MIDP 图形编程简介

现状：	草稿
版本：	0.9.4
日期：	2002 年 11 月 25 日
作者：	Forum Nokia

目 录

否认声明：

本文提供的内容适用“ 概不保证” (as is)原则。即没有任何形式的保证，包括对产品可销售、适合特定目的以 及其它由本文任何建议、规范和范例衍生出来的任何保证。另外，本文提供的信息是初级的，因此在最终版本确定之前其可能有很大改动。本文目的仅是提供信息参考。

诺基亚公司不承诺承担任何责任，包括对任何所有权的侵害责任，尽管这些所有权与实施本文给出的内容有关。诺基亚公司不保证或声称使用本文内容不会侵害上述所有权。

诺基亚保留对本文，在未经事先通知的情况下，随时进行变更的权力。

本文给出的电话 UI 图片仅为帮助说明之用，它们不是任何真正的设备。

Nokia 和 Nokia Connecting People 是诺基亚公司的注册商标。Java 以及基于 Java 的商标是 Sun Microsystems 公司的注册商标。本文中提到的其它产品和公 司名称可能是其相应公司的商标或商号。

许可声明：

允许对本文进行仅用于个人使用目的的下载和打印。在此没有许可任何其它知识产权。

1 引言

1.1 目的

J2ME™ 移动信息设备描述(MIDP)和有限联接设备配置(CLDC)分别定义在[MIDPSPEC]和[CLDCSPEC]规范之中。这些规范定义了 MIDP 应用的基础部分，而 MIDP 应用被称为 MIDlet。本文将通过 MIDlet 应用的一个简单例子， 对 MIDlet 图形编程进行简要地介绍。

本文假定读者已经熟悉 java 编程方法。同时还假定读者具有 MIDP 编程的基本知识，例如，假定读者已经阅读了诺基亚论坛的文章“ *MIDP 编程*[MIDPPROG]简介(*A Brief Introduction to MIDP Programming* [MIDPPROG])。

阅读本文之后，读者将能够更好地理解如何为 MIDP 应用创建图形用户接口。创建方法包括：

- 采用高级和低级 MIDP 用户接口 API (参见 §2.2)
- 根据不同的设备特性对用户接口进行适配 (参见 §2.2)

1.2 参考文献

MIDPSPEC 移动信息设备(Mobile Information Device) (JSR-37)， JSR 规范， Java 2 平台， Micro Edition，1.0a， 2000 年 12 月 15 日，
http://java.sun.com/products/midp/

CLDCSPEC 有限联接设备配置(Connected, Limited Device Configuration) (JSR-30)，JSR 规范， Java 2 平台， Micro Edition，版本 1.0，2000 年 5 月 19 日，
http://java.sun.com/products/cldc/

MIDPPROG MIDP 编程简介(A Brief Introduction to MIDP Programming)
诺基亚论坛(Forum Nokia)， 2001 年，
http://www.forum.nokia.com

TICTACTOE1 Scott Violet，TicTacToe 修订版，2001 年 3 月，
http://java.sun.com/products/jfc/tsc/articles/tictactoe/index.html

TICTACTOE2 TicTacToe JDK1.0 演示 applet，1995 年，
http://java.sun.com/applets/jdk/1.0/demo/TicTacToe/TicTacToe.java

2 MIDLET 图形编程

2.1 MIDlet 图形编程概述

移动信息设备描述(Mobile Information Device Profile (MIDP))[MIDPSPEC] 为运行在 MIDP 容 器中的 MIDlets 应用定义了一个 API。此 API 本身是建立在 CLDC API [CLDCSPEC]之上的。 MIDP 用户接口 API 的 Java 类设计不是基于 Java Abstract Window Toolkit (AWT) 类的， 而是为移动电话和寻呼机这类小型移动信息设备特别设计的。这类设备只有有限的屏幕尺寸和键盘性能。当程序员采用 MIDP 编写图形应用时，他们只能使用 MIDP 或 CLDC API。

MIDP 用户接口的基本抽象图形是屏幕。`Screen` 类对面向设备的图形和用户交互进行了封装。每次应用只能显示一个屏幕，而且只能浏览或使用屏幕上的条目。图 2.1 给出了一些基于屏幕的 `MIDP` 图形用户接口(`GUI`)的范例。

