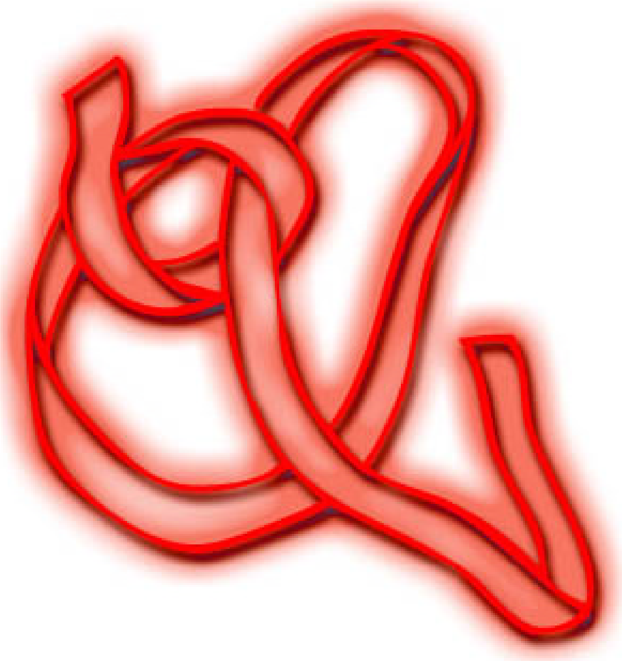
SMART TALKING NODES



superfluxus media

SmartTalkingNodes

Introduction

The basic idea, which led to this program, was the dissatisfaction with the available means of creating complex dialogues. Theoretically, all this can be done in CScript, but it is hard to imagine that anybody would bother to write all that code.

In particular when it comes to tree structures that extend the amount of one or two written pages with a few lots of bracketed brackets, it seems just a masochistic tendency doing that.

If one has understood the power of recursion, which is the very core of tree programming, it is obvious that the only viable way is a language like C++ and a tool like the treeview of MFC.

Bringing this together it's a story as such, but not of common interest.

The program is done.

It consists of tree parts:

a) a Gamestudio demo project, which is merely for demonstrational purpose

b) the SmartTalkingEditor (the piece you are working with in this moment)

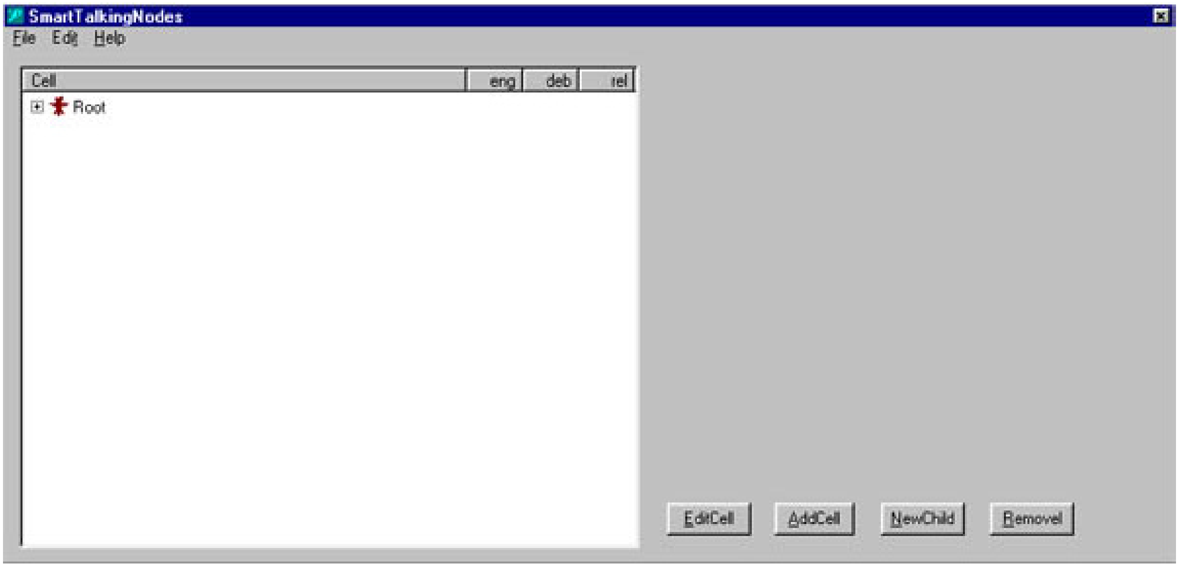
c) a set of files which you will need if you want to embed the dialogue system into your project

