DOOPSI_Docs

COLLABORATORS			
	TITLE : DOOPSI_Docs		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		June 5, 2025	

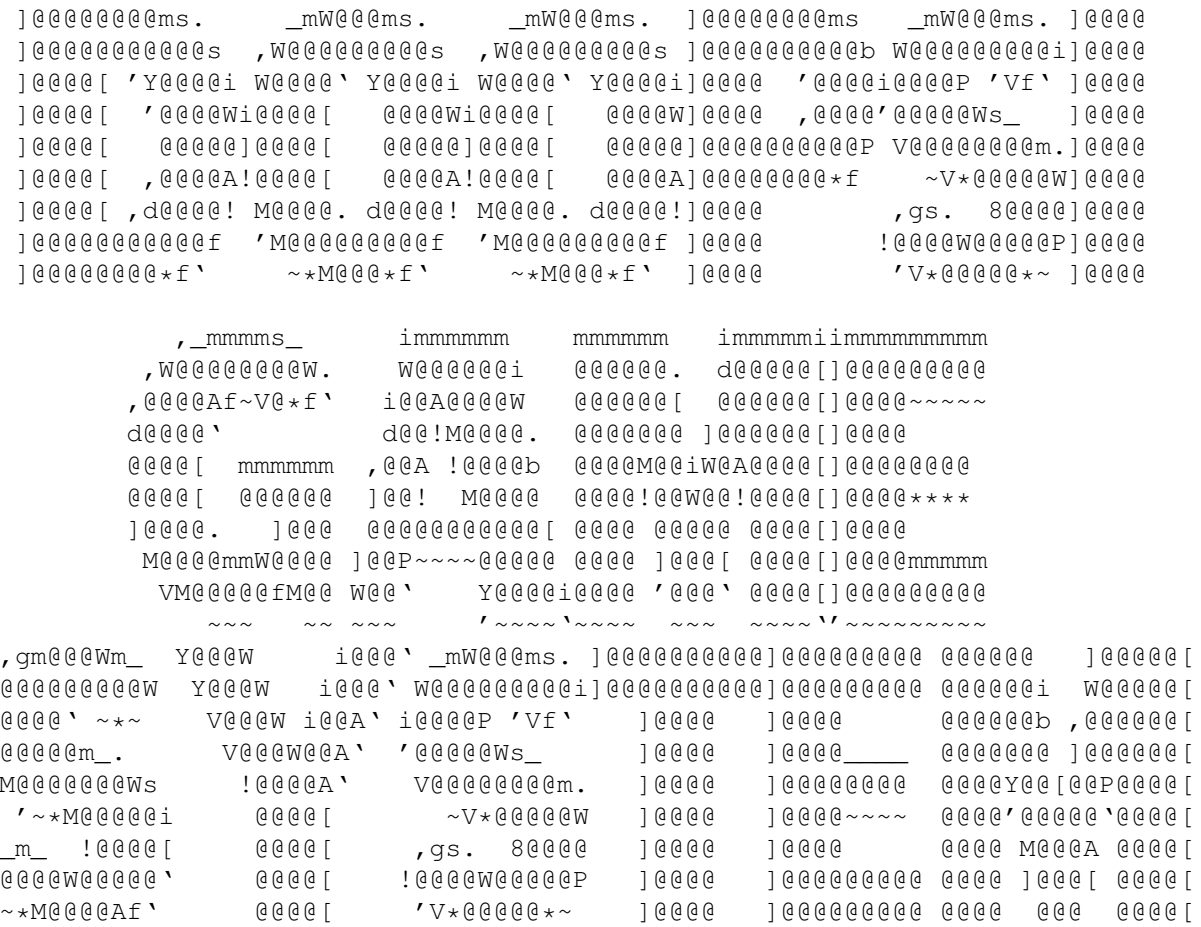
REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

DOOPSI_Docs

1.1 DOOPSI-GS - DOOPSI Game System



Version 1.10

DOOPSI PLAYER

1.2 DOOPSI-GS - Introduction

****
Introduction.
****

The name of the game.

Doopsi-GS means: Dynamical Object Oriented Programming System Interface - Game System. We named our program this way for many reasons: Doopsi remember us of Boopsi, the Object Oriented Intuition System of our beloved Amiga; it's an Object Oriented program in the technical meaning of the word, that is to say as far as its internal structure is concerned (some highlights of this structure are given in the Appendix C); it's a Programming Interface because it simplifies the the programming task making it completely automatic and quick.

This Programming System is very different from other products devoted to the same scope because it is extremely easy to use, even for people that has little programming skill: the user draws all the graphics and supplies all the sounds and musics he needs and then our program puts all things together. The idea that lies under all the inner workings of this program is straightforward: you first define the scenes, then put some objects on them and, at last, you specify how each one of the objects must react when the guy that is playing acts on it (this is the only coding part you have to face). Well, to be sincere there are other parts of the work I've not included in the previous description, but you'll learn them as you get more involved in Doopsi: for example, one of the tasks you have to deal with is to build the path along which the Man walks (the Man with the capital letter is the chief character of the adventure), or another task is to write the dialogs between the Man and some other object, and so on.

