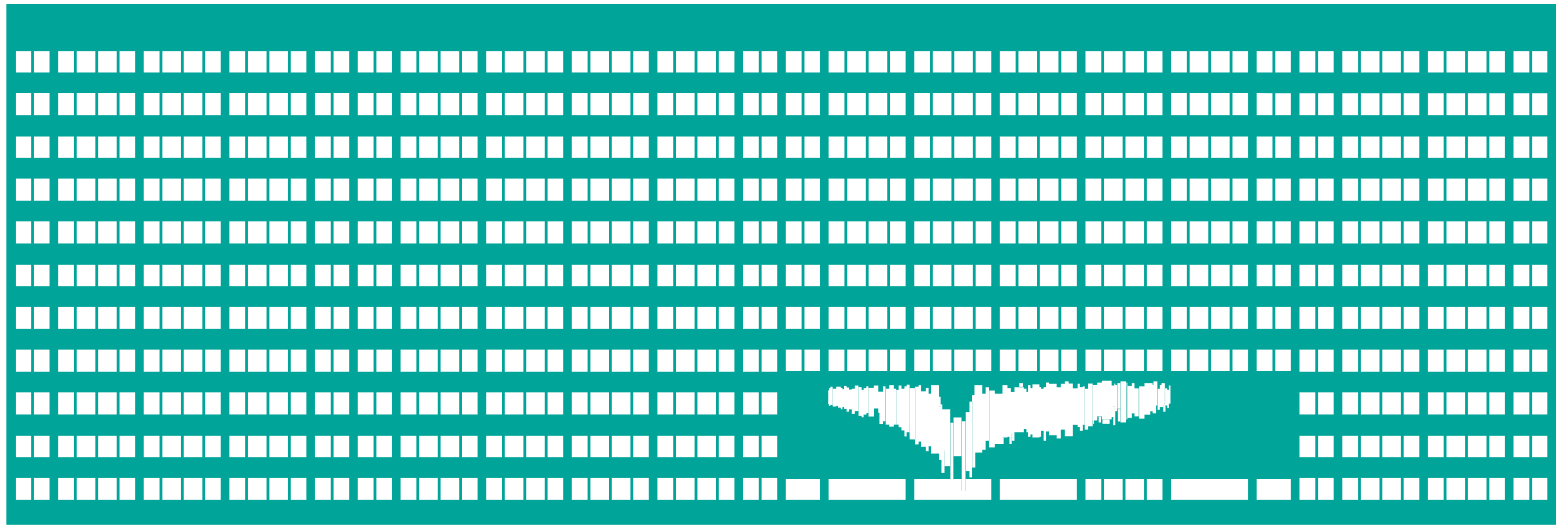
VŠB TECHNICKÁ UNIVERZITA OSTRAVA

VSB TECHNICAL UNIVERSITY OF OSTRAVA



www.vsb.cz

Android – Game Development

Michal Krumnikl

Introduction

- **Motivation from History**
  - „The Nintendo Game Boy and Game Boy Advance together account for about 200 million devices“
  - „Sony’s Playstation Portable has just passed 50 million devices“
  - „There are more than 100 million Nintendo DS Devices throughout the world“
- **Current Statistics**
  - „74c of every dollar spent in the app store was spent on mobile games“
  - „62% of people install a game on their phone within a week of owning it“
  - „Users spend more than 43% of their “smartphone time” playing games“
  - „21% of the Android and 25% of iOS apps downloaded are games. But 78% of the players worldwide belong to the Android game markets“
  - „51% of global revenues is attributed to mobile games, as opposed to 25% to console and 24% to PC“

Source: https://techjury.net/stats-about/mobile-gaming/

Introduction

- Game Types
  - Causal Games
  - Puzzles
  - Tower-Defense Games
  - Actions and Arcades
  - Strategic vs. Tactical
    - Tactical choices apply right now
    - Strategic choices affect course of game over medium or long term
  - Augmented Reality + VR
    - ARCore, Vuforia

Introduction

- **Definition**

  „A computer game is a software program in which one or more players make decisions through the control of game objects and resources, in pursuit of a goal.”

- **Play** - Interactions to elicit emotions
- **Game** - Object that provides rule-bound play
- **Frame** - The border of a game’s context
  - Inside the frame is in the game
  - Outside the frame is real life
- **Aesthetics** - Emotional responses during play

Introduction

- **Choice**
  - A question asked of the player
  - Choices must be non-trivial, with *upside* and *downside*
  - Often desirable and undesirable effects
  - Should relate to player goals
  - **Game should be *series* of interesting choices**
- **Outcome**
  - The end result of a given choice

- **Possibility space**
  - Represents the set of possible events
  - A “landscape” of choice and outcome

- **Well designed game should require strategy**

High Technical Demands

- **Technical implementation is hard**
  - Real-time and responsive (at least 25 frames per second)
  - Battery and resource demanding
  - Dynamic experience (many objects to draw and update)
- **Logic implementation**
  - Must be separated from technical implementation
  - Intuitive to player
- **Quality of content**
  - Story
  - Graphics
  - Sound

Storyline

- Best - not chunks of action with static facts
  - **Details revealed to audience – let them figure it out**
  - **Get emotional involvement from audience**
- Storytellers knew tricks for creating good stories long before Shakespeare – Game Designers should employ
  - **Obstacles, Plot Points, Foreshadowing …**

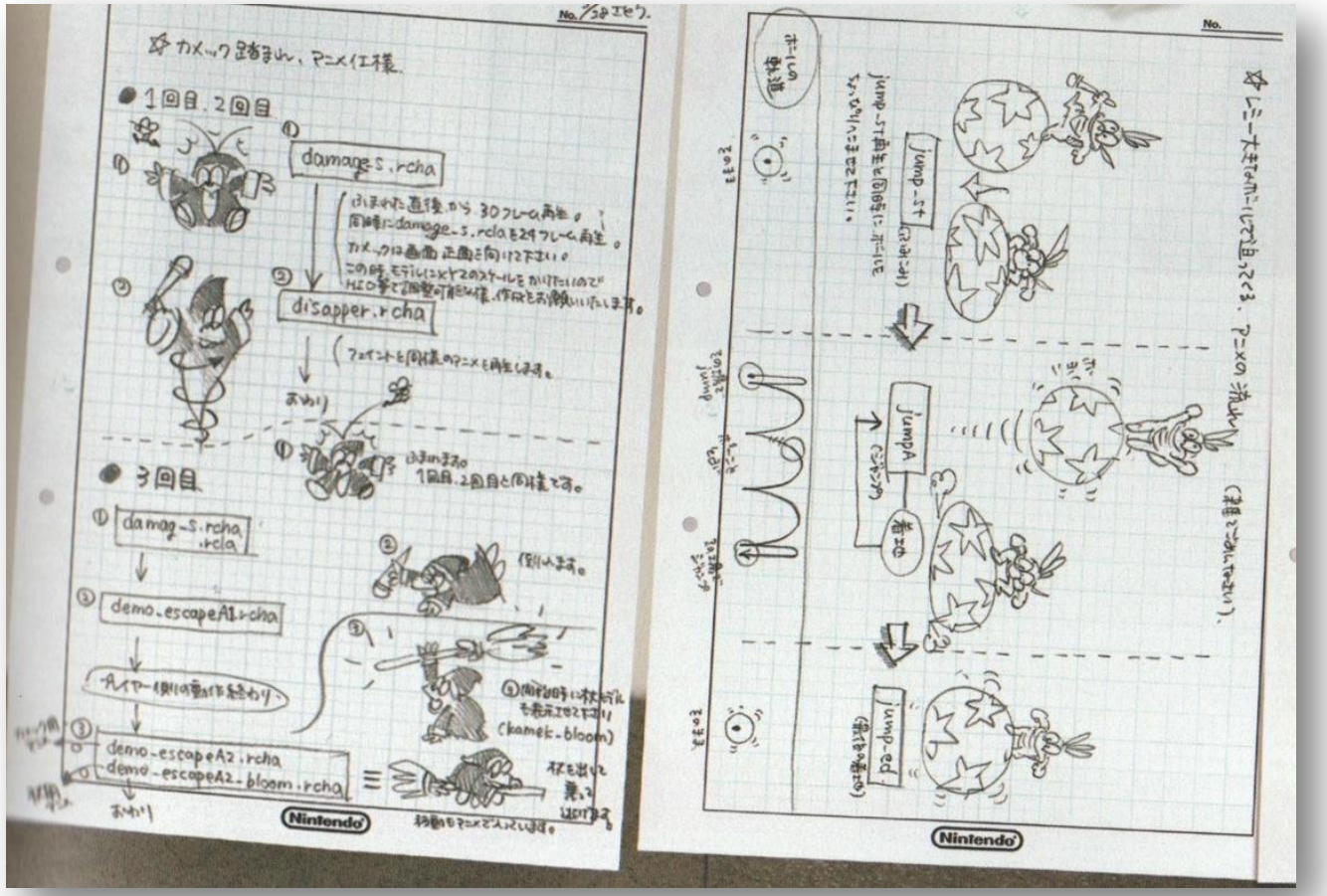
Old man runs to hero in inn. Says "Vampire on hill. You have to kill it."

vs.

