Bouncing and Actor Class Directions
Part 2: Drawing Graphics, Adding Touch Event Data, and Adding Accelerometer

Description:

Now that we have an App base where we can create Actors that move, bounce, and draw shapes on the screen, we need to add features that allow us to draw graphics, and respond to user events such as touch gestures and Acceleration (or tilting) the mobile device. These directions will walk you through adding the additional code to enable these features. Please read the descriptions carefully and make sure to type in the comments so you can refer back to this code later.

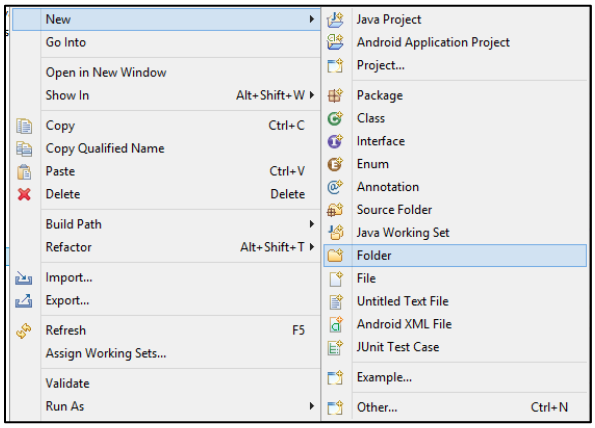
Section 1: Giving your Actor the ability to draw Graphics.

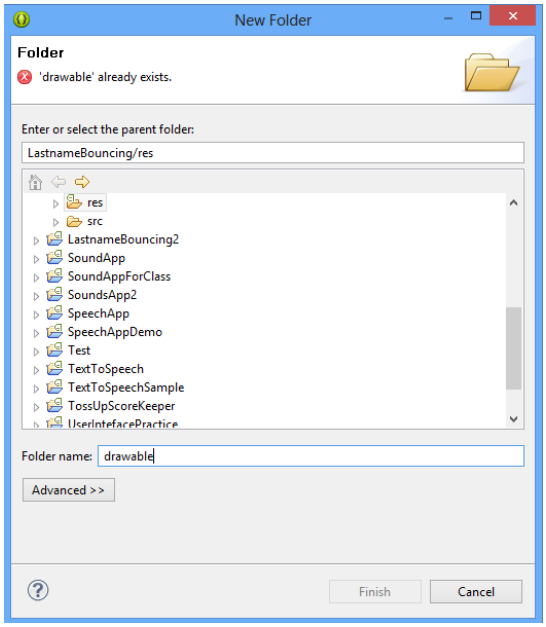
We want the Actor to be able to draw images in addition to shapes. The outline of this procedure is:

1. Create a drawable folder in the res area.
2. Add a 'Context' field to the Actor Class. This allows us to access the res media such as sounds or images.
3. Add 'costume' and 'graphic' fields to store image Resource Data and the Bitmap of the image.
4. Edit the constructor to include the 'Context' field.
5. Add modifier functions to add costume data.
6. Add accessor functions to access the Bitmap information
7. Create a function to allow the Actor to draw the bitmap onto the screen.

Process:

1. Expand the res folder in the App package and right click on res. Select New ->Folder and name the folder 'drawable.'





2. Find a small image to place in the drawable folder. I am going to use the tardis.png available at: http://nebomusic.net/androidlessons/gameimages/tardis.png. You can use any image smaller than 128 by 128 pixels.
    a. Save the image to the desktop.
    b. Make sure the image name only has lower case letters without spaces.
    c. Copy the image into the /res/drawable folder.

Other images are available at: http://nebomusic.net/androidlessons/gameimages/

3. Go to the Actor class and find the Fields (towards to top under the class declaration). Add the highlighted fields (Import where needed)

```
 9
10 public class Actor {
11     // Fields for Actor
12     private Point p; // for Location
13     private int c; // for color
14     private int s; // for size
15     private int dx; // for change in x speed
16     private int dy; // for change in y speed
17     private Paint paint; // Paint object to hold painter
18
19     // Context so Actor can get at graphic resources
20     private Context aContext;
21
22     // Integer for Drawable resource
23     private int costume;
24
25     // Stores Graphic for Costume
26     private BitmapDrawable graphic;
27
```

