Game API 入門

Copyright 2003 Sun Microsystems, Inc. All Rights Reserved. Enterprise Services

本章目的

在前面的章節中，我們曾經提過遊戲設計的基本概念。一個遊戲的大致結構不外乎是產生一個執行緒負責計算整個遊戲的狀態，然後再發出重繪事件要求螢幕重畫。

從 MIDP 2.0 開始，提供了 javax.microedition.lcdui.game 套件，此套件又稱作 Game API。

Game API 提供很多設計 Game 時常用的功能，讓程式設計師不用重新開發這些功能。Game API 也可以簡化遊戲的設計。由於 Game API 架構在 MIDP 的低階 API 上，因此它可以和低階 API 結合在一起。

本文將介紹 Game API 的使用方法。

Copyright 2003 Sun Microsystems, Inc. All Rights Reserved. Enterprise Services

參考資源與書目

Sun 官方網站文章

Creating 2D Action Games with the Game API http://wireless.java.sun.com/midp/articles/game/

其他文章

MIDP 2.0: The Game API http://www.microjava.com/articles/techtalk/game_api?content_id=4271
J2ME & Gaming(免費電子書) http://www.jasonlam604.com/books.php#j2megaming

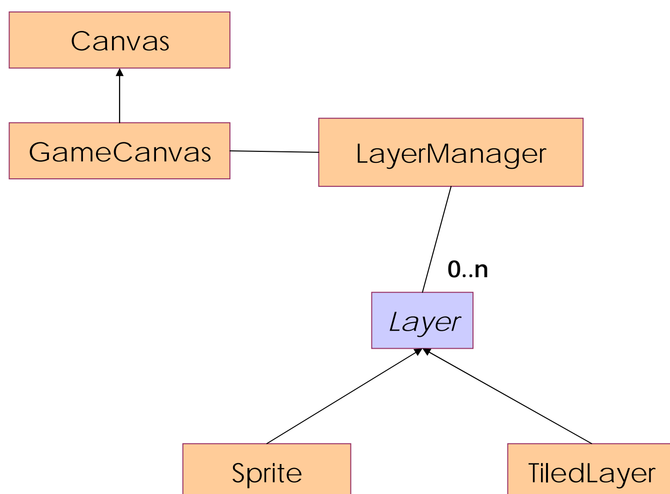
書籍

J2Me Game Programming

Copyright 2003 Sun Microsystems, Inc. All Rights Reserved. Enterprise Services

Game API 的結構體系

Game API 由五個類別所構成，它們的關係如下:



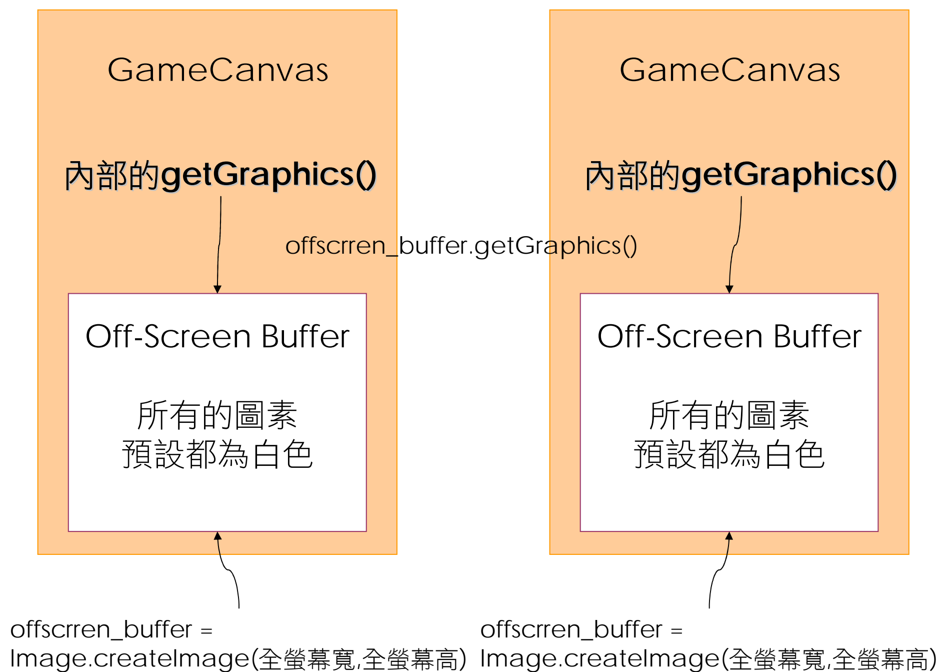
其中，GameCanvas 繼承自 Canvas，所以具有所有 Canvas 所具有的功能，還額外增加了一些便於遊戲設計的功能。比方說過去我們要等到 keyPressed() / keyRelease() / keyRepeated()被叫用之後，才能得知目前按鍵被按下的情形，現在 GameCanvas 直接提供 getKeyStates()，我們可以在同一個執行緒自己偵測按鍵的狀態，過去的 keyPressed() / keyRelease() / keyRepeated()同時間只能偵測到一個按鈕被按下，在某些裝置下，getKeyStates()則可以偵測到很多按鈕同時間被按下的情形。GameCanvas 也提供了 flushGraphics()的功能，這個功能相當於過去叫用 repaint()再叫用 serviceRepaints()的功能，而且還帶有雙緩衝區的概念，不過 flushGraphics()並不會產生重繪事件，而是直接將 Off-Screen 的內容顯示到螢幕上，所以在 GameCanvas 中，paint()的地位就不像過去那樣重要了。

LayerManager 提供讓我們可以管理許多圖層的功能，讓我們可以很方便地將前景和背景混合在同一個畫面之後再輸出到螢幕上。LayerManager 中可以有許多 Layer 的子類別，標準的 MIDP 提供 Sprite 和 TiledLayer 這兩種 Layer 的子類別。

Copyright 2003 Sun Microsystems, Inc. All Rights Reserved. Enterprise Services

使用 GameCanvas

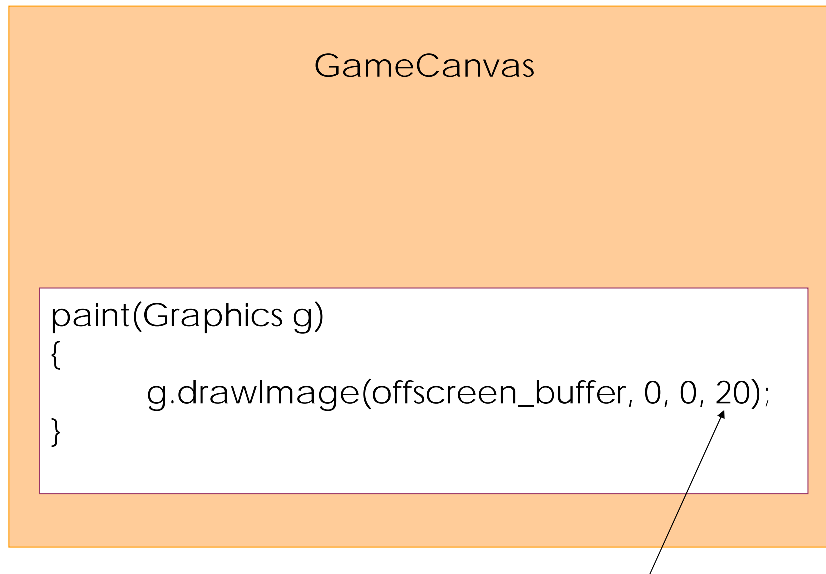
GameCanvas 的內部結構如下圖所示:



每產生一個 GameCanvas 的子類別，其內部就會自動產生一塊 Off-Screen，大小與全螢幕模式時一樣的高和寬。除非必要，否則請勿隨意產生太多的 GameCanvas，因為這樣會佔據太多的記憶體空間。

