**Android ja grafiikka**

**Harjoitustyö ja -seminaari**

Vesa-Tapani Vertainen

K5680

Android Application Development

Marraskuu 2017

Teknologia/ICT

Insinööri (AMK), tieto- ja viestintätekniikka

[1 Johdanto 1](#_Toc498285711)

[2 View, SurfaceView ja TextureView 1](#_Toc498285712)

[3 Canvas ja Paint 2](#_Toc498285713)

[4 Canvasiin piirtäminen 4](#_Toc498285714)

[5 Alpha ja XferMode 6](#_Toc498285715)

[6 Peli-osion piirtotoiminnot 8](#_Toc498285716)

[6.1 Rock-metodin piirtotoiminnot 10](#_Toc498285717)

[6.2 Jamppa-metodin piirtotoiminnot 11](#_Toc498285718)

[7 Pelin ohjaus 12](#_Toc498285719)

[Lähteet 13](#_Toc498285720)

# Johdanto

Tässä harjoitustyössä tutkin Java-kielen grafiikkamahdollisuuksia Android-ympäristössä. Keskityin tutkimaan lähinnä juuri Javan grafiikkakomentoja, ja jätin pois suoraan XML-kielellä toteutettavat grafiikat. Harjoitustyön taustalla oli peli-idea ”Jamppa maalla”, jota aloin toteuttaa samalla grafiikkatoimintoja tutkien.

# View, SurfaceView ja TextureView

View on käyttöliittymän peruskomponentti, joka varaa ruudulta nelikulmaisen alueen ja on vastuussa piirtotoiminnoista ja tapahtumista. View-komponentteja voi lisätä koodista käsin tai XML-layouteista. (View. N.d.)

View toimii hyvin ”piirtoalustana”, jos sovellus ei vaadi suurta prosessointia tai nopeaa ruudunpäivitystä. Tällöin voidaan luoda View ja piirtää canvasiin View.onDraw(Canvas) -takaisinkutsua käyttäen, mutta jos nopeus on keskeistä, kannattaa käyttää SurfaceView-luokkaa. SurfaceView:ssa sovellusta ei pakoteta odottamaan, että järjestelmän view-hierarkia on valmiina piirto-toimintoon vaan SurfaceView-objektiin voidaan piirtää omaa tahtia erillisessä säikeessä. Aluksi on luotava luokka, joka periytyy SurfaceView-luokasta. Tällöin luokan on implementoitava myös SurfaceHolder.Callback -rajapinta, jonka kautta päästään käsiksi Surface-objektiin. Surface-objektiin piirretään SurfaceHolderin välityksellä kutsumalla *getHolder()* -metodia. (Canvas and drawables. N.d.)

SurfaceView:ta käyttäen voidaan siis tavallaan piirtää ruutu taustalla olevaan pintaan, ja sitten ”kääntää” tämä pinta esiin. Pelissä käytin alkuruudun piirtotoiminnoissa ImageView:ta. ImageView on XML-layouttiin sisällytetty view, jossa on itsessään kuvien lataamis- ja optimointi-toiminnot. Varsinaisessa peliosuudessa käytin SurfaceView:ta.

API 14:stä lähtien Androidissa on ollut myös TextureView-luokka, jota voidaan käyttää ainoastaan laitteistokiihdytteisissä (hardware accelerated) ikkunoissa. Softapohjaisesti renderöidessä niin ruutuun ei piirry mitään. TextureView:ta ja SurfaceView:ta voidaan kumpaakin renderöidä erillisessä säikeessä, päinvastoin kuin muita viewejä. Erona SurfaceView:hun on se, että TextureView ei luo erillistä ikkunaa vaan käyttäytyy kuten tavallinen View, ja sitä voidaan mm. liikutella, muotoilla ja animoida. Ja esimerkiksi asettaa läpinäkyväksi. (TextureView. N.d.) TextureView:n heikkous SurfaceView:hun verrattuna on sen suurempi muistin tarve, ja lievästi hitaampi renderöinti (Min 2014).

// init surfaceView, access and control over underlying "surface"

surfaceHolder = getHolder();

SurfaceView:n *getHolder*-alustus.

// ruutuun piirtäminen, SurfaceView  
public void drawAll() {  
 // check surface validity  
 if (surfaceHolder.getSurface().isValid()){  
 //LOCK canvas  
 canvas = surfaceHolder.lockCanvas();  
 //draw background etc  
 canvas.drawColor(Color.*argb*(255, 100, 170, 50));  
 maali2.setColor(Color.*argb*(255, 140, 160, 255));  
 canvas.drawRect(0,0,ruudunlev,300,maali2);

. . .

