Game development for Series 40: Main menu implementation

This article explains how to add Main menu in a typical 2D adventure or quest game skeleton. This is the next article after Game development for Series 40 : adding touch support in wiki article series Rapid game development for Series 40 with NetBeans Visual Designer and Game Builder. The previous article ends when application has some gameplay and supports both the touch-screen and keypad-only devices. Please refer to Rapid game development for Series 40 with NetBeans Visual Designer and Game Builder article if you want to go through the instructions from very beginning.

05 Aug 2012

Introduction

This article is a part of wiki article series "Game development for Series 40". Here we explain how to create Rapid game development for Series 40 with NetBeans Visual Designer and Game Builder. In the series some important challenges will be highlighted or elaborated to solution. The ultimate goal of the article series is to show game development full cycle from application skeleton to close to production condition.

Main menu implementation

Main menu screen. It usually contains the following options:

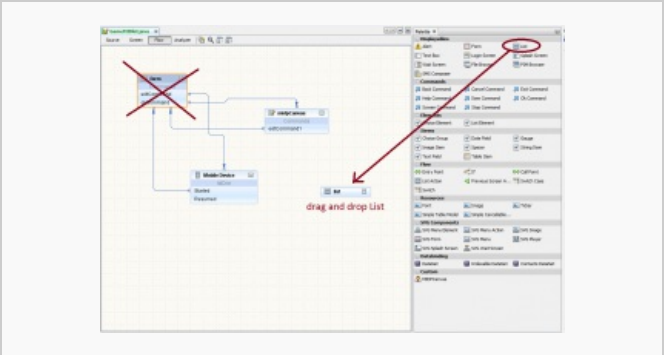
1. Start Game
2. High scores
3. Settings
4. About
5. Exit

The easiest way to create attractive main menu is to use Scalable 2D vector graphics. The problem is some devices does not support it. To check if SVG graphics is supported by device, please check the device specification for Scalable 2D Vector Graphics API for J2ME (JSR-226). We are going to use the Reflection API to detect device capability in run-time and enable or disable SVG main menu the same way as we filtered out touch screen functionality on keypad-only devices please check Game development for Series 40 : adding touch support for details.

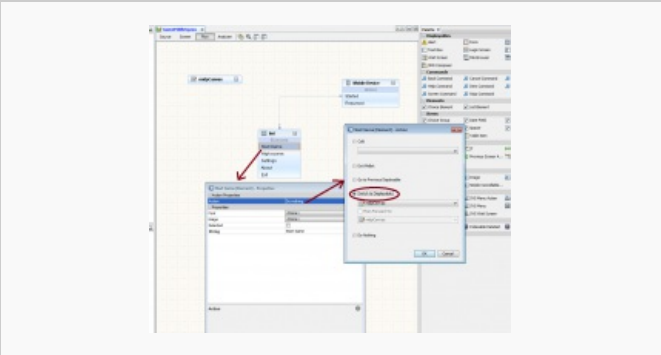
When we decided about SVG graphics based Main menu for capable devices what about low-range price devices as **Nokia N110** ? Usually devices with less capabilities has small screen and using standard LCDUI API List class is good enough for that. As alternative, Lightweight User Interface Toolkit is too heavy in size and does not bring much to small screen UI as simple LCDUI List

LCDUI API List based Main menu for not capable devices

Let us do it first as replacing `Form` with `List` in our midlet is the simplest task. I hope you might be able to recall that we have created our application skeleton by using NetBeans application master in the Step 1 described in article Rapid game development for Series 40 with NetBeans Visual Designer and Game Builder. The master creates application midlet with `Form` by default. To replace `Form` with `List` do it is several steps in NetBeans Visual Designer:

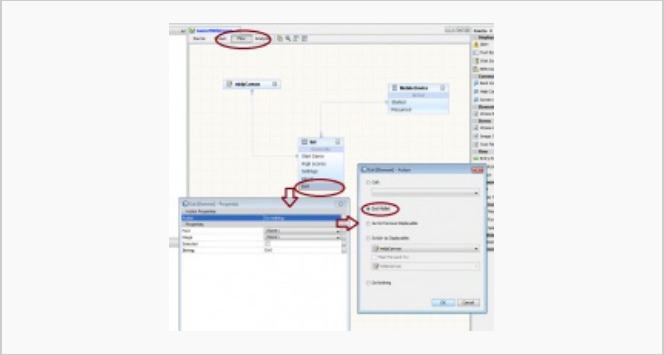


Visual designer - replacing `Form` with `List`



Visual designer - adding *Start Game command* to `List`

(C) Copyright Nokia 2014. All rights reserved.



