MIDP 2.0: Introduction to the Game API

Version 1.0, March 8, 2004
Contents

1. Introduction ................................................................................................................................................ 5

2. MIDP 2.0 Game API ..................................................................................................................................... 6
   2.1. GameCanvas ...............................................................................................................................................6
   2.2. Layer .............................................................................................................................................................7
   2.3. Sprite ............................................................................................................................................................7
   2.4. TiledLayer ....................................................................................................................................................8
   2.5. LayerManager ............................................................................................................................................9

3. MIDP 2.0 Media API ..................................................................................................................................10

4. Improvements to MIDP Low-Level Graphics API ......................................................................................11
   4.1. Transparency ..........................................................................................................................................11
   4.2. Full-Screen Mode ....................................................................................................................................11
   4.3. Vibration ...................................................................................................................................................12
   4.4. Flashing Backlight ................................................................................................................................12

5. References .................................................................................................................................................13

Appendix A. Sheepdog Game Source Code ......................................................................................................14
   A.1 SheepDogMIDlet.java ...........................................................................................................................14
   A.2 SplashScreen.java ..................................................................................................................................19
   A.3 MenuList.java ..........................................................................................................................................21
   A.4 InstructionsScreen.java ........................................................................................................................22
   A.5 HighScoreScreen.java ...........................................................................................................................23
   A.6 SheepdogCanvas.java .............................................................................................................................24
   A.7 Field.java ..................................................................................................................................................29
   A.8 Sheepdog.java ..........................................................................................................................................31
   A.9 Sheep.java ................................................................................................................................................34
   A.10 GameOverScreen.java ..........................................................................................................................40
   A.11 SoundEffects.java ..................................................................................................................................41
## Change History

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>March 8, 2004</td>
<td>Version 1.0</td>
<td>Initial document release</td>
</tr>
</tbody>
</table>

Copyright © 2004 Nokia Corporation. All rights reserved.

Nokia and Nokia Connecting People are registered trademarks of Nokia Corporation. Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. Other product and company names mentioned herein may be trademarks or trade names of their respective owners.

Disclaimer

The information in this document is provided “as is,” with no warranties whatsoever, including any warranty of merchantability, fitness for any particular purpose, or any warranty otherwise arising out of any proposal, specification, or sample. Furthermore, information provided in this document is preliminary, and may be changed substantially prior to final release. This document is provided for informational purposes only.

Nokia Corporation disclaims all liability, including liability for infringement of any proprietary rights, relating to implementation of information presented in this document. Nokia Corporation does not warrant or represent that such use will not infringe such rights.

Nokia Corporation retains the right to make changes to this specification at any time, without notice.

License

A license is hereby granted to download and print a copy of this specification for personal use only. No other license to any other intellectual property rights is granted herein.
1. Introduction

One of the new features of MIDP 2.0 [MIDP 2.0] is a new Game API that simplifies the writing of MIDP games. This tutorial introduces the following MIDP 2.0 Game API features:

- Class GameCanvas, a subclass of Canvas with game-related features
- Class Sprite, for animated sprites
- Class TiledLayer, for tiled game backgrounds, including animated tiles
- Class LayerManager, for managing Sprites and TiledLayers

The following new MIDP 2.0 features will also be covered, as they are useful for games:

- MIDP 2.0 Media API, for playing sound effects
- New features in MIDP low-level UI, including transparent images, full-screen mode, vibration, and backlight flashing

The features are illustrated with examples from Sheepdog, a simple MIDP 2.0 game whose source code is included in Appendix A, “Sheepdog Game Source Code.” Figure 1 illustrates some screen shots of the game:

![Sheepdog Screen Shots](image)

The game is intended to serve as a tutorial example, rather than a marketable game. We expect that a game development company would produce much better graphics and sound effects, and a more playable game.

This document assumes familiarity with Java™ programming and the basics of MIDP programming as covered in the Forum Nokia documents MIDP 1.0: Introduction to MIDlet Programming [MIDPPROG] and A Brief Introduction to MIDP Graphics [MIDPGRAP].
2. MIDP 2.0 Game API

The MIDP 2.0 Game API has its own package: `javax.microedition.lcdui.game`.

2.1. GameCanvas

Class `GameCanvas` is a subclass of the familiar `Canvas` class already present in MIDP 1.0. GameCanvas adds the following features:

- Off-screen graphics buffer
- Reading key states (instead of receiving key events)

`GameCanvas` replaces the `repaint` and `paint` mechanism of `Canvas` with a new `getGraphics` and `flushGraphics` mechanism. The `getGraphics` method returns a `Graphics` object for drawing into the off-screen graphics buffer, and the `flushGraphics` method copies the contents of the off-screen graphics buffer to the display. Therefore the typical animation thread `run` method seen in MIDP 1.0 Canvas subclasses:

```java
public void run()
{
    while (playing) {
        advanceAnimation();
        repaint();
        serviceRepaints();
    }
}

public void paint(Graphics g) {
    drawStuff(g);
}
```

is replaced by a slightly simpler one:

```java
public void run() {
    while (playing) {
        advanceAnimation(getKeyStates());
        drawStuff(getGraphics());
        flushGraphics();
    }
}
```

The `getGraphics` call returns a new object each time, so in fact it is better to call it once in the constructor of your `GameCanvas` subclass and keep the result in an instance variable. This is the approach used by the Sheepdog game’s `SheepdogCanvas` class (see Section A.6, “SheepdogCanvas.java”).

The method `getKeyStates` returns an integer with bits set for each “game” key that is currently pressed. In order that a very short key press doesn’t go unnoticed, `getKeyStates` will have its flag set in the next call, even if the key has already been released by then. This method can be used as shown below:

```java
int keyStates = getKeyStates();
if ((keyStates & LEFT_PRESSED) != 0)
    sprite.move(-1, 0); // left
if ((keyStates & RIGHT_PRESSED) != 0)
    sprite.move(1, 0); // right
```
Note that in the above code, if both left and right are pressed, the sprite will be moved both left and right. Not all phones support multiple key presses, but those that do should be taken into account. The Sheepdog game handles this neatly in method `tick` of class `SheepdogCanvas` (see Section A.6, “SheepdogCanvas.java”).

A boolean flag `suppressKeyEvents` in the `GameCanvas` constructor defines whether key events like `keyPressed` will be sent for the keys whose states are reported in `getKeyStates`. Other keys (e.g., number keys not mapped to game actions) will still send key events like `keyPressed` in either case.

2.2. Layer

Layer is an abstract class representing a drawable object on the display. Layers can be managed using a `LayerManager`. `Sprite` and `TiledLayer` are both subclasses of `Layer`, and you’ll probably want to use them rather than subclassing `Layer` directly.

2.3. Sprite

A `Sprite` is an animated object that you can move around the screen. In the Sheepdog game, the sheep and sheepdog are Sprites.

![Sheep and sheepdog sprites](image)

Figure 2: Sheep and sheepdog sprites

Each Sprite is created from an `Image` that contains all of its animation frames. Figure 3 shows the animation frame image for the sheepdog. The image background should be transparent, so that when it is drawn only the sprite itself is drawn.

