Using basic touch gestures

Overview

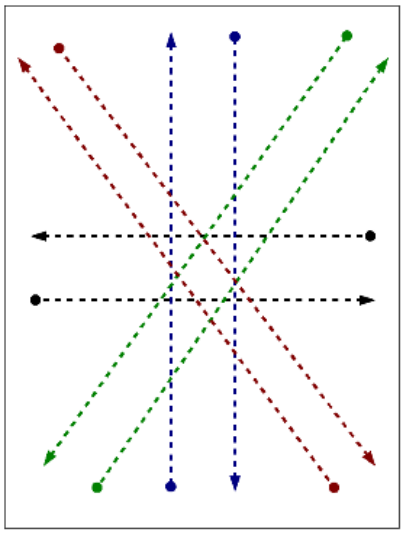
14 Dec 2008

The S60 5th Edition platform has brought full support of touch UI, and so provides possibilities for enriching application UIs. A UI that is too complicated and overladen with many pop-up menus and toolbar buttons can be quite annoying and inconvenient for the average user. To perform even a basic function such as selecting a menu item or clicking a toolbar button they have take **ATTENTIVE** look at the application.

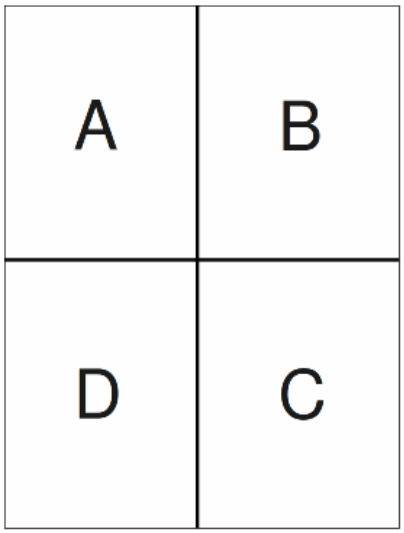
Touch UI support can help reduce the amount of user actions needed for these application functions, for example by mapping basic touch gestures to the application's main functions.

Basic touch gestures include the following: Up-Down/Down-Up, Left-Right/Right-Left, RightUp-LeftDown/LeftDown-RightUp, and RightDown-LeftUp/LeftUp-RightDown. When the application main functions are mapped to touch gestures, the user does not necessarily need to take an **ATTENTIVE** look at the application. For example, moving to the next track in media player could be performed with quite inaccurate Left-Right gesture (and moving to the previous track with a reverse Right-Left gesture).

To simplify the problem of identifying the gesture type, it can be supposed that basic gestures are described only with their start and end points. As a result, by knowing the relative locations of those points, we can easily identify the gesture type. Moreover, using touch gestures also enriches the user experience in general (the released menu items and toolbar buttons can be assigned to application's other functions).



Eight simplified basic touch gestures.



Four basic areas used for identifying gesture types.

Preconditions

(C) Copyright Nokia 2013. All rights reserved.

The following code sample is valid only for touch UI enabled devices. To check whether pen support is enabled, use the AknLayoutUtils::PenEnabled() methods. To simplify matters, the example assumes that touch UI support is present.

MMP file

```
LIBRARY avkon.lib euser.lib
CAPABILITY could be self-signed
```

Header file

```
// Eight basic touch gestures
enum TGestureType
{
 ENoneGesture = -1,
 EUpDown,
 EDownUp,
 ERightLeft,
 ELeftRight,
 ELeftUpRightDown,
 ELeftDownRightUp,
 ERightUpLeftDown,
 ERightDownLeftUp
};

// Four basic types of positions.
// The control rectangle can be devided into four main areas, for example A, B, C, and D .
enum TPositionType
{
 ENonePosition = -1,
 EAreaA,
 EAreaB,
 EAreaC,
 EAreaD
};

/**
 * Container class
 */

class CBasicGesturesExContainer : public CCoeControl
    {
     ...
private:
        /*
 * From CCoeControl, HandleResourceChange
 * This function gets called whenever a pointer event occurs.
 * @param aEvent The pointer event.
 */
    void HandlePointerEventL( const TPointerEvent& aEvent );

private:
 /*
 * Identifies the gesture type.
 * @param aStartPoint The start point coordinates.
 * @param aEndPoint The end point coordinates.
 */
 TGestureType GetGestureType(const TPoint& aStartPoint, const TPoint& aEndPoint);

 /*
 * Handles the specified gesture.
 * @param aGesture The gesture type.
 */
 void HandleGesture(const TGestureType& aGesture);

private: // data
 /*
 * Stores the gesture start point coordinate
 */
 TPoint iStartPoint;

 /*
 * Indicates whether there is any gesture.
        * EFalse by default.
 */
 TBool iGesture;
   ...

    };
```

Source file

```
...
void CBasicGesturesExContainer::ConstructL(const TRect& aRect)
    {
 ...

 // Enables handling of drag events
 EnableDragEvents();
        ...
    }
void CBasicGesturesExContainer::HandlePointerEventL( const TPointerEvent& aEvent )
    {
     switch (aEvent.iType)
     {
   case TPointerEvent::EButton1Down:
   // Save tap position (because it could be the beginning of the gesture)
   iStartPoint = aEvent.iPosition;
```