// UNLOCK canvas  
surfaceHolder.unlockCanvasAndPost(canvas); // "flip screen"

Canvas ”lukitaan” *lockCanvas*-metodilla, suoritetaan piirtotoiminnot, ja ”käännetään” kuva näkyviin *unlockCanvasAndPost*-metodilla.

# Canvas ja Paint

Yksinkertaista, staattista, 2D-grafiikkaa voidaan piirtää suoraan layoutin View-objektiin, mutta dynaamisesti muuttuvaan grafiikkaan, kuten peleihin, on käytettävä canvasia. Canvas välitetään luokan onDraw(Canvas)-metodille, tai luodaan täysin uusi canvas. Canvasta luotaessa on määriteltävä sen pohjaksi bitmap. (Canvas and drawables. N.d)

Canvasiin voidaan piirtää muun muassa muotoja ja polkuja, bitmappeja ja tekstiä. Canvasilla olevia kuvia voidaan käännellä, skaalata, kääntää ym. Myös sen tila voidaan tallentaa tai ladata. Paint puolestaan määrittää piirtämisessä käytetyn ”siveltimen” värin ja tyypin. Väriä, tyyliä (solid fill, outline stroke), fontti, filttereitä, ”transfer modea” ym voidaan muutella. (Graphics Basics 2010.)

Paint maali = new Paint(); // luodaan "maali"  
Paint peittomaali = new Paint();// luodaan "peittomaali"

Pelin alkuruutua varten luodut kaksi Paint-tyyliä.

// Display = size and density of a logical display  
ruutu = getWindowManager().getDefaultDisplay();  
  
ruutukoko = new Point(); // get resolution  
ruutu.getSize(ruutukoko); // current \_app window\_ size

Ruudun koko on haettu *getWindowManager.getDefaultDisplay*:n kautta.

Bitmap bitmappi = Bitmap.*createBitmap*(ruutukoko.x, ruutukoko.y, Bitmap.Config.*RGB\_565*);

// Create a new canvas to draw on, and link it to the bitmap  
Canvas canvas = new Canvas(bitmappi);

Ylläolevassa koodinpätkässä näkyy, miten luodaan bitmap canvasin alustaksi, ja yhdistetään se canvasiin.

Bitmappia luodessa on määriteltävä sen tyyppi, kuten edellisen koodin ”*Bitmap.Config.RGB\_565*”. Tämä määrittää bitmapin pikseleiden tallennustavan, ja se vaikuttaa sovelluksen käyttämään muistin määrään.

|  |  |
| --- | --- |
| ALPHA\_8 | Each pixel is stored as a single translucency (alpha) channel. |
| ARGB\_4444 | This field was deprecated in API level 13. Use ARGB\_8888 instead. |
| ARGB\_8888 | Each pixel is stored on 4 bytes. |
| HARDWARE | Special configuration, when bitmap is stored only in graphic memory. |
| RGBA\_F16 | Each pixels is stored on 8 bytes. |
| RGB\_565 | Each pixel is stored on 2 bytes and only the RGB channels are encoded: red is stored with 5 bits of precision (32 possible values), green is stored with 6 bits of precision (64 possible values) and blue is stored with 5 bits of precision. |

Taulukko 1. Bitmap-konfiguraatiot. (Bitmap.Config. N.d.)



Kuva 1. Alkuruudun tausta, jmm\_logo.png.

Ylläolevissa taustakuvissa vasemmalla on käytetty ARGB\_8888 -moodia, joka käyttää jokaisen pikselin tallennukseen 4 tavua. Oikeanpuoleisessa RGB\_565:ttä, joka käyttää muistia 2 tavua/pikseli. Tällaisessa liukuvärikuvassa näkyy selvästi, että RGB\_565:n bittimäärä ei ole riittävä.

# Canvasiin piirtäminen

Kuva, joka sisältää kohteen teksti, merkki

Kuvaus luotu, korkea luotettavuus

Kuva 2. Pelin alkuruutu.

