

CODE AS
DESIGN ELEMENT

INDEX

Diplomarbeit zu Erlangung des akademischen Grades
“Mag. für mediale Kommunikationsgestaltung (FH)”
im Studiengang Intermedia

Eingereicht bei Prof. DI Karl-Heinz Weidmann und
Maurizio Poletto an der Fachhochschule Vorarlberg

Stefan Rinner, Dornbirn 2001-06

Abstract	2
Introduction	3
The current situation	5
The designers' view	6
The programmers' view	7
The current tools	9
Macromedia Director	10
Programming by example - Pavlov	13
Approaches to change the situation	17
Extreme programming	17
Design by numbers	18
!scribble, !scribble-IMPORT	21
!scribble user-guide	23
!scribble-IMPORT user-guide	33
Examples	35
!scribble in detail	43
Creating your own modules	43
The settings files	45
The backend	45
The modules in detail	47
Upcoming features	57
Conclusions	59
Bibliography	61

ABSTRACT

This document describes the current situation regarding the use of code as a design element within multimedia projects. After describing the current situation, its causes and other's attempts to facilitate the use of interactivity, !scribble is presented.

!scribble is the author's approach to promote the use and exploration of interactivity and motion. It is a tool allowing not just single tests but also collaborative experimentation and the use of its outcome within normal production processes.

INTRODUCTION

The idea for this thesis evolved out of the author's personal experience. The importance and power of the combination of design and programming became evident to the author during his studies and jobs. The first idea was to develop code libraries for programmers using Macromedia Director. Public accessible algorithms written in Lingo are still hardly available and the Lingo community would have been probably enthusiastic about these libraries. Over time, the idea mutated to the vision of a tool to empower designers. The result is a tool that is ready for use and freely available. Nevertheless, the development and further improvement will not stop after finishing the thesis. Hopefully the tool will be used by a broad audience to prove its helpfulness, and with a bit of luck, other developers will start to add their own modules and will invent new features the author has never thought of.

THE CURRENT SITUATION

First, this document describes the situation in a very ‘black and white’ way. Yes, there are programmers who have a design education or at least care about design. Sure, there are designers who do programming themselves or understand the idea and the needs of programming.

However, usually programmers and designers do not share any common knowledge; they just share a common task. Both have to develop new ideas and they even do it in a very similar way.

Kent Beck, author of ‘Extreme programming explained: embrace change’, describes, for instance, the importance and benefits of drawing in the software-development-process: *‘First, there is nothing wrong with designing software using explicit pictures instead of purely mental or textual models of the system, and there is much to be said for a graphical approach. Trouble drawing the pictures can give you subtle clues about the health of a design. If you find it impossible to reduce the number of elements in the picture to manageable levels; if there is an obvious asymmetry; if there are many more lines than there are boxes. All of these are clues to a bad design that become evident from a graphical representation of the design.’* ([BE] p. 111)

In a common multimedia project, design and programming are seen as different steps in the process. First, the design concept is created. After its production, the finished screens are handed over to the programmers, and now everything should be moving and be interactive.

The designer is responsible for the ‘look’, the programmer adds some interactivity, and nobody cares about the result of this combination. Without a perfect look and feel, the project will not be successful. Projects will ‘work’ even if programmer and designers do not speak with each other at all – but something will be missing.

‘Programming is the process of abstracting a graphic form into its fundamental geometric components’ ([MA] p. 446)

In fact, it is not important that every designer can program and that every programmer is able to design. A huge step would be if both would appreciate the other’s work and would share common knowledge about the things which make multimedia so special – interactivity and motion. If both understand this, designers will develop applications which are more than a slideshow of nicely designed screens, and programmers will invest more work than

just programming some tri-state-buttons and some nifty, but hidden gimmicks. It would help if both think about multimedia development like game development – multimedia should offer a really strong impact regarding visuals and interactivity, multimedia has to be immersive and not just informative or entertaining. André La Mothe defines a game and its development as: *‘A video game is more than a program; it is a conduit into another dimension, a dimension synthesized by the imagination and realized by sheer will. Technically, a video game is nothing less than a work of art.’* ([LM1] p. xxvii)

THE DESIGNERS' VIEW

Multimedia is still a new medium and therefore many designers think the screen is just another type of paper or canvas. They think their task within multimedia projects is to place objects by using rules derived from centuries of experience in print design. Sure, universities and other organizations have started initiatives to teach and explore multimedia. Most have concentrated again on the visual part of the design. Some have thought about reusing well-known rules from print design.

Several new ideas of visual representations have been developed. Many have thought that the new and exciting part of multimedia is video, sound and non-linear content. But the idea that makes multimedia really special is sophisticated interactivity and motion – the things that evolve out of programming. Without code, multimedia is more or less the same as present day television. Even if designers start to develop ideas about interactivity, it is hard for them to attain – or just to scribble their thoughts. To develop ideas for visual design, designers have perfect tools – their sketchbook and a pen. A sketchbook is not just handy for the development of a vision, it is also useful for presenting ideas. Designers use their sketches to express ideas which they can not describe verbally. Illustrating interactivity or motion in full detail using the traditional tools is rather impossible – *‘Writing programs to create illustrations never makes much sense to students because they can be drawn much more easily by hand. However, as interactions are introduced, the differences between paper and computer become clear to students of any level.’* ([MA] p. 430).

Talking about ideas concerning interactivity or motion is even harder due to the fact that no defined and common vocabulary has been established up to now.

Designers have to rely absolutely on the programmers for creating their envisioned way of interactivity and therefore presenting their ideas in a concise way is crucial. Demonstrating ideas to clients is even harder and that is why a tool for easy and fast creation of prototypes is needed.

The current professional tools used for creating multimedia applications are designed for programmers. These tools – their metaphors and their graphical user interface - are intended for fast and smooth production and not for experiments. Although they allow a lot of manipulation via the user interface, more complex tasks, but even quite simple interactivity, rely on heavy scripting.

Other tools intended for experimentation, for instance Design by Numbers, are widely unknown, don't offer any compatibility with industry standard tools and are often restricted in several ways – for instance customizability or extendibility.

THE PROGRAMMERS' VIEW

Programmers are rarely involved in the design process. In some cases, there is even no contact and communication between the designers and programmers due to big and complex company structures or outsourcing. Usually during the conception stage of a project programmers are solely concerned with requirements, schedules or cost estimations. Programmers who are interested in adding additional value to a project by using innovative techniques of interactivity have at least a way to develop and sketch their ideas, but the mostly low visual impact of their proposals doesn't impress any designer. Schedules in multimedia projects rarely work and the person responsible for programming is the last one in the production chain. Usually no time to follow one's own experiments is available and programmers are mostly very defensive against ideas which sound complicated and time consuming. Everything which cannot be done by using some standard library, can become potentially time consuming.

THE CURRENT TOOLS

For programming languages that are widely used in academia and industry a vast amount of ready-to-use algorithms is available and a huge number of papers and books discuss and explain various problems.

The industry standard tool in multimedia is Macromedia Director. Its programming language is so primitive that no computing science graduate would ever touch it and offers such a bad performance and so many awkward bugs that no one trained in high-level languages would ever think about using it. Compared to C, C++ or Java the user-base is ridiculously small and therefore the number of public-accessible sources and amount of documentation is not sufficient. Programmers using Director have to rely on initiatives from other talented and philanthropic programmers on the web, presentations at various meetings and gatherings, reverse engineering of the competitor's stunning works or on books intended for people using high-level languages.

To program a multimedia application the programmer can choose between using general-purpose programming environments like Microsoft's Visual Studio or an authoring suite designed for the creation of rich-media applications. The integrated development environments offer an open architecture and good performance but an interface not intended for multimedia. On the other hand the authoring environments' interfaces are suitable for multimedia but don't offer any groundbreaking performance.

'Interactive tools like those in Visual Basic and C++ significantly decrease the time and expertise necessary to build standard interfaces, i.e. those with widgets such as menus, buttons, and list boxes. However, these systems provide little interactive support for designing interfaces containing application-specific graphics (arbitrary drawings) and animation.' ([WO1] p. 347)

Several companies are active in the authoring-environment market. Absolute market-leader is Macromedia with its products Director, Flash and Authorware. Other available software-packages are Asymetrix's Toolbook and Tribeworks' iShell.

This paper and its accompanying tools and examples concentrate mainly on Macromedia Director.

High-level programming languages are hardly used within the multimedia business.

