Desenvolvendo Jogos com MIDP 2.0

Mark A. Patel
Principal Staff Engineer
Motorola

Vanessa Sabino
GUJ

Desenvolvendo Jogos com MIDP 2.0

Aprender sobre as novas funcionalidades do MIDP 2.0 para gráficos e jogos

Entender os conceitos necessários para utilizá-las efetivamente em suas aplicações

A História dos Jogos para Celulares

Jogos para celulares estiveram presentes desde o princípio...

A História dos Jogos para Celulares

1977 – Lançamento dos Celulares

A História dos Jogos para Celulares

1997 – Snake

A História dos Jogos para Celulares

2000 – MIDP 1.0

A História dos Jogos para Celulares

2003 – MIDP 2.0

Desenvolvendo Jogos com MIDP 2.0

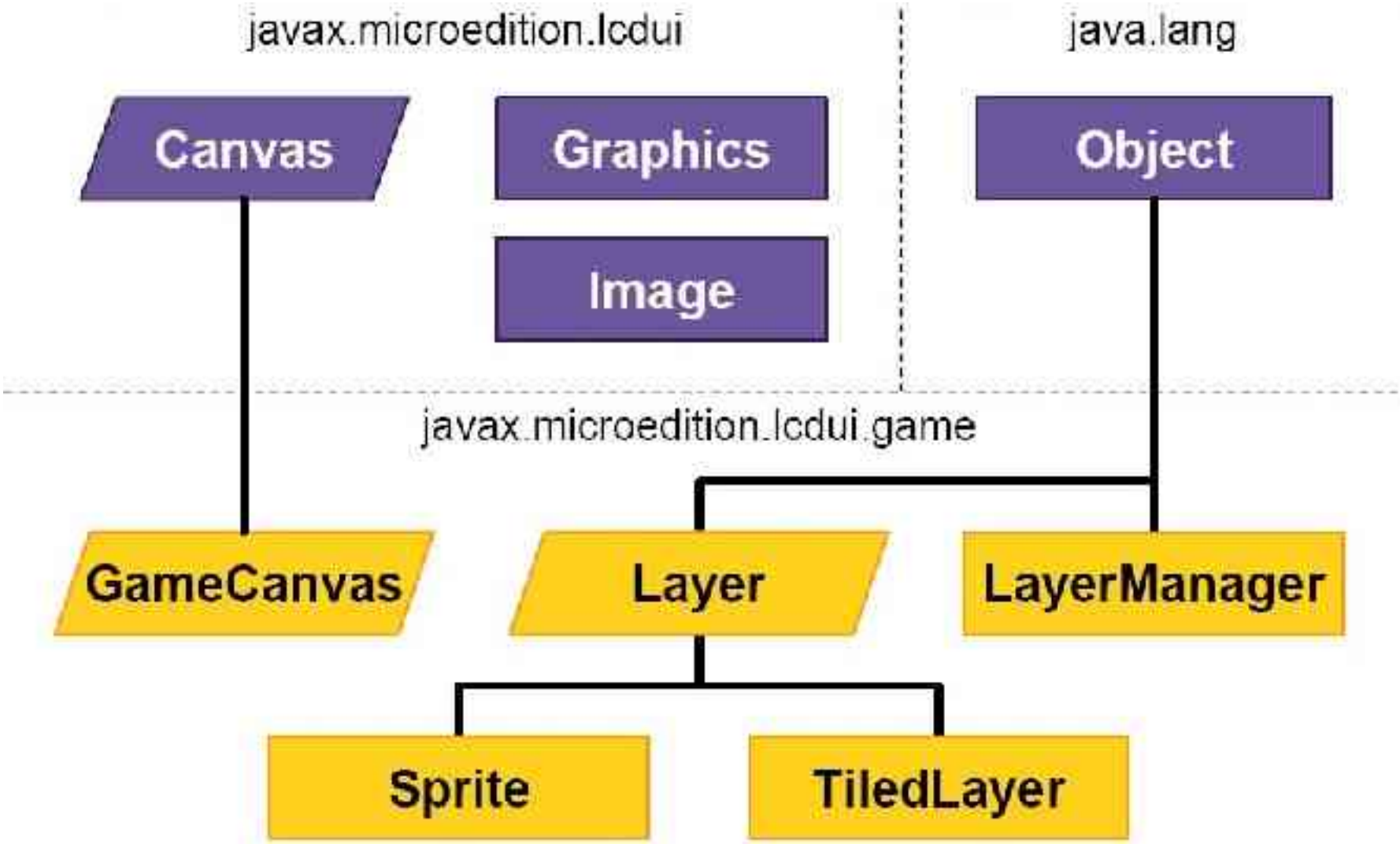
- A Game API
  - GameCanvas
  - Layer
  - Sprite
  - TiledLayer
  - LayerManager
  - Collision Detection
- Melhorias na manipulação gráfica em baixo nível

A Game API

A Game API

- Provê funcionalidades orientadas a jogos 2D
- Lida com várias tarefas comuns a várias aplicações
  - Simplifica o desenvolvimento
  - Reduz o tamanho da aplicação
  - Melhora a performance

