(These notes may not correspond exactly to my notes in the ppt)

G'morning! I'm Joachim Bengtsson, and this here is Frank Buß. We're here to talk about the open source project LuaPlayer, which is originally Frank's creation and is currently my own pet hobby.

What is LuaPlayer?
LuaPlayer is basically a runtime environment for the script language Lua, running as a homebrew application on the Sony Playstation Portable. What this means is, while hackers made writing homebrew apps for the PSP possible quite a while ago, it just got a whole lot easier. Actually, so much easier, that some people are using it as their *first* venture into programming! Who would've thought? I mean, console programming is supposed to be one of the most intricate and difficult things you could code. Not to mention coding for *portable* consoles!

Before we move on, let me give you a quick taste of LuaPlayer.

A quick taste of LuaPlayer...
Here's a Hello World-style application, straight from the bundled samples. It's pretty straightforward. Lua, the language, isn't necessarily object oriented, but it can be. New is a class method of Color, which we call to get a color green. We draw a string to the screen with that color. Next, we iterate from zero to twenty to draw some fancy lines, also to the screen. We flip the backbuffer to the visible screen. We loop forever, or until the start button is pressed. Inside that loop, we just wait.

That's it. Running that code on a PSP will give you this. Pretty fancy, I'd like to say, especially on that mighty screen.

PSP and its open-source API
I said before that console programming was supposed to be difficult. Before LuaPlayer, It still was. Let me show you why, by looking at the PSP and its open-source architecture.

Not very long after the PSP was released, a guy called nem over at p2dev.org said, "hey, I know this processor!", as it's not too different from the Playstation 2. He started writing code for it. Others quickly followed suit, and they began reverse-engineering the API, manually crafting header files and whatnot. Reverse engineering isn't all that easy, and I'm not entirely sure they always know what they're doing...

```
    sceUtilityGetSystemParamInt(PSP_SYSTEMPARAM_ID_INT_UNKNOWN,
                  &dialog->buttonSwap); // X/O button swap

    dialog->unknown[0] = 0x11;      // ???
    dialog->unknown[1] = 0x13;
    dialog->unknown[2] = 0x12;
    dialog->unknown[3] = 0x10;
```

;)

Remember that example we began with, that hello world-with-fancy-lines? Let's try something *easier* than that. It's an input demo from the 'official' unofficial homebrew SDK for the PSP.

We begin by including some basic header files and defining some necessary stubs.

Then, we have to define a callback thread, so that we can actually quit the app after us.
Here comes our actual code. Or, first we have to init and setup the callbacks and setting sampling mode for the key input... But after that! After that we can start polling keys and printing all we want. When we're done, we clean up.

There! Simple, wasn't it?

Ehrm. And that didn't even include any graphics. I'm not even going to go through all.. the.. code... needed... for... that...

As you've seen, all this is pretty complicated. Never the less, the PSP is a very powerful console. It has a brilliant display, lots of CPU power, and hardware support for 3D rendering. There's UDB, Wifi, a serial port interface, and you can add several gigabytes of storage with memory sticks. It has sort of a mini-DVD drive, called UMD, which can hold 1.8 GB of data.

I wanted to use that power, but I thought that it needn't be so complex. I didn't know how to make it easier, though. That is, until I saw this project by Frank here called "LuaPlayer" being announced on the PS2Dev forums. It was simple but brilliant. All of this complexity was hidden behind a veil of Lua, a language that I quickly set out to learn and love. It's also the language that *you*'ll get to learn the basics of now.

Lua, the programming language
Lua is a highly versatile language, because it's so simple. You have numbers, strings, booleans, userdata and tables. That's it.

If you've ever programmed before, the basic syntax of Lua should be familiar. Function calls look as you'd expect. If you're a C head, adding semi-colons is all okay. There's assignments and comparisons, conditionals and iteration. Like in pascal, blocks begin implicity but ends with 'end'.

The *complete* BNF for Lua is shown here behind me, and it's very short, which is probably the main reason why Lua so easy to use.

But don't think for a second that Lua is a toy language, or that you couldn't implement complex applications with it. In combination with the standard libraries and the table concept, it is in fact easier to solve a given problem with Lua than with for example C.

You can, however, call C functions from Lua, whenever you need a lot of speed, or hardware access, as in the case with LuaPlayer.

Behind me is a list of Lua's key features.
Lua has a very simple syntax. It's also dynamically typed and garbage collected, so you'll never have to worry about types or memory leaks. It has the very poweful concept of tables. Functions are first-class objects, like in Python. It's very easy to integrate with C and C++ projects, there are even tools to do that pretty much automatically. Lua has some very powerful procedural features, like coroutines and upvalues, but those won't be covered in this talk.