Pelin alkuruutua piirtäessä laitoin ensimmäisenä taustaksi bitmapin resursseista *drawBitmap*-metodilla:

// lataa tausta-bitmap resursseista  
Bitmap tausta

= BitmapFactory.*decodeResource*(getResources(), R.drawable.*jmm\_logo*);

// aseta "tausta" canvasiin (kuva, left, top, paint)   
canvas.drawBitmap(tausta,0,0,null);

Sitten ruudun alalaitaan ”nurmikko” *drawRect*-metodilla:

// "nurmikko"  
maali.setColor(Color.*parseColor*("#008800")); // asetetaan väri  
canvas.drawRect(0,900,ruutukoko.x,ruutukoko.y,maali); // laatikko

Otsikkotekstin J on tehty ensin ympyrästä (*drawCircle*) ja nelikulmiosta:

// J ympyröillä ja neliöillä  
canvas.drawRect(230,50,330,290, maali); // rect  
canvas.drawCircle(200,300,130,maali); // circle

Osa J:n ympyrästä piti saada peitettyä taustavärin värisellä ”Rect”:illä, jotta siihen saatiin J:n muoto. Ruudulta saadaan napattua väri Bitmap-luokan *getPixel*-metodilla:

// otetaan pikselin väri tausta-bitmapista ja "peittomaali":n arvoksi  
int pikseli = tausta.getPixel(20,20); // (x,y)  
peittomaali.setColor(pikseli); // aseta peittomaalin väri pikselistä

. . .

canvas.drawRect(50,100,250,300,peittomaali); // J:n peitto-osa

Muut kirjaimet on tehty *drawLine* -metodia käyttäen. Rk-muuttujalla säädetään kirjaimet ruutuun sopiviksi. Joka kirjaimesta tehtiin oma funktio, tässä esimerkkinä A:

// piirrä viivoilla A

public void piirraA(Canvas canvas, float xsij, float ysij, Paint maali){  
 canvas.drawLine(xsij,ysij,xsij+100/rk,ysij-380/rk, maali);  
 canvas.drawLine(xsij+100/rk,ysij-380/rk,xsij+200/rk,ysij, maali);  
 canvas.drawLine(xsij+50/rk,ysij-130/rk,xsij+150/rk,ysij-130/rk, maali);  
}

Viivan paksuus määriteltiin ensin Paint-luokan *setStrokeWidth*-metodilla:

// aseta viivan paksuus  
maali.setStrokeWidth(30);

Aiemmin haettu ruutukoko laitettiin ruudulle *drawText*-metodilla, tekstin koko määriteltiin Paint:in *setTextSize*-metodilla:

// fontin koon määrittely. Lasketaan ensin suhdeluku ruudun koon perusteella  
double kuvasuhde = (Math.*sqrt*(canvas.getWidth() \* canvas.getHeight()))/1600;

// tekstin koko  
maali.setTextSize((float) (getResources().getDimensionPixelSize(R.dimen.*fonttikoko*) \* kuvasuhde));

// text  
canvas.drawText("screen size: "+ruutukoko.x+" x " + ruutukoko.y,20,ruutukoko.y-30,maali);

Paint-luokalla on myös mm. *setAntiAlias*-metodi, jolla piirrettyjen kuvioiden kulmat saadaan pehmennettyä: maali.setAntiAlias(true).

Jotta ”bitmappi”-canvasiin piirretyt muutokset saadaan näkyviin, tarvitaan lopuksi:

// liitetään bitmappi view:hun  
alkukuvaView.setImageBitmap(bitmappi);

API 21:stä lähtien canvasissa on käytössä lisäksi mm. *drawOval* ja *drawArc*. Jälkimmäisellä voidaan piirtää esimerkiksi ympyrädiagrammeja.

# Alpha ja XferMode

Paintin *setColor* -metodilla voidaan määrittää paitsi väri, myös alpha, eli läpinäkyvyys, käyttäen Color-luokan argb-metodia. Alkuruudun väripallot on piirretty alphan arvolla 200, väreiltään ne ovat täysin sininen, vihreä ja punainen.

// vaihdetaan läpikuultava väri ympyröille  
maali.setColor(Color.*argb*(50, 255, 0, 0)); // alpha + rgb = 0-255

// väripallot  
int alpha = 200;  
int r = Color.*argb*(alpha, 255, 0, 0);  
int g = Color.*argb*(alpha, 0, 255, 0);  
int b = Color.*argb*(alpha, 0, 0, 255);  
maali.setColor(b);  
canvas.drawCircle(320,320,70,maali);  
maali.setColor(g);  
canvas.drawCircle(340,400,70,maali);  
maali.setColor(r);  
canvas.drawCircle(260,380,70,maali);



Kuva 3. Alkuruudun väripallot läpinäkyvyydellä.

