

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's 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 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, the answer is clear 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.

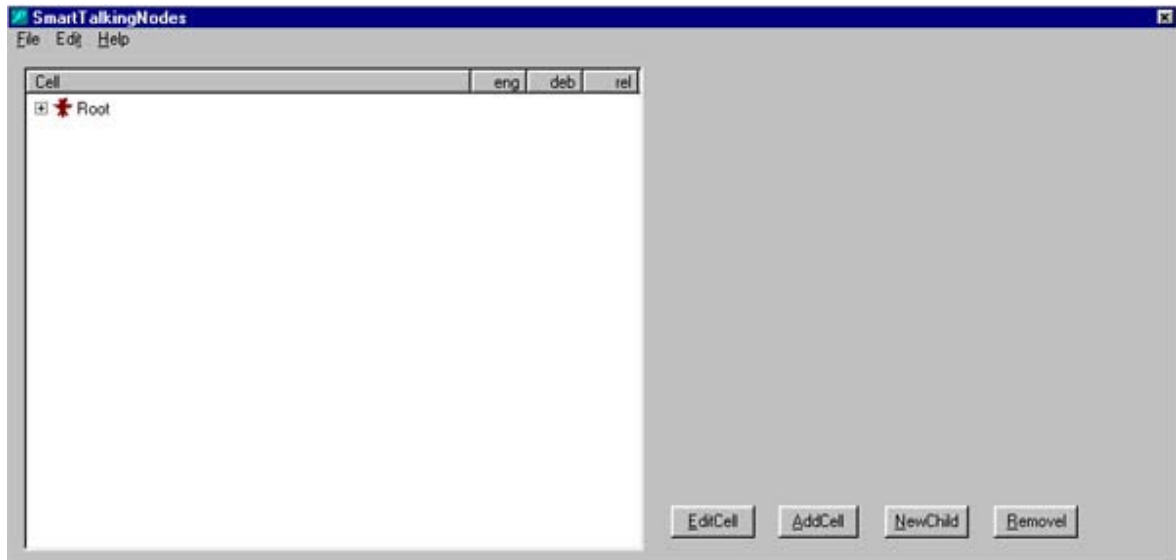
The program is done.

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.



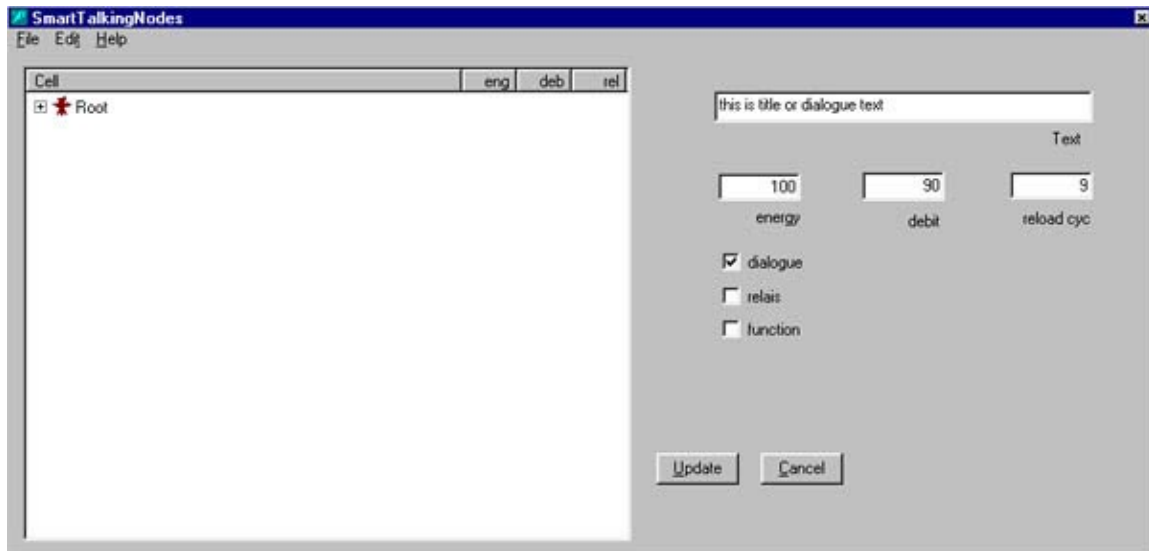
A dialogue knows two partners, you and the machine. Who comes first? None of them. The first layer is occupied by the **root** – which cannot be modified. A ***nod** has just one root – or the other way round: any root stands for a ***nod** file. Actually – you can edit just one ***nod** file in the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

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.

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 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).

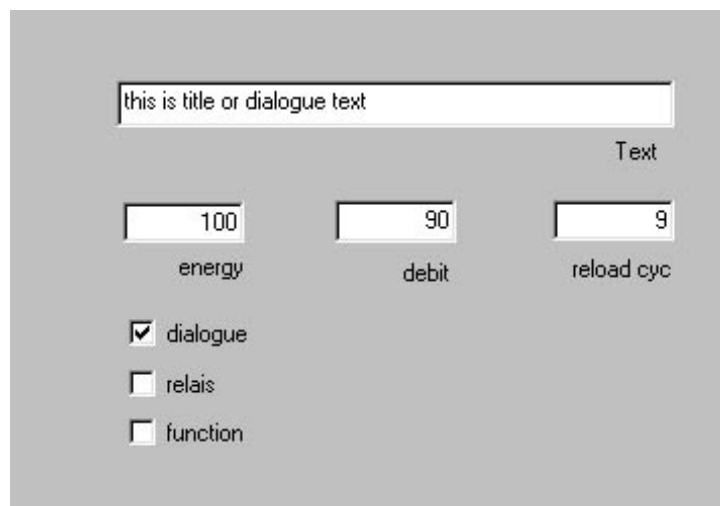
II. The energy system

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.



The screenshot shows a window titled "this is title or dialogue text". Below the title bar, there are three numeric input fields: "energy" with the value 100, "debit" with the value 90, and "reload cyc" with the value 9. To the right of these fields is a label "Text". Below the numeric fields, there are three checkboxes: "dialogue" (checked), "relais" (unchecked), and "function" (unchecked).

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 discharged 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, its energy will get 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 a 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).

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 start.

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

The image shows a graphical user interface for configuring a SmartTalkingNode. At the top, there is a text input field with the word "Greeting". Below this, there are three numeric input fields: "energy" with the value "100", "debit" with the value "90", and "reload cyc" with the value "9". Underneath these are three checkboxes: "dialogue" (unchecked), "relais" (unchecked), and "function" (checked). At the bottom, there is another text input field containing the word "beeper", with the label "function" centered below it.

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

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.

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.

In the current state (version 1.08) there's a limitation though. The function option processes option without value – but it would be much more elegant to offer values too. (Don't be afraid: this will be added soon).

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 Technical stuff

But let's go back to the details. Once you have written a dialogue, or a decision structure, as I prefer to say, you might one to save it. This is self explaining – as well as loading a file.

Nevertheless you will encounter two idiotic bugs, or better said: inconveniencies. When you save a file, please do not add the **nod* suffix. And second: when you load a file and try to edit, don't bother with me if it does not allow you to do this; just add another item (and delete if you don't need it), and everything works.

These are things to be fixed.

What is nice with the program is that it gives you a little internet explorer also. You can get to the SmartTalkingNodes– forum, leave your post and complaints (I hope not so), or you can jump to a few other 3d sites.



V The dll

It is quite obvious that you want to embed the functionality in your Gamestudio project. 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. You 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.

VI Will it be free?

There 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.