

Game development for Series 40 : adding touch support

This article explains how to add touch screen support in a typical 2D adventure or quest game skeleton. This is the next article after "[Rapid game development for S40 with NetBeans Visual Designer and Game Builder](#)" in wiki article series. The previous article ends when application skeleton is ready. Please refer to [Rapid game development for S40 with NetBeans Visual Designer and Game Builder](#) article if you want to go through the instructions from very beginning.



22 Jul
2012

Introduction

This article is a part of wiki article series "Game development for Series 40". Here we explain how to create [your game application from scratch](#). 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.

Touchscreen support

On Series 40 devices with a touch screen, touch interaction is supported by all LCDUI elements. LCDUI consisted of

1. High level API
2. Low level API

Low-level key or pointer input is not available in high-level APIs by default. They can only be accessed via [Canvas](#) or [CustomItem](#). For game development we are not interested in CustomItem because it is for application Forms. In game development we use [GameCanvas](#) as the game view. GameCanvas derives from Canvas. With GameCanvas, it is possible to check keys pressing status and capture pointer events that come from touch screen. For touch screen devices we are interested in the following methods:

- [pointerDragged\(int x, int y\)](#)
- [pointerPressed\(int x, int y\)](#)
- [pointerReleased\(int x, int y\)](#)

As you can see all we can get from the methods above is only pointer coordinates – not much for control implementation. Thus we need a virtual control consisted of screen buttons that will be placed on game view. Designing such control we are thinking how it can be re-used in a typical game.

Game control design approach

On- screen control

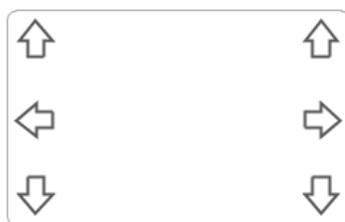
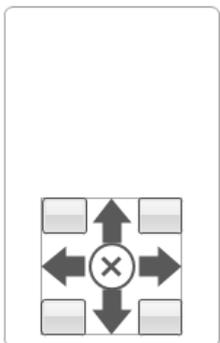
Game control visual appearance shall depend on device orientation. Because in games with landscape view user holds device with the both hands and the game control can be divided in two parts placed on left and right side of the game view. In contrary to landscape, in portrait orientation user uses one hand to hold device and control the game. Typical commands a game needs are:

- Navigate command :
 - Left
 - Right
 - Up
 - Down
- At least two extra commands that are context dependent:
 - Enter
 - Escape (select, menu, etc...)

Visual appearance

As you can see in the picture below, the end result is not as good as we expected. The game controls takes too much space leaving no room for the game scene.

Nokia 311 device screen in portrait and landscape orientations, 1:1 scale

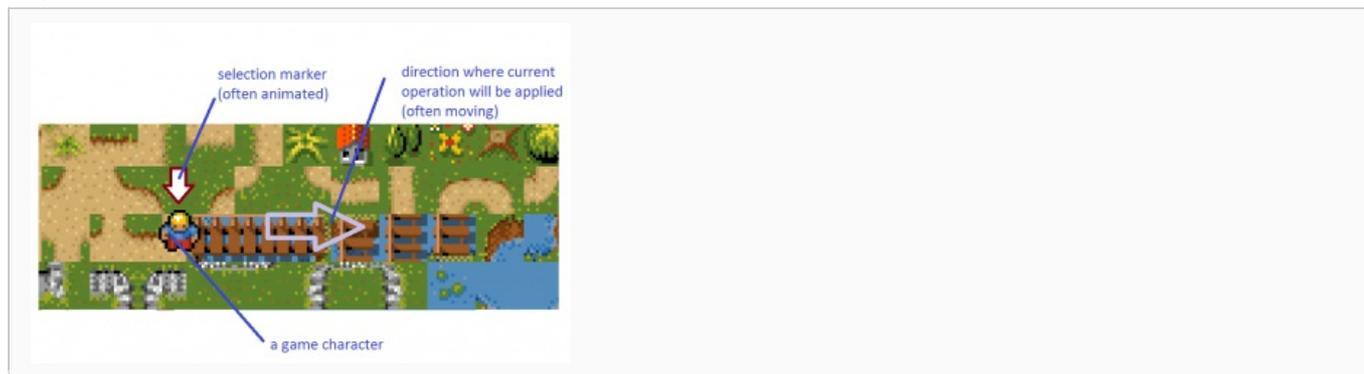


Note: Minimal button size on screen cannot be less than 6X6 millimeters otherwise it considerably affects usability

Logical control

In this approach game control does not appears on screen as common control items but consists of several elements that works together : selection mark and operation. User controls game character in two steps :

1. touch screen to select a game character
2. touch screen the second time to point direction where current operation will be applied. Implicitly current current operation is moving until the second point.



Step 3 introducing selection marker and game character animation

Please check the game project [File:S40 tutorial step 3-added animation and character selector.zip](#) updated with introducing selection marker and game character animation.