These are some features of Doopsi:

- OS sensitive (it works under V37+).

- completely localizable (it uses our "locator.library" to change languages, but in future versions we will adapt it to the standard "locale.library" if we can make it work also under V37).

- AGA compatible.

- the Editor is completely system friendly.

- handles bitmaps bigger than the screen area (with autoscroll).

- allows modules and samples to be played during the game.

- fully customizable (e.g.: you can also redraw the buttons of the Player).

The Doopsi Game System is made of two indipendent programs: the Editor and the Player. As their names might suggest, the Editor is the one that builds the adventure, while the Player exploits the datas produced by the Editor to make you play your game. You have to think to the Editor as generating some sort of program and to the Player as an interpreter of such a program: if occurs any runtime error the Player will prompt you with an error message and, if it can, continue the processing of the game, just as any interpreter of any language would do (well, actual languages can't continue after a runtime error, but we are better and we do!).

The manual has the following structure: this introducing section is followed by the description of the Installation procedure. Then we begin with the Main Window of the Doopsi Editor, containing the description of all the (many) buttons that appear (don't be frightened by them, you'll learn quickly how to use them). After the Main Window there come three sections about the three editors related to the scene construction: the Path Editor, the Spot Editor and the Screen Attributes Editor. Next there are the object related sections describing respectively: the Object Attributes Editor, the Doopsi Coder and the Anim Editor. There are only two editors left, and these constitute the subjects of the following sections: the Dialog Editor and the Preferences Editor. The main part of the manual is finished but we have also three appendices: the first describes all the Doopsi Language instructions (the Doopsi Language is used in the Doopsi Coder to program the objects); the second describes the outline of the console included in this package (the console is the bottom part of the game, the one with the actions on it); and the third contains some technical details about the inner structure of Doopsi Objects (this is not needed at a first reading, but it is included for more expert (or more curious) users). The Player also needs some explanation, and we describe it in the last section of the manual, located after all the appendices. That's all, folks.

One observation: probably this manual is a little chaotic, in the sense that the useful informations are spread all around and hidden between heaps of junks. This means that you have to be more careful in reading it not to skip some interesting (and vital) notions. We are sorry, but this is only the first version and we will continually rearrange it to improve it. So any comment is welcome and we will also keep a FAQ list in our Web pages.

1.3 DOOPSI-GS - Authors

****
The authors (the most interesting part of the manual):
****

this is the situation up to 27 - 6 - 1996 (date of first upload):

Andrea has just got a degree in Physics at the University of Milan, and is now studying to enter a Ph.D. course.

Fabio is working as a Project Manager at DeAgostini MultiMedia, in Milan.
He likes Amigas, cakes, girls, fun, OOP programming and DOOPSI ;)

Our addresses are:

Fabio Rotondo	Andrea Galimberti
c.so Vercelli, 9	via Villoresi, 87
28100 Novara	20029 Turbigo (Mi)
Italy	Italy
E-Mail: fsoft@intercom.it	
Phone: (ITA) - (0)321 - 459676	Phone: (ITA) - (0)331 - 871009

The Web pages are:

```
http://www.intercom.it/~fsoft/doopsi.html    -> DOOPSI Home Page!!!
http://www.intercom.it/~fsoft                -> Fabio Soft Home Page
```

DOOPSI Mailing list:

TO SUBSCRIBE send a mail to

```
fsoft@intercom.it
```

With:

```
SUBSCRIBE DOOPSI your_name@your_email.addr
```

in the body.

1.4 DOOPSI-GS - Greetings And Hellos

****
Greetings go to:
****

Mik and Gio of ClassX: for some useful routines. For their support, for their ideas, for beign them.

Giorgio Fornara: for the great graphics of the Demo that is not included in this package, but will appear sooner than possible. He has drawn also the Doopsi logo that appears on the Editor's screen.

Andrea Rotondo: for the Tutorial graphics.

Stefano Clemente: for beta-testing.

Lele and Paolo of Intercom: for creating DOOPSI - Mailing list.

Max Galimberti: for having tested this manual.

****
Hellos:
****

from Andrea to Laura: for having endured me until now.

from Fabio to his mommy (and also his father).

1.5 DOOPSI-GS - DISCLAIMER

****
LICENSE AGREEMENT FOR DOOPSI-GS
****

I. LICENSE RIGHTS AND RESTRICTIONS

Fabio Rotondo and Andrea Galimberti ("authors") grant you the non-exclusive, non-assignable right to use enclosed DOOPSI-GS software in object code form (the "DOOPSI") on a maximum of 1 computer system at one time. You may not reverse engeneer, decompile, or dissassemble DOOPSI, except to the extent that the foregoing restriction is expressly prohibited by applicable law. You may not rent or lease DOOPSI, or otherwise transfer DOOPSI and accompanying written materials. All rights not expressly granted are reserved by authors.