GameCanvas 預設的 paint()方法就是繪出 Off-Screen 的內容，如下圖所示:

Copyright 2003 Sun Microsystems, Inc. All Rights Reserved. Enterprise Services



所以一般來說，我們不需要在我們的類別之中重載(override)paint()方法。

為了使用 GameCanvas，我們撰寫主程式如下:

GameMIDlet.java

```
import javax.microedition.midlet.* ;
import javax.microedition.lcdui.* ;
public class GameMIDlet extends MIDlet
implements CommandListener
{
        Display display ;
        public GameMIDlet()
        {
                display = Display.getDisplay(this) ;
        }
        MyGameCanvas mgc ;
        public void startApp()
        {
                if(mgc==null)
                {
                        mgc = new MyGameCanvas();
                        mgc.addCommand(
                        new Command("開始", Command.OK, 1));
                        mgc.addCommand(
```


Copyright 2003 Sun Microsystems, Inc. All Rights Reserved. Enterprise Services

```
                    new Command("結束", Command.EXIT, 2));
                    mgc.setCommandListener(this);
                    display.setCurrent(mgc);
                }
        }
        public void commandAction(Command c,Displayable s)
        {
                String cmd = c.getLabel() ;
                if(cmd.equals("開始"))
                {
                        mgc.start() ;
                }else if(cmd.equals("結束"))
                {
                        mgc.exit() ;
                        notifyDestroyed() ;
                }
        }
        public void pauseApp()
        {

        }
        public void destroyApp(boolean unconditional)
        {
        }
}
```

我們把流程控制的功能放在主程式中，使用者一按下開始，就會 MyGameCanvas 裡的 start()方法開始遊戲迴圈；按下結束，我們就叫用 MyGameCanvas 的 stop()方法，stop()方法會設定結束旗標，藉此終止遊戲迴圈。MyGameCanvas 的程式碼如下:

MyGameCanvas.java

```
import javax.microedition.lcdui.* ;
import javax.microedition.lcdui.game.* ;
public class MyGameCanvas extends GameCanvas
implements Runnable
{
        public MyGameCanvas()
        {
                super(true) ;
```

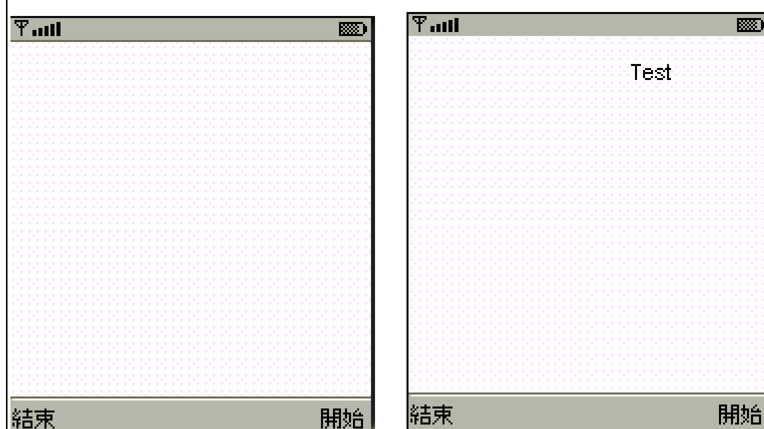

Copyright 2003 Sun Microsystems, Inc. All Rights Reserved. Enterprise Services

```
    }
    boolean conti = true ;
    int rate = 50 ;
    public void run()
    {
        long st = 0 ;
        long et = 0 ;
        Graphics g = getGraphics() ;
        while(conti)
        {
            st = System.currentTimeMillis() ;
            render(g) ;
            et = System.currentTimeMillis() ;
            if((et-st)<rate)
            {
                try
                {
                    Thread.sleep(rate-(et-st));
                }catch(Exception exp){}
            }
        }
    }
    int x = 50 ;
    int y = 10 ;
    public void render(Graphics g)
    {
        g.setColor(255,255,255);
        g.fillRect(0,0,getWidth(),getHeight());
        g.setColor(0,0,0);
        g.drawString("Test",x,y,0);
        x++ ;
        if(x > getWidth())
            x = 0 ;
        flushGraphics() ;
    }
    public void start()
    {
        Thread t = new Thread(this) ;
```


Copyright 2003 Sun Microsystems, Inc. All Rights Reserved. Enterprise Services

```
                t.start();
        }
        public void exit()
        {
                conti = false ;
        }
        public void keyPressed(int keycode)
        {
                System.out.println("按鍵事件");
        }
}
```

執行結果:



要享受 GameCanvas 的功能，第一件事情就是要繼承 GameCanvas。由於 GameCanvas 唯一的建構式需要一個參數，所以我們類別裡的建構式第一件事情就是叫用 GameCanvas(使用 super())，並傳入 true 或 false。這個 GameCanvas 建構式裡唯一的參數所代表的意義是:是否要抑制鍵盤事件，傳入 true 就是要抑制，傳入 false 就是不要。也就是說，如果我們傳入 true，那麼系統會抑制大多數的鍵盤事件的產生，所以我們所撰寫的 keyPressed() / keyReleased() / keyRepeated()將不會被呼叫(但是有些按鈕仍會產生鍵盤事件，看裝置而定。)。如果傳入 false，那麼就會像從前一樣，只要使用者按下了按鈕，就會產生鍵盤事件。

由於在 GameCanvas 中，我們有其他的方法(getKeyStates())可以取得按鍵被按下的狀態，因此大部分的情況下已經不需要鍵盤事件的輔助，如果我們確認不需要鍵盤事件，那麼抑制鍵盤事件可以增加處理效能，是較佳的選擇。

注意
所謂抑制鍵盤事件，指的是只有抑制目前這個 GameCanvas 的畫面，並不會抑制其他畫面的鍵盤事件。

在 GameCanvas 之中，我們都是將圖繪製在 Off-Screen 中，而不是像過去直

Copyright 2003 Sun Microsystems, Inc. All Rights Reserved. Enterprise Services

接繪製在螢幕上，所以我們會在程式中叫用 getGraphics()取得代表 Off-Screen 的 Graphics 物件，對此 Graphics 物件所做的任何動作，都是在 Off-Screen 上畫圖。不會影響到實際的螢幕。

注意
雖然每個 GameCanvas 之中，只有一個 Off-Screen，而且每次叫用 getGraphics() 時都會取得指向同一個 Off-Screen 的 Graphics，但是每叫用一次 getGraphics()，就會產生一個全新的物件，即使對這些物件的任何操作都會修改同一個 Off-Screen，但是為了不浪費記憶體，我們通常只取一次供往後使用。

當我們繪製好 Off-Screen 之後，就會叫用 flushGraphics()幫我們把 Off-Screen 繪製到螢幕上，flushGraphics()會等到 Off-Screen 真的在螢幕上畫出了才會返回。這個功能相當於過去叫用 repaint()再叫用 serviceRepaints()的功能，而且還帶有雙緩衝區的概念，不過 **flushGraphics()並不會產生重繪事件，而是直接將 Off-Screen 的內容顯示到螢幕上**，所以叫用 flushGraphics()時，並不會叫用 paint()方法。

flushGraphics()預設是將整個 Off-Screen 都重新在螢幕上畫出來，如果我們只希望 Off-Screen 某些部分重畫，那麼就叫用另外一個具有四個參數的 flushGraphics()，只要給定起始 X 座標、Y 座標、寬度、長度即可。

Copyright 2003 Sun Microsystems, Inc. All Rights Reserved. Enterprise Services

抓取鍵盤狀態

有別於過去等待鍵盤事件發生再更新狀態的做法，在 GameCanvas 之中，改由程式主動查詢按鍵的狀態，查詢的方法就是利用 getKeyStates()，使用方法如下:

