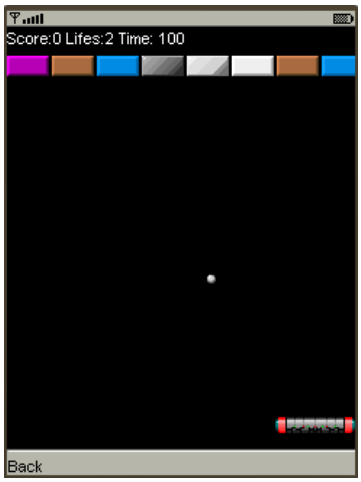
Como desenvolver um jogo em Java ME - Parte 3

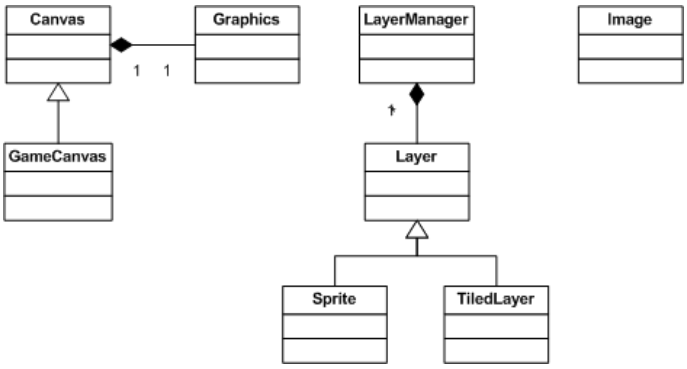
No final de nossa última lição completamos a interface gráfica para nosso jogo Arkanoid, exceto pela tela do jogo em si. Nosso objetivo é produzir um jogo tal qual o mostrado no screenshot abaixo:



Para a tela do game, não podemos usar os elementos de UI de alto nível porque precisamos ter total controle da maneira que nossos elementos são desenhados e como reagir a eventos de teclado. Para fazer isto precisamos usar as classes de interface gráfica de baixo nível disponíveis em J2ME.

As classes pertencentens ao grupo de baixo nível permitem controle detalhado dos elementos da tela e dos eventos. Com elas você pode especificar posição, cor e tamanho. A desvantagem de ter mais controle é a menor portabilidade, pois você precisa adaptar suas classes às diferentes capacidades e funcionalidades dos hardwares disponíveis.

O diagrama a seguir mostra as principais classes deste grupo:



O ponto de entrada é a classe Canvas, que nos dá acesso aos eventos de sistema:

- **keyPressed(), keyRepeated(), keyReleased()**, notificam o Canvas quando o teclado está sendo usado
- **pointerPressed(), pointerDragged(), pointerReleased()**, notificam o Canvas quando o pointer está sendo usado, para devices possuidores de touch screen.
- **paint()**, notifica o Canvas sobre quando ele precisa pintar a tela; este método lhe dá acesso ao objeto Graphics.
- getWidth(), getHeight(), fornecem acesso ao tamanho atual da tela disponível para se desenhar elementos.

A classe Graphics provê uma série de métodos usados para desenhar elementos primários na tela, a saber:

- **drawLine()**, desenha uma linha
- **drawRect()**, fillRect(), desenham um retângulo (cheio) na tela
- **drawArc()**, fillArc, desenha um arco na tela, o que pode ser usado para desenhar círculos
- **drawChar()**, drawChars, drawString, desenham caracteres na tela usando a fonte atual
- **drawImage()**, desenha uma imagem na tela
- **setFont()**, define a fonte atual sendo usada
- **setColor()**, define a cor atualmente sendo usada pelos métodos de desenho

Para criar nosso jogo, precisamos estender a classe Canvas, e implementar nosso próprio método paint().

```
import javax.microedition.lcdui.Canvas;
import javax.microedition.lcdui.Graphics;

public class MyCanvas extends Canvas{

  int width;
  int height;

  public MyCanvas() {

  }

  protected void paint(Graphics g) {
    // stores width and height
    width = getWidth();
    height = getHeight();
```


(C) Copyright Nokia 2013. All rights reserved.

```
    // set background color
    g.setColor(0,0,0);
    // clear screen
    g.fillRect(0, 0, width, height);
    // draw a red circle that represents a ball
    g.setColor(255,0,0);
    g.drawArc(100, 100, 5, 5, 0, 360);
    // draws a blue rectangle for the pad
    g.setColor(0,0,255);
    g.fillRect(100, 200, 15, 15);
  }
}
```

Para ativar nosso Canvas precisamos criá-lo em nosso MIDlet e então mostrá-lo em nosso método commandAction()

```
public Displayable initGameCanvas() {
  if (gameCanvas == null){
    gameCanvas = new MyCanvas();
    // add a back Command to return to the menu screen
    gameCanvas.addCommand(initBackCommand());
    // set the listener to our actions
    gameCanvas.setCommandListener(this);
  }
  return gameCanvas;
}
```

Após adicionar este código, você poderá entrar na tela New Game no menu principal, o que irá mostrar uma fantástica tela preta com uma bola vermelha e uma raquete azul; não é muito impressionante mas é um começo :) Agora precisamos animar tais elementos, vamos ver como fazê-lo.