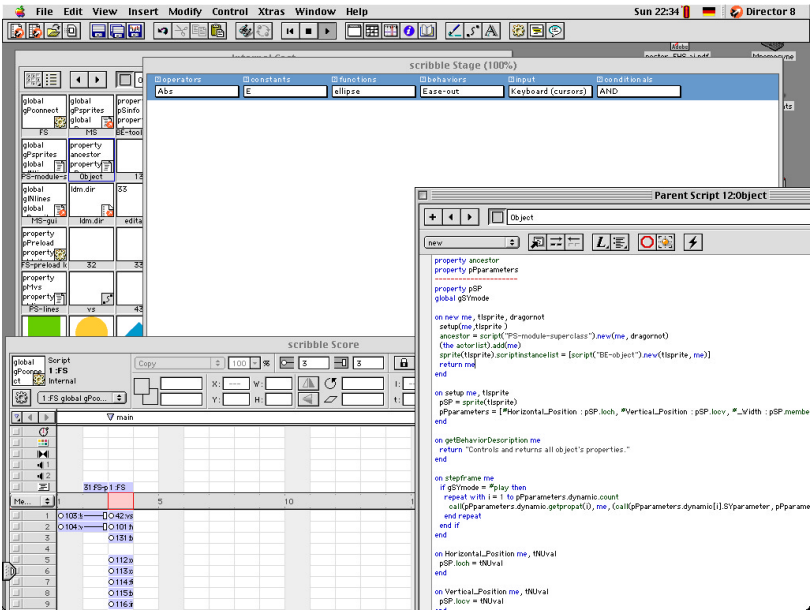
Products developed using C/C++ or Java are mostly found in academia, arts and in products from the game industry.

MACROMEDIA DIRECTOR

Until version six, Director offered two different means of use. Director was either a tool to create complex, but non-interactive, two-dimensional animations or a development tool for highly interactive and complex applications.

Creating an animation was an easy task, which could have been done by just using the graphical user interface. Director offered a simple and working metaphor – the stage, the cast, the score and several widgets and gadgets to help the user. The user interface offered no way to add interactivity – to achieve this, the

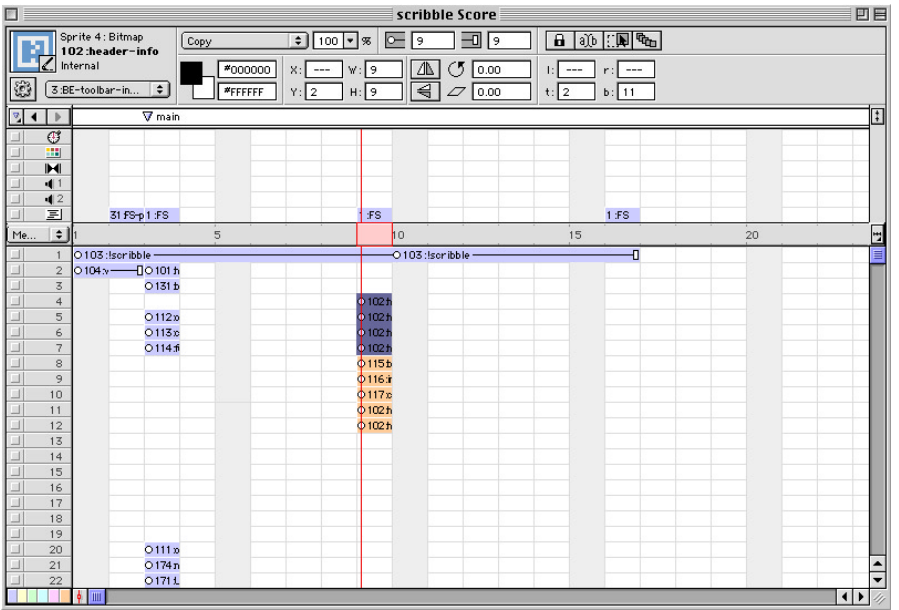
Fig. 1: Macromedia Director



user had to learn Lingo, Director’s integrated scripting-language. By using it and its various available extensions called Xtras, nearly every task can be accomplished. David Wolber, researcher in the field of new and easy authoring-environments describes Macromedia Director: ‘The most popular interactive methods for defining

animation are found in commercial key-frame animation systems such as Director. With these systems, designers manipulate a timeline (score) and an interface (stage) to specify the state of each graphic during each time frame. Interaction and context-triggered behavior can also be specified with these systems, but only by programming with a script language’ ([WO1] p. 348)

Fig. 2: Macromedia Director - the score



The upgrade to version six offered a new way of scripting – behaviors. These new type of scripts controls the behavior of visual objects and their properties can be changed during the authoring by using a graphical user interface. After this feature was introduced designers could get their hands on small objects they could combine and reuse.



Fig. 3: Macromedia Director - the library

Part of the version six package was a set of behaviors called library. This library has several advantages and disadvantages. It offers an easy way to add some basic interactivity and some of the behaviors are even quite useful to create complex interactivity.

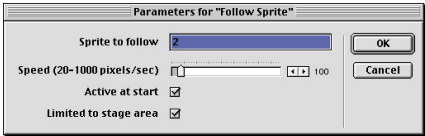
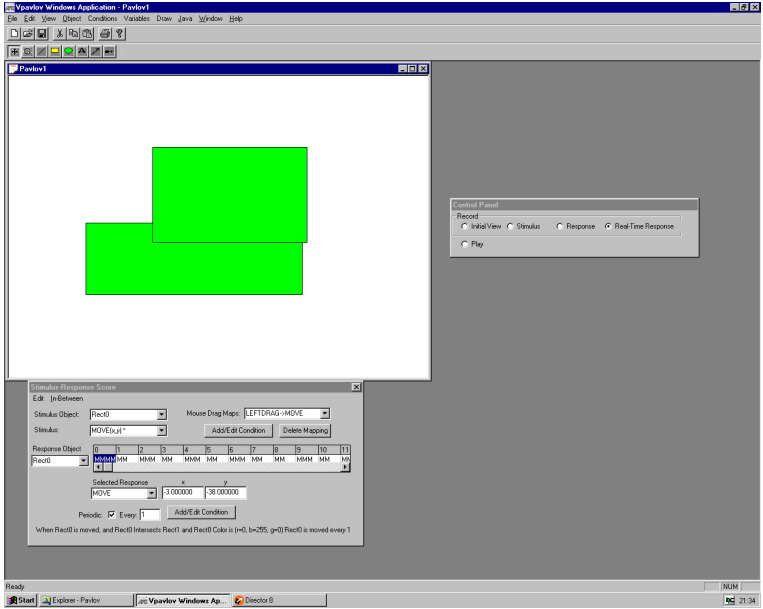


Fig. 4: Macromedia Director - a behavior dialog window

The disadvantages are that the code of these behaviors is hard to understand due to its abstruse complexity and some of them are even buggy. Additionally, the behaviors cannot be combined in a transparent way and most of the behaviors are just about translations or rotations and cannot be used to control other properties. Macromedia never tried to attract more customers from a non-programming background by providing additional and more useful behaviors. The latest upgrade to Director added a real-time 3-D engine but still many problems in the user interface and Lingo persist.

New versions did not offer any improvements for animation-orientated users and Macromedia does not promote Director anymore as an animation tool. Therefore, Director is a development tool for programmers – in fact, it is just a tool for one programmer due to the fact that collaborative work is nearly impossible.

Fig. 5: Pavlov



PROGRAMMING BY EXAMPLES – PAVLOV

An attempt to allow the creation of interactive applications without any real coding is seen in programming by example.

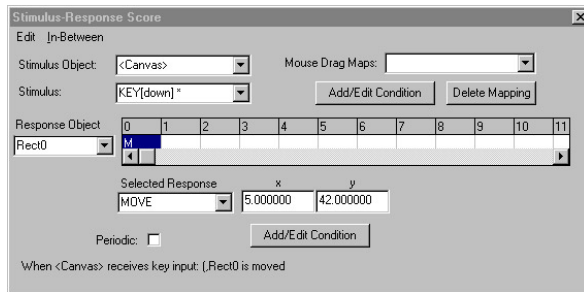


Fig. 6: Pavlov - the score

Its main principles are described as ‘... represent the objects and action visually, replace typing with pointing and dragging, provide rapid, incremental and reversible actions, and offer immediate and continuous feedback to users’ ([SH], p. v).

A project especially suitable for multimedia projects is Pavlov by David Wolber from the University of San Francisco.