Old man enters inn. Avoids hero. Purchases crucifix from another. Mumbles "you better have one if you are in these parts."

Vision

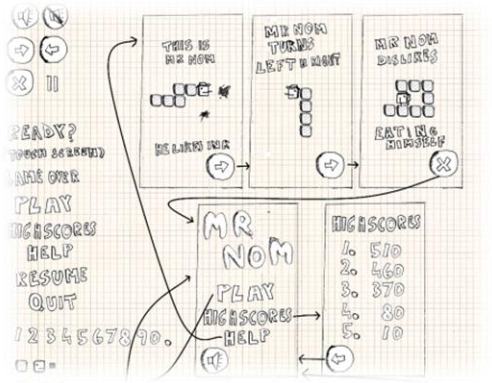
- **The "look" of the game**
- **Concept art**
  - Broad strokes, not pixel finished detail
  - Rough sketches of characters or settings

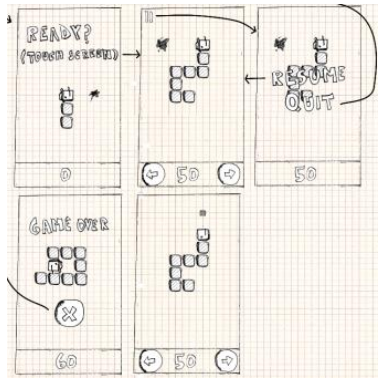


Source: https://www.reddit.com/r/nintendo/comments/1j19bv/rumor_miyamoto_is_working_on_a_new_1st_party/

From Design to Code

- **Simple games hints**
  - Display the name of your game
  - To **make things look consistent**, use a background
  - Players will usually want to actually play the game, so **let's throw in a Play button**.
  - Players want to **keep track of their progress** and awesomeness
  - Let's give the some help.
  - While your sound design will be lovely, some players might still prefer to play in silence.





Game Loop

- **Game Programming Patterns**
- **Decouple the progression of game time from user input and processor speed.**
- Allows the game to run smoothly regardless of a user's input or lack thereof

```
while( user_doesnt_exit ) {
        check_for_user_input();
        run_AI();
        move_enemies();
        resolve_collisions();
        draw_graphics();
        play_sounds();
}
```

Delta Timing

- Variably updating scenery based on the **elapsed time since the game last updated**.
  - Delta time gives us the length of the previous frame, which if you're running at 60 fps is 1/60 of a second, or about 16 milliseconds.
- Avoids the gameplay slowing down or speeding up depending on the complexity of what is happening at any given time.
- In network programming, keeps the game world of each computer in sync with the others, by making sure each client eventually sees the same activity at the same time, even if more time has passed since the last update for some clients than others.

Delta Timing

- Variably updating scenery based on the **elapsed time since the game last updated**.
  - Delta time gives us the length of the previous frame, which if you're running at 60 fps is 1/60 of a second, or about 16 milliseconds.

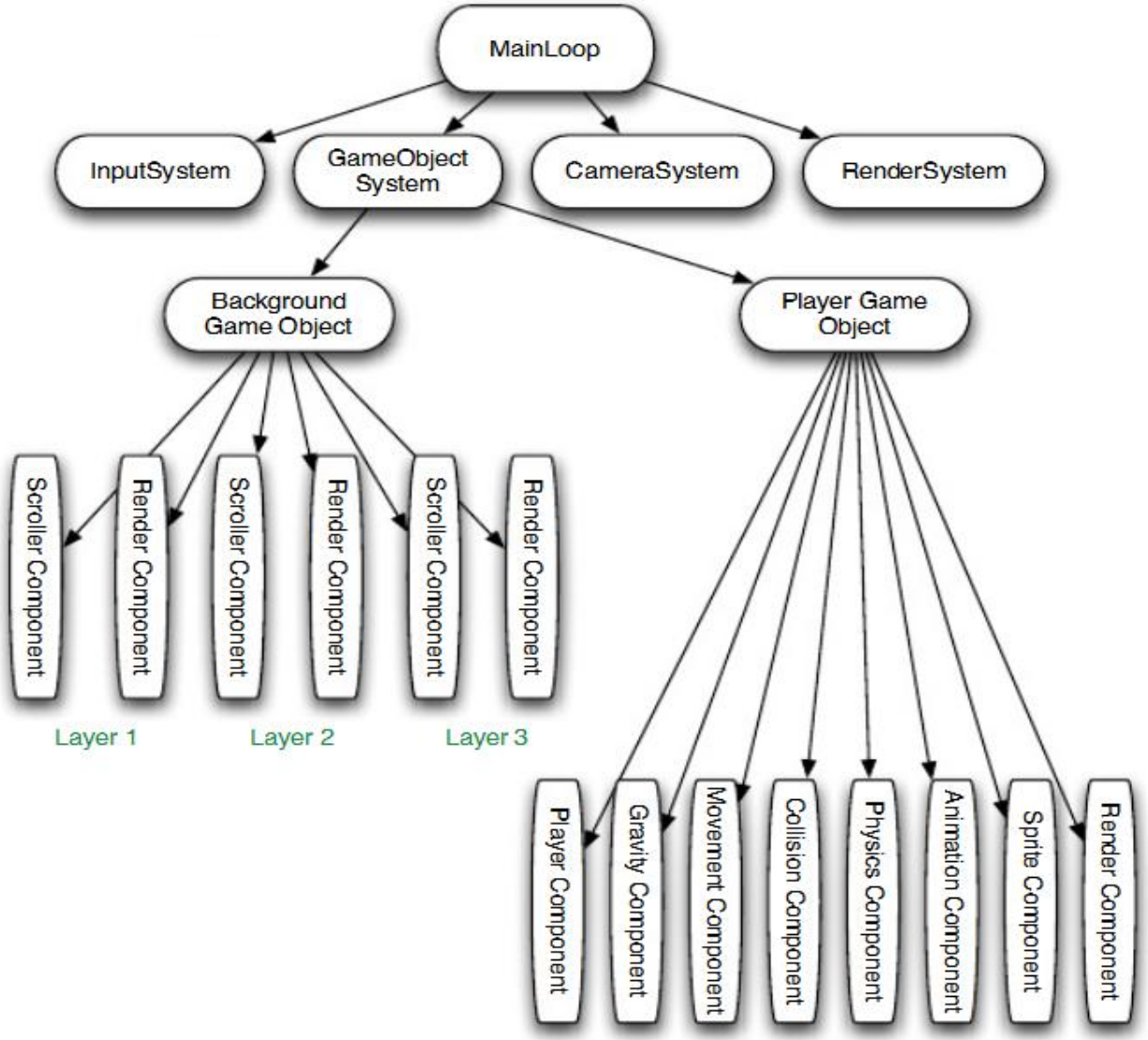
```
double lastTime = getCurrentTime();
while( user_doesnt_exit ) {
        double current = getCurrentTime();
        double elapsed = current - lastTime;
        check_for_user_input();
        run_AI(elapsed);
        move_enemies(elapsed);
        resolve_collisions();
        draw_graphics(elapsed);
        play_sounds();
}
```

Game Graph

- Possible advanced approach - "**game graph**" that can be traversed every frame, taking time and motion events as input and resulting in a list of things to draw to the screen.
  - **The root of the graph is the "main loop."**
  - **Children of the main loop get called once per frame.**
  - Children further down the tree might get called once per frame, depending on their parent.
  - "Game objects" are children of a "game manager" node, which only visits children within a certain activity bubble around the camera.
  - Game objects themselves are sub-graphs of "game components," each implementing a single characteristic or feature of the object.