Loop de jogo

Antes de continuar, você precisa entender a maneira como funciona uma típica aplicação de jogo. Um jogo ou animação é construído com o princípio de repetir continuamente um trecho de código. Este trecho monitora o valor de variáveis e atualiza o estado do jogo continuamente. Baseado no estado jogo, o código desenha/pinta/repinta a tela de jogo com os elementos que fazem o game. Os valores das variáveis podem mudar por causa da interação com o usuário ou do comportamento interno do jogo.

A excecução repetitiva é posta em efeito através de um loop contínuo. Antes de entrar no loop, uma variável pode ser checada para definir se o jogo deve ainda estar rodando, e se não, o loop pode ser encerrado. O código no loop deve permitir ao thread atual de execução dormir por alguns milisegundos para controlar a taxa na qual o update do estado do jogo é feito (isto é, a velocidade com que o jogo deve ser atualizado).

Para colocar os parágrafos acima em termos de código:

```
public class MyCanvas extends GameCanvas implements Runnable{
…
public void start() {
  run = true;
  Thread t = new Thread(this);
  t.start();
}

public void stop() {
  run = false;
}

public void run(){
  init();
  while (run){
    // update game elements, positions, colisions, etc..
    updateGameState();
    // check user input
    checkUserInput();
    // render screen
    updateGameScreen();
    // redraws screen
    flushGraphics();
    // controls at which rate the updates are done
    Thread.sleep(10);
  }
}
…
}
```

Como você ter notado, estamos usando a classe GameCanvas, que é uma especialização de Canvas, otimizada para jogos. Ela usa as seguintes técnicas:

- Double buffering, a classe GameCanvas usa uma imagem off-screen que é usada para todas as operações de pintura e desenho. Quando toda a pintura é feita, ela pode ser renderizada de uma só vez na tela principal, usando o método flushGraphics(), o que evita flickering e provê uma animação mais suave.
- Armazena o estado das teclas em um array, e através do método getKeyStates(), você pode acessar um vetor de bits que indica o estado de cada tecla usando as constantes definidas em Canvas:DOWN_PRESSED

Além do uso de GameCanvas, estamos usando um Thread para manter a animação rodando independentemente dos eventos do MIDlet; desta forma nossa animação jamais esperará por eventos do sistema para atualizar a si mesma. Retornando ao nosso jogo, Arkanoid, precisamos saber implementar a lógica específica para tal jogo. Em nosso caso, três tipos de entidades estão presentes:

- A raquete, nosso avatar é apenas um pequêno retângulo que se move para a direita e para a esquerda na parte inferior da tela.
- A bola, que inicia conectada à raquete, e quando você pressiona o botão "FIRE", se move verticalmente, com a velocidade horizontal da raquete.
- Os tijolos, blocos estáticos no top da tela que desaparecem quando atingidos pela bola.

O objetivo do jogo é fazer todos os tijolos desaparecerem o mais rápido possível, e não deixar a bola passar através do fundo da tela, longe da raquete. Precisamos monitorar as seguintes variáveis:

- escore do jogador, para cada tijolo destruído o jogador ganha 10 pontos
- número de vidas remanescentes, cada vez que a bola cai pela parte inferior da tela, o jogador perde uma vida
- tempo remanescente, já que o jogador tem um limite de tempo para completar o jogo.

Então comecemos a programar! Primeiramente vamos definir algumas classes para representar nossas entidades de jogo:

(C) Copyright Nokia 2013. All rights reserved.

Como você pode ver pelo diagrama, definimos uma classe chamada Entity. Esta será a classe-parente para todas as entidades do nosso jogo, provendo as seguintes funcionalidades:

- **x,y**: posição atual
- **speedX, speedY**: velocidade atual
- **witdth, height**: tamanho atual
- **update()**: método onde definiremos o comportamento da entidade.
- **paint()**: método onde desenharemos nossa entidade
- **collided()**: função que permite a checagem de colisão entre entidades.