He defines the difference between industry-standard tools and Pavlov, as ‘There are two main differences between Pavlov and animation systems such as Director. The first is that interaction is specified by demonstration instead of coding with a scripting language. The second is that the timeline-based editor in Pavlov organizes the animated activity by the stimulus that triggers it, instead of requiring the designer to

organize it manually within a single timeline.’ ([WO1] p. 385)

The use of Pavlov is easy and straightforward. The user presents a stimulus – he hits a key, moves the mouse or clicks on something and Pavlov asks for the response wanted. *Pavlov combines programming by demonstration (PBD) with animation mechanisms similar to those in Macromedia’s Director. Instead of writing scripting code to specify interaction, as is done in Director, the Pavlov designer demonstrates the events that trigger activity (and the activity itself): The resulting behaviors then appear in a stimulus-response score, ...*’ ([WO2] p.346)

Pavlov does not just react to the user’s stimuli, it also checks for other conditions, for instance the overlapping of graphics. The interface works very well and allows fast and efficient production – ‘The students estimated, on average, that completing a similar project would require 150 minutes with Visual C++ and 17.5 minutes with Pavlov (two expert Pavlov designers also implemented the Arcade game; both required less than 10 minutes).’ ([WO1] p. 384)

APPROACHES TO CHANGE THE SITUATION

This is not the first attempt to establish programming as a design element or to support designers and programmers in incorporating interactivity in their projects. Several initiatives in software development, like Extreme Programming, Programming by Example, the work at various research institutions, for instance the MIT, and initiatives based on the engagement of individuals improved the situation and provided help and insights.

EXTREME PROGRAMMING

Changing the organization and workflow of multimedia projects in a way that promotes discussion and collaboration between the participating parties allows the creation of new and innovative solutions. Often designers and programmers who are working on the same projects are not part of the same department, are working separated or even, in the case of outsourcing, in different companies.

Extreme Programming is a method of software development that encourages work in pairs and development driven by tests.

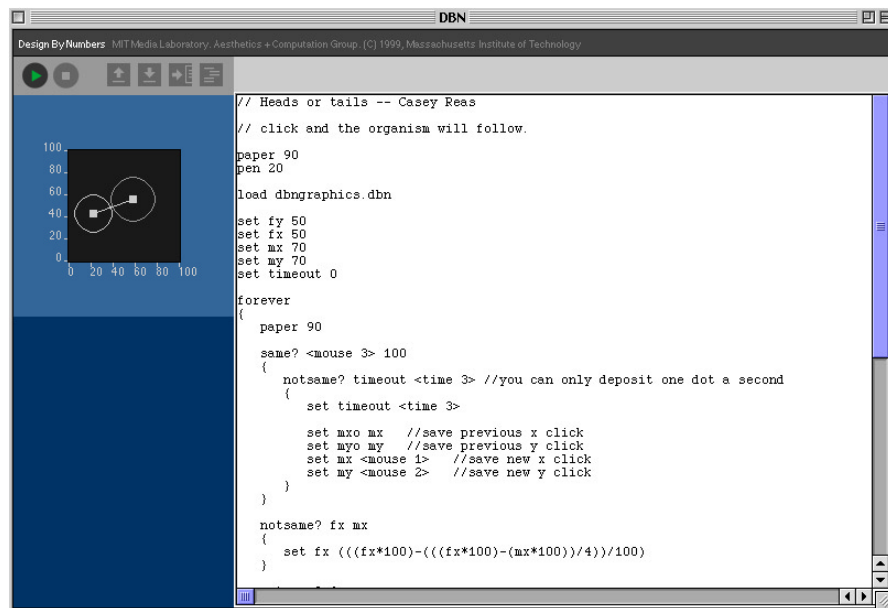
The basic ideas of Extreme Programming like pair programming and collective ownership certainly work and are effective for the development of multimedia projects. Kent Beck describes extreme programming as *'XP is a lightweight methodology for small-to-medium-sized teams developing software in the face of vague or rapidly changing requirements'* ([BE] p. XV) and this description resembles perfectly the usual situation within a multimedia-project.

DESIGN BY NUMBERS

'A comprehensive system was developed with sufficient limitations and subtle extensibilities to enable a wide range of digital expressions for the beginning computational art student. The result was the Design by Numbers system.' ([MA] p. 442)

Design By Numbers was developed by John Maeda and the Aesthetics and Computation Group within the Media Lab at the MIT. Design By Numbers allows the creation of static, dynamic and interactive graphics by using a simple programming language.

Fig. 7: Design by Numbers



On the MIT's website the aim of Design By Numbers is described as follows:

'Design By Numbers was created for visual designers and artists as an introduction to computational design. It is the result of a continuing endeavor by Professor John Maeda to teach the "idea" of computation to designers and artists. It is his belief that the quality of media art and design can only improve through establishing educational infrastructure in arts and technology schools that create strong, cross-disciplinary individuals.' ([MIT2])

The language is quite simple but offers everything needed – loops, conditionals and recursion.

```
Paper 0
Pen 100
Repeat A 0 100
{
  Pen A
  Line A 0 A 100
}
```

To make coding even easier DBN can be programmed not just in a syntax based on English vocabulary but as well in Spanish, French, and Japanese.

The current version can be obtained for free at <http://dbn.media.mit.edu/>.

DBN is not just a single tool; Casey Reas at the MIT developed a courseware system to promote and facilitate the use of DBN as tool in design-education.

'The Design By Numbers Courseware is a web-based environment built to facilitate the use of DBN as an educational tool for teaching computational design. It provides educators with an interface for inputting/editing problem statements and students with the ability to work in DBN and save their work directly to the server. Most importantly, it presents a unique viewing environment where students can critique their work within the context of the entire class.' ([MIT1])

Several years ago John Maeda also published a book about DBN to teach the use of DBN and showcase some work.

!scribble, !scribble-IMPORT

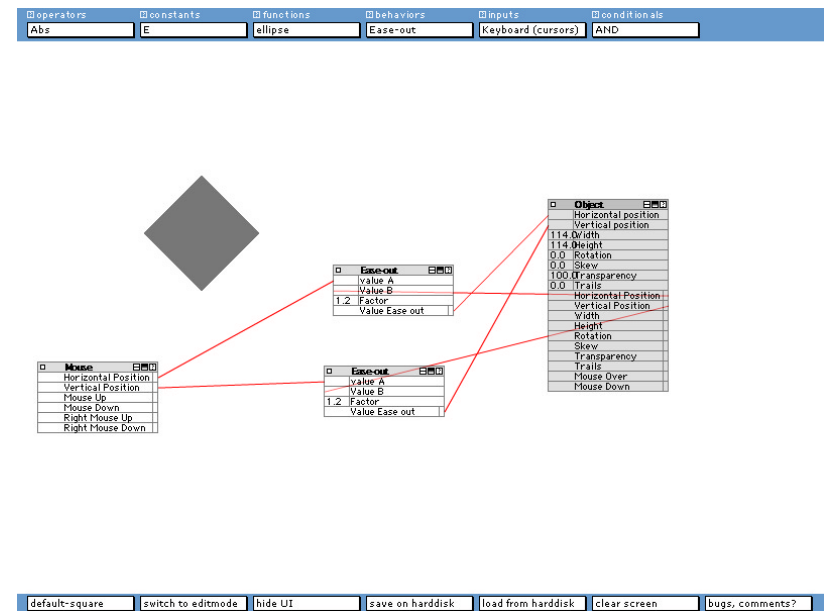


Fig. 8: !scribble

The practical outcome of this thesis consists of two applications - !scribble and !scribble-import.

First, !scribble enables the user to create complex interactivity or motion without any real programming. It is a tool to define properties and behaviors of visual objects in a graphic way and to present and share the outcomes.

Its most prominent features are the combination of behaviors, the incorporation of user input and the possibility to retrofit the user's own scripts into the application.

The application is easily extendible and allows the user to integrate his/her own visuals.

!scribble

USERGUIDE

It is possible to use nearly every kind of asset Director is capable of importing: bitmaps (JPEG's, PSD's, GIF's, PICT's, BMP's), Flash-files, other Director-movies, Videos or formatted text (HTML, RTF). This allows the easy integration of already existing elements, and the use of other interactive or animated content (Gif89a, Director and Flash-files) offers even more complex and stunning effects.

!scribble works like the rock'n'roll guitar-player's battery of effect devices. There is an input – an accord or a mouse movement – an effect-widget producing an output - sometimes again interconnected with a patch cord to another electronic gadget – and in the end some outcome: a trembling loud-speaker or a fascinatingly moving sprite. The application is intended as a tool for users in education and production. Its target group consists mainly of designers interested in interactivity and motion.

A basic knowledge of mathematics is recommended but !scribble can be used as well as a tool to visualize math in an educational context. After some examples are shown or the basic principles were explained, most users will be able to start building their own applications.