图 1：基于屏幕的 MIDP GUI

MIDP API 具有‘ 高级’ (‘high-level’)和‘ 低级’ (‘low-level’) UI 类。高级用户接口类 (例如： `Form`, `List`, `TextBox`, `TextField`, `Alert` 和 `Ticker`) 具有设备适配功能，它对图象、文本、文 本域以及单选按扭等进行支持。低级用户接口类(例如：类 `Canvas`)允许操作者任意绘图。在移动电话上运行的 MIDle 可以具有各种尺寸的彩色、具有灰度的或黑白的屏幕。高级类是一般 UI 元素的抽象，它便于 MIDlet 在不同设备之间的可移植性，而且能够确保适配不同设备的‘ 用户界面’ (look-and-feel)。低级 API 可以更直接地控制屏幕上显示的内容，但 MIDlet 设计者应该确保不 同设备(显示尺寸、键盘及色彩等存在差异)之间的可移植性。此范例给出了高级和低级 API 的用法。

所有 MIDP GUI 类都是 `javax.microedition.lcdui` 包的组成部分。

2.2 MIDlet 屏幕

MIDP 主要有两类屏幕：

- 高级屏幕
    - List 和 TextBox 是简单的高级屏幕类。用户不能对此类的屏幕额外增加 GUI 组件。为了在游戏开始时参与者能够对游戏进行选择，TicTacToe MIDlet 范例使用了这样的屏幕类，此屏幕类继承自被称为 ChoosePieceScreen 的 List 类。
    - 普通的 Form 屏幕类与 List 屏幕类相似，但它允许使用额外的图形组件，例如：图象、只读文本字段、可编辑文本字段、可编辑数据字段、可调指示器及选择组等。Form 组件项空闲时可以增加或删除。Formz 类在 TicTacToe 范例中没有使用。
- 低级屏幕
    - Canvas 屏幕类 (以及 Graphics 加 Image 类)能够在低级 API 基础上编写 UI。 这些类赋予了 MIDlet 编程人员最大绘画灵活性。编程人员可以画出各种图形元素：直线、弧线、矩形、圆角矩形、圆、文本文字(不同色彩、字体和大小)、以及 经剪切处理的位图等，大多数游戏 MIDlet 是按照基于 Canvas 屏幕类的主 GUI 元素进行编写的。

MIDlet 用户接口通常具有一个或多个屏幕。由于应用每次只能浏览一个屏幕，所以对 MIDlet 的结构进行周密的设计是非常必要的，这样就很容易处理屏幕之间的内容切换。

下面这一段代码描述了一种 MIDlet 进行屏幕切换的方法，这种方法基于屏幕类和相应的 MIDlet 回调函数。

```
Class MyMIDlet extends MIDlet {
    private FirstScreen firstScreen;
    private SecondScreen secondScreen;

    public MyMIDlet() { … }

    public void startApp() {
        Displayable current = Display.getDisplay(this).getCurrent();
        if (current == null)
        {
```

```
                firstScreen = new FirstScreen(this, …);
                Display.getDisplay(this).setCurrent(firstScreen);
                    // 显示应用的 UI 第一屏
            }
            else
            {
                Display.getDisplay(this).setCurrent(current);
            }

        }

        // 第一屏的回调函数被调用切换到下一屏
        public void firstScreenDone() {
            …
            secondScreen = new SecondScreen(this, …);
            display.getDisplay(this).setCurrent(secondScreen);
        }

        // 第二屏回调函数被调用应用结束
        public void secondScreenQuit() {
            …
            destroyApp(false);
            notifyDestroyed();
        }
        …
}
```

图 2：一个主 MIDlet 类范例

这个 MIDlet 有两个屏幕类 (FirstScreen 和 SecondScreen) 用于图形用户接口。当 MIDlet 开始运行时，它将当前显示的屏幕设置为 FirstScreen。如果需要从 FirstScreen 切换到 SecondScreen，则 FirstScreen 调用主 MIDlet 中的 firstScreenDone 方法 (图 2.3)。方法 firstScreenDone 创建 SecondScreen 并且将其设置为当前显示屏幕。

```
Class FirstScreen
    extends Form
    implements CommandListener
{
    private MyMIDlet midlet;

    public FirstScreen(MyMIDlet midlet)
    {
        this.midlet = midlet;
        …
    }

    public void commandAction(Command c)
    {
        if (c == cmdQuit) {
            parent.firstScreenDone();
        }
        …
    }
    …
}
```

