivity.this);
        }
    };
}

public void startRecording() {

    try {
        initRecorder();
        shareScreen();
    } catch (Exception e) {
        e.printStackTrace();
        mMediaRecorder = null;
        mMediaProjection = null;
    }
}

public void stopRecording() {

    if (mMediaRecorder == null) {
        return;
    }
    try {
        mMediaRecorder.setOnErrorListener(null);
        mMediaRecorder.stop();
        mMediaRecorder.reset();
        stopScreenSharing();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

public String getVideoPath() {
    return videoPath;
}

private void shareScreen() {
    if (mMediaProjection == null) {
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
            startActivityForResult(mProjectionManager.createScreenCaptureIntent(), REQUEST_CODE);
        }
        return;
    }
    mVirtualDisplay = createVirtualDisplay();
    mMediaRecorder.start();
}
```

```
    @TargetApi(Build.VERSION_CODES.LOLLIPOP)
    private VirtualDisplay createVirtualDisplay() {
        return mMediaProjection.createVirtualDisplay("MainActivity",
                DISPLAY_WIDTH, DISPLAY_HEIGHT, mScreenDensity,
                DisplayManager.VIRTUAL_DISPLAY_FLAG_AUTO_MIRROR,
                mMediaRecorder.getSurface(), null /*Callbacks*/, null
                /*Handler*/);
    }

    private void initRecorder() {
        try {

            Date date = Calendar.getInstance().getTime();
            DateFormat dateFormat = new SimpleDateFormat("yyyyMMdd-HHmmss");
            String strDate = dateFormat.format(date);

            mMediaRecorder = new MediaRecorder();
            mMediaRecorder.setVideoSource(MediaRecorder.VideoSource.SURFACE);
            mMediaRecorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
            videoPath =
Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOWNLOADS)+ "/"+
strDate +"_sr.mp4";
            mMediaRecorder.setOutputFile(videoPath);
            mMediaRecorder.setVideoSize(DISPLAY_WIDTH, DISPLAY_HEIGHT);
            mMediaRecorder.setVideoEncoder(MediaRecorder.VideoEncoder.H264);
            mMediaRecorder.setVideoEncodingBitRate(512 * 1000);
            mMediaRecorder.setVideoFrameRate(30);
            int rotation = getWindowManager().getDefaultDisplay().getRotation();
            int orientation = ORIENTATIONS.get(rotation + 90);
            mMediaRecorder.setOrientationHint(orientation);
            mMediaRecorder.prepare();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    @TargetApi(Build.VERSION_CODES.LOLLIPOP)
    private class MediaProjectionCallback extends MediaProjection.Callback {
        @Override
        public void onStop() {
            stopRecording();
        }
    }

    private void stopScreenSharing() {
        if (mVirtualDisplay == null) {
            return;
        }
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.KITKAT) {
            mVirtualDisplay.release();
        }
        mVirtualDisplay = null;
        mMediaRecorder.release();
        mMediaRecorder = null;
        destroyMediaProjection();
    }
```

```
    @Override
    public void onDestroy() {
        super.onDestroy();
        destroyMediaProjection();
    }

    private void destroyMediaProjection() {
        if (mMediaProjection != null) {
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
                mMediaProjection.unregisterCallback(mMediaProjectionCallback);
                mMediaProjection.stop();
                mMediaProjection = null;
            }

        }
        Log.i(TAG, "MediaProjection Stopped");
    }

    /**
     * Returns the name of the main component registered from JavaScript.
     * This is used to schedule rendering of the component.
     */
    @Override
    protected String getMainComponentName() {
        return "eyetest";
    }
}
```

ReactNative Application

App.js:

```
/**
 * Sample React Native App
 * https://github.com/facebook/react-native
 *
 * @format
 * @flow
 */

import React, {Component} from 'react';
import { createAppContainer, createStackNavigator, StackActions, NavigationActions } from 'react-
navigation';

import {connect} from "react-redux"

import Home from './Home'
import CameraScreen from './CameraScreen'
import MovingScreen from './MovingScreen'
import Settings from './Settings'
import { ActionChangeColor, ActionChangeSize, ActionChangeSpeed,
ActionChangeDirection,ActionChangeDelay,ActionChangeCycle, ActionChangeActiveColor,
ActionChangeActiveSize, ActionChangePathing } from './reducers/index.js';
```