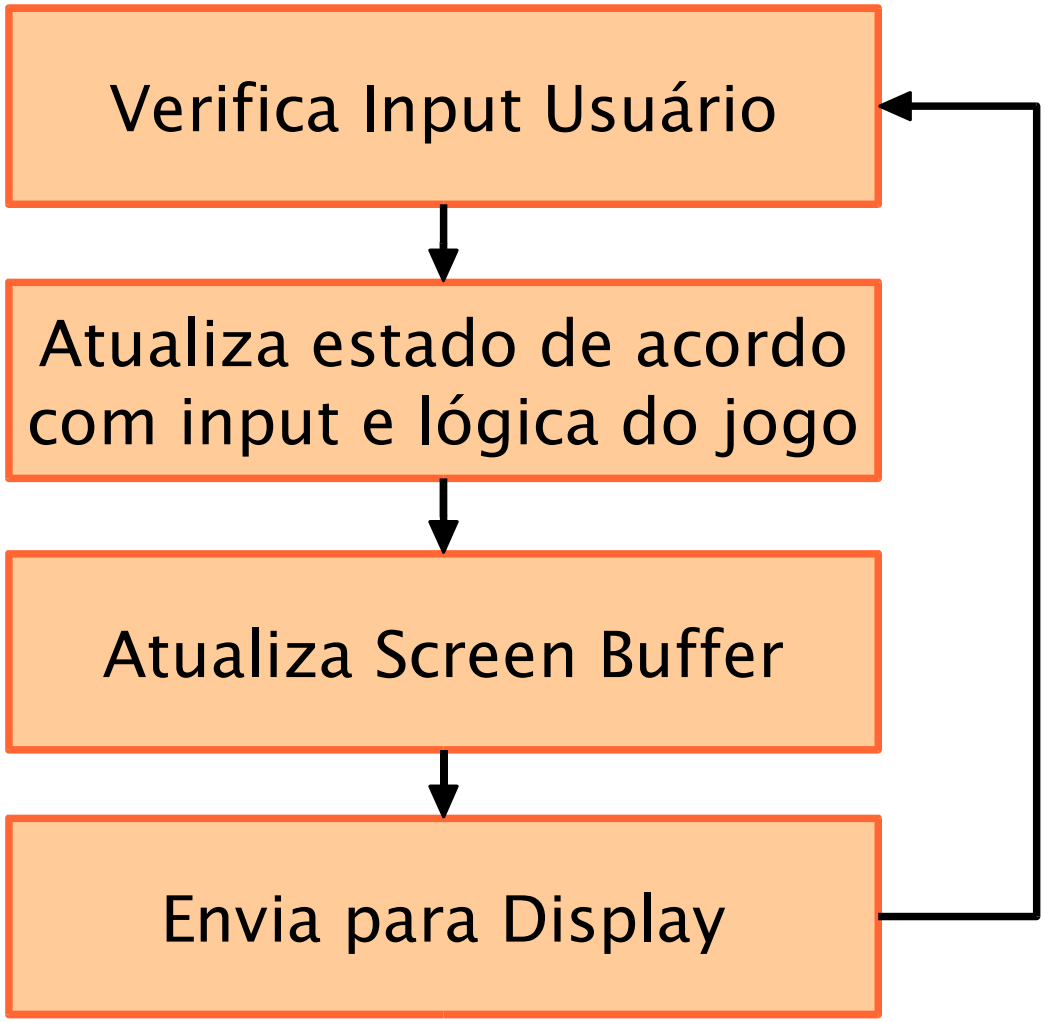
A Game API


javax.microedition.lcdui
java.lang
Canvas
Graphics
Image
Object
javax.microedition.lcdui.game
GameCanvas
Layer
LayerManager
Sprite
TiledLayer

GameCanvas

- Ligação com a interface com o usuário
- Mais apropriada para jogos que a Canvas
  - key polling
  - screen buffer dedicado
  - flushing síncrono

GameCanvas


Verifica Input Usuário
Atualiza estado de acordo com input e lógica do jogo
Atualiza Screen Buffer
Envia para Display

GameCanvas

KeyPolling

- método **`getKeyStates()`** retorna um número inteiro representando o estado das teclas de jogo
- Cada tecla é representada por um bit
  - 1 = pressionado
  - 0 = não pressionado
- É possível detectar mais de uma tecla pressionada simultâneamente
- Pode não ser suportado em todos aparelhos

GameCanvas

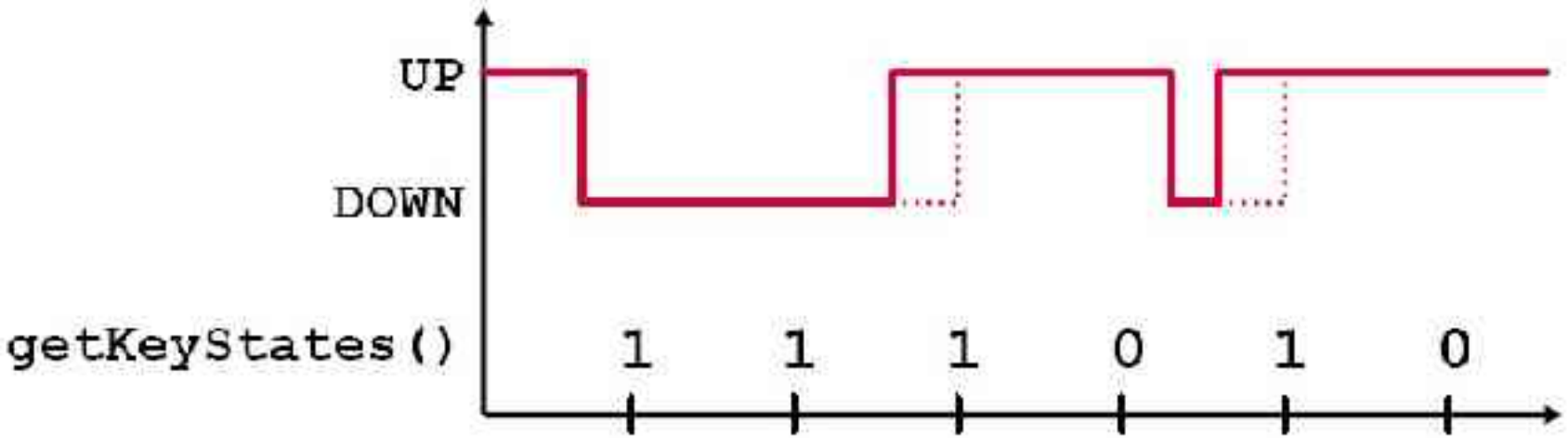
KeyPolling

```
int keys = getKeyStates();
if ((keys & UP_PRESSED) != 0)
  yPosition--;
if ((keys & DOWN_PRESSED) != 0)
  yPosition++;
if ((keys & FIRE_PRESSED) != 0)
  fire();
```