MyGameWithInputCanvas.java

```
import javax.microedition.lcdui.* ;
import javax.microedition.lcdui.game.* ;

public class MyGameWithInputCanvas extends GameCanvas
implements Runnable
{
	public MyGameWithInputCanvas()
	{
		super(true) ;
	}
	boolean conti = true ;
	int rate = 50 ;
	public void run()
	{
		long st = 0 ;
		long et = 0 ;
		Graphics g = getGraphics() ;
		while(conti)
		{
			st = System.currentTimeMillis() ;
			input() ;
			render(g) ;
			et = System.currentTimeMillis() ;
			if((et-st)<rate)
			{
				try
				{
					Thread.sleep(rate-(et-st));
				}catch(Exception exp){}
			}
```

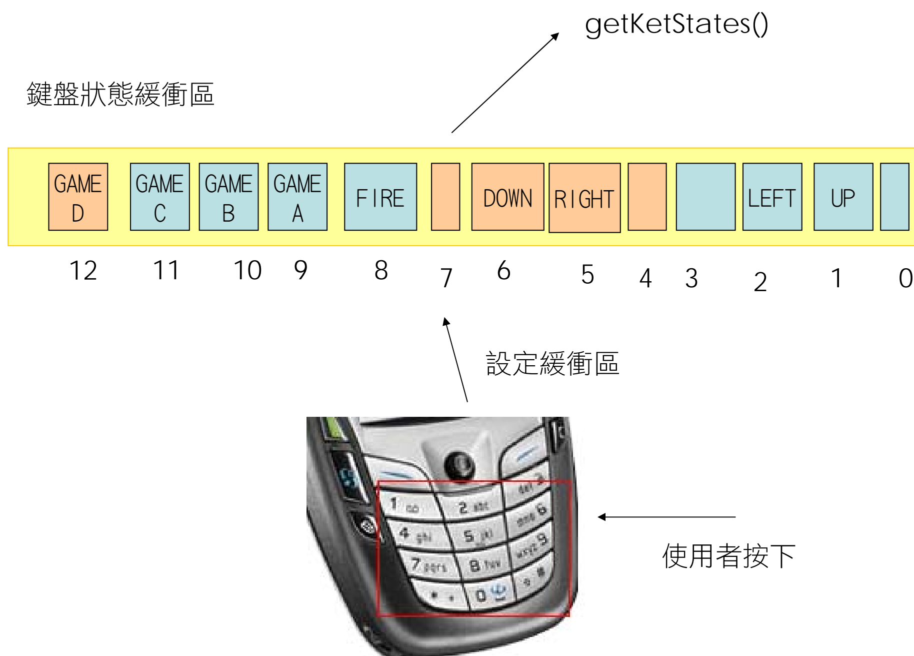

Copyright 2003 Sun Microsystems, Inc. All Rights Reserved. Enterprise Services

```
        }
    }
    int x = 50 ;
    int y = 10 ;
    public void input()
    {
        int keystate = getKeyStates() ;
        if((keystate & UP_PRESSED)!=0)
        {
            y = y - 2 ;
        }else if((keystate & DOWN_PRESSED)!=0)
        {
            y = y + 2 ;
        }

    }
    public void render(Graphics g)
    {
        g.setColor(255,255,255);
        g.fillRect(0,0,getWidth(),getHeight());
        g.setColor(0,0,0);
        g.drawString("Test",x,y,0);
        x++ ;
        if(x > getWidth())
            x = 0 ;
        flushGraphics() ;
    }
    public void start()
    {
        Thread t = new Thread(this) ;
        t.start();
    }
    public void exit()
    {
        conti = false ;
    }
    public void keyPressed(int keycode)
    {
```


Copyright 2003 Sun Microsystems, Inc. All Rights Reserved. Enterprise Services

```
                System.out.println("按鍵事件");
        }
}
```

鍵盤狀態的概念如下:



只要在程式叫用 getKeyStates()之前，該按鈕曾經被按下過，那麼緩衝區中的對應位元就會被設成 1；如果叫用 getKeyStates()之前，該按鈕不曾按下過，那麼緩衝區中的對應位元就會被設成 0。這樣的設計是為了讓程式不漏掉使用者曾經按下過按鈕(如果不這樣設計，改成讓 getKeyStates()反應當時按鍵的情況，那麼很可能兩者時間不同時就會漏掉而沒處理到)。當 getKeyStates()叫用之後，就會清除鍵盤狀態緩衝區的內容，所以理論上，連續叫用兩次 getKeyStates()，第二個 getKeyStates()就能反應當時按鈕被按下的情況。取出鍵盤狀態之後，利用位元運算就可以得知某個按鈕是否被按下了 GameCanvas 之中有定義的狀態為 UP_PRESSED、DWON_PRESSED、LEFT_PRESSED、RIGHT_PRESSED、FIRE_PRESSED、GAME_A_PRESSED、GAME_B_PRESSED、GAME_C_PRESSED、GAME_D_PRESSED。

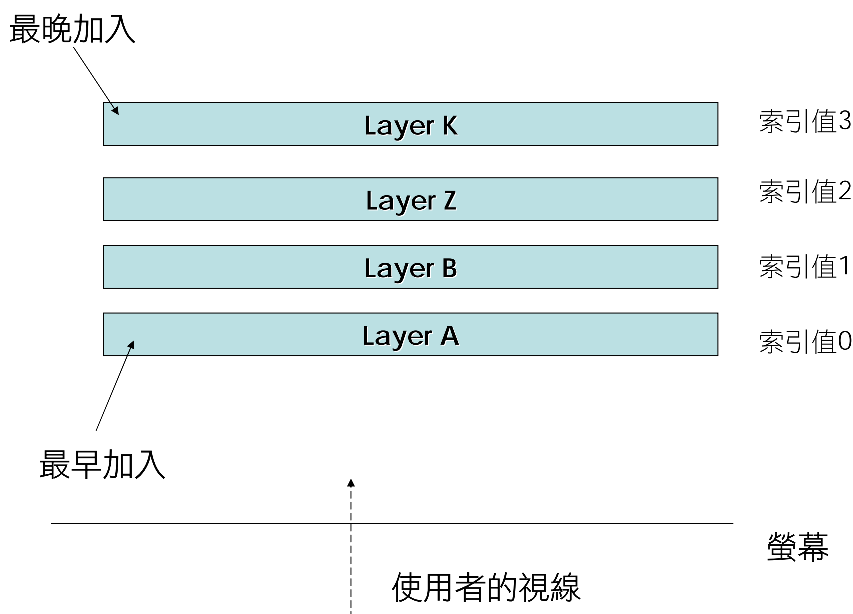
這樣的設計，除了讓我們能夠在同一個執行緒之中自己偵測按鍵的狀態之外，過去的 keyPressed() / keyReleased() / keyRepeated()同時間只能偵測到一個按鈕被按下，而在某些裝置下，getKeyStates()則可以偵測到很多按鈕同時間被按下的情形。

注意
有些裝置的 getKeyStates()是靠 keyPressed()/keyReleased()完成，那麼就不符合上述關於 getKeyStates()的說明

