# ANDROID IV.

KIV/MKZ 2016

L. Pešička

# OBSAH

- vibrace
- animace
- game development
- audio/video
- procesy a vlákna
- databáze

# OBRÁZEK NA POZADÍ

- obrázek do /res/drawable/bg.jpg
- do XML popisu layoutu:

```
<RelativeLayout
 …
android:layout_width="match_parent"
android:layout_height="match_parent"
android:background="@drawable/bg"
… >
```

# DLAŽDICE OBRÁZKŮ NA POZADÍ

android:background="@drawable/backrepeat"

soubor backrepeat.xml:

```
<bitmap
xmlns:android="http://schemas.android.com/apk/res/android"
    android:src="@drawable/bg"
    android:tileMode="repeat"
    android:dither="true" />
```

Odkaz:
http://androidforbeginners.blogspot.fr/2010/06/how-to-tile-background-image-in-android.html

# UKÁZKA A POZNÁMKA


Vhodně zvolený obrázek v pozadí významně ovlivní vzhled celé aplikace a její působení na uživatele.

Vypadat může různě na malém displeji telefonu a velkém displeji tabletu.

Je třeba dát pozor na autorská práva k obrázkům.

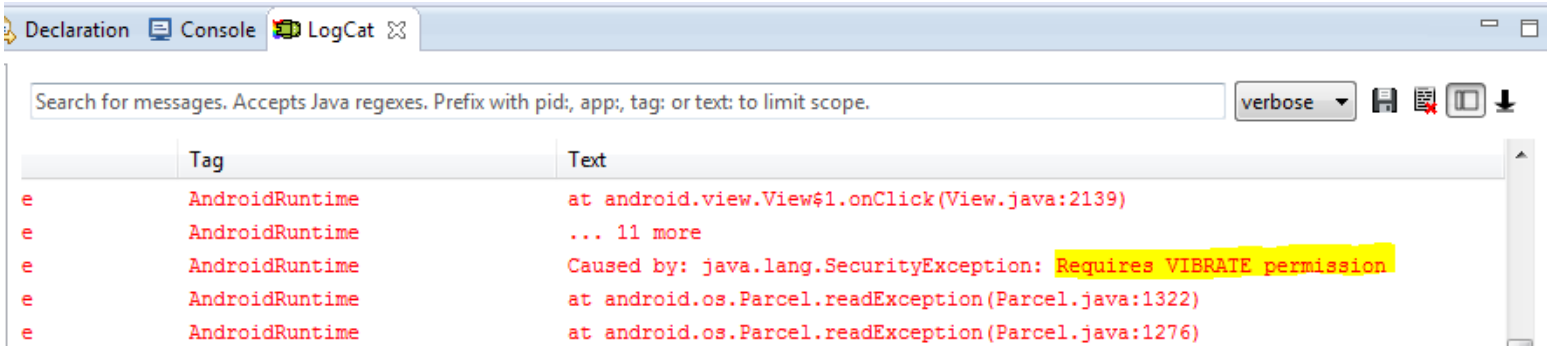
# VIBRACE

tiché upozornění na událost (neruší okolí)

zvýšení atraktivnosti  hry

- notifikace doplněná vibrací
- samostatná vibrace

právo do manifestu (pro obojí):



# VIBRACE

právo do manifestu:

```
<uses-permission android:name="android.permission.VIBRATE"/>
```

```
Vibrator vibrator = (Vibrator)
getSystemService(Context.VIBRATOR_SERVICE);
// vzor kolik ms má trvat on a off
// napřed jak dlouho čekat, jak dlouho vibrovat, jak dl. čekat
long []  vzor = {0, 100, 200, 250, 300 };
vibrator.vibrate(vzor, 0);          // 0 – opakuje od indexu 0, -1 ne
vibrator.vibrate(1000);             // nebo kolik ms vibruje
…
vibrator.cancel();                  // lze zrušit
```

# ANIMACE

- Lepší dojem z aplikace
  - Dynamika
- Upozornění uživatele
  - Přesun herní figurky
  - Zvýraznění posledního tahu (např. piškvorky)
- Animace jde vytvořit jednoduše popisem transformací obrázku
  - Není pracné na vytváření obrázků apod.

# VIEW ANIMACE (OD API1)

lze využít pro libovolný View

základní typy animace:

- **tween** animace
  - start, end point
  - velikost, rotace, transparentnost …
- **frame** animace
  - klasická posloupnost obrázků

- **property** animace
  - Další obecnější typ animace
  - Změna vlastností objektů pro daný časový interval

# TWEEN ANIMACE

- jednoduché transformace
  - pozice, velikost, rotace, transparentnost
  - např. rotace textu na obrazovce
- lze definovat v XML nebo v kódu
  - XML doporučeno
- definujeme
  - co se má dělat
  - kdy se má změna provést
  - jak dlouho bude trvat
  - sekvenční nebo simultánní transformace (více akcí najednou – stejný startovací čas)

# TWEEN XML

- res/anim
  - umístění XML souboru s animací
- jeden root element
  - <alpha>, <scale>, <translate>, <rotate>, <set>
- defaultně vše aplikováno současně
- sekvenčně
  - specifikovat startOffset atribut
- souřadnice
  - (0,0) levý horní roh
  - některé údaje – relativně k sobě nebo k rodiči
    50 – relativně k rodiči, 50% - vztaženo k sobě

# TWEEN XML - UKÁZKA ANIMACE

```
<set
    android:shareInterpolator="false"

xmlns:android="http://schemas.android.com/apk/res/android
">

    <scale
android:interpolator="@android:anim/accelerate_decelerate
_interpolator"
        android:fromXScale="1.0"
        android:toXScale="1.4"
        android:fromYScale="1.0"
        android:toYScale="0.6"
        android:pivotX="50%"
        android:pivotY="50%"
        android:fillAfter="false"
        android:duration="700" />
```

```
<set
    android:interpolator="@android:anim/decelerate_interpolator"
    xmlns:android="http://schemas.android.com/apk/res/android">

    <scale android:fromXScale="1.4"
        android:toXScale="0.0"
        android:fromYScale="0.6"
        android:toYScale="0.0"
        android:pivotX="50%"
        android:pivotY="50%"
        android:startOffset="700"
        android:duration="400"
        android:fillBefore="false" />

    <rotate android:fromDegrees="0"
        android:toDegrees="-45"
        android:toYScale="0.0"
        android:pivotX="50%"
        android:pivotY="50%"
        android:startOffset="700"
        android:duration="400" />
</set>
</set>
```

# TWEEN XML - VYVOLÁNÍ ANIMACE

soubor s animací: *mojeanim.xml*
jeho umístění: v *res/anim/*

```
// obrázek
ImageView mujObrazek = (ImageView)
                    findViewById (R.id.imageView);
// animace
Animation mojeAnimace = AnimationUtils.loadAnimation(this,
R.anim.mojeanim);

// animace aplikovaná na obrázek
mujObrazek.startAnimation(mojeAnimace);
```

# UKÁZKA


1. Robot „splaskne“
2. Robot zmizí
3. Robot se znovu objeví

```
public void tlacitko_Vibruj (View v) {

    Vibrator vibrator = (Vibrator)
            getSystemService(Context.VIBRATOR_SERVICE);

    if (vibrator != null) {
        long[] vzor = {1000, 2000, 1000, 2000, 1000};
        vibrator.vibrate(vzor, 0);
        vibrator.vibrate(1000);
    }

    ImageView mujObrazek = (ImageView) findViewById(R.id.imageView);

    Animation mojeAnimace = AnimationUtils.loadAnimation(this, R.anim.mojeanim);

    mujObrazek.startAnimation(mojeAnimace);

}
```

# Další způsob vyvolání animace

- Animation.setStartTime();
  - definovat čas startu animace
- View.setAnimation();
  - přiřazení animace

# INFO

- informace o atributech animace:

http://developer.android.com/guide/topics/resources/animation-resource.html

<alpha> .. průhlednost

android:fromAlpha (0.0 transparentní, 1.0 neprůhledný)

android:toAlpha

<scale> .. změna měřítka

lze specifikovat centrální bod obrázku (pivotX, pivotY)
např. 0,0 .. růst bude dolů vpravo

android:fromXScale

Float. Starting X size offset, where 1.0 is no change.

android:toXScale

android:fromYScale

android:toYScale

android:pivotX

Float. The X coordinate to remain fixed when the object is scaled.