图 3：包含 MIDlet 回调的 FirstScreen 范例

2.3 MIDP 用户接口 API

确保基于高级 API 类的 UI 对象具有可移植性和适用性是 MIDP 设备的职责。

另一方面，像 Canvas 和 Graphics 这样的类能够为编程人员提供控制 UI 视觉外观的更大自由度，而且可以监听低级键盘事件。确保具有不同特性(例如显示尺寸、彩色/黑白以及不同键盘类型等)的移动信息设备之间应用的可移植性也是编程人员的责任。例如，有必要使用 getWidth()和 getHeight()方法将 UI 外观与 一个或更多设备的 Canvas 可用尺寸进行适配。

下一章中的 TicTacToe MIDlet 范例将介绍：

- 简单易用的高级 API。
- 使用低级 API 绘图元素：例如直线、弧线、 字符串以及图形等。
- 具有不同显示尺寸的移动信息设备之间的 MIDlet 可移植性事项。
- 键盘代码与游戏动作之间的映射。

本节仅对 MIDP GUI 设计进行简单描述。如需了解更详细信息，请参阅**[MIDPSPEC]**。

3 一个范例： TICTACTOE MIDLET

3.1 设计

3.1.1 概述

下面是一个简单的 MIDlet 应用范例，此范例能够让游戏者，玩一个名字为” Tic Tac Toe”的人机游戏。此范例将介绍：

- 高级和低级用户接口组件的使用方法
- 多显示屏幕之间的切换
- 简单的命令处理
- 显示尺寸的动态适配
- 键盘事件处理

首先选择自己喜爱的图标(圆圈或十字)，然后开始游戏。MIDlet 或游戏者谁先来开局的选择是随机的。在每走一步之后，应用程序均要检查一下游戏的状态，以便确认游戏是否结束。游戏的可能结局有以下几种：游戏者获胜、MIDlet 获胜、或者平局(双方均未获胜)。在应用程序的运行过程中，能够显示双方的得分。游戏者随时可以开始新游戏或退出游戏。

下图给出了游戏屏幕出现的顺序：

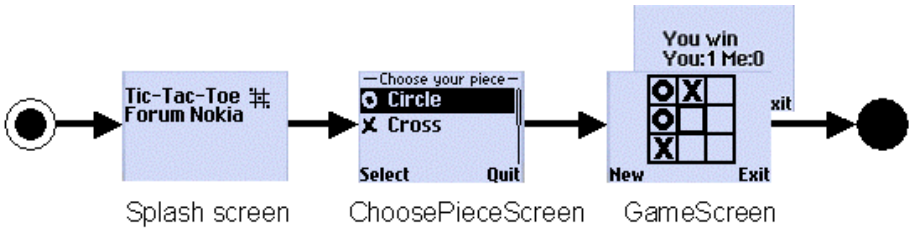

图 4：游戏屏幕顺序

图 3.1 中的屏幕图片给出了在游戏进行过程中 MIDlet 的 UI。

3.1.2 TicTacToeMIDlet

下面是 TicTacToeMIDlet 的类框图：

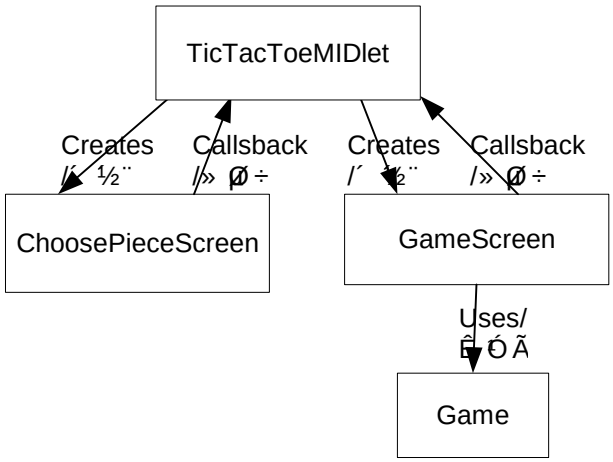

图 5： TicTacToeMIDlet 类框图

在 MIDlet 开始启动方法 startApp()时，将产生闪烁屏幕和游戏的第一个屏幕(ChoosePieceScreen)。在闪烁屏幕显示 4 秒之后，游戏的第一个屏幕开始显示。ChoosePieceScreen 允许游戏者选择自己喜爱的图标(圆圈或十字)。一旦图标选定，就可用 OK 命令确认。此时将由 ChoosePieceScreen 产生一个回调至主 MIDlet 的 choicePieceScreenDone()方法。