Paintilla on myös ”Transfermode”, joka määrittelee miten alla olevan kuvan pikselit yhdistyvät päälle piirrettäviin pikseleihin. Metodin nimi on *setXfermode*. Yleisimmin käytetty moodi on Porter-Duff. Porter-Duffista itsestään löytyy suuri määrä eri moodeja , mm. ”source in”, ”source atop”, ”add”, ”clear”, ”darken” ym. (PorterDuff.Mode. N.d.)

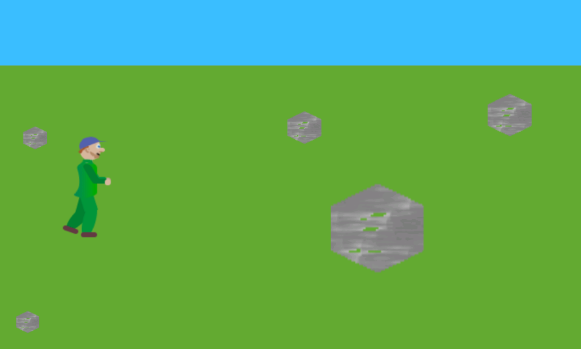
Pelin alkuruudun palloille asetin Porter-Duffin Add-moodilla, joka lisää lähdepikselit kohdepikseleihin ja värikyllästää ne (saturaatio):

maali.setXfermode(new PorterDuffXfermode(Mode.*ADD*));



Kuva 4. Alkuruudun väripallot läpinäkyvyydellä ja Porter-Duffin Add-moodilla.

# Peli-osion piirtotoiminnot



Kuva 5. Peliruutu.

Peli-osio tehtiin SurfaceView-tyyppiseksi. Sitä varten luotiin GameView-luokka, joka periytettiin SurfaceView-luokasta ja toteuttaa Runnable -rajapintaa, eli se ajetaan säikeessä.

Kuten aiemmin mainittiin, SurfaceView tarvitsee abstraktin SurfaceHolder-rajapinnan:

private SurfaceHolder surfaceHolder;

. . .

surfaceHolder = getHolder(); // käsiksi takana piilevään ”pintaan”

Liitetään peliin jamppa- ja kivi -oliot:

// init player, viedään myös ruudunkoko-tieto  
jamppa = new Jamppa(konteksti, ruudunleveys, ruudunkorkeus);  
  
. . .  
int rockCount = 5;  
for (int i = 0; i < rockCount; i++){  
 Rock kivi = new Rock(konteksti, ruudunleveys, ruudunkorkeus);  
 rocks.add(kivi);  
}

Koska GameView ajetaan säikeessä, tarvitaan run()-metodi, jossa suoritetaan kaikki säikeessä tapahtuvat toiminnot:

@Override  
public void run() {  
 while (pelimenossa) {  
 update();  
 drawAll();  
 control();  
 }  
}

Update-metodissa päivitetään jampan ja kivien liikkeet. Jamppa- ja Rock -luokkien update-metodit siirtävät jamppaa ja kiviä muuttamalla niiden x- ja y-arvoja.

public void update() {  
 // update player position  
 jamppa.update();  
  
 // update rocks  
 for (Rock kivi : rocks) {  
 kivi.update(jamppa.getNopeus());  
 }  
}

DrawAll-metodissa ”päivitetään ruutu”, kuten luku 2:ssa kuvailtiin, käyttäen SurfaceHolderin *lockCanvas ja unlockCanvasAndPost*-metodeita:

public void drawAll() {  
 // check surface validity  
 if (surfaceHolder.getSurface().isValid()){  
 //LOCK canvas  
 canvas = surfaceHolder.lockCanvas();  
 canvas.drawColor(Color.*argb*(255, 100, 170, 50));//draw backgr  
 maali2.setColor(Color.*argb*(255, 60, 190, 255)); // sky-color  
 canvas.drawRect(0,0,ruudunlev,300,maali2); // sky  
 // draw rocks  
 for (Rock kivi : rocks){  
 canvas.drawBitmap(  
 kivi.getRockBitmap(),  
 kivi.getX(),  
 kivi.getY(),  
 maali);  
 }  
 // draw jamppa  
 canvas.drawBitmap(  
 jamppa.getJamppaBitmap(),  
 jamppa.getX(),  
 jamppa.getY(),  
 maali);  
 // UNLOCK canvas  
 surfaceHolder.unlockCanvasAndPost(canvas); // "flip screen?"  
 }  
}

Control-metodissa määritellään pelin ”fps”, hidastamalla säiettä hieman:

peliThread.*sleep*(17); // delay thread = framerate.