```
const ConnectMovingScreen = connect(MapStateForMovingScreen)(MovingScreen)

const MapStateForMovingScreen = state => {
  const {settings} = state
  return {
    ...settings
  }
}
const MapStateForSettings = state => {
  const {settings} = state
  return {
    ...settings
  }
}

const MapActionsforSettings = dispatch => {
  return {
      changeColor: color => {
        dispatch({type:ActionChangeColor,payload:color})
      },
      changeSize: size => {
        dispatch({type:ActionChangeSize,payload:size})
      },
      changeSpeed: speedValue => {
        dispatch({type:ActionChangeSpeed,payload:speedValue})
      },

      changeDelay:delay => {
        dispatch({type:ActionChangeDelay,payload:delay})
      },
      changeCycle:cycle => {
        dispatch({type:ActionChangeCycle,payload:cycle})
      },
      changePath:pathing => {
        dispatch({type:ActionChangePathing,payload:pathing})
      },
      modifyPath:(payload) => {
        dispatch({type:ActionChangeDirection,payload})
      },

      changeActiveColorIndex :(index) => {
          dispatch({type:ActionChangeActiveColor,payload:index})
      },

      changeActiveSizeIndex: (index) => {
        dispatch({type:ActionChangeActiveSize,payload:index})
      }

  }
}

const ConnectSettings = connect(MapStateForSettings,MapActionsforSettings)(Settings)

const AppNavigator = createStackNavigator({
  Home: {
    screen: Home,
```

```
  },
  CameraScreen: {
    screen: CameraScreen,
    navigationOptions: {
      headerTitleStyle: { alignSelf: 'center' },
      title: 'Center Title',
      gesturesEnabled: true,
    },
  },
  MovingScreen: {
    screen:ConnectMovingScreen,
  },
  Settings: {
    screen:ConnectSettings,
    navigationOptions: {
      headerTitleStyle: { alignSelf: 'center' },
      title: 'Center Title',
      gesturesEnabled: true,
    },
  }
},
{
    initialRouteName: 'Home',
    defaultNavigationOptions: {
      headerStyle: {
        backgroundColor: '#05bbd3',
      },
      headerTintColor: '#fff',
      headerTitleStyle: {
        fontWeight: 'bold',
        textAlign:"center"
      },
    },
});
export default createAppContainer(AppNavigator);


class App extends Component {
  constructor(props) {
    super(props);
  }
  render() {
      return (
      <AppNavigator />
    );

  }
}
```

Home.js:

```
/**

 *
 * @format
 * @flow
 */
```

```
import React, {Component} from 'react';
import { createAppContainer, createStackNavigator, StackActions, NavigationActions } from 'react-
navigation';
import styles from '../stylesheet.js';
import {
  Easing,
  NativeModules
} from 'react-native';
import * as Animatable from 'react-native-animatable';
import {AppRegistry,PermissionsAndroid, Picker, Platform,ScrollView, AppState, TouchableHighlight,
Animated, StyleSheet, Text, View, Image, ImageBackground, TextInput, Button, Alert, TouchableOpacity,
DeviceEventEmitter, BackHandler } from 'react-native';

import SQLite from "react-native-sqlite-storage";
import CreateTables from './res/manager/CreateTables'
import { localDB } from './res/constants/constants'
import { withNavigation } from 'react-navigation';
import  {
  BackAndroid,
} from 'react-native';
import {connect} from "react-redux"
var db = SQLite.openDatabase(localDB.dbName, "1.0", "reactDemo Database", 200000);

export default class Home extends Component {
    static navigationOptions =({navigation})=> ({
      headerLeft:null,
      headerTitle: 'Concussion Protocol Exam',
      headerRight: (
        <TouchableOpacity
          onPress={()=> navigation.navigate('Settings')}
        >
          <Image source={require('../images/settings.png')} style={{ width:25, height:25, marginRight:20 }}/>
        </TouchableOpacity>
      ),
    });

    constructor(props) {
      super(props);
      this.state = {
        userData: '',
      };
    }
    componentDidMount = () => {
      //Checking for the permission just after component loaded
       async function requestStoragePermission() {
         try {
           const granted = await PermissionsAndroid.request(
             PermissionsAndroid.PERMISSIONS.WRITE_EXTERNAL_STORAGE,{
               'title': 'AndoridPermissionExample App storage Permission',
               'message': 'AndoridPermissionExample App needs access to your storage '
             }
           )
          //  if (granted === PermissionsAndroid.RESULTS.GRANTED) {
          //    //To Check, If Permission is granted
          //    alert("You can use the Storage");
```