ChoosePieceScreen 是由高级 API List 类实现的。

图 6： ChoosePieceScreen 是一个高级 UI List 子类

choosePieceScreenDone() 回调在这一应用中为游戏创建并且显示游戏的第一个屏幕之后的屏幕，即主屏幕(GameScreen)。

当每次轮到游戏者走时，可以使用 GameScreen 的箭头键和‘ 选择’ （‘SELECT’） 按钮来选择 一个要占有的空方块。每走一步之后，应用都要对游戏状态进行检查，以查看是否有满足游戏结束的条件产生，如果有它们将显示在屏幕上。游戏者可以通过按 GameScreen 的‘ 退出’ （‘Quit’) 命令按钮来终止游戏，或通过按‘ 新’ (‘New’) 按钮重新开局。‘ 退出’ 命令将产生一个至 TicTacToeMIDlet 的 quit()方法的回调。然后，MIDlet 将调用其生命周期方法 destroyApp()来完全终止 MIDlet。

游戏逻辑封装在另一个类 Game 之中。本文仅涉及 MIDlet 图形设计，而对游戏逻辑不作进一步描述。如需要与现有的 applet Java 编程进行比较，请参阅**[TICTACTOE1] [TICTACTOE2]** 中描 述的游戏逻辑。

GameScreen 可以用低级 Canvas 和 Graphics 类来实现。它使用 Canvas、Image 和 Graphics 对象进行图形表述。

GameScreen 首先对基于 canvas 尺寸的显示模板进行初始化。这样，MIDlet 将能够以各种显示尺寸在移动信息设备上运行。本范例使用 Image 对象表示模板。然后，GameScreen 将根据由 ChoosePieceScreen 定义的游戏参与者选择，为游戏参与者和 MIDlet 分配圆圈或十字图案。接下来，游戏进行初始化(其中包括随机地确定谁先开始)，此时游戏正式开始。

为了使得 GameScreen 具有可移植性，MIDletUP 键盘代码必须和游戏动作映射，例如，这些动作有 UP、DOWN、LEFT、RIGHT 和 FIRE，它们可以在具有不同键盘的移动信息设备上使用。当键盘按下时，keyPressed()方法将判断它是方向键还是‘FIRE/SELECT’键。如果是方向键，则焦点光标移动协助玩游戏者选择一个希望占有的方块。SELECT 键可以用来选择需要占有的方块。当监测到游戏结束的满足条件时，屏幕将会显示描述获胜者的相应信息以及此时的游戏总分。

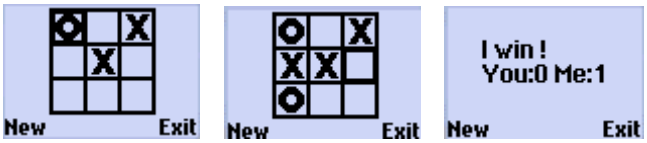

图 7：属于低级 Canvas 子类的 GameScreen

3.2 TicTacToeMIDlet.java

TicTacToeMIDlet 非常简单。它负责处理 MIDlet 生命周期事件。根据需要，它创建屏幕对象，并且处理屏幕的回调。choosePieceScreenDone 回调可以用来创建 GameScreen。而 GameScreen 能够用退出方法来结束游戏。

```
package example.tictactoe;

import java.io.IOException;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.io.*;


public class TicTacToeMIDlet
    extends MIDlet
{
    private ChoosePieceScreen choosePieceScreen;
    private GameScreen gameScreen;


    public TicTacToeMIDlet()
    {
    }


    public void startApp()
    {
        Displayable current = Display.getDisplay(this).getCurrent();
        if (current == null)
        {
            // first time we've been called/

            // Get the logo image/
            Image logo = null;
            try
            {
                logo = Image.createImage("/tictactoe.png");
            }
            catch (IOException e)
            {
                // just use null image/
            }


            Alert splashScreen = new Alert(null, "Tic-Tac-Toe\nForum
Nokia",
                logo, AlertType.INFO);
            splashScreen.setTimeout(4000); // 4 seconds
            choosePieceScreen = new ChoosePieceScreen(this);
```

```
                Display.getDisplay(this).setCurrent(splashScreen,
                    choosePieceScreen);
            }
            else
            {
                Display.getDisplay(this).setCurrent(current);
            }
        }


        public void pauseApp()
        {
        }


        public void destroyApp(boolean unconditional)
        {
        }


        public void quit()
        {
            destroyApp(false);
            notifyDestroyed();
        }


        public void choosePieceScreenDone(boolean isPlayerCircle)
        {
            gameScreen = new GameScreen(this, isPlayerCircle);
            Display.getDisplay(this).setCurrent(gameScreen);
        }
    }
```