As a starting point, the examples provided and included in the distribution are quite efficient to give an overview of the tool's features and power. Additionally, within the application, tooltips act as guidance for the user.

After successfully creating an interesting example of interactivity the user can save the settings to an external file.

!scribble writes the current state into an XML-file and stores the used visual objects within an accompanying folder. The second tool which was developed is called !scribble-import and allows the use of settings created in !scribble within multimedia-projects. The tool, running as MIAW-Xtra within Director, reads a setting file, creates all needed scripts and sprites and defines the behaviors' properties as needed.

The main advantage of this tool is the direct incorporation of the designer's input into the final project. The designer's experiments do not have to be programmed again and the result will look absolutely the same as the prototype the designer created.

At the moment !scribble comes in two different forms: an online and an offline version. In the near future an additional standalone presentation application will be developed. Its purpose is to provide an easy and foolproof way of presenting the user's experiments to others.

The Shockwave version is played back in your preferred browser and does not need any manual downloads or installation. Due to Shockwave's security architecture it does not allow any import or export of files from or to your harddisk. The offline version is much more convenient but has to be downloaded first. The advantages are that the user does not have to be connected to the Internet all the time and the possibility of importing and exporting his/her own assets and settings.

!scribble's interface consists of three basic elements: panels, objects and lines. Objects are graphics provided by !scribble, imported or drawn by the user. They are manipulated via panels, boxes showing all their properties and handlers (read inputs and outputs). Scripts used to manipulate the properties dynamically are called modules. Each of those is controlled again via an accompanying panel.

The main idea and power of !scribble is the combination of modules. A module is represented as a panel consisting of inputs and outputs and lines illustrate the connections between modules. Every output can be connected to multiple inputs but not more than one output can be connected to a single input. If one property (one input) should be affected by several outputs they have to be combined first to one value – for instance by adding them to one sum.

First, the user has to instantiate objects and modules. The properties can be either static or dynamic. Static values are defined by the user by filling in the wanted value or by setting them interactively during edit-mode. Properties' values get dynamic by connecting another module's output to it. Connection is accomplished by clicking on an output, keeping the mouse pressed and releasing it over the intended property. To facilitate this procedure several hints are presented to the user, for instance changing cursors and highlighted areas. After successfully connecting two panels, the connection is illustrated as red line.

The main window is structured into three areas – a toolbar at the top, which displays all available modules and is used for creating new instances, a toolbar at the bottom to control the application and the objects, and the stage where all the panels and objects are placed.

THE MODULE-TOOLBAR

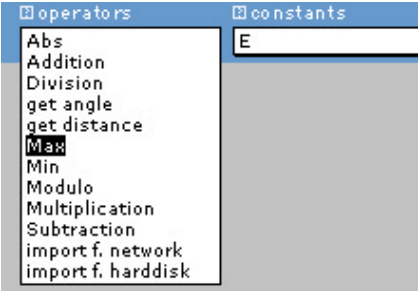
The toolbar at the top offers access to all available modules. The modules are organized into six categories: operators, constants, functions, behaviors, inputs and conditionals. A dropdown menu shows all modules within a category. By selecting a module, !scribble generates a new instance of the module. A panel appearing on the stage represents the new instance visually. In the standard distribution of !scribble many modules are included. To extend !scribble’s power even further the user can import his/her own scripts. A repository of modules on a public webserver should facilitate and promote sharing of modules. After selecting ‘import f. network’, the user is directed to this collection, but by altering the URL manually, the import from other locations is possible as well. The offline version is granted access to the local file system and therefore offers the possibility of importing modules from the user’s local storage or other mounted volumes. The modules are simple text files containing several handlers written in Lingo. The files’ file-extension is ‘!s’.

It is important to make sure that the imported modules are correct regarding syntax and semantics. Scripts imported dynamically during runtime cannot be examined for errors and therefore could crash the whole application. It is highly recommended to test the scripts in depth before they are made available publicly.

OPERATORS

The category ‘operators’ contains basic mathematical operations. The current list consists of: abs, addition, division, get angle, get distance, max, min, modulo, multiplication and subtraction.

Fig. 9: !scribble - the ‘operators’ popup menu



These are useful for some basic behaviors and for tweaking or combining other modules.

CONSTANTS

The modules in the ‘constants’ category allow an easy way to access constants.

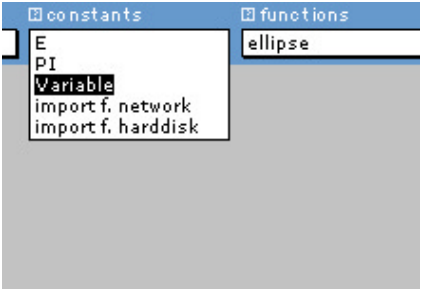


Fig. 10: !scribble - the ‘constants’ popup menu

At the moment π , e and a user-definable variable are part of this category.

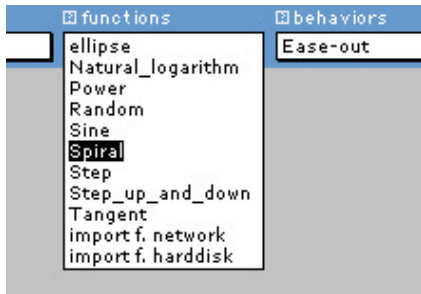


Fig. 11: !scribble - the 'functions' popup menu

FUNCTIONS

The 'functions' drop-down menu shows: cycloid, ellipse, lemniscate, natural logarithm, power, random, sine, spiral, step, step up and down and tangent. These functions create sequences of numbers (step, random) or modify a single number.

BEHAVIORS

'Behaviors' are mainly algorithms, which were developed for interactive applications. The algorithms are based on formulas from various fields or just a combination of simple operators. Currently just ease-out and serial-chain are part of this group.

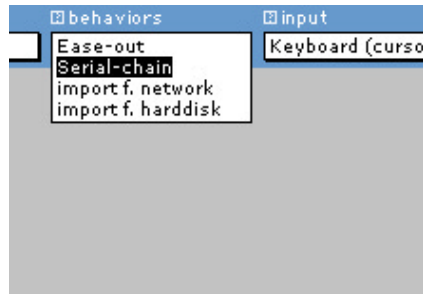


Fig. 12: !scribble - the 'behaviors' popup menu

INPUTS

This group's modules allow incorporation of the user's input. The distribution contains the modules keyboard (cursors), keyboard (digits) and mouse.

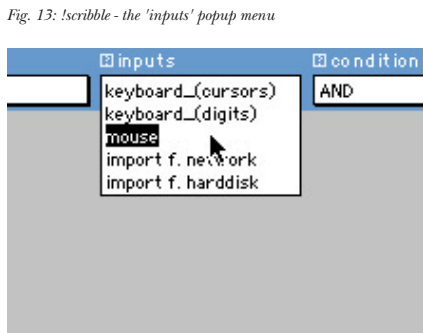


Fig. 13: !scribble - the 'inputs' popup menu

CONDITIONALS

Conditionals allow internal logic within complex systems and are important to build systems which react on events and interaction. The category includes Boolean expressions like AND and OR and comparative operators like equal, not equal and bigger than.

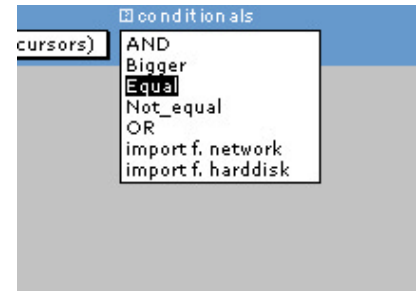


Fig. 14: !scribble - the 'conditionals' popup menu

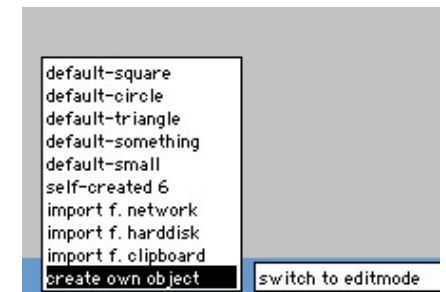
THE MODE AND TOOLS TOOLBAR

The toolbar at the bottom enables the user to control the application and the objects. The toolbar's features are: changing !scribble's mode and appearance, creating, importing and exporting documents and objects, and providing feedback to the author.

OBJECTS

The 'objects' popup menu lists all available objects and offers four ways of getting additional ones. The user can import files from the Internet, his/her local harddisk or just simply paste them from the clipboard.