```
        //  } else {
        //    alert("Storage permission denied");
        //  }
        } catch (err) {
          alert("err",err);
          console.warn(err)
        }
      }
      if (Platform.OS === 'android') {
         //Calling the permission function
         requestStoragePermission();
      }else{
         alert('IOS device found');
      }

      db.transaction(function (txn) {
       //txn.executeSql('DROP TABLE IF EXISTS ' + localDB.tableName.tblLogin, []);
       txn.executeSql('CREATE TABLE IF NOT EXISTS ' + localDB.tableName.settings + ' (id  INTEGER
PRIMARY KEY AUTOINCREMENT,size INTEGER,color TEXT,speed FLOAT,delay FLOAT,cycle
INTEGER, pathing INTEGER )', []);
       console.log('create databse success.')
    });

    db.transaction(function(txn) {
      txn.executeSql(
        "SELECT * FROM settings",
        [],
        (tx, res) => {
          console.log(res.rows.length);
          console.log( res.rows.item(0));
          if (res.rows.length == 0) {
            txn.executeSql(
              "INSERT INTO " +
                localDB.tableName.settings +
                " (size,color,speed,delay,cycle,pathing) VALUES (:size,:color,:speed,:delay,:cycle,:pathing)",
              [
                75,
                "red",
                1.0,
                0.75,
                1,
                0
              ],
              (txn, results) => {
                console.log('Results', results.rowsAffected);
                if (results.rowsAffected > 0) {
                  Alert.alert(
                    'Success',
                  );
                }
                }
            );
          }
        }
      );
    });
```

```
  }
  render() {
   return (
     <View style={styles.container}>
      <View style={styles.ImageContainer}>
         <Image
          style={{ height:150, width:150, position:"absolute",top:50, resizeMode:"contain",  padding:30 }}
          source={require('./eye.png')}
         />
        <TouchableOpacity style={[styles.loginbtn, styles.positionBottom]}
         onPress={() => {
          this.props.navigation.dispatch(StackActions.reset({
           index: 0,
           actions: [
            NavigationActions.navigate({ routeName: 'CameraScreen' })
           ],
          }))
        }}

        >
         <Text style={styles.text}>Begin Test</Text>
        </TouchableOpacity>
      </View>
     </View>
    );
  }
 }
```

CameraScreen.js

```
/**

 *
 * @format
 * @flow
 */

import React, {Component} from 'react';
import { createAppContainer, createStackNavigator, StackActions, NavigationActions } from 'react-
navigation';
import styles from '../stylesheet.js';
import {
  Easing,
  NativeModules
} from 'react-native';
import * as Animatable from 'react-native-animatable';
import {AppRegistry,PermissionsAndroid, Picker, Platform,ScrollView, AppState, TouchableHighlight,
Animated, StyleSheet, Text, View, Image, ImageBackground, TextInput, Button, Alert, TouchableOpacity,
DeviceEventEmitter, BackHandler } from 'react-native';
import { RNCamera } from 'react-native-camera';

import SQLite from "react-native-sqlite-storage";

import { localDB } from './res/constants/constants'
import { withNavigation } from 'react-navigation';
import  {
```

```
  BackAndroid,
} from 'react-native';
import {connect} from "react-redux"
var db = SQLite.openDatabase(localDB.dbName, "1.0", "reactDemo Database", 200000);

export default class CameraScreen extends Component{


   static navigationOptions =({navigation})=> ({
    headerTitle: 'Concussion Protocol Exam',
    headerRight: (
     <TouchableOpacity
       onPress={()=> navigation.navigate('Settings')}
     >
      <Image source={require('../images/settings.png')} style={{ width:25, height:25, marginRight:20 }}/>
     </TouchableOpacity>
    ),
   });

   constructor(props) {

    super(props)

    this.state = {

     TextInput_Name: '',
     NumberHolder : 1,
     CurrentDate:''
    }

   }

   componentDidMount(){

     this.GenerateRandomNumber()
     this.setCurrentdateTime();

    db.transaction(function (txn) {
      //txn.executeSql('DROP TABLE IF EXISTS ' + localDB.tableName.tblLogin, []);
      txn.executeSql('CREATE TABLE IF NOT EXISTS ' + localDB.tableName.patient + ' (id  INTEGER
PRIMARY KEY AUTOINCREMENT,userId TEXT,patientName TEXT,datetime TEXT )', []);
      console.log('create databse success.')
   });


   }

   setCurrentdateTime=()=>{

    var date = new Date().getDate(); //Current Date
    var month = new Date().getMonth() + 1; //Current Month
    var year = new Date().getFullYear(); //Current Year
    var hours = new Date().getHours(); //Current Hours
    var min = new Date().getMinutes(); //Current Minutes
    var sec = new Date().getSeconds(); //Current Seconds
     if (date < 10) {
```

```
    date = '0' + date;
    }
    if (month < 10) {
      month = '0' + month;
    }
    if (hours < 10) {
      hours = '0' + hours;
    }
    if (min < 10) {
      min = '0' + min;
    }
    if (sec < 10) {
      sec = '0' + sec;
    }
   // this.setState({
   //   //Setting the value of the date time
   //   CurrentDate:
   //   ,
   // });

   var dateTime= year + '' + month + '' + date + '-' + hours + '' + min + '' + sec
   this.setState({
    CurrentDate: dateTime
   })


