ADDING ARRAYS AND LOOPS TO THE GAME

This guide goes through the steps given in the 'Adding arrays and loops to the game' video in week 4

This document explains changes made to the code in the video. In order to make changes to the code, you have to make sure you have the correct codebase for Week 4. Please make sure you have copied v4.java file's content into your project - this was explained in the 'Welcome to week 4' video, and you can find a PDF guide at the bottom of the page.
Note: The code changes are made to 'TheGame.java' file.

Defining SadBall

The first changes you will be made will be to the section of code under:

```
public class TheGame extends GameThread{
```

to define the data needed for the sad balls. Add below statements to the code:

```
private Bitmap mSadBall;

private float[] mSadBallX = {-100, -100, -100};
private float[] mSadBallY = new float[3];
```

Notice that the same Bitmap mSadBall is used to print all three sad faces in the screen.

Storing Coordinates of 3 SadBalls

Coordinates of each SadBall (or sad face) is stored in arrays. Altogether you need 6 coordinates: X and Y coordinates for SadBall 1; X and Y coordinates for SadBall 2 and finally X and Y coordinates for SadBall 3.

Two arrays are used to store the coordinates of these balls. The array mSadBallX is used to hold the three X coordinates while mSadBallY is used to hold the three Y coordinates.

```
private float[] mSadBallX = {-100,-100,-100};
private float[] mSadBallY = new float[3];
```

mSadBallX and mSadBallY are two float arrays each containing 3 elements.

NOTE : The two different ways of creating and initializing arrays that was

introduced in the step 'Using Arrays' is shown in practice here.

You could also have written it as:

```
private float[] mSadBallX = new float[3];
```

...and later assign values to mSadBallX's elements as we have seen in the tutorial on arrays. We have used different ways here so that we can demonstrate both to you.

Creating a drawable SadBall image

Creating the sad ball is inside TheGame method, in code block:

```
public TheGame(GameView gameView) {
```

Creating the sad ball is done with this code:

```
mSadBall = BitmapFactory.decodeResource
      (gameView.getContext().getResources(),
            R.drawable.sad_ball);
```

Assign coordinates for SadBall Locations

The next section you need to change is the setupBeginning method. Here you give the coordinates of each sad ball to be drawn on the screen.

```
//Place all SadBalls forming a pyramid underneath the SmileyBall
mSadBallX[0] = mCanvasWidth / 3;
mSadBallY[0] = mCanvasHeight / 3;

mSadBallX[1] = mCanvasWidth - mCanvasWidth / 3;
mSadBallY[1] = mCanvasHeight / 3;

mSadBallX[2] = mCanvasWidth / 2;
mSadBallY[2] = mCanvasHeight / 5;
```

If you have time you could change these locations and see where they get drawn.

Drawing the Three SadBalls

This is done inside method doDraw. That is inside this code block

```
protected void doDraw(Canvas canvas) {
```

We use a construct we learnt this week: a FOR loop to draw the three sad balls.

```
//Loop through all SadBall
for(int i = 0; i < mSadBallX.length; i++) {
  //Draw SadBall in position i
  canvas.drawBitmap(mSadBall, mSadBallX[i] - mSadBall.getWidth()/2,
mSadBallY[i]     - mSadBall.getHeight()/2, null);
}
```

Notice the use of array indices to access each SadBall's X and Y coordinates.

```
//Loop through all SadBall
for(int i = 0; i < mSadBallX.length; i++) {
    //Draw SadBall in position i
    canvas.drawBitmap(mSadBall, mSadBallX[i] - mSadBall.getWidth() / 2,
                      mSadBallY[i] - mSadBall.getHeight() / 2, null);
}
```

Figure 1: Using indices to access each SadBall's X and Y coordinates

Notice the use of the built-in **length** property of an array to find the length.

Changes to updateGame

In **updateGame** you will be detecting collisions, and you need to check collisions for each sad ball. Again we will use a loop to accomplish this task.

```
//Loop through all SadBalls
for(int i = 0; i < mSadBallX.length; i++) {
//Perform collisions(if necessary) between SadBall in position i and little ball

      //Get actual distance (without square root - remember?)
      //between the mBall and the ball being checked
      //here the variable distanceBetweenBallAndPaddle is reused
      //if you want to add a different say distanceBetweenBallAndSadBall you can
      distanceBetweenBallAndPaddle = (mSadBallX[i] - mBallX) * (mSadBallX[i]
- mBallX) + (mSadBallY[i] - mBallY) *(mSadBallY[i] - mBallY);

      //Check if the actual distance is lower than the allowed => collision
      if(mMinDistanceBetweenBallAndPaddle >= distanceBetweenBallAndPaddle) {

            //Get the present velocity (this should also be the velocity
            //going away after the collision)
            float speedOfBall = (float) Math.sqrt(mBallSpeedX*mBallSpeedX
+ mBallSpeedY*mBallSpeedY);

            //Change the direction of the ball
            mBallSpeedX = mBallX - mSadBallX[i];
            mBallSpeedY = mBallY - mSadBallY[i];

            //Get the velocity after the collision
            float newSpeedOfBall = (float) Math.sqrt(mBallSpeedX*mBallSpeedX +
mBallSpeedY*mBallSpeedY);

            //using the fraction between the original speed and
            //present speed to calculate the needed
            //speeds in X and Y to get the original velocity
            //but with the new angle.
            mBallSpeedX = mBallSpeedX * speedOfBall / newSpeedOfBall;
            mBallSpeedY = mBallSpeedY * speedOfBall / newSpeedOfBall;
      }
}
```

NOTE: we were able to reuse the value calculated for mMinDistanceBetweenBallAndPaddle because our paddle and the sad balls had the same radius. If you change the images to a different sized sad face this minimum distance will have to be re calculated.

Also notice the use of the array length to loop through in the for loop.

```
//Loop through all SadBalls
for(int i = 0; i < mSadBallX.length; i++) {
```

You can play with the code by adding or removing items say obstacle. You can explore the code. Don't worry about breaking the code, we will provide a clean code at the beginning of next week for everyone to start with.

If you want to challenge yourself even more look at having boxes instead of balls. Collision detection with boxes or blocks is much harder. But see what you can do, explore and learn.

Additional Resources:

To find out more about Java arrays refer to the official Java tutorial from Oracle
http://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html