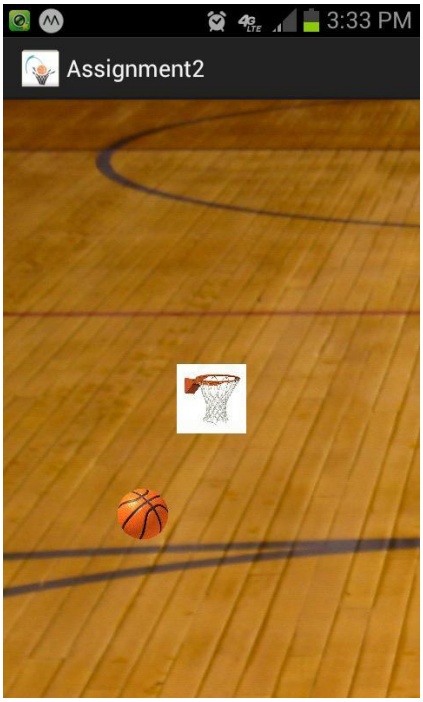
Assignment #3
Mobile Device Development
Accelerometer Basketball App
Due: 10/22/2020@11:59pm

In this assignment we use the acceleration values from accelerometer sensor to move a basketball on the screen. For this assignment you need to write three classes. Main activity class; SimulationView class; and Particle class. Some parts of the code are provided here.



Create an Android project

Create an Android project with details as follows:

Application Name: *AccGame*
Project Name: *AccGame*
Package Name: *edu.sjsu.android.accelerometer*
Minimum Required SDK: *API 15*
Target SDK: < *your own device*>
Compile With: <*your own device*>

The Android Manifest file

Here you should add the necessary permission and feature to the manifest.

- Use of wake lock to keep the screen on.
- Edit the <activity> element so the game runs full screen in **portrait** mode. Note if you are using Android Studio 3.6+, when you try to set portrait mode, the editor will show an error (Expecting android:screenOrientation="unspecified"…). You can simply ignore this for now since it won't cause any build failure. Or add xmlns:tools=http://schemas.android.com/tools tools:ignore="LockedOrientationActivity" after package="…"

The Main Activity

In the main activity class, To ensures optimum utilization of system resources, use PowerManager in onCreate() method with *SCREEN_BRIGHT_WAKE_LOCK* value to keep the screen bright while it's on.

```
public class MainActivity extends Activity {
    private static final String TAG = "edu.sjsu.android.accelerometer:MainActivity";
    private PowerManager.WakeLock mWakeLock;
    // The view
    private SimulationView mSimulationView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        PowerManager mPowerManager = (PowerManager) getSystemService(POWER_SERVICE);
        mWakeLock = mPowerManager.newWakeLock(
                                WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON, TAG);
        mSimulationView = new SimulationView(this);

        // Set to the simulation view instead of layout file.
        setContentView(mSimulationView);
    }
```

Also in the MainActivity class complete onResume() onPause() methods by acquiring the wake lock and releasing it in the above mentioned methods accordingly.

```
@Override
    protected void onResume() {
        super.onResume();
        // acquire wakelock
        mWakeLock.acquire();
        // start simulation to register the listener
        mSimulationView.startSimulation();
    }

    @Override
    protected void onPause() {
        super.onPause();
        // Release wakelock
        mWakeLock.release();
        // stop simulation to unregister the listener
        mSimulationView.stopSimulation();
    }
```

The Simulation View

Create a new class named *SimulationView*. Create a custom view by extending *View* and make it implement the *SensorEventListener* interface.

- Define *SensorManager*, *Sensor*, and *Display*
- Define the following variables for the graphical user interface:

```
private Bitmap mField;
private Bitmap mBasket;
private Bitmap mBitmap;
private static final int BALL_SIZE = 64;
private static final int BASKET_SIZE = 80;

private float mXOrigin;
private float mYOrigin;
private float mHorizontalBound;
private float mVerticalBound;
```

- Add the following code to SimulationView constructor and onSizeChanged() method. (ball, basket and field are drawables. Learn how to add drawables to your app.)

```
public SimulationView(Context context) {
    super(context);
    // Initialize images from drawable
    Bitmap ball = BitmapFactory.decodeResource(getResources(), R.drawable.ball);
    mBitMAP = Bitmap.createScaledBitmap(ball, BALL_SIZE, BALL_SIZE, true);
    Bitmap basket = BitmapFactory.decodeResource(getResources(), R.drawable.basket);
    mBasket = Bitmap.createScaledBitmap(basket, BASKET_SIZE, BASKET_SIZE, true);
    Options opts = new Options();
    opts.inPreferredConfig = Bitmap.Config.RGB_565;
    mField = BitmapFactory.decodeResource(getResources(), R.drawable.field, opts);
```

The implementation till now just creates the basic resources required for drawing on the screen. Next implement the callbacks for *SensorEventListener*.