 }


GenerateRandomNumber=()=>
{

var RandomNumber = Math.floor(Math.random() * 1000) + 1 ;

this.setState({

  NumberHolder : RandomNumber

})
this.state.NumberHolder
// Alert.alert(RandomNumber)
}

  Send_Data_Function = () => {

   this.props.navigation.navigate('MovingScreen', {
      NameOBJ: this.state.TextInput_Name
   });
   // var userId= this.state.TextInput_Name+this.state.NumberHolder.toString()
   // Alert.alert(this.state.CurrentDate)
   // db.transaction(function(txn) {
   //   txn.executeSql(
   //     "INSERT INTO " +
   //       localDB.tableName.patient +
   //       " (userId,patientName,datetime) VALUES (:userId,:patientName,:datetime)",
   //     [
```

```
  //       userId,
  //       this.state.TextInput_Name,
  //       this.state.CurrentDate,
  //     ],
  //     (txn, results) => {
  //       console.log('Results', results.rowsAffected);
  //       if (results.rowsAffected > 0) {
  //         this.props.navigation.navigate('Move', {
  //              NameOBJ: userId
  //           });
  //       }
  //       }
  //    );
  // }.bind(this));


}

render(){
  return(
      <View style={styles.cameracontainer}>
        <View style={{flex: 2, backgroundColor: 'powderblue', margin:10,}} >
        <RNCamera
        style={styles.pcamera}
         ref={ref => {
         this.camera = ref;
        }}
        // captureMode={RNCamera.constants.CaptureMode.video}
        // aspect={RNCamera.constants.Aspect.fill}
        type={RNCamera.Constants.Type.front}
        flashMode={RNCamera.Constants.FlashMode.on}
        onGoogleVisionBarcodesDetected={({ barcodes }) => {
          console.log(barcodes);
        }}
      />

        </View>
        <View style={styles.textContainer} >
          <View style={styles.inputContainer}>
            {/* <Text style={styles.inputText}></Text> */}
            <TextInput style={styles.input}
             onChangeText={data => this.setState({ TextInput_Name: data })}
             placeholder={'Exam/Patient Id'}
              placeholderTextColor={'#000'} />
          </View>
          <View style={{width:'100%',justifyContent:'center', alignItems:'center',}}>
          <TouchableOpacity style={styles.loginbtn}
               onPress={this.Send_Data_Function}
           >
            <Text style={styles.text}>Start Exam</Text>
          </TouchableOpacity>
          </View>
        </View>
      </View>
  );
}
```

```
  // takePicture = async function() {
  //   if (this.camera) {
  //     const options = { quality: 0.5, base64: true };
  //     const data = await this.camera.takePictureAsync(options)
  //     console.log(data.uri);
  //   }
  // };
 }
```

MovingScreen.js:

```
/**

 *
 * @format
 * @flow
 */

import React, {Component} from 'react';
import { createAppContainer, createStackNavigator, StackActions, NavigationActions } from 'react-
navigation';
import styles from '../stylesheet.js';
import {
  Easing,
  NativeModules
} from 'react-native';
import * as Animatable from 'react-native-animatable';
import {AppRegistry,PermissionsAndroid, Picker, Platform,ScrollView, AppState, TouchableHighlight,
Animated, StyleSheet, Text, View, Image, ImageBackground, TextInput, Button, Alert, TouchableOpacity,
DeviceEventEmitter, BackHandler } from 'react-native';
import { RNCamera } from 'react-native-camera';
import { ButtonGroup,Slider } from 'react-native-elements';
import { Divider } from 'react-native-elements';
import { Dimensions } from 'react-native';
import ToggleSwitch from 'toggle-switch-react-native'
import RNFetchBlob from 'react-native-fetch-blob'
import {ToastAndroid} from 'react-native';
import SQLite from "react-native-sqlite-storage";
import CreateTables from './res/manager/CreateTables'
import { localDB } from './res/constants/constants'
import { withNavigation } from 'react-navigation';
import  {
  BackAndroid,
} from 'react-native';
import {connect} from "react-redux"
import { ActionChangeColor, ActionChangeSize, ActionChangeSpeed,
ActionChangeDirection,ActionChangeDelay,ActionChangeCycle, ActionChangeActiveColor,
ActionChangeActiveSize, ActionChangePathing } from './reducers/index.js';

var db = SQLite.openDatabase(localDB.dbName, "1.0", "reactDemo Database", 200000);
const { RecorderManager } = NativeModules;  //import screen recording from native modules
(android/app/src/main/java/com/eyetest/mainActivity.java)

export default class MovingScreen extends Component{
  static navigationOptions =({navigation})=> ({header: null});
```

```
 constructor(props){
  super(props);
  this.state = {
   timer:5,
   timerRunning:false,
   recording:false,
   userData: '',
   currentTime: Date.now()
  }
  this.onTimerComplete = this.onTimerComplete.bind(this);
  this.startRecording=this.startRecording.bind(this);
  this.stopRecording=this.stopRecording.bind(this);
  this.startSpy = this.startSpy.bind(this);
  this.writeFile = this.writeFile.bind(this);
  this._isMounted = false;
  this.animatedValue = new Animated.Value(-70);
  this.translateValue = new Animated.ValueXY({x: 0, y: 0});
 }
 state = {appState: AppState.currentState};

