Tutorial 3: Controller input

http://www.codemii.com/2008/08/24/tutorial-3-controller-input/

Welcome to the third Wii programming tutorial that will cover the gamecube/Wii controller interaction and using gcube. I'm going to assume that you've read the previous 2 tutorials.

Firstly download this tutorial3-blank zip file which will contain some standard source code, the makefile and the programmer's notepad project file. Extract this zip file in C:\devkitPro\examples\gamecube. Open up tutorial3.pnproj which is the programmer's notepad project file, expand the source directory and then click on main.c to show the source code.

So in main.c we have our usual include statements, our screen variables and all of the initialisation of the video and controller in the Initialise function. Below the initialise function we have our main function, the function that is always run first, which calls Initialise to set up the video/controller and then print out "Hello World"; all stuff we should already know. Just in case you are wondering the \ n in the printf statement is actually a line break. If we were to print some more text, it would appear on the next line down.

If you want, you can compile the source code (Alt + 1) and as usual you will see two files appear (tutorial3.dol and tutorial3.elf). The makefile that was included in the zip file uses the gamecube rules, so we can actually run tutorial3.dol on gcube. Double click to run tutorial3.dol and associate it with gcube (C:\devkitPro\emulators\gcube\gcube.exe). This will then bring up gcube and show the "Hello World!" text which has made things a lot easier for us. You can now close down gcube.

There is a slight difference on how we call the controller functions between the gamecube and the wii which you may have noticed. The difference is that for gamecube we have statements that start with PAD_ whilst on the Wii the statements start with WPAD_ (W for Wii).

So let's begin with capturing the gamecube controller inputs. First things first, we need to scan for any input which is done by using PAD_ScanPads();. If you don't have the call to this function then the controller will be useless.

There are 3 different button states which include whether the button has been pushed (PAD_ButtonsDown) if a button has been held down (PAD_ButtonsHeld) and if a button has been released (PAD_ButtonsUp).

In this example we want to check if a certain button, say the A button is has been pressed, in this case we would use the PAD_ButtonsDown as shown below.

```
u16 buttonsDown = PAD_ButtonsDown(0);
```

We are assigning the variable buttonsDown with the function of PAD_ButtonsDown. The 0 in brackets represents the controller number, 0 for the first controller, 1 for the second controller, etc.

Now we need to check if the A button was pressed by doing an if statement, firstly checking if the buttonsDown variable has a value and if the key that was pressed was the A button.

```
if( buttonsDown & PAD_BUTTON_A ) {
    printf("Button A pressed.\n");
}
```

You could say that the buttonsDown variable acts like a Boolean; if no button was pressed the variable remains as 0 (false) and if a button was pressed the variable will contain a value (true). So if a button was pressed and that button was A, it will print “Button A pressed.”

We can change button A to any of the other buttons which can be found in “C:\devkitPro\libogc\include\ogc\pad.h” The pad.h file lists all the buttons and functions which relate to the controller.

So now we want to check if the A button is held down and when the A button is released which is shown below. The code is similar to the code supplied above, except that we are using the different function names and assigning them to different variables.

```
u16 buttonsHeld = PAD_ButtonsHeld(0);
if (buttonsHeld & PAD_BUTTON_A ) {
        printf("Button A is being held down.\n");
}

u16 buttonsUp = PAD_ButtonsUp(0);
if (buttonsUp & PAD_BUTTON_A ) {
        printf("Button A released.\n");
}
```

We have access to the 2 joysticks on the gamecube controller by using PAD_SubStickX, PAD_SubStickY, PAD_StickX and PAD_StickY. When using the joysticks you don’t need to call buttonsDown or anything else.

What you want is to define how much movement of the joystick relates to that input being detected in your application. Use a lower number to detect the slightest movement and a higher number to detect harsh movements. For example, if we wish to detect movements of the joystick when the joystick is moved right to the top and bottom we should use -18 and 18 as shown below. Remember in an x/y axis, negative y is down, positive y is up, negative x is left and positive x is right.

```
if (PAD_StickY(0) > 18) {
        printf(“Joystick moved up.\n”);
}

if (PAD_StickY(0) < -18) {
        printf(“Joystick moved down.\n”);
}
```

So what we are doing is checking if the joystick has been moved up or down and if so we print out which way it has been moved. Note that for the joystick it’s like the function PAD_ButtonsHeld, it will keep printing out moved up or moved down until you let go of the joystick.

