Juegos en 2D

Juegos en 2D

- Los caracteres y el campo de juego utilizan dos dimensiones (*x* y *y*).
- Ejemplos:
  - Juegos de mesa: Gato, Memoria, Reversi, Bejeweled, etc.
  - Juegos de arcade: Pac-Man, Invasores del espacio, Digger, etc.
- No requieren cámara virtual ni proyecciones.

Proceso de diseño

- Componentes del juego:
  - Objetos: pelotas, fantasmas, Mario, dragones, etc.
  - Campo de juego: tablero, espacio sideral, paisaje, etc.
- Reglas:
  - Objetivos.
  - Objetos autónomos y objetos controlados por el jugador.
  - Movimiento.
  - Puntaje.
  - Condiciones de victoria.

Game loop (ciclo del juego)

- Corre en un hilo (thread) aparte del hilo principal.

```
while (!termine) {
   revisar input del usuario
   ejecutar AI si existe
   mover los objetos autónomos
   resolver colisiones
   revisar condiciones de victoria o derrota
   actualizar las gráficas
   tocar música
}
```

Primer juego

- Una pelota que está confinada en una caja simulada por los límites de la pantalla del dispositivo.
- Objetos: pelota, paredes.
- Métodos: mueve, rebota, pinta.

Clase Pelota

```
public class Pelota {
    private float x, y;     // Centro
    private float dx, dy;   // Velocidad
    private float radio;
    private Paint paint;
    private RectF rect;     // Rectángulo contenedor

    public Pelota(float u, float v, float r, float du, float dv, int color)
    {
        x = u;
        y = v;
        radio = r;
        dx = du;
        dy = dv;
        rect = new RectF(x - radio, y - radio, x + radio, y + radio);
        paint = new Paint();
        paint.setColor(color);
    }

    public void mueve()
    {
        x += dx;
        y += dy;
        rect.set(x - radio, y - radio, x + radio, y + radio);
    }

    public void rebota(int w, int h)
    {
        if (x - radio <= 0 || x + radio >= w) {
            rebotaX();
        }
        if (y - radio <= 0 || y + radio >= h) {
            rebotaY();
        }
    }
```

Clase Pelota

```
    public void rebotaX()
    {
        dx = -dx;
    }

    public void rebotaY()
    {
        dy = -dy;
    }

    public RectF getRect()
    {
        return rect;
    }

    public void rebota(Pelota p)
    {
        if (rect.intersect(p.getRect())) {
            rebotaX(); rebotaY();
            p.rebotaX(); p.rebotaY();
        }
    }

    public void pinta(Canvas c)
    {
        c.drawCircle(x, y, radio, paint);
    }
}
```

Clase PelotaView

```
public class PelotasView extends SurfaceView
                implements SurfaceHolder.Callback {
    private PelotasThread thread;
    private Paint blackPaint; // Fondo
    private Pelota pelota;    // Pelota
    private int width, height;  // Tamaño de la pantalla

    public PelotasView(Context context)
    {
        super(context);
        inica();
    }

    public PelotasView(Context context, AttributeSet attrs)
    {
        super(context, attrs);
        inica();
    }

    public PelotasView(Context context, AttributeSet attrs, int defStyleAttr)
    {
        super(context, attrs, defStyleAttr);
        inica();
    }

    private void inica()
    {
        pelota = new Pelota(20, 40, 15, 2, 1, Color.RED);

        blackPaint = new Paint();
        blackPaint.setColor(Color.BLACK);
        blackPaint.setStyle(Paint.Style.FILL);

        getHolder().addCallback(this);
        thread = new PelotasThread(getHolder());
        setFocusable(true);
    }
```

Clase PelotaView

```
protected void pinta(Canvas canvas)
{
    if (canvas == null) {
        return;
    }

    // Limpia la pantalla
    canvas.drawRect(0, 0, width, height, blackPaint);

    // Dibuja la pelota
    pelota.pinta(canvas);
}

private void update()
{
    pelotas.mueve();
    pelotas.rebota(width, height);
}

public void surfaceChanged(SurfaceHolder holder, int format, int w, int h)
{
    width = w;
    height = h;
}

public void surfaceCreated(SurfaceHolder holder)
{
    thread.setRunning(true);
    thread.start();
}

public void surfaceDestroyed(SurfaceHolder holder)
{
}
```

Clase PelotaThread

```
    private class PelotasThread extends Thread {
        private SurfaceHolder surfaceHolder;
        private boolean isRunning = false;

        public PelotasThread(SurfaceHolder holder)
        {
            this.surfaceHolder = holder;
        }

        public void setRunning(boolean run)
        {
            this.isRunning = run;
        }

        @Override
        public void run()
        {
            while(isRunning) {

                update();

                Canvas c = null;
                try {
                    c = this.surfaceHolder.lockCanvas(null);
                    synchronized(this.surfaceHolder) {
                        PelotasView.this.pinta(c);
                    }
                }
                finally {
                    if (c != null) {
                        surfaceHolder.unlockCanvasAndPost(c);
                    }
                }
            }
        }
    }
}
```

Layout

```
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <com.tesi.pelotas.PelotasView
        android:id="@+id/pelotasView"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"/>

</FrameLayout>
```

Actividad principal

```
public class MainActivity extends Activity {
    private PelotasView view;

    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        view = (PelotasView) findViewById(R.id.pelotasView);
    }
}
```

Extensiones

- Pelota con imagen.
- Agregar un fondo de imagen.
- Agregar mas pelotas y permitir que reboten entre si.

Cargar imágenes

- Usar *BitmapFactory* para cargar la imagen:

```
Bitmap b = BitmapFactory.decodeResource(getResources(), R.drawable.nombre);
```

- Usar *drawBitmap* para pintar la imagen:

```
canvas.drawBitmap(b, x, y, null);
```

- Para el caso especial del fondo usar:

```
canvas.drawBitmap(b, new Rect(0, 0, b.getWidth(), b.getHeight()),
                     new Rect(0, 0, width, height), null);
```

Cargar imágenes

- En la clase *Pelotas*:
  - Agregar un constructor que permita recibir un bitmap.
  - Modificar el método *pinta*.