 /*  ------------------------------BEGIN onTimerComplete Function------------------------  */
 onTimerComplete(){
  // console.log('====================================');
  // console.log(this.camera);
  console.log('====================================');
  setTimeout( () => {
   this.animate();
   this.startTimer();
   console.log('animation start');
  }, 200);
  console.log('====================================');
 }
 /*  ------------------------------END onTimerComplete Function------------------------  */

 /*  ------------------------------BEGIN startTimer Function------------------------  */
 startTimer(){
  if (this.state.userData.pathing) {
   this.duration = 750 + ( (
    2 * ( ( ( Dimensions.get('window').width / 2 - this.state.userData.size / 2 ) / this.state.userData.speed )
* 10 ) + // 2 short durations along the x-axis
    1 * ( ( ( ( Dimensions.get('window').width / 2 - this.state.userData.size / 2 ) / this.state.userData.speed )
* 10 ) * 2 ) + // 1 long durations along the x-axis
    0 * ( ( ( Dimensions.get('window').height / 2 - this.state.userData.size / 2 ) / this.state.userData.speed )
* 10 ) + // 0 short durations along the y-axis
    0 * ( ( ( ( Dimensions.get('window').height / 2 - this.state.userData.size / 2 ) / this.state.userData.speed
) * 10 ) * 2 ) + // 0 long durations along the y-axis
    3 * ( this.state.userData.delay * 1000 ) // 3 transition point (TP) delays (incl. Start Delay)
   ) * this.state.userData.cycle );
  } else {
   this.duration = 750 + ( (
    2 * ( ( ( Dimensions.get('window').width / 2 - this.state.userData.size / 2 ) / this.state.userData.speed )
* 10 ) + // 2 short durations along the x-axis
    1 * ( ( ( ( Dimensions.get('window').width / 2 - this.state.userData.size / 2 ) / this.state.userData.speed )
* 10 ) * 2 ) + // 1 long durations along the x-axis
    4 * ( ( ( Dimensions.get('window').height / 2 - this.state.userData.size / 2 ) / this.state.userData.speed )
* 10 ) + // 4 short durations along the y-axis
```

```
      2 * ( ( ( ( Dimensions.get('window').height / 2 - this.state.userData.size / 2 ) / this.state.userData.speed
) * 10 ) * 2 ) + // 2 long durations along the y-axis
      9 * ( this.state.userData.delay * 1000 ) // 9 transition point (TP) delays (incl. Start Delay)
      ) * this.state.userData.cycle );
   }

   setTimeout( () => {
     console.log( this.duration );
     this.stopRecording();
   }, this.duration );
 }
 /*  ------------------------------END startTimer Function----------------------  */

 /*  ------------------------------BEGIN screen recording start Function----------------------  */
 startSpy(){
   RecorderManager.start(
     () => console.log("SUCCESS IN RECORDING START"),
     (error) => console.log("Error: " + error)
   );
 }
 /*  ------------------------------END screen recording start Function----------------------  */

 /*  ------------------------------BEGIN screen recording stop Function----------------------  */
 stopSpy(){
   RecorderManager.stop(
     () => console.log("SUCCESS IN STOP "),
     (error) => console.log("Error: " + error)
   );
 }
 /*  ------------------------------END screen recording stop Function----------------------  */

 /*  ------------------------------BEGIN Front-Facing recording stop Function----------------------  */
 stopRecording(){
   this.duration = 750;

   setTimeout( () => {
     console.log( this.duration );
     this.stopSpy(); //stop screen recording
   }, this.duration );

   setTimeout( () => {
     console.log( this.duration );
     this.camera.stopRecording(); //stop front facing
   }, this.duration );

   setTimeout( () => {
     console.log( this.duration );
     this.stopapp(); //back to home screen
   }, this.duration );
 }
 /*  ------------------------------END Front-Facing recording stop Function----------------------  */

 /*  ------------------------------BEGIN Front-Facing recording start Function----------------------  */
 startRecording() {
   // this.startSpy()
   this._isMounted = true;
```

```
  this._isMounted && this.setState({ recording: true });
  // default to mp4 for android as codec is not set
  this.camera.recordAsync().then(data => {
    const {uri} = data
    RNFetchBlob.fs.readFile(
      uri,
      'base64'
    ).then( this.writeFile )
    .catch(
      err => console.log('FILE READ ERROR: ' + err.message)
    )
  }).catch(a=>console.log(a.message))
 }
 /*  ------------------------------END Front-Facing recording start Function-----------------------  */

