Lab 7: Using a SurfaceView and Implementing Runnable

The View introduced last week is insufficient for action type games. This week will introduce SurfaceView, a special sub-class of View that is suitable for this. Basic use of a SurfaceView is similar to using a View, but to make a SurfaceView really useful we need to set up a separate thread by implementing runnable. This will give us the standard structure we need for games.

There is an overhead in doing all this in terms of the amount of code that is introduced. Most of the code we will take "as read", that is it needs to be there but we do not need to modify it. The important thing is to recognise which bits of code need to be changed for our game purposes.

Using a SurfaceView

Start a new project with the main activity called GameActivity. This file can remain the same as that used last week and is reproduced below:

```
import android.app.Activity;
import android.os.Bundle;

public class GameActivity extends Activity {
	GameView GV;

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		GV = new GameView(this);
		setContentView(GV);
	}
}
```

Create the new java class (remember to select the package folder with your java files in it and right click) and call it GameView. Include the following code. There should be one error that you need to fix by pasting the ball.png image into the drawable folder.

```
import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Canvas;
import android.graphics.Color;
import android.view.SurfaceHolder;
import android.view.SurfaceView;

public class GameView extends SurfaceView implements Runnable {
	// variables needed to implement runnable
	private SurfaceHolder holder;
	Thread thread = null;
	volatile boolean running = false;
	static final long FPS = 10;
```

```
	// variables needed for game
	private Bitmap ball1;
	private int x = 0;

	 public GameView(Context context) {
		super(context);
		thread = new Thread(this);
		holder = getHolder();
		holder.addCallback(new SurfaceHolder.Callback() {

			@Override
			public void surfaceDestroyed(SurfaceHolder holder) {
			boolean retry = true;
			running = false;
			while (retry) {
				try {
					thread.join();
					retry = false;
				}
				catch (InterruptedException e) {}
			}
		}

		@Override
		public void surfaceCreated(SurfaceHolder holder) {
			running = true;
			thread.start();
		}

		@Override
		public void surfaceChanged(SurfaceHolder holder, int format, int width, int height) {}
	});
	ball1 = BitmapFactory.decodeResource(getResources(), R.drawable.ball);
}

@Override
public void run() {
	long ticksPS = 1000 / FPS;
	long startTime;
	long sleepTime;

	while (running) {
		Canvas c = null;
		startTime = System.currentTimeMillis();
		try {
			c = getHolder().lockCanvas();
			synchronized (getHolder()) {
			update();
			onDraw(c);
		}
	}
```

```
        finally {
                if (c != null) {
                getHolder().unlockCanvasAndPost(c);
        }
}
sleepTime = ticksPS-(System.currentTimeMillis() - startTime);
        try {
                if (sleepTime > 0)
                        thread.sleep(sleepTime);
                else
                        thread.sleep(10);
                }
                catch (Exception e) {}
        }
}
private void update(){
        x++;
}

@Override
protected void onDraw(Canvas canvas) {
        canvas.drawColor(Color.WHITE);
        canvas.drawBitmap(ball1, x, 10, null);
        }
}
```

There is quite a lot of code here – however most is standard and will not need to be edited. The parts you might have to change are highlighted in yellow. There are a number of points to note to use this code and identify where you will need to make changes.

Firstly note that GameView now extends SurfaceView. Identify the constructor method (public GameView(Context context)) and the methods within it.

The SurfaceView implements Runnable – this gives us a big gain in introducing a separate thread as discussed in lectures. Essentially whilst the Boolean *running* is true the method run() will be repeated at a regular rate. The two method calls you need to identify in the heart of run() are to update() and onDraw(). The starting update() method is about as simple as it can be.

The onDraw() method is the important method which does the drawing on the screen. The example has a few sample lines of what it can do. Notice that, as with the View last week, the bitmap has to be declared and connected to the file in the drawable folder.

So, put simply, run keeps calling onDraw() and all the animation or gameplay exists in the onDraw method. The loop is set to run FPS times a second. You can change this if you wish.

[Note: there may be a red error message in GameView underlining onDraw() – this is a rare occasion when you can ignore this and the app will run].
What is the significance of the following line in onDraw()?

```
        canvas.drawColor(Color.WHITE);
```

Now do some modifications to onDraw() to change the movement of the ball.

Exercises:
Speed up the movement of the ball. You can do this by increasing the step each time onDraw is called, for example x = x + 5; or you might want to declare a variable xSpeed in the GameView class (private int xSpeed = 5;) and use x = x + xSpeed;

Get the ball to bounce of the right side of the screeen – the trick here is to do a test on the ball's position and change the sign of xSpeed when required (e.g. xSpeed = - xSpeed).
Get the ball to bounce backwards and forwards.
Get the ball to move down the centre of the screen and reappear at the top when it moves of the screen.
Get the ball to move diagonally down the screen, by including the y coordinate, and bouncing of all the edges of the screen. You can look up some algorithms for the code to implement for this.