Manejar varias pelotas

- En *PelotasView* declarar un arreglo de pelotas:

```
private Pelota pelotas[] = new Pelota[4];
```

- Cargar las imágenes en *inicia*() e inicializar el arreglo.

```
Bitmap b1 = BitmapFactory.decodeResource(getResources(), R.drawable.pelota1);
Bitmap b2 = BitmapFactory.decodeResource(getResources(), R.drawable.pelota2);
Bitmap b3 = BitmapFactory.decodeResource(getResources(), R.drawable.pelota3);
Bitmap b4 = BitmapFactory.decodeResource(getResources(), R.drawable.pelota4);

pelotas[0] = new Pelota(b1, 10, 5, 1, 1);
pelotas[1] = new Pelota(b2, 20, 95, 2, -1);
pelotas[2] = new Pelota(b3, 130, 45, -1, 1);
pelotas[3] = new Pelota(b4, 0, 75, -1, 3);
```

Manejar varias pelotas

- Dibujar todas las pelotas en *pinta*().

```
for (Pelota pelota : pelotas) {
    pelota.pinta(canvas);
}
```

Manejar varias pelotas

- Actualizar todas las pelotas en *update*().

```
private void update()
{
    for (int i = 0; i < pelotas.length; i++) {
        pelotas[i].mueve();
        pelotas[i].rebota(width, height);

        for (int j = i + 1; j < pelotas.length; j++) {
            pelotas[i].rebota(pelotas[j]);
        }
    }
}
```

Tiempo

- La velocidad de este game loop depende de la velocidad de la CPU y de la densidad de la pantalla.

```
public void run()
{
    while(isRunning) {

        update();

        Canvas c = null;
        try {
            c = this.surfaceHolder.lockCanvas(null);
            synchronized(this.surfaceHolder) {
                PelotasView.this.pinta(c);
            }
        }
        finally {
            if (c != null) {
                surfaceHolder.unlockCanvasAndPost(c);
            }
        }
    }
}
```

Solución

- Usar un game loop que dependa del tiempo.
- Mover los objetos de acuerdo al tiempo transcurrido.

Game loop

```
public void run()
{
    long last = System.currentTimeMillis(), now;
    float dt, gdt = 0;

    while(isRunning) {

        // frame
        now = System.currentTimeMillis();
        dt  = (float) Math.min(1, (now - last) / 1000.0);
        gdt = gdt + dt;

        while (gdt > step) {
            gdt = gdt - step;
            update(step);
        }
        last = now;

        Canvas c = null;
        try {
            c = this.surfaceHolder.lockCanvas(null);
            synchronized(this.surfaceHolder) {
                GameView.this.pinta(c);
            }
        }
        finally {
            if (c != null) {
                surfaceHolder.unlockCanvasAndPost(c);
            }
        }
    }
}
```

Game loop

- *step* es una variable global y es el inverso del número de frames por segundo (fps) deseados.
- Si se quieren 60 frames por segundo:

```
private float step = 1.0f / 60.0f;
```

Método *update*()

- En la clase *PelotaView*:

```
private void update(float dt)
{
    for (int i = 0; i < pelotas.length; i++) {
        pelotas[i].mueve(dt);
        pelotas[i].rebota(width, height);

        for (int j = i + 1; j < pelotas.length; j++) {
            pelotas[i].rebota(pelotas[j]);
        }
    }
}
```

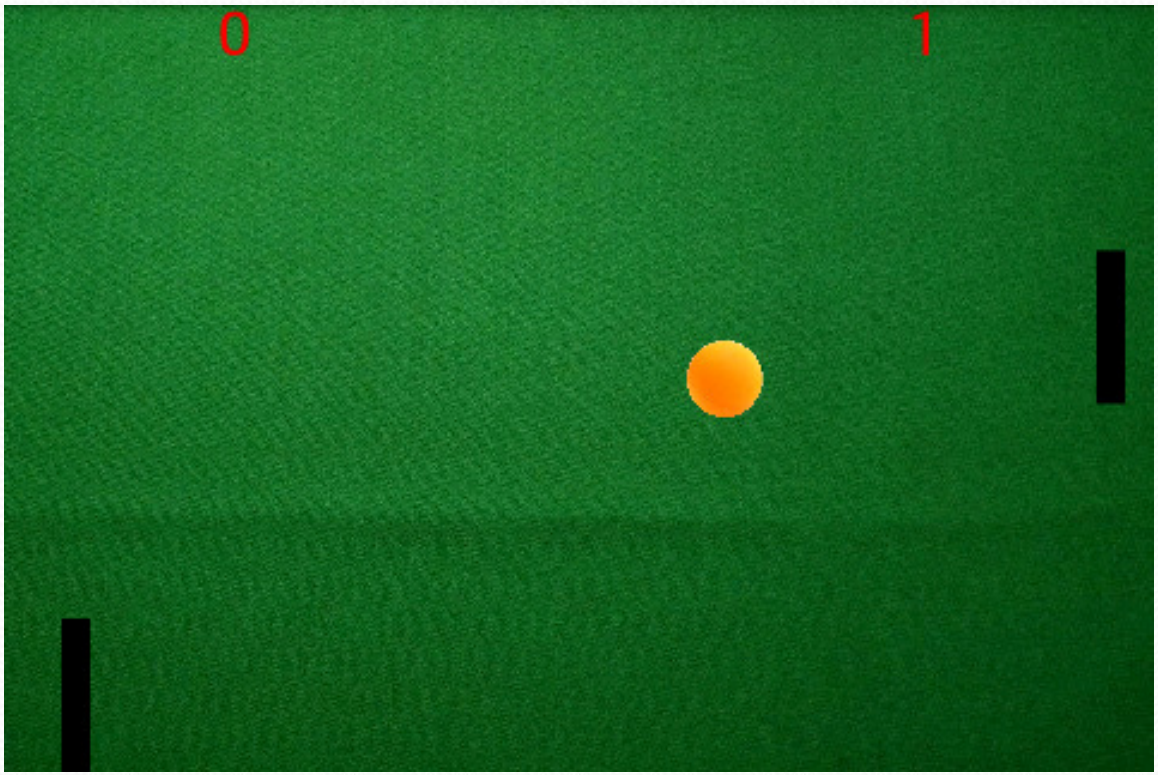
Método *mueve*()

- En la clase *Pelota*:

```
public void mueve(float dt)
{
    x += dx * dt;
    y += dy * dt;
    rect.set(x - radio / 2.0f, y - radio / 2.0f,
             x + radio / 2.0f, y + radio / 2.0f);
}
```

Segundo juego

- Ping-pong clásico.