3.3 ChoosePieceScreen.java

ChoosePieceScreen 是基于表单的高级 API 屏幕，它允许游戏者对圆圈和十字图标进行选择。在玩游戏者按下‘OK’ 按钮之后，ChoosePieceScreen 使用 MIDlet 回调方法 choosePieceScreenDone 来表示玩游戏者的选择。

```
package example.tictactoe;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.io.*;


public class ChoosePieceScreen
    extends List
    implements CommandListener
{
    private static final String CIRCLE_TEXT = "Circle";
    private static final String CROSS_TEXT = "Cross";

    private final TicTacToeMIDlet midlet;
    private final Command quitCommand;

    public ChoosePieceScreen(TicTacToeMIDlet midlet)
    {
        super("Choose your piece", List.IMPLICIT);

        this.midlet = midlet;

        append(CIRCLE_TEXT, loadImage("/circle.png"));
        append(CROSS_TEXT,  loadImage("/cross.png"));

        quitCommand = new Command("Quit", Command.EXIT, 2);
        addCommand(quitCommand);
        setCommandListener(this);
    }


    public void commandAction(Command c, Displayable d)
    {
        boolean isPlayerCircle =
            getString(getSelectedIndex()).equals(CIRCLE_TEXT);

        if (c == List.SELECT_COMMAND)
        {
            midlet.choosePieceScreenDone(isPlayerCircle);
        }
        else // quitCommand/
        {
```

```
                midlet.quit();
            }
        }


        private Image loadImage(String imageFile)
        {
            Image image = null;
            try
            {
                image = Image.createImage(imageFile);
            }
            catch (Exception e)
            {
                // Use a 'null' image in the choice list (i.e. text only
    choices)./
            }
            return image;
        }
    }
```

3.4 GameScreen.java

GameScreen 使用低级 API Canvas 屏幕、以及 Image 和 Graphics 类来表示游戏的模板、图标以及 最终状态信息。如需更详细信息，请参阅绘图函数，drawCircle, drawCross, drawPiece, drawPlayerCursor 和 drawBoard 函数。 GameScreen 屏幕使用 MIDlet 回调方法 quit 来结束游 戏。

