

# Aula 06 – Gráficos 2D

Programação em Java para a Plataforma Android



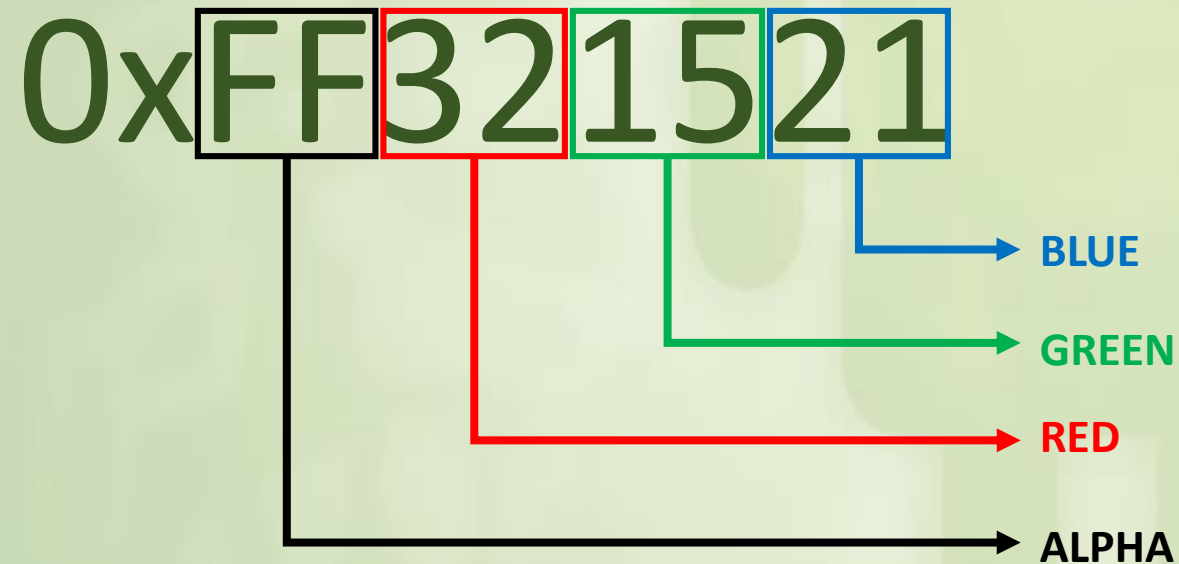
# Agenda

- Representando cores
- Desenhando formas simples
- Lidando com eventos de baixo nível
- Como criar animações simples?
- Como adicionar um key-pad à aplicação?
- Representando iterações entre atividades



# Cores

- O Android representa cores como inteiros de 32 bits, em formato hexadecimal



Red, Green e Blue são óbvios. Mas o que é Alpha?

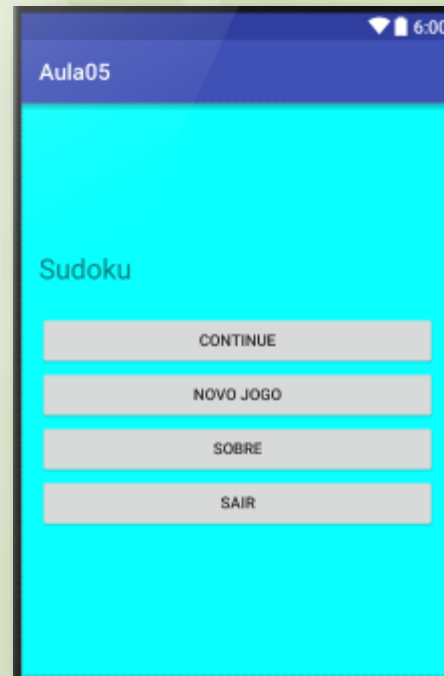


# Exemplos de cores

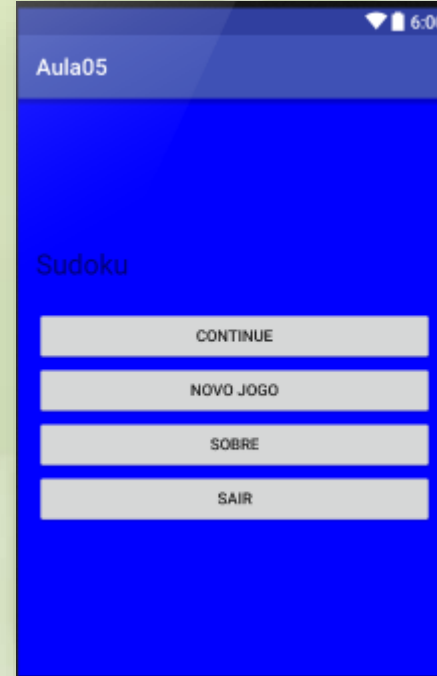
0x3500FFFF



0xFF00FFFF



0xFF0000FF



0X000000FF



# View

- A classe básica que descreve elementos gráficos é *View*
- Essa classe possui o método *onDraw*, que determina o que será desenhado na tela do dispositivo
- Esse método recebe um objeto do tipo *Canvas*, que representa a tela do dispositivo



# Gráficos de baixo nível

```
public class GraphicsView extends View {  
    public GraphicsView(Context context, AttributeSet attrs) {  
        super(context, attrs);  
    }  
  
    @Override  
    protected void onDraw(Canvas canvas) {  
        Path circle = new Path();  
        circle.addCircle(canvas.getWidth()/2, canvas.getHeight()/2,  
                          canvas.getWidth()/3, Path.Direction.CW);  
        Paint aPaint = new Paint();  
        aPaint.setColor(Color.LTGRAY);  
        canvas.drawPath(circle, aPaint);  
        Paint bPaint = new Paint();  
        bPaint.setColor(Color.BLUE);  
        bPaint.setTextSize(100);  
        canvas.drawTextOnPath("Mais vale um passáro na mão do que dois voando!",  
                               circle, 0, 20, bPaint);  
    }  
}
```

O que a classe  
GraphicsView  
desenha?