4. Modify the Constructor of the Actor class to include the Context data. (Note, this will bring up errors in the AnimationView class where we created Actors. We will need to modify this code later.)

```
27
28      // Constructor
29      public Actor(Context context, int x, int y, int col, int size) {
30          // initialize values
31          p = new Point(x, y); // set the x and y position
32          c = col; // set the color
33          s = size;  // sets the size
34          paint = new Paint(); // creates Paint object
35          paint.setColor(c); // sets Paint color
36          dx = 0; // sets x speed to 0
37          dy = 0; // sets y speed to 0
38
39          // Set the Context
40          aContext = context;
41
42      } // end Constructor
43
```

5. Go to the bottom of the Actor class to add additional functions. Write the modifier to set the Costume resource for the Actor.

```
116
117     // Setters and Getters for Graphics
118
119     public void setCostume(int cost) {
120         costume = cost;
121         graphic = (BitmapDrawable)aContext.getResources().getDrawable(costume);
122     }
123
```

6. Write the return function to get the Bitmap of the costume:

```
123
124     public Bitmap getBitmap() {
125         return graphic.getBitmap();
126     }
127
```

7. Write the draw function to allow the Actor to draw itself on the canvas.

```
127
128     public void draw(Canvas c) {
129         c.drawBitmap(getBitmap(), p.x, p.y, paint);
130     }
131
```

8. We are finished with the Actor Class. Now we will move to the AnimationView class and make the modifications to test our Actor's drawing ability.

9. In the Animation View class we initialized several Actors to test the Apps ability to draw shapes and bounce. Now that we have added a Context object to the Actor's constructor, we need to modify the initialization of the Actors in AnimationView. Go to the constructor of AnimationView and find where we initialized the Actors. Add the word 'context' to the Actor initializers to fix the error.

Before (with errors):

```
27
28    public AnimationView(Context context, AttributeSet attrs) {
29        super(context, attrs);
30        // TODO Auto-generated constructor stub
31
32        // Initialize the Actors
33        joshua = new Actor(100, 100, Color.RED, 40);
34        rebecca = new Actor(200, 200, Color.BLUE, 50);
35        sandy = new Actor(300, 200, Color.GREEN, 100);
36
```

After: (Add 'context' to Actors) -> Fixes the Errors (Lines 33, 34, and 35 in this example)

```
27
28    public AnimationView(Context context, AttributeSet attrs) {
29        super(context, attrs);
30        // TODO Auto-generated constructor stub
31
32        // Initialize the Actors
33        joshua = new Actor(context, 100, 100, Color.RED, 40);
34        rebecca = new Actor(context, 200, 200, Color.BLUE, 50);
35        sandy = new Actor(context, 300, 200, Color.GREEN, 100);
36
```

10. We are going to create a new Actor that will draw the tardis. We will call this actor 'tardis'. Go to the Fields of the AnimationView class and add the Actor tardis.

```
10
11 public class AnimationView extends View {
12
13     // Create some Actors
14     private Actor joshua;
15     private Actor rebecca;
16     private Actor sandy;
17     private Actor tardis;
18
```

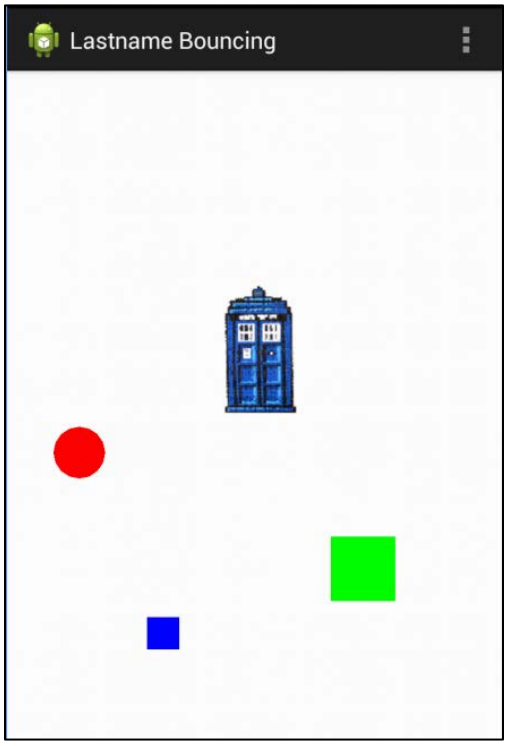
11. In the Constructor of AnimationView, add the initializer for the tardis Actor. Note that the R.drawable.tardis represents the tardis image we put in the drawable folder earlier. If you used a different image, your R.drawable.imageName will be different. Make sure it matches your image.

```
27
28      public AnimationView(Context context, AttributeSet attrs) {
29          super(context, attrs);
30          // TODO Auto-generated constructor stub
31
32          // Initialize the Actors
33          joshua = new Actor(context, 100, 100, Color.RED, 40);
34          rebecca = new Actor(context, 200, 200, Color.BLUE, 50);
35          sandy = new Actor(context, 300, 200, Color.GREEN, 100);
36
37          // Initialize the tardis
38          tardis = new Actor(context, 300, 300, Color.BLUE, 50);
39          tardis.setCostume(R.drawable.tardis);
40
```