Copyright 2003 Sun Microsystems, Inc. All Rights Reserved. Enterprise Services

LayerManager 與 Layer 的關係

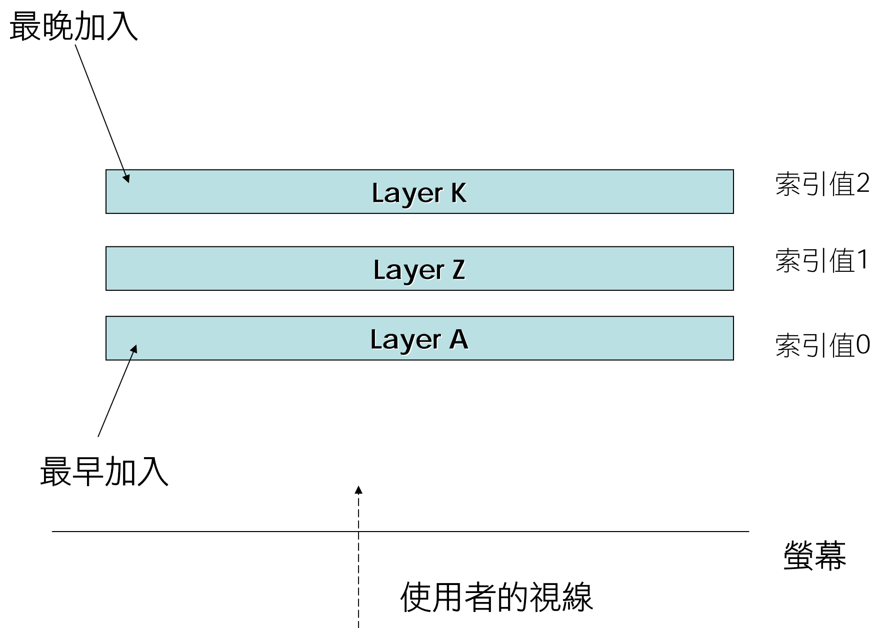
所有的 Spite 都該用 LayerManager 的 append 加入 LayerManager 之中，好讓 LayerManager 可以統一控管這些 Layer。每個 Layer 都會有個索引值，索引值從 0 開始，索引值越小代表離使用者越近。越晚加入的 Layer 會有越大的索引值:



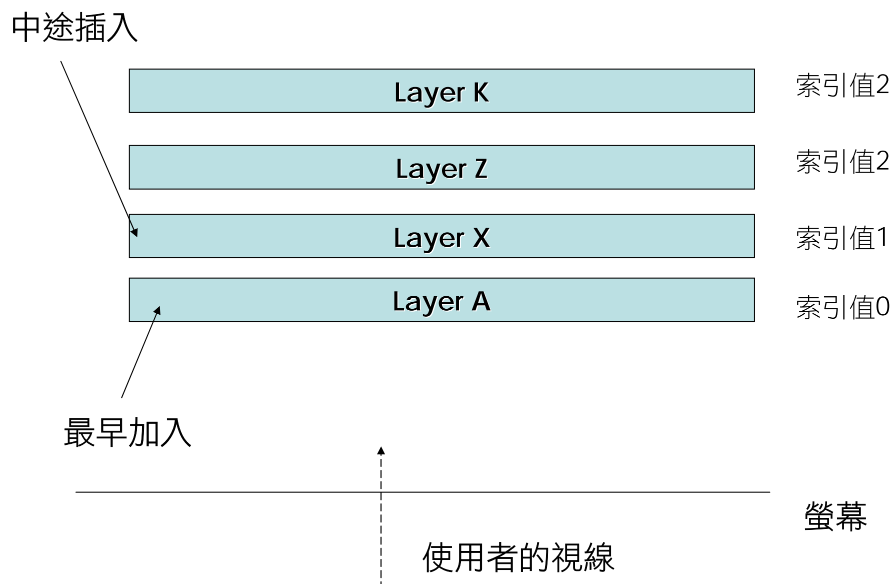
如果在加入時該 Layer 早已在 LayerManager 之中，那麼該 Layer 會先從 LayerManager 刪除，然後再行加入。

Copyright 2003 Sun Microsystems, Inc. All Rights Reserved. Enterprise Services

我們隨時可以利用 remove()將某個 layer 移出 LayerManager 之中，索引值會重新設定，舉例來説，我們將 Layer B 移出 LayerManager 之後，LayerManager 的內容如下:



我們也可以使用 insert 在任意位置插入 Layer，假設我們在索引值 1 的地方插入一個 Layer X，結果如下:



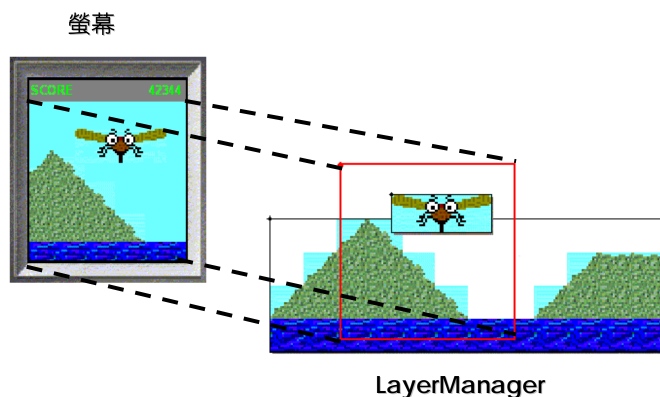
Copyright 2003 Sun Microsystems, Inc. All Rights Reserved. Enterprise Services

如果在插入時該 Layer 早已在 LayerManager 之中，那麼該 Layer 會先從 LayerManager 刪除，然後再行插入。

任何時候我們都可以利用 getLayer()取得某個索引值的 Layer。getSize()可以得知目前 LayerManager 裡頭到底有多少 Layer。

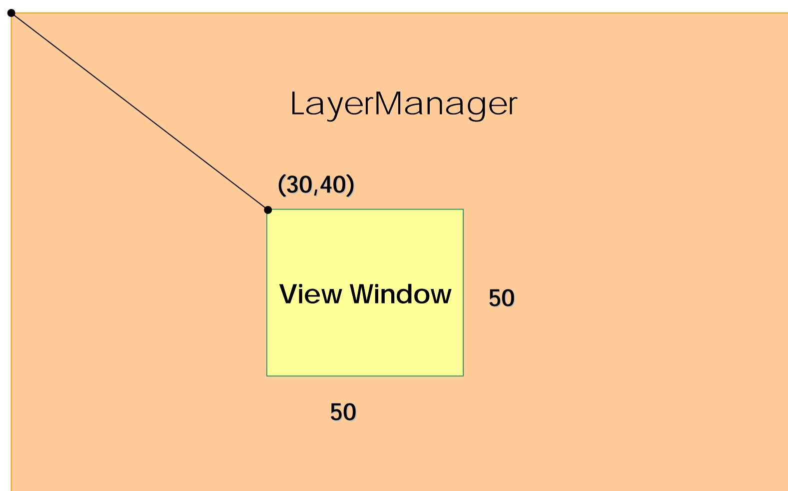
LayerManager 本身內部有一個座標系，此座標系與外界毫無關係，只要提到和 Layer 位置相關的座標，指的都是這個屬於 LayerManager 內部的座標系統。所以把 LayerManager 本身當作一個虛擬的螢幕來思考，不管外界的干擾。

但是，一但 LayerManager 要繪製到螢幕上時，就必須動用到一個名為 View Window 的概念。所謂 View Window，代表使用者所能看到的部分，也就是 LayerManager 願意讓螢幕顯示出來的部分:



我們可以使用 setViewWindow()來設定 View Window 的大小。假設我們叫用 mm.setViewWindow(30,40,50,50)
那麼結果如下圖所示:

