

Sprites

Download Wallpapers

MoboGenie.com/Download-Wallpapers

Use Mobogenie to Download Wallpaper Save Data Cost. Download Now!



[Back](#) | [Tutorial Home](#) | [Next](#)

In any successful game one of the contributing factors are excellent graphics. Most objects in a game are categorized as a special kind of graphic called sprites. A sprite can be anything from a bullet, monster, the main character, enemies, special power items, keys and doors to name a few.

Example of a Sprite



A lot times sprites are animated graphics. These animated graphics are made up of several instances of the same sprite with each of them slightly different. These sets of sprites are usually referred to as a set of frames. These frames can be render on the screen sequentially or non sequentially. Typically sprites are displayed sequentially for ease of coding.

Figure Illustrates a A Sprite Set representing the wait frame and 4 basic other frames.



Sprite Constructor

There are 3 constructors that come with the Sprite class

- Sprite (Image image) – Creates single frame sprite, non-animated
- Sprite (Sprite sprite) – Creates a new Sprite from another Sprite
- Sprite (Image image, int frameWidth, int frameHeight) – Creates an animated sprite with 2 more frames, the frameWidth is the width of one sprite and the height is the height of one sprite

Sprites Set with Grid Overlay



Taking another look at Figure 6 we can break down the entire sprite set into the individual frames. In this particular example the total width is 160 pixels, which divided by 5 gives you a frame width of 32 pixels. The height for each frame is 32 pixels. The height and weight do not have to be the same, but the width and height must remain constant for all sprites in the same set. In other words you cannot have frame one with the width of 32 pixels and the remaining sprites have a width of 16 pixels; all frames must have the same width. In the constructor Sprite (Image, image, int frameWidth, int frameHeight) you will notice you do not have to specify the number of frames, this will be automatically calculated by the sprite class.

Why 8 Bit, 16 Bit, 32 Bit?

You will notice most graphics including sprites will usually fall under the same width and height constraints. This is because the number of pixels used are in correlation to the number colors used. This is referred to as bit depth or color depth. The more bits per

pixel the more colors there are.

The formula is

$2^{\text{\# of bits}} = \text{total colors}$

Using the formula the bit depths are 8,16,24, and 32 bits translate to

- $2^8 = 256$ colors
- $2^{16} = 65,536$ colors
- $2^{24} = 16.7$ million colors
- $2^{32} = 16.7$ million colors plus an 8-bit alpha channel, an alpha channel is really a mask, it specifies how a color of one pixel when merged with another pixel should be. Merging occurs when one pixel is on top of another pixel.

However, you can have Sprites bigger then 8 by 8 pixels and only with 256 colors. Ideally this is probably your best option when dealing with graphics for mobile handsets. The more colors there are the more processing is required to render the graphics.

Sprite Collision

Sprite Collision is one of the most essential functions of any interactive action arcade like game. This can be anything from a sprite running into wall, a sprite firing a bullet or two sprites running into each other. Collision detection doesn't necessary means death or game over. It can mean unlimited amount of possibilities such as level ups, power ups, loss of traction in a racing game, opening a door, or indicator allowing the player to climb the ladder.

So how do we detect sprite collision? Naturally we should detect if the pixel of one Sprite overlaps/collides with another pixel of another Sprite.



Sprite Collision

Fortunately, the Sprite class comes with a method that does just that.

`collidesWith(Image image, int x, int y, boolean pixelLevel)`

or

`collidesWith(Sprite sprite, boolean pixelLevel)`

As you can see you can detect collision with another sprite or an image. The method with image as an input you need to specify the location of the image, this is referring to x and y of the image's top left upper corner. The pixelLevel is a boolean value where true indicates pixel level detection and false indicates rectangle intersects. You can define the size of the rectangle intersects you may want to consider defining the rectangle intersects to be slightly smaller then the image itself. This then eliminates the odd areas of the sprites where collision occurs because the invisible rectangles collide but there are no opaque pixels that have collided.

Pixel-Level detection is when the opaque pixel of one sprite overlaps another sprites opaque pixel. A more simple method of collision detection is the collision of rectangle intersects of the two sprites usually the rectangles are the size of the images' bounds.