I. The Editor

The first thing you want to want to do is to write a dialogue – and here you will encounter the *SmartTalkingNodes*- editor. In a way it look quite familiar – like any explorer. But nonetheless it should be explained in detail.

A dialogue knows two partners, you and the machine. Who comes first? None of them. The first layer is occupied by the root – which cannot be modified. A *nod has just one root – or the other way round: any root stands for a *nod file. Actually – you can edit just one *nod file in the editor (maybe I am going to extend that, maybe not).

Who comes then? The humane player or the machine? I decided that the machine is starting the dialogue, at least the layer 1 is reserved for the machine. There is no real necessity for that order – and you will be able to circumvent it, if you want the player to start a dialogue.



A. The dialogue

Ok, let’s write a dialogue.

We want a new file, so we choose the options *new* and there’s just a root left.

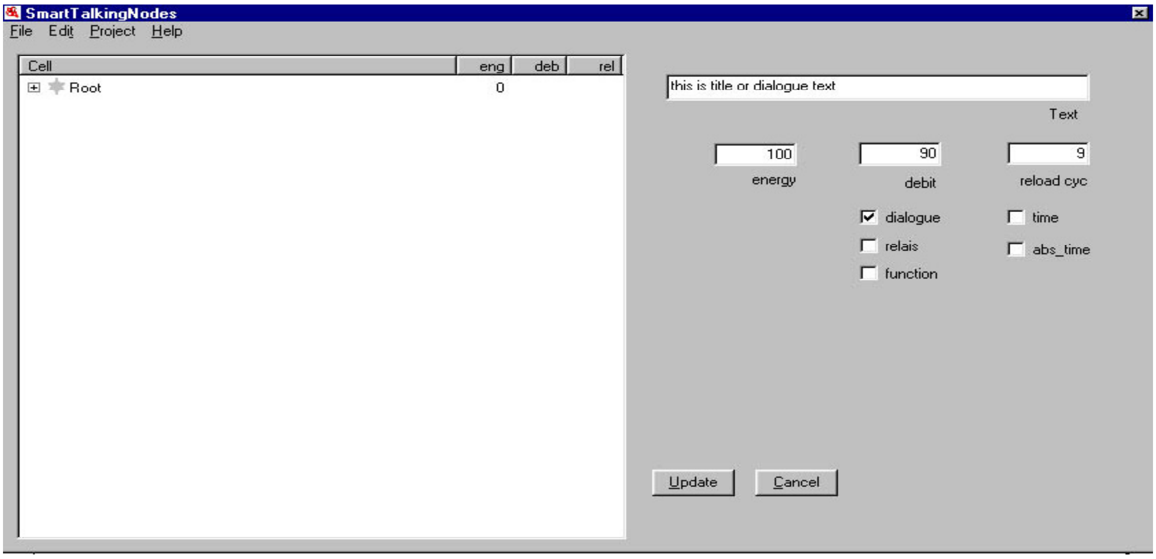
When we click on the *NewChild* button, the dialogue offers some option.

We type in the text (say ”I greet you, my dear”) and choose the *Update* button – and there it is.

You might think of using the *AddCell* button instead – which does not work. But why not? *AddCell* just adds another brethren (“I don’t like your face, get out of here”) of the double-clicked starting point. Since we have allowed only one root, there is no chance of adding another.