Game Graph

- **The root of the graph is the “main loop.”**
- **Children of the main loop get called once per frame.**



AI for Games

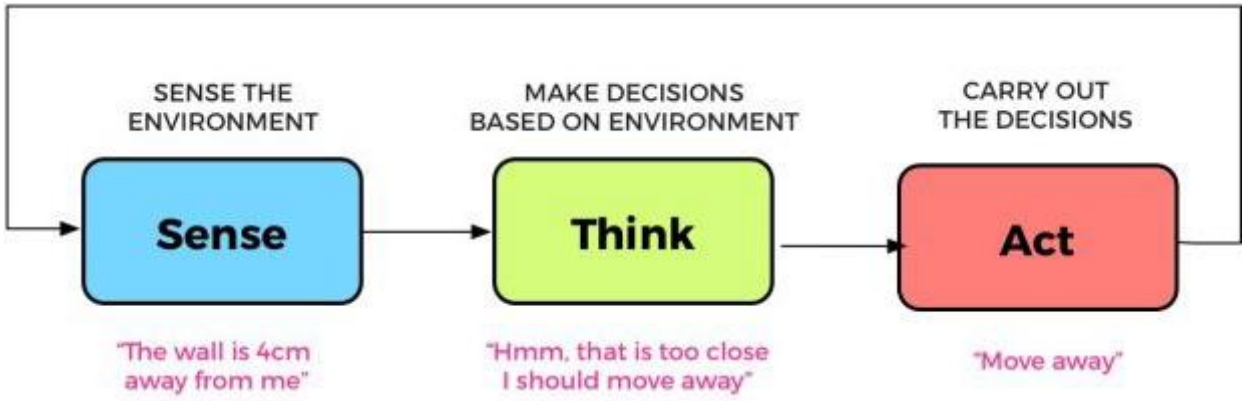
- **Must be smart, but purposely flawed**
  - Loose in a fun, challenging way
- No unintended weaknesses
  - No “golden path” to defeat
  - **Must not look dumb**
- **Must perform in real time** (CPU)
- Configurable by designers
  - Not hard coded by programmer
- “Amount” and type of AI for game can vary
  - RTS needs global strategy, FPS needs modeling of individual units at “footstep” level
  - RTS most demanding: 3 full-time AI programmers
  - Puzzle, street fighting: 1 part-time AI programmer

Game Agents

- Most AI focuses around game agent
  - think of agent as NPC (non-player character), enemy, ally or neutral
- Loops through: **sense-think-act cycle**
  - Acting is event specific, so talk about sense+think
- **Sensing**
  - Gather current world state: barriers, opponents, objects
  - Needs limitations : avoid “cheating” by looking at game data
  - Typically, same constraints as player (vision, hearing range)
  - Often done simply by distance direction (not computed as per actual vision)
  - Model communication (data to other agents) and reaction times (can build in delay)

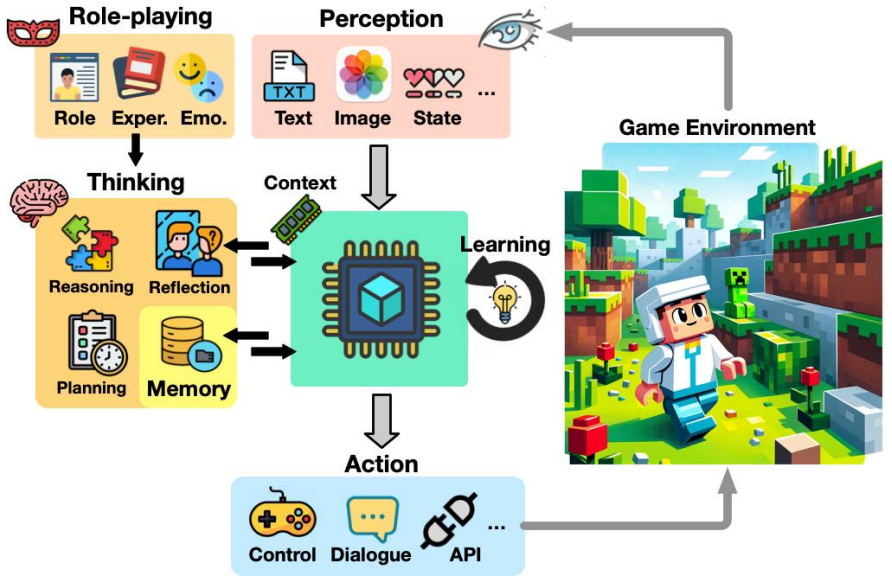
Game Agents

- **Thinking**
  - Evaluate information and make decision
  - As simple or elaborate as required
  - Two ways:
    - Precoded expert knowledge, typically hand-crafted if-then rules + randomness to make unpredictable
    - Search algorithm for best (optimal) solution
- **Search**
  - Look ahead and see what move to do next
- **Machine learning**
  - Evaluate past actions, use for future
  - Techniques show promise, but typically too slow
  - Need to learn and remember
- **Cheating Agent ?**



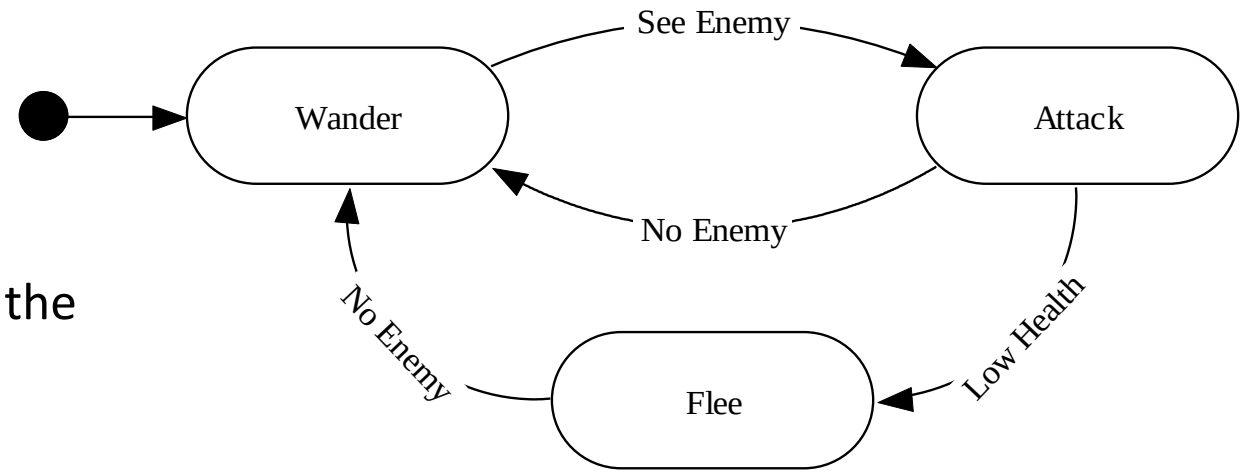
Agent Thinking

- **Expert Knowledge**
  - **Finite state machines, decision trees**, … (FSM most popular, details next)
  - Appealing since simple, natural, embodies common sense
    - Ex: if you see enemy weaker than you, attack. If you see enemy stronger, then go get help
  - Often quite adequate for many AI tasks
  - Trouble is, often does not scale
    - Complex situations have many factors
    - Add more rules, becomes brittle
- **Making agents stupid**
  - Many cases, easy to make agents dominate
- **Today use of LLMs**
  - https://arxiv.org/html/2404.02039v1



FSM in Games

- **Abstract model of computation**
- Formally:
  - Set of states
  - A starting state
  - An input vocabulary
  - A transition function that maps inputs and the current state to a next state



FSM in Games

- **Most common game AI software pattern**
  - Natural correspondence between states and behaviors
  - Easy to diagram
  - Easy to program
  - Easy to debug
  - Completely general to any problem
- Problems
  - Explosion of states
  - Often created with ad hoc structure

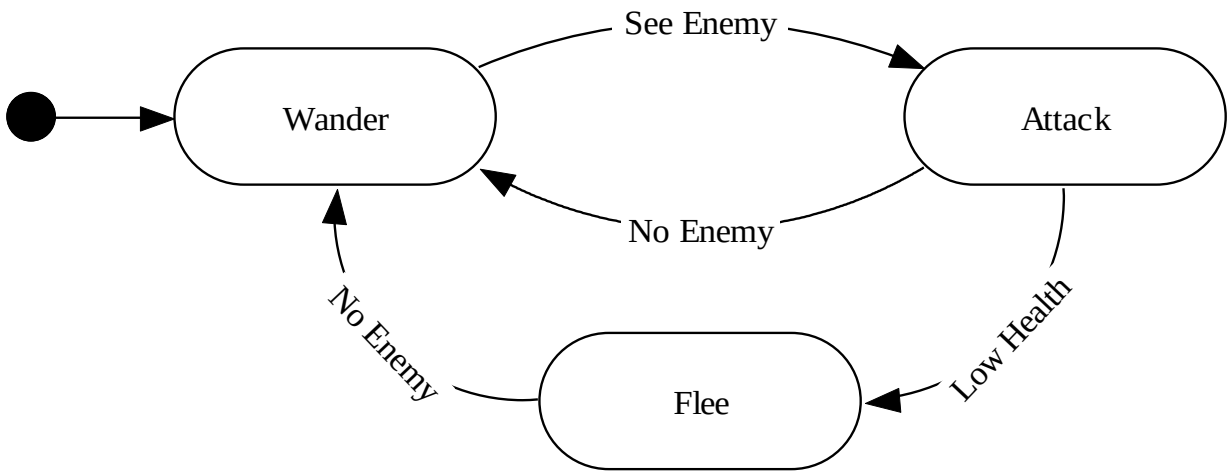
- **Three implementation approaches**
  - Hardcoded (switch statement)
  - Scripted
  - Hybrid Approach