Note: At step 3 only animation and new sprite "Selection marker" were added. To see game control in action please check further project steps. When project is broken down in steps then it is convenient to examine code delta in some file diff tools (as [WinMerge open source project](#) for example) for better understanding.

Step 3 implementation comments

To animate your [Sprite](#) use methods [nextFrame\(\)](#) and [prevFrame\(\)](#). To add animation image sequence to Sprite check [previous article](#) in the series.

Animation code :

```
void setPlayerMovingUp(boolean b) {
.....
    gameDesign.getFatman().setPosition(xposFatman, yposFatman);
    gameDesign.getFatman().nextFrame(); // enabling animation. When game loop is working it will cause to display each t
.....
}
```

Step 4 introducing game character "Fatman" moving control

As it is pointed above. `GameCanvas` can receive pointer events :

- `pointerDragged(int x, int y)`
- `pointerPressed(int x, int y)`
- `pointerReleased(int x, int y)`

all we need is to introduce a state machine in `LayerManager`. State machine supports the following states:

- Idle
- MapPanning, when user is able to pan game world
- CharacterSelected, when user selects a `Sprite` to apply operation lately : moving
- CharacterDirected, when user executes operation on selected sprite.

Taking into account crowing amount of functionality resides in one class and corresponding growing complexity of code maintenance, we have chosen State design pattern for our state machine. There are Context is `LayerManager` and several states implement State interface. Not big deal please check "State" design pattern for details. Implementation code you can find by link [File:S40 tutorial step 4-character moving.zip](#)

Step 5 making game character moving smoothly

As you perhaps noticed, on the project step 4 we have done game character control movement not perfect because while movement the character disappears from current position and suddenly appears on position of destination. Not good. We need smooth animation while the character moving. Now you can realize all benefits of "State" design pattern we have chosen on step 4. From some point in development it is impossible to maintain and extend logic in switch operator! Otherwise you end up with spaghetti code. Please check the final implementation [File:S40 tutorial step 5-character moving smoothly.zip](#)



Note: Do not keep context passed by reference in State class member variable. Java garbage collector keeps objects until references to them are reachable. If you keep reference to objects they will not be destroyed. That has bad consequences known as "memory leaks" in Java.



Note: Please remember after event `pointerPressed` follows event `pointerReleased`. If you created a state on `pointerPressed`, `pointerReleased` event will be handled by new state!

Adding Gestures API to support game world pan

Game world size is bigger than display size. If you try the application on keyboard-only device, map panning starts when the character reaches display edge. To implement map panning on touch support device we need to use [Gestures API](#) because `Canvas.pointerDragged` event is not processed correctly.

It is impossible to get movement delta in pixels. This event handler brings some data that is totally irrelevant to current movement. Nokia Gestures API is intended for Nokia devices and give more possibilities in touch screen handling. Another improvement in code implementation is re-working state machine

to not allocating new state every time that may lead to holding references to objects and thus java memory leaking

Please download the game application project touch support complete [File:S40 tutorial step 6-gestures API.zip](#)

Supporting keypad-only devices

If you try [File:S40 tutorial step 6-gestures API.zip](#) on keypad-only device as **Nokia 110**, it will pop up error "Class not found". The problem appears because this device does not contain Gestures API classes. Fortunately java has mechanism that allow to examine class instances and obtain their meta-data. That is [Reflection API](#) please check [how it can be used](#). For our purpose we use call `Class.forName` to obtain information about Gesture API. If the call succeeds then we are running on a touch device. In case of exception we are on keypad-only device:

```
// for backward compatibility with keyboard devices
try{
    Class.forName("com.nokia.mid.ui.gestures.GestureRegistrationManager");
    // This is a touch device
    new MyGesture(this);
} catch (Throwable e){
}
```



Note: It is important to have conditionally instantiated class defined in separate file -- in our case `MyGesture.java` -- i.e. inline class will not work with such trick

Please download and try [File:S40 tutorial step 7 - gestures API and keypad-only devices.zip](#). Later on we will use Reflection API to instantiate conditionally SVG supporting.

See the next article in the series dedicated to Main menu introducing !

What else can be done to the control (out of the article series)

There is some room for the control improvement is left outside of the implementation.

1. when there are several game objects user can control -- for example squad in strategy game - user should be able to cancel a selection and choose other one. Usually it is done so if user selects already selected object (or character) selection should be canceled. That functionality is easy to implement in our state machine -- just hide selection `aContext.gameDesign.getSelmark().setVisible(false)`; in proper state.

Summary

Touch screen device supporting is not as simple as just handling pointer event in your application implementation. Because of extremely small screen on Series 40 devices, user interface and especially game control shall be not only handy but also it shall not obscure game scene. Straightforward design approach does not work considering minimal geometry for push button on screen 6X6 millimeters in size. The articles suggest touch control implementation suitable for quest or strategy games.