Now all we need to do is place all the above into a while statement so the controller state is continually scanned and the if statements can be run. We'll also add in the if statement to detect when the start button is pressed, so it can terminate. Below is our final code (tutorial3-gamecube) with all of our modifications. Place this code after the printf("Hello World.\n");

```
while(1) {

	PAD_ScanPads();

	u16 buttonsDown = PAD_ButtonsDown(0);
	if( buttonsDown & PAD_BUTTON_A ) {
		printf("Button A pressed.\n");
	}

	u16 buttonsHeld = PAD_ButtonsHeld(0);
	if (buttonsHeld & PAD_BUTTON_A ) {
		printf("Button A is being held down.\n");
	}

	u16 buttonsUp = PAD_ButtonsUp(0);
	if (buttonsUp & PAD_BUTTON_A ) {
		printf("Button A released.\n");
	}

	if (PAD_StickY(0) > 18) {
		printf("Joystick moved up.\n");
	}

	if (PAD_StickY(0) < -18) {
		printf("Joystick moved down.\n");
	}

	if (buttonsDown & PAD_BUTTON_START) {
		exit(0);
	}
}
```

Hit compile (Alt + 1) and then run the tutorial3.dol file again. The gcube A button is the Q key on your keyboard and the joystick up and down is the 8 and 5 on your keypad. Try pressing the A button and you will see it'll print that the key is down and that it's being held. Release the key and it will print that the A button has been released as we expected. Press the up and down on the joystick (8 and 2) and you'll see that it will say repeatedly say it's moving up or down.

So lets now go now and compile our project for the Wii instead of the gamecube. In the makefile, firstly change the gamecube_rules to wii_rules in line that reads:

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

Now we need to add the Wii and the Bluetooth libraries to our compiler. Modify the libs line to read:

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

Save the makefile and open up main.c.

In the include section include the wiiuse file:

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

We will need to comment out any PAD_Stick statements since the Wiimote doesn't have a joystick on it. So comment out lines if (PAD_StickY(0) > 18) { to printf("Joystick moved down.\n"); } as shown below.

```
/*if (PAD_StickY(0) > 18) {
        printf("Joystick moved up.\n");
}

if (PAD_StickY(0) < -18) {
        printf("Joystick moved down.\n");
}*/
```

The last thing we need to do is change all instances of PAD_ to WPAD_ which is easy enough as well as changing the PAD_BUTTON_START text to WPAD_BUTTON_HOME.

Your main.c file (tutorial3-wii) should look like the one found below:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <malloc.h>
#include <ogcsys.h>
#include <gccore.h>
#include <wiiuse/wpad.h>

static u32 *xfb;
static GXRModeObj *rmode;

void Initialise() {

 VIDEO_Init();
 WPAD_Init();

 rmode = VIDEO_GetPreferredMode(NULL);

 xfb = MEM_K0_TO_K1(SYS_AllocateFramebuffer(rmode));
 console_init(xfb,20,20, rmode->fbWidth,rmode->xfbHeight,rmode->fbWidth*VI_DISPLAY_PIX_SZ);

        VIDEO_Configure(rmode);
        VIDEO_SetNextFramebuffer(xfb);
        VIDEO_SetBlack(FALSE);
        VIDEO_Flush();
        VIDEO_WaitVSync();
        if(rmode->viTVMode&VI_NON_INTERLACE) VIDEO_WaitVSync();
}

int main() {

        Initialise();

        printf("Hello World!\n");

        while(1) {

                WPAD_ScanPads();

                u16 buttonsDown = WPAD_ButtonsDown(0);
                if( buttonsDown & WPAD_BUTTON_A ) {
                        printf("Button A pressed.\n");
                }
```

```
		u16 buttonsHeld = WPAD_ButtonsHeld(0);
		if (buttonsHeld & WPAD_BUTTON_A ) {
			printf("Button A is being held down.\n");
		}

		u16 buttonsUp = WPAD_ButtonsUp(0);
		if (buttonsUp & WPAD_BUTTON_A ) {
			printf("Button A released.\n");
		}

		/*if (PAD_StickY(0) > 18) {
			printf("Joystick moved up.\n");
		}

		if (PAD_StickY(0) < -18) {
			printf("Joystick moved down.\n");
		}*/

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

	return 0;
}
```

Clean (remove) the build folder and the elf/dol by pressing Alt + 2 and then recompile the source again. Now you can rename tutorial3.elf to boot.elf and place it in a directory called tutorial3 under the /apps/ directory of your SD card and Load it with the homebrew channel. Now you have a homebrew application that interacts with the Wiimote.