Hardcoded FSM

```
void RunLogic( int * state ) {
    switch( state )
    {
        case 0:  //Wander
            Wander();
            if( SeeEnemy() )     { *state = 1; }
            break;
        case 1:  //Attack
            Attack();
            if( LowOnHealth() ) { *state = 2; }
            if( NoEnemy() )     { *state = 0; }
            break;
        case 2:  //Flee
            Flee();
            if( NoEnemy() )     { *state = 0; }
            break;
    }
}
```

Hardcoded FSM

```
void RunLogic( int * state ) {
    switch( state )
    {
        case 0:  //Wander
            Wander();
            if( SeeEnemy() )     { *state = 1; }
            break;
        case 1:  //Attack
            Attack();
            if( LowOnHealth() ) { *state = 2; }
            if( NoEnemy() )     { *state = 0; }
            break;
        case 2:  //Flee
            Flee();
            if( NoEnemy() )     { *state = 0; }
            break;
    }
}
```



FSM in Games

- **Many possible extensions to basic FSM**
  - OnEnter, OnExit
  - Timers
  - Global state, substates
  - Stack-Based (states or entire FSMs)
  - Multiple concurrent FSMs, Messaging

```
AgentFSM
{
    State( STATE_Wander )
        OnUpdate
            Execute( Wander )
            if( SeeEnemy )    SetState( STATE_Attack )
        OnEvent( AttackedByEnemy )
            SetState( Attack )
    State( STATE_Attack )
```

Android Game Development Kit

- The tools and libraries for Android game development - https://developer.android.com/games/agdk
- **Android Game Development Kit** (**AGDK)**
  - The core set of Android game development tools and libraries.
  - Includes C/C++ game integration, performance tuning, high-performance audio, and additional features for using or customizing game engines.
- **Android Game Development Extension** (**AGDE**)
  - Visual Studio extension that allows you to develop games in Visual Studio that include Android as a target platform.
- **Android GPU Inspector** (**AGI**)
  - An advanced graphics and system profiling tool that provides extensive tracing and analysis of individual frames.

Android Game Development Kit

- **AGDK** – Samples repository - https://github.com/android/games-samples
- **Libraries**
  - **Frame Pacing** - deliver frames at a consistent pace, and adjusts the pace based on the performance.
  - **Game Activity** - game development in C or C++ with access to Android Jetpack
  - **Game Controller** - accessing connections, features, device info, and input data.
  - **Game Text Input** - displays and hides the soft keyboard, and manages text updates.
  - **Memory Advice API (Beta)** - stay within safety limits for memory
  - **Oboe High-performance audio** - reduces audio latency
  - **Android Performance Tuner** - identifies performance issues
  - **Library wrapper (Beta)** - generate C/C++ code to access JAR libraries from your native app.

GameActivity - Jetpack library

- **Interacting with Android framework through the Java-side component. Passing app cycle commands, input events, and input text to the native side.**
  - GameActivity renders into a **SurfaceView**
  - For touch and key input events, GameActivity has a completely new implementation with the **android_input_buffer** interface, separate from the InputQueue that NativeActivity uses.
  - GameActivity is a derived class of AppCompatActivity, which lets you seamlessly use other Jetpack components. ActionBar, Fragment, and others are all available.
  - GameActivity adds text input functionality by integrating the **GameTextInput library**.
  - Apps derived from GameActivity are expected to build all three **parts of C/C++ code** into one library.

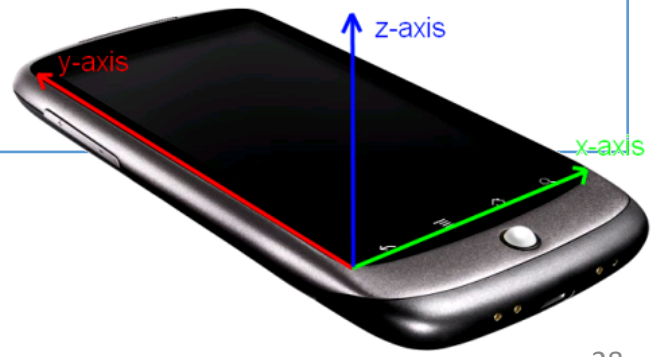
Android Implementation Specifics

- **Inputs**
  - Mobile device specifics
- **Graphics**
  - Screen size limitations
- **Power Management**
  - Battery life - https://developer.android.com/games/optimize/adpf/performance-hint-api
  - Thermal management - https://developer.android.com/games/optimize/adpf/thermal
- **Libraries**
  - 3rd party game libraries

Inputs

- Modes of operation
  - **Polling**
  - **Event-based handling**
- Touchscreen events
  - Touch down – when a finger is touched to screen
  - Touch drag – finger is dragged across the screen
  - Touch up – finger is lifted from screen
- Keyboard events
  - Key down – key is pressed
  - Key up – key is lifted
- Accelerometer

```
public boolean onTouchEvent(MotionEvent evt) {

   int action = evt.getAction();
   switch(action) {
         case MotionEvent.ACTION_DOWN:
                  Log.d(TAG, "Down");
                  break;
         case MotionEvent.ACTION_UP:
                  Log.d(TAG, "Up");
                  break;
         case MotionEvent.ACTION_MOVE:
                  Log.d(TAG, "Move");
                  cx = (int) evt.getX();
                  cy = (int) evt.getY();
         break;
   }
   invalidate();
   return true;
}
```



Graphics

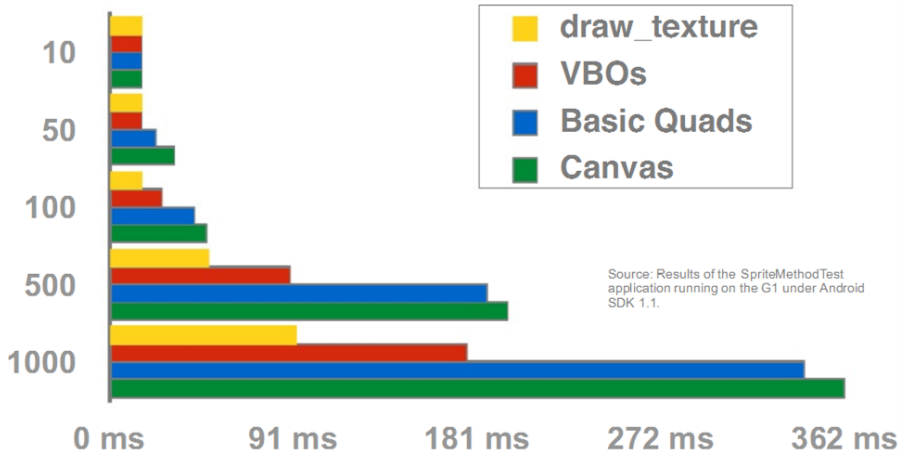
- **2D graphics library**
  - ***android.graphics.drawable***
  - Provides classes to manage a variety of visual elements that are intended for display only, such as bitmaps and gradients. These elements are often used by widgets as background images or simply as indicators (for example, a volume level indicator).
- **3D graphics library**
  - ***javax.microedition.khronos.opengles***
  - A package containing the Khronos OpenGL(R) ES interfaces. Although this specification incorporates portions of the OpenGL ES documentation, the documentation available at the Khronos Web site should always be considered definitive.

3D with OpenGL

- Android supports **OpenGL ES API**
  - Versions of OpenGL ES are loosely peered to versions of the primary OpenGL standard.
    - OpenGL ES 3.1 - This API specification is supported by Android 5.0 (API level 21) and higher.
  - The specific API provided by Android is similar to the J2ME JSR239 OpenGL ES API.
    - http://www.khronos.org/opengles/
- Android 7.0 adds support for **Vulkan**
  - The Android platform includes an Android-specific implementation of the Vulkan API specification from the Khronos Group.
  - Low-overhead, cross-platform API for high performance 3D graphics.
  - Vulkan overview
    - https://www.khronos.org/assets/uploads/developers/library/overview/Vk_201602_Overview_Feb16.pdf

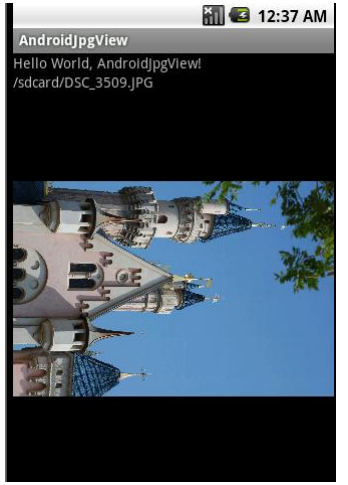
Drawing Methods Comparison

- **Canvas**
  - CPU-based 2D drawing. Used for most of the Android UI.
  - Fast for a small number of blits. (~ 10 sprites < 16 ms)
  - Very straightforward and easy to use.
- **OpenGL ES**
  - 2D and 3D drawing.
  - Hardware accelerated on some platforms (like the G1).
  - Scales to much more complex scenes than Canvas.
  - Various 2D drawing methods:
    - Quads with orthographic projection
    - VBO quads
    - draw_texture extension
  - OpenGL ES 1.0 is guaranteed



Drawing to a View

- **View object**
  - Draw simple graphics that do not need to change dynamically and are not part of a performance-intensive game.
  - Usually a static graphic or predefined animation, within an otherwise static application.
  - Draw to the background of a View or to the content of an ImageView in your layout.
- **ImageView**
  - Displays an arbitrary image, such as an icon.
  - The ImageView class can load images from various sources, takes care of computing its measurement from the image so that it can be used in any layout manager
  - provides various display options such as scaling and tinting.