 /*  ------------------------------BEGIN file write Function-----------------------  */
 writeFile(data){
  // Alert.alert();
  console.log(this.state.patientName);
  RNFetchBlob.fs.writeFile(

RNFetchBlob.fs.dirs.DownloadDir+`/${this.state.CurrentDate}_${this.props.navigation.state.params.Name
OBJ}_vr.mp4`,
    data,
    'base64'
  ).then(
    ()=> {
      this.setState( state => ( { ...state, examStatus:'completed' } ) )
    }
  ).catch(console.log);
 }
 /*  ------------------------------END file write Function-----------------------  */

 /*  ------------------------------BEGIN setcurrent datetime-----------------------  */
 setCurrentdateTime = () => {
  var date = new Date().getDate(); //Current Date
  var month = new Date().getMonth() + 1; //Current Month
  var year = new Date().getFullYear(); //Current Year
  var hours = new Date().getHours(); //Current Hours
  var min = new Date().getMinutes(); //Current Minutes
  var sec = new Date().getSeconds() + 4; //Current Seconds
  if (date < 10) {
    date = '0' + date;
  }
  if (month < 10) {
    month = '0' + month;
  }
  if (hours < 10) {
    hours = '0' + hours;
  }
  if (min < 10) {
    min = '0' + min;
  }
  if (sec < 10) {
    sec = '0' + sec;
  }
```

```
  var dateTime = year + '' + month + '' + date + '-' + hours + '' + min + '' + sec;
  this.setState({
    CurrentDate: dateTime
  });
 }
 /*  ------------------------------END setcurrent datetime----------------------  */

 /*  ------------------------------BEGIN animate Function----------------------  */
 animate() {
  //console.log(this.state.userData.pathing)
  if (this.state.userData.pathing) {
   this.animateLTRPath();
  } else {
   this.animateHPath();
  }
 }
 /*  ------------------------------END animate Function----------------------  */

 /*  ------------------------------BEGIN animateLTRPath Function----------------------  */
 animateLTRPath() {
  Animated.sequence([
   Animated.loop(
    Animated.sequence([
     /* Start point to Transition Point 1 (TP1) (Short)*/
     Animated.timing(
      this.translateValue,
      {
       toValue: {
        x: -( ( Dimensions.get('window').width / 2 ) - ( this.state.userData.size / 2 ) ),
        y: 0
       },
       duration: ( ( ( Dimensions.get('window').width / 2 ) - ( this.state.userData.size / 2 ) ) /
this.state.userData.speed ) * 10,
       delay: ( this.state.userData.delay * 1000 ),
       easing: Easing.linear
      }
     ),
     /* TP4 to TP5 (Long) */
     Animated.timing(
      this.translateValue,
      {
       toValue: {
        x: ( ( Dimensions.get('window').width / 2 ) - ( this.state.userData.size / 2 ) ),
        y: 0
       },
       duration: ( ( ( ( Dimensions.get('window').width / 2 ) - ( this.state.userData.size / 2 ) ) /
this.state.userData.speed ) * 10 ) * 2,
       delay: ( this.state.userData.delay * 1000 ),
       easing: Easing.linear
      }
     ),
     /* TP8 to Finish point (Short) */
     Animated.timing(
      this.translateValue,
      {
       toValue: {
```

```
            x: 0,
            y: 0
           },
           duration: ( ( ( Dimensions.get('window').width / 2 ) - ( this.state.userData.size / 2 ) ) /
this.state.userData.speed ) * 10,
           delay: ( this.state.userData.delay * 1000 ),
           easing: Easing.linear
          }
         )
        ]),
        { iterations: this.state.userData.cycle }
      )
    ]).start( console.log( 'animation finish' ) );
  }
  /*  ------------------------------END animateLTRPath Function----------------------  */

  /*  ------------------------------BEGIN animateHPath Function----------------------  */
  animateHPath() {
    Animated.sequence([
      Animated.loop(
        Animated.sequence([
          /* Start point to Transition Point 1 (TP1) (Short)*/
          Animated.timing(
            this.translateValue,
            {
              toValue: {
                x: -( ( Dimensions.get('window').width / 2 ) - ( this.state.userData.size / 2 ) ),
                y: 0
              },
              duration: ( ( ( Dimensions.get('window').width / 2 ) - ( this.state.userData.size / 2 ) ) /
this.state.userData.speed ) * 10,
              delay: ( this.state.userData.delay * 1000 ),
              easing: Easing.linear
            }
          ),
          /* TP1 to TP2 (Short)*/
          Animated.timing(
            this.translateValue,
            {
              toValue: {
                x: -( ( Dimensions.get('window').width / 2 ) - ( this.state.userData.size / 2 ) ),
                y: -( ( Dimensions.get('window').height / 2 ) - ( this.state.userData.size / 2 ) )
              },
              duration: ( ( ( Dimensions.get('window').height / 2 ) - ( this.state.userData.size / 2 ) ) /
this.state.userData.speed ) * 10,
              delay: ( this.state.userData.delay * 1000 ),
              easing: Easing.linear
            }
          ),
          /* TP2 to TP3 (Long) */
          Animated.timing(
            this.translateValue,
            {
              toValue: {
                x: -( ( Dimensions.get('window').width / 2 ) - ( this.state.userData.size / 2 ) ),
                y: ( ( Dimensions.get('window').height / 2 ) - ( this.state.userData.size / 2 ) )
```

