Tutorial 4: Cursors

http://www.codemii.com/2008/08/31/tutorial-4-cursors/

In this fourth Wii programming tutorial, I will cover displaying a cursor by drawing a small square, movement of the cursor and then using the Wii IR to move the cursor. Before reading this tutorial I'm assuming you have picked up a few things and therefore I won't be re-explaining the common code in the zip file or initial steps in detail.

Firstly download this tutorial4-blank which will contain the required files to get us started. Extract this zip file in C:\devkitPro\examples\gamecube. Open up tutorial4.pnproj and then click on main.c to show the source code.

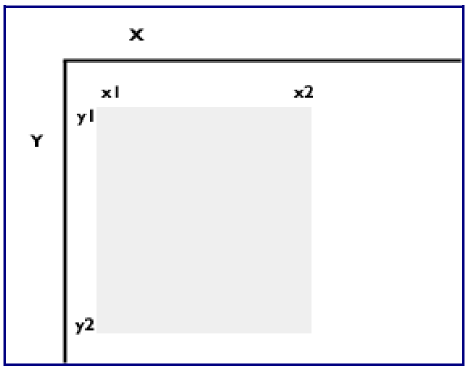
From tutorial 3, we know how to incorporate the controller in our source code, so now all we need to use a cursor in our application, is to draw one. The source code below is our drawing functions. I'm not sure whose code it but thanks to the person whom made it.

```
void DrawHLine (int x1, int x2, int y, int color) {
   int i;
   y = 320 * y;
   x1 >>= 1;
   x2 >>= 1;
   for (i = x1; i <= x2; i++) {
      u32 *tmpfb = xfb;
      tmpfb[y+i] = color;
   }
}

void DrawVLine (int x, int y1, int y2, int color) {
   int i;
   x >>= 1;
   for (i = y1; i <= y2; i++) {
      u32 *tmpfb = xfb;
      tmpfb[x + (640 * i) / 2] = color;
   }
}

void DrawBox (int x1, int y1, int x2, int y2, int color) {
   DrawHLine (x1, x2, y1, color);
   DrawHLine (x1, x2, y2, color);
   DrawVLine (x1, y1, y2, color);
   DrawVLine (x2, y1, y2, color);
}
```

Place the above code after the Initialise function. We will be calling the DrawBox function and passing 5 parameters to it. The parameters of DrawBox are the first x co-ordinate, the first y co-ordinate, the second x co-ordinate, the second y co-ordinate and the colour of the lines as shown below:

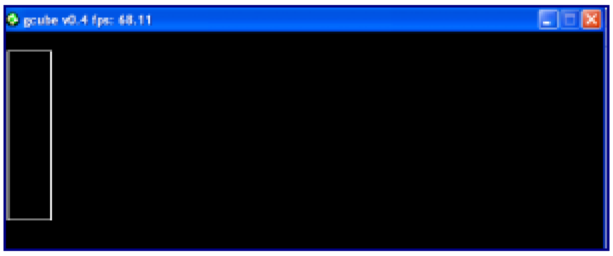
```
DrawBox (5, 20, 50, 200, COLOR_WHITE);
```

When drawing things on the screen, x = 0 and y = 0 means that we would be at the top left of the screen. If we put the following code as shown above, we would get a line that looks something like:



Once we call the DrawBox function, it breaks the square that we want to draw into 4 lines, which are 2 horizontal and 2 vertical lines. The DrawHLine and DrawVLine functions write to the frame buffer which will be displayed on the screen that we want a certain dot on the screen to be a certain colour. The loop is used so that we go through every dot that we want to change to a certain colour.

tutorial4-draw-square is the finished source code if you need it. Compile the source code (Alt + 1) and run the dol file with gcube. You should see a similar screen to the one shown below.



Moving the Cursor

We now wish to draw a smaller square which will represent our cursor. You can use something like:

```
DrawBox (300, 250, 301, 251, COLOR_WHITE);
```

Now that we've got our cursor set up, we need to be able to use the controls to move the cursor. To do this we need to use variables to control the cursor's x and y co-ordinates on the screen.

We can have two variables, one for the x axis named cursor_x and one for the y axis named cursor_y. These variables will have the default value of 200 and 200, so that it's near the centre of the screen.

```
int cursor_x = 300;
int cursor_y = 250;
```

The next thing to do is to capture the movement of the joystick, so when we move the joystick up or down it modifies the cursor_y variable and when we move it left or right it modifies the cursor_x variable. We can do this by either de-incrementing (subtracting 1) or incrementing (adding 1 to) the variables.

```
if (PAD_StickY(0) > 18) {
        cursor_y--;
}
if (PAD_StickY(0) < -18) {
        cursor_y++;
}
if (PAD_StickX(0) > 18) {
        cursor_x++;
}
if (PAD_StickX(0) < -18) {
        cursor_x--;
}
```

The — means de-increment and the ++ means increment. As an example, if we move the joystick up, we would de-increment cursor_y. So if cursor_y was 200 and we moved the joystick up, cursor_y's value would be 199.