12. Go to the onDraw() function within the AnimationView class. Add the command for the tardis to draw itself to the screen.

```
56      // Create the onDraw Method - in all View Classes
57      public void onDraw(Canvas c) {
58
59          // Actors Move
60          joshua.move();
61          rebecca.move();
62          joshua.bounce(c);
63          rebecca.bounce(c);
64          sandy.move();
65          sandy.bounce(c);
66
67          // have the Actors draw themselves
68          joshua.drawCircle(c);
69          rebecca.drawSquare(c);
70          sandy.drawSquare(c);
71          tardis.draw(c);
72
73          // Call the Runnable for Animation
74          h.postDelayed(r, RATE);
75
76      } // End onDraw
77
```

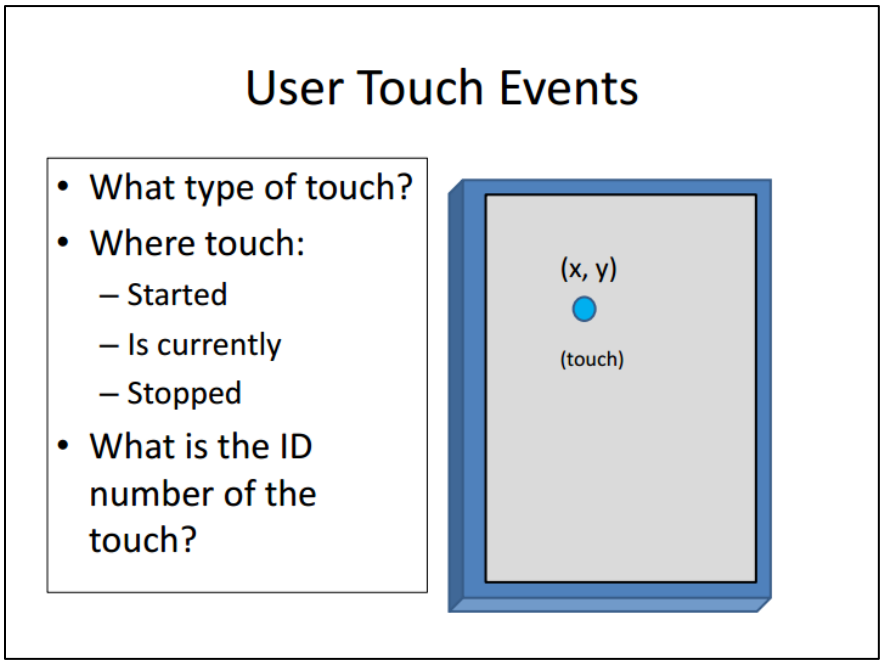
13. Save and run the App. You should see your graphic object (the tardis in this example) in the App Screen.



Section 2: Adding TouchEvent Data to the App.

14. Now that we have an Actor that can draw images, we will move on to the next objective, which is gathering Touch Event data and having Actors that you can control with touch gestures.

15. Description of Touch Data. A slideshow containing information about reading Touch Data can be found at: http://nebomusic.net/androidlessons/TouchEventData.pdf. In summary, we need to know this information about a touch event:



16. The structure of onTouchEvent() is as follows: ***(There are overview slides – do not type anything yet . . .)***

Structure of onTouchEvent() Function

```
public boolean onTouchEvent(MotionEvent event) {

        int action = event.getActionMasked();
        int actionIndex = event.getActionIndex();


        return true; // required
}
```

The object `event` represents the object through which we will gather the touch data.

