The Keypad SCI class

Lars Skovlund

January 17, 2007

1 Introduction

This is the documentation for the Keypad SCI class, written by Lars Skovlund. I have attempted to make the class as general as possible. Therefore, the class and its descendant DisplayKeypad support arbitrarily long strings of any kind, so it is possible, for instance, to have each button represent a word so the user can construct sentences. The DisplayKeypad class prints the input string somewhere on the screen via the Display kernel call (all parameters are selectable via properties).
The following is a complete feature list for the keypad:

- No keyboard input, for generality reasons.
- Special button functionality, such as ENTER, BACKSPACE and CANCEL.
- Optionally invokes a script when initially showing the keypad and when the ENTER button is pressed.
- Optional AUTO EXIT mode in which the keypad triggers as soon as the input string reaches its maximum length, as if the user explicitly pressed the ENTER button.
- Can require a specific number of buttons to be pressed, or accept any number of button presses.
- Can play a sound when a button is pressed.
- Optional text display (DisplayKeypad).
- A background view can be used or not.

2 Example of use

2.1 Declarations

```
(local kptxt[5])

(instance myKeyPad of DisplayKeypad
        (properties
                maxLength       4
                autoExit        FALSE
                script          0
                longestText     1       /* one character */
                color           clBLACK
                back            clWHITE
                x               20
                y               160
requireMax TRUE
        )

)

(instance key1 of KeypadButton
        (properties
                x       100
                y       100
                view    0
                loop    0
                text    "1"
        )
)

(instance key2 of KeypadButton
        (properties
                x       120
                y       100
                view    0
                loop    2
                text    "2"
        )
)

(instance key3 of KeypadButton
        (properties
                x       140
                y       100
```

```
                view    0
                loop    1
                text    "3"
        )
)

(instance key4 of KeypadButton
        (properties
                x       160
                y       100
                view    0
                loop    5
                special KEYPAD_SPECIAL_CANCEL
        )
)

(instance key5 of KeypadButton
        (properties
                x       180
                y       100
                view    0
                loop    6
                special KEYPAD_SPECIAL_BKSP
        )
)

(instance key6 of KeypadButton
        (properties
                x       200
                y       100
                view    0
                loop    3
                special KEYPAD_SPECIAL_ENTER
        )
)
```

2.2 Handling the **enter** and **cancel** buttons

```
(instance KeypadScript of Script
        (properties)
        (method (changeState newState)
                (super:changeState(newState))
                (switch (newState)
                        (case KEYPAD_DONE_STATE
                          ((send client:hide(FALSE))
```

```
                            FormatPrint("You typed: %s" @kptxt)))
(case KEYPAD_ERROR_STATE
                            (FormatPrint("You need to press %d buttons."
                                   (send client:maxLength())))
                )
        )
)
```

2.3 Initialization and activation

```
(myKeyPad:setScript(KeypadScript) outputString(@kptxt)
  init(key1 key2 key3 key4 key5 key6))

(myKeyPad:init() show()) /* The init ensures that the keypad can
                            be used any number of times. */
```

3 What to draw

You will need a view consisting of several loops, one for each button. Each loop should contain two cels: The button in its normal and depressed state, respectively. It is important that the cels are ordered in this way; the code depends on it, and you should not set the **cel** property to anything when declaring KeypadButtons.

If you want a background view you can draw this as a separate loop. Declare it in your code as an instance of the Prop class. Then refer to it in your Keypad instance using the **backView** property.

4 Class reference

4.1 Keypad class

Superclass: **EventHandler**

This class provides the base functionality for keypads.

Methods of the Keypad class:

(Keypad init: buttons . . .)

This function initializes the object, as is the case with every SCI class. In particular, it resets the input string and notifies the controlling script. Buttons is a list of button views (instances of KeypadButton) to include in the keypad.

(Keypad show:)

Displays the keypad. The player is prevented from moving until the keypad is dismissed.

(Keypad hide: doCue)

Hides the keypad. DoCue should be set to TRUE if the "enter" button was pressed, FALSE otherwise. When the script is called, the buttons have not yet been removed from the screen. If you want this, your script must explicitly call **hide** with a `FALSE` parameter (to avoid infinite recursion)—this is what the example code does, see section 2.2.

(Keypad backspace:)

Self-explanatory. You will need to override this if you use text of varying lengths for the buttons.

(Keypad draw:)

(Keypad setSize:)

Do nothing. They are only here to prevent the game from crashing in certain situations (in particular, restoring the game to a position where the keypad is displayed).

(Keypad depress: which)

Simulates the pressing of a button. Which refers to an actual instance of KeypadButton.

(Keypad setScript: newScript params)

Sets the controlling script of the keypad. This script is called once when the keypad is initially displayed and once when the user presses "enter". It is not called if the user cancels the dialog.

Properties of the Keypad class:

outputString Points to the output buffer. Remember to set this at run time, and remember that it must be big enough for anything the player might press. Use a local variable, as shown in section 2.1.

maxLength Indicates the length of the output buffer.

autoExit Boolean variable which toggles the "auto exit" mode, described in the introduction.

backView Points to an instance of Prop which is displayed when the keypad is activated. It is meant to be used for a background, hence the name.

backPri Indicates the priority of the background. May be omitted; the priority of the buttons is always 1 greater than this value, if specified.

btnSound Points to an instance of Sound which is played whenever the user presses a button.

longestText Indicates the length of the longest button text.

script Indicates the controlling script. Use **setScript** to set this, never set it directly.

state Boolean variable. Indicates whether the keypad is currently shown or not.

special Not used for anything. Works around a chicken-and-egg problem regarding selector names in SCI Studio.

requireMax Setting this to TRUE makes the keypad require **maxLength** characters to be filled in. If this requirement is not met when the player presses ENTER, the attached script executes its KEYPAD_ERROR_STATE. You can do whatever you want here, including doing nothing, displaying an error message, or blowing up the universe. The keypad remains functional after the script returns.

Notes

For the predefined script, three states are predefined. Constants are provided in the enclosed header file.

4.2 KeypadButton class

Superclass: **Prop**

This class provides the button views to be used for the keypad; they provide an event handler and a special feed-back interface for the Keypad class. None of the overridden methods are meant to be called directly.

Methods of the KeypadButton class:

(KeypadButton init: theClient priority)

(KeypadButton handleEvent: event)

Properties of the KeypadButton class:

text The text to add when the button is pressed.

special The special meaning that this button is to have.

client A pointer to the keypad which "owns" this button. Filled in automatically.

Notes

Three special button types are provided beside the default, namely ENTER, BACKSPACE and CANCEL.

4.3 DisplayKeypad class

Superclass: **Keypad**

This class provides the additional functionality of displaying the current text string at a predetermined position on the screen. All of the listed properties have the same format as those passed to the Display kernel function.

Properties of the DisplayKeypad class:

underBits stores a handle to the screen area under the text string.

x

y Coordinates where the text string is to be displayed.

font The font to use.

color The foreground color.

back The background color.

width The width to use for wrapping the text, if that should ever be necessary.

align The alignment to use.

5 Constant reference

```
(define KEYPAD_INIT_STATE 0)
(define KEYPAD_DONE_STATE 1)
(define KEYPAD_ERROR_STATE 2)

(define KEYPAD_SPECIAL_NONE 0)
(define KEYPAD_SPECIAL_ENTER 1)
(define KEYPAD_SPECIAL_BKSP 2)
(define KEYPAD_SPECIAL_CANCEL   3)
```