But it will work when we double click our new cell – and use the *AddCell* way instead.

In a way the differentiation between *NewChild* and *AddCell* is a little redundant, one could live well with the *NewChild* button.



Cell	eng	deb	rel
⊞ Root	0		

Text: this is title or dialogue text

energy: 100

debit: 90

reload cyc: 9

☑ dialogue
☐ relais
☐ function
☐ time
☐ abs_time

So – this is easy to do. You can create dialogues of unlimited size. The only limitation there is the depth of the dialogues. It is limited to 64 layers – but in layer 64 you could use a function to step to another dialogue or to another branch of tree – so this is not a real limit.

You might ask why we want the machine to have two options.
Psychologically there is no argument against it, the machine might know hundreds of states, from hostility to passionate love. So the question, more precisely articulated, goes: who decides if the machine gives an hostile or charming welcome.

It's you. And you do by means of the other values – you have seen, by means of energy, debit, and cycle reload.
This seems very dark on the first glance, so you should read the following lines (which are, in a way, a more structural approach).

II. The energy system

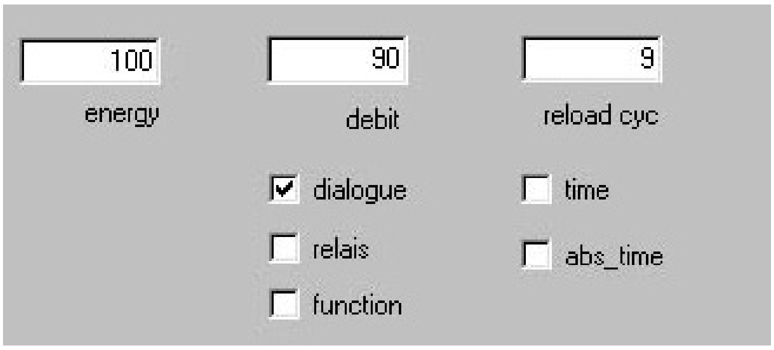
Ok, we all know nodes, we know our file explorer. There is nothing special about it. - Yes, you're right, but in terms of programming tree structures (or linked lists, as they are called) are pretty complicated. But I should not worry. It's done after all.

You have the *SmartNodes Editor*, which is similar to what you know (with a few inconveniencies – and a lack of documentation).

For example, how easy it is to control the keys with the keyboard.

Just use the right arrow – and the tree will expand.
Use the left one – the tree will compress.

But let’s come to our subject, the question, why there are so strange input fields like energy, debit and reload cycle.



100 energy

90 debit

9 reload cyc

☑ dialogue
☐ relais
☐ function

☐ time
☐ abs_time

First: take a look at a metaphor. Think of each node as a living cell, or better, think of it as a neuron which is charged and decharged constantly. This is what I would consider to be energy – and happens when the machine explores a level of various nodes.
It looks for the one which has the biggest resistance. This is the one to make the intruder halt. There’s a collision – the living cell, the node, does what we want it to do, but doing it loses energy. It is discharged.

The specific amount of energy loss is, what you input into the field *debit*. The moment an action takes place, this sum is debited (so it works like a normal bank account).

Remains the last term: *cycle reload*. In contrast to our bank accounts, the sad rules of economy, I prefer the idea of a self-nourishing, self-healing monad. So our node is able to gain energy – up to the maximum which is set by the resistance value. Each time the program encounters the node, it’s energy will gets recharged a little bit. And the term *cycle reload* signifies the number of times it needs to get recharged again.

But what is the sense of that?

Imagine the following situation. You’re in a game, running two and fro, and the man at the corner always greets you with the same old words. Wouldn’t it be nicer if he just noticed, that you passed a dozen times – and that he would change his attitude.
The first time he might say *Good morning, Sir*, next time, *Nice to see you* – and the third time, in a chummy way: *Oh, there you are again*. What does this have to do with energy system?