android:pivotY

# INFO

- <translate>
  - vertikální / horizontální pohyb
  - viz dokumentace
- <rotate>
  - rotační animace
  - android:fromDegrees
  - Float. Starting angular position, in degrees.
  - android:toDegrees
  - android:pivotX
  - Float nebo procentně. The X souřadnice centra rotace. Buď v pixelech relativně k levé hraně objektu (např. "5"), v % relativně k levé hraně ("5%"), v procentech relativně k levé hraně rodičovského kontejneru ("5%p").
  - android:pivotY

# INTERPOLÁTOR

- ovlivňuje rychlost změn v animaci
- zrychlovat, zpomalovat, opakovat, ohraničit
- atribut android:interpolator
- androidí + vlastní

nastavení:

```
<set android:interpolator=
    "@android:anim/accelerate_interpolator">
    ...
</set>
```

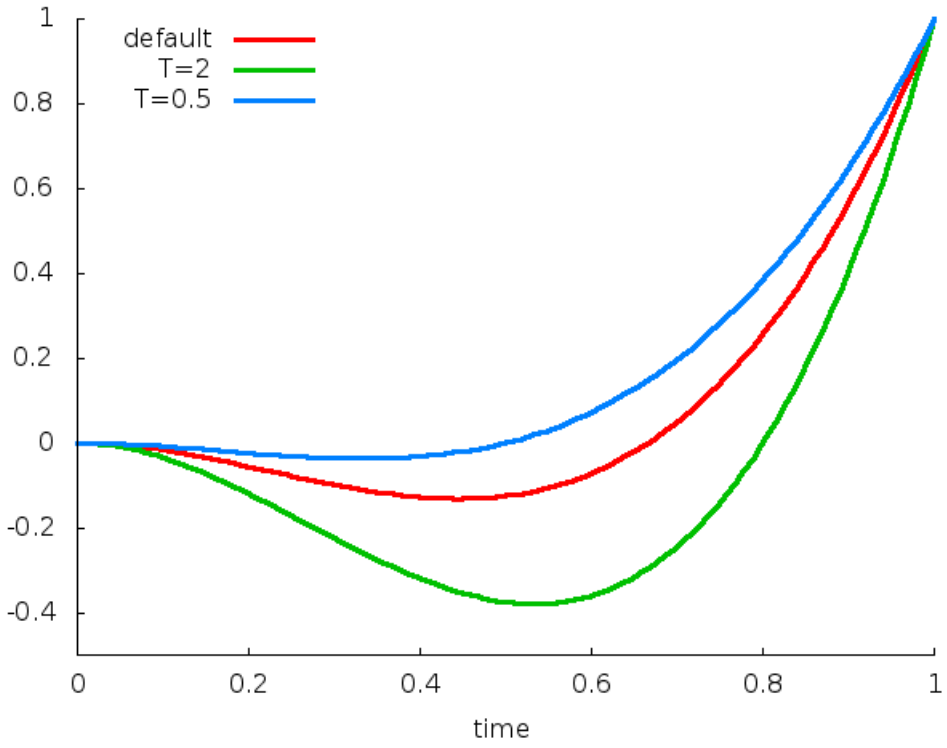
# Příklady androidích interpolátorů

- @android:anim/accelerate_decelerate_interpolator
- @android:anim/accelerate_interpolator
- @android:anim/anticipate_interpolator
  -- začne pohybem zpátky, pak se pohne dopředu
- @android:anim/anticipate_overshoot_interpolator
  -- začne zpátky, pak dopředu, přeběhne cíl a vrátí se
- @android:anim/bounce_interpolator
  -- pohupuje se
- @android:anim/cycle_interpolator
- @android:anim/decelerate_interpolator
- @android:anim/linear_interpolator
  -- tempo změn je konstantní
- @android:anim/overshoot_interpolator
  -- přeběhne cíl a vrátí se

# MATEMATIKA ZA INTERPOLÁTORY

http://cogitolearning.co.uk/?p=1078

ukázka – anticipate_interpolator:



# UKÁZKA - VIDEO

http://www.techrepublic.com/blog/software-engineer/behold-the-animation-magic-of-an-android-interpolator/

# FRAME ANIMACE

- klasická sekvence obrázků
- popis lze v XML (prefer.) i kódu
- element <animation-list>
- posloupnost elementů <item>

jednorázově

```
<animation-list
xmlns:android="http://schemas.android.com/apk/res/android"
    android:oneshot="true">
    <item android:drawable="@drawable/rocket_thrust1"
android:duration="200" />
    <item android:drawable="@drawable/rocket_thrust2"
android:duration="200" />
    <item android:drawable="@drawable/rocket_thrust3"
android:duration="200" />
</animation-list>
```

# FRAME ANIMACE - ZOBRAZENÍ

```
AnimationDrawable rocketAnimation;

public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);

  ImageView rocketImage = (ImageView)
                findViewById(R.id.rocket_image);
  rocketImage.setBackgroundResource(R.drawable.rocket_thrust);
  rocketAnimation = (AnimationDrawable) rocketImage.getBackground();
}

public boolean onTouchEvent(MotionEvent event) {
  if (event.getAction() == MotionEvent.ACTION_DOWN) {
    rocketAnimation.start();
    return true;
  }
  return super.onTouchEvent(event);
}
```

# POZNÁMKA

chceme-li animaci spustit bezprostředně,

můžeme zavolat rocketAnimation.start()

v metodě onWindowFocusChanged() aktivity,

v onCreate() ještě nejde

# PROPERTY ANIMATION – OD ANDROID 3.0 (API 11)

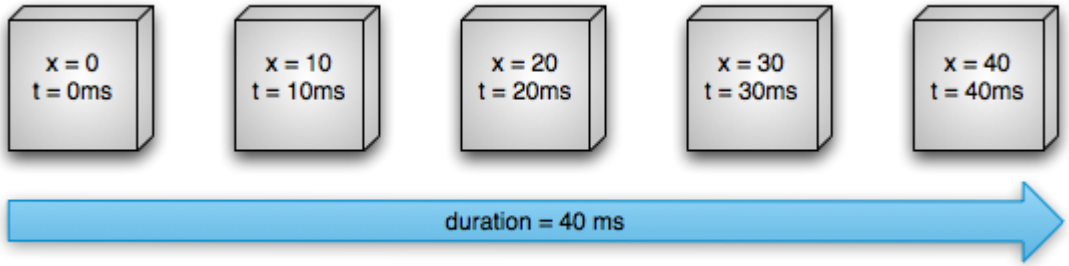
- lze měnit vlastnosti libovolné vlastnosti objektu v čase, bez ohledu na to, zda je vykreslen na obrazovce

- View Animace
  - pouze animuje View objekty
  - udává, kde je View vykreslen, ale nemění View jako takové
    např. animované tlačítko „pluje“ po obrazovce, ale zpracovává kliknutí jen na původní pozici

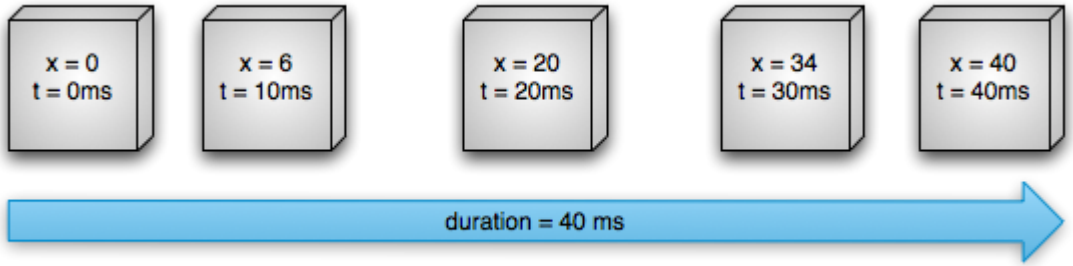
# PROPERTY ANIMATION

- trvání (defaultně 300ms)
- časová interpolace (změna v čase)
- počet opakování
  - kolikrát opakovat
  - přehrát pozpátku
- animator set
  - přehrávání postupně, současně, po zpoždění
- frame refresh delay
  - defaultně 10ms

# PROPERTY ANIMATION




lineární animace




nelineární animace

zdroj obrázků: Android SDK

# PROPERTY ANIMATION

- Animator – tiká
- Interpolátor  0..1  funkce dle času
- Evaluátor x0 + y * (x1 - x0)