Reglas

- Dos jugadores (raquetas):
  - Izquierda: controlada por el jugador.
  - Derecha: controlada por la computadora.
- La pelota puede rebotar arriba, abajo y en las raquetas.
- Si sale por la parte izquierda, la computadora se anota un punto.
- Si sale por la parte derecha, el jugador se anota un punto.

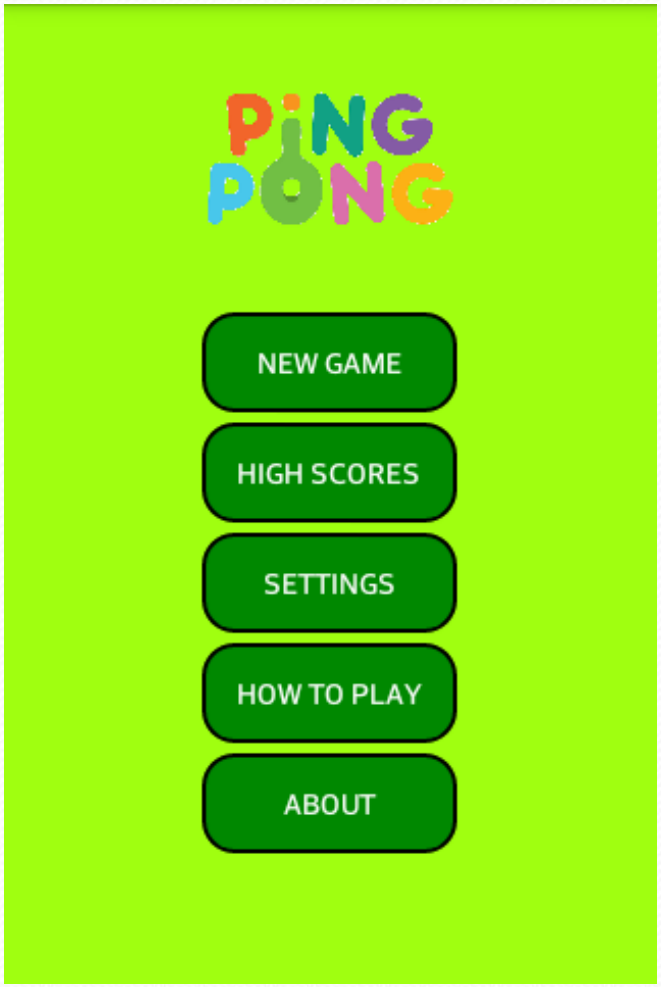
Características del juego

- Capacidad de hacer pausas.
- Color de las raquetas configurable.
- Música y efectos de sonido.
- Máximo 5 puntos.

Objetos

- Raqueta.
- Pelota.
- Fondo.
- Marcador.

Menú principal



Botones redondeados

- button_rounded_corners.xml en drawable.

```xml
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item>
        <shape android:shape="rectangle">
            <solid android:color="#008800"/>
            <stroke android:width="2dp" android:color="#000000"/>
            <!-- corners allow us to make the rounded corners button -->
            <corners android:radius="15dp"/>
        </shape>
    </item>
</selector>
```

Fuente: http://android--code.blogspot.mx/2015/01/android-rounded-corners-button.html

Layout principal

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#A0FF10"
    tools:context="com.tesi.pingpongextra.MainActivity">

    <ImageView
        android:src="@drawable/ping_pong_logo"
        android:layout_marginBottom="15dp"
        android:layout_gravity="center"
        android:layout_width="160dp"
        android:layout_height="120dp"/>

    <Button
        android:id="@+id/btnPlay"
        android:text="@string/play"
        android:onClick="buttonListener"
        android:background="@drawable/button_rounded_corners"
        android:padding="15dp"
        android:layout_marginBottom="5dp"
        android:textColor="#ffffff"
        android:layout_gravity="center"
        android:layout_width="125dp"
        android:layout_height="wrap_content"/>
```

Layout principal

```
<Button
    android:id="@+id/btnRecords"
    android:text="@string/records"
    android:onClick="buttonListener"
    android:background="@drawable/button_rounded_corners"
    android:padding="15dp"
    android:layout_marginBottom="5dp"
    android:textColor="#ffffff"
    android:layout_gravity="center"
    android:layout_width="125dp"
    android:layout_height="wrap_content"/>

<Button
    android:id="@+id/btnSettings"
    android:text="@string/settings"
    android:onClick="buttonListener"
    android:background="@drawable/button_rounded_corners"
    android:padding="15dp"
    android:layout_marginBottom="5dp"
    android:textColor="#ffffff"
    android:layout_gravity="center"
    android:layout_width="125dp"
    android:layout_height="wrap_content"/>
```

Layout principal

```
    <Button
        android:id="@+id/btnHelp"
        android:text="@string/help"
        android:onClick="buttonListener"
        android:background="@drawable/button_rounded_corners"
        android:padding="15dp"
        android:layout_marginBottom="5dp"
        android:textColor="#ffffff"
        android:layout_gravity="center"
        android:layout_width="125dp"
        android:layout_height="wrap_content"/>

    <Button
        android:id="@+id/btnAbout"
        android:text="@string/about"
        android:onClick="buttonListener"
        android:background="@drawable/button_rounded_corners"
        android:padding="15dp"
        android:layout_marginBottom="5dp"
        android:textColor="#ffffff"
        android:layout_gravity="center"
        android:layout_width="125dp"
        android:layout_height="wrap_content"/>

</LinearLayout>
```

Actividad principal

```
protected void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);