GameCanvas

KeyPolling

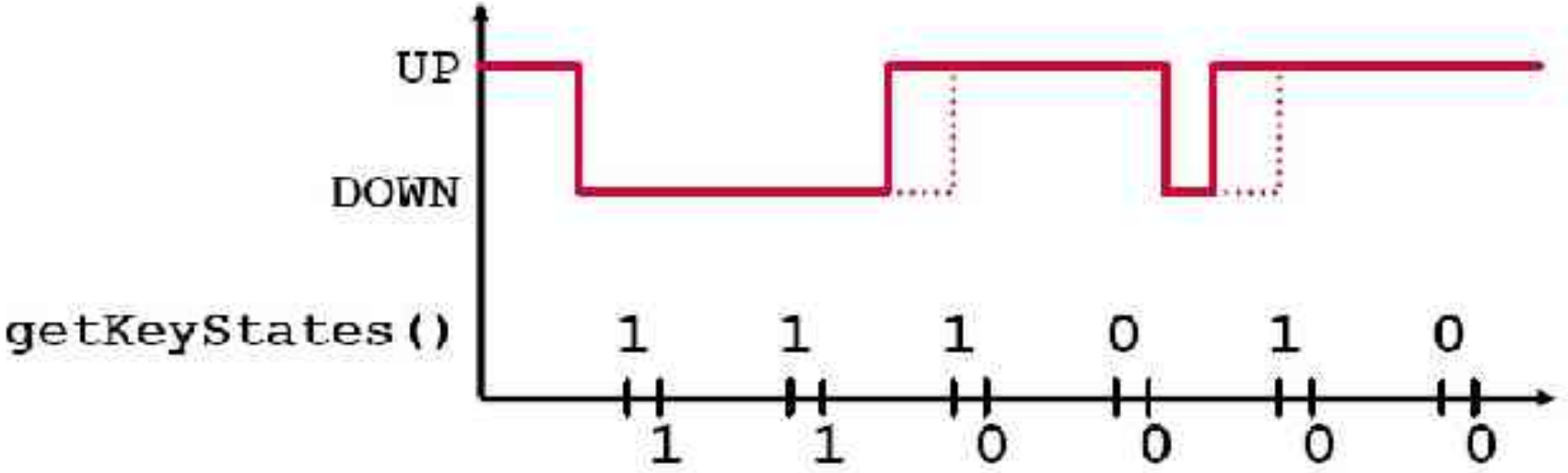
- As teclas pressionadas são armazenadas entre as chamadas do **`getKeyStates()`**



GameCanvas

KeyPolling

- Este comportamento pode ser desabilitado fazendo chamadas duplas ao método **`getKeyStates()`**



GameCanvas

Screen Buffer

- Dimensões iguais a da GameCanvas
  - **`getWidth()`**
  - **`getHeight()`**
- Não é modificado por outras entidades
- Objetos Graphics podem ser obtidos conforme necessário
- Flushing síncrono
  - Screen buffer inteiro ou parcial

Layer

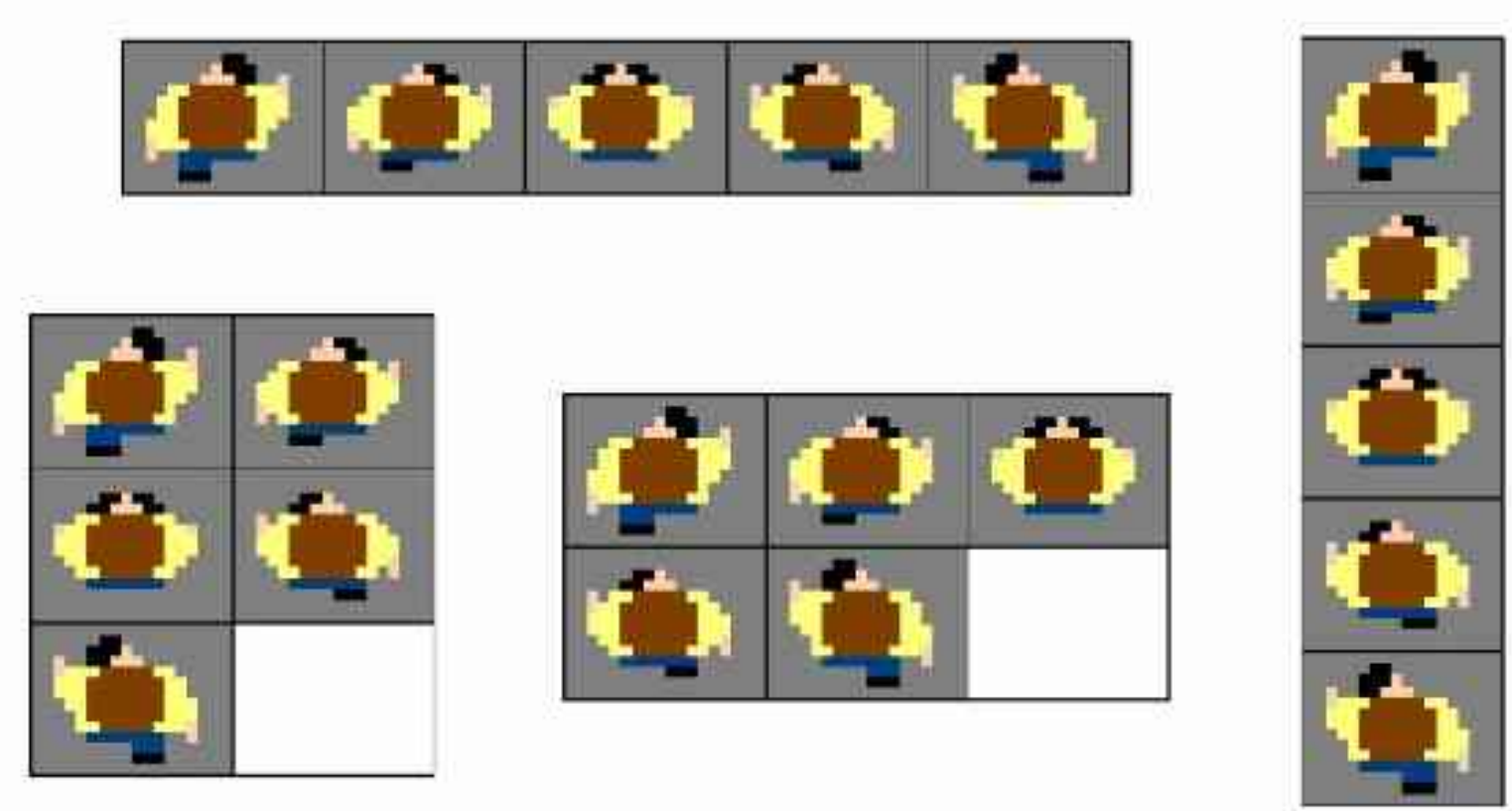
- Entidade visual básica
- Tamanho retangular
  - x, y, largura, altura
- Visível ou invisível
- Superclasse de Sprite e TiledLayer

Sprite

- Mostra um de vários quadros (*frames*)
- Transformações básicas
- *Pixel de referência* simplifica o posicionamento