Como usar  
GraphicsView?



# Gráficos de baixo nível



Como usar  
GraphicsView?



# Views são usadas como Layouts

```
public class GraphicsViewActivity extends AppCompatActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(new GraphicsView(this));  
    }  
}
```

O que quer dizer  
a palavra “**this**”  
para a nossa  
View?





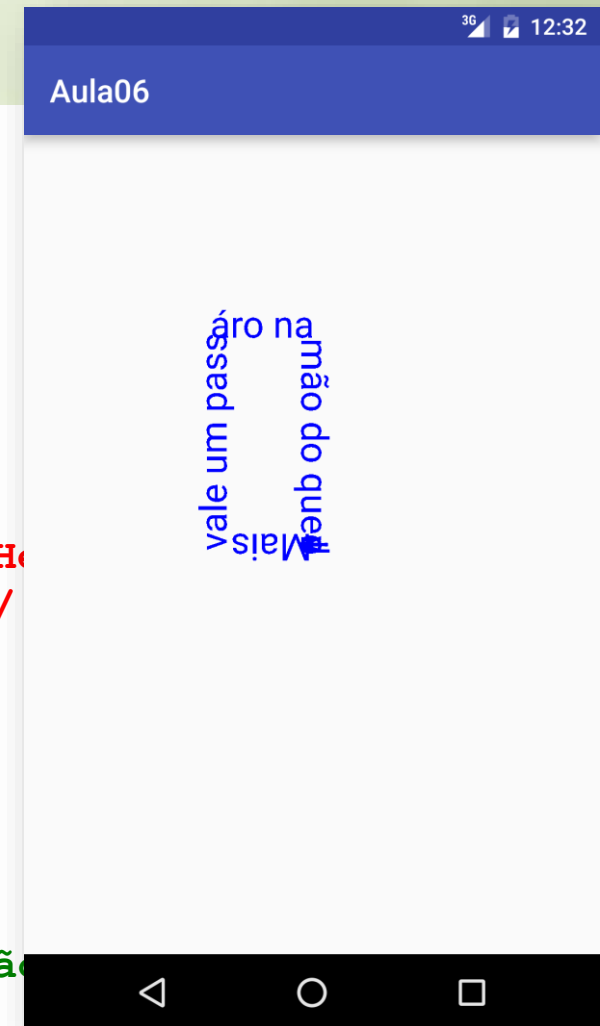
# Mais um exemplo

```
public class GraphicsView extends View {  
  
    public GraphicsView(Context context) {  
        super(context);  
    }  
    @Override  
    protected void onDraw(Canvas canvas) {  
        Path circle = new Path();  
        circle.addRect(canvas.getWidth() / 2, canvas.getHeight() / 2,  
            canvas.getWidth() / 3, canvas.getWidth() / 3, Path.Direction.CW);  
        Paint aPaint = new Paint();  
        aPaint.setColor(Color.LTGRAY);  
        canvas.drawPath(circle, aPaint);  
        Paint bPaint = new Paint();  
        bPaint.setColor(Color.BLUE);  
        bPaint.setTextSize(70);  
        canvas.drawTextOnPath("Mais vale um passáro na mão do que dois voando!",  
            circle, 0, 20, bPaint);  
    }  
}
```



# Mais um exemplo

```
public class GraphicsView extends View {  
  
    public GraphicsView(Context context) {  
        super(context);  
    }  
    @Override  
    protected void onDraw(Canvas canvas) {  
        Path circle = new Path();  
        circle.addRect(canvas.getWidth() / 2, canvas.getHeight() / 2,  
            canvas.getWidth() / 3, canvas.getHeight() / 3,  
            Path.Direction.CW);  
        Paint aPaint = new Paint();  
        aPaint.setColor(Color.LTGRAY);  
        canvas.drawPath(circle, aPaint);  
        Paint bPaint = new Paint();  
        bPaint.setColor(Color.BLUE);  
        bPaint.setTextSize(70);  
        canvas.drawTextOnPath("Mais vale um passáro na mão do que  
            circle, 0, 20, bPaint);  
    }  
}
```



# Dica legal: tamanho da janela

```
private int height;
private int width;

public GraphicsView(Context context) {
    super(context);
    getWindowSize(context);
}

private void getWindowSize(Context context) {
    DisplayMetrics metrics = new DisplayMetrics();
    WindowManager wm = (WindowManager)
        context.getSystemService(Context.WINDOW_SERVICE);
    Display display = wm.getDefaultDisplay();
    display.getMetrics(metrics);
    height = metrics.heightPixels;
    width = metrics.widthPixels;
}
```



# Drawables

- Drawables são entidades que podem ser visualizadas na tela do dispositivo
- Em geral esses recursos ficam armazenados na pasta drawable
- Exemplos incluem figuras (jpeg, png, svg), padrões de cores, desenhos vetoriais, camadas, etc



# Exemplo de Drawable

- Esse arquivo será usado como pano de fundo em nosso exemplo corrente:

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android">
  <gradient
    android:startColor="#FFFFFF"
    android:endColor="#808080"
    android:angle="270" />
</shape>
```

O que esse  
drawable faz?



# Exemplo de Drawable

- Esse arquivo será usado como pano de fundo em nosso exemplo corrente:

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android">
<gradient
    android:startColor="#FFFFFF"
    android:endColor="#808080"
    android:angle="270" />
</shape>
```

E como usá-lo?



