WiiCursor

Summary

WiiCursor is a program that enables you to control the mouse cursor with a Wiimote. The buttons on the Wiimote can be configured to fire different mouse and keyboard events. WiiCursor has only been tested on Windows XP and might not work on Windows Vista.

Usage

WiiCursor uses the WiiMoteLib by Brian Peek, which can be found at http://www.codeplex.com/WiiMoteLib/

In order to use WiiCursor the first thing you have to do, is to connect your WiiMote using Bluetooth. Brian peek shows how in this article http://blogs.msdn.com/coding4fun/archive/2007/03/14/1879033.aspx under the heading "Getting connected"

Secondly you should place a Wii Sensor bar under your screen. I recommend that you buy a battery driven sensor bar for this. I have bought one from Logic3 which works fine. You can also build one yourself as shown here.

As the least appealing alternative you can use the one that comes with the Wii, but this means that you have to have it connected to your Wii and your Wii has to be turned on, to provide the power. This way your Wii is used as a rather weird looking power adapter.

When you start WiiCursor the only thing that happens is that the WiiCursor Icon appears in the System Tray in the lower right corner.

If the Sensor bar is turned on, you immediately get to control the mouse cursor with the Wiimote.

If you press a button on the Wiimote what will happen depends on the chosen configuration.

Configuration

In the Installation directory, typically C:\Program Files\JERN\WiiCursor\ there is a folder called WCXML.

In this directory there is a Schema file "WCConfiguration.xsd" and some XML files. The XML files have to comply with the Schema.

For each XML file in the WCXML folder a submenu is added to the Configuration Menu of the WiiCursor Tray Icon.

An example file is shown here

```
<?xml version="1.0" encoding="utf-8" ?>
<WCConfiguration>
  <Name>DrawCursor Support</Name>
```

WiiCursor

```
  <Default>true</Default>
  <ConnectionWaitTime>5000</ConnectionWaitTime>
  <ConnectionRetries>50</ConnectionRetries>
  <Keys>
    <A>MouseLeftButtonDown</A>
    <B>KeyEscape+Exit</B>
    <Plus>KeyF7</Plus>
    <Minus></Minus>
    <Home></Home>
    <One>KeyDelete</One>
    <Two>Pause</Two>
    <Up></Up>
    <Down></Down>
    <Left></Left>
    <Right></Right>
  </Keys>
</WCConfiguration>
```

As you can see the XML file has 4 basic settings and a 5th which contains an entry for each button on the Wiimote.

1. `Name`: The name of the Submenu.
2. `Default`: Denotes if the file contains the default configuration of WiiCursor. I.e. the one chosen by the system when WiiCursor starts. One and only one file has to be default. This is actually not validated by the program. If the Default setting is not present in the file, it means the same as if Default is set to false.
3. `ConnectionWaitTime`: If the Wiimote is not connected via Bluetooth the WiiCursor program will fail to connect to the Wiimote when started. This setting denotes how many milliseconds the program will wait before trying to connect again.
4. `ConnectionRetries`: This indicates how many times the program will try to connect to the wiimote before giving up. I.e. with the above settings the program will try for 50 x 5000 milliseconds which equals 250000 milliseconds, or a bit more than 4 minutes.

Each button on the Wiimote can be configured to fire an event, e.g. simulating that a mousebutton or a key was pressed.

Each button can in fact fire more than one event. In the example above the “B” button first sends the Escape Key and then fires the Exit event.

Each event is explained here. I plan to add more events in the future. All Key and Mouse simulations will happen at the cursor location.

Name of event	Explanation
0-255	Numerical value corresponding to an ASCII code. E.g 65 means “A”
Pause	This event temporarily stops the Wiimote from interacting

	with the Desktop. This means that you get a chance to use the normal mouse. Triggering Pause once more will make the Wiimote work again.
Exit	Exit stops the WiiCursor program altogether.
KeyEscape	Sends the Escape key to the desktop
KeyDelete	Sends the Delete key to the desktop
KeyF1	Sends the F1 key to the desktop
KeyF2	Sends the F2 key to the desktop
KeyF3	Sends the F3 key to the desktop
KeyF4	Sends the F4 key to the desktop
KeyF5	Sends the F5 key to the desktop
KeyF6	Sends the F6 key to the desktop
KeyF7	Sends the F7 key to the desktop
KeyF8	Sends the F8 key to the desktop
KeyF9	Sends the F9 key to the desktop
KeyF10	Sends the F10 key to the desktop
KeyF11	Sends the F11 key to the desktop
KeyF12	Sends the F12 key to the desktop
KeyArrowUp	Sends the arrow up key to the desktop
KeyArrowDown	Sends the arrow down key to the desktop
KeyArrowLeft	Sends the arrow right key to the desktop
KeyArrowRight	Sends the arrow left key to the desktop
MouseLeftButtonDown	Sends a mouse left button down event to the desktop. The Key stays “down” until the key on the Wiimote is released.
MouseLeftClick	Sends a mouse left click to the desktop.
MouseLeftDoubleClick	Sends a mouse left double-click to the desktop (can also be simulated by pressing a button that activates MouseLeftClick twice)
MouseRightButtonDown	Sends a mouse right button down event to the desktop. The key stays “down” until the key on the Wiimote is released.
MouseRightClick	Sends a mouse right click to the desktop.

WiiCursor comes with three examples of XML configuration, the most important is WCForDrawCursor.xml for drawing on the screen with DrawCursor (see the other program on http://www.codeplex.com/WiiCursor) and WCPowerPoint.xml for presenting powerpoints.

Inner workings

The movements of the Mouse Cursor is controlled by utilizing the Infrared camera in the front of the Wiimote combined with a Wii Sensorbar.

WiiCursor calculates the cursor position by calculating the distance to the Infrared LED’s on each side of the Sensor Bar. When one LED is invisible it uses only one LED and it’s former position to calculate from.

WiiCursor uses a Win32 API called mouse_event for simulating Mouse events.

For Key events it uses the SendKeys.SendWait method. SendWait only worked if I put the

WiiCursor

<add key="SendKeys" value="SendInput"/>

In AppSettings in the app.config. This is how it worked on my machine. According to MSDN this resolves timing issues on some machine. If you have problems getting WiiCursor to work, you might try to delete this setting.

The Configuration is done by reading each XML file into a simple object, using LinqToXML. Linq is also used to Create the Submenu Items and hooking up the Click events to the respective XML files.