Sprite

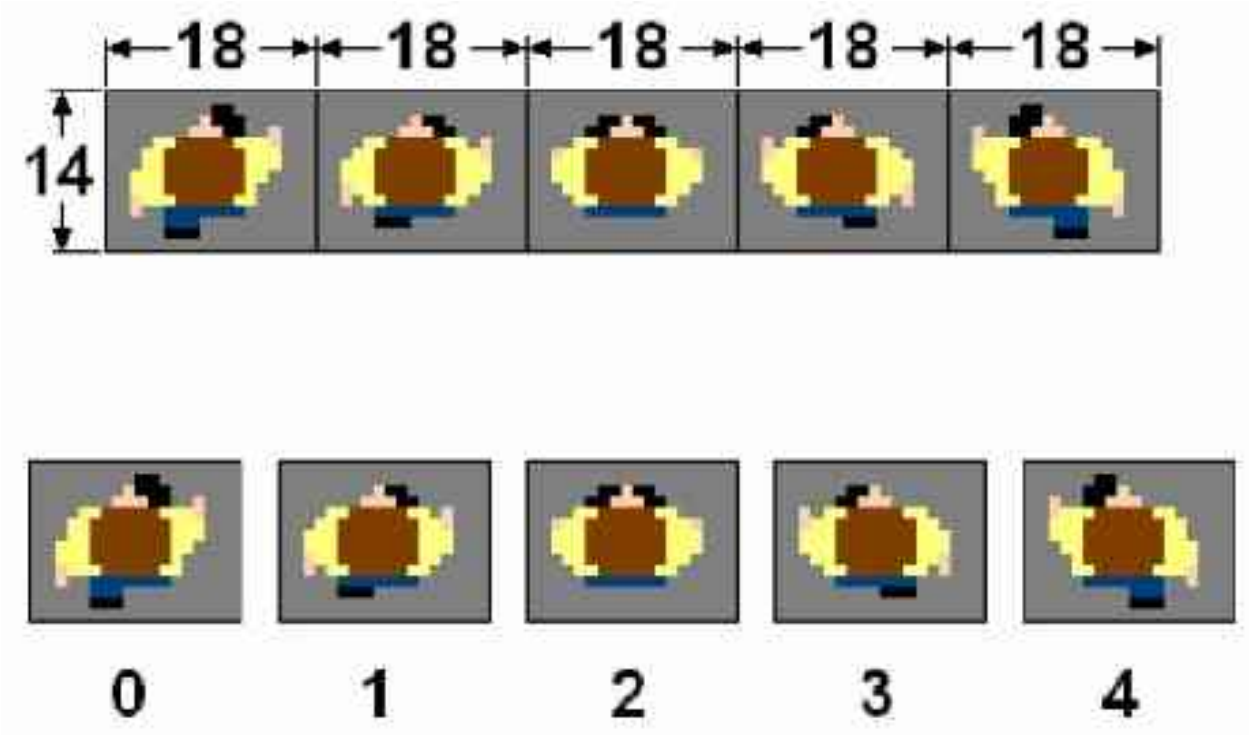
Frames



Sprite

Frames

```
Sprite dude = new Sprite(img, 18, 14);
```



Sprite

Frames

- Quadro atual é renderizado quando o método paint é chamado
- Controle direto do quadro a ser mostrado
  - `setFrameIndex(int frame)`
  - `nextFrame()`
  - `prevFrame()`

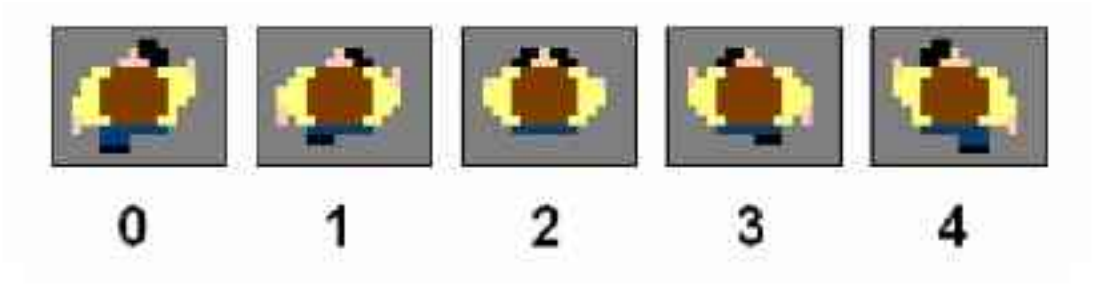
Sprite

Seqüências de Frames

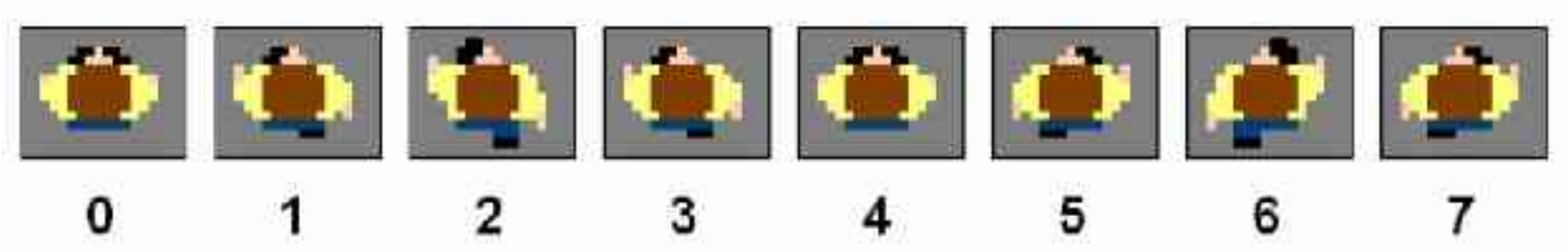
- Seqüência padrão
  - Mostra cada quadro em ordem
  - 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3...
- Seqüência especial
- Define uma ordem arbitrária em que os quadros aparecem
- Útil para animações
- Jeito fácil de variar o tempo em que as frames aparecem

Sprite

Seqüências de Frames




```
int[] seq = {2, 3, 4, 3, 2, 1, 0, 1};
dude.setFrameSequence(seq);
```



Sprite

Transformações

TRANS_NONE

TRANS_MIRROR

TRANS_ROT90

TRANS_MIRROR_ROT90

TRANS_ROT180

TRANS_MIRROR_ROT180

TRANS_ROT270

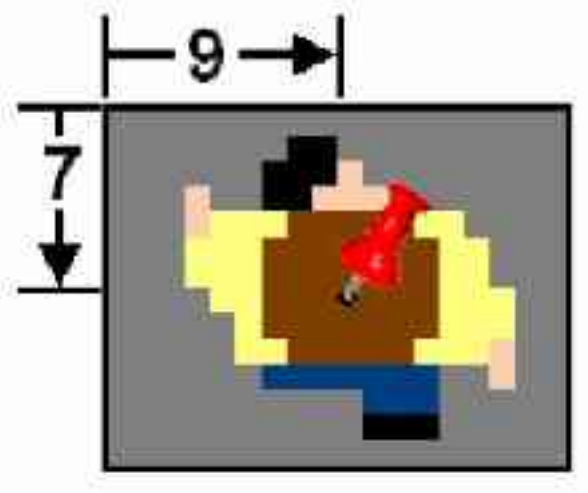
TRANS_MIRROR_ROT270

Sprite

Pixel de referência

- Um pixel é selecionado para definir a localização do Sprite
  - relativo ao canto superior esquerdo do quadro

```
dude.defineReferencePixel(9,7);
```