Let's first take a look at Lua's most prominent feature: tables. The way you use Lua tables resemble PHP arrays or Python dictionaries. You can index them with numbers, or strings, or even functions, as they're backed by hash tables. There's some syntactic sugar to make indexing and table creation easy and natural.

For creation, you can do it curly brackets-style, index it with integers like in C, index PHP style with strings, or use it as a dictionary. Of course, tables can be nested.

Now, here's the cool part -- indexing a table with dot notation. When seeing that, some of you can probably figure out how Lua does object orientation.

Because you see, as I said before, functions are first-class objects. Thus, we can use lambda constructs, like in Lisp and Python, like so.

```
foo = function(x) print(2*x) end
table = { 7, foo }
table[2](table[1])  -->  14
```

Notice that function foo(x) is just syntactic sugar for foo = function(x)

That brings us to metatables. A metatable event is a value that you can set to any table to give it functionality in a way that resembles operator overloading in C++. Let me show you an example.

Defining the __add event to vectorFuncs allows us to use the plus operator on two copies of vectorFuncs. The setmetatable call copies all the metatable events from vectorFuncs onto v1. Now we can add vectors, and print them, too, with that tostring function.

Lets take this one step further, and make it a full-fledged object-oriented Vector class. The __index metatable event lets you define where lua is to look if the object is indexed for a value it doesn't have itself. In the example behind me, this means that any table that is created with Vector.create, that is, any table that has Vector as its __index function, will use Vector's values and functions if they aren't available in the table itself. In practice, this means that you've just created an instance! Or at least, almost. Notice this part. The colon operator is syntactic sugar for adding self as the first argument to any function call.

Inheritance and many other concepts come naturally and easilly with this metatable concept.

Combining the two

Now, the true power of Lua comes with a decent C or C++ backend. Lua itself is powerful, but in the end, it is an interpreted language. Plotting pixels one at a time with lua, renders you about 80 000 pixels per second, less than one frame per second. Doing it in pure C gives you 72 *million* pixels, or about 500 frames per second. Of course, you have to find a balance there, finding an interface that suits your application domain. LuaPlayer is designed for simplicity and not speed. It can

draw sprites, primitives and text fast enough for a sidescroller.

Here's a nice diagram that Frank threw together. At the top we see the lua script being executed. It's saying to draw a line, simply. The script is interpreted by the Lua library interpreter, which converts it to a C function call. This C function then talks directly with the screen memory buffer, the kind of stuff that ordinary PSP C hackers have to do at one time or another.

Programming for LuaPlayer

There are 5 modules in Lua Player, which are implemented in C and which can be called from Lua Scripts: Graphics, Controls, Timer, System and Sound.

The graphics module provides image and color objects.You can load PNG images or create offscreen images and draw parts or the whole image with or without alpha channel to other images. The screen object is just an image object, which means you can draw images to screen the same way you do for other images.

The Controls module lets you read the buttons and the analog stick by polling.

The timer's a basic millisecond timer.

System handles file operations, battery, usb, and serial port operations.

The Sound and Music module lets you play whatever the mikmod library can play, which is some mod formats and wave sound.

Let me show you another example of some LuaPlayer code, now that you're Lua experts. ;)

LuaPlayer uses double buffering, which means that whatever you draw to the screen object, isn't show until you "flip" the backbuffer. Behind me is an example of how you could do animation with a moving object on a background.

We use the functions of the Graphics module to load the sprite images, a background and a smiley face, into memory. These will be automatically released when they're not needed anymore. We loop forever, and within that loop, we draw the background to the screen object, calculate the new coordinates for the smiley face, and then draw the face to those coordinates. Next, we flip the screen object onto the real screen. And we check the controls if the user wants to quit using the start key.

Community response

The combination of the beautiful PSP and the simple Lua was just great. There's a popular site for PSP homebrew news called pspupdates.com. After LuaPlayer was released, suddenly at least half of the news items were about small LuaPlayer nuggets, or even whole games, people had thrown together. The impressive part is that these are pretty complete games, most of them written in just a few days, on top of an environment that was born just three months ago! It's astounding. Let me show you some of the creations.

There’s a file browser, classics like snake, minesweeper and pong, wedgewars, psp air hockey, the highly addictive port of Dr Mario, Pyramid Panic: a platform puzzle game, a port of sokoban, dots, breakout, and bomberman.

And check this out. This is Notepad, it's a notepad application, with an onscreen keyboard, it got menus and dialogs and everything. Pretty great.

We got a mail just last week from a 13 year old kid, who wanted to thank us for introducing him to the world of programming. There's something strangely motivating about writing code for something as cool as the PSP. Equally motivating is the simplicity of both Lua, the language, and LuaPlayer's API.

Live Demo
Next up is a live demo. Frank, show us your Ninja stealth Lua moves.