The integer `action` holds the type of touch event.

The integer `actionIndex` holds the ID of the touch (which touch in multi touch situations)

17. There are several types of touch events based on how the user interacts with the screen:

Android Touch Events

- MotionEvent.ACTION_DOWN
- MotionEvent.ACTION_UP
- MotionEvent.ACTION_MOVE

And many others . . .

18. To read touch event data, use the .getX() or .getY() functions. Examples include:

MotionEvent Data

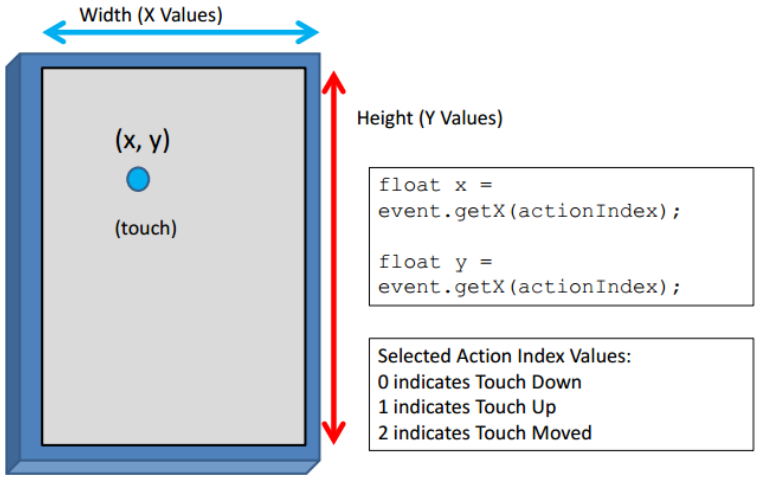
Given a MotionEvent object called `event` and
An object named `actionIndex` for the index of event:

```
event.getX(actionIndex)
// Returns X position of touch at
actionIndex

event.getY(actionIndex)
// Returns Y position of touch at
actionIndex
```

19. MotionEvent data is broken down according to type of touch:

MotionEvent Data




```
float x =
event.getX(actionIndex);

float y =
event.getX(actionIndex);
```

Selected Action Index Values:
0 indicates Touch Down
1 indicates Touch Up
2 indicates Touch Moved

20. We will now write touch event data for the AnimationView. Deploying an onTouchEvent() function is fairly straightforward. Go to AnimationView class and move towards the bottom of the class (after the Runnable r). Each View class has a function onTouchEvent() that can be re-defined.

21. Add the following function to your AnimationView class:

```
95
96      // Function to Grab touch event data
97      public boolean onTouchEvent(MotionEvent event) {
98          // Fetch data from touch event
99          int action = event.getActionMasked(); // get type of action
100         int actionIdex = event.getActionIndex(); // get index of action
101
102
103         return true;
104     } // end onTouchEvent
105
```

22. Now we will use the onTouchEvent data to move one of the Actors to where the user touches the screen. I am going to use the Actor sandy (the large green square). Add these lines (102 and 103) within the onTouchEvent() function:

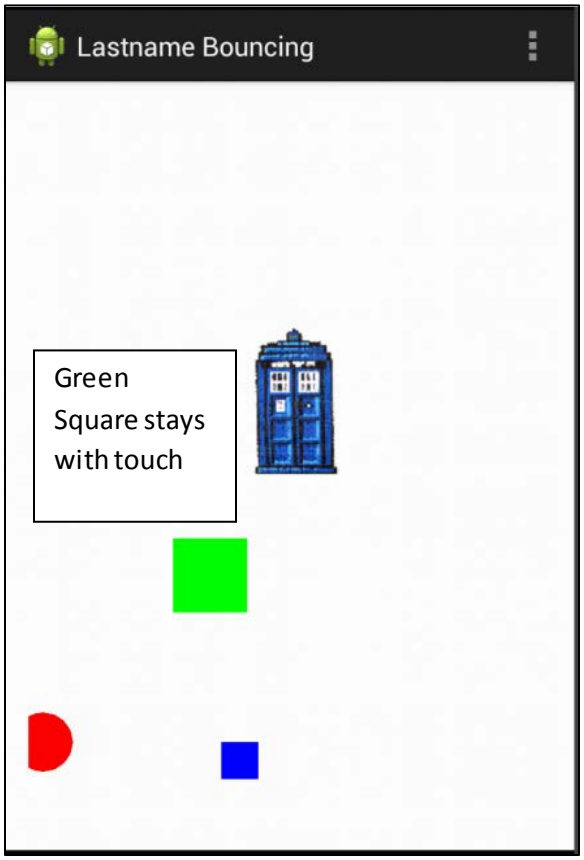
```
95
96      // Function to Grab touch event data
97      public boolean onTouchEvent(MotionEvent event) {
98          // Fetch data from touch event
99          int action = event.getActionMasked(); // get type of action
100         int actionIdex = event.getActionIndex(); // get index of action
101
102         // set Position of Sandy to touch data
103         sandy.goTo((int)event.getX(), (int)event.getY());
104
105         return true;
106     } // end onTouchEvent
107
```