Simple: it allows you to program that. The formal *Good morning* might have a high debit value and the biggest resistance (we are always formal if we don’t know the other) , say

> Good morning, Sir
>
> resistance 500
>
> debit 450

We would a high number of reload cycles, say ten.

The *Nice to see you* could have a resistance of 200 – and a smaller debit, let’s say 100. Reload cycles 5.
The first time we pass the greeting procedure we would encounter the formal answer (which is discharged afterwards). And what happens? Now the informal *Nice to see you* with a resistance value of 200 (in comparison to the remaining 50) is Number One.

In consequence this system allows you to dynamize the reaction of the system. It’s up to you and your finetuning.

For a while I had the idea to call the program *Breathing Trees*, which is quite a good description of this pulsating and ever moving machinery – but I fear nobody would have understood. But I should do it anyway.

III Why SmartTalkingNodes could serve as a general problem solver

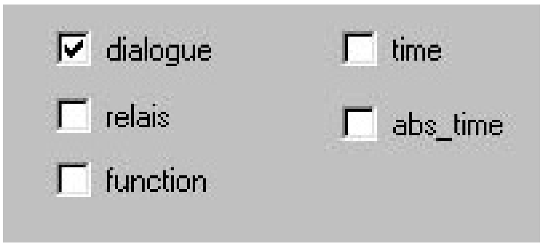
That’s for the energy system. You might wonder about the three option buttons.

What gets obvious here is that SmartTalkingNodes is not only meant to create dialogues, but also serves as e general programming tool. You can use it to control a decision tree of functions.

This is understandable, but what is a *relais*? French word for *relays*.

This gets a little complicated – and you have to think about what happens when a decision is made.

When a communication process takes place, some kind of agent travels through the tree. In a given tree he looks for the cell with the biggest resistance – and chooses it. Imagine that this cell would be a relais instead of a piece of text. What happens? The agent would enter the level below (and since there can be as many option as you are eager to write, the search process begins anew and the candidate with the biggest resistance is chosen – which could be once again a relais).


☑ dialogue
☐ relais
☐ function
☐ time
☐ abs_time

What the relais does – it gives you, instead of an item , another tree, so it is comparable to a folder (and the relation of a dialogue item to a relais is that of file to folder).

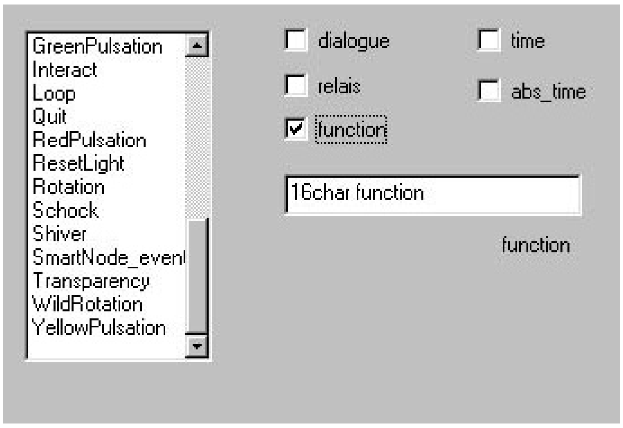
Let’s come to the function option. We have already that there are different graphical representations of cells – the functional one is a grey star.

A function can serve in twofold way.

First: it feeds back a dialogue item plus a function. In the demonstration you always here a beeper sound. This is a wdl function – which is called by the dll.

The entry would look like this

Only if you click the function option you will have the additional function field where you can type in a functions name.


GreenPulsation
Interact
Loop
Quit
RedPulsation
ResetLight
Rotation
Schock
Shiver
SmartNode_event
Transparency
WildRotation
YellowPulsation

☐ dialogue
☐ relais
☑ function
☐ time
☐ abs_time

function: 16char function

It could be any WDL function you have written yourself. The only thing which is important is that it exists. Otherwise it will not be processed.
To make this errorpoof, I have integrated a nice feature - the import of wdl files (see later).