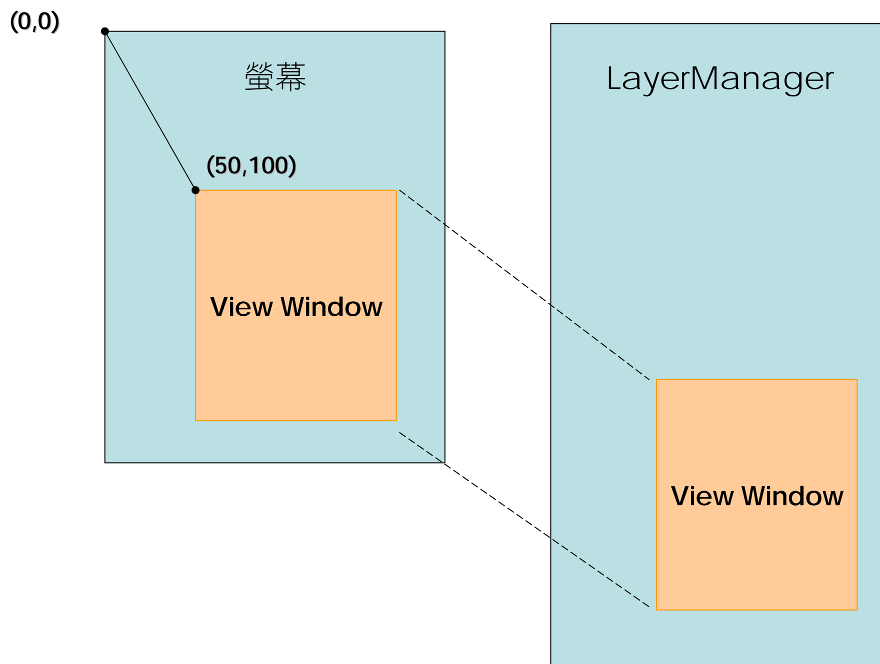
Copyright 2003 Sun Microsystems, Inc. All Rights Reserved. Enterprise Services



這裡所指定的 30,40，指的是相對於 LayerManager 座標系的座標。預設情況下，LayerManager 的 View Window 會設定為
(0,0,**Integer.MAX_VALUE**, **Integer.MAX_VALUE**)
，也就是 LayerManager 座標系的座標的原點開始，然後涵蓋整個 LayerManager 所佔據的位置。

至於 View Window 從螢幕的哪裡畫起，則由 LayerManager 的 paint()方法來決定。舉例來說，如果有個名為 mm 的 LayerManager，當我們叫用
mm.paint(g,50,100)
時，代表把 View Window 的內容畫在螢幕上 50,100 的位置。

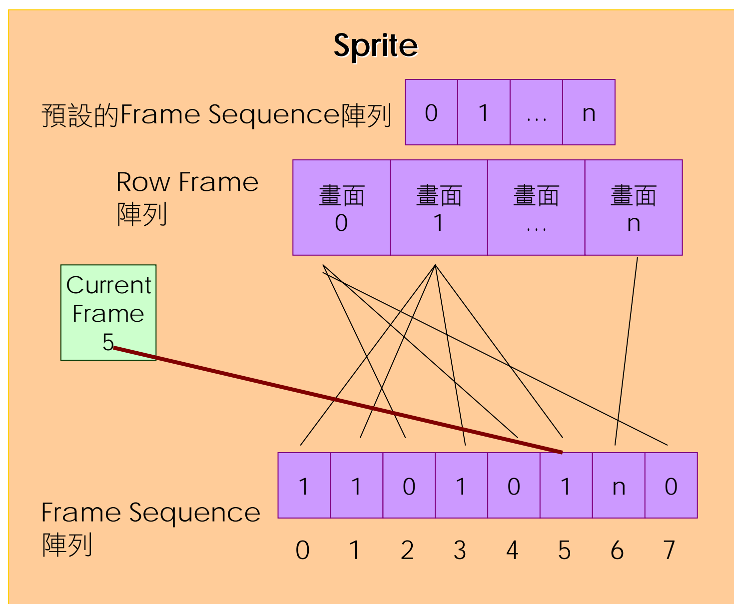
Copyright 2003 Sun Microsystems, Inc. All Rights Reserved. Enterprise Services



Copyright 2003 Sun Microsystems, Inc. All Rights Reserved. Enterprise Services

Sprite 的結構

所謂的 Sprite，就是畫面上獨立移動的圖形。Game API 提供我們 Sprite 類別以方便地建立 Sprite。Sprite 的內部結構如下:



也就是說，Sprite 類別會根據讀入的影像先建立一個 Row Frame 陣列，另外一個 Frame Sequence 陣列的內容都是畫面的索引值。Current Frame 指的是目前螢幕上顯示的畫面。**要用到 Frame Sequence 時，會先看看使用者有沒有自行定義 Frame Sequence 陣列，如果有就用使用者定義的，否則就採用預設的 Frame Sequence 陣列**。

當 Sprite 要繪製時，繪圖函式先依據 Current Frame 紀錄的索引值，找到 Frame Sequence 在該索引值的內容，以上述範例來說，Current Frame 內容為 5，所以會找到 Frame Sequence[5]， Frame Sequence[5]的內容為 1，所以最後繪出的畫面為 Row Frame[1]的內容。

一開始產生 Sprite 時，Current Frame 預設是指向 Frame Sequence 的第一個元素(索引值 0)。任何時候，我們都可以利用 Sprite 的 setImage()重設 Sprite 內部的 Raw Frame 陣列和預設的 Frame Sequence 陣列。如果新的 Raw Frame 陣列比舊的小(也就是畫面叫少)，那麼使用者自己設定的 Frame Sequence 就會被捨棄不用，而且 Current Frame 將再度指向 Frame Sequence 的第一個元素(索引值 0)。

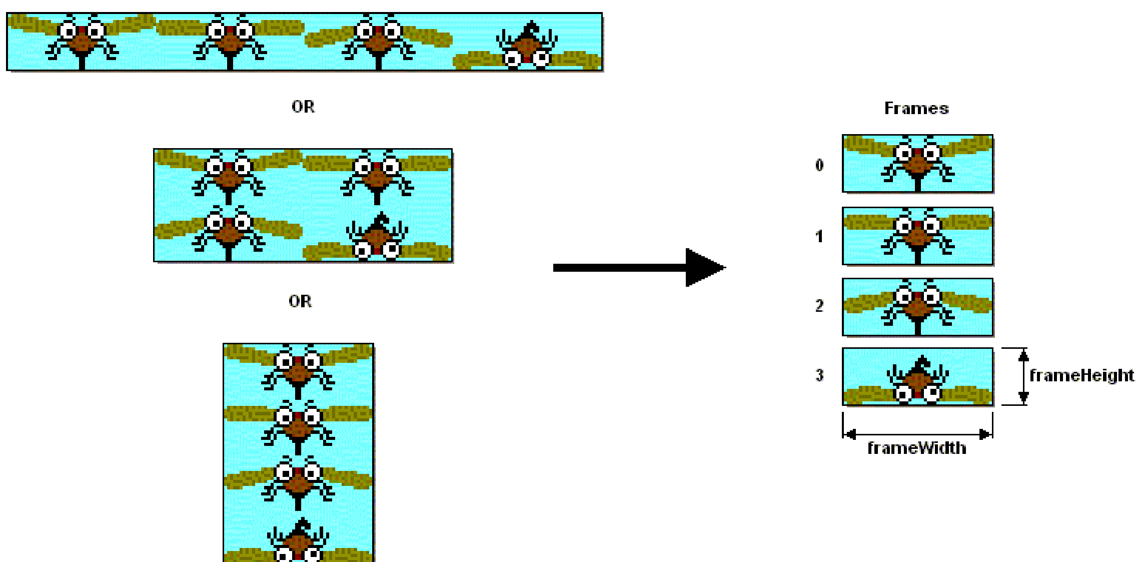
Copyright 2003 Sun Microsystems, Inc. All Rights Reserved. Enterprise Services

使用 Sprite

我們可以先利用 Image 將圖片載入進來，然後利用 Sprite 的建構式建立 Sprite，Sprite 的建構式只要給定 Image 物件，並告知每個圖片的大小，那麼我們的圖片會自動被切割成等尺寸的連續畫面(Frame)。如果只有給定 Image 物件，沒有給圖片大小，那麼整張圖片都會被當作是一個畫面。Sprite 有個建構是可以直接接受另外一個 Sprite 物件，產生一個全新的 Sprite。