# Exemplo de Drawable

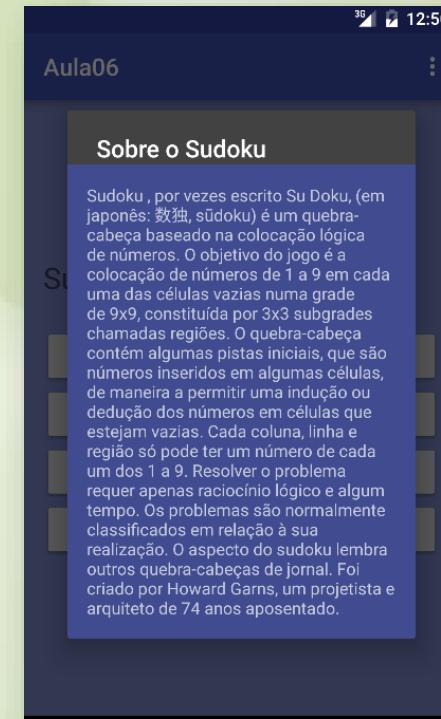
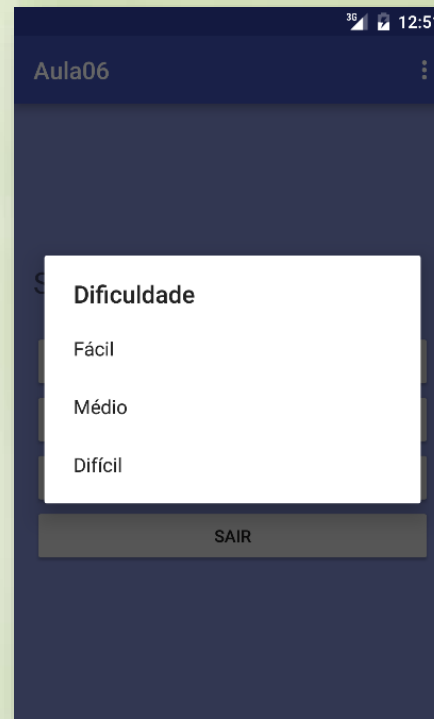
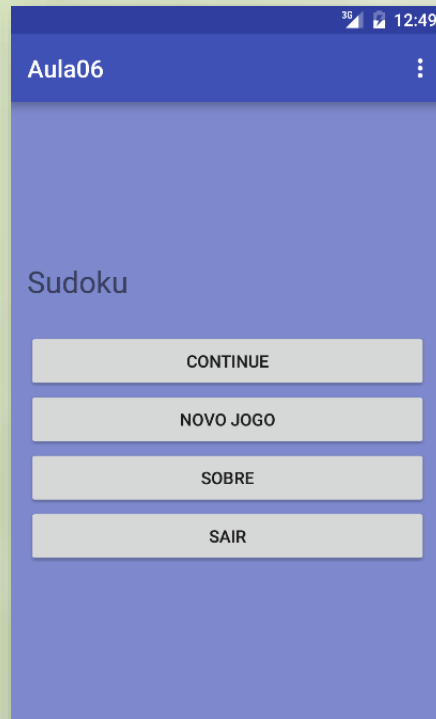
- Esse arquivo será usado como pano de fundo em nosso exemplo corrente:

```
public GraphicsView(Context context) {  
    super(context);  
    getWindowSize(context);  
    setBackgroundResource(R.drawable.gradient);  
}
```



# Continuando nosso Sudoku

- Até agora temos um aplicativo que contém um belo layout inicial, e uma caixa “About”!





# Começando um novo jogo

Sudoku.java

```
private void iniciarNovoJogo() {  
    new AlertDialog.Builder(this)  
        .setTitle(R.string.new_game_title)  
        .setItems(R.array.difficulty, new DialogInterface.OnClickListener() {  
            @Override  
            public void onClick(DialogInterface dialog, int difficulty) {  
                novoJogo(difficulty);  
            }  
        }) .show();  
}
```

Em que consiste  
o método  
**novoJogo**?



# Criando um novo jogo

```
private void novoJogo(int difficulty) {  
    Log.d("novoJogo", "selecionou o " + difficulty);  
    Intent i = new Intent(Sudoku.this, Game.class);  
    i.putExtra(Game.KEY_DIFFICULTY, difficulty);  
    startActivity(i);  
}
```

Sudoku.java



# Setup Inicial

- A classe Game, sendo uma atividade, precisa ser registrada no manifesto

```
<activity android:name=".Game"  
    android:label="@string/game_title">
```

AndroidManifest.xml

- Criemos também algumas strings para usarmos em nossa implementação de Game

```
<string name="game_title">Game</string>  
<string name="no_moves_label">Nomoves</string>  
<string name="keypad_title">Keypad</string>
```

strings.xml



# A classe Game

Game.java

```
public class Game extends Activity {  
    public static final String KEY_DIFFICULTY = "KEY_DIFFICULTY";  
    public static final int DIFFICULT_EASY = 0;  
    public static final int DIFFICULT_MEDIUM = 1;  
    public static final int DIFFICULT_HARD = 2;  
    private static final String TAG = "Sudoku";  
    private int puzzle[] = new int[9 * 9];  
    private PuzzleView puzzleView;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        Log.d(TAG, "onCreate");  
        int difficulty = getIntent().getIntExtra(KEY_DIFFICULTY, DIFFICULT_EASY);  
    }  
}
```

Que dois  
argumentos são  
esses?

Precisamos agora  
definir o jogo  
propriamente dito

O que um "Game"  
sabe fazer?


Quais os passos  
para criarmos o  
jogo?



# A classe Game

Game.java

```
public class Game extends Activity {  
    public static final String KEY_DIFFICULTY = "KEY_DIFFICULTY";  
    public static final int DIFFICULT_EASY = 0;  
    public static final int DIFFICULT_MEDIUM = 1;  
    public static final int DIFFICULT_HARD = 2;  
    private static final String TAG = "Sudoku";  
    private int puzzle[] = new int[9 * 9];  
    private PuzzleView puzzleView;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        Log.d(TAG, "onCreate");  
        int difficulty = getIntent().getIntExtra(KEY_DIFFICULTY, DIFFICULT_EASY);  
    }  
}
```



Mas, antes de prosseguirmos, por que estamos representando uma matriz desse jeito? Dica: tem a ver com eficiência.