After we learned this, it is no big stress to our imagination to fancy hundreds of ways we could use this.

For example: My mechanical partner could not only inform me about this or that but he would make the corresponding movement, he could nod his head, roll his eyes, whatever. Since one can integrate dialogue and function there is no problem doing that.

When you understand SmartTalkingNodes not as a tool for dialogues, but for programming, you will realize that you can do nearly anything with it.

You do not have to write all these awful *if*-brackets , which make the code so illegible, but you could this tool as a kind of strategic general's hill, the point that gives you an oversight over your battle.

IV SmartProcesses

We started wth dialogues. This is one way to use SmartTalkingNodes – but it's not the only one.
I have to admit that I deeply setimate conversational skills, that I passionately love language – but there's time for action also.

So the dialogue side of the program is just one – you can use it, as mentioned, as a tool for controlling processes.

In the Gamestudio action set you will see the differenciation. There are *SmartDialogues* and there *SmartProcesses*. We will deal with the latter now.

In the current state of the engine (1.1) there is just one type of process you can control.
It just begins with the root and travels his way down until the end is reached.

Although this process might be very sophisticated (it follow the energy system and therefor gives you an endless variety) I would call it a mechanical process. It's basically like pressing a button – and something happens which you cannot manipulate anymore.

In the following versions there will be processes which can be manipulated at runtime (which work in dialogic way, so to speak) and – what would be the coronation – processes that interact on other processes. But that's future – and we go back to our aim of setting up a process.

For this you would not click the *dialogue* checkbutton, but the *function* checkbutton.
If you do this you will see an entry where you can type in a function.

But there is a more elegant way to deal with that.

The program allows you to import your wdl functions – and renders them available for immeadiate use. This is done by the *file_importWDL* option – and, wow, there they are.

Just select the appropriate one – and the treem item and the function are now linked.

There is a special word to say to the way you have to code your WDL functions in order to make the engine run properly.

You can basically use your self-defined function, you may (if you alter the SmartProcess action) use the *my* expression. But there is one thing to do.

In the *.wdl file you have to add add a single line to the end of the function

```
Ready(my.ID);
```

This is crucial. If yu forget to add it there might be a crash.
Why is this so? The Gamestudio engine has to signal the dll that a process is done – and this call to be executed from wdl. Maybe I will find a better solution, I don't know.