![Sheepdog animation frame image, 225 x 15 pixels](image)

Figure 3: Sheepdog animation frame image, 225 x 15 pixels

You’ll notice that there are no animation frames of the sheepdog running to the right. This is because you can use method `setTransform` to specify a transform for the sprite to be drawn with: when the sheepdog is running right, use the left-facing animation frames and a horizontal mirror transform (`Sprite.TRANS_MIRROR`).

Transforms are performed about a reference pixel. By default this is the top left pixel, and if you didn’t change this your sheepdog would be reflected 14 pixels to the left of where you expected. Instead, call `Sprite` method `defineReferencePixel` and set the reference pixel to be the center of the sprite. (Because even-width sprites don’t have a single center pixel, it helps to use odd-width sprites.)

Class `Sprite` also has useful methods for detecting whether sprites overlap (“collide with”) other sprites. In the Sheepdog game, you use `Sprite` method `defineCollisionRectangle` to make the collision detection rectangle slightly smaller than the sprite – this is because if only the edges of two sprites overlap, it usually appears to the player as if they don’t overlap yet. It’s possible to make the collision detection “pixel level,” which means that two sprites are judged to collide only if their opaque (nontransparent) pixels overlap, but this might be slower than just comparing collision rectangles.
A TiledLayer is an object on the screen made up of a rectangular grid of tiles. TiledLayers are used for game backgrounds or much like scenery backdrops in a theater. It is possible to have several overlapping TiledLayers, and to scroll them by just changing their position with the `setPosition` method.

A TiledLayer has a set of tiles, which is created from an Image.

Figure 4: Field tile image, 160 × 16 pixels

Figure 4 shows the tile image used by the Sheepdog game. There are ten 16 × 16-pixel static tiles made from this image. The TiledLayer has a 12 × 12 grid of cells, shown in Table 1. The "land" cells hold values from 1 to 5 for open field, 9 for the brown fold walls, and 10 for the inside of the fold. The "sea" cells hold values from –1 to –3. There are no cells holding value 0 (transparent).

<p>| | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
</tbody>
</table>

Table 1: Grid of cells for field

The resulting TiledLayer is shown in Figure 5.

Figure 5: TiledLayer field

The cell values –1 to –3 are animated tiles, created using TiledLayer method `createAnimatedTile`. Each points to one of the blue sea static tiles numbered 6 to 8. By
continuously changing which static tile each points to, you can make the sea move without changing the grid cell values themselves.

2.5. LayerManager

Class LayerManager allows you to manage all of your Layers (Sprites and TiledLayers) easily. Simply add them all to your LayerManager, then instead of painting them all individually just tell the LayerManager to paint. The LayerManager keeps track of their "Z-order" (by default, layers added later are nearer the back).
3. MIDP 2.0 Media API

MIDP 2.0 includes a subset of the Mobile Media API [MMAPI] in its own packages:
javax.microedition.media and javax.microedition.media.control. MIDP 2.0 phones
may choose to implement just this subset or the full Mobile Media API. In either case, the phones may
support only one sound at a time or support playing several simultaneously.

The feature most useful for games is the ability to play short sound files as sound effects. The MIDP 2.0
Media API specifies no mandatory sound file formats, but it is reasonable to expect at least WAV (8 kHz
mono PCM) and MIDI file formats to be supported.

The code to play a sound sample from a resource file in the MIDlet's JAR file appears below:

```java
try {
    InputStream is = getClass().getResourceAsStream("/dog.wav");
    Player p = Manager.createPlayer(is, "audio/x-wav");
    p.prefetch();
    p.start();
} catch (IOException ex) {
} catch (MediaException ex) {
}
```

The same player can be reused by calling `start` again. This has no effect if the player is already
playing, so to make sure it starts from the beginning you can use code like this:

```java
p.stop();
p.setMediaTime(0L);
p.start();
```

This approach is used in the Sheepdog game's SoundEffects class (see Section A.11,
"SoundEffects.java").
4. Improvements to MIDP Low-Level Graphics API

4.1. Transparency

Whereas in MIDP 1.0 support for transparent PNG images was optional, in MIDP 2.0 it is now mandatory (however support for partial transparency is still optional – phones may treat partially transparent pixels as fully transparent). Transparent Images are typically used in Sprites, but it is also useful to have partially transparent TiledLayers if you want to create a pseudo 3-D effect by having multiple background layers scrolling at different rates.

This affects the following existing methods of class Image:

- `createImage(String)`
- `createImage(byte[], int, int)`

and the following methods that are new in MIDP 2.0:

- `createImage(InputStream)`
- `createImage(Image)`
- `createImage(Image, int, int, int, int, int)`

The last two create new Images from all or part of an existing Image: if that existing Image has transparency information, it will be copied into the new Image.

It is also now possible to create a transparent image from an array of pixel data:

- `createRGBImage(int[], int, int, boolean)`

Each int in the array represents one pixel. If the boolean parameter is `true`, each pixel value is interpreted as ARGB, where the ”A” is ”alpha” (opacity): if alpha is 0xFF, the pixel is fully opaque, and if alpha is 0x00, the pixel is fully transparent. So, for example, 0xFF00FFFF is fully opaque cyan, 0x8800FF00 is semi-opaque green, and 0x00000000 is fully transparent black (if a pixel is fully transparent, it doesn’t really matter what color it is).

4.2. Full-Screen Mode

Class Canvas has a new method `setFullScreenMode(boolean)`, which allows you to set it to full-screen mode. Since GameCanvas is a subclass of Canvas, this method works for GameCanvas, too. In full-screen mode, a Canvas displays no title, Ticker, or Command labels.

A Canvas in full-screen mode may still have Commands, accessible by some other means than Command labels on the Canvas. If it has no Commands, then the keys that would have been used for Commands (e.g., soft keys) may deliver key events (in Nokia’s MIDP 2.0 implementations, they do). These keys probably don’t have obvious Unicode character representations, so their key codes will probably be negative (in Nokia’s MIDP 2.0 implementations, they are). This allows a useful rule of thumb for GameCanvas:

- If GameCanvas is suppressing key events for game keys, and keyPressed is called with a negative key code, a special nongame key such as a soft key has been pressed.
The Sheepdog game’s class SheepdogCanvas uses this so that it can return users to the menu screen if they press a soft key during the game.

4.3. Vibration

A method vibrate(int) has been added to class Display, allowing the game to activate the phone’s vibration function if it has one. The parameter is the duration in milliseconds of the vibration – there is no way to control the frequency of the vibration. The Sheepdog game uses this method (in SheepdogMIDlet, called from SheepdogCanvas, called from Sheepdog) to vibrate when the sheepdog hits an obstacle.

4.4. Flashing Backlight

A method flashBacklight(int) has been added to class Display, allowing the game to flash the phone’s backlight if the phone supports this. The parameter is the duration in milliseconds of the flashing – there is no way to control the frequency of the flashing. The Sheepdog game uses this method (in SheepdogMIDlet, called by GameOverScreen) to flash the backlight when the game-over screen is first displayed.
5. References


Appendix A. Sheepdog Game Source Code

A.1 SheepDogMIDlet.java

As with previous Forum Nokia MIDlet examples, the MIDlet is used as a state machine that manages the various screens and transitions between them.

