This is the third PSP tutorial

M5: Controls revisited

Hi everybody,

This tutorial is all about controls. For this and all tutorials I assume that you have some C++ experience and that the basics are clear to you. This tutorial will show output to see what controls you have pressed. You need to use the source code of tutorial 1 for this tutorial.

The implementation

We are going to implement every button we can use in this tutorial. I will also explain somethings about them. Lets take a look at the new code in the Controls() function of the game loop:

GameApp.cpp

This code is in the load function. We have to enable the use of the analog stick:

```
// turn on the analog stick
sceCtrlSetSamplingCycle(0);
sceCtrlSetSamplingMode(PSP_CTRL_MODE_ANALOG);
```

We set the cycling to 0 and then turn on the analog mode. The 0 of the cycling is used normally but play around with the values and see what happens.

Well now lets take a look at the Controls() function.

```
int GameApp::Controls() {
   // reading controls
   SceCtrlData pad;
   sceCtrlPeekBufferPositive(&pad;, 1);
```

The first two lines of code retrieve the controldata into the pad variable. I use sceCtrlPeekBufferPositive instead of sceCtrlReadBufferPositive because the latter waits for the VBlank and the first does not. Thus the first is faster, which we want.

```
   float movementHorizontal, movementVertical;
   // getting the analog stick
   movementHorizontal = (float)(pad.Lx)-128.0f;
   //pspDebugScreenPrintf("Horizontal = %f\n",movementHorizontal);
   movementVertical = (float)(pad.Ly)-128.0f;
   //pspDebugScreenPrintf("movementVertical = %f\n",movementVertical);
```

In these five lines we get the analog values. In this tutorial I have commented these debug lines because the will be shown every frame and then you don't see the button pressing that well anymore. So If you want to see the values uncomment these two lines and start using the analog stick :).

I have subtracted 128 from the values retrieved to let the center be at 0,0 ranging from -128 to 128. In games this can be handy when you use it for turning or walking.

```
   if (pad.Buttons != 0){
      if (pad.Buttons & PSP_CTRL_LEFT){
         pspDebugScreenPrintf("Pressed the left button...\n");
      }
      if (pad.Buttons & PSP_CTRL_RIGHT){
         pspDebugScreenPrintf("Pressed the right button...\n");
      }
      if (pad.Buttons & PSP_CTRL_UP){
         pspDebugScreenPrintf("Pressed the up button...\n");
      }
      if (pad.Buttons & PSP_CTRL_DOWN){
         pspDebugScreenPrintf("Pressed the down button...\n");
      }
      if (pad.Buttons & PSP_CTRL_CROSS){
         pspDebugScreenPrintf("Pressed the cross button...\n");
      }
      if (pad.Buttons & PSP_CTRL_SQUARE){
         pspDebugScreenPrintf("Pressed the square button...\n");
      }
      if (pad.Buttons & PSP_CTRL_TRIANGLE){
         pspDebugScreenPrintf("Pressed the triangle button...\n");
      }
      if (pad.Buttons & PSP_CTRL_CIRCLE){
         pspDebugScreenPrintf("Pressed the circle button...\n");
      }
      if (pad.Buttons & PSP_CTRL_START){
         pspDebugScreenPrintf("Pressed the start button...\n");
      }
      if (pad.Buttons & PSP_CTRL_SELECT){
         pspDebugScreenPrintf("Pressed the select button...\n");
      }
      if (pad.Buttons & PSP_CTRL_RTRIGGER){
         pspDebugScreenPrintf("Pressed the Right trigger button...\n");
      }
      if (pad.Buttons & PSP_CTRL_LTRIGGER){
         pspDebugScreenPrintf("Pressed the Left trigger button...\n");
      }
   }

   return 0;
}
```

This lines check which buttons are pressed. If the button is pressed the debug print function is called.

