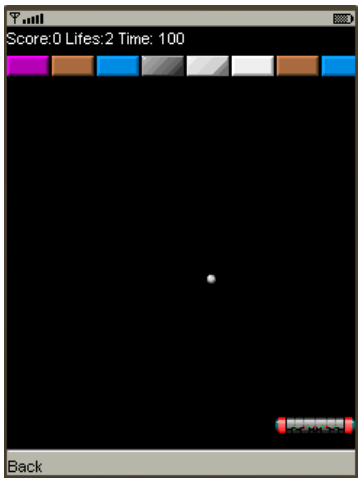
Developing a 2D game in Java ME - Part 3

This article shows how to use Java ME's low-level interface classes to create the game screen and access key presses. This is the third in a series of articles that cover all the basics of developing an application for mobile devices using Java ME, learning the main libraries, classes, and methods available in Java ME.

Introduction

24 Aug 2008

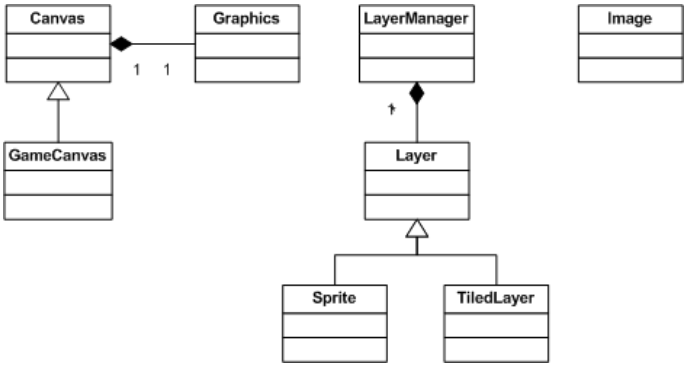
At the end of the previous article we completed the game menu interface for the Arkanoid game. The Game screen was, however, not yet created. The goal of this article is to produce a game screen that is similar to the following screenshot.



The high-level UI elements cannot be used for the game screen because we need to have full control of the way the game elements are drawn and how the game reacts to keypad events. In order to do this, we need to use the low-level interface classes available in Java ME.

The classes belonging to the low-level group allow detailed control of the screen elements and events. Using these classes you can specify the position, color, and size. The trade-off of more control is less portability because you need to adapt to the capabilities of each device.

The following diagram shows the main classes of this group:



The entry point is the Canvas class, which gives you access to the system events:

- **keyPressed(), keyRepeated(), keyReleased()** notify the canvas when the keypad is used.
- **pointerPressed(), pointerDragged(), pointerReleased()** notify the canvas when the pointer is used, available on phones with a stylus.
- **paint()** notifies the Canvas when it needs to paint the screen. This method gives access to the Graphics object.
- **getWidth(), getHeight()** give access to the current size of the screen available to be drawn on.

The Graphics class provides the methods for direct drawing on the screen:

- **drawLine()** draws a line.
- **drawRect(), fillRect()** draws a rectangle on the screen.
- **fillTriangle()** draws a filled triangle.
- **drawArc(), fillArc()** draws an arc on the screen, can be used to draw circles.
- **drawChar(), drawChars(), drawString()** draws characters to the screen using the current font.
- **drawImage()** draws a bitmap image on the screen.
- **setFont()** defines the current font.
- **setColor()** defines the current color being used by the draw methods.

To create the game screen we need to extend the Canvas class and implement a custom paint method.

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

public class MyCanvas extends Canvas {
```


(C) Copyright Nokia 2013. All rights reserved.

```
  int width;
  int height;

  public MyCanvas() {

  }

  protected void paint(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 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);
  }
}
```

To activate the Canvas, create it in the MIDlet class and display it on the CommandAction method.