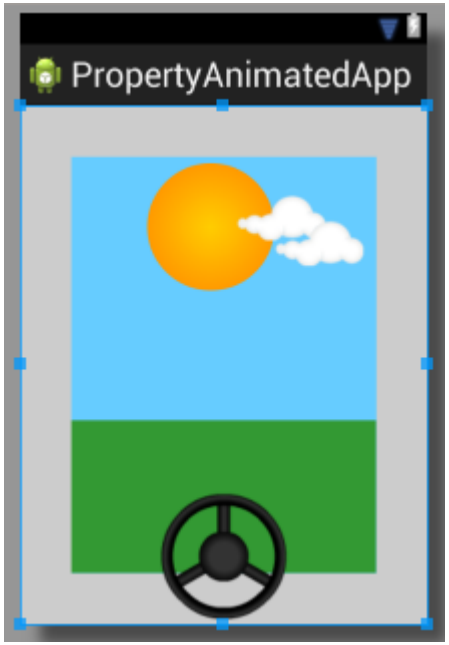
```
ObjectAnimator anim =
ObjectAnimator.ofFloat(v, View.ROTATION_X, 0, 180);

anim.setRepeatCount(ObjectAnimator.INFINITE);
anim.setDuration(1000);
anim.start();
```

1. Plays `bounceAnim`.
2. Plays `squashAnim1`, `squashAnim2`, `stretchAnim1`, and `stretchAnim2` at the same time.
3. Plays `bounceBackAnim`.
4. Plays `fadeAnim`.

```
AnimatorSet bouncer = new AnimatorSet();

bouncer.play(bounceAnim).before(squashAnim1);
bouncer.play(squashAnim1).with(squashAnim2);
bouncer.play(squashAnim1).with(stretchAnim1);
bouncer.play(squashAnim1).with(stretchAnim2);
bouncer.play(bounceBackAnim).after(stretchAnim2);

ValueAnimator fadeAnim = ObjectAnimator.ofFloat(newBall,
"alpha", 1f, 0f);
fadeAnim.setDuration(250);
AnimatorSet animatorSet = new AnimatorSet();
animatorSet.play(bouncer).before(fadeAnim);
animatorSet.start()
```

# POZNÁMKA

- Pěkný tutoriál
- http://code.tutsplus.com/tutorials/android-sdk-creating-a-simple-property-animation--mobile-15022



# GAME DEVELOPMENT

Ukázka vývoje hry, založeno na:

http://www.edu4java.com/androidgame.html

1. Herní smyčka (game loop)
2. Vykreslení stavu (update, draw)
3. Herní logika (detekce kolizí spritů)
4. Zdroje (obrázky, zvuky)

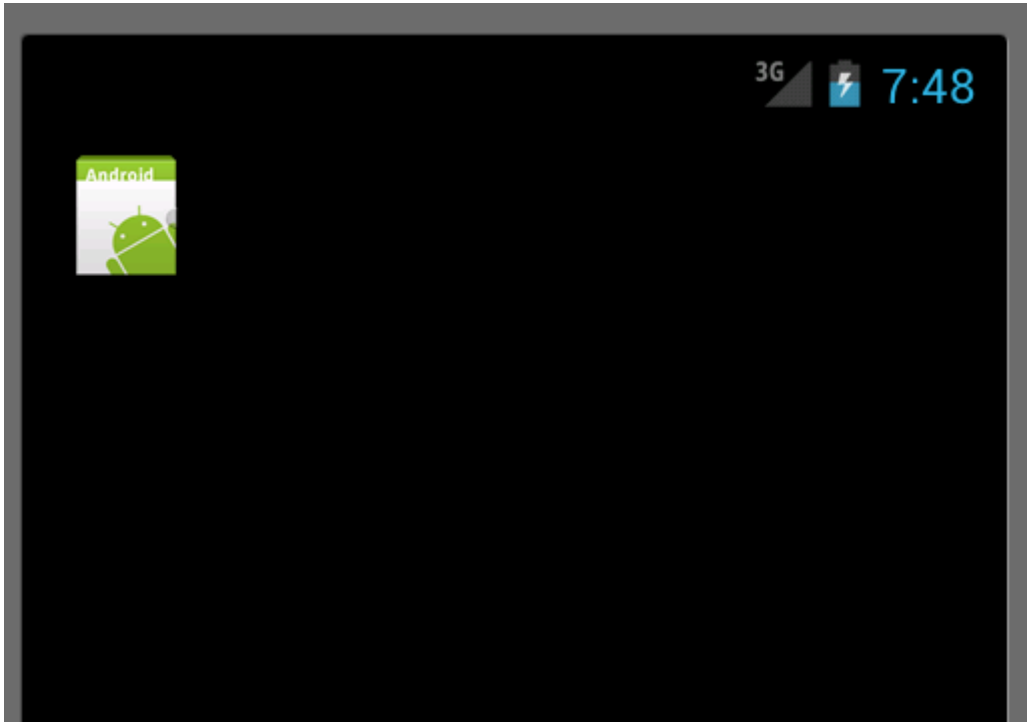
# AKTIVITA (MAIN.JAVA)

```
public class Main extends Activity {

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        requestWindowFeature(Window.FEATURE_NO_TITLE);

        setContentView(new GameView(this));
    }
}
```

# GAMEVIEW - EXTENDS VIEW

```
public class GameView extends View {

    private Bitmap bmp;


    public GameView(Context context) {

        super(context);
        bmp = BitmapFactory.decodeResource(getResources(), R.drawable.icon);

    }

    protected void onDraw(Canvas canvas) {

        canvas.drawColor(Color.BLACK);
        canvas.drawBitmap(bmp, 10, 10, null);

    }
}
```

# EMULÁTOR



# SURFACEVIEW

nahradíme GameView extends **SurfaceView**

v konstruktoru:

```
holder = getHolder();
holder.addCallback(new SurfaceHolder.Callback() {
…
public void surfaceDestroyed(SurfaceHolder holder) { }
 public void surfaceCreated(SurfaceHolder holder) {
        Canvas c = holder.lockCanvas(null);
        onDraw(c);
        holder.unlockCanvasAndPost(c);
 }
```

# POZNÁMKA

Sledujte dokumentaci

http://developer.android.com/reference/android/view/SurfaceHolder.html#lockCanvas%28%29

# GAME LOOP

opakování dvou akcí:

- update scény
  - nová pozice postaviček
  - herní skóre atd.
- kreslení
  - opakováním vzniká dojem animace, pohybu

nové vlákno GameLoopThread:

public class GameLoopThread extends Thread {}

# ZMĚNA I V GAMEVIEW

mění pozici zobrazovaného obrázku

„pluje po obrazovce“

http://www.edu4java.com/en/androidgame/androidgame3.html

pohyb je proměnlivý, abychom dostali konstantní FPS – sleep

# ČASOVÁNÍ

```
startTime = System.currentTimeMillis();

// zde proběhne vykreslení

sleepTime = ticksPS-(System.currentTimeMillis() -
startTime);

try {
    if (sleepTime > 0)
            sleep(sleepTime);
     else
            sleep(10);
  } catch (Exception e) {}
```

zajištění plynulého pohybu bez závislosti na výkonu CPU

# SPRITE

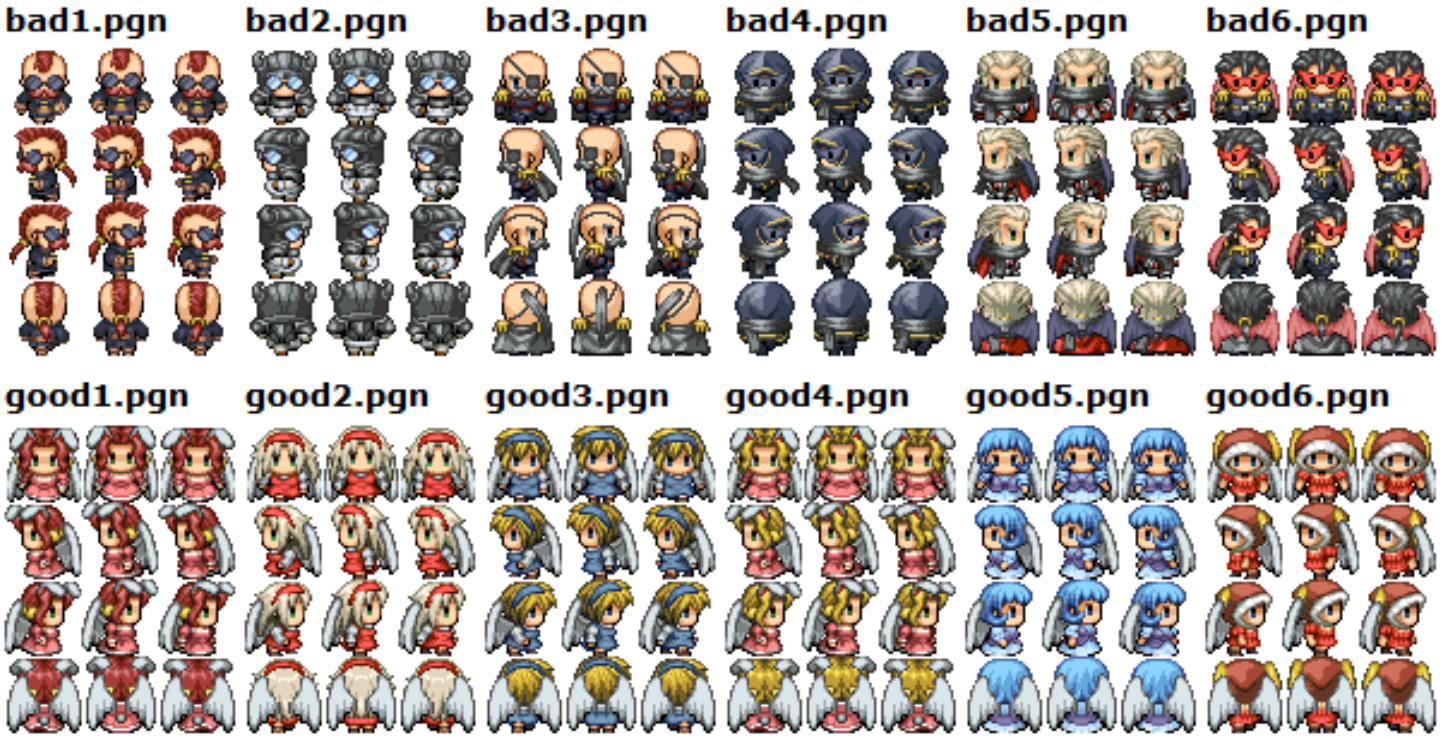
- částečně transparentní 2D rastrový obrázek
- umístěný na obrázek pozadí
- většinou více spritů najednou na obrazovce
- může reprezentovat hráče, protihráče
- má pozici (x,y) a rychlost a směr pohybu
- může obsahovat animaci, zvuky