Fig. 15: !scribble - the 'objects' popup menu



The option 'create own object' opens up a simple vector-shape editor, which allows the user to draw his own rectangles, triangles, circles or free forms. After clicking 'export', the newly created object is available in the 'objects' list.

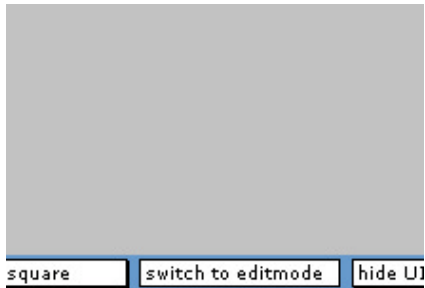


Fig. 16: !scribble - the 'switch to play/edit-mode' popup menu

SWITCH TO PLAY/EDIT-MODE

!scribble runs either in 'play' or in 'edit mode'.

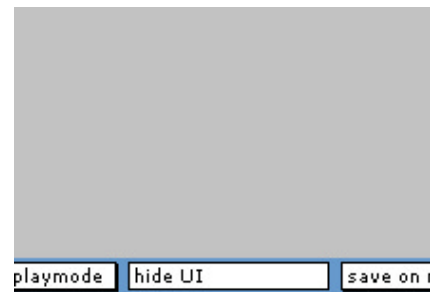
When running in 'play mode' all property values are recomputed every frame. In 'edit mode', no formulas are evaluated and the user can change several objects' properties interactively,

for instance the position. The newly defined properties are set as static values. Therefore, existing connections of modules controlling the object's properties are deleted. The shortcut for switching the mode is 'm'.

SHOW/HIDE UI

To create effects that are more complex the user has to combine many modules which can lead to screen clutter. There are two ways of freeing up screen space – by collapsing the panels or simply by toggling the visibility of the panels and lines. Hitting the 'u'-key has the same result as clicking the 'Show/Hide UI' button.

Fig. 17: !scribble - the 'show/hide-UI' popup menu



SAVE ON NETWORK/HARDDISK

!scribble is able to save your settings and all used visual objects. The settings are stored in an XML-file, and the application copies the assets to the same folder. !scribble's online version stores the files using the backend on <http://www.rinner.st/diploma-stuff/!scribble/builds/CURRENT/settings/>.



Fig. 18: !scribble - the 'save on network/harddisk' popup menu

Users are encouraged to make their experiments available for the public by loading them up to the server.



Fig. 19: !scribble - the 'load from network/harddisk' popup menu

LOAD FROM NETWORK/HARDDISK

To open the user's or others' experiments again the user has to import them by clicking the 'Load from network'/'Load from harddisk' button. The offline version allows the import from the harddisk and the Internet, the online version is only able to get files from the web.

CLEAR SCREEN/NEW DOCUMENT

If the trails of an object are set to 1, the screen buffer does not get completely redrawn. The option ‘Clear screen’ forces a complete redraw and cleans up the screen.
‘New document’ removes all objects and modules and resets everything else.



Fig. 20: !scribble - the 'clear screen/new document' popup menu

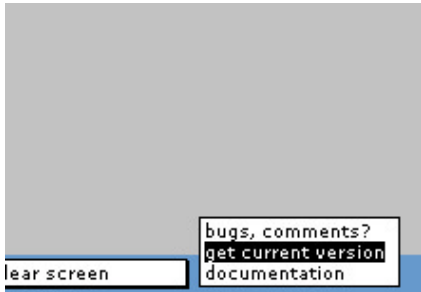


Fig. 21: !scribble - the 'bugs, comments?' popup menu

BUGS, COMMENTS?

The button to the farthest right in the bottom toolbar offers access to !scribble’s website and contact to the author. The menu provides direct contact to the author by email, a link to the upcoming online version of the user guide and a pointer to a site containing download links to the most up-to-date version of !scribble.

THE PANELS

The panels and lines are not just part of the user-interface and allow the user to manipulate properties. The interface is as well an illustration of the user’s interaction.
A panel consists of three elements: a title, inputs and outputs.

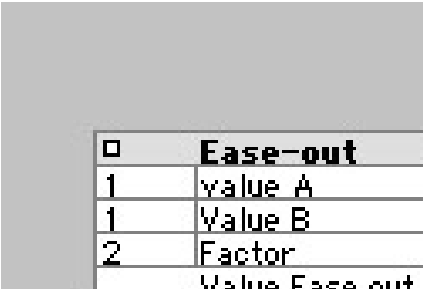


Fig. 22: !scribble - a panel

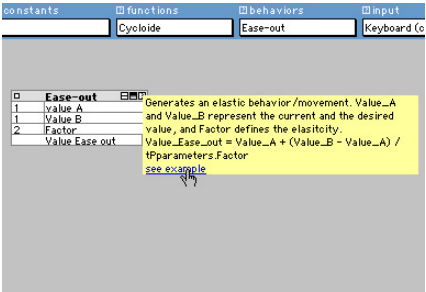
TITLE

The panel’s title states the name of the module or the object.
Three buttons on the right allow the manipulation of the panel itself and assist the user.

The first button toggles the panel’s size – either to the height of the title line or full size. This is mainly helpful to clean up screen space and to organize complex documents.
Clicking on the middle button changes the panel’s transparency.
The third button’s purpose is to assist the user. After rolling over or clicking the question mark, a tooltip appears and explains briefly the panel’s use and features. If the module’s author has provided a tutorial file and a link to it in the tooltip the file can be opened after clicking on the link.

To move a panel the user has to click the title line and keep the mouse pressed while moving it.

Fig. 23: !scribble - a panel and its accompanying tooltip



!scribble-IMPORT USERGUIDE

After rolling the mouse over the title of an object panel a red outline around the object under control illustrates the relation between panel and object. There is also a button on the left – its purpose is to close the panel.

INPUT

The panel's inputs are values used for the outputs' computation. These values can be either static or dynamic. Changing a static value is accomplished by clicking it and typing in a new value. If the value has been dynamic up to now and the user clicks it, the input becomes static.

OUTPUT

Outputs return the result of a computation. This value can be used to manipulate an object's properties or as a basis for further calculations.

By clicking an output, keeping the mouse pressed and by releasing it over another panel's input, this property is based on the first panel's output.

To assist the user in connecting panels, several hints, for instance changing cursors and highlighted areas, are provided.

!scribble-import is a tool running within Director. To use this application, it has to reside in Director's Xtras folder and is accessible by the user via the Xtras menu. After launching, !scribble-import runs within its own window.

To import a setting into a project the user has to follow five steps.

First, the user has to browse his hddisk and define !scribble's location within

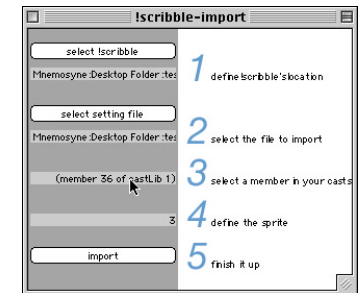


Fig. 25: !scribble-IMPORT

To define which member the new sprite should be based on, the user has to select a member in one of his/her project's castlibs.

Step four is to define the channel in which !scribble-import has to place the sprite.

After finishing the import, the sprite is placed in the score and all needed scripts were imported. A behavior was attached to the sprite, which instantiates and controls the motion and interactivity.

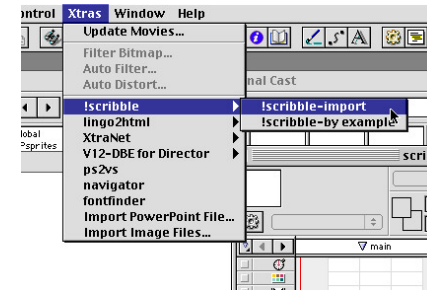


Fig. 24: Macromedia Director - the Xtra menu

the file-system. This is necessary to be able to import !scribble's modules. The path is written to an external preference-file and therefore does not need to be redefined every time.

Second, it is necessary to select the file, to be imported.

EXAMPLES

The examples mentioned in this chapter are part of the current distribution. In the offline package these files are stored in the folder 'examples', users of the online-version find them at <http://www.rinner.st/diploma-stuff/!scribble/builds/CURRENT/examples/>.

They can get opened via the 'load from network/harddisk' popup menu or by following the links in the panels' tooltips.

The examples may not impress at first sight but help to prove !scribble's versatility and power.

OBJECTS FOLLOWING THE MOUSE

FOLLOWING MOUSE

To order an object to follow the mouse it is necessary to connect the horizontal and vertical position of the mouse to the object's position. The object's regpoint is repositioned in every frame without any delay under the mouse cursor. This behavior is useful to replace the cursor or for instance to place something like a tooltip next to the mouse.