Compile the code and look what happens :D. When you press a button the game application puts out the text which tells the button which is pressed. This is not the most ideal way of using the controls but we will overcome this problem in the tips and tricks chapter. Please note also that if you work in Kernel Mode that you can also use: PSP_CTRL_NOTE, PSP_CTRL_SCREEN, PSP_CTRL_VOLUP, PSP_CTRL_VOLDOWN, PSP_CTRL_DISC, PSP_CTRL_WLAN_UP, PSP_CTRL_REMOTE, PSP_CTRL_MS.

Source files

Download the source files.

Tips and tricks

The above code just prints out debug text when a button is pressed, however there are some techniques useful for games which I will explain here.

Only handle the press once

The code in this function:

```
if (pad.Buttons & PSP_CTRL_CROSS){
   pspDebugScreenPrintf("Pressed the cross button...\n");
}
```

This debug output is printed every frame when the button is pressed. Since framerates of 60 fps are not uncommon, pressing the button and catching it this way results in a multiple call to that code per press. What we want is that when we press it the code is only executed once until your finger releases the button again. To do this we use this code:

```
if (pad.Buttons != 0){
   if (pad.Buttons & PSP_CTRL_CROSS){
      if (ButtonCross == 0) {
         ButtonCross = 1;
         // call some code
      }
   }
   else { ButtonCross = 0; }
}
else {
   ButtonCross = 0;
}
```

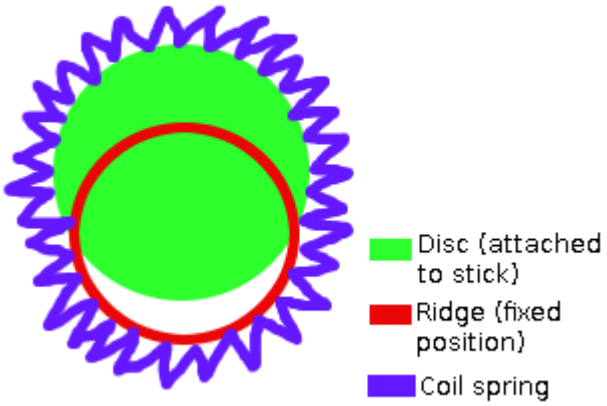
As you can see the code will only be executed if ButtonCross is 0, but as soon as we enter the function it is set to 1. As long as pad.Buttons is not 0 and the Cross button is pressed it runs that code but then ButtonCross is not 0 anymore so the code past that won't be executed. Whenever you do not press the Cross button or do not press any button at all the value is set to 0 again. This way the code is executed only once when you press the button.

Using the analog controller

As you can see probebly when using the above tutorial, the analog is not always zero when released to its center. This can be very annoying when you don't know this. For example: If you create a racing game but you still steer when yo udon't use the analog stick. That can be very annoying. Here is what exactly happens with the analog stick:

Quote mc (ps2dev forums): "*The analog stick is of a very simple construction. The stick itself is fixed to a small plastic disc, which rests inside of a circular coil spring. The spring is suspened around a plastic ridge, that the disc can slide under, but not the spring. (See the diagram below.) When you push the stick in some direction, the disc will slide out under the ridge, dislocating a part of the spring. Since the other side of the spring is suspended by the ridge, the spring can't follow the disc by simple translation, and will instead stretch to fit around the disc. This tightens the spring,*

producing a force to pull back the disc (and thus the stick) when you let go. However, as the disc slides back, the spring contracts, resulting in less and less force being applied. When the stick has reached a certain point (+20, or whatever), there spring will have contracted so much that there is not enough stored force left to overcome the friction between the disc and the ridge. So the lack of good centering is the result of cheap design. Not _bad_ design, just cheap. "



So if it is not so exact we should use that also. What I did was creating a deathzone as I like to call it. For example, when the analog value is lower than -32 or higher than 32 only use the value in that case.

```
movement = (float)(pad.Lx)-128.0f;

if (movement < 32.0f && movement > -32.0f) {
   // run code
}
```

This way you don't do anything with the value when you don't use the analog stick.