此屏幕能够根据显示特性(高、宽、色彩等)进行适配。另外，请注意可以用 4 向或 2 向导航键控制光标的移动方向。

此屏幕使用 Game 类，主游戏逻辑封装在其中。

```
package example.tictactoe;

import java.util.Random;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;


class GameScreen
    extends Canvas
    implements CommandListener
{
    private static final int BLACK = 0x00000000;
    private static final int WHITE = 0x00FFFFFF;
    private static final int RED   = 0x00FF0000;
    private static final int BLUE  = 0x000000FF;
    private static final int NO_MOVE  = -1;

    private final TicTacToeMIDlet midlet;
    private final Game game;
    private final Command exitCommand;
    private final Command newGameCommand;
    private final Random random = new Random();

    private int screenWidth, screenHeight;
    private int boardCellSize, boardSize, boardTop, boardLeft;

    private boolean playerIsCircle;
    private boolean computerIsCircle;
    private int preCursorPosition, cursorPosition;
    private int computerMove = NO_MOVE;
    private int playerMove = NO_MOVE;
    private int computerGamesWonTally = 0;
    private int playerGamesWonTally = 0;
    private boolean isRestart;
```

```
public GameScreen(TicTacToeMIDlet midlet, boolean playerIsCircle)
{
     this.midlet = midlet;
     this.playerIsCircle = playerIsCircle;
     computerIsCircle = !playerIsCircle;

     game = new Game(random);

     initializeBoard();

     // configure Screen commands
     exitCommand = new Command("Exit", Command.EXIT, 1);
     newGameCommand = new Command("New", Command.SCREEN, 2);
     addCommand(exitCommand);
     addCommand(newGameCommand);
     setCommandListener(this);

     // begin the game play
     initialize();
}


// Initialize the Game and Game screen. Also used for game restarts.
private void initialize()
{
    game.initialize();

    preCursorPosition = cursorPosition = 0;
    playerMove = NO_MOVE;

    boolean computerFirst = ((random.nextInt() & 1) == 0);
    if (computerFirst)
    {
        computerMove = game.makeComputerMove();
    }
    else
    {
        computerMove = NO_MOVE;
    }

    isRestart = true;
    repaint();
}


public void paint(Graphics g)
{
    if (game.isGameOver())
    {
        paintGameOver(g);
    }
```

```
        else
        {
            paintGame(g);
        }
    }


    private void paintGame(Graphics g)
    {
        if (isRestart)
        {
            // clean the canvas
            g.setColor(WHITE);
            g.fillRect(0, 0, screenWidth, screenHeight);

            drawBoard(g);
            isRestart = false;
        }

        drawCursor(g);

        if (playerMove != NO_MOVE)
        {
            drawPiece(g, playerIsCircle, playerMove);
        }

        if (computerMove != NO_MOVE)
        {
            drawPiece(g, computerIsCircle, computerMove);
        }
    }


    private void paintGameOver(Graphics g)
    {
        String statusMsg = null;
        if(game.isComputerWinner())
        {
            statusMsg = "I win !";
            computerGamesWonTally++;
        }
        else if (game.isPlayerWinner())
        {
            statusMsg = "You win";
            playerGamesWonTally++;
        }
        else
        {
            statusMsg = "Stalemate";
        }
        String tallyMsg = "You:" + playerGamesWonTally +
                          " Me:" + computerGamesWonTally;
```

```
        Font font = Font.getFont(Font.FACE_SYSTEM, Font.STYLE_PLAIN,
                                 Font.SIZE_MEDIUM);
        int strHeight = font.getHeight();
        int statusMsgWidth = font.stringWidth(statusMsg);
        int tallyMsgWidth = font.stringWidth(tallyMsg);
        int strWidth = tallyMsgWidth;
        if (statusMsgWidth > tallyMsgWidth)
        {
            strWidth = statusMsgWidth;
        }

        // Get the {x, y} position for painting the strings.
        int x = (screenWidth - strWidth) / 2;
        x = x < 0 ? 0 : x;

        int y = (screenHeight - 2 * strHeight) / 2;
        y = y < 0 ? 0 : y;

        // clean the canvas
        g.setColor(WHITE);
        g.fillRect(0, 0, screenWidth, screenHeight);

        // paint the strings' text
        g.setColor(BLACK);
        g.drawString(statusMsg, x, y, (Graphics.TOP | Graphics.LEFT));
        g.drawString(tallyMsg,  x, (y + 1 + strHeight),
                     (Graphics.TOP | Graphics.LEFT));
    }


    public void commandAction(Command c, Displayable d)
    {
        if (c == exitCommand)
        {
            midlet.quit();
        }
        else if (c == newGameCommand)
        {
            initialize();