II. NO WARRANTIES

DOOPSI and accompanying written materials are provided "as is", without warranty of any kind. To the maximum extent permitted by law, authors declaim all warranties, either express or implied, including but not limited to implied warranties of merchantability, fitness for a particular purpose and noninfringment. The entire risk arising out of the use or performance of DOOPSI and any accompanying written materials remains with you.

III. NO LIABILITY FOR CONSEQUENTIAL DAMGES

To the maximum extent permitted by applicable law: in no event shall the authors or their suppliers be liable for any damages whatsoever (including, without limitation, damages for loss of business profits, business interruption, loss of business information, or other pecuniary loss) arising out of the use of or inability to use DOOPSI, even if the authors have been advised of the possibility of such damages. Because some states/jurisdictions do not allow the exclusion or limitation of liability for consequential or incidental damages, the above limitation may not apply to you.

IV. SHARING AND SPREADING

You are authorized in spreading and sharing this document along with all the rest of the archive, only if the program is a "SHAREWARE" version or it is a "SPECIAL EDITION". The "SPECIAL EDITION" may not be uploaded to Aminet or to any BBS and may not appear upon any magazine without the written permission of the authors. You can, and you are invited to do so,

copy and spread the original DOOPSI SHAREWARE VERSION archive, but you MUST NOT COPY in any form the DOOPSI REGISTERED VERSION, except for personal use.

V. MISCELLANEOUS

If you acquired this product in the United States, this Agreement is governed by the laws of the State of Washington. If you acquired this product outside the United States, local law may apply. If either party employs attorneys to enforce any rights arising out of or relating to this Agreement, the prevailing party shall be entitled to recover reasonable attorneys fees, costs and expenses.

1.6 DOOPSI-GS - Installation

****
Installation.
****

There is an Installer script to accomplish this task, but for the ones that like to do it manually we describe what should be installed and where. (By the way: to use the script you must have the Installer program and the Lha utility.)

The three archives have already the final directory structure. You have first to depack the Programs.lha archive; the latter creates the Doopsi directory which contains the Editor and the Player. There's also a second directory in which you find two libraries: the "locator.library" and the famous "reqtools.library"; you have to copy these libraries to your Libs: directory. The final step consists in making the following assignment: Assign Doopsi: to the Doopsi directory.

The default language is english, but are supplied also the catalogs to install the italian language: you just have to extract the Catalogs.lha in the same directory where you put the Doopsi drawer (not IN the Doopsi drawer, but NEXT to such a drawer). This creates a Catalogs directory in the Doopsi directory and also adds the file "player.catalog" in the "Doopsi/Data" drawer.

Then you have to depack the Tutorial.lha archive to the Same directory where you put the Doopsi drawer (not IN the Doopsi drawer, but NEXT to such a drawer): this will create a directory Tutorial in the Doopsi drawer, where you can find some graphical datas to start immediately.

To sum up: the archives have been built in such a way that, if you open them all in the same directory, their contents will go to the right places.

The resulting directory structure is:

```
DOOPSI (DIR)
|
+---Catalogs  (DIR)
|
```

```
+---Docs      (DIR)
|
+---palettes  (DIR)
|
+---Data      (DIR)
|
`---Tutorial  (DIR)
    |
    +---Objects   (DIR)
    |
    +---Berny     (DIR)
    |
    +---Scenes    (DIR)
    |
    `---Sounds    (DIR)
```

The Editor Must find in its directory the file "doopsi.dat". The Catalogs directory is needed only if want to use a language different from english (up to now only the italian catalogs are supplied): you copy in this drawer only the catalogs of your favourite language and leave out all the others.

The Player Must find in the directory you choose with the Editor (see the Preferences window section of this manual) the following files: "player.dat" and "panel.iff". The "panel.iff" is the console image you can change (see the Appendix B). The "player.catalog" file is needed only if you want to use a language different from english, otherwise you can throw it away.

The catalogs' sources will be supplied (if wanted), with the executables used to create them, in a separate package.

IMPORTANT NOTE: The Editor needs the Doopsi: assignment, but the Player has been built in such a way to be completely relocable (no assignments needed). When you start the Player from a Shell you just have to make the Player directory the current one and then call the program: the Player searches then for the directory you selected using the Editor (the default one is the "Data" directory created by the archives), where it Must find the files "player.dat" and "panel.iff".

The commandline to start the Player is: Player Doopsi_file_name. If you forget to give a filename, the Player will open a file requester and asks you to select a file.

1.7 DOOPSI-GS - Main Window

```
****************
The Main window.
****************
```

The story begins from here.

First of all, let's describe the window contents. The topmost part is

divided in two sections: the left one is related to scenes, while the right one is dedicated to objects. Then there comes a row with miscellaneous editors and, last but not least, the final row contains gadgets devoted to load and save your work as a whole.