Visual designer - adding *The application Exit command* to `List`

Note: popup dialog on the screenshots are invoked from context menu (right button click)

Now re-build you application and try. You should be able to use "Start" and "Exit" menu items to start your game and exit the application correspondingly.

SVG graphics based Main menu for advanced devices

Scalable 2D Vector Graphics API gives us not many classes to facilitate our work. In general it can be considered as low level interface for scalable graphics. Fortunately NetBeans community provides `org.netbeans.microedition.svg` package that allow us to implement animated SVG based user interface really fast. To facilitate development even more NetBeans IDE contains *Visual Designer* where you can find all necessary components from `org.netbeans.microedition.svg` package in a SVG components section of *Component Palette*. It allows you to build user interface with components drag and drop technique without diving into the implementation details while focusing on art work for your application.

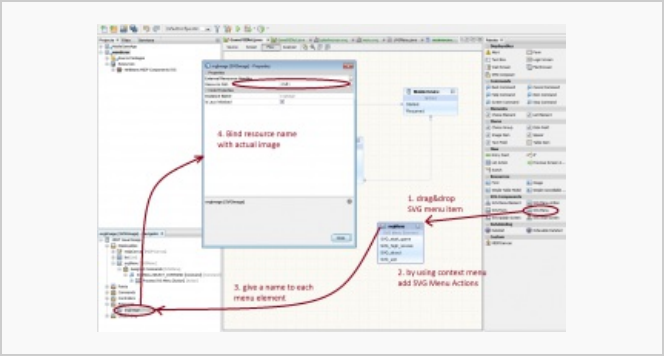
Application development workflow with SVG based UI

1. Add an SVG image to the Project in <project_home>/src directory of your project.
2. Add an SVG Graphic to the Application Flow

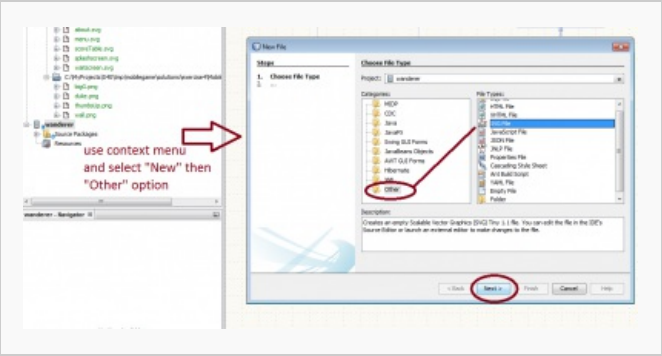
Drag an `SVGMenu`, `SVGSplashScreen`, or `SVGWaitScreen` component from the Component Palette into the Flow Designer.

Drag and drop an SVG graphic from the Projects window into the component.

Alternately, you can double-click a component to open it in the Screen Designer, then drag an SVG file into the open screen.



Adding SVG menu in four steps



Adding new SVG image.

Visual Designer contains SVG graphics editor, not much powerful but you can use third party application tools to produce SVG content. The editor allows you to layout element visually. To modify element properties you have to use text editor. If you are fine with editing XML in text editor this link may be helpful

To get more information on NetBeans tools please check NetBeans help. For example working with SVG is explained in the document chapter found by path *"Java ME applications --> Adding and Editing SVG Graphic Files"* Visual designer overview can be seen here

How SVG graphic elements are bounded to `org.netbeans.microedition.svg.SVGMenu` ?

That is very important question to make your SVG menu reacting on user key pressing. Note, i did not mentioned about user touch screen input -- supporting touch screen in SVG menu is a different story. So far we are trying to make it working with keypad only. SVG grahics elements are bounded to SVGMenu by element id. There is quite simple name convention: element id in SVG and

(C) Copyright Nokia 2014. All rights reserved.

menu element shall match. The following fragment demonstrates that rule:

```
    <g transform="matrix(2.121951,0.0,0.0,1.4285715,34.0,46.0) id="button_0">
............. <!--  SVG source. Not important for understanding -->
    </g>
.............
    <g transform="matrix(2.121951,0.0,0.0,1.4285715,34.0,46.0) id="button_1">
.............
```

```
    public SVGMenu getSvgMenu() {
        if (svgMenu == null) {
            // write pre-init user code here
            svgMenu = new SVGMenu(getSvgImageMainmenu(), getDisplay());
            svgMenu.setTitle("svgMenu");
            svgMenu.addCommand(SVGMenu.SELECT_COMMAND);
            svgMenu.setCommandListener(this);
            svgMenu.setFullScreenMode(true);
            svgMenu.addMenuElement("button_0");
            svgMenu.addMenuElement("button_1");
            svgMenu.addMenuElement("button_2");
            svgMenu.addMenuElement("button_3");
            svgMenu.addMenuElement("button_4");
            // write post-init user code here
        }
        return svgMenu;
    }
```

Note: above is a generated code. I used SVGButton element in place of menu items. There can be a text, a label or anything. However if your use already made components from Visual Designer Palette you will not have to design element visual effect and input focus transitions.