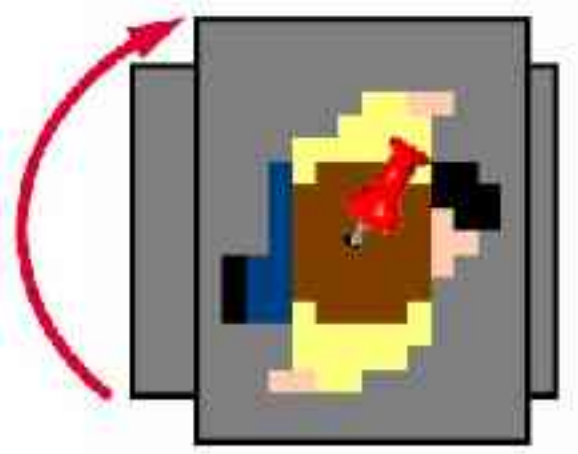


Sprite

Pixel de referência

- Funciona como o centro das transformações

```
dude.setTransform(Sprite.TRANS_ROT90);
```



Sprite

Pixel de referência

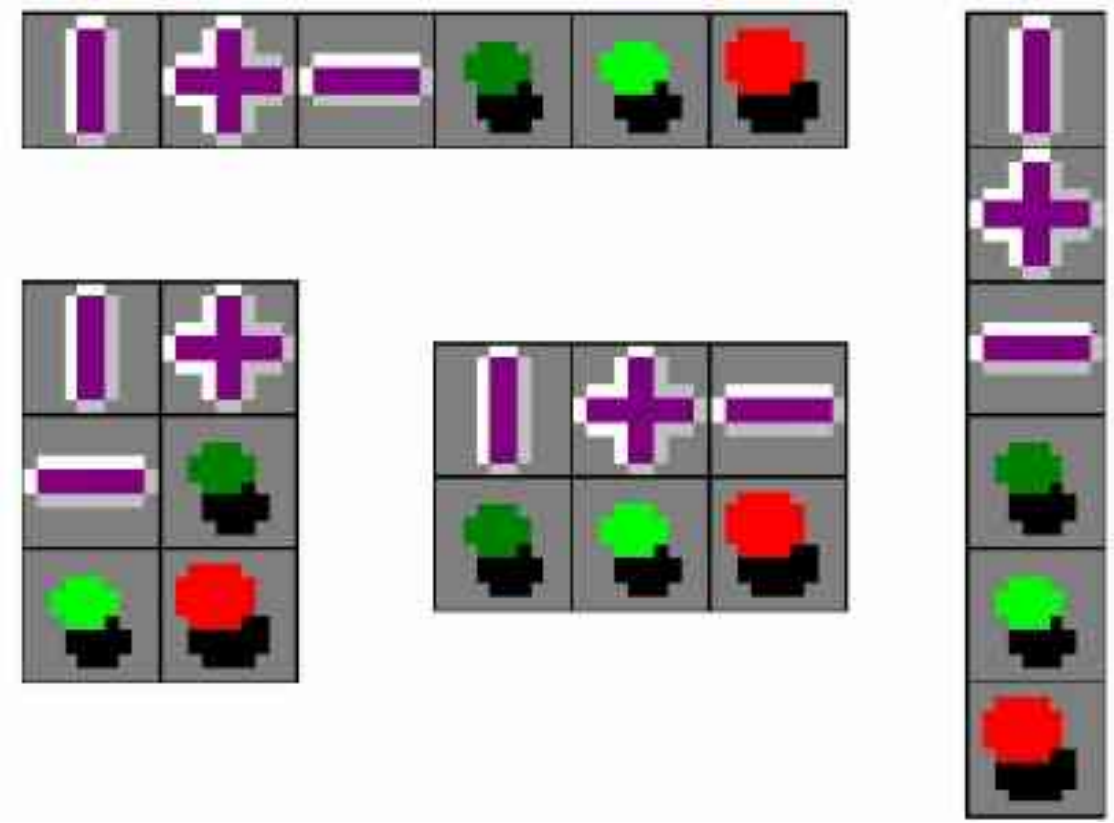
- Pode ser utilizado para determinar e obter a localização do Sprite
  - `setRefPixelPosition(int x, int y);`
  - `int getRefPixelX();`
  - `int getRefPixelY();`

TiledLayer

- Matriz de *células*
- Cada *célula* pode mostrar uma *“tile”*
- *tiles animadas* simplificam os efeitos de animação

TiledLayer

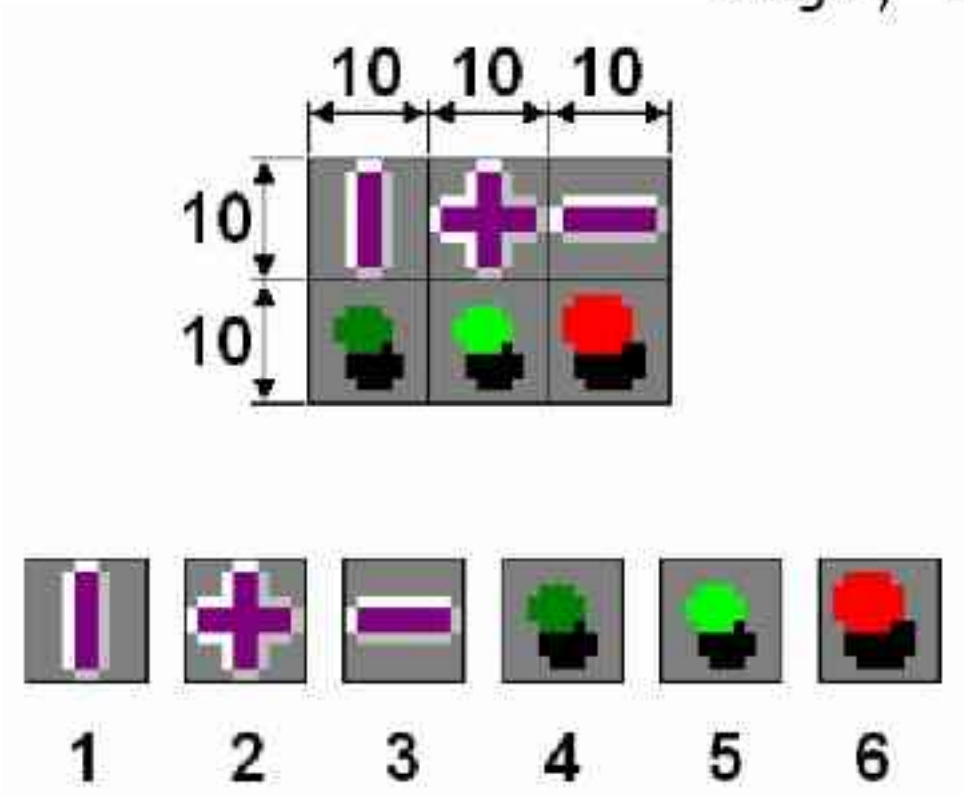
Tiles



TiledLayer

Tiles

```
TiledLayer maze = new TiledLayer(60, 60,
                                 img2, 10, 10);
```