Let's start with the scene section. To create a new scene you have to press the "New" button and, after having selected a background image from the file requester, you are asked to enter the scene name: this name is the identifier that Doopsi uses to address the scene, so you are invited not to call two different scenes with the same name. After the background has been displayed you are ready to work on this scene (e.g.: putting objects on it). To select the scene to work with, you have to use the "List" gadget: click on the desired name in the list and then on the "Ok" button. The "Kill" gadget is for discarding the current scene.

When you press the "Save" button the editor asks if you want to save only the current scene or all the scenes created up to now. With the "Load" gadget you can load both types of file: the Editor automatically recognizes if the file you selected contains one or more scenes. If the objects linked to the scene that you are going to load have already been loaded, then the editor can immediately position them on the scene itself; if you load the objects after the scene has been loaded, then you have to position such objects yourself, by clicking on the "R" gadget ("R" stands for "refresh"). The "Load" gadget can also extract the scenes' informations from a file saved with the "Save All" gadget located at the window bottom row (see the gadget's description for more informations).

The remaining gadgets all open new editors, and you are referred to the sections of this manual dedicated to such editors for more specific informations. The "Attrs" one is for entering the Screen Attributes editor, where you can change some features of your scene; the "Spot editor" button calls the Spot Editor to put Spots on the scene; the "Path editor" gadget enters the Path Editor used to edit the path on which your Man is supposed to walk.

Now you know how to create a new scene and you are ready to put objects on it, so let's have a look at the object section. The "New" gadget creates a new object and opens the Object Attributes editor to allow the user to edit such an object (see the section about the Object Attributes editor). The "List" gadget displays the list of all objects created up to now: to work on an object just make it the current one by selecting its name from the lister. The "Kill" button discards the current object.

The "Save" gadget asks if you want to save only the current object or all the objects created. The "Load" button recognizes if the file you selected contains one or more objects and adds them to the list. This button can also extract the objects' datas from a file saved with the "Save All" gadget.

The "Attrib" gadget opens the Object Attributes editor (the one that is entered automatically when you create a new object); the "Doopsi code" button enters the Doopsi Coder used to specify what the Player should do when the user acts on an object; the "Anim Editor" gadget calls the Anim Editor that allows you to convert an object to an AnimObject by building an animation from its images.

You can move all the objects you have created and positioned on the scene (using the Object Attributes editor) by clicking on them with the left mouse button and (holding this button down) dragging them around the scene. By clicking on them with the right mouse button you make the selected object the current one and call the Object Attributes editor.

The next section contains three buttons: the "Prefs" one enters the Preferences editor where you can change for example some names the Player uses as defaults, or your favourite paths, and so on; the "Dialog Editor" calls the Dialog Editor used to create dialogs between your Man and any other character of your story; the "Info" gadget tells you how many objects and scenes you created up to now and the amount of free memory. The "Test Game" button allows you to test your adventure by running the Player: the Editor searches for the Player in the directory you specified in the "Player" field located in the Preferences Editor.

Now we describe the last row. The "Save All" gadget is for saving all the adventure: the file contains scenes, objects, paths, spots, dialogs and preferences, that is to say all the Player needs to start your game. The "Save Final" button differs from the "Save All" gadget because the file saved will be encrypted: the Editor asks you the key used to encrypt the file. We added this option because in such a way other people cannot look into your main Doopsi file to see how you built your adventure, while they still can play with it without knowing the key. The "Load All" gadget allows you to load a file saved with the "Save All" or the "Save Final" gadget, if you want to modify it (if the file is encrypted you have to know the key to decrypt it before loading the data contained). The "Kill All" button is to delete all your previous work and restart from the begginning. The "About Doopsi" button displays a little message you ought to read.

Gadget reference.

Scene section:

New	Creates a new scene: you are asked to select the background image and to enter the name of the scene.
Load	Loads a file containing one or more scenes.
Save	Saves a file containing some scene’s data: you are asked if you want to save only the current scene or all the scenes created up to now.
List	Lists all the scenes created and allows you to move through the scenes by selecting the current one.
Kill	Discards the current scene.
Attrs	Opens the Scene Attributes editor.
Spot editor	Calls the Spot editor.

Path editor	Enters the Path editor.
R	Refreshes the screen.

Object section:

New	Creates a new object and opens the Object Attributes editor to edit it.
Load	Loads a file containing one or more objects.
Save	Saves a file containing some object's datas: you are asked if you want to save only the current object or all the objects created.
List	Lists all the objects: you can move through them by selecting the current one.
Kill	Discards the current object.
Attrib	Opens the Object Attributes editor.
Doopsi code	Calls the Doopsi Coder.
Anim editor	Enters the Anim editor.

Other gadgets:

Prefs	Opens the Preferences editor.
Dialog editor	Enters the Dialog editor.
Test Game	Runs the Player.
Project Info	Displays the some infos about your work.
Load All	Loads a file containing all the game's datas (if the file is encrypted you are asked for the key to decrypt it).
Save All	Saves a file containing all the game's datas, that is to say the file used by the Player.
Save Final	Saves an encrypted file containing all the game's datas: the Editor asks for the key to encrypt the file.
Kill All	Deletes all your previous work.
About	Some notes about us and Doopsi.