# A classe Game

```
public class Game extends Activity {  
    public static final String KEY_DIFFICULTY = "KEY_DIFFICULTY";  
    public static final int DIFFICULT_EASY = 0;  
    public static final int DIFFICULT_MEDIUM = 1;  
    public static final int DIFFICULT_HARD = 2;  
    private static final String TAG = "Sudoku";  
    private int puzzle[] = new int[9 * 9];  
    private PuzzleView puzzleView;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        Log.d(TAG, "onCreate");  
        int difficulty = getIntent().getIntExtra(KEY_DIFFICULTY, DIFFICULT_EASY);  
        puzzle = getPuzzle(difficulty);  
        calculateUsedTiles();  
        puzzleView = new PuzzleView(this);  
        setContentView(puzzleView);  
        puzzleView.requestFocus();  
    }  
}
```

A classe **PuzzleView**  
ainda não existe

Vários **métodos**  
ainda não foram  
implementados

Game.java



# PuzzleView

- A classe PuzzleView é responsável por desenhar o tabuleiro do jogo
  - O que ela precisa saber fazer?



# PuzzleView

```
public class PuzzleView extends View {  
    private static final String TAG = "Sudoku";  
    private final Game game;  
  
    public PuzzleView(Context context) {  
        super(context);  
        this.game = (Game) context;  
        setFocusable(true);  
        setFocusableInTouchMode(true);  
    }  
  
    @Override  
    protected void onSizeChanged(int w, int h, int oldw, int oldh) { }  
}
```

Agora precisamos descobrir o tamanho de um quadrado de sudoku. Como fazemos isto?

PuzzleView.java





# Computando pequenos quadrados

PuzzleView.java

```
private float width; // Largura de um quadrado
private float height; // Altura de um quadrado
private int selX; // Índice X selecionado
private int selY; // Índice Y selecionado
private final Rect selRect = new Rect();

@Override
protected void onSizeChanged(int w, int h, int oldw, int oldh) {
    width = w / 9f;
    height = h / 9f;
    getRect(selX, selY, selRect);
    Log.d(TAG, "onSizeChanged: width " + width + ", height " + height);
    super.onSizeChanged(w, h, oldw, oldh);
}

private void getRect(int x, int y, Rect rect){
    rect.set((int) (x * width), (int) (y * height),
        (int) (x * width * width), (int) (y * height * height));
}
```



# Marcando a seleção

PuzzleView.java

```
private float width; // Largura de um quadrado
private float height; // Altura de um quadrado
private int selX; // Índice X selecionado
private int selY; // Índice Y selecionado
private final Rect selRect = new Rect();
```

```
@Override
protected void onSizeChanged(int w, int h, int oldw, int oldh) {
    width = w / 9f;
    height = h / 9f;
    getRect(selX, selY, selRect);
    Log.d(TAG, "onSizeChanged: width " + width + ", height " + height);
    super.onSizeChanged(w, h, oldw, oldh);
}
```

```
private void getRect(int x, int y, Rect rect){
    rect.set((int) (x * width), (int) (y * height),
            (int) (x * width * width), (int) (y * height * height));
}
```

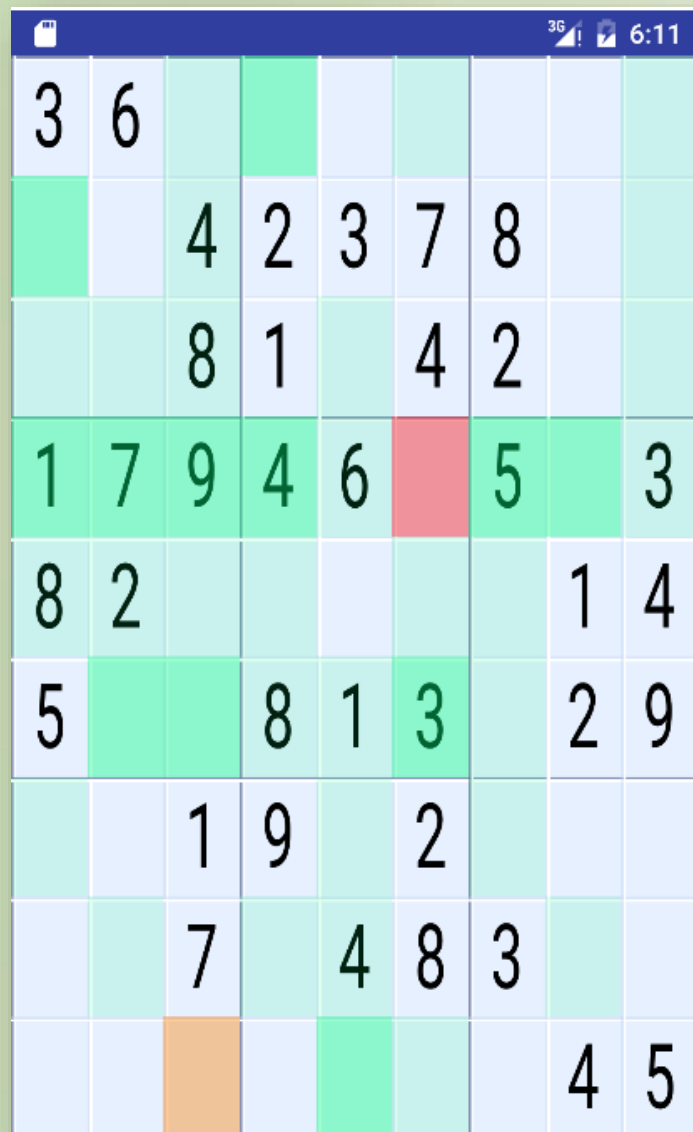
Essa parte do código será útil somente mais tarde, mas para que serve esse código?

3	6	
		4

3	6	
		4



# Desenhando o tabuleiro



3	6							
		4	2	3	7	8		
		8	1		4	2		
1	7	9	4	6		5		3
8	2						1	4
5			8	1	3		2	9
		1	9		2			
		7		4	8	3		
							4	5

Quais os passos envolvidos no desenho do tabuleiro?

Seria possível estabelecermos uma sequência de passos para esse desenho?

O que são as cores dos pequenos retângulos?