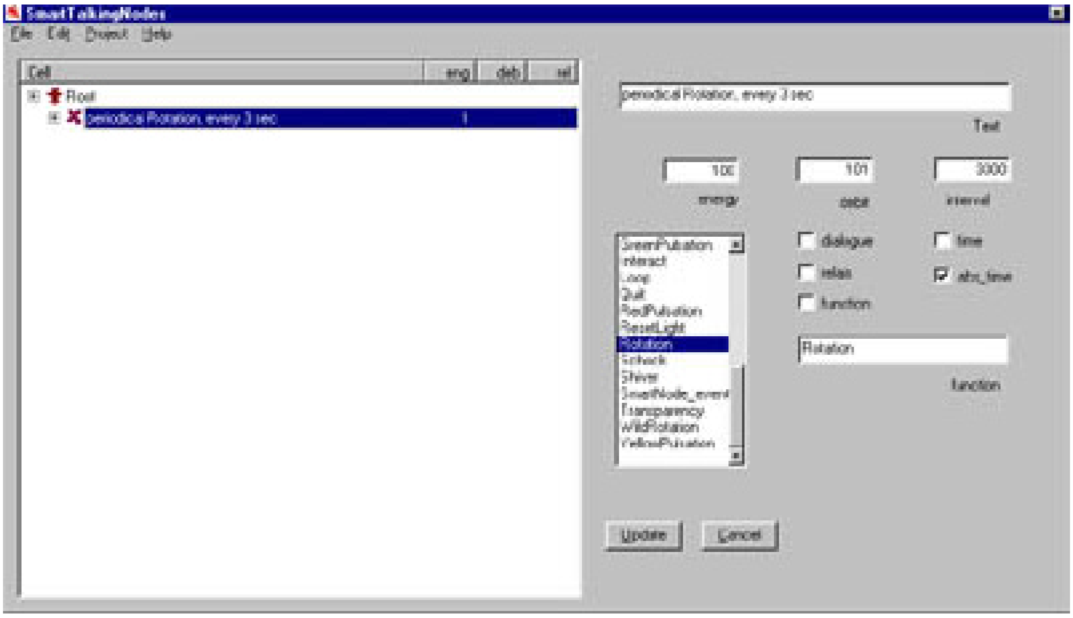
V Time control

There is something new - which needs explanation. You have – a short glance to the program window tells you – the option to work with time controlled processes.

There's a differenciation between *time* and *abs_time* processes.
Although it stands in the second play, I will begin with the latter, simply because it is easier to understand.

In a way an *absolute timer* is an alien element for this program, or better: for the underlying philosophy. It menas: that an action takes places every x millisecond. This action does not care at all about tree structure and energy – it is executed, merciless, like the tick of a clock.

So if you use it – be aware, that its position in the tree structure is without meaning. To make such an item recognisable I would recommend to place it in the first level.



periodical Rotation, every 3 sec
Text
100 energy
101
3000 interval
☐ dialogue
☐ relax
☐ function
☐ time
☑ abs_time
Rotation
function

Once we understood the concept of absolute timers, it is easy to find out what the other timer option brings. It follow the energy system – and when the time has come, the program analyses if the cell is decharged – and if it is, it is reloaded again.

What is the difference to the regular energy system.

It is very easy. Unlike the regular cells which reload in a series of steps, this timer cells make a flash, comes t zero and is recharged after a certain period of time.

We have a row of options

A b c d

With resistance values

100 66 44 33

A would be our timer cell, the others would be regular cells.

If you define a timer cell, the menu changes a little bit. You do not a reload cycle but an interval filed.
Let's imagine our timer cell A would trigger a piece of music, or maybe a news shows which starts every hour.

So we would use an interval of 3.600.0000 = 36000 sec = 1 hour

When our agent travels into this branch he would look for the cell with the biggest resistance, our *News*.
It starts and gets decharged. But then – set to 0 - it disappear for a while, exactly for one hour.
After that the cell is recharged and will get triggered if the agent encounters the branch.

The advantage of this timer cell is obvious. You may finetune the process controll with that.

VII The dll

It is quite obvious that you want to embed the functionality in your Gamestudio procject. This is quite easy to achieve – you have to add just a few lines of code.

First thing you have to do though is to copy the respective files into your folder.

These are:

SmartNodes.dll
SmartNodes.wdl

And then you have to tell your program that you want to include this:

```
include <movement.wdl>;
include <messages.wdl>;
include <menu.wdl>;      // must be inserted before doors and
weapons

include <SmartNodes.wdl>; // the important line
```

At the end of the main function please insert the following three lines:

```
// client_move(); // for a possible multiplayer game
// call further functions here...

nodes_handle = dll_open(“SmartNodes.dll”);
InitFonts(0);
Loop();
```

There’s another thing, which is important. You have to incorporate the closing of the DirectX pointer before you leave. Please add the following to your exit procedure.

```
ExitDll(0);
```

Now you have set up your environment.