1.8 DOOPSI-GS - Notes

```
****************************************
Some notes on how the Player finds the Man.
```

```
*****************************************
```

This problem arises because the Man is an AnimObject (actually four objects: one for each direction) like all the other AnimObjects you may link to your game. Thus the Player has to know which objects represent the chief character of the story. The Man's internal name is composed of a root name (the default one is "Man") and an extension indicating the direction (e.g.: U for up, D for down, etc.): so the AnimObject containing the animation representing the Man walking right may be called (internal name always!) "ManR", and so on. You need also other four AnimObjects representing the Man speaking in the four directions: the name of such AnimObjects is built from the root name, adding to it a first extension (the default is "Talk") and the second extension is the direction as before. So the tipical name of an AnimObject showing the Man speaking to the left is "ManTalkL".

You can choose your favourite root and extensions by writing them in the Preferences editor.

A last note about extensions. The Doopsi Coder contains instructions used to move (and/or animate) objects on the scene during the game: you are asked to supply the name of the AnimObject containing the needed animation. Here is the interesting part: if such a name ends with one of the four extensions indicating a direction, then the Player is allowed to choose the needed direction. It does so by removing the extension and replacing it with the one representing the right direction, then it searches for an AnimObject with the resulting name an, if found, uses it. For example: if the name you selected as argument for the SetMoveAnim instruction is "FlyR" and the fly has to move up, then the Player searches for an AnimObject called "FlyU". In other words: the extension you put in is unimportant because it is going to be replaced before use, but it does tell the Player to be careful about directions. To see which instructions have this feature consult the Appendix A.

General observations you should read:

The Editor produces a file containing your adventure's structure, but not the graphics and sounds you linked to the game because we don't want to merge all the adventure in only one enourmous file; thus, in the file generated there are only the paths of the linked files. You have to remember this if you want to distribute your game to friends: for example, you can make the paths relative by using an assignment.

1.9 DOOPSI-GS - Path Editor

```
************
Path Editor.
************
```

Introduction.

The Path Editor helps adventure creators to develop paths on which the Man (or other objects) will move. The creation of paths is probably one of the

more complex task you have to face and requires some attention. The path created in a scene will limit the Man's movements during the game. This requires the creator to plan accurately the creation of his path.

The path theory.

A path is made of nodes and branches: the nodes are the stop points. In every node you can store some special commands that allow a better control of the Man's movements in the scene, but we'll talk later about these features. Every node has some branches (max. 7 per node) that allow links with other nodes.

The path determines the Man's freedom: the path is a sort of "web", an "invisible railroad" on which the Man will move. There's no way to make the Man walk outside the path: if you ask the Man to reach a point outside the path limits, then the Man will reach only the point nearest to the one selected. This feature isn't a limit: in this way you can even create more than one path in a scene and forbid certain areas until a particular condition is verified.

With a bit of practice, you'll certainly design complex and various paths. Path Editor will trustly follow you step by step in this task. Happy working.

NOTE: the following tutorials presume you already know other features of Doopsi, such as scene creation, anim creation using AnimWorkShop and object handling with Object Attributes.

1.10 DOOPSI-GS - Path Tutorial 1

Tutorial 1 - A Simple Path

In this tutorial we'll learn to draw a little path of four nodes and to link these nodes to each other with very simple branches. The aim of this tutorial is to get you acquainted with Path Editor's basic commands. You'll learn to add and delete nodes, to add branches and to force redrawing of the scene (needed in some occasions). In the next one, we'll talk about more complex arguments. Execute this tutorial more than one time, until you'll be sure you have complete knowledge of the commands.

To follow this tutorial, you must have already created a scene called "tutorial_room1".

Press the "Path Editor" button in the Doopsi Main Window.

Do not be afraid of the great number of buttons because, as time goes by, you'll perfectly know them.

First of all, it is better to place all nodes on the screen. Naturally, you can add some of them later, but in this way you'll be certain to cover all the important locations.

To add a node on the scene, you just have to do these steps:

- Press "Add Node(s)"
  From now on, until you choose to stop, every mouse click on the scene will generate a new node.

- Create four nodes, arranged in a square.

- Press "Add Branch(es)" to add branches.
  From now on, until you choose to stop, you'll be able to add branches between nodes.

- Press on the first node (it will change colour)
- Press on the second node (it will change colour and a straight line will link these two nodes)

We have just created the first link. To create another one, follow these steps:

- Press on the second node (it will change colour)
- Press on the third node (it will change colour and a strainght line will link these two nodes)

Follow these steps to link each node to the next, creating a box.

Now that you have created these nodes, you can test the movements that the player will be able to do during the adventure:

- press the button "Try Movements" and select the starting node (for example, the first). The node will change colour.

- Select the ending node (for example, the third). The node will change colour.

The program will try to compute the path between the two selected nodes and will show it, step by step, highlighting the nodes. If you have selected the first and the third node and you have linked them in the way we described in the example, you should see that the Editor highlights nodes one, two and three. Naturally this is a quite simple example, but the possibility of testing a path is very useful in complex scenes.