There is a third method, `collidesWidth(TiledLayer tiledLayer, Boolean pixelLevel)`

This is similar to the last two methods except the collision is checked against a graphic tile layer. TiledLayers will be explained in more detail in the next section.

Display Sprite

To display or render the sprite simply call the paint method, you will need to pass in the Graphics object as it is the required parameter. Reminder you may have to use the setVisible(boolean) method first calling the paint method, you will need to pass in the Boolean value true.

Display Sprite Sequence

There are few methods available when dealing with sprite frame sequence.

- getFrameSequenceLength() – returns the number of elements in a frame sequence
- getFrame() – retrieves the current entry index number in the frame sequence, this is not the frame that was currently displayed
- nextFrame() – set frame sequence to the next frame, if the sequence is at the last frame it starts at first frame
- prevFrame() – set frame sequence to the previous frame, if the sequence is at the first frame it sets the frame to the last frame
- setFrame(int sequenceIndex) – to manually set the sequence in the frame sequence
- setFrameSequence(int[] sequence) – to manually preset a predefined frame sequence

See J2ME API for more details.

Sprite Transparency

A reminder you need to beware of transparent and non-transparent images depending on the situation. If it is a simple game with little animation such as TicTacToe transparent images may not be important or required. In highly interactive games where there is a lot potential for sprite collision and/or varying background images you may want to consider using transparent images. As well you should check with the mobile handset manufacturer and ensure that the mobile handset you will be deploying to does indeed support transparent images if you do decide to use transparent images.

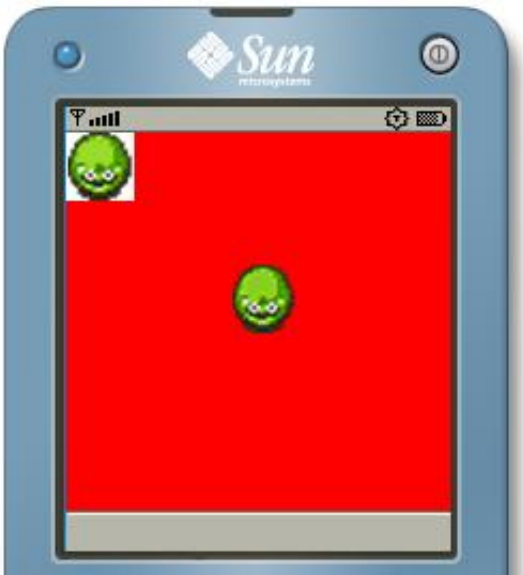










Figure: Non Transparent Sprite Vs Transparent Sprite.

Sprite Transforms

There are methods that manipulate that can manipulate a Sprite in rotations of 90 degrees and mirrored versions of the

rotations. The transform is set by invoking the `setTransform(transform)` method. The parameter is actually an integer value; however, there are valid predefined static values available for your usage:

Static Fields	Integer Value	Transformed Image
TRANS_NONE	0	
TRANS_MIRROR_ROT180	1	
TRANS_MIRROR	2	
TRANS_ROT180	3	
TRANS_MIRROR_ROT270	4	
TRANS_ROT90	5	
TRANS_ROT270	6	
TRANS_MIRROR_ROT90	7	

When transforms are applied the sprite is automatically repositioned so that it remains in the center. Therefore the reference point for the sprite does not change, but the values returned from `getX()` and `getY()` will change according to the current position of the upper-left corner of the sprite.

Sprite Optimization

As reminder you should verify what colors and resolution is supported on the mobile handsets you are going to deploy to as well all production graphics should be optimized. Image optimization is usually the process of removing colors the human eye cannot see. The benefit of optimization is of course reduction in image size, which means over all reduction of the jar file.

Basic Sprite Example

The following is a simple sprite example, which builds on the last example from the `GameCanvas` section; you will notice you can only move the center sprite the sprite in the corner is there demonstrate what happens if transparency is not used with the image.