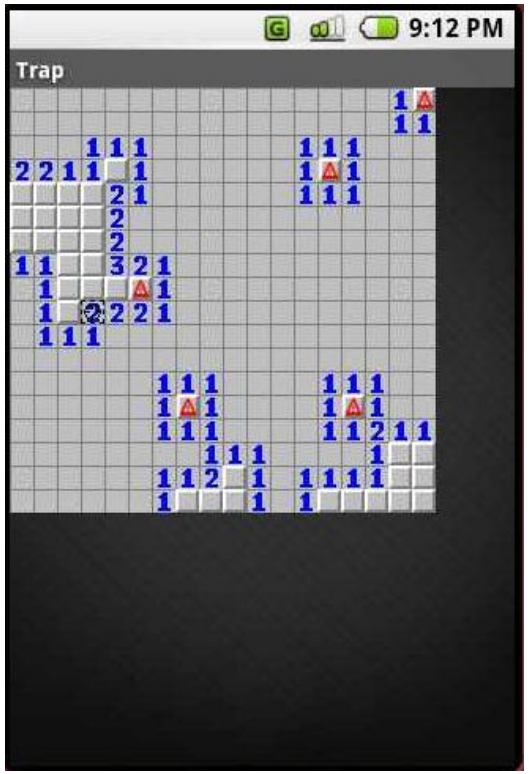


Drawing to a View

- Application does not require a significant amount of processing or frame-rate speed.
  - Extend the *View* class (or descendant thereof)
  - Define the *onDraw()* callback method. This method will be called by the Android framework to request that your View draw itself.
- The Android framework will only call *onDraw()* as necessary.
- Each time that your application is prepared to be drawn, you must request your View be invalidated by calling ***invalidate()***.
- In order to request an invalidate from a thread other than your main Activity's thread, you must call ***postInvalidate()***.

Drawing to a Canvas

- For application that needs to regularly re-draw itself.
- Video game should be drawing to the Canvas.
- There is more than one way to do this:
  - In the same thread as your UI Activity, wherein you create a custom View component in your layout, call ***invalidate()*** and then handle the ***onDraw()*** callback.
  - Or, in a separate thread, wherein you manage a ***SurfaceView*** and perform draws to the Canvas as fast as your thread is capable (you do not need to request *invalidate()*).



Draw with a Canvas

- A Canvas works for you as a pretense, or interface, to the actual surface upon which your graphics will be drawn — it holds all of your "draw" calls.
- Drawing is actually performed upon an underlying Bitmap, which is placed into the window.
  - In the event that you're drawing within the *onDraw()* callback method, the Canvas is provided for you and you need only place your drawing calls upon it.
  - You can also acquire a Canvas from *SurfaceHolder.lockCanvas()*, when dealing with a SurfaceView object.

```
Bitmap b = Bitmap.createBitmap(100, 100, Bitmap.Config.ARGB_8888);
Canvas c = new Canvas(b);
```

```
Spacer(
    modifier = Modifier
        .fillMaxSize()
        .drawBehind {
            // this = DrawScope
        }
)
```

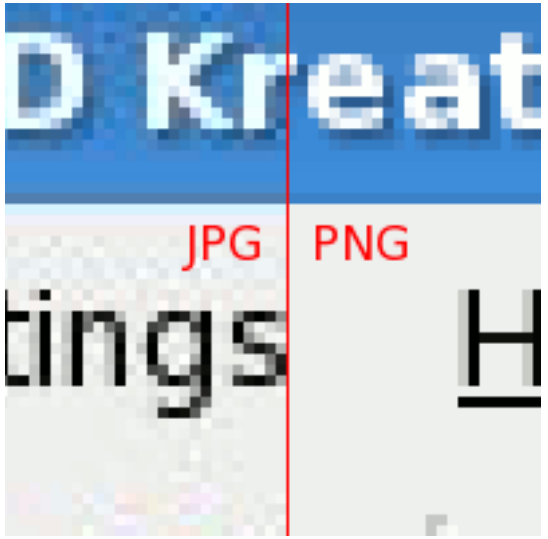
Draw with a Canvas

- It's recommended that you ultimately draw your final graphics through a Canvas offered to you by *View.onDraw()* or *SurfaceHolder.lockCanvas()*

```
Spacer(
    modifier = Modifier
        .fillMaxSize()
        .drawBehind {
            // this = DrawScope
        }
)
```

```
public class SplashScreen extends View {
    private Paint paint;
    private int cx;
    private int cy;
    private float radius;

    public SplashScreen(Context context) {
        super(context);
        paint = new Paint();
        paint.setColor(Color.GREEN);
        paint.setAntiAlias(true);
        cx = 200; cy = 200; radius = 50;
    }

    @Override
    protected void onDraw(Canvas canvas) {
        super.onDraw(canvas);
        canvas.drawCircle(cx,cy,radius,paint);
    }
}
```

Image Format and Compression



- Problem - Image 1024x1024, RGB888 = 3MB (in RGB565 = 2MB)
- **JPEG image format**
  - lossy compression for digital photography
  - typically achieves 10:1 compression with little perceptible loss in image quality.
  - maximum image size of 65535×65535
- **PNG image format**
  - lossless data compression
  - compression ration is around 2,7:1
- Same picture 402KB PNG, 35,7KB JPEG

Source 24bb png : 197kb

Indexed : 73kb

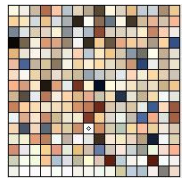
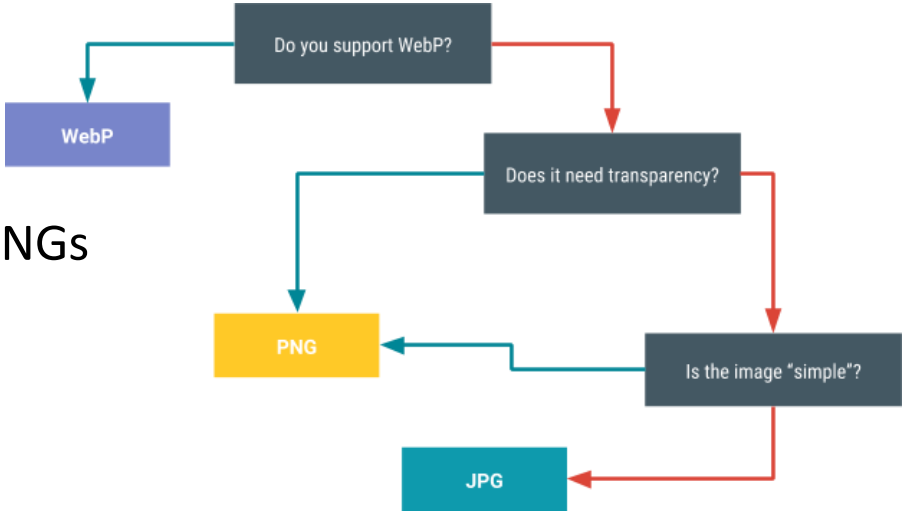


Image Format and Compression

- **WebP**
  - supported from Android 4.2.1 (API level 17)
  - WebP lossless image files are, on average, 26% smaller than PNGs
  - WebP lossy images are 25-34% smaller than comparable JPG



JPG : 201 k
PNG : 636 k

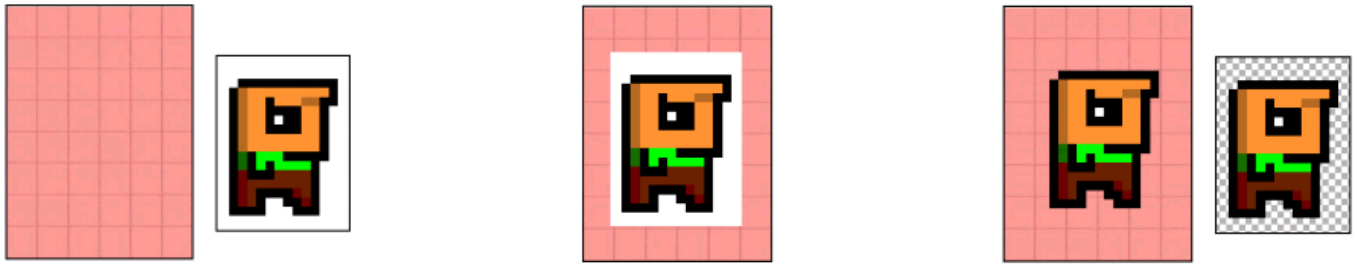
JPG : 82 k
PNG : 50 k

Image Format and Compression

- **Butteraugli**
  - Library to test an image's **Psychovisual Error Threshold**
  - for PNG, JPG, WebP lossy, and WebP lossless compressions

- **Lowest quality setting before a user perceives noticeable distortion in the resulting image**

Alpha Compositing and Blending



- 24-bit RGB triplet in a 32-bit integer - there are 8 unused bits in 32-bit integer. **Alpha value** specify the translucency of a pixel from 0 to 255.
  - *red = src.red * src.alpha + dst.red * (1 – src.alpha)*
- ARGB8888, BGRA8888 or ARGB4444
- The JPEG format does not support storing alpha values per pixel. Use the PNG format in that case.