Now create a new node where you want and link to it other nodes, following the previously described steps. Try again to use the "Try Movements".

Before we end this tutorial, let's try to delete a node from the scene:

- Press "Kill Node"
- Select the node you wish to remove.
- The node will be erased.

At this point, some graphics on the scene could be damaged and the node you have just erased seems to be still there. This is not a Doopsi problem, and you need to refresh completely the graphic. Press "Redraw All" to

update the graphic, you'll see that now the deleted node will not appear anymore.

1.11 DOOPSI-GS - Path Tutorial 2

Tutorial 2 - Node Programming

Paths are very powerful tools. To exploit them to the best, you need to exercise for a little while. What we'll try to explain here is how to "code" a node.

One node can be coded to do these activities:

- Modify the Man's animations.
- Modify priority man/objects
- Set changing of a scene
- Check whether the node is available or not (checking condition)

All of these are complex operations and require a deep study of path generation.

Let's see how it is possible to code a node and what are the conditions to take into account while projecting a path.

To code a node, you have just to press the "Code" gadget inside the Path Editor and then select the node you wish to modify.

- Modify the Man's animations.

  Press the "Set Anim" button: a Lister will pop up with all objects (both animated and not). Select the one you want (Must be an AnimObject).

- Modify Priority Man/Objects

  Press the gadget "Change Pri", the selection "Man On Objects" and "Objects On Man" will swicth.

- Set Changing Of A Scene

  Press "Set Start", a Lister with all scene names will pop up. Select the one you wish to go to when that node is touched by the Man.

- Check whether the node is available (checking condition)

  Press "Set Condition", a Lister with all objects will pop up. Select the object to check, a Status Requester will pop up: insert the Status value you wish to check for.

Here there is a little example:

Suppose you have the following nodes:

```
    1 ---- 2 ---- 3 ---- 4 ---- 5
```

We want that node 5 takes us to the "Office" scene, but only when the "door" object is open (let's suppose that Status = 1 when the door is open). We have also to take care that the Man CAN NOT reach the node 5, when the door is closed.

Let's start to program node 5: press the "Code" button and then click once on node 5; then click on "Set Start", selecting the "Office" scene: from now on all the times the Man gets on node 5, he will be moved to the "Office" scene.

Now code the node n.4: press the "Code" button and then select node 4; click on "Set Condition": select the "door" object from the lister and insert Status value 1. Done! Now, if the door is closed, node 5 cannot be reached.

1.12 DOOPSI-GS - Path Editor General Description

General description.

Well, probably this is the editor with more buttons, but these gadgets are gathered together in groups. We'll examine one group at a time.

The first group I'm going to describe is the one located at the upper left corner of the window. The "Add node(s)" buttons activates the "adding mode", that is to say, after you pressed on this gadget you can add how many nodes you need until you are satisfied: you just have to click on the scene at the point where you want to put a node. To exit the "adding mode" you have to press again the "Add node(s)" gadget (or another gadget in the same group). The "Kill node" button asks you to select the node you want to kill, and then the Editor will discard it from the scene: the "dead" node will leave a black square on the scene, so if you want to refresh your graphics you have to click once on the "Refresh All" button. The "Refresh All" gadget redraws all the graphics contained in the current scene. The "Add Branch" button is used to enter the "adding mode" (this time of branches, obviuosly): to add a branch between two nodes click on the starting node and then to the ending one. You can repeat this steps until you have finished linking all your nodes; then press the "Add Branch" button again (or any other button in the same group) to exit the "adding mode". The "Code" gadget asks you to select the node to code; to end the coding session click again on the "Code" button: the Editor will ask you for another node to code, but you can click on the "Message" field to make it stop definitely.

The coding of nodes brings us directly to the group entitled "Change Settings". The "Set Anim" button is used to change the Man's animations when the Man walks on that node: the Editor opens a lister with all the names of the objects created and waits the user to select an item from the list. At this point the string "Default" at the left of the gadget will change to the chosen name. From this node on, the Player will use the root part of the chosen name as the root to build all the Man's animations names (if this doesn't mean nothing to you, then you should read the "Some notes on how the Player finds the Man" located right after the Main Window

section of this manual). The "Change Pri" gadget switches the priority from "Man On Objects" to "Objects On Man" and back: the former means that the Man will appear in front of all objects when walking toward that node, the latter means just the contrary (you can find some more notes about how this works in the Player section of the manual). The "Set Start" button opens a lister with the names of the scenes created so far: the name of the selected scene will appear at the left of the gadget in the "Start" field. If this field is nonempty then the Player, when the Man walks on that node, searches for a scene with the chosen name; if the scene has been found then the Player will leave the previous scene and enter the new one. In the new scene the Man will be positioned on the node containing (in the "Start" field) the name of the scene where the Man came from. (It's as if the "ending" node of the first scene points to the "starting" node of the second one and vice versa.)