This class cooperates with class SplashScreen (through methods splashScreenPainted and splashScreenDone) to perform background initialization in method init while the splash screen is being displayed.

The methods readRecordStore and writeRecordStore are used to keep the high score in a record store named “BESTTIME”.

```java
// unnamed package
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.rms.*;
import java.util.*;
import java.io.*;

public class SheepdogMIDlet
extends MIDlet
implements Runnable
{
    private static final String RS_NAME = "BESTTIME";
    private MenuList menuList;
    private SheepdogCanvas sheepdogCanvas;
    private boolean initDone = false;
    private static final Random random = new Random();
    private boolean hasBestTime = false;
    private long bestTime;

    public SheepdogMIDlet()
    {
    }

    public void startApp()
    {
        Displayable current = Display.getDisplay(this).getCurrent();
        if (current == null)
        {
            // first time we've been called
            Display.getDisplay(this).setCurrent(new SplashScreen(this));
        }
        else
        {
            if (current == sheepdogCanvas)
            {
                sheepdogCanvas.start(); // start its animation thread
            }
            Display.getDisplay(this).setCurrent(current);
        }
    }

    public void pauseApp()
    {
        Displayable current = Display.getDisplay(this).getCurrent();
        if (current == sheepdogCanvas)
        {
            sheepdogCanvas.pause(); // stop its animation thread
        }
        Display.getDisplay(this).setCurrent(current);
    }
    }"
public void destroyApp(boolean unconditional)
{
    if (sheepdogCanvas != null)
    {
        sheepdogCanvas.stop();  // kill its animation thread
    }
}

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

public void run()
{
    init();
}

private synchronized void init()
{
    if (!initDone)
    {
        readRecordStore();
        SoundEffects.getInstance();
        menuList = new MenuList(this);
        sheepdogCanvas = new SheepdogCanvas(this);
        initDone = true;
    }
}

void splashScreenPainted()
{
    new Thread(this).start();  // start background initialization
}

void splashScreenDone()
{
    init();  // if not already done
    Display.getDisplay(this).setCurrent(menuList);
}

void menuListContinue()
{
    Display.getDisplay(this).setCurrent(sheepdogCanvas);
    sheepdogCanvas.start();
}

void menuListNewGame()
{
    sheepdogCanvas.init();
    Display.getDisplay(this).setCurrent(sheepdogCanvas);
    sheepdogCanvas.start();
}
void menuListInstructions() {
    // create and discard a new Instructions screen each time, to
    // avoid keeping heap memory for it when it’s not in use
    Display.getDisplay(this).setCurrent(new InstructionsScreen(this));
}

void menuListHighScore() {
    // create and discard a new High Score screen each time, to
    // avoid keeping heap memory for it when it’s not in use
    Display.getDisplay(this).setCurrent(new HighScoreScreen(this));
}

void menuListQuit() {
    quit();
}

void sheepdogCanvasMenu() {
    sheepdogCanvas.stop();
    menuList.setGameActive(true);
    Display.getDisplay(this).setCurrent(menuList);
}

void sheepdogCanvasGameOver(long time) {
    sheepdogCanvas.stop();
    menuList.setGameActive(false);
    Display.getDisplay(this).setCurrent(new GameOverScreen(this, time));
}

void gameOverDone() {
    Display.getDisplay(this).setCurrent(menuList);
}

void instructionsBack() {
    Display.getDisplay(this).setCurrent(menuList);
}

void highScoreBack() {
    Display.getDisplay(this).setCurrent(menuList);
}

// method needed by lots of classes, shared by putting it here
static Image createImage(String filename) {
    Image image = null;
    try {
        image = Image.createImage(filename);
    } catch (java.io.IOException ex) {
        // just let return value be null
    }
}
// method needed by lots of classes, shared by putting it here
static int random(int size)
{
    return (random.nextInt() & 0x7FFFFFFF) % size;
}

// only the MIDlet has access to the display, so put this method here
void flashBacklight(int millis)
{
    Display.getDisplay(this).flashBacklight(millis);
}

// only the MIDlet has access to the display, so put this method here
void vibrate(int millis)
{
    Display.getDisplay(this).vibrate(millis);
}

long getBestTime()
{
    return hasBestTime ? bestTime : -1;
}

boolean checkBestTime(long time)
{
    if (!hasBestTime || (time < bestTime))
    {
        hasBestTime = true;
        bestTime = time;
        writeRecordStore();
        return true;
    }
    else
    {
        return false;
    }
}

private void readRecordStore()
{
    hasBestTime = false;
    RecordStore rs = null;
    ByteArrayInputStream bais = null;
    DataInputStream dis = null;
    try
    {
        rs = RecordStore.openRecordStore(RS_NAME, false);
        byte[] data = rs.getRecord(1);
        bais = new ByteArrayInputStream(data);
        dis = new DataInputStream(bais);
        bestTime = dis.readLong();
        hasBestTime = true;
    }
    catch (IOException ex)
    {
        // hasBestTime will still be false
    }
    catch (RecordStoreException ex)
// hasBestTime will still be false
}
finally
{
    if (dis != null)
    {
        try
        {
            dis.close();
        }
        catch (IOException ex)
        {
            // no error handling necessary here
        }
    }
    if (bais != null)
    {
        try
        {
            bais.close();
        }
        catch (IOException ex)
        {
            // no error handling necessary here
        }
    }
    if (rs != null)
    {
        try
        {
            rs.closeRecordStore();
        }
        catch (RecordStoreException ex)
        {
            // no error handling necessary here
        }
    }
}

// this will only be called when we have a best time
private void writeRecordStore()
{
    RecordStore rs = null;
    ByteArrayOutputStream baos = null;
    DataOutputStream dos = null;
    try
    {
        rs = RecordStore.openRecordStore(RS_NAME, true);
        baos = new ByteArrayOutputStream();
        dos = new DataOutputStream(baos);
        dos.writeLong(bestTime);
        byte[] data = baos.toByteArray();
        if (rs.getNumRecords() == 0)
        {
            // new record store
            rs.addRecord(data, 0, data.length);
        }
        else
        {
            // existing record store: will have one record, id 1
            rs.setRecord(1, data, 0, data.length);
        }
    }
    catch (IOException ex)
    {
A.2 SplashScreen.java

The splash screen displays an image in the center of the screen and a red line around the edge (so that it looks good in all screen sizes). After it has drawn the screen the first time, it frees the image for garbage collection (by setting the reference to it to null) and calls back to the MIDlet. After three seconds or the first key press, it again calls back to the MIDlet. In this way, the MIDlet can do initialization while the splash screen is displayed.
// unnamed package

import javax.microedition.lcdui.*;

class SplashScreen
    extends Canvas
    implements Runnable
{
    private final SheepdogMIDlet midlet;
    private Image splashImage;
    private volatile boolean dismissed = false;

    SplashScreen(SheepdogMIDlet midlet)
    {
        this.midlet = midlet;
        setFullScreenMode(true);
        splashImage = SheepdogMIDlet.createImage("/splash.png");
        new Thread(this).start();
    }

    public void run()
    {
        synchronized(this)
        {
            try
            {
                wait(3000L); // 3 seconds
            }
            catch (InterruptedException e)
            {
                // can't happen in MIDP: no Thread.interrupt method
            }
            dismiss();
        }
    }

    public void paint(Graphics g)
    {
        int width = getWidth();
        int height = getHeight();
        g.setColor(0x00FFFFFF); // white
        g.fillRect(0, 0, width, height);
        g.setColor(0x00FF0000); // red
        g.drawRect(1, 1, width-3, height-3); // red border one pixel from edge
        if (splashImage != null)
        {
            g.drawImage(splashImage,
                        width/2,
                        height/2,
                        Graphics.VCENTER | Graphics.HCENTER);
            splashImage = null;
            midlet.splashScreenPainted();
        }
    }

    public synchronized void keyPressed(int keyCode)
    {
        dismiss();
    }
}
private void dismiss()
{
    if (!dismissed)
    {
        dismissed = true;
        midlet.splashScreenDone();
    }
}

A.3 MenuList.java

The menu list is the screen shown after the splash screen, or when the game is over, or when the user presses a nongame key during the game (and thereby pauses it). If the game has been paused, there is an extra menu item, Continue, at the top of the list.

// unnamed package
import javax.microedition.lcdui.*;

class MenuList
    extends List
    implements CommandListener
{
    private SheepdogMIDlet midlet;
    private Command exitCommand;
    private boolean gameActive = false;

    MenuList(SheepdogMIDlet midlet)
    {
        super("Sheepdog", List.IMPLICIT);
        this.midlet = midlet;
        append("New game", null);
        append("High score", null);
        append("Instructions", null);
        exitCommand = new Command("Exit", Command.EXIT, 1);
        addCommand(exitCommand);
        setCommandListener(this);
    }

    void setGameActive(boolean active)
    {
        if (active && !gameActive)
        {
            gameActive = true;
            insert(0, "Continue", null);
        }
    }

else if (!active && gameActive)
{
    gameActive = false;
    delete(0);
}
}

public void commandAction(Command c, Displayable d)
{
    if (c == List.SELECT_COMMAND)
    {
        int index = getSelectedIndex();
        if (index != -1) // should never be -1
        {
            if (!gameActive)
            {
                index++;
            }
            switch (index)
            {
                case 0: // Continue
                    midlet.menuListContinue();
                    break;
                case 1: // New game
                    midlet.menuListNewGame();
                    break;
                case 2: // High score
                    midlet.menuListHighScore();
                    break;
                case 3:
                    midlet.menuListInstructions();
                    break;
                default:
                    // can't happen
                    break;
            }
        }
        else if (c == exitCommand)
        {
            midlet.menuListQuit();
        }
    }
}

A.4 InstructionsScreen.java

Figure 8: Instructions screen

The instructions screen displays instruction text to the user.

// unnamed package
import javax.microedition.lcdui.*;
class InstructionsScreen  
    extends Form  
    implements CommandListener  
    
    private final SheepdogMIDlet midlet;  
    private final Command backCommand;  
    
    private static final String instructions =  
        "Herd the sheep into the fold as quickly as you can.\n" +  
        "Sheep won't leave the fold once they've entered it.\n" +  
        "If they're not behaving, bark using the Fire key!";  
    
    InstructionsScreen(SheepdogMIDlet midlet)  
    {  
        super("Instructions");  
        this.midlet = midlet;  
        append(new StringItem(null, instructions));  
        backCommand = new Command("Back", Command.BACK, 1);  
        addCommand(backCommand);  
        setCommandListener(this);  
    }  
    
    public void commandAction(Command c, Displayable d)  
    {  
        midlet.instructionsBack();  
    }  
}  

A.5 HighScoreScreen.java  

