Como desenvolver um jogo em Java ME - Parte 4

Em nossa última lição, completamos nosso GameCanvas com as principais interações entre os elementos. Agora que temos todos os elementos de jogo construídos é hora de melhorar a interface de nossa aplicação.

A idéia aqui é usar algumas imagens para representar nossas entidades em vez de usar métodos da classe Graphics para desenhar elementos primitivos. Com isto em mente, eu criei algumas imagens, baseadas no pacote SpriteLib (gratuito) para usar em nosso jogo.

Imagem

Para acessar e mostrar estas imagens em nosso Midlet, precisamos usar outra classe do grupo de baixo nível de elementos gráficos, a class Image. Esta classe armazena dados da imagem independentemente do display do aparelho, em um buffer off-screen. As imagens são mutáveis ou imutáveis, dependendo da maneira com que são criadas. Imagens imutáveis são geralmente criadas carregando-se dados de arquivos, resource bundles, ou da rede. Para criar imagens imutáveis você pode usar um dos métodos estáticos *createImage*, mas lembre-se que tais imagens não podem ser modificadas após sua criação

Imagens mutáveis são criadas como imagens brancas, contendo apenas pixels da mesma cor. Para criá-las use os construtores da classe Image. A aplicação pode então renderizar em uma imagem mutável chamando o método *getGraphics()* da classe Image para obter uma instância da classe Graphics própria para este propósito.

Para nosso jogo usaremos imagens imutáveis para representar as entidades de jogo. Vamos começar modificando nossa classe Pad. Primeiro, precisamos criar uma imagem em nosso construtor:

```
Image image;
public Pad() {
    try {
      image = Image.createImage("/pad.png");
      width = image.getWidth();
      height = image.getHeight();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
```

Então reescrevemos o método paint() para que desenhe a imagem recém-criada:

```
public void paint(Graphics g) {
    g.drawImage(image, x,y, Graphics.TOP | Graphics.LEFT);
  }
```

Como você pode perceber, estamos utilizado o formato PNG, que é o único obrigatório em todas as implementações de MIDP (Português); alguns devices podem suportar outros formatos, como JPEG mas PNG é a única opção segura para todos os devices.

Rodando seu midlet agora, você verá sua nova raquete com uma visual bem melhor! Vamos fazer a mesma coisa para a entidade Bola:

```
 Image image;
 public Ball() {
    try {
      image = Image.createImage("/ball.png");
      width = image.getWidth();
      height = image.getHeight();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }

  public void paint(Graphics g) {
    g.drawImage(image, x,y, Graphics.TOP | Graphics.LEFT);
  }
```

Se você rodar seu midlet agora, terá uma bola com um visual interessante. Então vamos fazer a mesma coisa com os tijolos. Nossa imagem para os tijolos é uma grande imagem com vários tijolos dentro dela, então precisamos de um jeito de recordar cada imagem de tijolo de dentro da imagem maior. Poderíamos fazer isso utilizando o método *setClip* mas em MIDP 2.0 (Português) existe um caminho mais fácil, a classe Sprite.

Sprite

"Sprite" é um termo comum em jogos. Ele se refere a um elemento visual composto por uma imagem, normalmente animado, que pode ser movido pela tela independentemente de outros elementos. A classe Sprite em MIDP 2.0 é a representação deste conceito, pois ela permite a criação de Sprites baseados em Imagems com diversos frames. Ela pode mudar os frames, controlar animações e checar por colisões com outros elementos.

(C) Copyright Nokia 2013. All rights reserved.

Poderíamos usar todas essas funcionalidades em nossas entidades de jogo, mas por agora vamos começar com a classe Brick.

```
public static int BRICK_FRAMES = 20;
Image image = null;
Sprite sprite = null;
public Brick(){
    // load image
    image = Image.createImage("/bricks.png");
    // create the sprite with 20 frames one for each brick
    sprite = new Sprite(image,image.getWidth()/BRICK_FRAMES, image.getHeight());
    width = sprite.getWidth();
    height = sprite.getHeight();
  }
```

Este código cria um Sprite com 20 frames (quadros), um para cada tijolo disponível na imagem. Agora, antes de pintarmos um Sprite, precisamos modificar o frame que iremos usar:

```
  public void paint(Graphics g) {
    if (active){
      sprite.nextFrame();
      sprite.setPosition(x, y);
      sprite.paint(g);
    }
  }
```

Agora se você rodar seu midlet, terá vários tijolos coloridos. Um problema que temos agora é que estamos carregando a imagem "bricks.png" para cada Brick criado, o que não é muito otimizado. Vamos refazer o código para resolver este problema:

```
  public Brick(Image image, int numFrames, int frameSelected){
    sprite = new Sprite(image,image.getWidth()/numFrames,image.getHeight());
    // set frame
    sprite.setFrame(frameSelected);
    // get size for collision detection
    width = sprite.getWidth();
    height = sprite.getHeight();
  }

  public void paint(Graphics g) {
    if (active){
      sprite.setPosition(x, y);
      sprite.paint(g);
    }
  }
```

Agora carregamos a imagem apenas uma vez, em nosso método init(), e selecionamos um frame para ser usado durante toda a vida de um tijolo.

```
   // create bricks
    Image bricksFrames = null;
    try {
      bricksFrames = Image.createImage("/bricks.png");
    } catch (IOException e) {
      e.printStackTrace();
    }
    Brick brick = new Brick(bricksFrames, 20, 0);
    bricks = new Vector();
    for (int i=0; (i*(brick.width+2)) < getWidth(); i++){
      brick = new Brick(bricksFrames, 20, i);
      brick.sprite.setFrame(i);
      brick.x = (i*(brick.width+2));
      brick.y = 20;
      bricks.addElement(brick);
    }
```

Agora você tem uma linha legal de tijolos com sua própria cor! Nosso jogo está ficando bastante atraente visualmnete. Na próxima lição, aprenderemos como salvar os maiores escores.

Downloads:

- Full Source Code
- Jad File
- Jar File

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.