    // If the Android version is lower than Jellybean, use this call to hide
    // the status bar.
    if (Build.VERSION.SDK_INT < 16) {
        getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
                WindowManager.LayoutParams.FLAG_FULLSCREEN);
    }
    else {
        View decorView = getWindow().getDecorView();
        // Hide the status bar.
        int uiOptions = View.SYSTEM_UI_FLAG_FULLSCREEN;
        decorView.setSystemUiVisibility(uiOptions);
        // Remember that you should never show the action bar if the
        // status bar is hidden, so hide that too if necessary.
        ActionBar actionBar = getActionBar();
        if (actionBar != null) {
            actionBar.hide();
        }
        else {
            Log.d("tag", "xyz actionbar nulo");
        }
    }

    setContentView(R.layout.activity_main);
}
```

Actividad principal

```
public void buttonListener(View view)
{
    if (view.getId() == R.id.btnPlay) {
        Intent intent = new Intent(this, GameActivity.class);
        startActivity(intent);
    }
    else if (view.getId() == R.id.btnRecords) {
    }
    else if (view.getId() == R.id.btnSettings) {
        Intent intent = new Intent(this, SettingsActivity.class);
        startActivity(intent);
    }
    else if (view.getId() == R.id.btnHelp) {
    }
    else if (view.getId() == R.id.btnAbout) {
    }
}
```

Eliminar la barra de acción

- En el archivo *styles.xml* en *res/values* cambiar el estilo.

```
<resources>

    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>

</resources>
```

Layout GameActivity

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_game"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.tesi.pingpongextra.GameActivity">

    <com.tesi.pingpongextra.GameView
        android:id="@+id/gameView"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</FrameLayout>
```

GameActivity

```
public class GameActivity extends AppCompatActivity {
    private GameView view;

    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_game);
        if (Build.VERSION.SDK_INT < 16) {
            getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
                    WindowManager.LayoutParams.FLAG_FULLSCREEN);
        }
        else {
            View decorView = getWindow().getDecorView();
            int uiOptions = View.SYSTEM_UI_FLAG_FULLSCREEN;
            decorView.setSystemUiVisibility(uiOptions);
            ActionBar actionBar = getActionBar();
            if (actionBar != null) {
                actionBar.hide();
            }
            else {
                Log.d("tag", "xyz actionbar nulo");
            }
        }

        view = (GameView) findViewById(R.id.gameView);
        view.setState(GameView.STATE_READY);
    }
```

GameActivity

```
    protected void onPause()
    {
        super.onPause();
        view.pause();
    }

    protected void onResume()
    {
        super.onResume();
        view.resume();
    }
}
```

GameView: variables

```
public class GameView extends SurfaceView implements SurfaceHolder.Callback {
    // Common
    // Ciclo de vida tomado de LunarLander
    public static final int STATE_LOSE    = 1;
    public static final int STATE_PAUSE   = 2;
    public static final int STATE_READY   = 3;
    public static final int STATE_RUNNING = 4;
    public static final int STATE_WIN     = 5;

    private GameThread thread;
    private int state;
    private SurfaceHolder surfaceHolder;
    private float step = 1.0f / 30.0f;   // 30 frames x segundo
    private Paint blackPaint;
    private int width, height;

    // Specific
    private Ball ball;
    private Racket racket1, racket2;
    private Bitmap background;
    private int points1, points2;
    private Paint paintBG, paintScore, paintMessage;
    private SoundPool sound;
    private int tennisSound, racketColor;
    private String userName;
    private boolean hasMusic;
    private BackgroundMusic backgroundMusic;
```

GameView: constructores

```
public GameView(Context context)
{
    super(context);
    inica();
}

public GameView(Context context, AttributeSet attrs)
{
    super(context, attrs);
    inica();
}

public GameView(Context context, AttributeSet attrs, int defStyleAttr)
{
    super(context, attrs, defStyleAttr);
    inica();
}
```

GameView: inicia

```
private void inica()
{
    userName = "Prueba";
    hasMusic = true;
    String racketColorString = "0xff000000";
    long temp = Long.decode(racketColorString);
    racketColor = (int) temp;
    if (hasMusic) {
        backgroundMusic = new BackgroundMusic(getContext());
    }

    sound = new SoundPool(5, AudioManager.STREAM_MUSIC, 0);
    tennisSound = sound.load(getContext(), R.raw.tennisball2, 1);

    blackPaint = new Paint();
    blackPaint.setColor(Color.WHITE);
    blackPaint.setStyle(Paint.Style.FILL);

    paintScore = new Paint();
    paintScore.setColor(Color.RED);
    paintScore.setTextAlign(Paint.Align.CENTER);
    paintScore.setTextSize(26);
    paintScore.setTypeface(Typeface.create("Arial", Typeface.NORMAL));
    paintScore.setAntiAlias(true);
```

GameView: inicia

```
    paintMessage = new Paint();
    paintMessage.setColor(Color.WHITE);
    paintMessage.setTextAlign(Paint.Align.CENTER);
    paintMessage.setTextSize(32);
    paintMessage.setTypeface(Typeface.create("Arial", Typeface.NORMAL));
    paintMessage.setAntiAlias(true);

    paintBG = new Paint();
    paintBG.setAntiAlias(true);

    background = BitmapFactory.decodeResource(getResources(), R.drawable.table);

    racket1 = new Racket(0, 0, 0, 0, racketColor);
    racket2 = new Racket(0, 0, 0, 0, 0, racketColor);

    Bitmap ballImage = BitmapFactory.decodeResource(getResources(), R.drawable.ball);
    ball = new Ball(0, 0, 0, 0, 0, ballImage);

    getHolder().addCallback(this);
    setFocusable(true);
}
```

GameView: onTouch

```
public boolean onTouchEvent(MotionEvent event)
{
    if (event.getX() < width / 3) {
        racket1.setY(event.getY(), height);
    }

    if (event.getAction() == MotionEvent.ACTION_UP) {
        synchronized (surfaceHolder) {
            switch (state) {

                case STATE_RUNNING:
                    if (event.getX() > 2 * width / 3) {
                        setState(STATE_PAUSE);
                    }
                    break;

                case STATE_PAUSE:
                    setState(STATE_RUNNING);
                    break;

                case STATE_LOSE:
                case STATE_WIN:
                case STATE_READY:
                    startGame();
                    break;
            }
        }
    }

    return true;
}
```