# Desenhando o tabuleiro

```
@Override
protected void onDraw(Canvas canvas) {
    // desenhe o fundo...
    Paint backgroundPaint = new Paint();
    backgroundPaint.setColor(ContextCompat.getColor(
        getContext(), R.color.puzzle_background));
    canvas.drawRect(0, 0, getWidth(), getHeight(), backgroundPaint);
    // desenhe as raias...
    // desenhe os números...
    // desenhe as dicas...
    // desenhe a área selecionada...
}
```

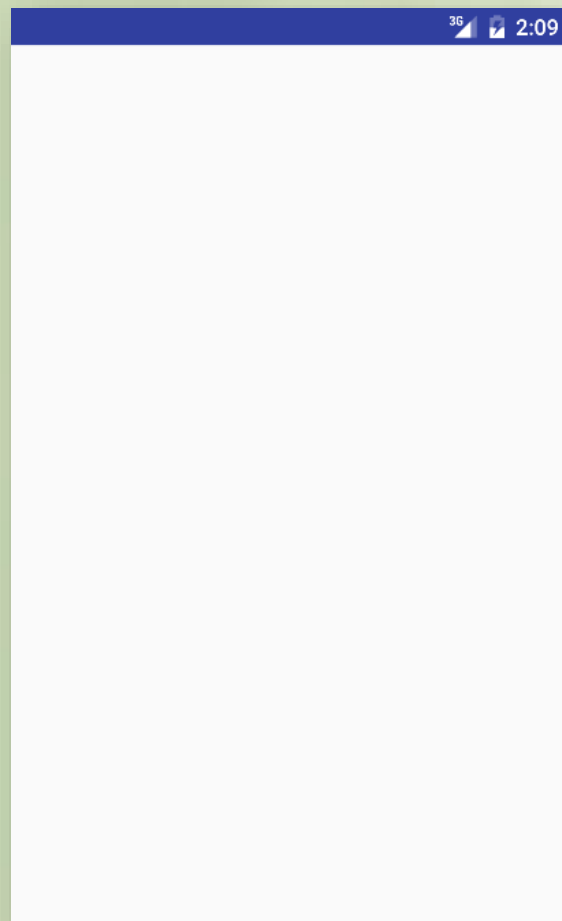
PuzzleView.java

```
<color name="puzzle_background">#ff000000</color>
<color name="puzzle_hilite">#ffffffff</color>
<color name="puzzle_light">#64c6d4ef</color>
<color name="puzzle_dark">#6456648f</color>
<color name="puzzle_foreground">#ff000000</color>
<color name="puzzle_hint_0">#64ff0000</color>
<color name="puzzle_hint_1">#6400ff80</color>
<color name="puzzle_hint_2">#2000ff80</color>
<color name="puzzle_selected">#64ff8000</color>
```

Colors.xml



# Desenhando o tabuleiro



Como desenhar a grade do Sudoku?

Lembre-se temos grades maiores e menores.

Como desenhar a grade menor?



# Auto-relevo

```
// desenhe as raias...
Paint darkPaint = new Paint();
darkPaint.setColor(ContextCompat.getColor(getContext(), R.color.puzzle_dark));
Paint hilitePaint = new Paint();
hilitePaint.setColor(ContextCompat.getColor(getContext(), R.color.puzzle_hilite));
Paint lighthPaint = new Paint();
lighthPaint.setColor(ContextCompat.getColor(getContext(), R.color.puzzle_light));

for (int i = 1; i < 9; i++){
    // deasenhar as linhas menores
    if (i % 3 != 0) {
        canvas.drawRect(0, i * height, getWidth(), i * height + 3, lighthPaint);
        canvas.drawRect(0, i * height + 4, getWidth(), i * height + 7, hilitePaint);
        canvas.drawRect(i * width, 0, i * width + 3, getHeight(), lighthPaint);
        canvas.drawRect(i * width + 4, 0, i * width + 7, getHeight(), hilitePaint);
    }
}
```

PuzzleView.java

Por que são sempre  
desenhadas duas  
linhas?



# Grade maior

Mas ainda  
falta o mais  
difícil:  
desenhar os  
números!

```
for (int i = 1; i < 9; i++){  
    // desenhar as linhas menores  
    if (i % 3 != 0) {  
        canvas.drawRect(0, i * height, getWidth(), i * height + 3, lightPaint);  
        canvas.drawRect(0, i * height + 4, getWidth(), i * height + 7, hilitePaint);  
        canvas.drawRect(i * width, 0, i * width + 3, getHeight(), lightPaint);  
        canvas.drawRect(i * width + 4, 0, i * width + 7, getHeight(), hilitePaint);  
    }  
    // desenhe as linhas maiores  
    else {  
        canvas.drawRect(0, i * height, getWidth(), i * height + 3, darkPaint);  
        canvas.drawRect(0, i * height + 4, getWidth(), i * height + 7, hilitePaint);  
        canvas.drawRect(i * width, 0, i * width + 3, getHeight(), darkPaint);  
        canvas.drawRect(i * width + 4, 0, i * width + 7, getHeight(), hilitePaint);  
    }  
}
```

# Preparando o terreno para os números

PuzzleView.java

```
// desenhe os números...  
Paint foreground = new Paint(Paint.ANTI_ALIAS_FLAG);  
foreground.setColor(ContextCompat.getColor(getContext(), R.color.puzzle_foreground));  
foreground.setStyle(Paint.Style.FILL);  
foreground.setTextSize(height * 0.75f);  
foreground.setTextScaleX(width / height);  
foreground.setTextAlign(Paint.Align.CENTER);
```

O que é *aliasing*?

Como representar  
e armazenar os  
números?

E como desenhar  
os números?