# JAK VYROBIT SPRITE

- nakreslit
- využít dostupných zdrojů

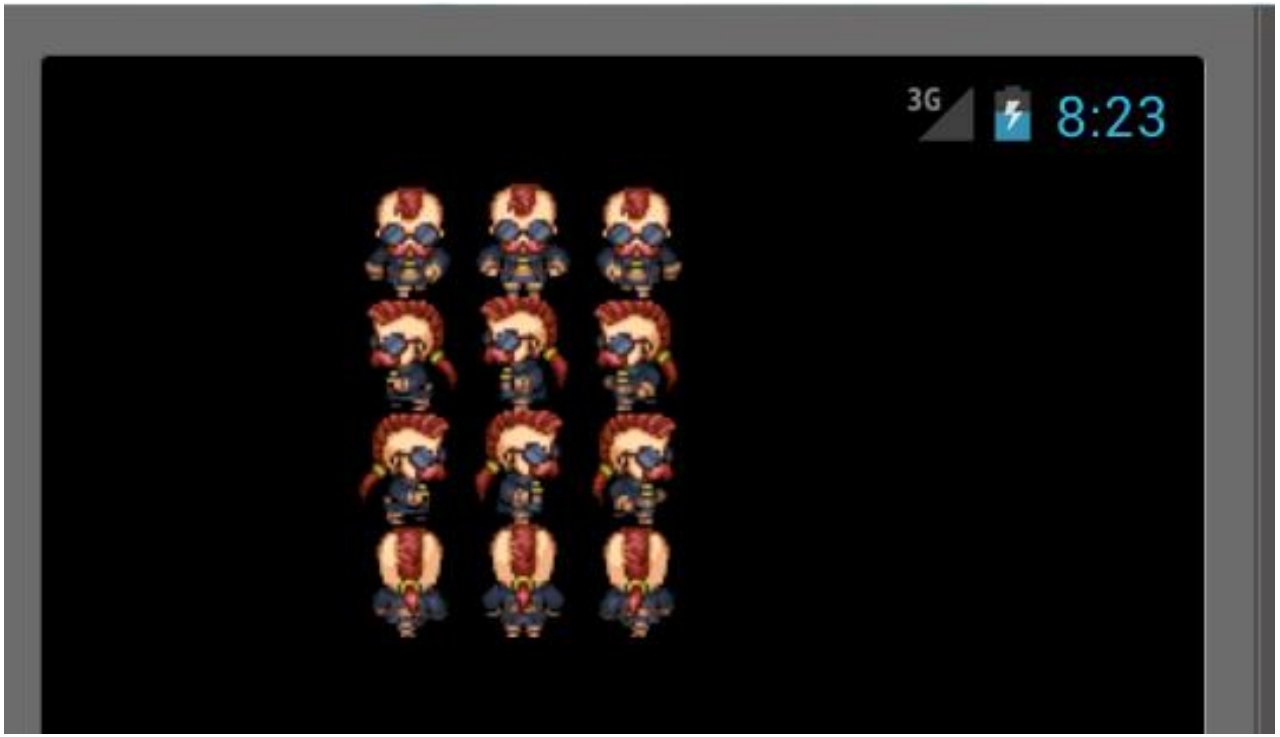
http://www.famitsu.com/freegame/tool/chibi/index2.html

# SPRITE




Součástí 1 obrázku může být více fází pohybu objektu, typicky se zobrazí jenom výřez daného obrázku – tj. aktuální požadovaný tvar

# EMULÁTOR - ZOBRAZENÍ ZATÍM VŠECH ČÁSTÍ SPRITU



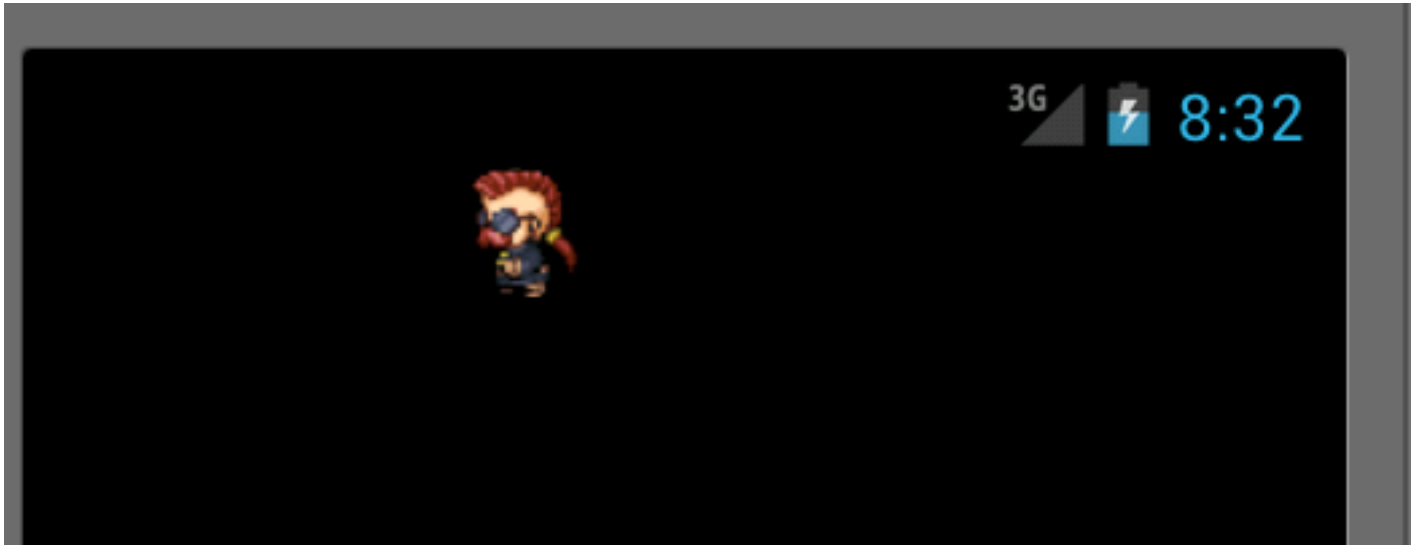
# ZOBRAZENÍ SPRITU

```
// sprite je 4 x 3 obrázky
private static final int BMP_ROWS = 4;
private static final int BMP_COLUMNS = 3;

private void update() {
   if (x > gameView.getWidth() - width - xSpeed) {
                  xSpeed = -5;   }
   if (x + xSpeed < 0) {  xSpeed = 5; }

x = x + xSpeed;
currentFrame = ++currentFrame % BMP_COLUMNS;
 }
```

# ZOBRAZENÍ SPRITU

```
public void onDraw(Canvas canvas) {
        update();
        int srcX = currentFrame * width;
        int srcY = 1 * height;
        Rect src = new Rect(srcX, srcY, srcX + width, srcY +
height);
        Rect dst = new Rect(x, y, x + width, y + height);
        canvas.drawBitmap(bmp, src, dst, null);
    }
```