```
          },
          duration: ( ( ( ( Dimensions.get('window').height / 2 ) - ( this.state.userData.size / 2 ) ) /
this.state.userData.speed ) * 10 ) * 2,
          delay: ( this.state.userData.delay * 1000 ),
          easing: Easing.linear
        }
      ),
      /* TP3 to TP4 (Short) */
      Animated.timing(
        this.translateValue,
        {
          toValue: {
            x: -( ( Dimensions.get('window').width / 2 ) - ( this.state.userData.size / 2 ) ),
            y: 0
          },
          duration: ( ( ( Dimensions.get('window').height / 2 ) - ( this.state.userData.size / 2 ) ) /
this.state.userData.speed ) * 10,
          delay: ( this.state.userData.delay * 1000 ),
          easing: Easing.linear
        }
      ),
      /* TP4 to TP5 (Long) */
      Animated.timing(
        this.translateValue,
        {
          toValue: {
            x: ( ( Dimensions.get('window').width / 2 ) - ( this.state.userData.size / 2 ) ),
            y: 0
          },
          duration: ( ( ( ( Dimensions.get('window').width / 2 ) - ( this.state.userData.size / 2 ) ) /
this.state.userData.speed ) * 10 ) * 2,
          delay: ( this.state.userData.delay * 1000 ),
          easing: Easing.linear
        }
      ),
      /* TP5 to TP6 (Short) */
      Animated.timing(
        this.translateValue,
        {
          toValue: {
            x: ( ( Dimensions.get('window').width / 2 ) - ( this.state.userData.size / 2 ) ),
            y: -( ( Dimensions.get('window').height / 2 ) - ( this.state.userData.size / 2 ) )
          },
          duration: ( ( ( Dimensions.get('window').height / 2 ) - ( this.state.userData.size / 2 ) ) /
this.state.userData.speed ) * 10,
          delay: ( this.state.userData.delay * 1000 ),
          easing: Easing.linear
        }
      ),
      /* TP6 to TP7 (Long) */
      Animated.timing(
        this.translateValue,
        {
          toValue: {
            x: ( ( Dimensions.get('window').width / 2 ) - ( this.state.userData.size / 2 ) ),
            y: ( ( Dimensions.get('window').height / 2 ) - ( this.state.userData.size / 2 ) )
```

```
            },
            duration: ( ( ( ( Dimensions.get('window').height / 2 ) - ( this.state.userData.size / 2 ) ) /
this.state.userData.speed ) * 10 ) * 2,
            delay: ( this.state.userData.delay * 1000 ),
            easing: Easing.linear
          }
        ),
        /* TP7 to TP8 (Short) */
        Animated.timing(
          this.translateValue,
          {
            toValue: {
              x: ( ( Dimensions.get('window').width / 2 ) - ( this.state.userData.size / 2 ) ),
              y: 0
            },
            duration: ( ( ( Dimensions.get('window').height / 2 ) - ( this.state.userData.size / 2 ) ) /
this.state.userData.speed ) * 10,
            delay: ( this.state.userData.delay * 1000 ),
            easing: Easing.linear
          }
        ),
        /* TP8 to Finish point (Short) */
        Animated.timing(
          this.translateValue,
          {
            toValue: {
              x: 0,
              y: 0
            },
            duration: ( ( ( Dimensions.get('window').width / 2 ) - ( this.state.userData.size / 2 ) ) /
this.state.userData.speed ) * 10,
            delay: ( this.state.userData.delay * 1000 ),
            easing: Easing.linear
          }
        )
      ]),
      { iterations: this.state.userData.cycle }
    )
  ]).start( console.log( 'animation finish' ) );
 }
 /*  ------------------------------END animateHPath Function-----------------------  */

//   this function call when screen start
componentDidMount () {
  console.log(Dimensions.get('window').width);
  // var that = this;
  // AppState.addEventListener('change', this._handleAppStateChange);
this.setCurrentdateTime();


   db.transaction(function(txn) {
     txn.executeSql(
       "SELECT * FROM patient ORDER BY id DESC LIMIT 1;",
       [],
       (tx, res) => {
         for (let i = 0; i < res.rows.length; ++i) {
```