        }
    }


    private void initializeBoard()
    {
        screenWidth = getWidth();
        screenHeight = getHeight();

        if (screenWidth > screenHeight)
        {
            boardCellSize = (screenHeight - 2) / 3;
```

```
            boardLeft = (screenWidth - (boardCellSize * 3)) / 2;
            boardTop = 1;
        }
        else
        {
            boardCellSize = (screenWidth - 2) / 3;
            boardLeft = 1;
            boardTop = (screenHeight - boardCellSize * 3) / 2;
        }
    }


    protected void keyPressed(int keyCode)
    {
        // can't continue playing until the player restarts
        if (game.isGameOver())
        {
            return;
        }

        int gameAction = getGameAction(keyCode);

        switch (gameAction)
        {
        case FIRE:
            doPlayerMove();
            break;

        case RIGHT:
            doMoveCursor(1, 0);
            break;

        case DOWN:
            doMoveCursor(0, 1);
            break;

        case LEFT:
            doMoveCursor(-1, 0);
            break;

        case UP:
            doMoveCursor(0, -1);
            break;

        default:
            break;
        }
    }


    private void doPlayerMove()
    {
```

```
    if (game.isFree(cursorPosition))
    {
        // player move
        game.makePlayerMove(cursorPosition);
        playerMove = cursorPosition;

        // computer move
        if (!game.isGameOver())
        {
            computerMove = game.makeComputerMove();
        }

        repaint();
    }
}


private void doMoveCursor(int dx, int dy)
{
    int newCursorPosition = cursorPosition + dx + 3 * dy;

    if ((newCursorPosition >= 0) && (newCursorPosition < 9))
    {
        preCursorPosition = cursorPosition;
        cursorPosition = newCursorPosition;
        repaint();
    }
}


// Draw a CIRCLE or CROSS piece on the board
private void drawPiece(Graphics g, boolean isCircle, int pos)
{
    int x = ((pos % 3) * boardCellSize) + 3;
    int y = ((pos / 3) * boardCellSize) + 3;

    if (isCircle)
    {
        drawCircle(g, x, y);
    }
    else
    {
        drawCross(g, x, y);
    }
}


// Draw blue CIRCLE onto the board image
private void drawCircle(Graphics g, int x, int y)
{
    g.setColor(BLUE);
    g.fillArc(x + boardLeft, y + boardTop,
```

```
                    boardCellSize - 4, boardCellSize - 4, 0, 360);

            g.setColor(WHITE);
            g.fillArc(x + 4 + boardLeft, y + 4 + boardTop,
                    boardCellSize - 4 - 8, boardCellSize - 4 - 8, 0, 360);
        }


        // Draw red CROSS onto the board image
        private void drawCross(Graphics g, int x, int y)
        {
            g.setColor(RED);
            for (int i = 0; i < 4; i++)
            {
                g.drawLine(x + 1 + i + boardLeft, y + boardTop,
                           x + boardCellSize - 4 - 4 + i + boardLeft,
                           y + boardCellSize - 5 + boardTop);
                g.drawLine(x + 1 + i + boardLeft, y + boardCellSize - 5 +
boardTop,
                           x + boardCellSize - 4 - 4 + i + boardLeft,
                           y + boardTop);
            }
        }


        // Visually indicates a Player selected square on the board image
        private void drawCursor(Graphics g)
        {
            // draw cursor at selected Player square.
            g.setColor(WHITE);
            g.drawRect(((preCursorPosition % 3) * boardCellSize) + 2 +
boardLeft,
                       ((preCursorPosition/3) * boardCellSize) + 2 + boardTop,
                       boardCellSize - 3,
                       boardCellSize - 3);

            // draw cursor at selected Player square.
            g.setColor(BLACK);
            g.drawRect(((cursorPosition % 3) * boardCellSize) + 2 + boardLeft,
                       ((cursorPosition/3) * boardCellSize) + 2 + boardTop,
                       boardCellSize - 3,
                       boardCellSize - 3);
        }


        private void drawBoard(Graphics g)
        {
            // clean the board
            g.setColor(WHITE);
            g.fillRect(0, 0, screenWidth, screenHeight);

            // draw the board
            g.setColor(BLACK);
```

```
            for (int i = 0; i < 4; i++)
            {
                g.fillRect(boardLeft, boardCellSize * i + boardTop,
                           (boardCellSize * 3) + 2, 2);
                g.fillRect(boardCellSize * i + boardLeft, boardTop,
                           2, boardCellSize * 3);
            }
        }
    }
```