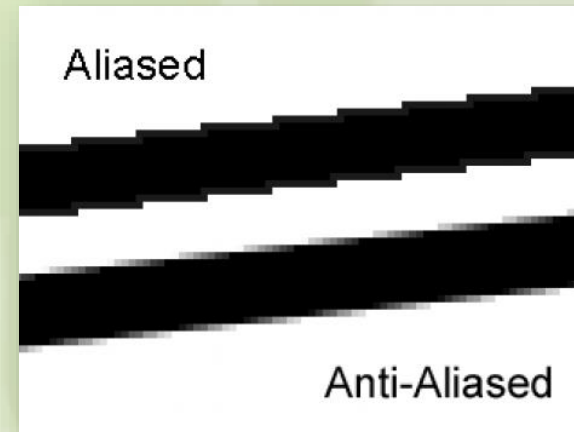


# Preparando o terreno para os números

PuzzleView.java

```
// desenhe os números...
Paint foreground = new Paint(Paint.ANTI_ALIAS_FLAG);
foreground.setColor(ContextCompat.getColor(getContext(), R.color.puzzle_foreground));
foreground.setStyle(Paint.Style.FILL);
foreground.setTextSize(height * 0.75f);
foreground.setTextScaleX(width / height);
foreground.setTextAlign(Paint.Align.CENTER);
```

O que é *aliasing*?



# Desenhando os números

PuzzleView.java

```
// desenhando os números no centro dos quadrados
Paint.FontMetrics fm = foreground.getFontMetrics();
// centralizando em X: usar alinhamento
float x = width / 2;
// centralizando em Y: mensurando posicionamento
float y = height / 2 - (fm.ascent + fm.descent) / 2;
for (int i = 0; i < 9; i++) {
    for (int j = 0; j < 9; j++) {
        canvas.drawText(this.game.getTileString(i, j), i * width + x, j
            * height + y, foreground);
    }
}
```

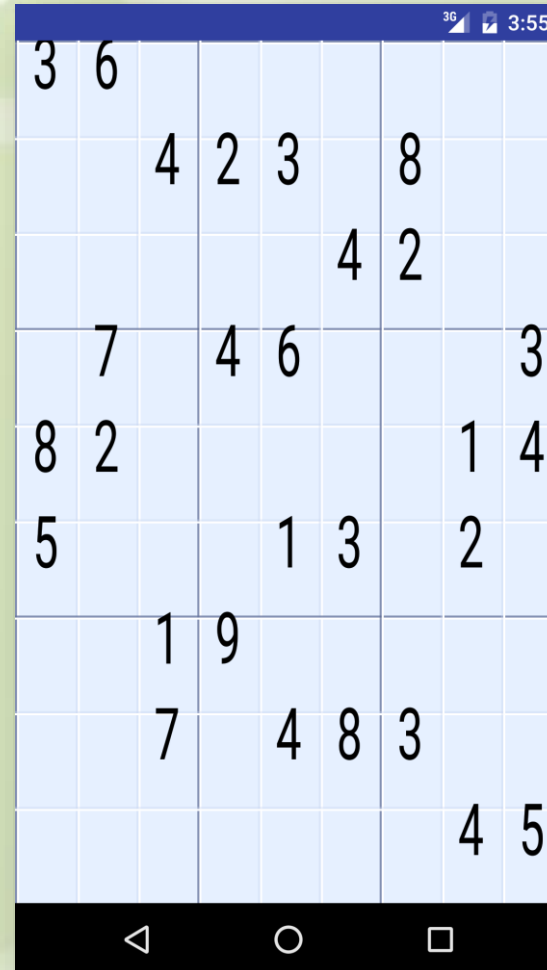
Como ficaria o tabuleiro sem **isso**?

Método ainda a ser implementado.  
Neste ponto vamos deixa-lo retornando sempre a letra "x"



# Aspect Ratio

- Esse ajuste, por  $height / 2 - (fm.ascent + fm.descente) / 2$ , é necessário, senão estaríamos colocando a base da fonte no centro do quadrado



# O que temos por enquanto

- Quatro tipos de linhas, formando dois tamanhos de grades, são desenhados
- Números são desenhados entre as grades
- Mas ainda falta interagir com o usuário
- E falta a lógica do jogo!

3	6								
		4	2	3		8			
					4	2			
	7		4	6					3
8	2							1	4
5				1	3		2		
		1	9						
		7		4	8	3			
								4	5



# Selecioneando células

- Seria interessante que o usuário pudesse selecionar uma célula do tabuleiro
  1. A célula selecionada deveria ser marcada com uma cor diferente
  2. O usuário deveria ser capaz de mover a célula selecionada

Tentemos primeiro resolver o problema 1: como marcar a célula selecionada?

Como colorir essa célula?



# Guardando a célula selecionada

- Retângulos são muito importantes em programação gráfica
  - Android possui até mesmo uma classe para representa-los: Rect.java

PuzzleView.java (ainda no método onDraw)

```
// desenhe a área selecionada...
Log.d(TAG, "selRect="+selRect);
Paint selectedPaint = new Paint();
selectedPaint.setColor(ContextCompat.getColor(getContext(), R.color.puzzle_selected));
canvas.drawRect(selRect, selectedPaint);
```

O que é esse  
objeto mesmo?



# Selecionando retângulos

PuzzleView.java

```
private void select(int x, int y) {  
    invalidate(selRect);  
    selX = Math.min(Math.max(x, 0), 8);  
    selY = Math.min(Math.max(y, 0), 8);  
    getRect(selX, selY, selRect);  
    invalidate(selRect);  
}
```

Qual o uso de  
invalidate  
nesse exemplo?

Como o método  
select é chamado?

A área retangular funciona  
como um cursor que recebe  
eventos do usuário. Como  
movimentar esse cursor?



# Criando um Key Pad

## Tela de teclas:

- A tela de teclas deve aparecer sempre que o usuário apertar o seletor do telefone.
- A tela permite que o usuário escolha o número que será armazenado naquela posição.

Como seria o layout dessa tela de teclas?