TiledLayer

Células

- Podem mostrar uma tile ou ficar vazias (0)
  - Células vazias são transparentes
- O conteúdo de uma célula pode ser alterado a qualquer momento

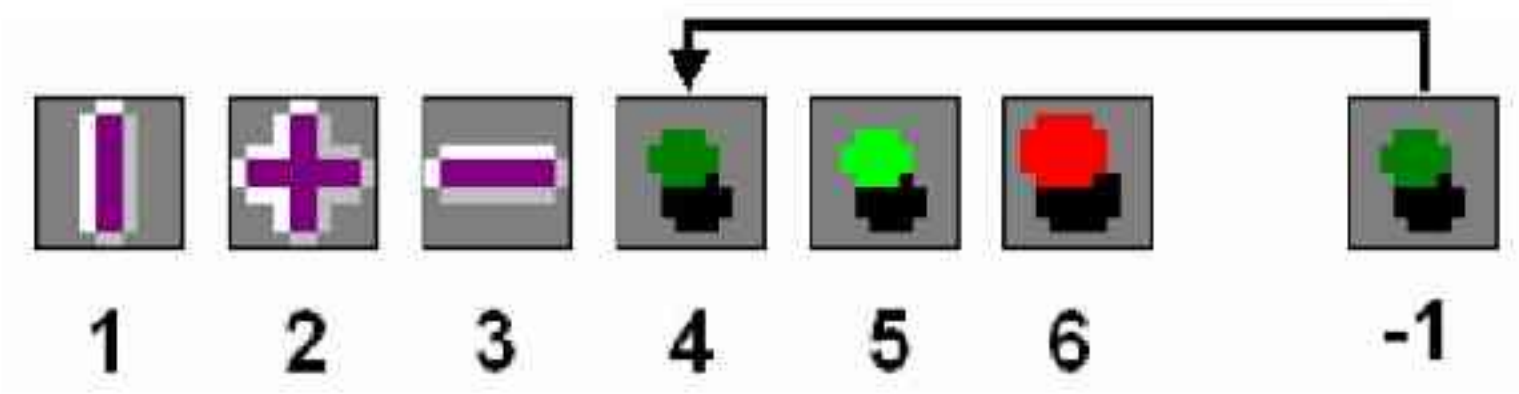
```
setCell(int col, int row,
        int tileIndex)
fillCells(int col, int row,
          int width, int height,
          int tileIndex)
```

TiledLayer

Tiles Animadas

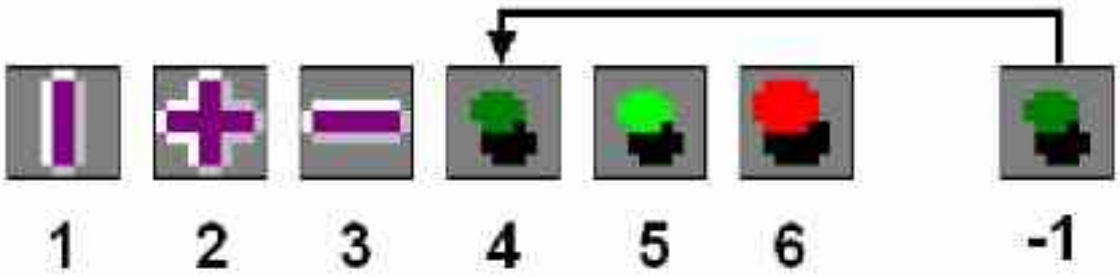
- Tiles especiais que são ligadas dinâmicamente a uma tile estática

```
int foodTile = maze.createAnimatedTile(4);
```