And here is the project step source code File:S40 tutorial step 8 SVG menu keypad only.zip. Note at this step it does not support touch screen.

Adding touch screen support

Examining `SVGMenu` we can see that it is derived from `Canvas` based class (through inheritance). In `SVGMenu` inheritance tree (i mean several of classes) there is implemented an ordinary scheme when `Canvas` instantiates internally `CommandListener` and receives system events on user input. `SVGMenu` implementation handles only `public void keyPressed(int keyCode)` ignoring pointer events. Pointer events are exactly our concern for touch screen support. Quite straightforward approach would be to derive a new class from `SVGMenu` and setup new `CommandListener`, but that is not possible because of private class members in `SVGMenu`. For some reason, svg menu package developers have prevented inheritance. I think that is done by mistake -- someone just prefers private keyword over protected :). So finally there is nothing to do as copy-paste SVGMenu implementation in new class just to setup custom CommandListener ... Arrrrh

Note: We have to customize `GameMIDlet` code that we initially generated with NetBeans Visual Designer. At that point we cannot use Visual Designer because it will ruin our implementation. From the other hand we cannot use NetBeans editor either because it protects generated code from manual modification. So please use your favorite text editor. I use Notepad++.

Design approach

Original `SVGMenu` implementation provides sequential access to SVG element in the list by using high level calls "set focus next" and "set focus previous". We cannot easily extend that scheme because we need direct access to element and we need somehow to obtain the element coordinates. SVG element properties can be obtained by name -- thus such extension design would involve a name convention that is fragile and totally in-acceptable. Fortunately there is another way. W3 SVG DOM

(C) Copyright Nokia 2014. All rights reserved.

model contains Events and EventsListeners that allow to assign a pointer click handler immediately to SVG element. That is nice! Few steps we need to add pointer support:

1. implement `org.w3c.dom.events.EventListener` interface
2. add eventListener to each menu item

through inheritance hierarchy derives finally To see needed update please check the code delta File:Adding touch support to SVG code delta Report.zip Wander how little code update is needed for touch support! And please check on Emulator how it works : File:S40 tutorial step 9 SVG menu touchscreen 1 debugoutput.zip. While clicking on menu item you should see debug output in NetBeans:

```
pointerPressed(int x, int y) : 123, 108
handleEvent : type = click,
targetcom.nokia.mid.impl.isa.dom.svg.CBFSVGLocatableElement@14896eb0
pointerPressed(int x, int y) : 130, 164
handleEvent : type = click,
targetcom.nokia.mid.impl.isa.dom.svg.CBFSVGLocatableElement@14895644
pointerPressed(int x, int y) : 131, 217
handleEvent : type = click,
targetcom.nokia.mid.impl.isa.dom.svg.CBFSVGLocatableElement@14893de8
pointerPressed(int x, int y) : 131, 280
handleEvent : type = click,
targetcom.nokia.mid.impl.isa.dom.svg.CBFSVGLocatableElement@145f5c9c
```

After we have made debug version working, there is no problem to handle `Displayable` switching. It is better to download the final project step File:S40 tutorial step 10 SVG menu touchscreen complete.zip and examine code delta to previous version. Note this code contains device capability detection code to disable SVG menu and enable List for low-capable device. Source code brief explanation please see below.

Backward compatibility with low-price devices with lack of SVG support

If we detect low-price device with missing SVG capability, we need somehow to enable List menu functionality in the application run-time. That is easy with Reflection API as we have already done that to separate Gesture API instantiation on keypad-only devices. Please see previous article for details.

For backward compatibility with devices that lack SVG the following function was introduced :

```
    private void showMainMenu(){
        try{
            Class.forName("org.w3c.dom.events.EventListener");
            switchDisplayable(null, getSvgMenu().getSvgCanvas());
        }catch(Exception e){
            switchDisplayable(null, getList());
        }
    }
```

This function uses Reflection API to detect whether SVG supporting is enabled on device. Depending on the checking result it will enable or disable SVG menu. For details please check source code File:S40 tutorial step 10 SVG menu touchscreen complete.zip

Summary

This article completes game application example skeleton that is suitable to run on touch screen devices and key-pad only devices

(C) Copyright Nokia 2014. All rights reserved.

(C) Copyright Nokia 2014. All rights reserved.