```java  
import javax.microedition.lcdui.*;  
  
class HighScoreScreen  
    extends Form  
    implements CommandListener  
    
    private final SheepdogMIDlet midlet;  
    private final Command backCommand;  
    
    HighScoreScreen(SheepdogMIDlet midlet)  
```  

Figure 9: High-score screen  

The high-score screen is used to display the current high score (the best time so far). Since the initial  
placing of the sheep is random, this isn’t actually very meaningful in this game.  

```java  
// unnamed package  
import javax.microedition.lcdui.*;  
  
class HighScoreScreen  
    extends Form  
    implements CommandListener  
    
    private final SheepdogMIDlet midlet;  
    private final Command backCommand;  
  ```
{  
super("High score");  
this.midlet = midlet;  

long bestTime = midlet.getBestTime();  
String text = (bestTime == -1) ? "none yet"  
    : (Long.toString(bestTime) + "s");  
append(new StringItem("Best time", text));

backCommand = new Command("Back", Command.BACK, 1);  
addCommand(backCommand);  
setCommandListener(this);
}

public void commandAction(Command c, Displayable d)  
{  
    midlet.highScoreBack();  
}

A.6 SheepdogCanvas.java

Figure 10: Sheepdog canvas

The sheepdog canvas is the main game screen. It displays a sheepdog and five sheep on a grassy island surrounded by water. If the screen is smaller than the game area, it scrolls the screen to keep the view focused on the sheepdog.

// unnamed package
import java.util.Random;
import java.util.Vector;
import javax.microedition.lcdui.*;
import javax.microedition.lcdui.game.*;
import javax.microedition.media.*;
import java.io.*;

class SheepdogCanvas  
    extends GameCanvas  
    implements Runnable
{  
    // shared direction constants  
    static final int NONE = -1;  
    static final int UP = 0;  
    static final int LEFT = 1;  
    static final int DOWN = 2;  
    static final int RIGHT = 3;

    private static final int MILLIS_PER_TICK = 50;  
    private static final int NUM_SHEEP = 5;

    private final SheepdogMIDlet midlet;  
    private final Field field;
private final Sheepdog sheepdog;
private final Vector sheep = new Vector();
private final LayerManager layerManager;
private final Graphics graphics;
private long gameDuration;
private long startTime;
private volatile Thread animationThread = null;

SheepdogCanvas(SheepdogMIDlet midlet)
{
    super(true); // suppress key events for game keys
    this.midlet = midlet;
    graphics = getGraphics();

    layerManager = new LayerManager();
    field = new Field();
    sheepdog = new Sheepdog(this);
    layerManager.append(sheepdog);
    for (int i = 0; i < NUM_SHEEP; ++i)
    {
        Sheep newSheep = new Sheep(this);
        layerManager.append(newSheep);
        sheep.addElement(newSheep);
    }
    layerManager.append(field); // last layer, behind sprites
    init();
}

public void keyPressed(int keyCode)
{
    // The constructor suppresses key events for game keys, so we'll
    // only get key events for non-game keys. The number keys, * & #
    // have positive keyCodes, so negative keyCodes mean non-game
    // special keys like soft-keys. We'll use key-presses on special
    // keys to take us to the menu.
    if (keyCode < 0)
    {
        stop();
        midlet.sheepdogCanvasMenu();
    }
}

void init()
{
    sheepdog.setPosition(field.getSheepdogStartX(),
                        field.getSheepdogStartY());

    for (int i = 0; i < sheep.size(); ++i)
    {
        Sheep sh = (Sheep)(sheep.elementAt(i));
        // find a valid position for the sheep
        do
        {
            int x = midlet.random(field.getWidth() - Sheep.WIDTH);
            int y = midlet.random(field.getHeight() - Sheep.HEIGHT);
            sh.setPosition(x, y);
        } while (field.containsImpassableArea(sh.getX(),
                                              sh.getY(),
                                              sh.getWidth(),
                                              sh.getHeight()) ||
                  field.containsCoilArea(sh.getX(),
                                          sh.getY(),
                                          sh.getWidth(),
                                          sh.getHeight()));
    }
}
public synchronized void start()
{
    animationThread = new Thread(this);
    animationThread.start();
    startTime = System.currentTimeMillis() - gameDuration;
}

public synchronized void stop()
{
    gameDuration = System.currentTimeMillis() - startTime;
    animationThread = null;
}

public void run()
{
    Thread currentThread = Thread.currentThread();
    try
    {
        // This ends when animationThread is set to null, or when
        // it is subsequently set to a new thread; either way, the
        // current thread should terminate
        while (currentThread == animationThread)
        {
            long startTime = System.currentTimeMillis();
            // Don't advance game or draw if canvas is covered by
            // a system screen.
            if (isShown())
            {
                tick();
                draw();
                flushGraphics();
            }
            long timeTaken = System.currentTimeMillis() - startTime;
            if (timeTaken < MILLIS_PER_TICK)
            {
                synchronized (this)
                {
                    wait(MILLIS_PER_TICK - timeTaken);
                }
            }
            else
            {
                currentThread.yield();
            }
        }
    }
    catch (InterruptedException e)
    {
        // won't be thrown
    }
}

private void tick()
{
    // If player presses two or more direction buttons, we ignore them
    // all. But pressing fire is independent. The code below also ignores

    overlapsSheepdog(sh) ||
    overlapsSheep(sh, i) ||
    field.inFold(sh));
}
// direction buttons if GAME_A..GAME_D are pressed.
int keyStates = getKeyStates();
boolean bark = (keyStates & FIRE_PRESSED) != 0;
keyStates &= ~FIRE_PRESSED;
int direction = (keyStates == UP_PRESSED) ? UP :
                (keyStates == LEFT_PRESSED) ? LEFT:
                (keyStates == DOWN_PRESSED) ? DOWN :
                (keyStates == RIGHT_PRESSED) ? RIGHT : NONE;
sheepdog.tick(direction, bark);

for (int i = 0; i < sheep.size(); ++i)
  {  
    Sheep sh = (Sheep)(sheep.elementAt(i));
    sh.tick();
  }
field.tick();

Field getField()
{  
  return field;
}

Sheepdog getSheepdog()
{  
  return sheepdog;
}

Vector getSheep()
{  
  return sheep;
}

void handleDogBark()
{  
  for (int i = 0; i < sheep.size(); ++i)
    {  
      Sheep sh = (Sheep)(sheep.elementAt(i));
      sh.handleDogBark();
    }
}

boolean overlapsSheepdog(Sprite sprite)
{  
  return sprite.collidesWith(sheepdog, false); // false -> not pixelLevel