FOLLOWING MOUSE SMOOTHLY

The use of 'ease out' allows the creation of more smooth or organic effects. It takes three arguments – the current value, the desired future value and a factor defining the speed or elasticity. The factor has to be greater than 1 to create elasticity. To create an elastic motion the mouse position and the current object position are used as parameters.

'Ease out's power is not restricted to altering positions and following the mouse. The module is useful for any task where smooth change is needed. The desired future value can be also based on something different than the user's input.

FOLLOWING, ROTATING MOUSE

The way an object follows the mouse can be spiced up a little bit by adding behaviors based on the distance of the object to the mouse. In this example the object's rotation is connected to the distance, this creates a more dynamic and chasing effect.

FOLLOWING, FOLLOWING

‘Ease out’ is also useful to achieve the effect of objects following objects, following objects. To accomplish this, the current value of the leading object has to be used as the desired future value of the following object.

ROTATE TO MOUSE

To create the illusion that an object is ‘watching’ the mouse, the object has to rotate according to the current mouse position.

The rotation value is based on the result of ‘get angle’. ‘Get angle’ returns the angle (in degrees) of the line formed by the two given points.

FOLLOWING MOUSE (SPIRAL)

An object’s movement corresponding to the mouse is not necessarily about following the cursor. The object’s motion can be based also on a formula, in this case on a spiral, and the mouse controls one of the formula’s main parameters. To make the spiral movement more obvious the object’s trails are set to 1.

RIGID SERIAL CHAIN

This example illustrates the use of the ‘serial chain’ behavior. Objects connected by a rigid serial chain try to keep a defined distance – they follow each other but avoid getting nearer than defined by the radius. In this example, an object follows the mouse and is itself chased by another object.

FADING, BLINKING

Changing an object’s transparency can be based on several different formulas. The result is based on the formulas’ characteristics.

To set up an example is quite easy – for instance ‘step’ or ‘step up and down’ produce a sequence of numbers within a defined range. The values are used as parameters for formulas and their results control an object’s transparency.

LINEAR FADING

Linear fading is the simplest way of fading. The transparency value meanders within the defined range. The speed is constant without any acceleration or deceleration.

LEMNISCATE FADING

Using a lemniscate to control an object’s transparency results in some slow and weak flickering.

SINE FADING

Fading based on sine curves is regular and organic. Its speed seems constant and natural and without any peaks or abrupt changes.

MOVING OBJECTS

Moving objects is basically changing an object's position according to a function or interaction. It is possible to move an object in a linear way or based on a function or curve.

'Step' and 'step up and down' are used for simple linear movement or as a basis for more complicated motions. It is important to understand the difference between the two modules. If the smallest value is 0 and the biggest value is 3, 'step' will compute a sequence like this: 0, 1, 2, 3, 0, 1, 2, 3, ..., 'Step up and down' would return such a sequence: 0, 1, 2, 3, 2, 1, 0, 1, ...

STEP

The 'step' example shows an object moved and rotated by the 'step' module. It starts at 100, goes up to 700 and starts again at 100.

STEP UP/DOWN

This example illustrates the 'step up and down' module. As in the 'step' example an object is moved and rotated. The difference is that the object starts at 100 goes up to 700, moves down to 100, ...

CYCLOID MOVEMENT

The object in this example staggers horizontally over the stage. Its reeling movement is based on a cycloid.

HORIZONTAL LINEAR AND VERTICAL SINE MOVEMENT

The example illustrates the use of the sine function. An object moves horizontally according to a 'step up and down' function and vertically following the graph of a sine function.

ROLLOVERS

Rollovers are a common way to signify an object on the screen as interactive. After moving the mouse cursor over an object, the object reacts. Mostly the object changes its appearance by swapping images or altering transparency. Every object has an output labeled 'mouse over'. If the mouse is over the object this output returns 1 otherwise 0 is returned.

SIMPLE ROLLOVER

If an object's transparency is 50 and should be altered to 100 when the mouse rolls over, the object's 'Mouse over' is multiplied by 100. The result of the multiplication and the value 50 are the parameters of a 'max' module, and the result is the object's transparency.

ELASTIC ROLLOVER

An elastic rollover differs from the simple rollover by the fact, that the output of 'max' is not connected directly to the object's transparency but via an 'ease out' combining the current and the desired transparency.

RANDOM

The random setting demonstrates the functionality of the random function. An object is placed randomly within a defined area on the stage. The area is defined by the two random functions' range-settings. To illustrate the random placement the object leaves trails on the stage.

CONDITIONALS

ROLLOVER BASED ON CONDITIONALS

To demonstrate the use of conditionals this example shows a rollover built up by using conditional operators. An object is following the mouse and a combination of conditionals checks if this object is within another object. To express it in pseudo-code: if the horizontal position of object A is bigger than the left edge and smaller than the right edge of object B, and if object A's vertical position is within the range defined by object B's top and bottom position, then the transparency of object B is set to 100.

INPUTS

CURSOR BASED MOVEMENT

This example illustrates the use of the 'cursor' input module. Its four outputs return either 1 or 0. The results get added to the current position of the object.

!scribble IN DETAIL

CREATING YOUR OWN MODULES

One of the main goals during the development of !scribble was to keep the system easily extendible. The author tried to keep the architecture as simple and consistent as possible. Every interested user should be able to create his own modules. The system allows the user to concentrate on the functionality of his script and without having to think about module management and inter-module communication. All these tasks were already addressed in the modules' ancestor script.

A working module has to contain at least four handlers: Director's usual constructor – `new()`, `setup()`, to define the module's properties, `getBehaviorDescription()`, which returns a description of the module's functionality and a handler with a freely definable name to compute the module's output.

These lines containing the new-handler and the property-declaration can be reused without any further modifications:

```
property ancestor
property pParameters
```

```
on new me
  setup(me)
  ancestor =
    script("PS-module-superclass").new(me)
  return me
end
```

The setup handler contains a property list defining the module's properties. It is recommended to choose short and widely understandable names and to ignore software industry's dogmas like Hungarian notation.

```
on setup me
  pParameters = [#Factor_A : 1, #Factor_B : 1]
end
```

The handler `getBehaviorDescription()` returns a string describing the use of the module.

The string is formatted as HTML, line-breaks can be added via `<br>`, and to illustrate a module's function it is possible to link to a tutorial file via the `<a>` tag. The file has to be stored in the example folder next to the application.

```
on getBehaviorDescription me
  return "Multiplies Factor_A with Factor_B
and returns the product. If Factor_A is
3 and Factor_B is 5, 15 would be
returned.<br>Product = Factor_A *
Factor_B<br><a href='multiplication'>see
example</a>"
end
```

A user-defined handler is used to compute the result.

The handler `getCurrentParameters()` provided by the ancestor script computes all needed values for calculating the final result and returns them in a property list.

```
on Product me
  tPparameters = me.getCurrentParameters()
  return tPparameters.Factor_A *
tPparameters.Factor_B
end
```

If additional properties are needed to calculate the final value, but the developer doesn't want them to show up in the panels, these have to be declared in the normal way and not within the setup handler. If a handler, for instance a procedure for calculating constants, should not be accessible from the GUI, its name has to start with an underscore.

THE SETTINGS FILES

The settings are stored in XML-files. The decision to use XML was based on: human-readable format, broad availability of parsers, structured storage of information and easy up- and downloading via HTTP.

The saved information consists of all used modules and objects, their current static values, the sources of their dynamic values, the filenames of the visual objects and the panels' positions. !scribble generates an own folder for each settings file and stores the file itself and all external objects there.

Andrew White's XML-parser is used for !scribble's internal XML-parsing. The source can be obtained for free at <http://www.cathode.co.uk/ucon99/>.

THE BACKEND

!scribble's website and collaborative features are based on a combination of some PHP and vanillaSite.

PHP is responsible for the storage of modules and objects on the server. The functionality is basically uploading files, reporting the content of folders and some HTTP-redirecting.

VanillaSite, a Wiki-Clone developed by Christian Langreiter offers many features suitable for managing interconnected content.

On <http://www.rinner.st/diploma/> vanillaSite is used to create, edit and display the content and its interconnections. It also proved helpful for managing the bibliography, the link-collection and outlining.

Both products – PHP and vanillaSite are freely available.

More information and downloads can be found at <http://www.php.net/> and <http://www.langreiter.com/vanilla/>.

THE MODULES IN DETAIL

OPERATORS

ABS

This module returns the absolute value of value A. The absolute value of any number is positive e.g. $\text{abs}(-4)$ is 4, $\text{abs}(17)$ is 17.