(C) Copyright Nokia 2013. All rights reserved.

```
    break;

    case TPointerEvent::EButton1Up:
    if (iGesture)
    {
     // Idenditify type of gesture
     TGestureType gesture = GetGestureType(iStartPoint, aEvent.iPosition);

     // Perform necessary action depends on gesture type
     HandleGesture(gesture);
    }

    // Reset gesture indicator
    iGesture = EFalse;
    break;

    case TPointerEvent::EDrag:
    iGesture = ETrue;
    break;

        default:
        break;
      }
     }

TGestureType CBasicGesturesExContainer::GetGestureType(const TPoint& aStartPoint, const TPoint& aEndPoint)
{
 // Default return value
 TGestureType returnValue = EUpDown;

 // Control's extent
 TRect rect(Rect());
 TInt rectHalfWidth = rect.Width()/2;
 TInt rectHalfHeight = rect.Height()/2;

 // Start point position type
 TInt startPointLocation = EAreaA;
 // End point position type
 TInt endPointLocation = EAreaA;

 // Determine in which logical part of the control is the start point
 if (aStartPoint.iX <= rectHalfWidth)
 {
  if (aStartPoint.iY <= rectHalfHeight)
  {
   startPointLocation = EAreaA;
  }
  else if (aStartPoint.iY > rectHalfHeight)
  {
   startPointLocation = EAreaD;
  }
 }
 else if (aStartPoint.iX > rectHalfWidth)
 {
  if (aStartPoint.iY <= rectHalfHeight)
  {
   startPointLocation = EAreaB;
  }
  else if (aStartPoint.iY > rectHalfHeight)
  {
   startPointLocation = EAreaC;
  }
 }

 // Determine in which logical part of the control is the end point
 if (aEndPoint.iX <= rectHalfWidth)
 {
  if (aEndPoint.iY <= rectHalfHeight)
  {
   endPointLocation = EAreaA;
  }
  else if (aEndPoint.iY > rectHalfHeight)
  {
   endPointLocation = EAreaD;
  }
 }
 else if (aEndPoint.iX > rectHalfWidth)
 {
  if (aEndPoint.iY <= rectHalfHeight)
  {
   endPointLocation = EAreaB;
  }
  else if (aEndPoint.iY > rectHalfHeight)
  {
   endPointLocation = EAreaC;
  }
 }

 // Get the actual type of the gesture
 switch (startPointLocation)
 {
  case EAreaA:
   switch (endPointLocation)
   {
    case EAreaA:
     returnValue = ENoneGesture;
    break;

    case EAreaB:
     returnValue = ELeftRight;
    break;

    case EAreaC:
     returnValue = ELeftUpRightDown;
    break;

    case EAreaD:
     returnValue = EUpDown;
    break;

    default:
     returnValue = ENoneGesture;
    break;
   }
  break;

  case EAreaB:
   switch (endPointLocation)
   {
    case EAreaA:
```


(C) Copyright Nokia 2013. All rights reserved.

```
      returnValue = ERightLeft;
     break;

     case EAreaB:
      returnValue = ENoneGesture;
     break;

     case EAreaC:
      returnValue = EUpDown;
     break;

     case EAreaD:
      returnValue = ERightUpLeftDown;
     break;

     default:
      returnValue = ENoneGesture;
     break;
    }
   break;

   case EAreaC:
    switch (endPointLocation)
    {
     case EAreaA:
      returnValue = ERightDownLeftUp;
     break;

     case EAreaB:
      returnValue = EDownUp;
     break;

     case EAreaC:
      returnValue = ENoneGesture;
     break;

     case EAreaD:
      returnValue = ERightLeft;
     break;

     default:
      returnValue = ENoneGesture;
     break;
    }
   break;

   case EAreaD:
    switch (endPointLocation)
    {
     case EAreaA:
      returnValue = EDownUp;
     break;

     case EAreaB:
      returnValue = ELeftDownRightUp;
     break;

     case EAreaC:
      returnValue = ELeftRight;
     break;

     case EAreaD:
      returnValue = ENoneGesture;
     break;

     default:
      returnValue = ENoneGesture;
     break;
    }
   break;

   default:
    returnValue = ENoneGesture;
   break;
  }

  return returnValue;
 }

 void CBasicGesturesExContainer::HandleGesture(const TGestureType& aGesture)
 {
  TBuf<64> messageText(KNullDesC);

  // Handle gesture
  switch(aGesture)
  {
   case EUpDown:
    messageText = _L("Gesture: from Up to Down");
   break;

   case EDownUp:
    messageText = _L("Gesture: from Down to Up");
   break;

   case ERightLeft:
    messageText = _L("Gesture: from Right to Left");
   break;

   case ELeftRight:
    messageText = _L("Gesture: from Left to Right");
   break;

   case ELeftUpRightDown:
    messageText = _L("Gesture: from Left-Up to Right-Down");
   break;

   case ELeftDownRightUp:
    messageText = _L("Gesture: from Left-Down to Right-Up");
   break;

   case ERightUpLeftDown:
    messageText = _L("Gesture: from Right-Up to Left-Down");
   break;

   case ERightDownLeftUp:
    messageText = _L("Gesture: from Right-Down to Left-Up");
   break;

   default:
    // Gesture type's not identified,
```


(C) Copyright Nokia 2013. All rights reserved.

```
    // do nothing
                        return;
   break;
  }

  CAknInformationNote* note = new ( ELeave ) CAknInformationNote;

     // Show the information note with a previously defined text
     note->ExecuteLD(messageText);
 }
 ...
```

Limitations

Identifying of touch gestures is simplified. Only eight basic ones can be distinguished.

Postconditions

Basic touch gestures are identified and can be mapped to the required application functions.

See also

- How to check whether Pen Support is Enabled
- How to Handle Pointer Events in a Custom Control

Example source code

Media:BasicTouchGestures.zip

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.