}

boolean overlapsSheep(Sprite sprite)
{  
  return overlapsSheep(sprite, sheep.size());
}

// whether the sprite overlaps the first 'count' sheep
boolean overlapsSheep(Sprite sprite, int count)
{  
  for (int i = 0; i < count; ++i)
    {  
      Sheep sh = (Sheep)(sheep.elementAt(i));
      if (sprite.collidesWith(sh, false)) // false -> not pixelLevel
      {  

boolean overlapsOtherSheep(Sprite sprite)
{
    for (int i = 0; i < sheep.size(); ++i)
    {
        Object obj = sheep.elementAt(i);
        if (obj != sprite)
        {
            Sheep sh = (Sheep)obj;
            if (sprite.collidesWith(sh, false)) // false -> not pixelLevel
            {
                return true;
            }
        }
    }
    return false;
}

void vibrate(int millis)
{
    midlet.vibrate(millis);
}

// draw game
private void draw()
{
    int width = getWidth();
    int height = getHeight();

    // clear screen to grey
    graphics.setColor(0x00888888);
    graphics.fillRect(0, 0, width, height);

    // clip and translate to centre
    int dx = origin(sheepdog.getX() + sheepdog.getWidth() / 2,
                    field.getWidth(),
                    width);
    int dy = origin(sheepdog.getY() + sheepdog.getHeight() / 2,
                    field.getHeight(),
                    height);
    graphics.setClip(dx, dy, field.getWidth(), field.getHeight());
    graphics.translate(dx, dy);

    // draw background and sprites
    layerManager.paint(g, 0, 0);

    // undo clip & translate
    graphics.translate(-dx, -dy);
    graphics.setClip(0, 0, width, height);

    // display time & score
    long time = (System.currentTimeMillis() - startTime) / 1000;
    int score = numSheepInFold();
    graphics.setColor(0x00FFFFFF); // white
    graphics.drawString(Integer.toString(score),
                        1,
                        1,
                        Graphics.TOP | Graphics.LEFT);
    graphics.drawString(Long.toString(time),
                        width - 2,
                        Graphics.TOP | Graphics.LEFT);
if (score == sheep.size())
{
    midlet.sheepdogCanvasGameOver(time);
}

// If the screen is bigger than the field // in the screen. Otherwise we center the screen on the focus, except // that we don't scroll beyond the edges of the field.
private int origin(int focus, int fieldLength, int screenLength)
{
    int origin;
    if (screenLength >= fieldLength)
    {
        origin = (screenLength - fieldLength) / 2;
    }
    else if (focus <= screenLength / 2)
    {
        origin = 0;
    }
    else if (focus >= (fieldLength - screenLength / 2))
    {
        origin = screenLength - fieldLength;
    }
    else
    {
        origin = screenLength / 2 - focus;
    }
    return origin;
}

int numSheepInFold()
{
    int count = 0;
    for (int i = 0; i < sheep.size(); ++i)
    {
        Sheep sh = (Sheep)(sheep.elementAt(i));
        if (field.inFold(sh))
        {
            count++;
        }
    }
    return count;
}

A.7 Field.java

This class is a TiledLayer that draws the grassy field and water. The water tiles are animated so that they constantly change a little, simulating waves. Its tile image is shown in Figure 11.

Figure 11: Field tile image, 160 × 16 pixels

// unnamed package
import javax.microedition.lcdui.*;
import javax.microedition.lcdui.game.*;
class Field
extends TiledLayer
{
private static final int WIDTH_IN_TILES = 12;
private static final int HEIGHT_IN_TILES = 12;
private static final int TILE_WIDTH = 16;
private static final int TILE_HEIGHT = 16;

private static int[][] cellTiles =
{{-3, -2, -3, -1, -2, -1, -3, -1, -2, -3, -1, -2},
{-2, 3, 4, 3, 1, 2, 3, 2, 1, 5, 2, -3},
{-1, 2, 1, 2, 3, 4, 5, 3, 2, 4, 3, -1},
{-2, 1, 4, 9, 9, 9, 4, 5, 2, 1, -2},
{-3, 3, 5, 9, 10, 10, 2, 1, 3, 5, -1},
{-2, 2, 3, 9, 10, 10, 5, 4, 2, 1, -3},
{-1, 4, 2, 9, 9, 9, 3, 1, 3, 2, -2},
{-3, 2, 5, 1, 3, 1, 4, 2, 5, 4, 3, -3},
{-2, 1, 4, 2, 5, 2, 3, 4, 2, 1, 2, -1},
{-1, 5, 1, 4, 3, 4, 1, 2, 3, 4, 1, -2},
{-3, 2, 4, 5, 2, 3, 2, 4, 1, 2, 3, -3},
{-2, -3, -2, -1, -2, -1, -3, -1, -2}));
private static int FOLD_TILE = 10;
private static int FENCE_TILE = 9;
private static int[][] waterFrames = {{6, 7, 8}, {7, 8, 6}, {8, 6, 7}};

private int tickCount = 0;

Field()
{
    super(WIDTH_IN_TILES,
         HEIGHT_IN_TILES,
         SheepdogMIDlet.createImage("/field.png"),
         TILE_WIDTH,
         TILE_HEIGHT);

    createAnimatedTile(waterFrames[0][0]); // tile -1
    createAnimatedTile(waterFrames[1][0]); // tile -2
    createAnimatedTile(waterFrames[2][0]); // tile -3

    for (int row = 0; row < HEIGHT_IN_TILES; ++row)
    {
        for (int column = 0; column < WIDTH_IN_TILES; ++column)
        {
            setCell(column, row, cellTiles[row][column]);
        }
    }

    int getSheepdogStartX()
    {
        return getWidth() - 50;
    }

    int getSheepdogStartY()
    {
        return getHeight() - 50;
    }

    void tick()
    {
        int tickState = (tickCount++ >> 3); // slow down x8
        int tile = tickState % 3;
        setAnimatedTile(-1-tile, waterFrames[tile][(tickState % 9) / 3]);
    }
// return true if any part of the rectangle overlaps a water tile
// or the fence
boolean containsImpassableArea(int x, int y, int width, int height) {
    int rowMin = y / TILE_HEIGHT;
    int rowMax = (y + height - 1) / TILE_HEIGHT;
    int columnMin = x / TILE_WIDTH;
    int columnMax = (x + width - 1) / TILE_WIDTH;

    for (int row = rowMin; row <= rowMax; ++row) {
        for (int column = columnMin; column <= columnMax; ++column) {
            int cell = getCell(column, row);
            if ((cell < 0) || (cell == FENCE_TILE)) {
                return true;
            }
        }
    }
    return false;
}

// returns true if every pixel of the sprite is in the fold
boolean inFold(Sprite s) {
    // we can assume that the sprite’s reference pixel is unchanged
    int rowMin = s.getY() / TILE_HEIGHT;
    int rowMax = (s.getY() + s.getHeight() - 1) / TILE_HEIGHT;
    int columnMin = s.getX() / TILE_WIDTH;
    int columnMax = (s.getX() + s.getWidth() - 1) / TILE_WIDTH;

    for (int row = rowMin; row <= rowMax; ++row) {
        for (int column = columnMin; column <= columnMax; ++column) {
            if (getCell(column, row) != FOLD_TILE) {
                return false;
            }
        }
    }
    return true;
}

A.8 Sheepdog.java

This class is a Sprite that draws the sheepdog. Its animation frame image is shown in Figure 12. Note that there are no frames for the sheepdog facing right: when it is facing right, the facing-left image is used with an image transform of TRANS_MIRROR.

Figure 12: Sheepdog animation frame image, 225 × 15 pixels

// unnamed package

import javax.microedition.lcdui.*;