VII How SmartTalkingNodes affects Gamestudio wed

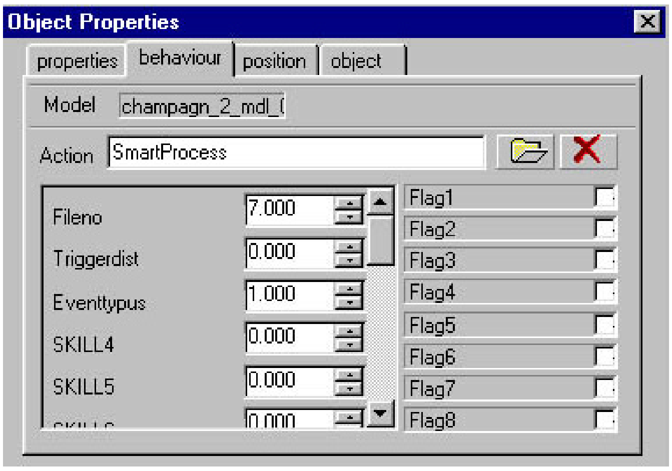
When you open up your Gamestudio project you will that there have been two new actions added to your action menu.

SmartDialogue
SmartProcess

We mentioned that – and also that you don’t have to bother about coding in WED (except addng a line to at the end of your self defined functions)

The menu needs certain information to fill in.

If you choose dialogue you will be asked to fill in a FileNumber.
It would be smart if you could tye in the *nod file you just created – but that’s not possible. So we have to identify the file by number.
Open up the SmartNodes wdl and search for “function AssignFile”.
It may look like this:



Model: champagn_2_mdl_(
Action: SmartProcess

Fileno: 7.000
Triggerdist: 0.000
Eventtypus: 1.000
SKILL4: 0.000
SKILL5: 0.000

☐ Flag1
☐ Flag2
☐ Flag3
☐ Flag4
☐ Flag5
☐ Flag6
☐ Flag7
☐ Flag8

```
function AssignFile()
{
if (my.FileNo > 7)
 {
 my.string1 = “default.nod”;
 }
else
 {
 if (my.FileNo == 1)
        {
        my.string1 = “Blue.nod”;
        }
 if (my.FileNo == 2)
        {
        my.string1 = “Yellow.nod”;
        }
 if (my.FileNo == 3)
        {
        my.string1 = “Red.nod”;
        }

 if (my.FileNo == 4)
        {
        my.string1 = “Green.nod”;
        }

 if (my.FileNo == 5)
        {
        my.string1 = “Timer.nod”;
        }

 if (my.FileNo == 6)
        {
        my.string1 = “Process.nod”;
        }

 if (my.FileNo == 7)
        {
        my.string1 = “ProcessNew.nod”;
        }


 }
```

The meaning of that is quite obvious. This function translated the number into a file’s name.
If you add your own *nod you may replace the names, or you can an unlimited number of new if clauses.
The only thing you have to do is to modify the first line also

if (my.FileNo > 7)

In our example the 7 is the maximum amount of *nod files. If you add further files set up this number.

Nex thing you should fill in (I skip field number 2) is the event_type.

0 = trigger
1 = click
2 = push
3 = shoot
4 = scan
5 = touch

If yoo choose 0 (and only then) the second field gets important – which receives the trigger distance.

Wasn’t that easy? No need for coding – just time for some creative inteligence and fantasy.

VIII The interface

As a experienced programmer might be interested to use the dialogue strings in the way you like. There’s no problem with that.

The SmartNodes.dll gives you an interface.

```
string Antwort = “Antwort”;

string Option1 = „Option1“;
string Option2 = “Option2”;
string Option3 = “Option3”;
string Option4 = “Option4”;
string Option5 = “Option5”;
string Option6 = “Option6”;
string Option7 = “Option7”;
string Option8 = “Option8”;
string Option9 = “Option9”;
string Option10 = “Option10”;
string Option11 = “Option11”;
string Option12 = “Option12”;
string Option13 = “Option13”;
string Option14 = “Option14”;
string Option15 = “Option15”;
string Option16 = “Option16”;
string Option17 = “Option17”;
string Option18 = “Option18”;
string Option19 = “Option19”;
string Option20 = “Option20”;
```