# Rock-metodin piirtotoiminnot

Rock-metodissa liikutellaan kiviä ruudun editse. Kun kivi poistuu vasemmasta reunasta, se regeneroidaan oikeaan reunaan. Samassa yhteydessä kiven koko arvotaan, ja kuva skaalataan alkuperäisestä rockBitmap-kuvasta scaledRockBitmap -kuvaksi Bitmap-luokan *createScaledBitmap*-metodilla:

// rock off the screen -> back to right  
if (x < (0 - scaledRockBitmap.getWidth())) {  
 x = ruudunlev;  
 Random noppa = new Random();  
 y = (noppa.nextInt(ruudunkork - 300))+200; // random koko   
  
 // scale the rocks with double random  
 Random dblNoppa = new Random();  
 double kerroin = (0.1 + (2 - 0.5) \* dblNoppa.nextDouble())\*kivenkoko;  
 int koko = (int)kerroin;  
  
 scaledRockBitmap = Bitmap.*createScaledBitmap*(rockBitmap,koko,koko,true);

GameView:hun palautetaan kiven kuva getRockBitmap-metodissa:

public Bitmap getRockBitmap() { return scaledRockBitmap; }

# Jamppa-metodin piirtotoiminnot

Jampan animointi on tehty hyvin yksinkertaisesti lataamalla kaksi kuvaa, jamppaBitmap ja jamppaBitmap2, ja vaihtelemalla niitä. Näiden ajastettua vaihtamista varten on otettu käyttöön handler, jota kutsutaan 300 millisekunnin välein:

// Jamppa framechange handler  
handler = new Handler();  
runnable = new Runnable() {  
 public void run() {  
 // change frames  
 if (jamppaFrame){  
 jamppaFrame = false;  
 } else {  
 jamppaFrame = true;  
 }  
 handler.postDelayed(this, 300); // 300ms delay  
 }  
};  
runnable.run();

Tässä säikeessä yksinkertaisesti vaihdetaan jamppaFrame-booleanin arvoa (true/false) 300ms välein.

Sitten GameView:lle palautetaan bitmappinä vuorotellen jamppaBitmap tai jamppaBitmap2:

// animate jamppa-frames  
public Bitmap getJamppaBitmap() {  
 if (jamppaFrame) {  
 return jamppaBitmap;  
 } else {  
 return jamppaBitmap2;  
 }  
}

# Pelin ohjaus

Jampan ohjaus on myös perin yksinkertainen: kun näyttöä ei kosketeta, jamppa nousee ylös ja näyttöä kosketettaessa siirtyy alas.

@Override  
 public boolean onTouchEvent(MotionEvent liikeEventti){  
 // get motion event, UP = touches screen, DOWN = no touch  
 switch (liikeEventti.getAction() & MotionEvent.*ACTION\_MASK*){  
 case MotionEvent.*ACTION\_UP*:  
 jamppa.meneYlos();

break;  
 case MotionEvent.*ACTION\_DOWN*:  
 jamppa.meneAlas();

int r = jamppa.getY();

Log.*d*("MyTagGoesHere", "jamppa alas"+r);  
 break;  
 }  
 return true;  
 }  
}

# Lähteet

Bitmap.Config. N.d. Android-manuaali developer.android.com-sivustolla. Viitattu 11.11.2017. <Https://developer.android.com/reference/android/graphics/Bitmap.Config.html>

Canvas and drawables. N.d. Android-manuaali developer.android.com-sivustolla. Viitattu 11.11.2017. <Https://developer.android.com/guide/topics/graphics/2d-graphics.html>

Graphics basics. 2010. Artikkeli Android.Graphics-sivustolla. Viitattu 11.11.2017. <Http://android.nakatome.net/2010/04/graphics-basics.html>

Min, H. 2014. Android SurfaceView vs TextureView -artikkeli. Viitattu 11.11.2017. <Https://github.com/crosswalk-project/crosswalk-website/wiki/Android-SurfaceView-vs-TextureView>

PorterDuff.Mode. N.d. Android-manuaali developer.android.com-sivustolla. Viitattu 12.11.2017. <Https://developer.android.com/reference/android/graphics/PorterDuff.Mode.html>

TextureView. N.d. Android-manuaali developer.android.com-sivustolla. Viitattu 11.11.2017. <Https://developer.android.com/reference/android/view/TextureView.html>

View. N.d. Android-manuaali developer.android.com-sivustolla. Viitattu 11.11.2017. <Https://developer.android.com/reference/android/view/View.html>