# EMULÁTOR




rozfázovaný pohyb spritu
canvas.drawBitmap(bmp, src, dst, null);
src – zdrojový obdélník uvnitř bitmapy
dst – cílový obdélník v canvasu kam se bude kreslit

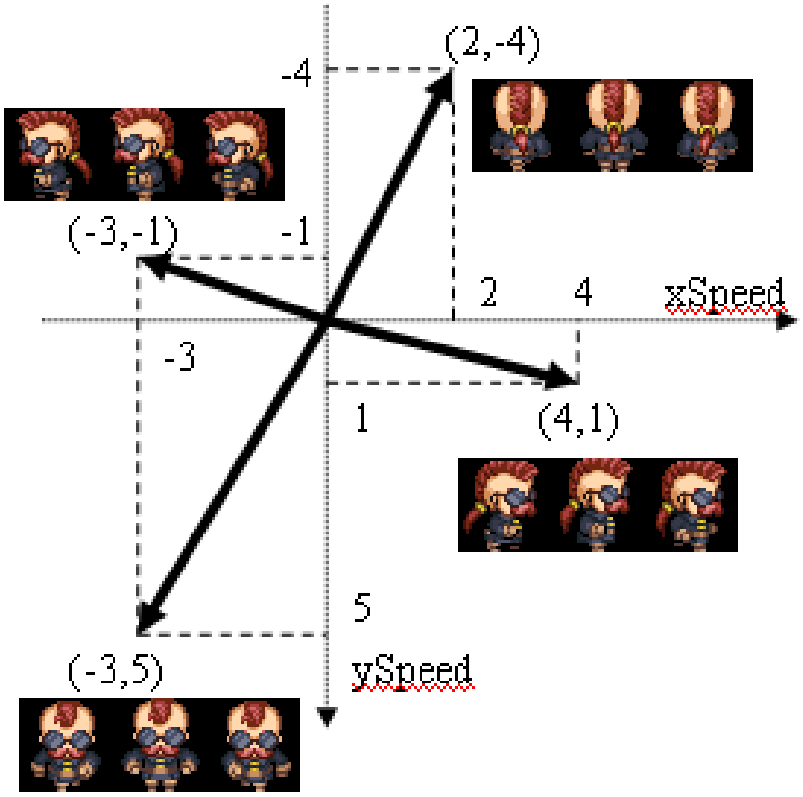
# POHYBY SPRITU

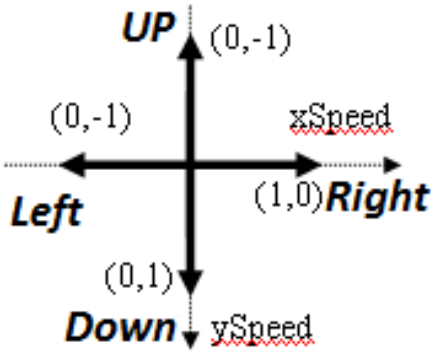
*0 1 2 Frames*

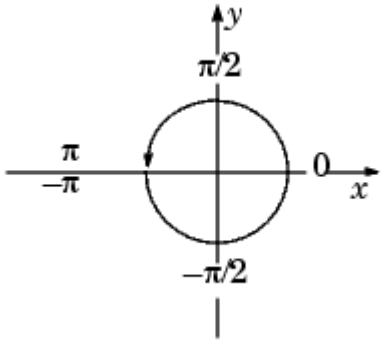
*0 down animation*
*1 left animation*
*2 right animation*
*3 up animation*

vodorovný a svislý pohyb je jednoduchý
při šikmém pohybu, jaký další obrázek máme použít?

# ZA VŠÍM JE MATEMATIKA ☺




http://www.edu4java.com/en/androidgame/androidgame5.html

```
// direction = 0 up, 1 left, 2 down, 3 right,
// animation = 3 up, 1 left, 0 down, 2 right
int[] DIRECTION_TO_ANIMATION_MAP = { 3, 1, 0, 2 };

private int getAnimationRow() {
      double dirDouble = (Math.atan2(xSpeed, ySpeed) / (Math.PI / 2) + 2);
      int direction = (int) Math.round(dirDouble) % BMP_ROWS;
      return DIRECTION_TO_ANIMATION_MAP[direction];
}
```

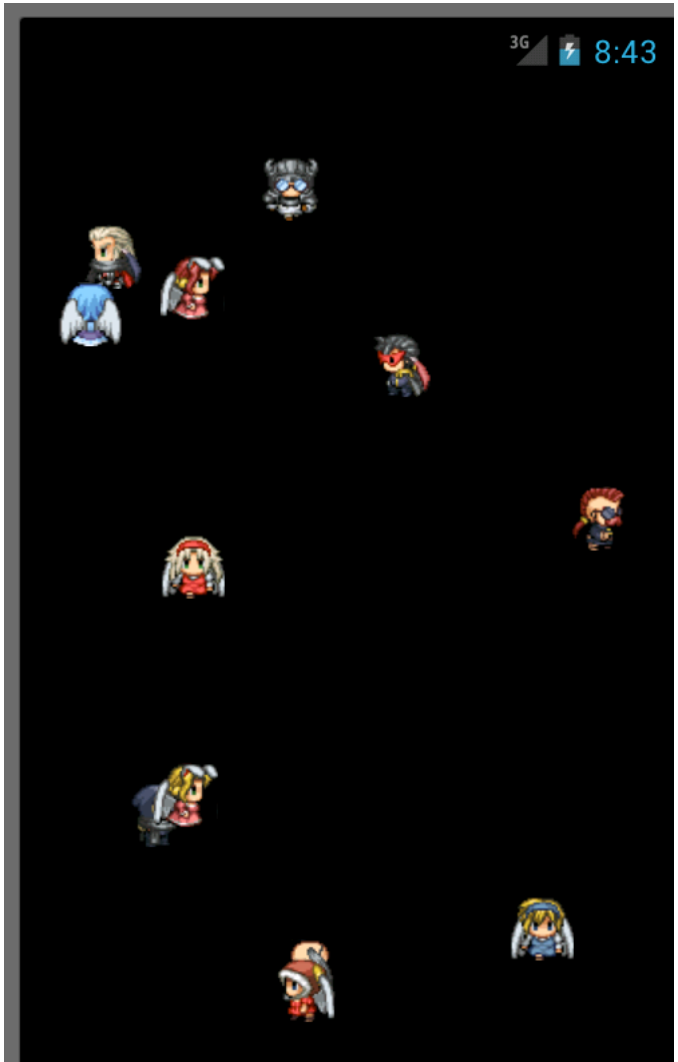
In this table I try to show you what I do in this function:

1. with atan2(xSpeed,ySpeed) I get the radian angle from (-PI to PI)
2. I divide the angle by PI/2 and get a *double* from (-2 to 2)
3. I add 2 to change the range from (0 to 4)
4. I use % (module) to reduce the range to (0 to 3) (notice the 0 and 4 were the same direction)
5. I map each direction to the correct animation using the array `{ 3, 1, 0, 2 }`

| | x | y | atan2(x,y) | atan2(x,y)/(PI/2) | (atan2(x,y)/(PI/2)+2)%4 | bmp row (from 0) |
|---|---|---|---|---|---|---|
| up | 0 | -1 | PI or -PI | 2 or -2 | 4 or 0 | 3 |
| right | 1 | 0 | PI/2 | 1 | 3 | 2 |
| down | 0 | 1 | 0 | 0 | 2 | 0 |
| left | -1 | 0 | -PI/2 | -1 | 1 | 1 |

Note: here x and y are xSpeed and ySpeed.