import javax.microedition.lcdui.game.*;
class Sheepdog
extends Sprite {
    static final int WIDTH = 15;
    static final int HEIGHT = 15;
    static final int VIBRATION_MILLIS = 200;

    private final SheepdogCanvas canvas;
    private boolean barking = false;
    private int[][][] animations = {{{0}, // stand up
                                    {1, 2, 3, 4}, // run up
                                    {5}, // stand left
                                    {6, 7, 8, 9}, // run left
                                    {10}, // stand down
                                    {11, 12, 13, 14}}}; // run down

    private int animationTick = 0;
    private static final int STAND = 0;
    private static final int RUN = 1;
    private int currentDirection = SheepdogCanvas.LEFT;

    Sheepdog(SheepdogCanvas canvas)
    {
        super(SheepdogMIDlet.createImage("/dog.png"), WIDTH, HEIGHT);
        defineCollisionRectangle(2, 2, WIDTH-4, WIDTH-4);
        defineReferencePixel(WIDTH/2, HEIGHT/2);

        this.canvas = canvas;
    }

    void tick(int direction, boolean bark)
    {
        animationTick++;
        Field field = canvas.getField();
        boolean moving = false;
        switch (direction)
        {
            case SheepdogCanvas.UP:
                currentDirection = direction;
                if ((getY() > 0) &&
                    !field.containsImpassableArea(getX(),
                                               getY() - 1,
                                               getWidth(),
                                               1) &&
                    moveSuccessfully(0, -1))
                {
                    moving = true;
                }
                else
                {
                    canvas.vibrate(VIBRATION_MILLIS);
                }
                break;
            case SheepdogCanvas.LEFT:
                currentDirection = direction;
                if ((getX() > 0) &&
                    !field.containsImpassableArea(getX() - 1,
                                               getY(),
                                               1,
                                               getHeight()) &&
                    moveSuccessfully(-1, 0))
                {
                    moving = true;
                }
                else
                {

canvas.vibrate(VIBRATION_MILLIS);
} break;
case SheepdogCanvas.DOWN:
currentDirection = direction;
if ((getY() + getHeight() < field.getWidth()) &&
!field.containsImpassableArea(getX(),
getY() + getHeight(),
getWidth(),
1) &&
moveSuccessfully(0, 1))
{
    moving = true;
} else
{
canvas.vibrate(VIBRATION_MILLIS);
} break;
case SheepdogCanvas.RIGHT:
currentDirection = direction;
if ((getX() + getWidth() < field.getWidth()) &&
!field.containsImpassableArea(getX() + getWidth(),
getY(),
1,
getHeight()) &&
moveSuccessfully(1, 0))
{
    moving = true;
} else
{
canvas.vibrate(VIBRATION_MILLIS);
} break;
default: // must be NONE
    break;
}
if (moving)
{
    advanceRunningAnimation();
} else
{
    setStandingAnimation();
}

// implement a toggle, so bark only happens once per click
// (will therefore not register very rapid multiple-clicks)
if (bark)
{
    if (!barking)
    {
        SoundEffects.getInstance().startDogSound();
barking = true;
canvas.handleDogBark();
    }
}
else
{
barking = false;
}

private boolean moveSuccessfully(int dx, int dy)
move(dx, dy);
if (canvas.overlapsSheep(this))
{
    move(-dx, -dy);
    return false;
}
else
{
    return true;
}

private void advanceRunningAnimation()
{
    int[] sequence;
    if (currentDirection == SheepdogCanvas.RIGHT)
    {
        sequence = animations[SheepdogCanvas.LEFT][RUN];
        setTransform(TRANS_MIRROR);
    }
    else
    {
        sequence = animations[currentDirection][RUN];
        setTransform(TRANS_NONE);
    }
    setFrame((animationTick >> 1) % sequence.length);
}

private void setStandingAnimation()
{
    if (currentDirection == SheepdogCanvas.RIGHT)
    {
        setFrame(animations[SheepdogCanvas.LEFT][STAND][0]);
        setTransform(TRANS_MIRROR);
    }
    else
    {
        setFrame(animations[currentDirection][STAND][0]);
        setTransform(TRANS_NONE);
    }
}

### A.9 Sheep.java

This class is a Sprite that draws a sheep. Its animation frame image is shown in Figure 13. Note that there are no frames for the sheep facing right: when it is facing right, the facing-left image is used with an image transform of TRANS_MIRROR.

![Sheep animation frame image, 225 x 15 pixels](image)

// unnamed package
import javax.microedition.lcdui.*;
import javax.microedition.lcdui.game.*;
import java.util.*;
import java.util.*;

class Sheep
    extends Sprite
{
    static final int WIDTH = 15;
static final int HEIGHT = 15;

private final SheepdogCanvas canvas;
private int[][][][] animations = {{{0},  // stand up
{1, 2, 3, 4}},  // run up
{5},  // stand left
{6, 7, 8, 9}},  // run left
{10},  // stand down
{11, 12, 13, 14}}};  // run down

private int animationTick;
private static int numSheep = 0;
private static final int STAND = 0;
private static final int RUN = 1;
private int currentDirection = SheepdogCanvas.DOWN;

private final int flockFactor;
private final int minDogFactor;
private final int maxDogFactor;

Sheep(SheepdogCanvas canvas)
{
    super(SheepdogMIDlet.createImage("/sheep.png"), WIDTH, HEIGHT);
defineCollisionRectangle(2, 2, WIDTH-4, WIDTH-4);
defineReferencePixel(WIDTH/2, HEIGHT/2);

    this.canvas = canvas;
    animationTick = numSheep++;
    flockFactor = 100 + SheepdogMIDlet.random(100);
    minDogFactor = SheepdogMIDlet.random(20);
    maxDogFactor = minDogFactor + 10;
    dogFactor = minDogFactor;
}

void tick()
{
    // sheep are 4x as slow as dogs
    if ((animationTick++ % 4) != 0)
    {
        return;
    }

    // adjust dog factor
    adjustDogFactor();

    // ARTIFICIAL INTELLIGENCE SECTION
    // - wants to move away from dog, if dog is close
    // - wants to move closer to flock (average position of other
        // sheep) if they are close
    // - if preferred direction is diagonal and major direction is
        // blocked, take minor direction
    // - each sheep varies in how much it's scared of the dog, and
        // how much it wants to flock
    // We do this by calculating a weighted direction vector

    // First calculate dog effect
    Sheepdog sheepdog = canvas.getSheepdog();
    int dx = sheepdog.getX() - getX();
    int dy = sheepdog.getY() - getY();
    int sumsq = dx * dx + dy * dy;
    Field field = canvas.getField();
    int dogEffectX =
        dogFactor * dx * field.getWidth() * field.getWidth() / sumsq;
    int dogEffectY =
        dogFactor * dy * field.getHeight() * field.getHeight() / sumsq;
Next calculate flock effect