```
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;
}
```

After you add this code, you can enter the New Game option in the main menu that will display a black screen with a red ball and a blue pad. Now these elements need to be animated.

Game loop

Before you continue, you need to understand the way typical game applications work. A game or animation is built by repetitiously executing a piece of code. This piece of code tracks the value of the variables and updates the game state accordingly. Based on the game state, the code draws/paints/repaints the game screen with the elements that build the game. The values of the variables may change because of user interaction or internal game behavior.

This is created by putting the code that is repeated in a continuous loop. Before entering the loop, a variable can be checked to see if the game should still be running. If not, the loop can be exited. The code in the loop should allow the current execution thread to sleep every few milliseconds to control the rate at which the game state is updated (that is, how fast the game screen should be refreshed). To put it in coding terms:

```
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, collisions, etc.
      updateGameState();
      // check user input
      checkUserInput();
      // render screen
      updateGameScreen(getGraphics());
      // redraw screen
      this.flushGraphics();
      // controls at which rate are the updates done
      try {
        Thread.sleep(10);
      } catch (InterruptedException e) {
        e.printStackTrace();
      }
    }
    midlet.endGame(lifes, score, time);
  }
…
}
```

As you may have noticed, the GameCanvas class is used. The GameCanvas is a special case of the Canvas class, optimized for games. It uses the following techniques:

- Double buffering. The GameCanvas uses an off-screen image which is used for all painting operations. When painting is completed, it can be rendered to the screen using the `flushGraphics()` method. This eliminates flickering and gives a smoother animation.
- Storing the key state in arrays. Through the `getKeyStates()` method you can access a bit array that indicates the state of each key.

Besides using GameCanvas, use a Thread to keep the animation running independently of MIDlet events. This way the animation will never wait for system events to repaint itself. In the Arkanoid game we now need to implement the specific game logic for the game. The game has three kinds of entities:

- The pad, our game avatar, is just a small rectangle that moves from left to right in the bottom of the screen.
- The ball, when hit with the pad it bounces to other direction
- The bricks are static blocks in the upper part of the screen. When they are hit by the ball they disappear.

(C) Copyright Nokia 2013. All rights reserved.

The goal of the game is to make the bricks disappear as fast as possible, and not let the ball get away through the bottom of the screen. The game needs to keep track of three variables:

- Score of the player. The player earns 10 points each time a brick is hit.
- Number of lifes remaining. Each time the ball goes through the bottom of the screen the player looses a life.
- Time remaining. The player must complete the game in a limited time.

First define some classes to represent the game entities.

The class Entity will be the parent for all the game entities. It provides the following functionalities:

- **x,y**: Defines the current position of the entity.
- **speedX, speedY**: Defines the speed of the entity.
- **width, height**: Defines the size of the entity.
- **update()**: Defines the the behavior of the entity.
- **paint()**: Method where the graphics class is used to draw the entity.
- **collided()**: Helper function that allows to check collision between entities.

Next, extend the class Entity for each game element and implement the update and paint methods. For the ball:

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

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

  /**
   * paints the ball
   */
  public void paint(Graphics g) {
    g.setColor(color);
    g.fillArc(x, y, radius*2, radius*2, 0, 360);
  }

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

For the pad:

```
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 to speed
    x += speedX;

    // check if the world bounds are reached
    if (x < minLimit) {
      x = minLimit;
    }

    if (x + width > maxLimit){
      x = maxLimit - width;
    }
  }

}
```

And finally for the bricks:

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

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

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

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


(C) Copyright Nokia 2013. All rights reserved.

Now create and configure all these classes in the canvas class. Create an `init()` method in the Canvas class.

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

    // 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 a 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 ball get out of screen
    wallMaxY = getHeight() + 4 * ball.radius;

    // 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);
    }
  }
```

After all objects have been created, update their state and paint them. Add some code to the `updateGameState()` and `updateGameScreen()` methods:

```
  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);
    }
  }
```

```
  public void updateGameState() {
    pad.update();
    ball.update();

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

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

The game needs to react to keypad events to move the player pad:

```
  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;
    }
  }
```

Now if you run the game in the emulator you should have a real game screen, where you can play your own game: Move the pad, hit the ball, and erase all the bricks.

(C) Copyright Nokia 2013. All rights reserved.

The next article explains how to use images to get a better looking game.

Downloads

- Source files
- Binary files

Go to Developing a 2D game in Java ME - Part 4

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.