# Tabelas

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/keypad" android:orientation="vertical"
    android:layout_width="wrap_content" android:layout_height="wrap_content"
    android:stretchColumns="*">
    <TableRow>
        <Button android:id="@+id/keypad_1" android:text="1"/>
        <Button android:id="@+id/keypad_2" android:text="2"/>
        <Button android:id="@+id/keypad_3" android:text="3"/>
    </TableRow>
    <TableRow>
        <Button android:id="@+id/keypad_4" android:text="4"/>
        <Button android:id="@+id/keypad_5" android:text="5"/>
        <Button android:id="@+id/keypad_6" android:text="6"/>
    </TableRow>
    <TableRow>
        <Button android:id="@+id/keypad_7" android:text="7"/>
        <Button android:id="@+id/keypad_8" android:text="8"/>
        <Button android:id="@+id/keypad_9" android:text="9"/>
    </TableRow>
</TableLayout>
```



1	2	3
4	5	6
7	8	9

Quem “cria” o  
Key Pad? Como  
essa atividade é  
invocada?

# Chamando o Keypad

Game.java

```
protected void showKeypadOrError(int x, int y) {  
    int tiles[] = getUsedTiles(x, y);  
    if (tiles.length == 9) {  
        Toast toast = Toast.makeText(this, R.string.no_moves_label,  
            Toast.LENGTH_SHORT);  
        toast.setGravity(Gravity.CENTER, 0, 0);  
        toast.show();  
    } else {  
        Dialog v = new Keypad(this, tiles, puzzleView);  
        v.show();  
    }  
}
```

Para que serve  
**esse** código?

Gostaríamos de habilitar  
somente **aqueles**  
**números** que são válidos  
para a posição.

Como é a  
implementação da  
classe Keypad?



# A classe Keypad

Keypad.java

```
public class Keypad extends Dialog {  
    private final View keys[] = new View[9];  
    private View keypad;  
    private final int useds[];  
    private final PuzzleView puzzleView;  
  
    public Keypad(Context context, int useds[], PuzzleView puzzleView) {  
        super(context);  
        this.useds = useds;  
        this.puzzleView = puzzleView;  
    }  
  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        ...  
    }  
}
```

Como é a  
implementação  
de onCreate?



# Keypad.onCreate()

Keypad.java

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setTitle(R.string.keypad_title);  
    setContentView(R.layout.keypad);  
    findViews();  
    for (int element : useds) {  
        if (element != 0)  
            keys[element - 1].setVisibility(View.INVISIBLE);  
    }  
    setListeners();  
}
```

Para que  
serve **esse**  
código?

Implementemos  
os “**escutadores**  
**de eventos**”.



# Listeners

KeyPad.java

```
private void setListeners() {  
    for (int i = 0; i < keys.length; i++) {  
        final int t = i + 1;  
        keys[i].setOnClickListener(new View.OnClickListener() {  
            public void onClick(View v) {  
                returnResult(t);  
            }  
        });  
    }  
    keypad.setOnClickListener(new View.OnClickListener() {  
        public void onClick(View v) {  
            returnResult(0);  
        }  
    });  
}
```

E como seria a  
implementação  
de **returnResult**?



# Listeners

```
private void setListeners() {  
    for (int i = 0; i < keys.length; i++) {  
        final int t = i + 1;  
        keys[i].setOnClickListener(new View.OnClickListener() {  
            public void onClick(View v) {  
                returnResult(t);  
            }  
        });  
    }  
    keypad.setOnClickListener(new View.OnClickListener() {  
        public void onClick(View v) {  
            returnResult(0);  
        }  
    });  
}
```

```
private void returnResult(int tile) {  
    puzzleView.setSelectedTile(tile);  
    dismiss();  
}
```

Keypad.java

E qual a  
semântica de  
**dismiss()**?



# Listeners

KeyPad.java

```
private void setListeners() {  
    for (int i = 0; i < keys.length; i++) {  
        final int t = i + 1;  
        keys[i].setOnClickListener(new View.OnClickListener() {  
            public void onClick(View v) {  
                returnResult(t);  
            }  
        });  
    }  
    keypad.setOnClickListener(new View.OnClickListener() {  
        public void onClick(View v) {  
            returnResult(0);  
        }  
    });  
}
```

O que são esses  
objetos chamados  
“**Keys**”?



# Views e Teclas

KeyPad.java

```
private final View keys[] = new View[9];

private void findViews() {
    keypad = findViewById(R.id.keypad);
    keys[0] = findViewById(R.id.keypad_1);
    keys[1] = findViewById(R.id.keypad_2);
    keys[2] = findViewById(R.id.keypad_3);
    keys[3] = findViewById(R.id.keypad_4);
    keys[4] = findViewById(R.id.keypad_5);
    keys[5] = findViewById(R.id.keypad_6);
    keys[6] = findViewById(R.id.keypad_7);
    keys[7] = findViewById(R.id.keypad_8);
    keys[8] = findViewById(R.id.keypad_9);
}
```

Agora devemos concentrar-nos na lógica do jogo.

Primeiro, precisamos implementar `setTileIsValid`





# setTileIfValid

Game.java

```
protected boolean setTileIfValid(int x, int y, int value) {  
    if (fromPuzzleString(currentPuzzle)[y * 9 + x] != 0) {  
        return false;  
    }  
    int tiles[] = getUsedTiles(x, y);  
    if (value != 0) {  
        for (int tile : tiles) {  
            if (tile == value)  
                return false;  
        }  
    }  
    setTile(x, y, value);  
    calculateUsedTiles();  
    return true;  
}
```

Em poucas  
palavras: o que  
esse método  
está fazendo?

E como seria a  
implementação  
de **getUsedTiles**?