GameView: eventos

```
public void startGame()
{
    synchronized (surfaceHolder) {
        points1 = points2 = 0;
        racket1.setPosition(width / 20, height / 2);
        racket2.setPosition(19 * width / 20, height / 2);
        ball.setPosition(width / 2, height / 2);
        setState(STATE_RUNNING);
    }
}

public void setState(int s)
{
    state = s;
}

public void pause()
{
    if (state == STATE_RUNNING) {
        setState(STATE_PAUSE);
        if (hasMusic) {
            backgroundMusic.cancel(true);
            backgroundMusic.turnOff();
        }
    }
}

public void resume()
{
    if (hasMusic) {
        backgroundMusic.execute();
    }
}
```

GameView: update

```
private void update(float dt)
{
    synchronized (surfaceHolder) {
        ball.move(dt);
        if (ball.getX() < 0) {
            points2++;
            ball.setPosition(width / 2, height / 2);
        }
        else if (ball.getX() > width) {
            points1++;
            ball.setPosition(width / 2, height / 2);
        }

        ball.rebound(height);
        if (ball.rebound(racket1)) {
            sound.play(tennisSound, 1, 1, 1, 0, 1);
        }
        if (ball.rebound(racket2)) {
            sound.play(tennisSound, 1, 1, 1, 0, 1);
        }

        if (ball.getSpeedY() < 0) {
            racket2.moveUp(dt);
        }
        else {
            racket2.moveDown(height, dt);
        }

        if (points1 == 5) {
            state = STATE_WIN;
        }

        if (points2 == 5) {
            state = STATE_LOSE;
        }
    }
}
```

GameView: pinta

```
protected void pinta(Canvas canvas)
{
    if (canvas == null) {
        return;
    }

    canvas.drawRect(0, 0, width, height, blackPaint);

    canvas.drawBitmap(background,
            new Rect(0, 0, background.getWidth(), background.getHeight()),
            new Rect(0, 0, width, height), paintBG);
    racket1.draw(canvas);
    racket2.draw(canvas);
    ball.draw(canvas);
    canvas.drawText(points1 + "", width / 5, height / 15, paintScore);
    canvas.drawText(points2 + "", 4 * width / 5, height / 15, paintScore);
```

GameView: pinta

```
    StringBuilder builder = new StringBuilder();
    switch (state) {

        case STATE_LOSE:
            builder.append("You lose");
            if (userName.length() > 0) {
                builder.append(", ").append(userName);
            }
            builder.append("!");
            canvas.drawText(builder.toString(), width / 2, height / 2, paintMessage);
            break;

        case STATE_WIN:
            builder.append("You win");
            if (userName.length() > 0) {
                builder.append(", ").append(userName);
            }
            builder.append("!");
            canvas.drawText(builder.toString(), width / 2, height / 2, paintMessage);
            break;

        case STATE_PAUSE:
            builder.append("Pause, touch the screen to resume");
            if (userName.length() > 0) {
                builder.append(", ").append(userName);
            }
            canvas.drawText(builder.toString(), width / 2, height / 2, paintMessage);
            break;

        case STATE_READY:
            builder.append("Touch the screen to play");
            if (userName.length() > 0) {
                builder.append(", ").append(userName);
            }
            canvas.drawText(builder.toString(), width / 2, height / 2, paintMessage);
            break;
    }
}
```

GameView: callbacks

```
public void surfaceChanged(SurfaceHolder holder, int format, int w, int h)
{
    width = w;
    height = h;

    racket1.setPosition(width / 20, height / 2);
    racket1.setSize(width / 40, height / 5);

    racket2.setPosition(19 * width / 20, height / 2);
    racket2.setSize(width / 40, height / 5);
    racket2.setSpeed(15 * height / 100);

    ball.setPosition(width / 2, height / 2);
    ball.setRadius(width / 30);
    ball.setSpeed(35 * width / 100, 35 * height / 100);
}

public void surfaceCreated(SurfaceHolder holder)
{
    thread = new GameThread(getHolder());
    thread.setRunning(true);
    thread.start();
}

public void surfaceDestroyed(SurfaceHolder holder)
{
    if (thread == null) {
        return;
    }

    boolean retry = true;
    thread.setRunning(false);
    while (retry) {
        try {
            thread.join();
            retry = false;
        }
        catch (InterruptedException e) {
            Log.d("tag", "xyz error en SurfaceDestroyed");
        }
    }
}
```

GameThread

```
private class GameThread extends Thread {
    private boolean isRunning = false;

    public GameThread(SurfaceHolder holder)
    {
        surfaceHolder = holder;
    }

    public void setRunning(boolean run)
    {
        isRunning = run;
    }

    public boolean isRunning()
    {
        return isRunning;
    }
```

GameThread: run

```
    public void run()
    {
        long last = System.currentTimeMillis(), now;
        double dt, gdt = 0;

        while (isRunning) {

            // frame
            now = System.currentTimeMillis();
            if (state == STATE_RUNNING) {
                dt = Math.min(1, (now - last) / 1000.0);
                gdt = gdt + dt;

                while (gdt > step) {
                    gdt = gdt - step;
                    update(step);
                }
            }
            last = now;

            Canvas c = null;
            try {
                c = surfaceHolder.lockCanvas(null);
                synchronized (surfaceHolder) {
                    GameView.this.pinta(c);
                }
            }
            finally {
                if (c != null) {
                    surfaceHolder.unlockCanvasAndPost(c);
                }
            }
        }
    }
}
```

BackgroundMusic

```
public class BackgroundMusic extends AsyncTask<Void, Void, Void> {
    private Context context;
    private MediaPlayer player;

    public BackgroundMusic(Context c)
    {
        context = c;
    }

    protected Void doInBackground(Void... params)
    {
        player = MediaPlayer.create(context, R.raw.reconstruct);
        player.setLooping(true);    // Set looping
        player.setVolume(100, 100);
        player.start();

        return null;
    }

    public void turnOff()
    {
        player.stop();
        player = null;
    }

    public void onCancelled(Void result)
    {
        player.stop();
        player = null;
    }
}
}
```

Implementando settings

- Crear una actividad que extienda a *PreferenceActivity*.
- Definir un archivo xml en la carpeta *res/xml* con los settings.
- En el programa, recuperar los settings.

preferences.xml

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android"
                  android:layout_width="match_parent"
                  android:layout_height="match_parent">

    <EditTextPreference
        android:defaultValue=""
        android:key="username"
        android:summary="Please provide your username"
        android:title="Your Name" />