Canvas and Full-Screen Mode

- Canvas resolution

```
int width = canvas.getWidth();
int height = canvas.getHeight();
```

- Get rid of the activity's title bar.

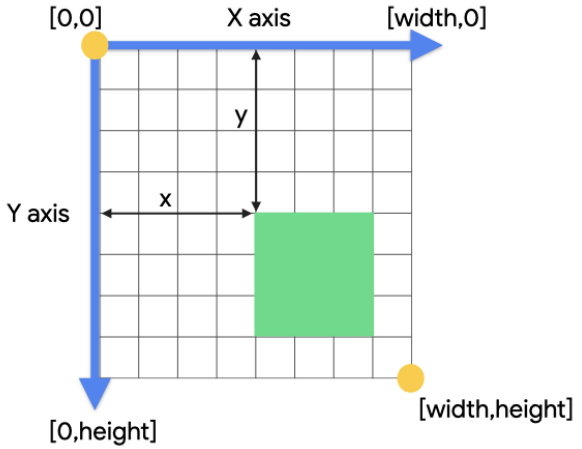
```
requestWindowFeature(Window.FEATURE_NO_TITLE);
```

- Activity go full-screen and eliminate the notification bar

```
getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
WindowManager.LayoutParams.FLAG_FULLSCREEN);
```

- Superclass method onCreate() is called after we execute these two methods.

```
private void setToFullScreen() {
   this.setSystemUiVisibility(View.SYSTEM_UI_FLAG_LOW_PROFILE
    | View.SYSTEM_UI_FLAG_FULLSCREEN
    | View.SYSTEM_UI_FLAG_LAYOUT_STABLE
    | View.SYSTEM_UI_FLAG_IMMERSIVE_STICKY
    | View.SYSTEM_UI_FLAG_LAYOUT_HIDE_NAVIGATION
    | View.SYSTEM_UI_FLAG_HIDE_NAVIGATION);
}
```



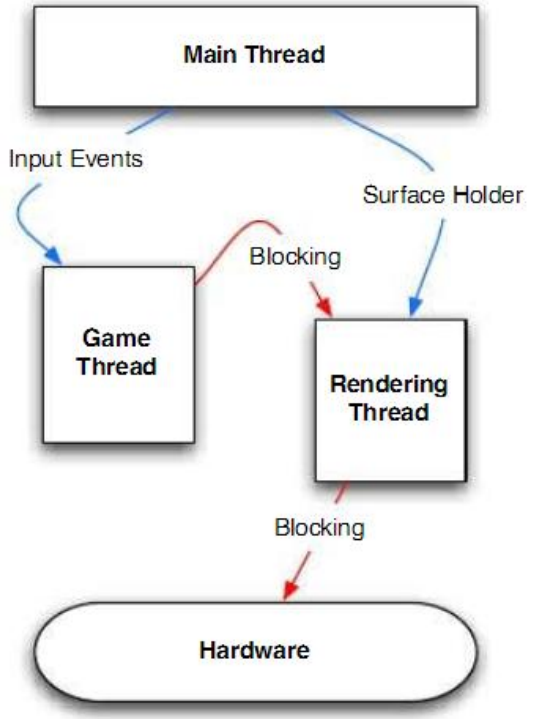
SurfaceView

- Surface is an **abstraction of a raw buffer** that is used by the screen compositor for rendering that specific View. The Surface can be hardware accelerated.
- Provides a dedicated drawing **surface embedded inside of a view hierarchy**.
- You can control the format of this surface and, if you like, its size; the *SurfaceView* takes care of placing the surface at the correct location on the screen
  - The surface is Z ordered so that it is behind the window holding its *SurfaceView*; the *SurfaceView* punches a hole in its window to allow its surface to be displayed. The view hierarchy will take care of correctly compositing with the Surface any siblings of the *SurfaceView* that would normally appear on top of it. This can be used to place overlays such as buttons on top of the Surface.

Drawing on the SurfaceView

- The SurfaceView is a special subclass of View that offers a dedicated drawing surface within the View hierarchy.
  - The aim is to offer this **drawing surface to an application's secondary thread**, so that the application isn't required to wait until the system's View hierarchy is ready to draw.
  - Instead, a secondary thread that has reference to a *SurfaceView* can draw to its own Canvas at its own pace.
- To begin, you need to create a new class that extends *SurfaceView*. The class should also implement ***SurfaceHolder.Callback***.
  - This subclass is an interface that will notify you with information about the underlying Surface, such as when it is created, changed, or destroyed.

Drawing on the SurfaceView



- Acquire an instance of the SurfaceHolder class
  - SurfaceHolder holder = surfaceView.getHolder();
- The SurfaceHolder is a wrapper around the Surface, and does some bookkeeping for us. It provides two methods:
  - ***Canvas SurfaceHolder.lockCanvas();***
    - The method locks the Surface for rendering and returns a nice Canvas instance we can use.
  - ***SurfaceHolder.unlockAndPost(Canvas canvas);***
    - The method unlocks the Surface again and makes sure that what we've drawn via the Canvas gets displayed on the screen.

Drawing on the SurfaceView

```
class RenderView extends SurfaceView
implements Runnable {

        Thread renderThread = null;
        SurfaceHolder holder;
        volatile boolean running = false;

 public RenderView(Context context) {
        super(context);
        holder = getHolder();
 }

 public void resume() {
        running = true;
        renderThread = new Thread(this);
        renderThread.start();
 }
```

```
public void run() {
      while(running) {
            if (!holder.getSurface().isValid())
                  continue;
            Canvas canvas = holder.lockCanvas();
            canvas.drawRGB(255, 0, 0);
            holder.unlockCanvasAndPost(canvas);
      }
}
public void pause() {
      running = false;
            while(true) {
                  try { renderThread.join(); }
                  catch (InterruptedException e) {
                   // retry }
}}
```

“Standard” Android 2D Graphics

- Packages with common classes used for drawing and animating in two-dimensions.
- **android.graphics.drawable**
  - Provides classes to manage a variety of visual elements that are intended for display only, such as bitmaps and gradients.
- **android.view.animation**
  - Provides classes that handle tweened animations.

Drawable

- **BitmapDrawable**
  - A Drawable that wraps a bitmap and can be tiled, stretched, or aligned. You can create a BitmapDrawable from a file path, an input stream, through XML inflation, or from a Bitmap object.
- **ShapeDrawable**
  - A Drawable object that draws primitive shapes. Takes a Shape object and manages its presence.
- **PictureDrawable**
  - Drawable subclass that wraps a Picture, allowing the picture to be used whereever a Drawable is supported.
- **LayerDrawable**
  - A Drawable that manages an array of other Drawables. These are drawn in array order, so the element with the largest index will be drawn on top.

Drawable

- Three ways to define and instantiate a Drawable:
  - using an image saved in your project resources;
  - using an XML file that defines the Drawable properties;
  - using the normal class constructors
- Creating from resource images
  - A simple way to add graphics to your application is by referencing an image file from your project resources. Supported file types are PNG (preferred), JPG (acceptable) and GIF (discouraged)
- Creating from resource XML

Vector Drawable/ShapeDrawable

- Draw primitive shapes and style them in any way
- A *ShapeDrawable* is an extension of *Drawable*
- A *ShapeDrawable* takes a Shape object and manages its presence on the screen.
- If no Shape is given, then the ShapeDrawable will default to a *RectShape*.
  - PathShape, RectShape
  - ArcShape, OvalShape, RoundRectShape
- ShapeDrawable has its own draw() method, so you can create a subclass of View that draws the ShapeDrawable during the View.onDraw() method.