# PŘIDÁNÍ VÍCE SPRITŮ




```
private void createSprites() {


sprites.add(createSprite(R.drawable
.bad1));

sprites.add(createSprite(R.drawable
.bad2));

sprites.add(createSprite(R.drawable
.bad3));

 }
```

# TOUCH EVENT

- dotyková událost
- zjistíme, zda je kolize s nějakým spritem
- pokud ano, odstraníme sprite ze seznamu
- při příštím onDraw odstraněný sprite nebude vykreslen

```
public boolean onTouchEvent(MotionEvent event) {
        for (int i = sprites.size()-1; i >= 0; i--) {
            Sprite sprite = sprites.get(i);
             if (sprite.isCollition(event.getX(),event.getY())) {
                  sprites.remove(sprite);
             } }
 return super.onTouchEvent(event);   }
```

# KOLIZE

```
public boolean isCollition(float x2, float y2) {

return x2 > x && x2 < x + width && y2 > y && y2 < y + height;

}
```

# Dočasné sprity - po kolizi s postavou

- Obrázek reprezentující zásah
- Stálá pozice (není směr, rychlost)
- Zmizí po nějaké době (odpočet)

# ZDROJ

Popisovaná hra:

http://www.edu4java.com/androidgame.html

jsou zde popsány jednotlivé zde uvedené části

Courseware MKZ – Cvičení – projekt v Android Studiu

# AUDIO-VIDEO

- přehrávání z různých zdrojů
  - soubory v raw resourcích  aplikace
  - soubory ve filesystému
  - datastream přes síťové spojení
- přehrávání audia – videa
  - třída **MediaPlayer**
- záznam audia – videa
  - třída **MediaRecorder**
  - emulátor nedisponuje prostředky pro záznam

# PŘEHRÁVÁNÍ AUDIA

- na standardní výstup
  - speaker nebo Bluetooth headset

- z raw zdroje
  - zvukový soubor dáme do res/raw
  - vytvoříme instanci MediaPlayer, zavoláme start()
  - zastavení přehrávání: stop()
  - později znovu přehrát: reset(), prepare(), start()
  - pauza: pause(), potom start()

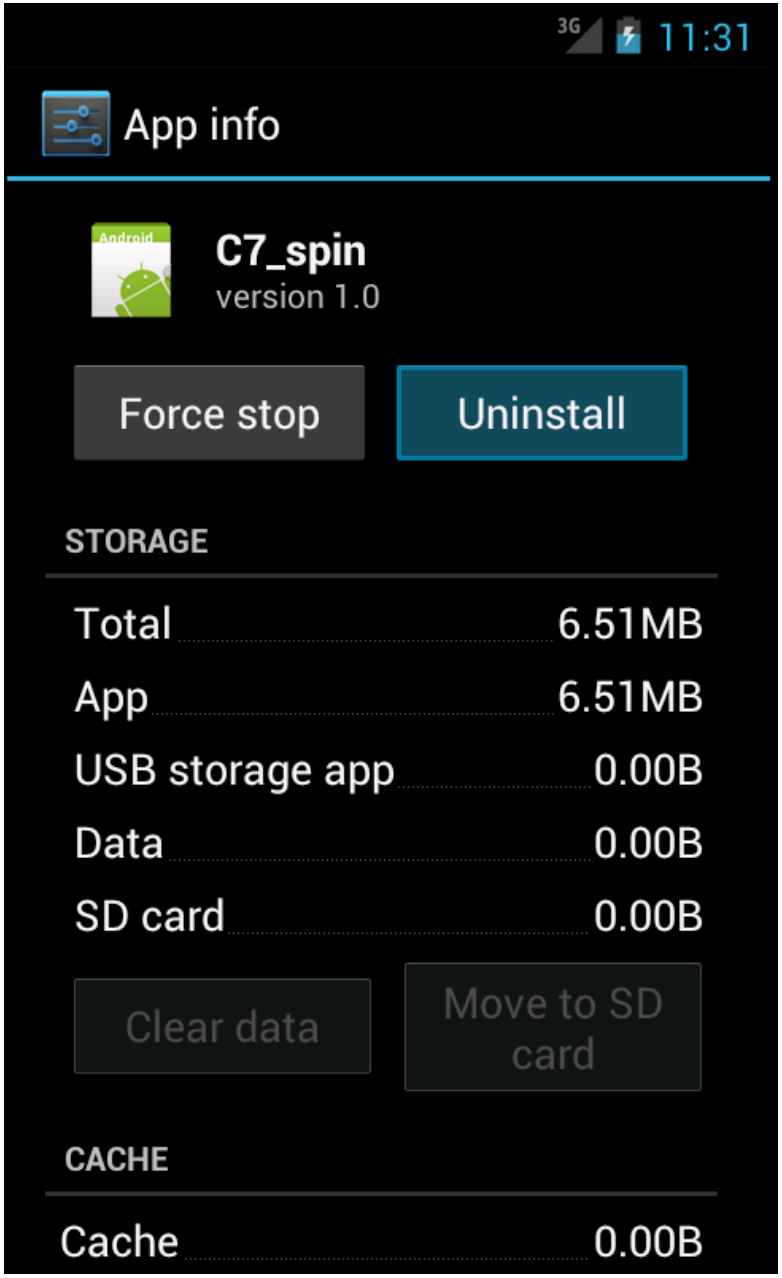
# PŘEHRÁVÁNÍ AUDIA

**z raw zdroje:**

```
MediaPlayer mp = MediaPlayer
        .create(context, R.raw.sound_file_1);
mp.start();
```

**ze souboru, streamu:**

```
MediaPlayer mp = new MediaPlayer();
mp.setDataSource(PATH_TO_FILE);
mp.prepare();
mp.start();
```




Ukázka aplikace C7_spin

součástí aplikace je i mp3 písnička uložená v res/raw/pisen1.mp3

všimněte si velikosti aplikace z původních  KB na 6.51MB

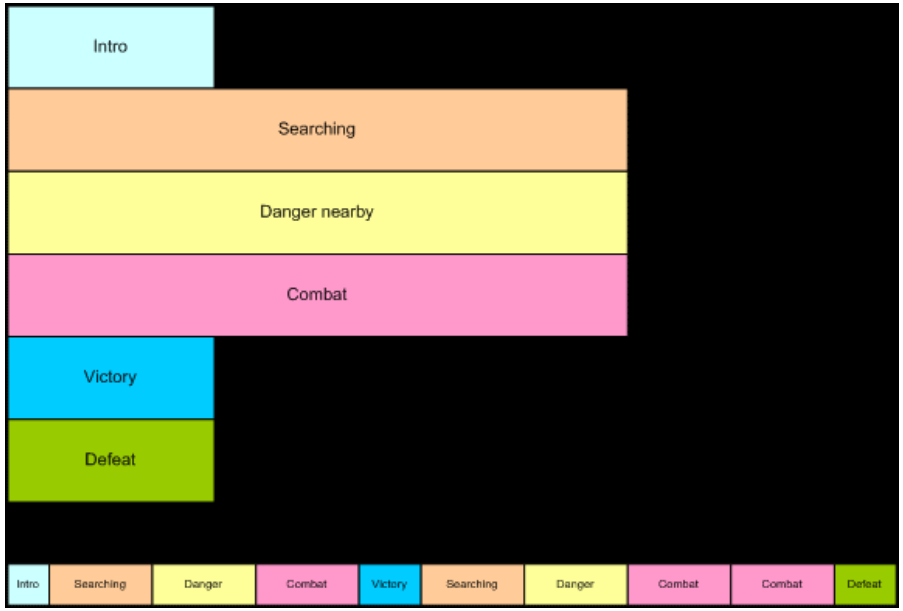
po spuštění aplikace hudba hraje
po stisku tlačítka HOME hraje stále

# PŘEHRÁVÁNÍ JET OBSAHU

JET

- interaktivní přehrávač hudby pro malá zařízení
- interaktivní soundtracky v MIDI
  - odpovídají různým situacím ve hře
- http://developer.android.com/guide/topics/media/jet/jetcreator_manual.html

intro
poblíž nebezpečí
bitva
vítězství
prohra



# PŘEHRÁVÁNÍ JET OBSAHU

```
JetPlayer myJet = JetPlayer.getJetPlayer();
myJet.loadJetFile("/sdcard/level1.jet");
byte segmentId = 0;

// queue segment 5, repeat once, use General MIDI, transpose by -1 octave
myJet.queueJetSegment(5, -1, 1, -1, 0, segmentId++);

// queue segment 2
myJet.queueJetSegment(2, -1, 0, 0, 0, segmentId++);

myJet.play();
```

# NAHRÁVÁNÍ AUDIA

- kompletní příklad v SDK
  http://developer.android.com/guide/topics/media/index.html

1. vytvořit instanci MediaRecorder
2. nastavení zdroje audia
   MediaRecorder.setAudioSource() .. typicky mikrofon
3. nastavení výstupního formátu
   MediaRecorder.setOutputFormat()
4. nastavení jména výstupního souboru
   MediaRecorder.setOutputFile()
5. nastavení audio encoderu
   MediaRecorder.setAudioEncoder()
6. příprava
   MediaRecorder.prepare()
7. spustíme nahrávání - MediaRecorder.start()
8. zastavení nahrávání - MediaRecorder.stop()
9. uvolnění zdrojů - MediaRecorder.release()