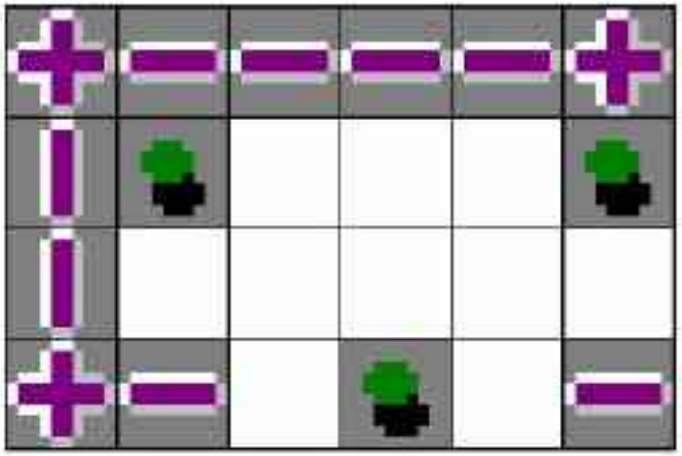


TiledLayer

Tiles Animadas



2	3	3	3	3	2
1	-1	0	0	0	-1
1	0	0	0	0	0
2	3	0	-1	0	3

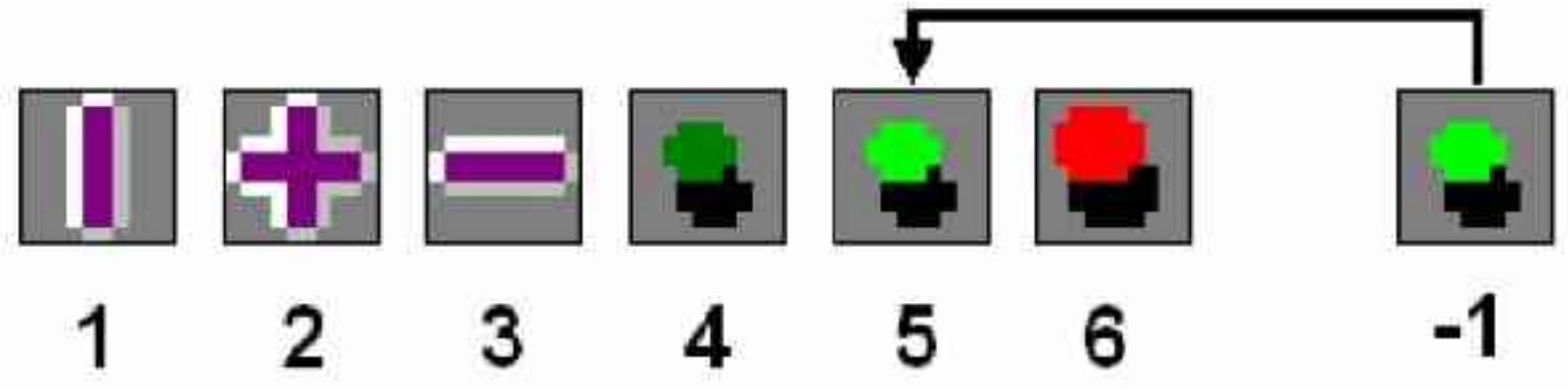


TiledLayer

Tiles Animadas

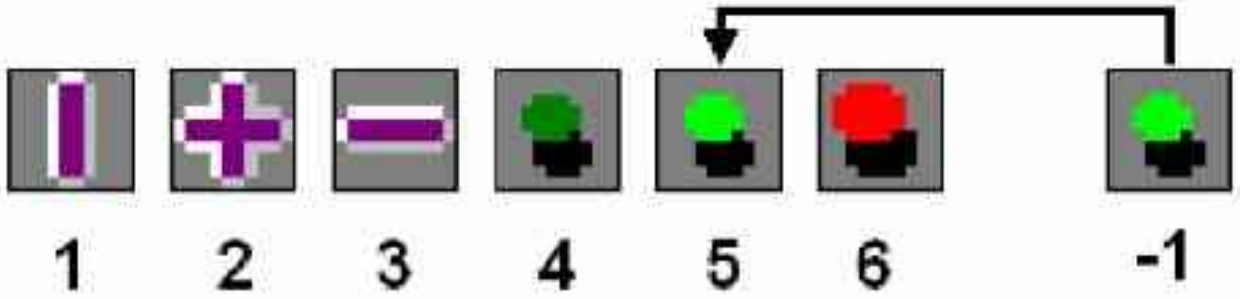
- A tile pode ser alterada conforme necessário

```
maze.setAnimatedTile(foodTile, 5);
```

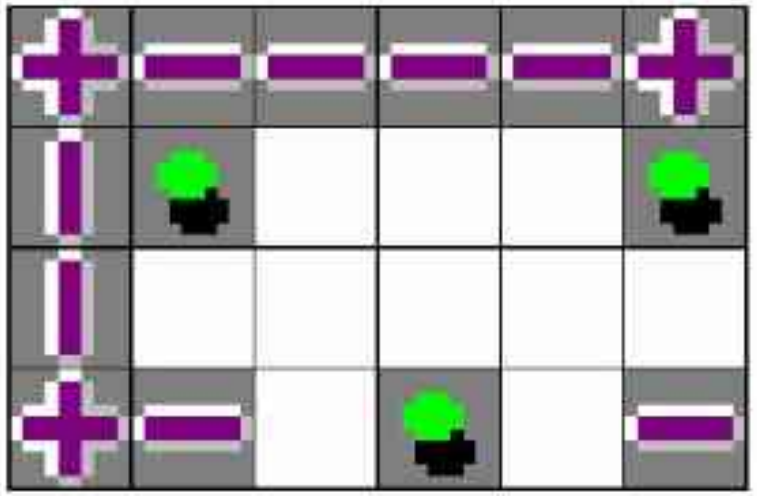


TiledLayer

Tiles Animadas



2	3	3	3	3	2
1	-1	0	0	0	-1
1	0	0	0	0	0
2	3	0	-1	0	3



LayerManager

- Gerencia uma lista de Layers
- Simplifica rolagem de tela e disposição dos elementos
- Automatiza a renderização das Layers

LayerManager

Adicionando camadas

- Layers são adicionadas a uma lista indexada
  - **`append(Layer l)`**
  - **`insert(Layer l , int index)`**
- Lista implica em uma z-order
  - Índice 0 é o mais próximo do usuário
  - Índices maiores estão mais longe do usuário

LayerManager

View Window

- Controla o que é visto pelo usuário
  - **`setViewWindow(int x, int y, int width, int height)`**
- Efeitos de scrolling são obtidos movendo a view window

LayerManager

Renderizando

- A visualização de uma **`LayerManager`** pode ser renderizada em uma localização específica na tela
  - **`paint(Graphics, int x, int y)`**
- Layers fora da área de visualização não são renderizadas

Detecção de Colisão

- Checa vários tipos de colisão:
  - **`Sprite vs. Sprite`**
  - **`Sprite vs. TiledLayer`**
  - **`Sprite vs. Image`**
- Métodos do **`Sprite`**
  - Retângulo de colisão definido pelo usuário
- Boundary-level ou pixel-level

Detecção de Colisão

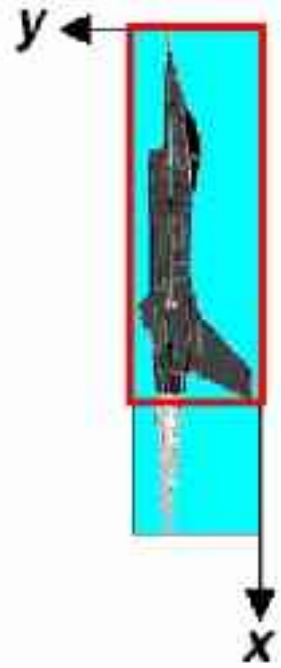
Retângulo de Colisão

- Define a região de um **`Sprite`** a ser incluída na detecção de colisão
- Ajusta-se automaticamente quando as transformações são aplicadas