```java
int flockDx = 0;
int flockDy = 0;
Vector sheep = canvas.getSheep();
for (int i = 0; i < sheep.size(); ++i)
    { Sheep sh = (Sheep)(sheep.elementAt(i));
      if (sh != this)
          { flockDx += getX() - sh.getX();
            flockDy += getY() - sh.getY();
            }
    } int flockEffectX = (flockDx * flockFactor) / (sheep.size() - 1);
int flockEffectY = (flockDy * flockFactor) / (sheep.size() - 1);

// Now calculate total effect
int totalEffectX = dogEffectX - flockEffectX;
int totalEffectY = dogEffectY - flockEffectY;

// Determine preferred directions
int firstDirection;
int secondDirection;
int thirdDirection;
if (Math.abs(totalEffectY) > Math.abs(totalEffectX))
    { // Prefer to move vertically
      if (totalEffectY > 0)
          { firstDirection = SheepdogCanvas.DOWN; }
      else
          { firstDirection = SheepdogCanvas.UP; }
      if (totalEffectX > 0)
          { secondDirection = SheepdogCanvas.RIGHT;
            thirdDirection = SheepdogCanvas.NONE; }
      else if (totalEffectX < 0)
          { secondDirection = SheepdogCanvas.LEFT;
            thirdDirection = SheepdogCanvas.NONE; }
      else // totalEffectX == 0
          { if (SheepdogMIDlet.random(2) == 0)
              { secondDirection = SheepdogCanvas.LEFT;
                thirdDirection = SheepdogCanvas.RIGHT; }
          else
              { secondDirection = SheepdogCanvas.RIGHT;
                thirdDirection = SheepdogCanvas.LEFT; }
          }
    }
else
    { // Prefer to move horizontally
      if (totalEffectX > 0)
          { firstDirection = SheepdogCanvas.RIGHT; }
      else
          { }
    }
```
firstDirection = SheepdogCanvas.LEFT;
}
if (totalEffectY > 0)
{
    secondDirection = SheepdogCanvas.DOWN;
    thirdDirection = SheepdogCanvas.NONE;
} else if (totalEffectY < 0)
{
    secondDirection = SheepdogCanvas.UP;
    thirdDirection = SheepdogCanvas.NONE;
} else // totalEffectY == 0
{
    if (SheepdogMIDlet.random(2) == 0)
    {
        secondDirection = SheepdogCanvas.UP;
        thirdDirection = SheepdogCanvas.DOWN;
    } else
    {
        secondDirection = SheepdogCanvas.DOWN;
        thirdDirection = SheepdogCanvas.UP;
    }
}

// if we can move in the preferred directions, do so, else
// stand facing the dog
if (tryMove(firstDirection) ||
    tryMove(secondDirection) ||
    ((thirdDirection != SheepdogCanvas.NONE) &&
    tryMove(thirdDirection)))
{
    advanceRunningAnimation();
} else
{
    if (Math.abs(dx) > Math.abs(dy))
    {
        if (dx > 0)
        {
            currentDirection = SheepdogCanvas.LEFT;
        } else
        {
            currentDirection = SheepdogCanvas.RIGHT;
        }
    } else
    {
        if (dy > 0)
        {
            currentDirection = SheepdogCanvas.UP;
        } else
        {
            currentDirection = SheepdogCanvas.DOWN;
        }
    }
    setStandingAnimation();
}

// Will baa occasionally if dog is close. Dog distance ranges from
// about 11 minimum to double width of field
int dogDistance = Math.abs(dx) + Math.abs(dy);
if (SheepdogMIDlet.random(dogDistance - 10) == 0)
{
SoundEffects.getInstance().startSheepSound();
}

private void adjustDogFactor()
{
    dogFactor += SheepdogMIDlet.random(4) - 2;  // -2..1
    if (dogFactor < minDogFactor)
    {
        dogFactor = minDogFactor;
    }
    else if (dogFactor > maxDogFactor)
    {
        dogFactor = maxDogFactor;
    }
}

private boolean tryMove(int direction)
{
    Field field = canvas.getField();
    boolean blocked = true;
    int dx = 0;
    int dy = 0;
    switch (direction)
    {
    case SheepdogCanvas.UP:
        if ((getY() > 0) &&
            !field.containsImpassableArea(getX(),
            getY() - 1,
            getWidth(),
            1))
        {
            blocked = false;
            dy = -1;
        }
        break;
    case SheepdogCanvas.LEFT:
        if ((getX() > 0) &&
            !field.containsImpassableArea(getX() - 1,
            getY(),
            1,
            getHeight()))
        {
            blocked = false;
            dx = -1;
        }
        break;
    case SheepdogCanvas.DOWN:
        if ((getY() + getHeight() - 1 < field.getWidth()) &&
            !field.containsImpassableArea(getX(),
            getY() + getHeight(),
            getWidth(),
            1))
        {
            blocked = false;
            dy = 1;
        }
        break;
    case SheepdogCanvas.RIGHT:
        if ((getX() + getWidth() - 1 < field.getWidth()) &&
            !field.containsImpassableArea(getX() + getWidth(),
            getY(),
            1,
            getHeight()))
        {
            blocked = false;
        }
        break;
    }
dx = 1;
break;
default:
    // can't happen
    break;
}

boolean success = false;
if (!blocked)
{
    boolean wasInFold = field.inFold(this);
    move(dx, dy);
    if (canvas.overlapsOtherSheep(this) ||
        canvas.overlapsSheepdog(this) ||
        (wasInFold && !field.inFold(this)))
    {
        move(-dx, -dy);
    }
    else
    {
        currentDirection = direction;
        success = true;
    }
}
return success;

private void advanceRunningAnimation()
{
    int[] sequence;
    if (currentDirection == SheepdogCanvas.RIGHT)