```
          console.log( res.rows.item(i));
          this.setState({
            userInfo: res.rows.item(i),
          });
        }
         this.setState({
           patientName: this.state.userInfo['patientName'],
           datetime: this.state.userInfo['datetime']
        })
      }
    );
  }.bind(this));


  // console.log()
  // console.log(this.state.datetime)

  // Start timer

const interval = setInterval(() => {
    if (this.state.timer == 0) {
      clearInterval(interval)
      this.onTimerComplete()

    }else{
      this.setState(state => ({...state,timer:this.state.timer-1}))
    }

    switch(this.state.timer){

      case 1:
        this.startSpy() //when timer is on 1 start Screen Recoring
        console.log('SR recording Start')
          break
          case 0:
        this.startRecording()//when timer is on 0 or end Front facing starting
        // this.callToast()
        // console.log('animation start')
          break

    }
    console.log(this.state.timer)

  }, 1000);

  db.transaction(function(txn) {
    txn.executeSql(
      "SELECT * FROM settings",
      [],
      (tx, res) => {
        for (let i = 0; i < res.rows.length; ++i) {
          console.log( res.rows.item(i));
          this.setState({
            userData: res.rows.item(i),
          });
        }
```

```
      }
    );
  }.bind(this));


  setInterval( () => {
    this.setState({
      timeStamp : new Date().toLocaleString()
    })
  },1000)

  this.intervalID = setInterval(
    () => this.tick(),
    1000
  );
}


  tick() {
    this.setState({
      time: new Date().toLocaleString()
    });
  }



  stopapp(){
    setTimeout(() => {
      Alert.alert('Exam Complete','Video saved Connect computer to download',[
        {
          'text':'Exit',
          onPress:()=>{
            this.props.navigation.dispatch(StackActions.reset({
              index: 0,
              actions: [
                NavigationActions.navigate({ routeName: 'Home' })
              ],
            }))
          }
        }
      ])
    }, 1000);
  }

  render() {
    // console.log(this.state.userData.size);
    const translateTransform = this.translateValue.getTranslateTransform();
    // console.log(movingMargin);
    const {timer} = this.state

    return <React.Fragment>
          <RNCamera
            style={styles.camera}
            ref={ref => {
            this.camera = ref;
```

```
        }}

        type={RNCamera.Constants.Type.front}
        flashMode={RNCamera.Constants.FlashMode.on}
        onGoogleVisionBarcodesDetected={({ barcodes }) => {
          console.log(barcodes);
        }}
      />

      {
        timer == 0
        ?

          <View style={{ flex: 1 }}>
                <Animated.View style={{
                  backgroundColor:this.state.userData.color,
                  width:this.state.userData.size,
                  height:this.state.userData.size,
                  borderRadius:50,
                  top:Dimensions.get('window').height/2-this.state.userData.size/2,
                  left:Dimensions.get('window').width/2-this.state.userData.size/2,
                  position: "absolute",
                  transform: translateTransform
                  }} />
               </View>



        :
        <View style={{flex:1,alignItems:'center',justifyContent:'center'}}>
          <Text style={{fontSize:100,color:'#000'}}>
        {this.state.timer}
       </Text>
      </View>
        }
   </React.Fragment>
  }
}
```

IRB Exemption Protocol Approval

From: Karen H Larwin <khlarwin@ysu.edu>
Sent: Tuesday, December 15, 2020 11:18 AM
To: Dr. Abdu Arslanyilmaz <aarslanyilmaz@ysu.edu>; joey2250@gmail.com <joey2250@gmail.com>
Cc: ckcoy@ysu.edu <ckcoy@ysu.edu>; Sal Sanders <sasanders@ysu.edu>; Angie J Urmson Jeffries <ajurmsonjeffries@ysu.edu>
Subject: Re: IRB Forms #070-21(ltr)

Dear Investigators,

Your protocol entitled *Utilizing a Mobile Device to Implement a Dual-Recording Eye Exam* has been reviewed and is deemed to meet the criteria of an exempt protocol in which data is collected in the regular educational setting. You are using a single-subject design to test an application that you have developed. While the focus of your research is not the human participant, you are using a human participant to test the app that you developed. Your participant is at least 18 years of age. You will not report any personal information about the participant with the data.

The research project meets the expectations of 45 CFR 46.104(b)(1) and is therefore approved. You may begin the investigation immediately. Please note that it is the responsibility of the principal investigator to report immediately to the YSU IRB any deviations from the protocol and/or any adverse events that occur. Please reference your protocol number 070-21 in all correspondence about the research associated with this protocol.

Good luck on your research and congratulations on your graduation.
Karen

Karen H. Larwin, Ph.D.
Distinguished Professor & YSU IRB Chair
Beeghly College of Liberal Arts, Social Sciences, & Education
Youngstown State University
One University Plaza
Youngstown, Ohio 44555-0001