Então estenderemos a classe Entity para cada um dos elementos do nosso jogo e implementaremos os métodos update() e paint() Para a bola, temos:

```
public class Ball extends Entity {
  public int radium = 2;

  public Ball(int radium){
    this.radium = radium;
    width = radium * 2;
    height = radium * 2;
    // red color
    this.color = 0x00FF0000;
  }

  /**
   * Paints the ball using a circle
   */
  public void paint(Graphics g) {
    g.setColor(color);
    g.fillArc(x, y, radium*2, radium*2, 0, 360);
  }

  /***
   * Updates the ball position.
   */
  public void update() {
    // update position
    oldX=x;
    oldY=y;
    x += speedX;
    y += speedY;
  }
}
```

Para a raquete:

```
public class Pad extends Entity{
  int minLimit = 0;
  int maxLimit = 1;

  public Pad(int width, int height) {
    this.width = width;
    this.height = height;
  }

  public void paint(Graphics g) {
    g.setColor(0,0,255);
    g.fillRect(x, y, width, height);
  }

  public void update() {
    // change x position according the speed
    x += speedX;
    // check if world bounds are reached
    if (x < minLimit) {
      x = minLimit;
    }
    if (x+width > maxLimit){
      x = maxLimit - width;
    }
  }
}
```

Para os tijolos:

```
public class Brick extends Entity {
  boolean active = true;

  public Brick(int color){
    this.color = color;
  }

  public void paint(Graphics g) {
    // only paints if still active
    if (active){
      g.setColor(color);
      g.fillRect(x, y, width, height);
    }

  }

  public void update() {
    // the bricks don't move
  }
}
```

Agora precisamos criar e configurar todas essas classes em nosso Canvas; para isso criaremos um método init() na classe Canvas:

```
public void init(){
    // resets lifes
    lifes = 3;
```


(C) Copyright Nokia 2013. All rights reserved.

```
    // resets score
    score = 0;
    // resets time
    time = 0;
    // bricks hit
    bricksHit = 0;
    // create a pad
    pad = new Pad(getWidth()/10,getWidth()/10/4);
    pad.x = (this.getWidth()-pad.width) / 2;
    pad.y = this.getHeight() - (2*pad.height);
    pad.maxLimit = getWidth();
    pad.minLimit = 0;

    // create ball
    ball = new Ball(4);
    ball.x = getWidth() / 2;
    ball.y = getHeight() / 2;
    ball.speedX = 1;
    ball.speedY = 1;
    // set collision limits
    wallMinX = 0;
    wallMaxX = getWidth();
    wallMinY = 0;
    // to allow to get out of screen
    wallMaxY = getHeight() + 4 * ball.radium;

    // create bricks
    Brick brick;
    bricks = new Vector();
    for (int i=0; (i*(BRICK_WIDTH+2))<getWidth(); i++){
      brick = new Brick(Util.setColor(255,0,0));
      brick.width  = BRICK_WIDTH;
      brick.height = BRICK_HEIGHT;
      brick.x = (i*(brick.width+2));
      brick.y = 20;
      bricks.addElement(brick);
    }
  }
```

Agora que temos todos os objetos criados precisamos atualizar seu estado e pintá-los, então precisamos adicionar mais código para os métodos updateGameState() e updateGameScreen():

```
// draws elements to the screen
  protected void updateGameScreen(Graphics g) {
    // stores width and height
    width = getWidth();
    height = getHeight();
    // set background color
    g.setColor(0,0,0);
    // clear screen
    g.fillRect(0, 0, width, height);
    // draw score
    g.setColor(255,255,255);
    g.drawString("Score:"+score+" Lifes:"+lifes+" Time: "+time, 0, 0, Graphics.TOP|Graphics.LEFT);
    // draw game elements
    pad.paint(g);
    ball.paint(g);
    // draw bricks stored in the Vector bricks
    for (int i=0; i < bricks.size(); i++){
      Brick brick = (Brick)(bricks.elementAt(i));
      brick.paint(g);
    }
  }
  // updates state of all element in the game
  public void updateGameState(){
    pad.update();
    ball.update();

    checkBallCollisionWithWalls();
    checkBallCollisionWihPad();
    checkBallCollisionWithBricks();
    checkBallOutOfReach();

    // check if bricks ended
    if (bricksHit == bricks.size()){
      run = false;
    }
  }
```

A última coisa faltando é conectar os movimentos da raquete aos pressionamentos de teclas do teclado do aparelho:

```
  // update game entities according to use presses on keypad
  public void checkUserInput() {
    int state = getKeyStates();
    if ( (state & GameCanvas.LEFT_PRESSED) > 0) {
      // move left
      pad.speedX=-1;
    } else if ( (state & GameCanvas.RIGHT_PRESSED) > 0) {
      // move right
      pad.speedX=1;
    } else {
      // don't move
      pad.speedX=0;
    }
  }
```

Rodando o jogo no emulador você agora deve ter uma tela de jogo real, onde você pode jogar seu próprio jogo, mover a raquete para acertar a bola e apagar todos os tijolos!

Na próxima aula usaremos imagens em vez de elementos primitivos para ter uma interface melhor em nosso jogo.

Downloads:

- Full Source Code
- Jad File
- Jar File

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.