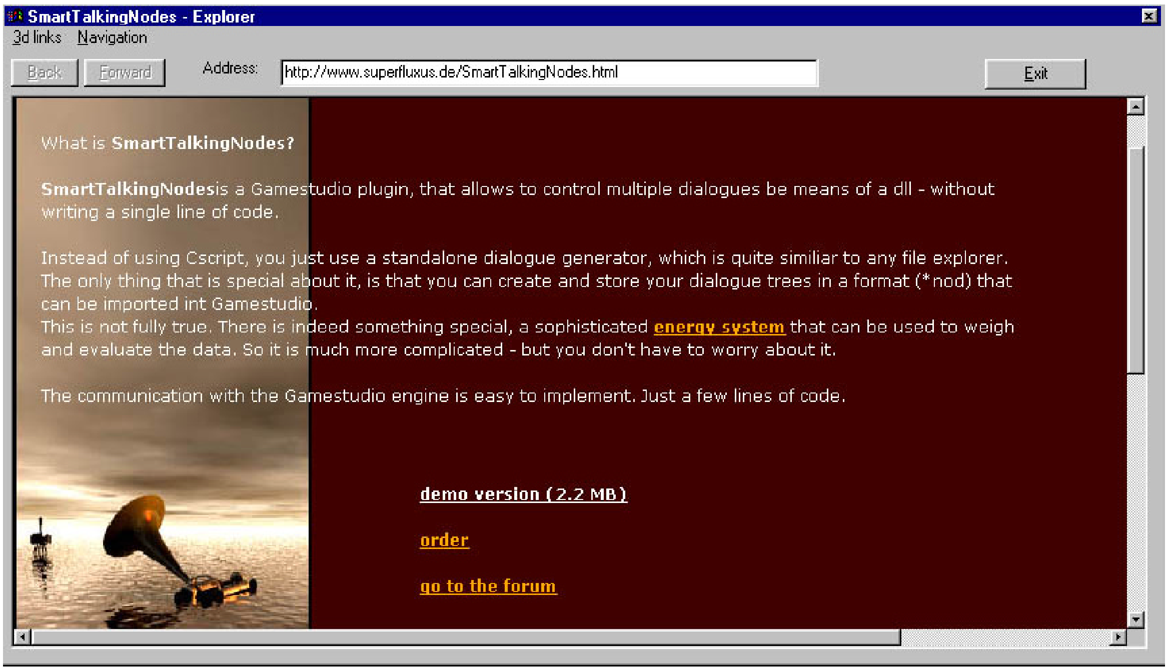
```
var op_no = 0;
```

This means: you have the answer of the machine and 20 option string plus a counter.
The counter is important because of the following reason.
Imagine that the first communication act consists of 20 option, the second of three. In the first act the dll would replace all the strings, but in the second just three of them. Option 4 to 20 would be unaltered.
By means of the counter op_no you know the number of options and have access to the full information the dll has processed.

It might be useful to add another 20 function strings- just give me a feedback if this is a desideratum and I can add that easily.

VII Will it be free?

The will be a free light version which allows just one dialogue per level. If you want full functionality you have to get registered – which is easily done and handled by ShareIt. There is a browser integrated which takes to the superfluxus site, forum, online help as well as registration.


SmartTalkingNodes - Explorer
3d links
Navigation
Back
Forward
Address:
http://www.superfluxus.de/SmartTalkingNodes.html
Exit

What is **SmartTalkingNodes?**

SmartTalkingNodesis a Gamestudio plugin, that allows to control multiple dialogues be means of a dll - without writing a single line of code.

Instead of using Cscript, you just use a standalone dialogue generator, which is quite similiar to any file explorer. The only thing that is special about it, is that you can create and store your dialogue trees in a format (*nod) that can be imported int Gamestudio.
This is not fully true. There is indeed something special, a sophisticated **energy system** that can be used to weigh and evaluate the data. So it is much more complicated - but you don't have to worry about it.

The communication with the Gamestudio engine is easy to implement. Just a few lines of code.

demo version (2.2 MB)

order

go to the forum