Main Game Canvas with Sprites

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

public class ExampleGameCanvas extends GameCanvas implements Runnable {
    private boolean isPlay; // Game Loop runs when isPlay is true
    private long delay;     // To give thread consistency
    private int currentX, currentY; // To hold current position of the 'X'
    private int width;      // To hold screen width
    private int height;     // To hold screen height

    // Sprites to be used
    private Sprite sprite;
    private Sprite nonTransparentSprite;

    // Constructor and initialization
```

```

public ExampleGameCanvas() throws Exception {
    super(true);
    width = getWidth();
    height = getHeight();
    currentX = width / 2;
    currentY = height / 2;
    delay = 20;

    // Load Images to Sprites
    Image image = Image.createImage("/transparent.png");
    sprite = new Sprite (image,32,32);

    Image imageTemp = Image.createImage("/nontransparent.png");
    nonTransparentSprite = new Sprite (imageTemp,32,32);

}

// Automatically start thread for game loop
public void start() {
    isPlay = true;
    Thread t = new Thread(this);
    t.start();
}

public void stop() { isPlay = false; }

// Main Game Loop
public void run() {
    Graphics g = getGraphics();
    while (isPlay == true) {

        input();
        drawScreen(g);
        try { Thread.sleep(delay); }
        catch (InterruptedException ie) {}
    }
}

// Method to Handle User Inputs
private void input() {
    int keyStates = getKeyStates();

    sprite.setFrame(0);

    // Left
    if ((keyStates & LEFT_PRESSED) != 0) {
        currentX = Math.max(0, currentX - 1);
        sprite.setFrame(1);
    }

    // Right
    if ((keyStates & RIGHT_PRESSED) != 0) {
        if (currentX + 5 < width) {
            currentX = Math.min(width, currentX + 1);
            sprite.setFrame(3);
        }
    }
}

```

```

// Up
if ((keyStates & UP_PRESSED) != 0) {
    currentY = Math.max(0, currentY - 1);
    sprite.setFrame(2);
}

// Down
if ((keyStates & DOWN_PRESSED) != 0) {
    if (currentY + 10 < height) {
        currentY = Math.min(height, currentY + 1);
        sprite.setFrame(4);
    }
}

// Method to Display Graphics
private void drawScreen(Graphics g) {
    //g.setColor(0xffffffff);
    g.setColor(0xFF0000);
    g.fillRect(0, 0, getWidth(), getHeight());
    g.setColor(0x0000ff);

    // display sprites
    sprite.setPosition(currentX,currentY);
    sprite.paint(g);
    nonTransparentSprite.paint(g);

    flushGraphics();
}
}

```

Main Midlet

```

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

public class ExampleGameSpriteMidlet extends MIDlet {
    private Display display;

    public void startApp() {
        try {
            display = Display.getDisplay(this);
            ExampleGameCanvas gameCanvas = new ExampleGameCanvas();
            gameCanvas.start();
            display.setCurrent(gameCanvas);
        } catch (Exception ex) {
            System.out.println(ex);
        }
    }