Reading Accelerometer data

Implement onSensorChanged() method of *SensorEventListener* in the SimulationView class. To do so, get the acceleration values along the three axis from *event.values* array and timestamp of the data from *event.timestamp*.

The sensor values are relative to the natural orientation of the device. But the display orientation may differ from the device orientation. e.g. If you rotate your phone, the system will reorient the display into portrait or landscape mode. Note that we had specified portrait mode for the MainActivity in the manifest so system won't reorient the display. However, for some devices e.g. tablets the natural orientation of the device could be landscape mode. So, its important to put logic as we have done to interpret the sensor data properly. To do so use *getRotation()* method from Display object.

For example:

```
if Surface.ROTATION_0 then X = event.values[0];
                           Y = event.values[1];
Else if Surface.ROTATION_90 then X = -event.values[1];
                           Y = event.values[0];
```

Add the following lines of code in the SimulationView constructor to initialize display. Also add appropriate code to initialize sensor as you learned in the lab assignment.

```
WindowManager mWindowManager = (WindowManager)
                        context.getSystemService(Context.WINDOW_SERVICE);
mDisplay = mWindowManager.getDefaultDisplay();
```

Now we are all set to start listening for sensor data but before that we need to register the listener. Add these two methods named *startSimulation()* and *stopSimulation()* in the SimulationView class to *registerListener* and *unregisterListener*

Modify the MainActivity class to invoke these methods from the so that the simulation is aligned with the lifecycle of the activity. (Hint: invoke *startSimulation* in *OnResume()* method and *stopSimulation* in *onPause()*)

Drawing on the screen

Use the acceleration values to calculate displacement of the particle along the X and Y axis. Additionally, add logic to create a bounce effect when it collides with the boundary.

```
public class Particle {
    private static final float COR = 0.7f;
    public float mPosX;
    public float mPosY;
    private float mVelX;
    private float mVelY;

    public void updatePosition(float sx, float sy, float sz, long timestamp) {
        float dt = (System.nanoTime() - timestamp) / 1000000000.0f;
        mVelX += -sx * dt;
        mVelY += -sy * dt;
        mPosX += mVelX * dt;
        mPosY += mVelY * dt;
    }
    public void resolveCollisionWithBounds(float mHorizontalBound, float mVerticalBound) {
        if (mPosX > mHorizontalBound) {
            mPosX = mHorizontalBound;
            mVelX = -mVelX * COR;
        } else if (mPosX < -mHorizontalBound) {
            mPosX = -mHorizontalBound;
            mVelX = -mVelX * COR;
        }
        if (mPosY > mVerticalBound) {
            mPosY = mVerticalBound;
            mVelY = -mVelY * COR;
        } else if (mPosY < -mVerticalBound) {
            mPosY = -mVerticalBound;
            mVelY = -mVelY * COR;
        }
    }
}
```

Finally, implement *onDraw()* method of the custom view to draw a basketball and filed on the screen. Add the following code to SimulationView class.

And invoke invalidate() so that the view is redrawn repeatedly.

```
@Override
protected void onDraw(Canvas canvas) {
    super.onDraw(canvas);

    canvas.drawBitmap(mField, 0, 0, null);
    canvas.drawBitmap(mBasket, mXOrigin - BASKET_SIZE / 2, mYOrigin - BASKET_SIZE / 2, null);

    mBall.updatePosition(mSensorX, mSensorY, mSensorZ, mSensorTimeStamp);
    mBall.resolveCollisionWithBounds(mHorizontalBound, mVerticalBound);

    canvas.drawBitmap(mBitMAP,
                (mXOrigin - BALL_SIZE / 2) + mBall.mPosX,
                (mYOrigin - BALL_SIZE / 2) - mBall.mPosY, null);

    invalidate();
}
```

Submission

1. Push your project directory along with the source to remote bitbucket repository by the due date.
2. Invite and share your project repository the Grader (yan.chen01@sjsu.edu) and Instructor (ramin.moazeni@sjsu.edu).
3. Submit a Readme.pdf to Canvas including your name, repository access link, instructions to run your program (if any), snapshot of your running program
4. Your project directory will be graded according to the state your project directory was in at due time when fetched.