The next group is the "Condition Flags". The "Set Condition" button allows the user to stop the Man's walk if a condition is not satisfied: the condition is of the type "If status of object is equal to". The button will bring up a lister with the names of the objects; after you selected an object, you have to enter the status' value you want to check for. If the Status field of the selected object doesn't contain the chosen value when the Man walks on that node, then the Man is not allowed to walk further than such a node. The "Erase Condition" clears the condition fields of the node.

The "Try Movements" button allows you to preview how the Man will walk on your path: you are asked to select the starting node and the ending node; then the Editor will highlight the nodes on which the Man will walk in going from the starting node to the ending one. The "Check Nodes" will check if all the nodes are properly linked and will remove any isolated node.

We have only three gadgets left. The "New Path" button erases completely the current path. The "Save Path" gadget saves the current path to a file, and the "Load Path" gadget loads a file saved with the previous button.

Some useful hints:

Always press the "Check Nodes" button before saving all
your work, because in this way you will prevent yourself from unpleasant surprises.

When positioning nodes don't forget the Man's dimensions: the hot spot of the Man is located at the bottom left corner of the Man's shape, so the Man stops always to the right of a node. This implies that the Man can reach the right edge (ot the top edge) of the scene before reaching the destination node if you position such a node too close to the edge of the scene: when the Man reaches the edge of a scene it stops, and so your node will never be reached.

When deciding the priority of the Man on objects take into account the Man's dimensions: if a node says "Objects On Man" and the following one says "Man On Object" and the Man, in going from the first node to the second one, is still partially covered by an object, then it can suddenly

be positioned in front of the covering object. So you may often need to select the priority "Objects On Man" even for a node that is not covered by any object to be sure that the Man is completely uncovered by objects before changing its priority.

Gadget reference.

Add Node(s)	Enters the "add mode": click on the scene to position how many nodes you need, and then click on this gadget again to exit the "add mode".
Kill Node	Asks you to select a node and then erases the selected node. Click again on it to disable the operation.
Add Branch	Enters the "add mode": click on the starting node and ending node of a branch until you have positioned all the links. Press again on this gadget to exit the "add mode".
Code	Enters the "coding mode". When you have finished to change the attributes of the node press this button again.
Refresh All	Redraws all the graphics of the scene.
Set Anim	Changes the "Anim" field of the node.
Change Pri	Changes the priority of the Man with respect to the objects.
Set Start	Selects the scene to go to when the Man walks on the node.
Set Condition	Sets the condition to be satisfied for the Man to walk further than the node.
Erase Condition	Clears the condition fields of a node.
New Path	Clears the current path.
Save Path	Saves the current path to a file.
Load Path	Loads a path from a file.

1.13 DOOPSI-GS - Spot Editor

****
The Spot editor.
****

This allows you to put "Spots" on the scene: they are simply little crosses with two coordinates and a name. You use this name to address a particular spot and Doopsi will use its coordinates. Spots are used, for example, with the Doopsi instruction MoveObject to mark the place where an object should go.

Spot Tutorial

Spot Editor General Description

1.14 DOOPSI-GS - Spot Tutorial 1

Here is a TUTORIAL:

- Create a new scene (using the "New" gadget in the scene section of the main editor window).

- Enter the Spot editor.

- Click on the "Add Spot" gadget: a message will appear saying "Add mode enabled".

- Click on the scene where you want to put a spot: you can repeat this last operation until you have positioned all your spots.

- Click again on the "Add Spot" gadget to disable the Add mode.

- Click on your scene to select a spot: a message will appear saying "Get Info"

- You can modify the currently selected spot name by just writing it in the string gadget "Spot Name" and pressing Return.

- Quit the Spot editor.

1.15 DOOPSI-GS - Spot General Description

General description.

To add some spots you just have to click on the "Add Spot" gadget to enter the Add mode: then you click on your scene wherever you want a spot. Because a spot without a name is nonsense the Spot Editor gives a default name to your spots, a name you can change at any time simply by rewriting it in the string gadget called "Spot Name" and pressing Return. From now on you can add spots at your will. When you are satisfied, the Add mode can be quitted by pressing the left mouse button again on the "Add Spot" gadget or on the "Kill Spot" one or on any other part of the editor window. To kill a spot press once the "Kill Spot" gadget: you are prompted to choose a spot to kill, so you have to click on the selected spot. That's all. By the way, the Kill mode is quitted immediately after you have killed (or decided not to kill by clicking again in the editor window) a

spot, thus to kill one more you have to press again the "Kill Spot" gadget, and so on.

The default mode (when you are not in the Add or Kill mode) is called "Get Info" mode: you are free to click on a spot to higlight it and display its name in the string gadget. As I told you before, this name can be modified every time it is displayed.

There is one more gadget left: the "Refresh" button. When you have positioned (and killed) a lot of spots your scene may appear in a mess: click on this gadget and Doopsi will clean the room for you.

Gadget reference:

Add Spot	Enters (and exits) Add mode.
Kill Spot	Deletes a spot.
Spot Name	Modifies the spot name.
Refresh	Redraws the scene.