ShapeDrawable

```
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <corners android:radius="8dp" />
    <gradient
        android:startColor="#000000"
        android:endColor="#0000dd"
        android:angle="45"/>
    <padding android:left="7dp"
        android:top="7dp"
        android:right="7dp"
        android:bottom="7dp" />
</shape>
```


```
<TextView
    android:background="@drawable/gradient_box"
    android:layout_height="wrap_content"
    android:layout_width="wrap_content" />
```

ShapeDrawable

```
<!-- res/drawable/battery_charging.xml -->
   <vector xmlns:android="http://schemas.android.com/apk/res/android"
       android:height="24dp"
       android:width="24dp"
       android:viewportWidth="24.0"
       android:viewportHeight="24.0">
      <group
            android:name="rotationGroup"
            android:pivotX="10.0"
            android:pivotY="10.0"
            android:rotation="15.0" >
         <path
           android:name="vect"
           android:fillColor="#FF000000"
           android:pathData="M15.67,4H14V2h-4v2H8.33C7.6,4 7,4.6 7,5.33V9h4.93L13,7v2h4V5.33C17,4.6 16.4,4 15.67,4z"
           android:fillAlpha=".3"/>
         <path
           android:name="draw"
           android:fillColor="#FF000000"
           android:pathData="M13,12.5h2L11,20v-5.5H9L11.93,9H7v11.67C7,21.4 7.6,22 8.33,22h7.33c0.74,0 1.34,-0.6 1.34,-
   1.33V9h-4v3.5z"/>
      </group>
   </vector>
```

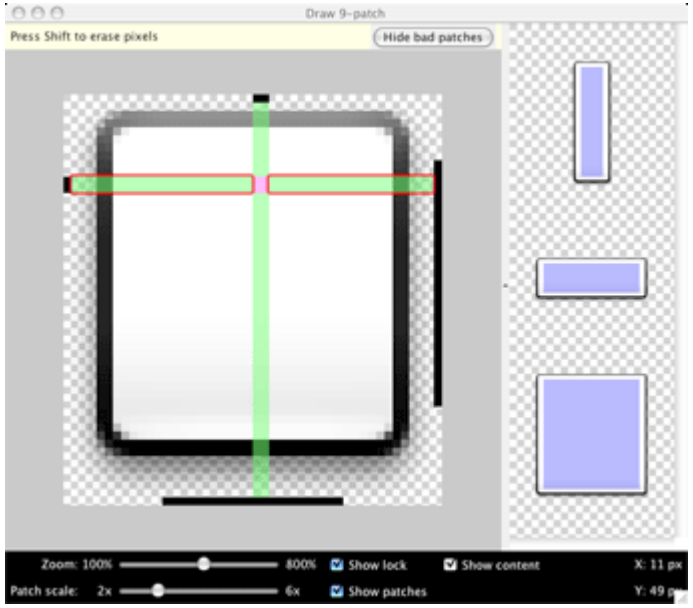
NinePatchDrawable

- Stretchable bitmap, which Android will automatically resize to accommodate the contents of the View in which you have placed it as the background.
- **PNG that includes an extra 1-pixel-wide border.**
- It must be saved with the extension **.9.png**, and saved into the /res/drawable/ directory of your project.
- The border is used to define the stretchable and static areas of the image. You indicate a stretchable section by drawing one (or more) 1-pixel-wide black line(s) in the left and top part of the border
- You can also define an optional drawable section of the image (effectively, the padding lines) by drawing a line on the right and bottom lines.

NinePatchDrawable

- The pink rectangle in the bottom image identifies the region in which the contents of the View are allowed. If the contents don't fit in this region, then the image will be stretched so that they do.

- The Draw 9-patch tool is a WYSIWYG graphics editor



Tween Animation

- Animation package provides all the classes
- Perform a **series of simple transformations on the contents of a View object**
  - position, size, rotation, and transparency
  - the animation XML file belongs in the /res/anim/
  - <alpha>, <scale>, <translate>, <rotate>, interpolator element, or <set>
- A sequence of animation instructions defines the tween animation, defined by either XML or Android code.
- The animation instructions define the transformations that you want to occur, when they will occur, and how long they should take to apply.
- Transformations can be sequential or simultaneous

Frame Animation

- *AnimationDrawable* is the basis for frame animations.
- **Sequence of different images**, played in order
- Usually a XML file that lists the frames that compose the animation.

```
<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>
```

- The start() method called on the AnimationDrawable cannot be called during the onCreate() method of your Activity, because the AnimationDrawable is not yet fully attached to the window.

OpenGL View

- ***GLSurfaceView*** is responsible for handling the view related functions of the OpenGL application. All the user inputs should be received by this class.
- https://developer.android.com/develop/ui/views/graphics/opengl/about-opengl

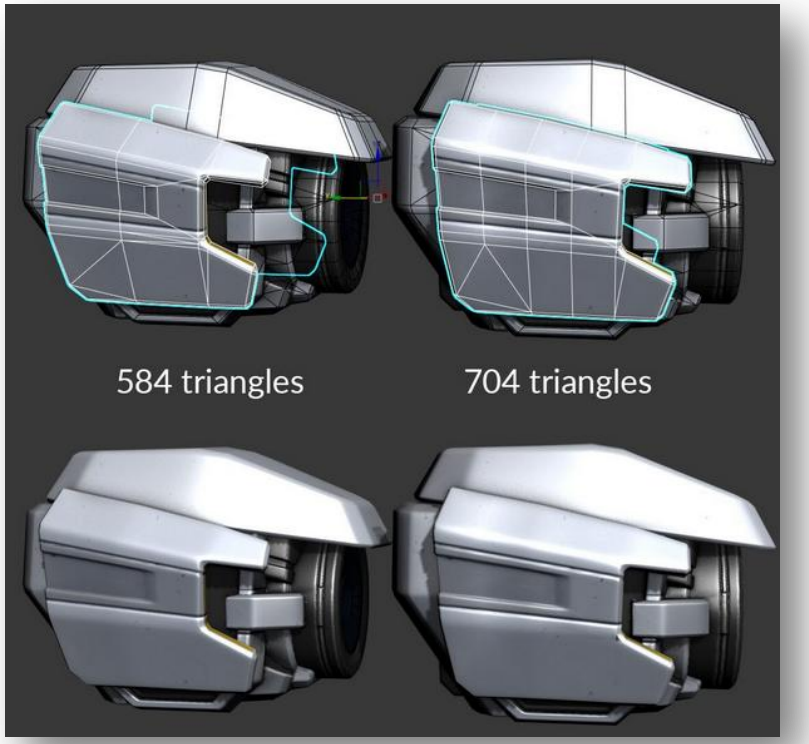
```
import android.content.Context;
import android.opengl.GLSurfaceView;

public class MainSurfaceView  extends GLSurfaceView {
  private MainRenderer mRenderer;

  public MainSurfaceView(Context context)
  {
    super(context);
    this.mRenderer = new MainRenderer(getContext());
    setRenderer(this.mRenderer);
  }
}
```

OpenGL View

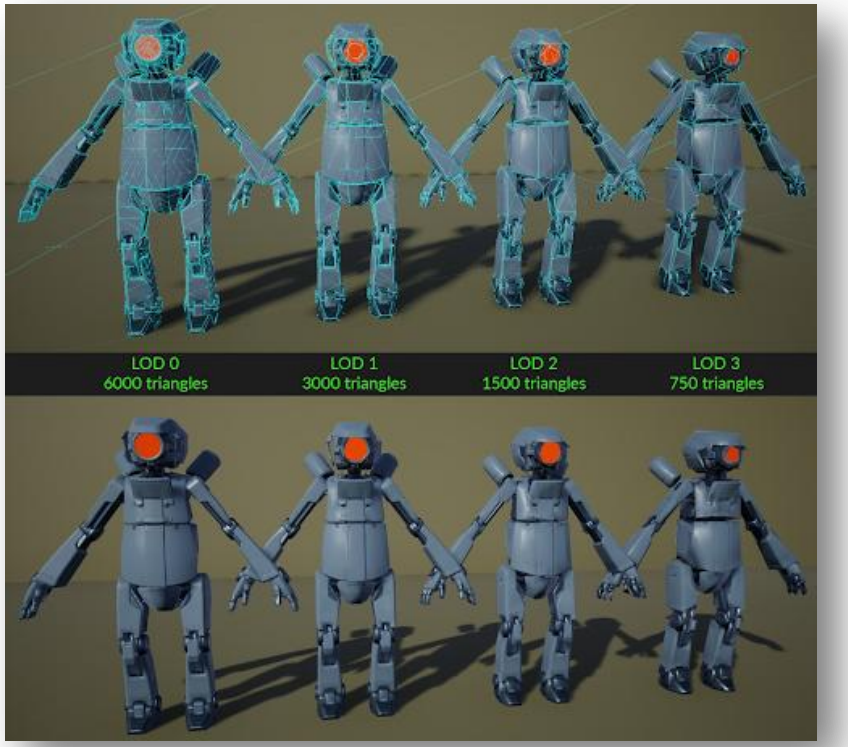
- Following example handles window resize in *onSurfaceChanged*, and we should place rendering calls in *onDrawFrame*.

```
public void onSurfaceCreated(GL10 gl, EGLConfig config)
{
 gl.glClearColor(0.5F, 0.5F, 1.0F, 1.0F);
}

public void onDrawFrame(GL10 gl)
{
 gl.glClear(gl.GL_COLOR_BUFFER_BIT | gl.GL_DEPTH_BUFFER_BIT );
}

public void onSurfaceChanged(GL10 gl, int width, int height
{
 gl.glViewport(0, 0, width, height);
}
```