    <CheckBoxPreference
        android:defaultValue="false"
        android:key="music"
        android:summary="This option if selected will allow the
                         application to play music."
        android:title="Music"/>

    <ListPreference
        android:defaultValue="0xff000000"
        android:entries="@array/listArray"
        android:entryValues="@array/listValues"
        android:key="racketColor"
        android:summary="Select the color of the rackets"
        android:title="Color of the rackets"/>

</PreferenceScreen>
```

Valores de ListPreference

- Archivo *array.xml* en *res/values*.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string-array name="listArray">
        <item>Black</item>
        <item>Red</item>
        <item>Brown</item>
        <item>Blue</item>
    </string-array>

    <string-array name="listValues">
        <item>0xff000000</item>
        <item>0xffff0000</item>
        <item>0xffd2691e</item>
        <item>0xff0000ff</item>
    </string-array>

</resources>
```

SettingsActivity

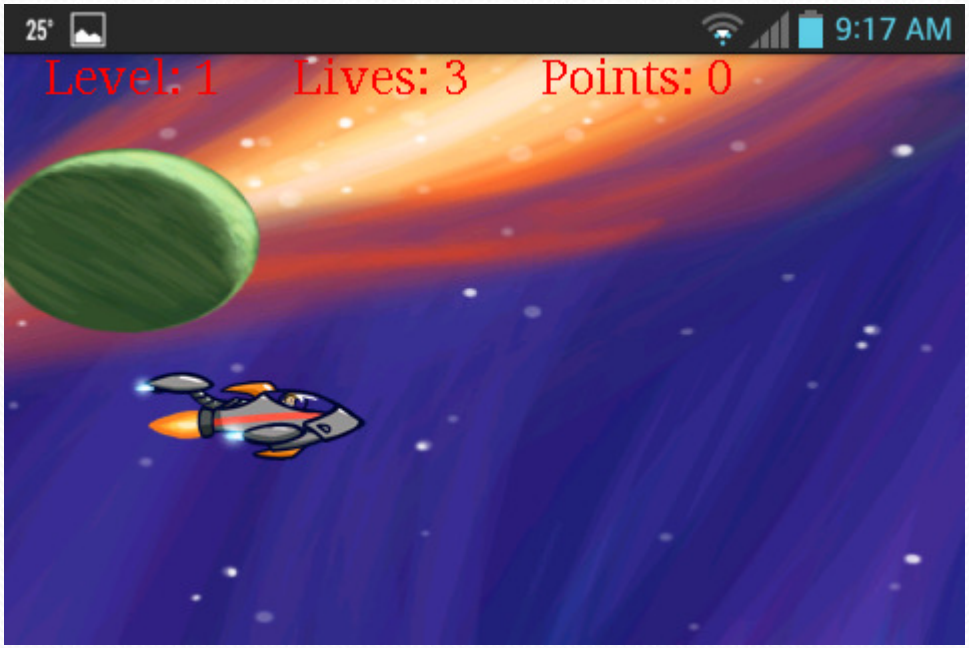
```
public class SettingsActivity extends PreferenceActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        getFragmentManager().beginTransaction().replace(
        android.R.id.content, new SettingsFragment()).commit();
    }

    public static class SettingsFragment extends PreferenceFragment {

        @Override
        public void onCreate(final Bundle savedInstanceState)
        {
            super.onCreate(savedInstanceState);
            addPreferencesFromResource(R.xml.preferences);
        }
    }
}
```

En inicia() de GameView

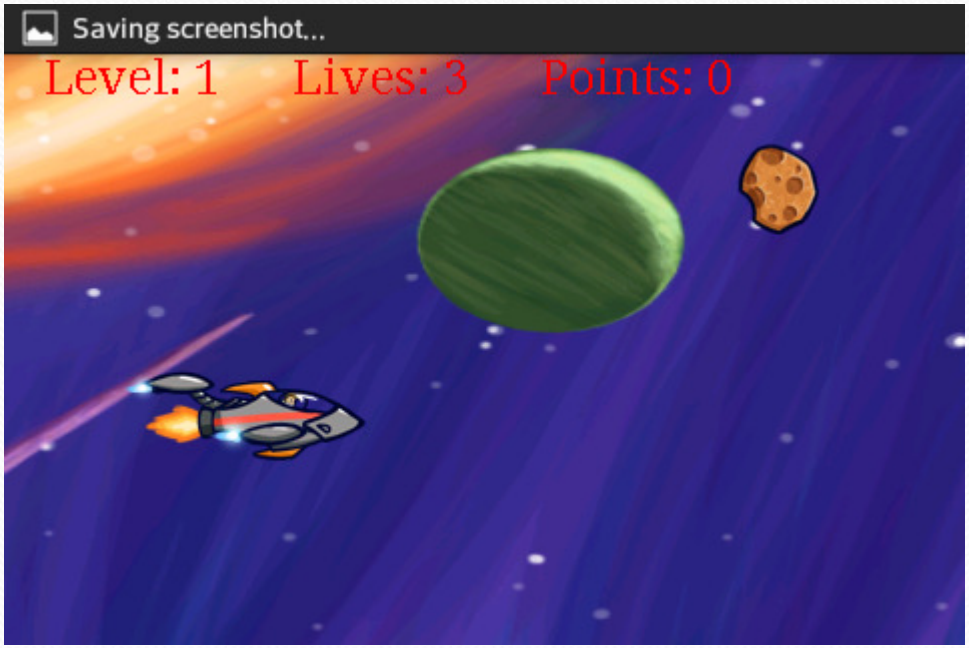
```
SharedPreferences preferences = PreferenceManager.getDefaultSharedPreferences(getContext());
userName = preferences.getString("username", "");
hasMusic = preferences.getBoolean("music", false);
String racketColorString = preferences.getString("racketColor", "0xff000000");
long temp = Long.decode(racketColorString);
racketColor = (int) temp;
if (hasMusic) {
    backgroundMusic = new BackgroundMusic(getContext());
}
```