All that's left now is to change where the image is being displayed every time we move the cursor. This is easily accomplished by using our variables with the DrawBox function as shown below:

```
DrawBox (cursor_x, cursor_y, cursor_x + 1, cursor_y + 1, COLOR_WHITE);
```

What we've done is used our cursor_x and cursor_y in place instead of the 300 (x1) and 250 (y1) values and have added 1 to the cursor to that the values will be 301 (x2) and 251 (y2) in order to draw a square.

The next thing to do is to clear the screen to black before showing the cursor and then using waiting for vertical sync. The reason we are clearing the screen is so that our cursor is cleared every time we move it. If we didn't clear the screen, then you will see the movement of the cursor stay on the screen.

The code below shows the complete controller code in a while loop with the screen being cleared:

```
int cursor_x = 300;
int cursor_y = 250;

while(1) {

        PAD_ScanPads();

        if (PAD_StickY(0) > 18) {
                cursor_y--;
        }
        if (PAD_StickY(0) < -18) {
                cursor_y++;
        }
        if (PAD_StickX(0) > 18) {
                cursor_x++;
        }
        if (PAD_StickX(0) < -18) {
                cursor_x--;
        }

        VIDEO_ClearFrameBuffer (rmode, xfb, COLOR_BLACK);

        DrawBox (cursor_x, cursor_y, cursor_x + 1, cursor_y + 1, COLOR_WHITE);

        VIDEO_WaitVSync();
}
```

The reason behind placing VIDEO_ClearFrameBuffer before DrawBox is so we don't clear the screen and then waste time doing other functions and then draw the cursor. If you had a lot of code which took a considerable amount of time to execute, you wouldn't see your cursor all of the time and it would be blinking. Instead we do the functions, clear the screen, draw the cursor and repeat the while loop, so therefore the cursor will stay on the screen all the time.

tutorial4-moving-cursor is what our source now looks like. Go ahead and re-compile the source and run the dol with gcube and use the Numpad up, down, left and right to move the small square which is our cursor.

Converting source to use Wii IR

We are now good to change our project to compile on the Wii instead of the gamecube. As per the last tutorial open your makefile and change the gamecube_rules to wii_rules in line that reads:

```
include $(DEVKITPPC)/gamecube_rules
```

Add the Wii and the Bluetooth libraries to our compiler as shown below:

```
LIBS     :=          -lwiiuse -lbte -logc -lm
```

In main.c include section include the wiiuse file:

```
#include <wiiuse/wpad.h>
```

As usual we will need to add a W in front of all the PAD_ commands and remove references to PAD_Substick.

When using the Wiimote, we firstly have to tell the system that we want to capture all the Wiimote's functionality which includes the Infra-red (IR) and the accelerometer. Another thing to tell the system is the resolution we want to bound the IR to. Both these commands are shown below and should be after the WPAD_Init function:

```
WPAD_SetVRes(0, 640, 480);
WPAD_SetDataFormat(WPAD_CHAN_0, WPAD_FMT_BTNS_ACC_IR);
```

The Wiimote's IR has a data structure which tells us things like are the co-ordinates valid, the x and y axis, the rotation and so forth. This information can be found in "C:\devkitPro\libogc\include\wiiuse\wiiuse.h".

We need to assign a variable to hold this data structure. We do this by naming the data structure which is ir_t and then assigning that data structure to our variable which is ir. The code below needs to be placed below the #includes which are at the top.

The reason we place the variable up the top is so that our variable will be a global which means it can be called and accessed from anywhere within our source code. If we were to assign it within a function, then the variable would only exist whilst inside that function and could not be called from anywhere else.

```
ir_t ir;
```

So now that we have assigned ir as the data structure ir_t, we need to constantly update the ir variable to get the latest readings from the Wiimote. This is done by using the below code:

```
WPAD_IR(0, &ir);
```

We are updating channel 0 (the first Wiimote's) IR data and feeding all that information into our ir variable.

The final step involved is to change the variables in the DrawBox function to use ir.x and ir.y instead of cursor_x and cursor_y. As you recall ir is our variable which contains different data structures, one of them being the x and y co-ordinates. As these are like variables in the ir variable, we reference them by using a dot between our main variable (ir) and the data structure variable (ir.x or ir.y).

```
DrawBox (ir.x, ir.y, ir.x + 1, ir.y + 1, COLOR_WHITE);
```

We can remove the cursor_x and cursor_y variables. The while loop looks as shown below:

```
while(1) {

            WPAD_ScanPads();
u32 pressed = WPAD_ButtonsDown(0);

            // IR Movement
            WPAD_IR(0, &ir);

if (pressed & WPAD_BUTTON_HOME) {
                        exit(0);
            }

            VIDEO_ClearFrameBuffer (rmode, xfb, COLOR_BLACK);

            DrawBox (ir.x, ir.y, ir.x + 1, ir.y + 1, COLOR_WHITE);

            VIDEO_WaitVSync();
}
```

tutorial4-wii-ir is what our source now looks like. Clean the source code, re-compile and tutorialr.elf to boot.elf and place it in a directory called tutorialr under the /apps/ directory of your SD card and load it with the homebrew channel. You will be able to control the dot on your screen with the Wiimote.

PDF by Pvt Ryan (http://www.ninet.org)

Changes: Formatting