Adding Touch Interactions
We can add interactions by adding an onTouchEvent() method to the GameView. Try the following method:

```
@Override
public boolean onTouchEvent(MotionEvent event) {
        x = (int) event.getX();
        y = (int) event.getY();
        return super.onTouchEvent(event);
}
```

Watch what happens when you touch the screen at any point. Can you explain this? Remember that x and y are class variables. Hardly great game play at the moment but a significant step forward in that the action now continues in-between interaction unlike the View based game. You should now be able to add some game elements such as a score and some text output, or a "game over" graphic that pops up.

Starting a New Activity from a View
Previously we started a new activity from within an activity – to start an activity from a view we need to refer back to the activity (the context) that created the view.

```
Intent intent = new Intent(getContext(),activityToStart.class);
getContext().startActivity(intent);
```

An easy way to test this is to create a new empty activity and use the above technique in the onTouch() method. In your work later you are going to want to be able to navigate to a new activity to quit a game or when a certain score has been achieved.

Aside: you are able to call other methods in the activity from the view using the following lines of code:

```
((GameActivity) getContext()).whatevermethod();
```

Sprites

The starting point of this part of work is the finished lab above – the GameActivity and GameView files are available on moodle just so everyone is at the same starting point. You will need to have a png graphic to display, such as ball.png.

You will create a sprite class for displaying a sprite on the screen. The code that moves and displays the sprite will be moved from GameView into the new sprite class. In our GameView we will call the constructor of the Sprite class sending it the GameView we are in and a bitmap which is the graphic which will represent the sprite on the screen.

We will progress through this in a number of steps starting from creating the sprite class and ending by displaying multiple randomly moving sprites.

Create a new empty project with the code for GameActivity and GameView. Select the package folder in the project and create a new class called Sprite. Copy in the code below.

```
import android.graphics.Bitmap;
import android.graphics.Canvas;

public class Sprite {
	private int x = 0;
	private GameView gameView;
	private Bitmap bmp;

	public Sprite(GameView gameView, Bitmap bmp) {
		this.gameView=gameView;
		this.bmp=bmp;
	}

	public void update() {
		x++;
	}

	public void onDraw(Canvas canvas) {
		canvas.drawBitmap(bmp, x , 10, null);
	}
}
```

Study this for a moment and identify the features mentioned in the lecture. Identify the constructor method. Note that the Sprite class has its own onDraw() method and an update() method.

In our GameView we need to declare a sprite and create it, after the line in which the ball image has been obtained:

```
	private Sprite sprite;
	sprite = new Sprite(this, ball);
```

To move the new sprite you now call methods within the sprite class rather than telling it how to move directly. So in the update() method of GameView make a call to sprite.update(); The sprite class also knows how to draw itself so in the onDraw() method in GameView make a call sprite.onDraw(canvas);

If you run this in the emulator you should see two balls moving on the screen – the one we started with and the new one within the sprite class. Check you can tell which is which.
To change the properties or motion of the sprite we now can do this by editing the sprite class. Use the sprite class below and check that you understand the output.

```
import android.graphics.Bitmap;
import android.graphics.Canvas;
import java.util.Random;

public class Sprite {
	private GameView gameView;
	private Bitmap bmp;

	private int x = 0;
	private int y = 0;
	private int xSpeed, ySpeed;
	private int width;
	private int height;

	public Sprite(GameView gameView, Bitmap bmp) {
		this.gameView=gameView;
		this.bmp=bmp;
		this.width = bmp.getWidth();
		this.height = bmp.getHeight();

		Random rnd = new Random();
		xSpeed = rnd.nextInt(100);
		ySpeed = rnd.nextInt(100);
	}

	public void update() {
		if (x > gameView.getWidth() - width - xSpeed || x + xSpeed < 0) {
		xSpeed = -xSpeed;
	}
	x = x + xSpeed;
	if (y > gameView.getHeight() - height - ySpeed || y + ySpeed < 0) {
		ySpeed = -ySpeed;
	}
	y = y + ySpeed;
	}
	public void onDraw(Canvas canvas) {
		canvas.drawBitmap(bmp, x , y, null);
	}
}
```

You are now in a position to try some game like interactions.

Appendix: Adding simple sounds

In the project folder select the res folder and create a new directory (by right clicking) which must be called raw. Paste a sound file into this folder ( ). In your class file declare a media player to get the sound to play:

```
MediaPlayer myPlayer;
```

In onCreate (or the constructor for a view) link the player with the sound in res/raw:

```
player = MediaPlayer.create(this, R.raw.explosion);
player.start();
```