3.5 Game.java

此类为游戏‘Tic Tac Toe’封装了主游戏逻辑。在这个范例中，游戏逻辑本身并无意义，因为它 主要是为了表现 MIDP 图形编程的基本概念。游戏逻辑的 WINS 数组部分是通过对 **[TICTACTOE2]**中的一个著名范例进行修改之后获得的。

注意游戏逻辑与游戏 UI 表示(参见类 GameScreen)是独立的，而且可以用其它可替代方式替换它。

```
package example.tictactoe;

import java.util.Random;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;


// The game logic for TicTacToe
class Game
{
    private static final int[] WINS =
    {
        // horizontals
        bit(0) | bit(1) | bit(2),
        bit(3) | bit(4) | bit(5),
        bit(6) | bit(7) | bit(8),
        // verticals
        bit(0) | bit(3) | bit(6),
        bit(1) | bit(4) | bit(7),
        bit(2) | bit(5) | bit(8),
        // diagonals
        bit(0) | bit(4) | bit(8),
        bit(2) | bit(4) | bit(6)
    };
    private static final int DRAWN_GAME =
        bit(0) | bit(1) | bit(2) |
        bit(3) | bit(4) | bit(5) |
        bit(6) | bit(7) | bit(8);

    private int playerState;
    private int computerState;
    private Random random;


    Game(Random random)
    {
        this.random = random;
        initialize();
    }
```

```
    void initialize()
    {
        playerState = 0;
        computerState = 0;
    }


    boolean isFree(int position)
    {
        int bit = bit(position);
        return (((playerState & bit) == 0) && ((computerState & bit) ==
0));
    }


    // The 'Contract' is that caller will always make valid moves.
    // We don't check that it's the player's turn.
    void makePlayerMove(int position)
    {
        playerState |= bit(position);
    }


    // The 'Contract' is that we will be called only when there is still
    // at least one free square.
    int makeComputerMove()
    {
        int move = getWinningComputerMove();
        if (move == -1)
        {
            // can't win
            move = getRequiredBlockingComputerMove();
            if (move == -1)
            {
                // don't need to block
                move = getRandomComputerMove();
            }
        }

        computerState |= bit(move);

        return move;
    }


    boolean isGameOver()
    {
        return isPlayerWinner() | isComputerWinner() | isGameDrawn();
    }
```

```
boolean isPlayerWinner()
{
    return isWin(playerState);
}


boolean isComputerWinner()
{
    return isWin(computerState);
}


boolean isGameDrawn()
{
    return (playerState | computerState) == DRAWN_GAME;
}


// Return a winning move if there is at least one, otherwise return -1
private int getWinningComputerMove()
{
    int move = -1;
    for (int i = 0; i < 9; ++i)
    {
        if (isFree(i) && isWin(computerState | bit(i)))
        {
            move = i;
            break;
        }
    }
    return move;
}


// Return a required blocking move if there is at least one (more
// than one and we've inevitably lost), otherwise return -1
private int getRequiredBlockingComputerMove()
{
    int move = -1;
    for (int i = 0; i < 9; ++i)
    {
        if (isFree(i) && isWin(playerState | bit(i)))
        {
            move = i;
            break;
        }
    }
    return move;
}
```

```
    // Return a random move in a free square,
    // or return -1 if none are available
    private int getRandomComputerMove()
    {
        int move = -1;

        // determine how many possible moves there are
        int numFreeSquares = 0;
        for (int i = 0; i < 9; ++i)
        {
            if (isFree(i))
            {
                numFreeSquares++;
            }
        }

        // if there is at least one possible move, pick randomly
        if (numFreeSquares > 0)
        {
            // shift twice to get rid of sign bit, then modulo
numFreeSquares
            int pick = ((random.nextInt()<<1)>>>1) % numFreeSquares;

            // now find the chosen free square by counting pick down to
zero
            for (int i = 0; i < 9; ++i)
            {
                if (isFree(i))
                {
                    if (pick == 0)
                    {
                        move = i;
                        break;
                    }
                    pick--;
                }
            }
        }

        return move;
    }


    private static boolean isWin(int state)
    {
        boolean isWinner = false;
        for (int i = 0; i < WINS.length; ++i)
        {
            if ((state & WINS[i]) == WINS[i])
            {
                isWinner = true;
                break;
            }
```

```
            }
            return isWinner;
        }


        private static int bit(int i)
        {
            return 1 << i;
        }
    }
```

3.6 TicTacToe.jad

The following is the application descriptor for the TicTacToeMIDlet.

下面是 TicTacToeMIDlet 的应用描述文件。

```
MIDlet-Name: TicTacToe
MIDlet-Vendor: Forum Nokia
MIDlet-Version: 1.1.1
MIDlet-Jar-Size: 11409
MIDlet-Jar-URL: TicTacToe.jar
MIDlet-1: TicTacToe, /tictactoe.png, example.tictactoe.TicTacToeMIDlet
```