Detecção de Colisão

Retângulo de Colisão

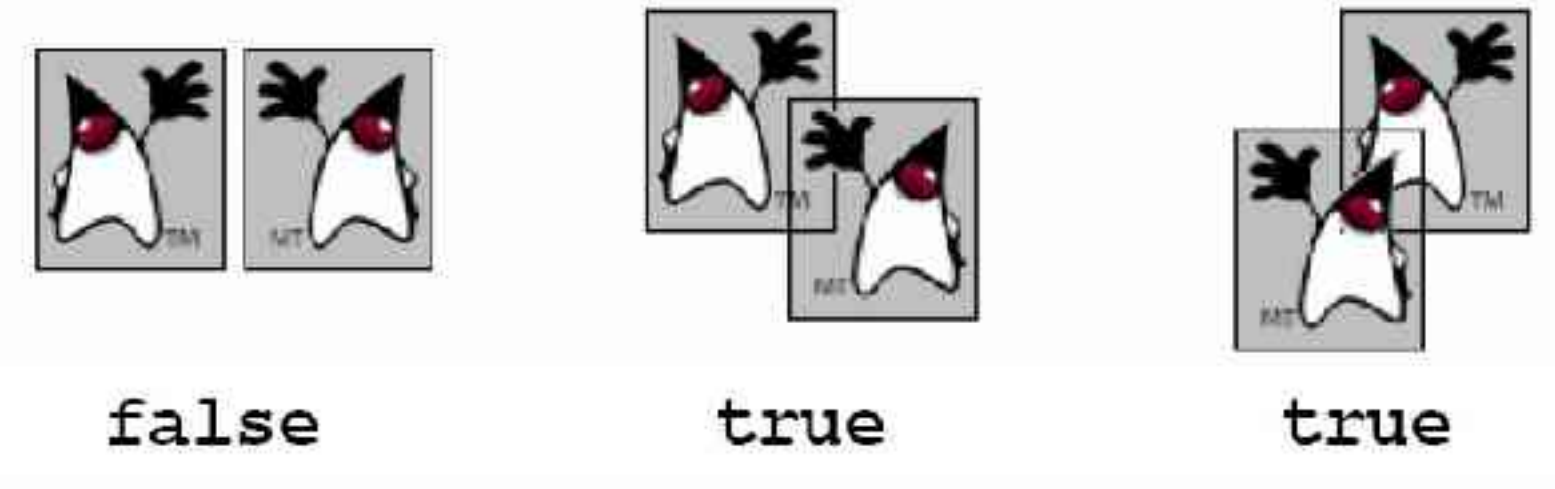
```
defineCollisionRectangle(int x, int y,
                         int width, int height)
```



Detecção de Colisão

Detecção pela borda

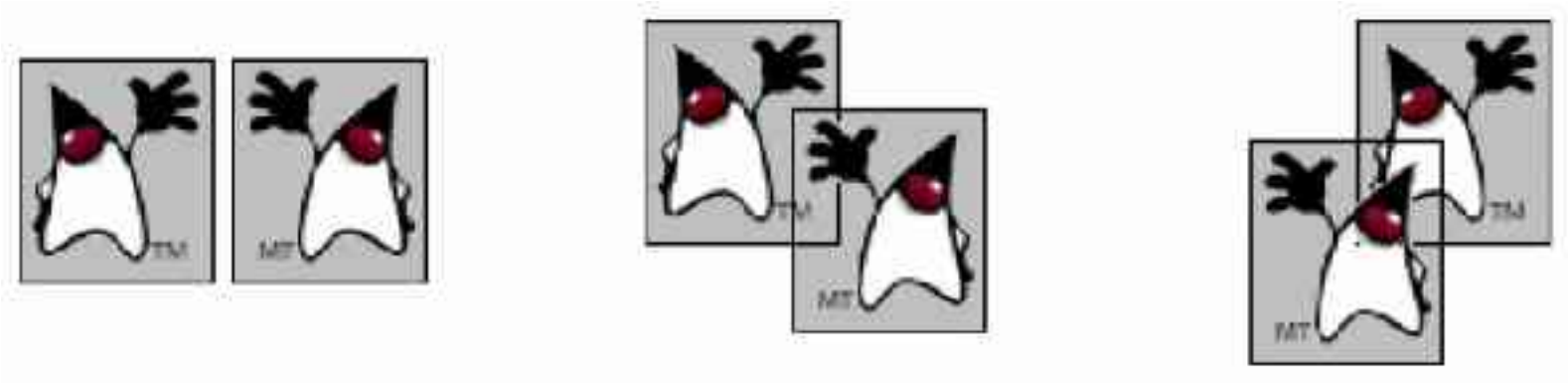
- Determinada pelas bordas de duas entidades
  - Rápida
  - Limitada a formas retangulares



Detecção de Colisão

Detecção pelo pixel

- Determina quando dois pixels opacos estão sobrepostos
  - Mais lenta
  - Pode detectar colisão entre formas arbitrárias



Melhorias para Gráficos em baixo nível

Melhorias para Gráficos

- Estende as capacidades básicas do MIDP 1.0
- Novos métodos na classe **`Graphics`**
- Melhorias na classe **`Image`**

Novos métodos na Graphics

FillsTriangle

- Preenche um triângulo utilizando a cor de desenho
- Os vértices podem estar em qualquer ordem

Novos métodos na Graphics

CopyArea

- Copia uma região retangular de pixels
- Origem e destino podem estar sobrepostos

Novos métodos na Graphics

DrawRegion

- Desenha uma região de uma imagem
  - x, y, altura, largura
- Transformações podem ser aplicadas à região
  - As mesmas que as definidas em Sprite
- Pontos âncora ficam em termos da região transformada

Novos métodos na Graphics

DrawRGB

- Renderiza uma região de valores RGB
- Dados fornecidos em uma array de inteiros
  - 0x00RRGGBB
  - 0xAARRGGBB
- Pode ser especificado um **`offset`** e **`scanlength`**

Melhorias em imagens

- Suporte básico de transparência é obrigatório
  - Alpha blending é opcional
- Apenas para imagens imutáveis
  - Imagem mutáveis são sempre opacas

Melhorias em imagens

CreateImage

- InputStream de dados binários
- Array de valores RGB
  - formato 0xAARRGGBB
- Região de uma imagem
  - x, y, altura, largura
  - Transformações

Melhorias em imagens

getRGB

- Obtém os valores dos pixels da região de uma imagem
- Dados armazenados em uma array de inteiros fornecida ao método
  - formato 0xAARRGGBB

Resumo

- A Game API provê funcionalidades específicas para jogos
- Melhorias no Graphics e Image provêem novas possibilidades e maior flexibilidade
- Aumenta a performance, simplifica o desenvolvimento e melhora a qualidade do conteúdo

Se você lembrar apenas de uma coisa...

O futuro dos jogos para celulares

está nas suas mãos!

Q&A

JavaOne℠

Sun's 2003 Worldwide Java Developer Conference

java.sun.com/javaone/

Mais Informações

Artigos

- http://wireless.java.sun.com/blueprints/articles/game/
- http://www.forum.nokia.com/main/1,6566,050_20,00.html
- http://www.microjava.com/
- http://www.jasonlam604.com/books.php

Comunidade

- http://www.guj.com.br/
- j2me-list@soujava.dev.java.net
-

Contato

- vanessasabino@yahoo.com.br
- http://www.alemdojava.cjb.net/ ⇦ código fonte