# NAHRÁVÁNÍ AUDIA

```
MediaRecorder mr = new MediaRecorder();
        mr.setAudioSource(MediaRecorder.AudioSource.MIC);

mr.setOutputFormat(MediaRecorder.OutputFormat.DEFAULT);
        mr.setOutputFile("/sdcard/music/moje");


mr.setAudioEncoder(MediaRecorder.AudioEncoder.DEFAULT);

try { mr.prepare(); } catch (Exception e) { }
mr.start();

mr.stop();
mr.release();
```

prozkoumejte možné hodnoty AudioSource, OutputFormat, AudioEncoder

# PROCESY A VLÁKNA

- nový Linuxový proces pro aplikaci
  - jedno vlákno (main thread)
- defaultně všechny komponenty stejné aplikace běží ve stejném procesu
- jakékoliv pomalé, blokující činnosti by se měli provádět v novém vlákně => nezpomalovat UI

# PROCESY - MANIFEST

<activity>, <service>, <receiver>,<provider>

mají atribut

android:process

možnosti:

- každá komponenta v samostatném procesu
- některé komponenty sdílejí proces
- lze i že komponenty různých aplikací běží ve stejném procesu (sdílejí linux user id a mají stejné certifikáty)

# PŘÍKLAD

AndroidManifest.xml file:
android:sharedUserId="aexp.share.sharedapp"

unikátní string sdílený mezi aplikacemi

atribut proces
(aplikace, aktivita, služba)

AndroidManifest.xml file.
android:process="aexp.share.sharedappprocess".

odkaz:

http://mylifewithandroid.blogspot.cz/2009/06/controlling-application-separation.html

# HIERARCHIE PROCESŮ

1. foregroud proces
2. visible proces
   - nemá komponenty v popředí, může ovlivnit to co je na obrazovce (je vidět „na pozadí“)
   - např. po zavolání onPause()
3. service proces
4. background proces
   - proces hostí aktivitu, která není aktuálně viditelná, zavolána onStop()
5. empty proces

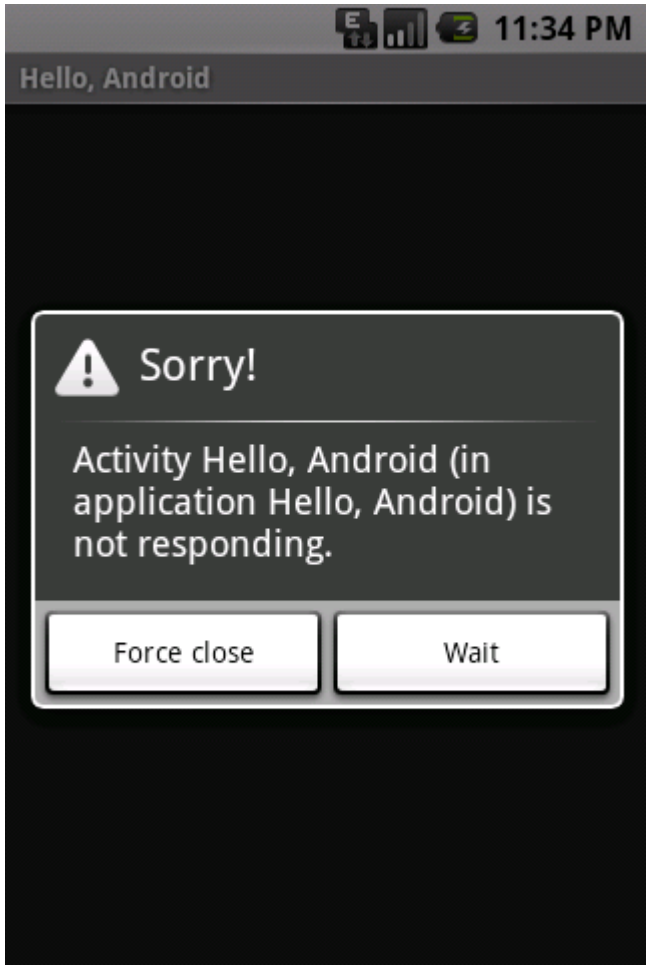
služba má přednost před aktivitou na pozadí

# VLÁKNA

- hlavní vlákno (main thread) – UI thread
- interakce komponent
- doručování událostí k UI widgetům
- běží zde i systémové callbacky onKeyDown()

- sítový přístup, databázové dotazy
  - blokuje UI
  - žádná událost nemůže být vyřízena, včetně kreslení (drawing events)
  - cca 5s – Application not responding dialog (ANR), ukončení Application Managerem

# ANR DIALOG




- pokračovat
- ukončit

zobrazí se, pokud aplikace nereaguje na uživatelský vstup

příklady příčin:

- čekání na síť
- generování náročných struktur v paměti
- výpočet další pozice ve hře

# STRICT MODE

zaveden od API 9

od API 11 nepovoluje síťové operace v UI vláknu

lze povolit:

```
StrictMode.ThreadPolicy policy = new StrictMode.
ThreadPolicy.Builder().permitAll().build();
StrictMode.setThreadPolicy(policy);
```

# PŘÍSTUP K UI

- UI není **thread-safe**
- nemůžeme manipulovat s UI z jiného vlákna – jen z UI vlákna

# PŘÍKLAD Z SDK - PROBLÉM

```
public void onClick(View v) {

    new Thread(new Runnable() {
        public void run() {
            Bitmap b = loadImageFromNetwork
                    ("http://example.com/image.png");
            mImageView.setImageBitmap(b);
        }
    }).start();
}
```

nové vlákno pro síťové operace (OK), ale modifikujeme ImageView z jiného než UI vlákna

# Přístup k UI z jiného vlákna

- Activity.runOnUiThread(Runnable)
  - akce je předána do fronty událostí UI vlákna
- View.post(Runnable)
  - přidání do message queue
- View.postDelayed(Runnable, long)
  - spuštění na UI vlákně po uplynutí času

# ÚPRAVA PŘÍKLADU

```
public void onClick(View v) {
    new Thread(new Runnable() {
        public void run() {
            final Bitmap bitmap = loadImageFromNetwork
                ("http://example.com/image.png");
            mImageView.post(new Runnable() {
                public void run() {
                    mImageView.setImageBitmap(bitmap);
                }
            });
        }
    }).start();
}
```

všimněte si 2 Runnable, 1 x start

manipulace s ImageView z UI vlákna

# Další možnosti

- využití Handleru
- AnsyncTask

# HANDLER

- vytvoření instance podtřídy **Handler**
- pro každou aktivitu potřebujeme 1
- může s ním komunikovat vlákno na pozadí
- provede operace ve vlákně UI

dva způsoby komunikace:

- zprávy
- objekty Runnable

# HANDLER - ZPRÁVY

- obtainMessage()
  - získáme z poolu objekt typu Message
  - prázdné nebo s parametry
- poslání zprávy:

| | |
|---|---|
| sendMessage() | Vloží zprávu do fronty |
| sendMessageAtFrontOfQueue() | Vloží zprávu na začátek fronty |
| sendMessageAtTime() | Vloží do fronty v určitý okamžik |
| sendMessageDelayed() | Po uplynutí doby (ms) |

# HANDLER – ZPRACOVÁNÍ ZPRÁVY

- handleMessage()
  - zavolá pro každou zprávu
  - rychle updatovat UI

# PŘ. HANDLER

```
public class Ukazka extends Activity {
ProgressBar pb1;
Handler ha = new Handler() {
    public void handleMessage(Message msg) {
            pb1.incrementProgressBy(10);
    }
};
….
```

# PŘ. HANDLER - WORKER VLÁKNO

```
Thread background = new Thread (
    new Runnable() {

   public void run() {
        …
        ha.sendMessage(handler.obtainMessage());
  }
}
```

# ASYNC TASK

- blokující operace je vykonána ve worker vlákně
- výsledky jsou aplikovány na UI vlákno

- udělat podtřídu **AsyncTask**()
- metoda doInBackground()
- metoda onPostExecute()
  - zpracování výsledků z doInBackground()
  - běží v UI vlákně

- task spustíme zavoláním execute z UI vlákna

# ÚPRAVA PŘÍKLADU S VYUŽITÍM ASYNC TASKU

- spuštění asynchronní úlohy:

```
public void onClick(View v) {

    new DownloadImageTask()
.execute("http://example.com/image.png");

}
```

# POKRAČOVÁNÍ …

```
private class DownloadImageTask extends
AsyncTask<String, Void, Bitmap> {

    /* vykoná ve worker vlákně */
    protected Bitmap doInBackground(String... urls)
    {
        return loadImageFromNetwork(urls[0]); }


    /* vykoná v UI vlákně, manipulace s UI */
    protected void onPostExecute(Bitmap result) {
        mImageView.setImageBitmap(result);
    }
}
```

# ASYNC TASK

- ve worker vlákně:
  - doInBackground()
- v UI vlákně:
  - onPreExecute()
  - onPostExecute()
  - onProgressUpdate()