$\text{Absolute_Value} = \text{abs}(\text{Value_A})$

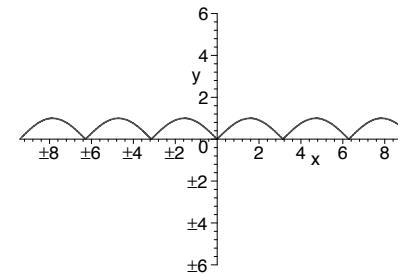


Fig. 26: $\text{abs}(\sin(x))$

This module is useful to avoid negative values or to mirror a function along the x-axis.

ADDITION

Summand A is added to summand B and the sum is returned.

$\text{Sum} = \text{Summand_A} + \text{Summand_B}$

DIVISION

After dividing the dividend by the divisor the result, the quotient, is returned.

$\text{Quotient} = \text{Dividend} / \text{Divisor}$

GET ANGLE

This panel returns the angle of the line formed by point A and B.

GET DISTANCE

Returns the distance of point A to point B.

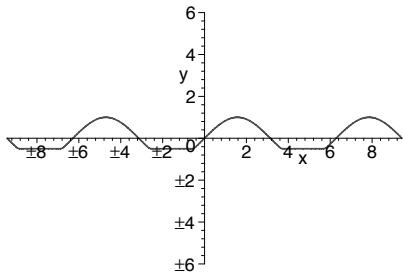


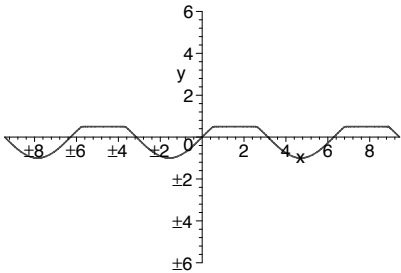
Fig. 27: $\max(\sin(x), -0.5)$

MAX

Max compares value A with value B and returns the bigger number. For instance if A is 17 and B is 5, 17 would be returned.

$$\text{Max_Value} = \max(\text{Value_A}, \text{Value_B})$$

Fig. 28: $\min(\sin(x), 0.5)$



MIN

Value A is compared to value B and the smaller number is the result. If value A is 2 and value B is 65, 2 would be returned.

$$\text{Min_Value} = \min(\text{Value_A}, \text{Value_B})$$

MODULO

Modulo is similar to a division, the dividend is divided by the divisor and the remainder (and not the quotient) will be returned. If the dividend is 12 and the divisor is 5, 2 would be returned.

$$\text{Remainder} = \text{Dividend} \bmod \text{Divisor}$$

MULTIPLICATION

The product of the multiplication of factor A by factor B is this function's result. If factor A is 3 and factor B is 5, 15 would be returned.

$$\text{Product} = \text{Factor_A} * \text{Factor_B}$$

SUBTRACTION

This module's result is the difference of the subtrahend and the minuend.

$$\text{Difference} = \text{Minuend} - \text{Subtrahend}$$

CONSTANTS

E

Returns the value of the Euler number, which is approximately 2.17.

$$\text{Value_E} = e$$

PI

Returns the value of π , which is around 3.14.

$$\text{Value_Pi} = \pi$$

VARIABLE

This just returns value A, nothing useful in a mathematical sense, but handy for building setups that are more complex or for using one value as basis for several different modules.

$$\text{Variable} = \text{Value_A}$$

BEHAVIORS

EASE-OUT

This behavior generates an elastic behavior or movement. Value A and value B represent the current and the desired value, and factor defines the elasticity. The resulting sequence starts at value A, moves fast towards value B and slows down.

$\text{Value_Ease_out} = \text{Value_A} + (\text{Value_B} - \text{Value_A}) / \text{Factor}$

SERIAL CHAIN

This behavior mimics the motion of objects connected by a rigid chain. The radius defines their distance. The calculation of object B's position is based on its current position, the radius and the position of the leading object (object A).

This behavior is useful for building up a chain of objects, which follow their leading objects but keep a defined distance from each other.

INPUTS

KEYBOARD (CURSORS)

Detects which of the cursor keys is pressed. The function returns either 0 or 1.

KEYBOARD (DIGITS)

Detects which of the number keys is pressed. The function returns either 0 or 1.

MOUSE

This module returns the mouse's position and the status of its buttons.

CONDITIONALS

AND

Returns 1 if value A and value B are true (not 0), otherwise, 0 is returned.

BIGGER

Returns 1 if value A is bigger than value B, otherwise, 0 is returned.

EQUAL

Returns 1 if value A is equal value B, otherwise 0 is returned.

NOT_EQUAL

Returns 1 if value A is not equal value B, otherwise 0 is returned.

OR

Returns 1 if value A, value B or both are true (not 0), otherwise 0 is returned.

FUNCTIONS

CYCLOID

A cycloid is '*... the path of a point on the rim of a wheel of radius R that rolls without slipping along the x -axis*' ([KR] p. 439). The graph drawn by the function looks like a sequence of half circles.

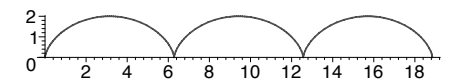


Fig. 29: a cycloid

ELLIPSE

Informally an ellipse is a squashed circle; mathematically an ellipse is defined by two points and a constant. Every point whose distance to both points is equal to the constant is part of the ellipse.

The ellipse's graph is based on horizontal and vertical value. Its proportions are defined by value A and value B, value T defines the current angle and should be kept between 0 and 2π .
 $\text{Horizontal_Value} = \text{Horizontal_Center} + \text{Value_A} * \cos(\text{Value_T})$
 $\text{Vertical_Value} = \text{Vertical_Center} + \text{Value_B} * \sin(\text{Value_T})$

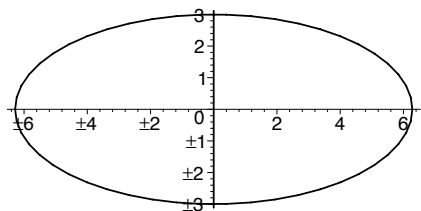


Fig. 30: an ellipse

LEMNISCATE

The graph of a Lemniscate looks like a horizontal 8. Useful to create some curvy movement.

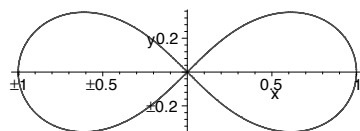


Fig. 31: a lemniscate

NATURAL LOGARITHM

The logarithm of a value A to a base is what you have to raise the base to the power of in order to get value A. The resulting graph moves fast along the y-axis but slows down soon.

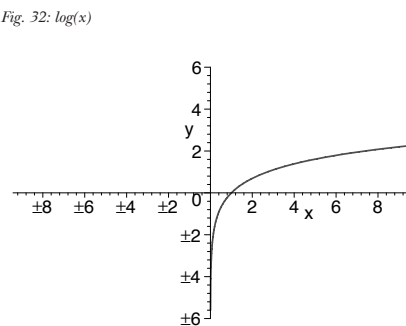


Fig. 32: $\log(x)$

POWER

Returns the Base to the power of the Exponent.

For instance if the base is 5 and the exponent is 2, 25 will be returned.

$\text{Power_Value} = \text{Base} \wedge \text{Exponent}$

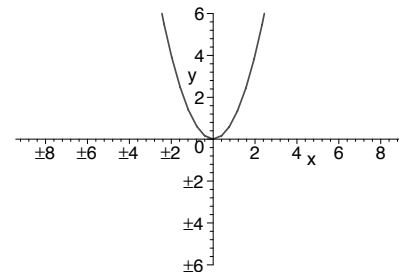


Fig. 33: x^2

RANDOM

This module generates a random number. The smallest and the biggest value define the range of possible values.

SINE

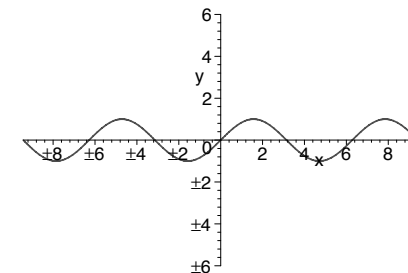
A sine-function is this module's result. The function is based on value A and its amplitude is scaled by a factor called 'Amplitude_Scale'.

$\text{Sine_Value} = \sin(\text{Value_A}) * \text{Amplitude_Scale}$

Sine is useful for smooth and periodic movement. The function's y-value meanders between -1 and 1.

Adding or subtracting sine curves results in overlays. Remember the addition of two sine curves with a phase-difference of π results in an extinction of both functions.