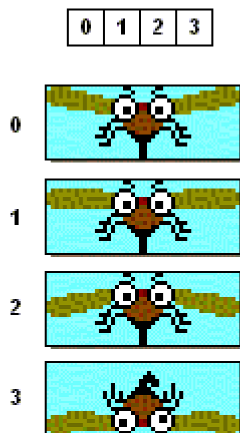
```
private Sprite createCharacter(String pic)
{
        Image img = null ;
        try
        {
                img = Image.createImage(pic);
        }catch(Exception exp)
        {
                System.out.println(exp);
        }
        return new Sprite(img,64,64)   ;
}
```

根據 MIDP 規格，切割的方式為由上到下，由左到右



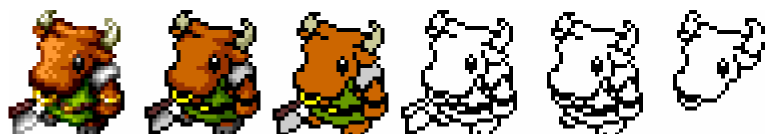
每個圖片被分割成畫面之後，就會被編號，從 0 開始編號

Copyright 2003 Sun Microsystems, Inc. All Rights Reserved. Enterprise Services

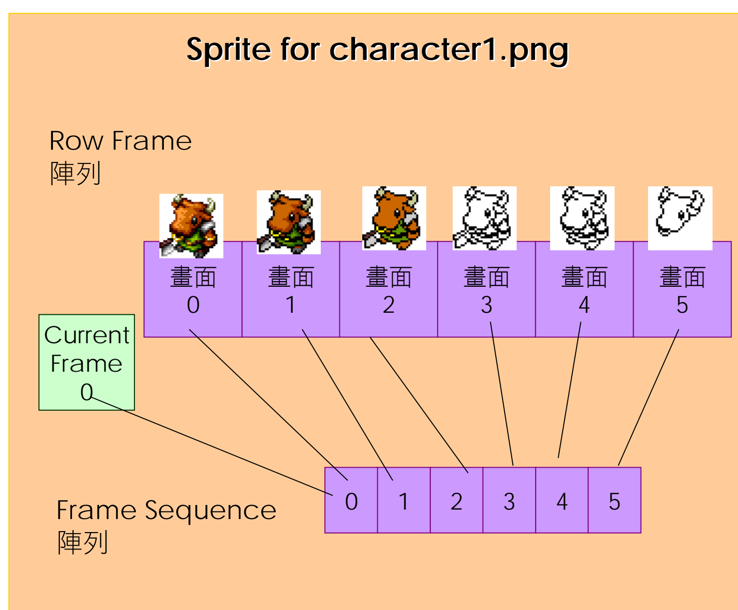


假設有兩張圖:

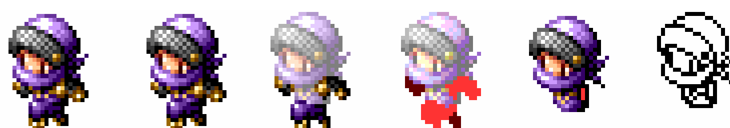
character1.png:



這裡有連續六張圖，每張圖的大小為 64 x 64 個像素，作成 Sprite 就會產生底下結構

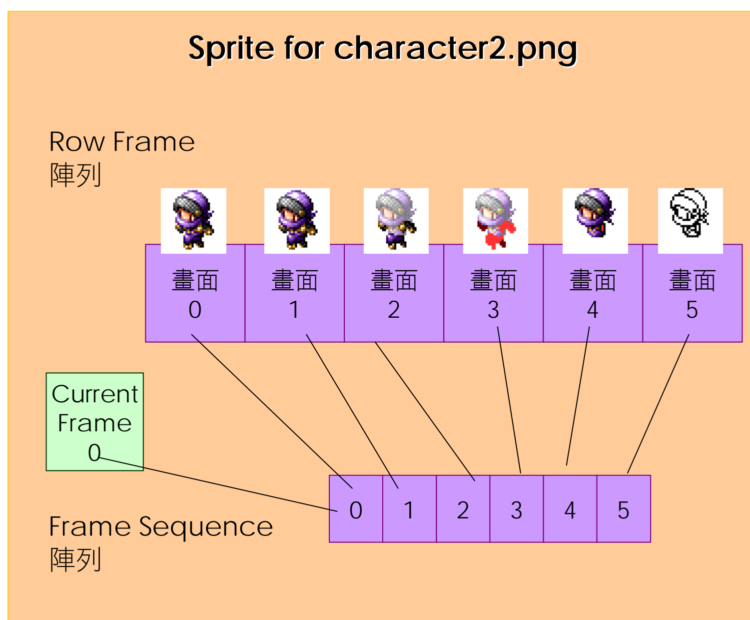


character2.png



這裡有連續六張圖，每張圖的大小為 64 x 64 個像素，作成 Sprite 就會產生底下結構

Copyright 2003 Sun Microsystems, Inc. All Rights Reserved. Enterprise Services



一旦建立 Sprite 之後，可以用 getRawFrameCount()取得 Raw Frame 的數量，或者用 getFrameSequence()取得 Frame Sequence()的數量。getFrame()可以取得目前 Current Frame 指向 Frame Sequence 陣列哪一個元素。setFrame()可以設定 Frame Sequence 某個元素的內容。我們可以叫用 nextFrame()/prevFrame()讓 Current Frame 指向 Frame Sequence 的下一個元素/上一個元素。

Sprite 的使用範例如下:

MyGameWithSpriteCanvas.java

```
import javax.microedition.lcdui.* ;
import javax.microedition.lcdui.game.* ;

public class MyGameWithSpriteCanvas extends GameCanvas
implements Runnable
{
        private LayerManager lm;
        private Sprite c1 ;
        public MyGameWithSpriteCanvas()
        {
                super(true) ;
                lm = new LayerManager();
                c1 = createCharacter("/pic/character1.png") ;
                lm.append(c1);
        }
```


Copyright 2003 Sun Microsystems, Inc. All Rights Reserved. Enterprise Services

```
private Sprite createCharacter(String pic)
{
    Image img = null ;
    try
    {
        img = Image.createImage(pic);
    }catch(Exception exp)
    {
        System.out.println(exp);
    }
    return new Sprite(img,64,64)  ;
}
boolean conti = true ;
int rate = 100 ;
public void run()
{
    long st = 0 ;
    long et = 0 ;
    Graphics g = getGraphics() ;
    while(conti)
    {
        st = System.currentTimeMillis() ;
        input() ;
        render(g) ;
        et = System.currentTimeMillis() ;
        if((et-st)<rate)
        {
            try
            {
                Thread.sleep(rate-(et-st));
            }catch(Exception exp){}
        }
    }
}
int x = 10 ;
int y = 10 ;
public void input()
{
```