- hodnota vrácená v doInBackground() je předána do onPostExecute()

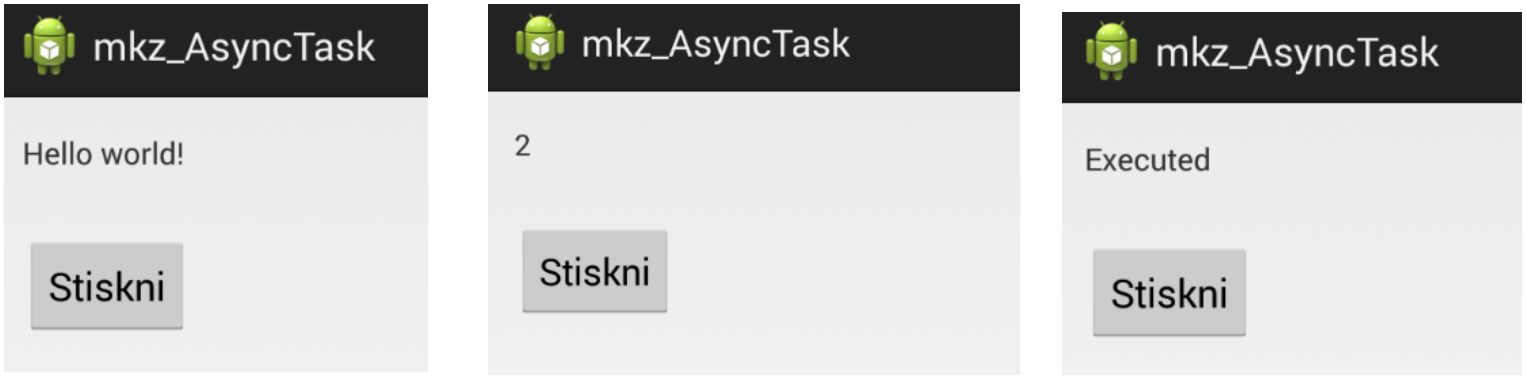
- kdekoliv v doInBackground() lze zavolat publishProgress() – vyvolá onProgressUpdate() v UI vlákně

# POUŽITÍ ONPROGRESSUPDATE()

```
private class LongOperation extends AsyncTask<String, Void, String> {
   int pocet = 0;   // průběžně se zvedá
   protected String doInBackground(String... params) {
            for(int i=0;i<5;i++) {
                …
                publishProgress();
            }
            return "Executed";
   }
   protected void onProgressUpdate(Void... values) {
            pocet++;
            TextView txt = (TextView) findViewById(R.id.textView1);
            txt.setText(String.valueOf(pocet));
    } … }
```

volá opakovaně **onProgressUpdate** a uživatel je informován, že se něco děje

# UKÁZKA PRŮBĚHU




1. před spuštěním asynctasku
2. průběžná informace v průběhu vykonávání
3. po vykonání práce na pozadí

# Meziprocesová komunikace (IPC)

- využití RPC
- aplikace zavolá bindService()

# DATABÁZE SQLITE

- součástí prostředí Android
  - libovolná aplikace může vytvářet databáze
- SQL
- rozdíl v typování dat
  - v CREATE TABLE specifikujeme datové typy sloupců => pro SQLite doporučení, ne závazek
  - např. vložení řetězce do sloupce INTEGER

# VYTVOŘENÍ DATABÁZE

- vytvoření podtřídy SQLiteOpenHelper

- konstruktor
  - zřetězit s konstruktorem SQLiteOpenHelper
  - kontext, název databáze, cursor factory, verzi databáze
- onCreate()
  - předá objekt typu SQLiteDatabase – naplnit tabulkami, daty, ..
- onUpgrade()
  - např. odstranění starých tabulek a vytvoření nových…

# VYTVOŘENÍ DATABÁZE

```
public class DatabaseHelper extends
SQLiteOpenHelper {

//konstruktor
//databaze DB_NAME, verze 1
public DatabaseHelper(Context context) {
  super(context, DB_NAME, null, 1);
}
```

# VYTVOŘENÍ DATABÁZE

```
public void onCreate(SQLiteDatabase db) {
db.execSQL("CREATE TABLE napoje (_id INTEGER PRIMARY
KEY AUTOINCREMENT, title TEXT, value REAL);");

ContentValues cv= new ContentValues();

cv.put("title", "Pivko");
cv.put("value",4.5);
db.insert("napoje", null, cv);

cv.put("title", "Medovina");
cv.put("value",20);
db.insert("napoje", null, cv);
}
```

# POUŽITÍ DATABÁZE

vytvoříme instanci SQLiteOpenHelper
a zavoláme getWriteableDatabase()
nebo getReadableDatabase()

```
db = (new DatabaseHelper(this))
        .getWritableDatabase();
```

- výsledkem je instance třídy SQLiteDatabase, s níž můžeme provádět dotazy, měnit data…
- po skončení práce: close()

# VYTVOŘENÍ TABULEK

- execSQL(), viz příklad dříve
  - primární klíč autoincrement integer
  - sloupec title
  - sloupec value
  - vytvoří index sloupce s primárním klíčem
  - další indexy lze CREATE INDEX
  - smazání indexu DROP INDEX
  - smazání tabulky DROP TABLE

# VKLÁDÁNÍ DAT

- použít execSQL() jako při vytváření tabulky
  - INSERT, UPDATE, DELETE
- použít metody insert(), update(), delete() objektu typu SQLiteDatabase
  - db.insert(tabulka, název sloupce, co se použije, je-li hodnota null ,hodnota)

# Čtení dat z databáze

- volání rawQuery()
  - provede SELECT přímo
- volání query()
  - vybudovat dotaz z jednotlivých složek

# RAWQUERY

- vrací instanci typu Cursor
  - obsahuje metody pro procházení výsledků dotazu

```
constantCursor = db.rawQuery(
   “SELECT _ID, title, value”+
    “FROM constants ORDER BY title”, null);
```

# QUERY()

- oddělené součásti příkazu SELECT
- vybuduje z nich dotaz
- parametry (co není třeba, null):
  - název tabulky
  - výčet sloupců
  - klauzule WHERE s pozičními parametry
  - výčet hodnot co nahradí poziční parametry
  - GROUP BY
  - ORDER BY
  - HAVING

# QUERY()

String[] columns = {“ID”, “inventory”};
String[] parms = {“abcdefg”};

Cursor result = db.query(“widgets”, columns,”name=?”, parms, null, null, null);

# BUILDERY

- SQLiteQueryBuilder
  - tvorba složitějších dotazů
  - zkombinovat výchozí hodnoty s předanými hodnotami
  - používá se pro dodavatele obsahu

# Použití kurzorů

- getCount()
  - kolik řádků je ve výsledné množině
- moveToFirst(), moveToNext()
- isAfterLast()
- getColumnNames(), getColumnIndex()
- getString(), getInt()
- requery()
  - znovu provést dotaz
- close()

# CURSOR

```
Cursor result = db.rawQuery("SELECT ID,
name, inventory FROM widgets");

result.moveToFirst();
while (!result.isAfterLast()) {
    int id = result.getInt(0);
    String name = result.getString(1);
    int inventory = result.getInt(2);
   …
   result.moveToNext();
}
result.close();
```

# NAPLNĚNÍ NAPŘ. LISTVIEW

```
ListAdapter adapter =
  new SimpleCursorAdapter (this,
     R.layout.row, constantsCursor,
     new String [] {“title”,”value”},
     new int[]  {R.id.title, R.id.value});

setListAdapter(adapter);
```

# NÁSTROJE

- součástí SDK konzolový program **sqlite3**
- lokace databáze:

*/data/data/balik/databases/nazev_databaze*

- kopírování db ze zařízení do počítače:
  adb pull
- upravená zpátky do zařízení:
  adb push

- Firefox SQLite Manager

# JAK DISTRIBUOVAT APLIKACI S DATABÁZÍ?

- vytvořit databázi externě
  - dát jí do adresáře assets
  - odsud databázi v programu zkopírovat
  - nevýhoda pro velké soubory: zabírá místo v apk a pak ještě vlastní vytvořená databáze
- vytvořit databázi programově

odkaz:
http://stackoverflow.com/questions/513084/how-to-ship-an-android-application-with-a-database

odkaz:

http://www.reigndesign.com/blog/using-your-own-sqlite-database-in-android-applications/

# POUŽITÁ LITERATURA


- Android SDK
- příklady z webu
- Mark L. Murphy: Android 2
- Grant Allen: Android 4 - Průvodce programováním mobilních aplikací