Fig. 34: $\sin(x)$



SPIRAL

This module draws a spiral.

The angle defines the current angle and the factor defines how fast the spiral is growing.

Horizontal_Value = Horizontal_Center +
(sin(Angle) * Angle * Factor)
Vertical_Value = Vertical_Center + (cos(Angle)
* Angle * Factor)

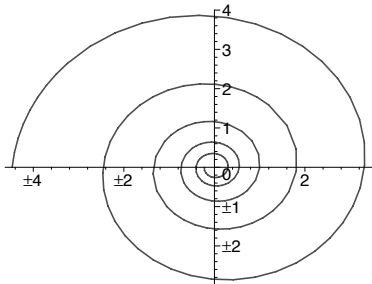


Fig. 35: a spiral

The easiest way to exchange the spiral's direction is to swap the horizontal with the vertical value.

STEP

This module returns a sequence of values starting at the smallest value, going up to the biggest value by adding the step value.

After reaching the biggest value the algorithm starts at the smallest value again.

Step_Value(n) = Smallest_Value + n *
Step_Value

STEP UP AND DOWN

This module returns a sequence of values starting at the smallest value, going up to the biggest value by adding the step value. After reaching the biggest value, the algorithm turns and goes down to the Smallest value again.

Step_Value(n) = Smallest_Value + n *
Step_Value

TANGENT

The function yields the tangent of a given value. The function is based on value A and its amplitude is scaled by a factor called 'Amplitude Scale'.

Tangent_Value = tan(Value_A) *
Amplitude_Scale

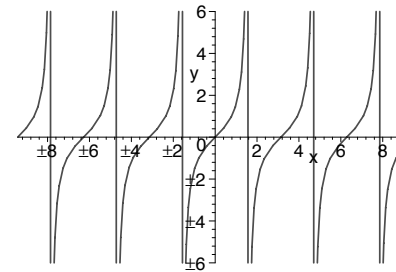


Fig. 36: tan(x)

UPCOMING FEATURES

Many additional ideas have come to the author's mind and dozen of great ideas have been proposed by friends and colleagues. These feature requests have not been implemented at the current stage in order to avoid losing the project's main focus. Nevertheless some of the ideas proved to be good and useful and therefore will be documented. If possible, the author will try to add these features (and many more) in the future. Information about freshly released updates and links to downloads will be available at <http://www.rinner.st/diploma/!scribble-builds>.

COLLABORATION

The idea of a collaborative version of !scribble evolved inspired by <http://www.vectorama.org/>. Vectorama is a website offering a playground for collaborative drawing. Up to ten users can draw on a shared canvas and chat with each other. The owners of the site offer huge libraries of pre-made vector graphics and an archive presenting snapshots of all drawings.

The collaborative version of !scribble could be done in a quite similar manner – a shared canvas for the users, libraries of objects and modules, archiving of all the results, an invitation to all users to upload their own objects and modules and a chat-area.

Thanks to recent improvements to Macromedia's multiuser server (UDP, server-sided scripting, access to the server's file-system) the implementation should be possible without any additional programming as required of the developers at Vectorama (they programmed their own multiuser-server as a Java-servlet).

CONCLUSIONS

IMPROVEMENTS OF THE USER INTERFACE

To make !scribble more usable and productive, several features should be added. The user should be able to change an object after placing it, for instance by exchanging it with another object or manipulating its form and color. To facilitate the building of complex settings, cloning of panels including their settings has to be implemented.

Debugging of settings would be much easier if the user could watch values directly in an additional window. Tools to allow rotating, scaling and skewing of objects during edit mode have to be added as well.

PROGRAMMING BY EXAMPLE LIKE FEATURES

By learning more about the ideas of Programming by Example the idea of an own PBE tool evolved. A product combining the stimulus response concept of Pavlov and !scribble's way of creating complex motion and interactivity could be a new and innovative

way of creating interactive multimedia applications. Some basic experiments about stimulus response based programming in Director have been already conducted, but the question of how to combine !scribble with PBE has not been fully answered as of this time.

STANDALONE-PLAYER

One of the main ideas of the project was to create a product which enables designers to create and present their experiments. To make the presentation even easier a standalone player for presentations will be developed. After starting the application it will automatically open a defined setting. Therefore no knowledge about !scribble is needed, and this player could be used, for instance, to send previews of ideas to clients in an easy and secure way.

By describing the current situation, it was shown that designers need new means to realize their ideas. The current tools did not prove especially useful for creating multimedia applications. The thesis describes Macromedia Director as a tool not suitable for experimentation with a user interface which doesn't allow the creation of complex prototypes without any programming. Tools which follow the principle of Programming by Example were described as useful for creating examples of interaction but none of them enabled the user to define multifarious motion. !scribble got designed as an open and versatile tool to empower designers and to facilitate the workflow of creation and production within multimedia projects.

At the current state, the tool is ready for being used by other interested designers and programmers. Further improvements are about enhancing usability or adding additional features, but will not affect the interfaces needed for creating new modules or importing/exporting settings files.

In the few cases when !scribble was shown to people who have not been involved in the thesis its main concept and current version was commented as good and useful. !scribble's development will not stop after the thesis. Plans to add the described upcoming features exist and !scribble will be presented at some meetings of Macromedia Director related mailing lists.

The thesis showed the industry's need for a fast and easy way of sketching ideas and provided a tool which is not just useful for creating some dummies but helps to express the designers' idea in a prototype which can get reused without any additional programming within real-world projects.

BIBLIOGRAPHY

[DE]

Deloura, Mark: *Game Programming Gems*. Rockland: Charles River Media, 2000.

[HA1]

Hashimoto, Osamu; Myers, Brad: 'A Graphical Styles for Building User Interfaces by Demonstration'. In: *Proceedings of the fifth annual ACM symposium on user interface software and technology*. 1992, p. 117 - 124.

[KA]

Kapitzki, Herbert W.: *Programmiertes Gestalten*. Karlsruhe: Verlag Dieter Gitzel, 1980.

[KR]

Kreyszig, Erwin: *Advanced Engineering Mathematics*. New York: John Wiley & Son Inc. (8th edition), 1999.

[LI]

Lieberman, Henry (ed.): *Your Wish is My Command. Programming by Example*. San Francisco: Morgan Kaufmann Publishers, 2001.

[LM1]

La Mothe, André: *Black Art of 3D Game Programming*. Corte Madera: The Waite Group, 1995.

[LM2]

La Mothe, André: *Tricks of the Windows Game Programming Gurus*. Indianapolis: Sams, 1999.

[MA]

Maeda, John: *Maeda@Media*. London: Thames & Hudson, 2000.

[MI]

Miyashita, Ken; Matsuoka, Satoshi; Takahashi, Shin: 'Declarative programming of graphical interfaces by visual examples'. In: *Proceedings of the fifth annual ACM symposium on user interface software and technology*. 1992, p. 107 - 116.

[MIT1]

Design by Numbers. DBN Courseware. <http://dbn.media.mit.edu/courseware.html>, last edited 2001-02-21, last viewed 2001-06-08.

[MIT2]

Design by Numbers. What is DBN?. <http://dbn.media.mit.edu/intro.html>,
last edited 2000-01-12, last viewed 2001-06-08.

[MJ-A]

Maeda, John; Antonelli, Paola: *Design by Numbers*. Cambridge: MIT Press, 1999.

[MY]

Myers, Brad A; McDaniel, Richard; Wolber David: 'Intelligence in Demonstrational Interfaces'. In: *Communications of the ACM*. 43, 3 (March), 2000. p. 82 - 89.

[PE]

Petre, Marian; Blackwell, Alan F.: 'Mental Imagery in Program Design and Visual Programming'. In: *International Journal of Human-Computer Studies*. 51, 1, 1999.
p. 7-30.

[SE]

Sedgewick, Robert: *Algorithmen*. Bonn, München: Addison-Wesley, 1991.

[SH]

Shneiderman, Ben: 'Foreword'. In: *Your Wish is My Command – Programming By Example*. Ed. Lieberman, Henry. San Diego: Morgan Kaufmann, 2000. p. v-viii.

[WO1]

Wolber, David: 'Pavlov: An Interface Builder for Designing Animated Interfaces'. In: *ACM Transactions on Computer-Human-Interaction*. Vol 4, No. 4 (December), 1997.
p. 247-386.

[WO2]

Wolber, David: Pavlov: 'Where PBD meets Macromedia Director'. In: *Your Wish is My Command – Programming by Example*. Ed. Lieberman, Henry. San Diego: Morgan Kaufmann, 2000. p. 345-350.