Tercer juego: JetBoy

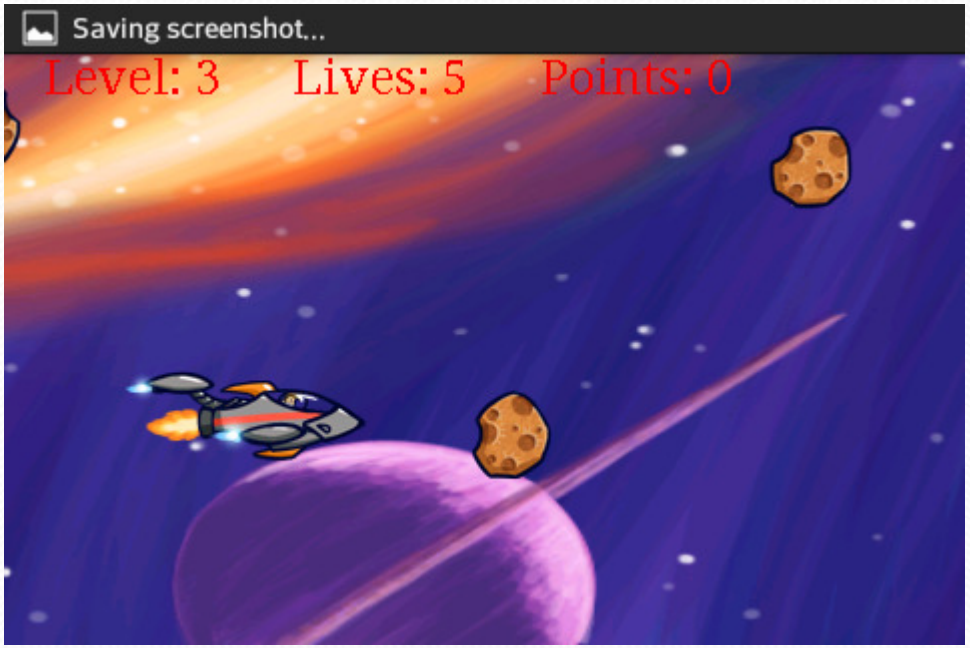
- Nave que viaja por el espacio disparando a los asteroides que le salen al paso.



JetBoy



JetBoy



Reglas

- La nave comienza con 5 vidas y en nivel 1.
- Cada vez que un asteroide toca la nave se pierde una vida.
- Cada asteroide destruido otorga 10 puntos.
- Cada 100 puntos se otorga una nueva vida y se pasa al siguiente nivel.
- Conforme se incrementa el nivel, salen más asteroides y éstos viajan más rápido.

Características

- Se cuentan con varias imágenes de la nave y de los asteroides para dar efectos de movimiento.
- Hay dos fondos para dar un efecto de “parallax scrolling.”
- Capacidad de hacer pausas, música, efectos de sonido, guardar records y configuraciones.

Movimiento con image frames

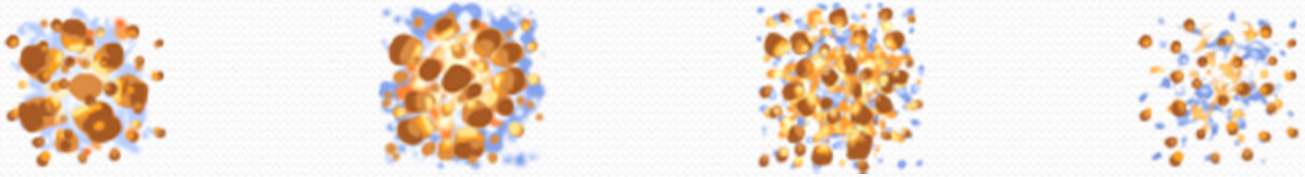
- Nave volando:

- Nave siendo golpeada:

Movimiento con image frames

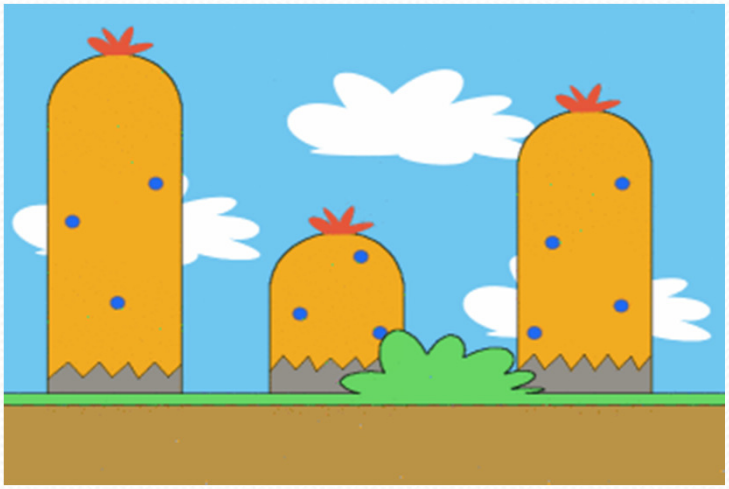
- Asteroide rotando:

- Asteroide explotando:



Parallax scrolling

- Hay más de una imagen de fondo.
- Cada imagen se mueve a velocidad distinta.



Crédito: By OhSqueezy - Own work, CC BY-SA 3.0,
https://commons.wikimedia.org/w/index.php?curid=15775352

Fondos

Clases

- Clase Ship para representar la nave.
- Clase Asteroid para representar un asteroide.
- Clase Laser para representar un disparo de laser.
- Nota 1: en un instante dado puede haber más de un asteroide y más de un laser en la pantalla y por lo tanto hay que guardarlos en arreglos.
- Nota 2: se necesita definir una estrategia para que los arreglos no crezcan demasiado y agoten la memoria.