Copyright 2003 Sun Microsystems, Inc. All Rights Reserved. Enterprise Services

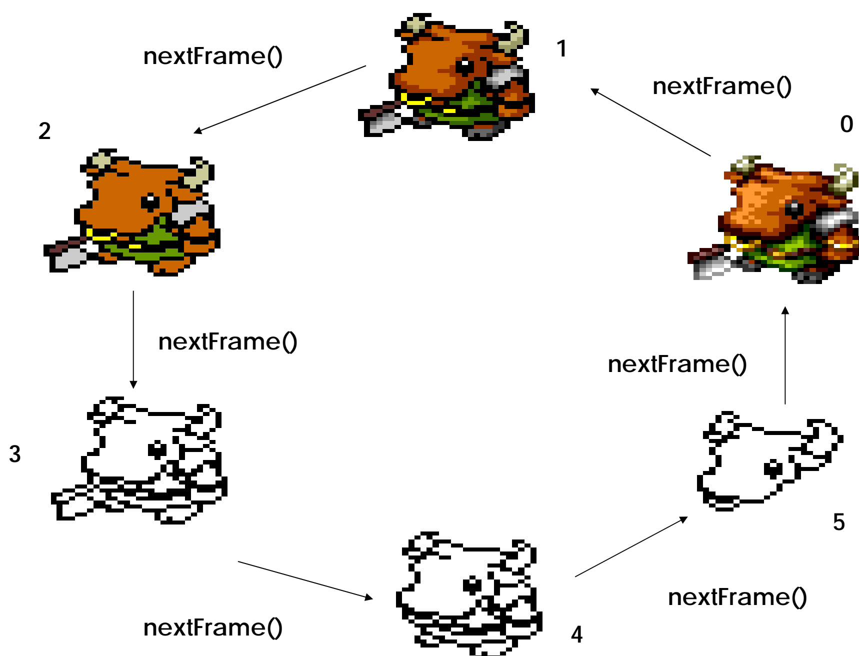
```
            int keystate = getKeyStates() ;
            if((keystate & UP_PRESSED)!=0)
            {
                    y = y - 2 ;
            }else if((keystate & DOWN_PRESSED)!=0)
            {
                    y = y + 2 ;
            }else if((keystate & LEFT_PRESSED)!=0)
            {
                    x = x - 2 ;
            }else if((keystate & RIGHT_PRESSED)!=0)
            {
                    x = x + 2 ;
            }

    }
    public void render(Graphics g)
    {
            g.setColor(255,255,255);
            g.fillRect(0,0,getWidth(),getHeight());

            c1.nextFrame();
            lm.setViewWindow(0,0,x,y);
            lm.paint(g,10,10);

            g.setColor(0,0,0);
            g.drawRect(10,10,128,128);
            flushGraphics() ;
    }
    public void start()
    {
            Thread t = new Thread(this) ;
            t.start();
    }
    public void exit()
    {
            conti = false ;
    }
```


Copyright 2003 Sun Microsystems, Inc. All Rights Reserved. Enterprise Services

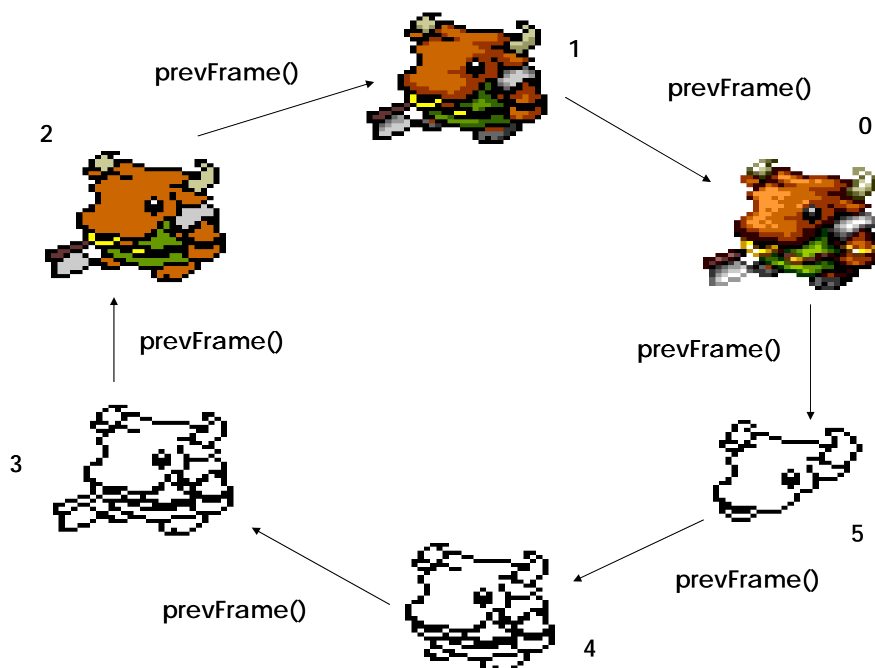
```
}
```

執行結果:

當 Current Frame 指向 Frame Sequence 的最開頭元素時，如果再叫用 prevFrame()，那麼 Current Frame 就會指向 Frame Sequence 的最後一個元素。反過來說，當 Current Frame 指向 Frame Sequence 的最後一個元素時，如果再叫用 nextFrame()，那麼 Current Frame 就會指向 Frame Sequence 的第一個元素。，如下圖所示:



Copyright 2003 Sun Microsystems, Inc. All Rights Reserved. Enterprise Services



如果我們想要自己定義畫面的顯示順序，可以利用 setFrameSequence()設定 Frame Sequence 的內容，只要傳入一個以畫面編號為內容的陣列即可，範例如下:

MyGameWithSpriteCanvas1.java

```
import javax.microedition.lcdui.* ;
import javax.microedition.lcdui.game.* ;

public class MyGameWithSpriteCanvas1 extends GameCanvas
implements Runnable
{
        private LayerManager lm;
        private Sprite c1 ;
        public MyGameWithSpriteCanvas1()
        {
                super(true) ;
                lm = new LayerManager();
                c1 = createCharacter("/pic/character1.png") ;
                int seq[] = new int[]{ 5,5,5,5,0,0,0,0} ;
                c1.setFrameSequence(seq);
                lm.append(c1);
        }
        private Sprite createCharacter(String pic)
        {
```


Copyright 2003 Sun Microsystems, Inc. All Rights Reserved. Enterprise Services

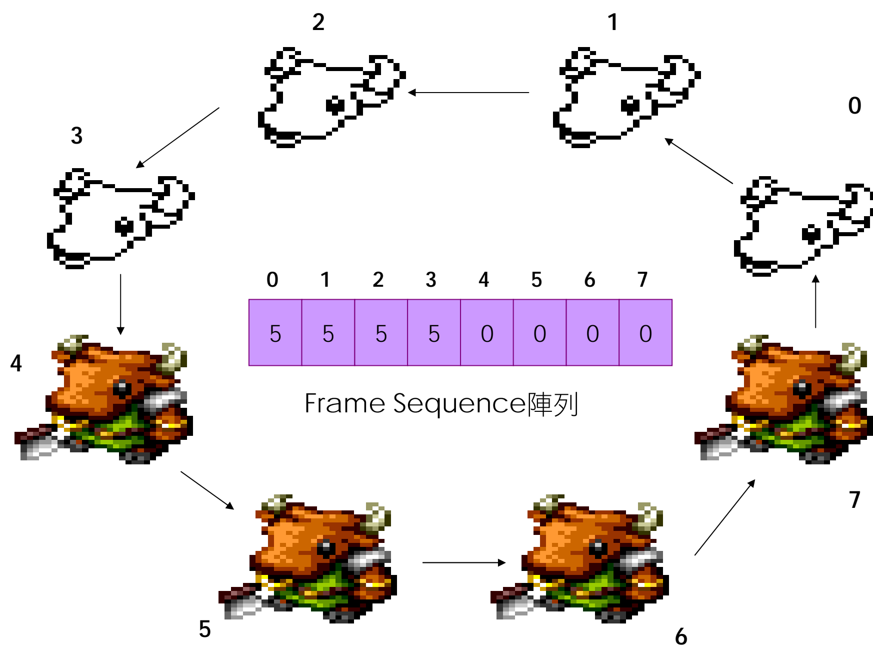
```
		Image img = null ;
		try
		{
			img = Image.createImage(pic);
		}catch(Exception exp)
		{
			System.out.println(exp);
		}
		return new Sprite(img,64,64)	;
	}
	boolean conti = true ;
	int rate = 100 ;
	public void run()
	{
		long st = 0 ;
		long et = 0 ;
		Graphics g = getGraphics() ;
		while(conti)
		{
			st = System.currentTimeMillis() ;
			input() ;
			render(g) ;
			et = System.currentTimeMillis() ;
			if((et-st)<rate)
			{
				try
				{
					Thread.sleep(rate-(et-st));
				}catch(Exception exp){}
			}
		}
	}
	int x = 10 ;
	int y = 10 ;
	public void input()
	{
		int keystate = getKeyStates() ;
		if((keystate & UP_PRESSED)!=0)
```