23. Go back to the onDraw() function in AnimationView and make sure sandy does not call sandy.move() or sandy.bounce() (Comment these lines out or delete)

```
55
56      // Create the onDraw Method - in all View Classes
57      public void onDraw(Canvas c) {
58
59          // Actors Move
60          joshua.move();
61          rebecca.move();
62          joshua.bounce(c);
63          rebecca.bounce(c);
64          // sandy.move();
65          // sandy.bounce(c);
66
67          // have the Actors draw themselves
68          joshua.drawCircle(c);
69          rebecca.drawSquare(c);
70          sandy.drawSquare(c);
71          tardis.draw(c);
72
73          // Call the Runnable for Animation
74          h.postDelayed(r, RATE);
75
76      } // End onDraw
77
```

Delete or comment out references to sandy.move() or sandy.bounce() so sandy will stay with the touch event

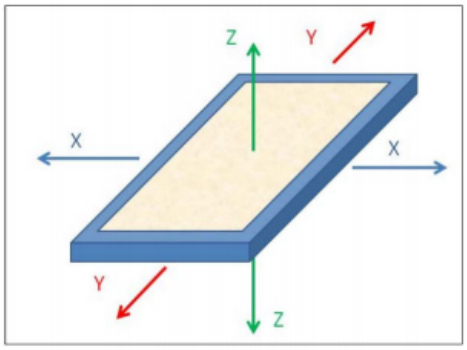
24. Run the App and touch the screen, the green square should stay with your finger touch events.



Section 3: Adding Accelerometer Data to App:

Most mobile 2D games work by tilting the phone to cause characters to move or change position. This phone tilting uses the device accelerometer to measure the change in acceleration in the x, y, an z axis. A description of the Accelerometer can be found at: http://nebomusic.net/androidlessons/Accelerometer.pdf. The steps are outlined as below:

Steps for Accelerometer Setup



1. Declare a field for SensorManager
2. In onCreate() – call the function to enable the listener.
3. Define the Listener as a function.
4. Write the inner class that defines the SensorEventListener and reads / reacts to data.

Of course, this will only work on an actual device. The emulator does not read accelerometer data. The following steps will add the ability for the BouncingApp to read data from the Accelerometer and move objects by tilting. Go ahead and code these in and we will test them with actual devices.

Process:

25. We will need to add some fields to the AnimationView class to hold the x, y, and z acceleration data. Go to the fields of AnimationView and add these fields:

```
10
11 public class AnimationView extends View {
12
13     // Create some Actors
14     private Actor joshua;
15     private Actor rebecca;
16     private Actor sandy;
17     private Actor tardis;
18
19     // values to hold the Accel Data
20     private float ax = 0;
21     private float ay = 0;
22     private float az = 0;
23
24     // Create the Handler for animation
25     private Handler h;
26     private int RATE = 30;  // about 30 Frames a Second
27
```

Note that we set these to 0 so the default value is that the phone is not reading data from the accelerometer.

26. These fields need to be able to be modified by other classes in the App. So we will write some modifier functions for these fields. Go to the bottom of the AnimationView class and write the three modifier functions for ax, ay, and az:

```
107
108     // Modifier fields for acceleration data on x, y, and z
109     public void setAX(float x) {
110         ax = x;
111     }
112
113     public void setAY(float y) {
114         ay = y;
115     }
116
117     public void setAZ(float z) {
118         az = z;
119     }
120
```

27. We will now move to the Main class. In the Main class we establish the sensorManager for the accelerometer, read the tilt data from the device, and call the AnimationView modifier functions to pass the tilt data to the animation.