Clase Ship

- Variables y constructor:

```
private Bitmap normalImages[], hitImages[];
private int indexNormal, indexHit;
private boolean hit;
private float x, y;
private RectF rect;
private Paint paint;

public Ship(Bitmap n[], Bitmap h[])
{
    normalImages = n;
    hitImages = h;
    hit = false;
    x = y = 0;
    indexNormal = indexHit = 0;
    rect = new RectF(x, y,
        x + normalImages[0].getWidth(),
        y + normalImages[0].getHeight());
    paint = new Paint();
    paint.setAntiAlias(true);
}
```

Clase Ship

- Método paint():

```
public void pinta(Canvas canvas)
{
    if (hit) {
        canvas.drawBitmap(hitImages[indexHit / 3], x, y, paint);
        indexHit++;
        if (indexHit == 3 * hitImages.length) {
            indexHit = 0;
            setHit(false);
        }
    }
    else {
        canvas.drawBitmap(normalImages[indexNormal / 2], x, y, paint);
        indexNormal++;
        if (indexNormal == 2 * normalImages.length) {
            indexNormal = 0;
        }
    }
}
```

Clase Asteroid

- Variables y constructor:

```
private Bitmap normalImages[], explosionImages[];
private int normalIndex, explosionIndex;
private boolean exploded, deleted;
private float x, y, dx;
private RectF rect;
private Paint paint;

public Asteroid(Bitmap img[], Bitmap expl[], float u, float v, float du)
{
    normalImages = img;
    explosionImages = expl;
    exploded = false;
    deleted = false;
    x = u;
    y = v;
    dx = du;
    rect = new RectF(x, y,
        x + img[0].getWidth(),
        y + img[0].getHeight());
    paint = new Paint();
    paint.setAntiAlias(true);
}
```

Clase Asteroid

- Método paint():

```
public void pinta(Canvas canvas)
{
    if (exploded) {
        canvas.drawBitmap(explosionImages[explosionIndex / 3], x, y, paint);
        explosionIndex++;
        if (explosionIndex == 3 * explosionImages.length) {
            explosionIndex = 0;
            deleted = true;
        }
    }
    else {
        canvas.drawBitmap(normalImages[normalIndex / 2], x, y, paint);
        normalIndex++;
        if (normalIndex == 2 * normalImages.length) {
            normalIndex = 0;
        }
    }
}
```

Clase GameView

- Arreglos para guardar asteroides y láseres:

```
private ArrayList<Asteroid> asteroids = new ArrayList<>();
private ArrayList<Laser> lasers = new ArrayList<>();
```

- Variables para manejar el fondo con parallax scrolling:

```
private int BGFarX = 0, BGNearX = 0;
private Bitmap BGImageFar, BGImageNear;
```

Método update

```
public void update(float dt)
{
    // decrement the far background
    BGFarX -= 1;

    // decrement the near background
    BGNearX -= 4;

    // move asteroids
    for (int i = 0; i < asteroids.size(); i++) {
        asteroids.get(i).move(dt);
    }

    // move lasers
    for (int i = 0; i < lasers.size(); i++) {
        lasers.get(i).move(dt);
    }
```

Método update

```
// checa colisiones
// asteroides con la nave
for (int i = 0; i < asteroids.size(); i++) {
    Asteroid a = asteroids.get(i);
    if (!a.isDeleted() && !a.isExploded()) {
        if (ship.rect.intersect(asteroids.get(i).rect)) {
            asteroids.get(i).setExploded(true);
            ship.setHit(true);
            lives--;
            break;
        }
    }
}
// asteroides con lasers
boolean borrado;
int j;
for (int i = 0; i < asteroids.size(); i++) {
    j = 0;
    borrado = false;
    while (!borrado && (j < lasers.size())) {
        if (asteroids.get(i).rect.intersect(lasers.get(j).rect)) {
            points += 10;
            borrado = true;
            asteroids.get(i).setExploded(true);
            lasers.remove(j);
        }
        else {
            j++;
        }
    }
}
```

Método update

```
int i;
// delete deleted and out of bounds asteroids
i = 0;
while (i < asteroids.size()) {
    if (asteroids.get(i).outOfBounds() || asteroids.get(i).isDeleted()) {
        asteroids.remove(i);
    }
    else {
        i++;
    }
}

// delete out of bounds lasers
i = 0;
while (i < lasers.size()) {
    if (lasers.get(i).outOfBounds(canvasWidth)) {
        lasers.remove(i);
    }
    else {
        i++;
    }
}
```

Método update

```
    // create asteroids
    long currentTime = System.currentTimeMillis();
    if ((currentTime - asteroidTime) > timeBetweenAsteroids) {
        asteroidTime = currentTime;
        int x = canvasWidth;
        int y = random.nextInt(canvasHeight);
        int dx = random.nextInt(11) + (canvasWidth / speed) - 5;
        asteroids.add(new Asteroid(asteroidImages, explosionImages, x, y, -dx));
    }

    if ((points % 100) == 0) {
        lives++;
    }

}
```

Método paint

```
protected void pinta(Canvas canvas)
{
    if (canvas == null) {
        return;
    }

    pintaBackground(canvas);

    ship.pinta(canvas);

    for (int i = 0; i < asteroids.size(); i++) {
        asteroids.get(i).pinta(canvas);
    }

    for (int i = 0; i < lasers.size(); i++) {
        lasers.get(i).pinta(canvas);
    }

    pintaLetreros();
    pintaMarcador();
}
```

Método pintaBackground

```
private void pintaBackground(Canvas canvas)
{
    // calculate the wrap factor for matching image draw
    int newBGFarX = BGImageFar.getWidth() - (-BGFarX);

    // Fondo lejano
    // if we have scrolled all the way, reset to start
    if (newBGFarX <= 0) {
        BGFarX = 0;
        // only need one draw
        canvas.drawBitmap(BGImageFar, BGFarX, 0, paintFondo);

    }
    else {
        // need to draw original and wrap
        canvas.drawBitmap(BGImageFar, BGFarX, 0, paintFondo);
        canvas.drawBitmap(BGImageFar, newBGFarX, 0, paintFondo);
    }
```

Método pintaBackground

```
    // Fondo cercano
    // same story different image...
    int newBGNearX = BGImageNear.getWidth() - (-BGNearX);

    if (newBGNearX <= 0) {
        BGNearX = 0;
        canvas.drawBitmap(BGImageNear, BGNearX, 0, paintFondo);
    }
    else {
        canvas.drawBitmap(BGImageNear, BGNearX, 0, paintFondo);
        canvas.drawBitmap(BGImageNear, newBGNearX, 0, paintFondo);
    }
}
```

Alternativas para hacer juegos

- OpenGL ES. Integrado con Android.
- libGDX. https://libgdx.badlogicgames.com/
- Unity 3D. https://unity3d.com/
- Unreal Engine. https://www.unrealengine.com/
- Y más: http://appindex.com/blog/the-big-list-of-android-ios-game-development-tools-engines-libraries-and-resources/