    {
        sequence = animations[SheepdogCanvas.LEFT][RUN];
        setTransform(TRANS_MIRROR);
    }
    else
    {
        sequence = animations[currentDirection][RUN];
        setTransform(TRANS_NONE);
    }
    setFrame(sequence[(animationTick >> 2) % sequence.length]);
}

private void setStandingAnimation()
{
    if (currentDirection == SheepdogCanvas.RIGHT)
    {
        setFrame(animations[SheepdogCanvas.LEFT][STAND][0]);
        setTransform(TRANS_MIRROR);
    }
    else
    {
        setFrame(animations[currentDirection][STAND][0]);
        setTransform(TRANS_NONE);
    }
}

void handleDogBark()
{
    // sheep should get nervous
    dogFactor += 5;
    if (dogFactor > maxDogFactor)
    {
    }
dogFactor = maxDogFactor;
}
}

A.10  GameOverScreen.java

Figure 14: Game-over screen

This screen is displayed when the player gets the fifth sheep into the fold. It shows the player's time, and indicates whether it's the new best time or shows the player what the best time is. When it appears, the screen flashes for a second and one of two short MIDI tunes is played.

Note the useful trick for drawing outlined text: first draw it four times in the outline color, offset up, down, left, and right. Then draw it in the text color at its normal location. Beware of using this in the main game screen, as text drawing can be quite slow and this makes it five times slower!

// unnamed package
import javax.microedition.lcdui.*;

class GameOverScreen
    extends Canvas
{
    private final SheepdogMIDlet midlet;
    private boolean wasBestTime;
    private long time;
    private long bestTime;

    GameOverScreen(SheepdogMIDlet midlet, long time)
    {
        super();
        this.midlet = midlet;
        this.time = time;
        setFullScreenMode(true);

        if (midlet.checkBestTime(time))
        {
            wasBestTime = true;
            bestTime = time;
            SoundEffects.getInstance().startHighScoreSound();
        }
        else
        {
            wasBestTime = false;
            bestTime = midlet.getBestTime();
            SoundEffects.getInstance().startGameOverSound();
        }
        midlet.flashBacklight(1000);  // 1 second
public void paint(Graphics g)
{
    int width = getWidth();
    int height = getHeight();

    // clear screen to green
    g.setColor(0x0000FF00);
    g.fillRect(0, 0, width, height);

    // Write message. We use a trick to make outlined text: we draw it
    // offset one pixel to the top, bottom, left & right in white, then
    // centred in black.
    g.setFont(Font.getFont(Font.FACE_PROPORTIONAL,
                          Font.STYLE_BOLD,
                          Font.SIZE_LARGE));
    int centerX = width / 2;
    int centerY = height / 2;
    g.setColor(0x00FFFFFF);   // white
    drawText(g, centerX, centerY - 1);
    drawText(g, centerX, centerY + 1);
    drawText(g, centerX - 1, centerY);
    drawText(g, centerX + 1, centerY);
    g.setColor(0x00000000);   // black
    drawText(g, centerX, centerY);
}

private void drawText(Graphics g, int centerX, int centerY)
{
    int fontHeight = g.getFont().getHeight();
    int textHeight = 3 * fontHeight;
    int topY = centerY - textHeight / 2;
    g.drawString("GAME OVER",
               centerX,
               topY,
               Graphics.HCENTER | Graphics.TOP);
    g.drawString("Time: " + time + "s",
               centerX,
               topY + fontHeight,
               Graphics.HCENTER | Graphics.TOP);
    g.drawString(wasBestTime ? "New best time!" :
                    ("Best time: " + bestTime + "s"),
               centerX,
               topY + 2 * fontHeight,
               Graphics.HCENTER | Graphics.TOP);
}

public void keyPressed(int keyCode)
{
    midlet.gameOverDone();
}

A.11 SoundEffects.java

This class uses the MIDP 2.0 Media API to play four sound effects: a sheep’s "baa," a dog’s bark, and
two short game-over MIDI tunes.

// unnamed package
import javax.microedition.media.*;
import java.io.*;

class SoundEffects
private static SoundEffects instance;
private Player sheepSoundPlayer;
private Player dogSoundPlayer;

private SoundEffects() {
  sheepSoundPlayer = createPlayer("/sheep.wav", "audio/x-wav");
  dogSoundPlayer = createPlayer("/dog.wav", "audio/x-wav");
}

static SoundEffects getInstance() {
  if (instance == null) {
    instance = new SoundEffects();
  }
  return instance;
}

void startSheepSound() {
  startPlayer(sheepSoundPlayer);
}

void startDogSound() {
  startPlayer(dogSoundPlayer);
}

void startGameOverSound() {
  startPlayer(createPlayer("/gameover.mid", "audio/midi"));}

void startHighScoreSound() {
  startPlayer(createPlayer("/highscore.mid", "audio/midi"));}

private void startPlayer(Player p) {
  if (p != null) {
    try {
      p.stop();
      p.setMediaTime(0L);
      p.start();
    } catch (MediaException me) {
      // ignore
    }
  }
}

private Player createPlayer(String filename, String format) {
  Player p = null;
try {
    InputStream is = getClass().getResourceAsStream(filename);
    p = Manager.createPlayer(is, format);
    p.prefetch();
} catch (IOException ioe)
    { // ignore
    }
    catch (MediaException me)
    { // ignore
    }
    return p;
}