28. Go to the Main class and add a field for a SensorManager:

```
11
12 public class Main extends Activity {
13
14     // Field for Animation View
15     private AnimationView animationView;
16
17     // Sensor Manager for Accelerometer
18     private SensorManager sensorManager;
19
```

29. Go to the onCreate() in the Main class and write the function to enable Listening for the Accelerometer. Do not worry now that the code shows an error. We will define this later in the class.

```
19
20     @Override
21     protected void onCreate(Bundle savedInstanceState) {
22         super.onCreate(savedInstanceState);
23         setContentView(R.layout.activity_main);
24
25         // Initialize the Animation View
26         animationView = (AnimationView)findViewById(R.id.animationView);
27
28         // Enable the listener - We will write this later in the class
29         enableAccelerometerListening();
30
31     } // end onCreate
32
```

30. We will now write the function to enableAccelerometerListening. Again, some errors will show. We will define the sensorEventListener later in the code. (Be careful of the placement of commas and parenthesis. Zoom in if you need to read the details.)

```
39
40     // Define the enableAccelerometerListening function
41     private void enableAccelerometerListening() {
42         // Initialize the Sensor Manager
43         sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
44         sensorManager.registerListener(sensorEventListener,
45                 sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER),
46                 SensorManager.SENSOR_DELAY_NORMAL);
47     }
48
```

31. Write the sensorEventListener function. This is an inner class function similar to the SeekBarListener. Use the import and then add unimplemented methods helpers to complete the function.

```
48
49      // Define the sensorEventListener
50      private SensorEventListener sensorEventListener =
51              new SensorEventListener() {
52
53                  @Override
54                  public void onAccuracyChanged(Sensor arg0, int arg1) {
55                      // TODO Auto-generated method stub
56                      // Not USed
57                  }
58
59                  @Override
60                  public void onSensorChanged(SensorEvent event) {
61                      // TODO Auto-generated method stub
62
63                  }
64
65      }; // end SensorEventListener
66
```

32. The onSensorChanged() function is what we are interested in. This function is called whenever the tilt of the device is changed. Here we will read the sensor data and then pass it to the AnimationView class. Add the lines of code to read the event data:

```
57
58                  @Override
59                  public void onSensorChanged(SensorEvent event) {
60                      // Gather the x, y, and z values from the accellerometer
61                      float x = event.values[0];
62                      float y = event.values[1];
63                      float z = event.values[2];
64
```

33. We will now pass these values to the AnimationView class. Add the calls to the modifier functions: (Note how we write animationView.setAX(x);)

```
57
58                  @Override
59                  public void onSensorChanged(SensorEvent event) {
60                      // Gather the x, y, and z values from the accellerometer
61                      float x = event.values[0];
62                      float y = event.values[1];
63                      float z = event.values[2];
64
65                      // pass the values to the AnimationView object
66                      animationView.setAX(x);
67                      animationView.setAY(y);
68                      animationView.setAZ(z);
69
70                  }
71
72      }; // end SensorEventListener
```

34. Now that the x, y, and z tilt data is being passed to AnimationView, we can use these values to modify the DX and DY of the actors. Go to the AnimationView class and find the onDraw() function. Write these lines of code to have the tardis Actor move with the tilt of the device.

```
55
56      // Create the onDraw Method - in all View Classes
57      public void onDraw(Canvas c) {
58
59          // Actors Move
60          joshua.move();
61          rebecca.move();
62          joshua.bounce(c);
63          rebecca.bounce(c);
64          // sandy.move();
65          // sandy.bounce(c);
66
67          // Read Accel data and move the tardis
68          tardis.changeDX(ax * -1); // read the x acceleration
69          tardis.changeDY(ay);      // read the y acceleration
70          tardis.move();            // Move
71          tardis.bounce(c);         // Bounce
72
73          // have the Actors draw themselves
74          joshua.drawCircle(c);
75          rebecca.drawSquare(c);
76          sandy.drawSquare(c);
77          tardis.draw(c);
78
```

35. Save and run on a real Android device. You should see the tardis moving with the tilt of the device. (And hopefully bounce off the edges!)