Copyright 2003 Sun Microsystems, Inc. All Rights Reserved. Enterprise Services

```
                {
                        y = y - 2 ;
                }else if((keystate & DOWN_PRESSED)!=0)
                {
                        y = y + 2 ;
                }else if((keystate & LEFT_PRESSED)!=0)
                {
                        x = x - 2 ;
                }else if((keystate & RIGHT_PRESSED)!=0)
                {
                        x = x + 2 ;
                }

        }
        public void render(Graphics g)
        {
                g.setColor(255,255,255);
                g.fillRect(0,0,getWidth(),getHeight());

                c1.nextFrame();
                lm.setViewWindow(0,0,x,y);
                lm.paint(g,10,10);

                g.setColor(0,0,0);
                g.drawRect(10,10,128,128);
                flushGraphics() ;
        }
        public void start()
        {
                Thread t = new Thread(this) ;
                t.start();
        }
        public void exit()
        {
                conti = false ;
        }
}
```

在範例程式中，我們把畫面的順序改成 5,5,5,5,0,0,0,0，如下圖所示:

Copyright 2003 Sun Microsystems, Inc. All Rights Reserved. Enterprise Services



Copyright 2003 Sun Microsystems, Inc. All Rights Reserved. Enterprise Services

LayerManager 與 Layer 的互動

當我們產生 Layer 之後，隨時可以用 getWidth()/getHeight()取得這個 Layer 的寬度和高度。

預設的情況下，Layer 的繪製起點(即 Layer 左上角的點)為相對於 LayerManager 起點的(0,0)位置，預設的 paint()方法會繪出整個 Layer。我們可以利用 layer 的 getX()和 getY()來取得繪製起點的 X 座標及 Y 座標。

LayerManager 會從索引值最大的 Layer 開始畫，最後畫到索引值最小的 Layer。所以索引值越小的 Layer 會覆蓋索引值較大的 Layer。範例程式如下:

MyGameWithSpriteCanvas2.java

```
import javax.microedition.lcdui.* ;
import javax.microedition.lcdui.game.* ;

public class MyGameWithSpriteCanvas2 extends GameCanvas
implements Runnable
{
        private LayerManager lm;
        private Sprite c1 ;
        private Sprite c2 ;
        public MyGameWithSpriteCanvas2()
        {
                super(true) ;
                lm = new LayerManager();
                c1 = createCharacter("/pic/character1.png") ;
                lm.append(c1);
                System.out.println("c1 層的高度"+c1.getHeight());
                System.out.println("c1 層的寬度"+c1.getWidth());
                c2 = createCharacter("/pic/character2.png") ;
                c2.setPosition(x,y);
                System.out.println("c2 層的高度"+c2.getHeight());
                System.out.println("c2 層的寬度"+c2.getWidth());
                lm.append(c2);
        }
        private Sprite createCharacter(String pic)
        {
```


Copyright 2003 Sun Microsystems, Inc. All Rights Reserved. Enterprise Services

```
            Image img = null ;
            try
            {
                        img = Image.createImage(pic);
            }catch(Exception exp)
            {
                        System.out.println(exp);
            }
            return new Sprite(img,64,64)    ;
}
boolean conti = true ;
int rate = 100 ;
public void run()
{
            long st = 0 ;
            long et = 0 ;
            Graphics g = getGraphics() ;
            while(conti)
            {
                        st = System.currentTimeMillis() ;
                        input() ;
                        render(g) ;
                        et = System.currentTimeMillis() ;
                        if((et-st)<rate)
                        {
                                    try
                                    {
                                                Thread.sleep(rate-(et-st));
                                    }catch(Exception exp){}
                        }
            }
}
int x = 32 ;
int y = 32 ;
public void input()
{
            int keystate = getKeyStates() ;
            if((keystate & UP_PRESSED)!=0)
```

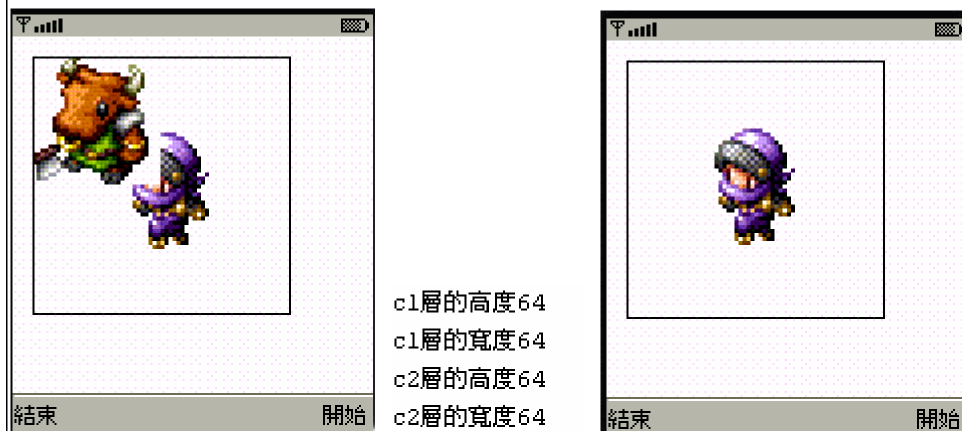

Copyright 2003 Sun Microsystems, Inc. All Rights Reserved. Enterprise Services

```
        {
                c2.move(0,-2);
        }else if((keystate & DOWN_PRESSED)!=0)
        {
                c2.move(0,2);
        }else if((keystate & LEFT_PRESSED)!=0)
        {
                c2.move(-2,0);
        }else if((keystate & RIGHT_PRESSED)!=0)
        {
                c2.move(2,0);
        }else if((keystate & FIRE_PRESSED)!=0)
        {
                if(c1.isVisible())
                {
                        c1.setVisible(false);
                }else
                {
                        c1.setVisible(true);
                }
        }


}
public void render(Graphics g)
{
        g.setColor(255,255,255);
        g.fillRect(0,0,getWidth(),getHeight());

        lm.paint(g,10,10);

        g.setColor(0,0,0);
        g.drawRect(10,10,128,128);
        flushGraphics() ;
}
public void start()
{
        Thread t = new Thread(this) ;
```


Copyright 2003 Sun Microsystems, Inc. All Rights Reserved. Enterprise Services

```
                t.start();
        }
        public void exit()
        {
                conti = false ;
        }
}
```

執行結果:



我們可以叫用 setPosition()來設定繪製起點，或者用 move()來移動繪製起始點。我們也可以利用 setVisible()來決定某個 Layer 是否要繪製在螢幕上。isVisible()可以讓我們知道目前的 Layer 是否能夠呈現在螢幕上。

Copyright 2003 Sun Microsystems, Inc. All Rights Reserved. Enterprise Services

結語

本章為各位介紹了 MIDP 2.0 之中新增的 Game API

一開始先介紹了 Game API 的結構體系，接著介紹了最重要的 GameCanvas 的用法和概念。最後講解了 LayerManager 與 Layer 之間的關係和互動方式。

本章使用 Sprite 作為第一個講解的 Layer 子類別，關於其進階用法與 TiledLayer 的用法，將在下一章說明。

Copyright 2003 Sun Microsystems, Inc. All Rights Reserved. Enterprise Services