# setTileIfValid

```
protected boolean setTileIfValid(int x, int y, int value) {  
    if (fromPuzzleString(currentPuzzle)[y * 9 + x] != 0) {  
        return false;  
    }  
    int tiles[] = getUsedTiles(x,  
    if (value != 0) {  
        for (int tile : tiles) {  
            if (tile == value)  
                return false;  
        }  
    }  
    setTile(x, y, value);  
    calculateUsedTiles();  
    return true;  
}
```

Game.java

```
private int used[][][] = new int[9][9][];  
  
protected int[] getUsedTiles(int x, int y) {  
    return used[x][y];  
}
```



# Inserindo os Números

Game.java

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    Log.d(TAG, "onCreate");  
    int difficulty = getIntent().getIntExtra(KEY_DIFFICULTY, DIFFICULT_EASY);  
    puzzle = getPuzzle(difficulty);  
    calculateUsedTiles();  
    puzzleView = new PuzzleView(this);  
    setContentView(puzzleView);  
    puzzleView.requestFocus();  
}
```

Sempre que atualizamos o tabuleiro do jogo, o método onCreate é chamado, e é preciso **calcular** quais números não são mais válidos para cada célula. Como fazer isto?



# Calculando os ladrilhos usados

Game.java

```
private void calculateUsedTiles () {  
    for (int x = 0; x < 9; x++) {  
        for (int y = 0; y < 9; y++) {  
            used[x][y] = calculateUsedTiles (x, y);  
        }  
    }  
}
```

Como calcular quais números não podem ser usados para uma certa posição do Sudoku?



# Representando o Puzzle

- Podemos representar o puzzle como um arranjo de tamanho  $9 * 9$

```
private int puzzle[] = new int[9 * 9];
```

 Game.java

Neste caso, como  
implementar os  
métodos:  
getTile(x, y) e  
setTile(x, y, value)?



# Representando o Puzzle

- Podemos representar o puzzle como um arranjo de tamanho  $9 * 9$

```
private int puzzle[] = new int[9 * 9];

private int getTile(int x, int y) {
    return puzzle[y * 9 + x];
}

private void setTile(int x, int y, int value) {
    puzzle[y * 9 + x] = value;
}
```

Game.java



# O Sudoku Inicial

Game.java

- Precisamos inicializar o tabuleiro com um puzzle
- Este pode ser fácil, médio ou difícil
- Podemos representar esses puzzles como strings

```
private final String easyPuzzle =  
"360000000004230800000004200" +  
"070460003820000014500013020" +  
"0019000000007048300000000045";  
  
private final String mediumPuzzle =  
"650000070000506000014000005" +  
"007009000002314700000700800" +  
"500000630000201000030000097";  
  
private final String hardPuzzle =  
"0090000000080605020501078000" +  
"000000700706040102004000000" +  
"0007209030903010800000000600";
```



# Obtendo um puzzle novo

```
private int[] getPuzzle(int diff) {  
    String puz;  
    switch (diff) {  
        case DIFFICULT_HARD:  
            puz = hardPuzzle;  
            break;  
        case DIFFICULT_MEDIUM:  
            puz = mediumPuzzle;  
            break;  
        case DIFFICULT_EASY:  
        default:  
            puz = easyPuzzle;  
            break;  
    }  
    currentPuzzle = puz;  
    return fromPuzzleString(puz);  
}
```

Game.java

Agora, implemente o método toPuzzleString, que converta um arranjo de inteiros, descrevendo um puzzle, em uma string.





# toPuzzleString

Game.java

```
static private String toPuzzleString(int[] puz) {  
    StringBuilder buf = new StringBuilder();  
    for (int element : puz) {  
        buf.append(element);  
    }  
    return buf.toString();  
}
```

Faça o caminho inverso  
agora: implemente o  
método fromPuzzleString  
que converta uma string em  
um arranjo de inteiros.



# fromPuzzleString

Game.java

```
static protected int[] fromPuzzleString(String string) {  
    int[] puz = new int[string.length()];  
    for (int i = 0; i < puz.length; i++) {  
        puz[i] = string.charAt(i) - '0';  
    }  
    return puz;  
}
```



# getString(x, y)

```
for (int i = 0; i < 9; i++) {  
    for (int j = 0; j < 9; j++) {  
        canvas.drawText(this.game.getString(i, j), i * width + x, j  
            * height + y, foreground);  
    }  
}
```

Implemente o método getString, que retorna uma string descrevendo o número em uma posição do tabuleiro, ou a string vazia se aquela posição estiver vaga.



# getString(x, y)

```
public String getString(int x, int y) {  
    return getTile(x, y) == 0 ? "" : Integer.toString(getTile(x, y));  
}
```

Game.java



# Cores de ajuda

- Podemos colorir os ladrilhos de forma diferente dependendo de quantos valores possíveis cada ladrilho pode assumir



# Dicas

```
// A cor da dica é baseada no número de opções restantes
Paint hint = new Paint();
int c[] = { ContextCompat.getColor(getContext(), R.color.puzzle_hint_0),
           ContextCompat.getColor(getContext(), R.color.puzzle_hint_1),
           ContextCompat.getColor(getContext(), R.color.puzzle_hint_2), };
Rect r = new Rect();
for (int i = 0; i < 9; i++) {
    for (int j = 0; j < 9; j++) {
        int movesleé = 9 - game.getUsedTiles(i, j).length;
        if (movesleé < c.length) {
            getRect(i, j, r);
            hint.setColor(c[movesleé]);
            canvas.drawRect(r, hint);
        }
    }
}
```



# Efeitos especiais

## Números inválidos:

Se o usuário entrar números inválidos, que já foram utilizados na linha ou na coluna, então fazemos a tela tremer por um segundo.



# Animações simples

- Android permite-nos definir efeitos de animação simples em XML.
- Podemos chamar esses efeitos a partir de “setSelectedTile”

```
public void setSelectedTile(int tile) {  
    if (game.setTileIfValid(selX, selY, tile)) {  
        invalidate();  
    } else {  
        Log.d(TAG, "setSelectedTile: invalid: " + tile);  
        startAnimation(AnimationUtils.loadAnimation(game, R.anim.shake));  
    }  
}
```





# Animação em XML

anim/shake.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<translate
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:fromXDelta="0"
    android:toXDelta="10"
    android:duration="1000"
    android:interpolator="@anim/cycle_7" />
```

- O número de vezes, e a aceleração de uma animação também são definidas em um **arquivo XML**

anim/cycle\_7.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<cycleInterpolator
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:cycles="7" />
```



# Dúvidas