    public Display getDisplay() {
        return display;
    }

    public void pauseApp() {

```

```

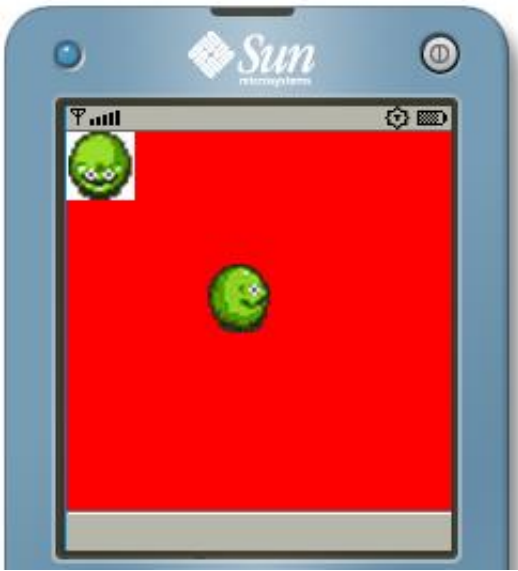
    }

    public void destroyApp(boolean unconditional) {
        exit();
    }

    public void exit() {
        System.gc();
        destroyApp(false);
        notifyDestroyed();
    }
}

```

Simple Sprite Example Emulator Screen Shot.



Extending the Sprite Class

There are few reasons why you might consider inheriting the Sprite class:

- Add addition Sprite functionality
- Use the Sprite class as your base Sprite class for all game sprites
- Encapsulating functionality

Referring to the first point, you may find that the current Sprite class supplied by sun may not have all the necessary features you need for example you might be making a race car game. In most racing games velocity is a factor, so you may want to inherit the Sprite class and add in the necessary variables and methods you need to use.

Another example where you may want to inherit the Sprite class is say for example you have several different types of sprites but each type has their own set of characteristics. For example, say the player sprite has attributes such as strength; agility, speed, intelligence and the enemy sprites have attributes indicating what kind of enemy it is and a life bar. Both sprites still share the same kinds of characteristics such as position and name. You can now create a base sprite class that contains the common features for all sprites, as well create child sprites that contain both their own characteristics and inherently still retain the common characteristics.

You probably have noticed in the Simple Sprite Example the only sprites movements are left, right, up and down. If you try to press 2 keys at once like right and up to produce an angled direction it simply defaults to one of the other basic directions. To produce a SW, SE, NW or NE movement you need to define both the proper sprites and the proper calculations. This is just a matter of some basic math manipulation, but more importantly where should this be done?

To keep your code a little more organized and clean you may consider encapsulating this work in a custom Sprite..

Making Your Own Sprite Class

If the Sprite class for whatever reason totally does not suit your needs you have the option to inherit the Layer class or completely start from scratch and implement your very own Sprite class, of course you will have to name it something else.

The following is an incomplete example of what Sprite class may look like if implemented from scratch, you can take this example and complete it to your customized requirements.

```
import java.microedition.lcdui.*;
public class MySprite {
    private Image image;
    private int positionX;
    private int positionY;
    private int frameWidth;
    private int frameHeight;
    private int numFrames;
    private int currentFrame;
    private boolean visible;
    public MySprite(Image image, int frameWidth, int frameHeight, int numFrames)
    throws Exception {
        this.image = image;
        this.frameWidth = frameWidth;
        this.frameHeight = frameHeight;
        this.numFrames = numFrames;
    }
    public int getX() {
        return this.positionX;
    }
    public void setX(int positionX) {
        this.positionX = positionX;
    }
    public int getY() {
        return this.positionY;
    }
    public void setY(int positionY) {
        this.positionY = position Y;
    }
    // Continue Your Code here
}
```

Where to Find Sprites

Well coding your game to handle sprites is one thing but producing or finding sprites is> another. Like most developers you probably do not have the skills to produce nice looking sprites.

Some well-known web links with sprites are:

- <http://www.arifeldman.com>
- <http://www.idevgames.com>
- <http://www.spriteworks.com>

Like anything else please read and abide by the terms of use before using any sprites you find for free.

Your other option is to simply hire professional sprite artists sometimes called pixel pushers. You may want to contract freelance graphic artists and find out their rates sometimes they will do it for free depending on what type the project is, especially if it is an open source project. You will need to do some investigation in where to find these types of freelancers. Just as developers hang out at developer sites, artists hang together in there own world.

Lastly you can always learn to make your own sprites. This may not be such a bad idea if you plan to make a simple 2D game and if you don't mind spending the time to learn more about sprite drawing. Granted if you do not have the artistic talent you are probably better off with the first 2 suggestions. But remember graphics are important but getting the game to work is more important. So in the beginning you may just want to use simple but not so pretty graphics until you have a fully function game. Then you can look at improving the graphics.

j2meSalsa goodies

Execute Sprite's Sample online

Execute on your browsers left frame.

Download demo code for deploying directly to WTK

[Click here](#) to download Zip File Containing WTK compatible file structure.

[Download 100,000+ Apps](#)

www.Mobogenie.asia

All-in-One Android Phone Manager. Free Download Now! (Windows Only)



[Back](#) | [Tutorial Home](#) | [Next](#)

2 comments



Leave a message...

Best ▾

Community

Share



Avatar

[earth38](#) • 3 years ago

Hi,

In the Main Game Canvas you used the following code, `public void stop() { isPlay = false; }` I removed this code and it runs fine.

I am puzzled why this is used, what does this line do.

Thanks

^ | ▾ Reply Share ›



[Tripperz_neon009](#) → [earth38](#) • 3 years ago

it is called to stop the game. your isplay in your run method will be in the boolean false.

^ | ▾ Reply Share ›

Subscribe

Add Disqus to your site

This page is a part of a frames based web site. If you have landed on this page from a search engine [click here](#) to view the complete page.