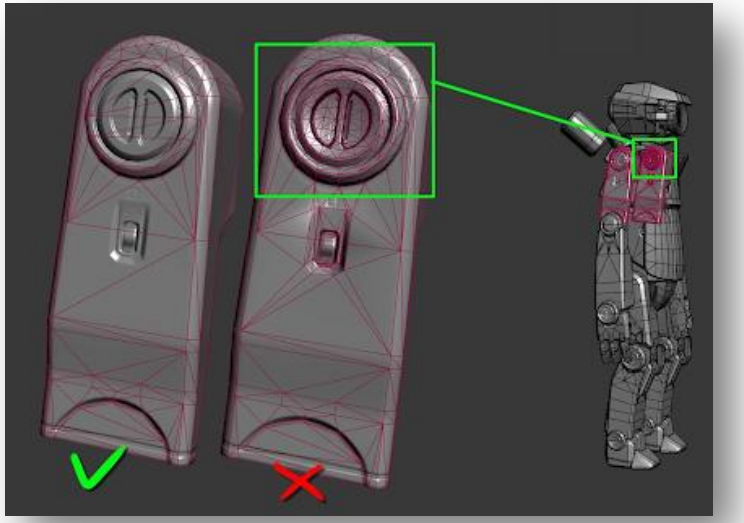
3D Optimization



Triangle reduction



Level of detail



Micro triangles removal

Audio and Video

- **Built-in encoders/decoders** for common media types
- You can play audio or video from media files
  - application's resources (raw resources)
  - standalone files in the filesystem
  - data stream arriving over a network connection
- List of supported **Core Media Formats**

- To play audio or video use the *MediaPlayer* class.
- To record audio or video use the *MediaRecorder* class.
  - Emulator doesn't have hardware to capture A/V

Audio and Video Playing File

- Create an instance of *MediaPlayer*, referencing that resource using *MediaPlayer.create*, and then call *start()*
- To stop playback, call *stop()*
- If you wish to later replay the media, then you must *reset()* and *prepare()* the MediaPlayer object before calling start() again
  - *create()* calls *prepare()* the first time

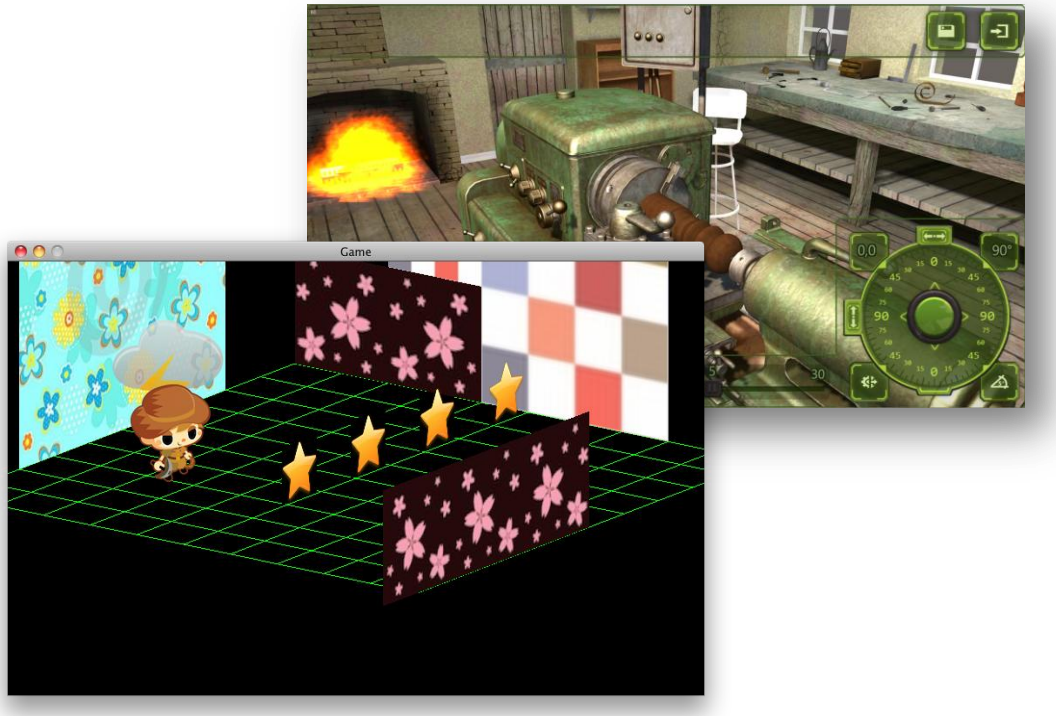
- Playing from a File or Stream
- Call *setDataSource()* with a String containing the path (local filesystem or URL) to the file you want to play

Game Libraries

- **Less time spent in technical details**
  - Less native calls exposed
  - Handling of devices
  - Automatic loading and handling resources
- **More time available for**
  - Game-specific logic programming
  - Developing content
- **Helps with portability**
  - Hardware abstraction
  - Platform abstraction

LibGDX

- Cross-platform
  - Windows, Linux, Mac, Android, iOS
- Free and fully open-source
- A Complete framework, but not a game engine
  - Low-level 3D with OpenGL
  - High-level 3D
  - High-level 2D
  - Physics with box2D
  - Math classes for 3D
- Utilities, Tools & Extensions



LibGDX

- **Drawing text using LibGDX and OpenGL**

```
BitmapFont font;
SpriteBatch batch;

public void create() {
        font = new BitmapFont();
        batch = new SpriteBatch();
}

public void render() {
        Gdx.graphics.getGL10().glClear(GL10.GL_COLOR_BUFFER_BIT);
        batch.begin();
        font.draw(batch, "Hello World!", 10, 10);
        batch.end();
}
```

LibGDX

- **Drawing text without LibGDX in OpenGL**

```
Bitmap bitmap = Bitmap.createBitmap(256, 256, Bitmap.Config.ARGB_4444);
Canvas canvas = new Canvas(bitmap);
bitmap.eraseColor(0);
Drawable background = context.getResources().getDrawable(R.drawable.background);
background.setBounds(0, 0, 256, 256);
background.draw(canvas); // draw the background to our bitmap
Paint textPaint = new Paint();
textPaint.setTextSize(32);
textPaint.setAntiAlias(true);
textPaint.setARGB(0xff, 0x00, 0x00, 0x00);
canvas.drawText("Hello World", 16,112, textPaint);
gl.glGenTextures(1, textures, 0);
gl.glBindTexture(GL10.GL_TEXTURE_2D, textures[0]);
gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_MIN_FILTER, GL10.GL_NEAREST);
gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_MAG_FILTER, GL10.GL_LINEAR);
gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_WRAP_S, GL10.GL_REPEAT);
gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_WRAP_T, GL10.GL_REPEAT);
GLUtils.texImage2D(GL10.GL_TEXTURE_2D, 0, bitmap, 0);
```

Other Game Libraries/Platforms

- Unity
- Unreal Engine
- Godot
- Cocos
- Corona SDK
- MonoGame
- AndEngine
- …


https://developer.android.com/games/engines/engines-overview

Final – Best Practices

- Garbage collector is your biggest enemy. Once it gets CPU time to do its dirty work, it will stop the world for up to 600 ms.
- Don’t use iterators, as they create new objects.
- Don’t use any of the standard Set or Map collection classes, as they create new objects on each insertion; use the SparseArray class provided by the Android API instead.
- Use StringBuffers instead of concatenating strings with the + operator.

References

- Ted Hagos, Mario Zechner, J.F. DiMarzio, Robert Green, Beginning Android Games Development From Beginner to Pro. Apress, 2020. ISBN 9781484261200
- Introduction to Game Development, edited by Steve Rabin, Charles River Media Incorporated, 2005. ISBN: 1-58450-377-7
- Sihao Hu, Tiansheng Huang, Fatih İlhan, Selim Tekin, A Survey on Large Language Model-Based Game Agents, arXiv:2404.02039v1 [cs.AI] 02 Apr 2024 - https://arxiv.org/html/2404.02039v1
- ZECHNER, Mario, Beginning Android Games, Apress
- The Game Development Process, Worcester Polytechnic Institute - http://web.cs.wpi.edu/~imgd1001/c06/
- PRUETT, Chris, Writing Real Time Games for Android http://www.scribd.com/doc/16917369/Writing-Real-Time-Games-for-Android
- http://developer.android.com/guide/index.html

Thank you for your attention

Mgr. Ing. Michal Krumnikl, Ph.D.

+420 597 325 867

michal.krumnikl@vsb.cz

www.vsb.cz