1.16 DOOPSI-GS - Screen Attributes

****
The Screen attibutes editor.
****

This editor allows you to change the name of the current scene, to load a different background without removing objects, path and spots you have already linked to that scene, and other useful things...

The bottom row says something like this: "Use Action Open of Object foo". In other words the Doopsi code contained in the specified action of the selected object will be executed every time you enter the scene during the game. To switch off this option you have to clear the object's name in the string gadget. A useful hint: if you want the code to be executed only the first time you enter the scene (or on special occasions) you can exploit the status field of the object: you set it to a particular value and start the Doopsi code with the instruction "IfStatus".

Gadget reference:

Change Background	Loads a new background for the scene. If at least a node of the existing path or a spot falls outside of the bitmap then the editor will warn you without deleting the offending item.
Scene Name	Changes te scene internal name.
Music Name	You can insert a name of a module file that will be played every time you enter the scene (during the

game, obviously) and will be stopped when you exit the scene. You can click on the gadget to open a file requester and select the module file, or you can write directly its name in the string gadget.

Use Action — This cycle gadget allows you to choose the action from which to read the Doopsi code that is executed when you enter the scene during the game.

of Object — Selects the object from which to read the Doopsi code specified by the "Use Action" gadget. This button opens a lister with all the objects' names. You can also enter the object's name manually using the string gadget.

1.17 DOOPSI-GS - Object Attributes

```
*****************************
The Object Attributes Editor.
*****************************
```

This editor allows you to specify an object characteristics and to put it on the scene.

Tutorial 1
Tutorial 2

Object Attributes General Description

1.18 DOOPSI-GS - Object Attributes Tutorial 1

A first tutorial: "the little sheet of paper".

- I suppose you already loaded a scene to put objects on it. Now create a new object by pressing the "New" gadget in the object section of the main window.

- Enter the Object Attributes editor.

- In the "Name" string gadget enter "the paper".

- Click on the "Internal Name" string gadget and write "paper".

- Click on the cycle gadget to select the "Graphic" object type because our object is going to be drawn on the scene.

- Press the "Add" button and, using the file requester, choose the "paper" image in the Tutorial directory. Now the image is linked to the object.

- Eventually press the "Pos" button to position the object on the scene.

- You can move the object to its final location by selecting it with the

mouse and dragging it around the scene.

1.19 DOOPSI-GS - Object Attributes Tutorial 2

A second tutorial: "an empty zone".

- Create a new object just as in the previous tutorial.

- Enter its name and (more important) internal name.

- Click on the cycle gadget to select the "Empty" object type: our object will be an empty frame (not drawn on the scene during the game).

- Press the "Set Zone" button: a rectangle will appear on the scene.

- Drag the rectangle around the scene with the mouse and scale it using the gagdet in the lower right corner of the rectangle itself.

- When you're done, click on the "Pos" gadget to link this object to the scene.

1.20 DOOPSI-GS - Object Attributes General Description

General description.

You can enter this editor by clicking on the "Attrib" button in the object section in the main window or by clicking the right mouse button on an object already positioned on the scene.

The first fields you have to fill are the "Name" and the "Internal name" string gadgets: the name of an object is the string that appears on the Player console display line when the pointer is on that object; the internal name is the identifier Doopsi uses to address an object: for this reason you are invited not to duplicate such a name.

Next comes the shapes zone. You have to click on the "Add" button to add some shapes to your object: the editor will open a file requester and, after you have selected a valid image file, it will show the image name (with full path) in the lister. To kill a shape just click on it and then press the "Kill" button. Then you have to tell the editor which shape to use when drawing the object on the scene; to do this select a shape in the lister and then press the "Default Shp" gadget: a number will appear showing you which one is the currently used shape. A similar procedure is needed to tell the Player which is the inventory shape: click on a shape and then on the "Inv. Shp" button. By the way: inventory shapes must be 32*32 pixels, otherwise they will be clipped before being displayed by the Player. There's one more gadget left: the cycle gadget is needed to choose if you want a "Graphic" object or an "Empty" one. The former has a default shape which is blitted on the scene, while the latter is only an empty rectangle used to frame something belonging to the background. To position and size such a rectangle you have to click on the "Set Zone" button: an Empty Zone will be drawn on the scene, and you can drag it with

the mouse by clicking in it; to resize it just click on the solid square in the lower right corner of the zone itself. It's important to notice that even Empty objects can have some shapes linked to them: you can use these shapes with the inventory related commands of the Player. Each object has room for more than one shape and more than one inventory shape because during the game you can change the object current shape or inventory shape using the shape related commands of the Player.

In the "Status" gadget you can enter the object status default value. The status field is a general purpose integer with no particular meaning, so you can use it for whatever you like, e.g.: you can store here the Open/Close status of a door by just writing 0 for Close and 1 for Open, and you can modify this status during the game with the ChangeStatus and other instructions of the Player.

When quitting this editor remember to always press the "Pos" button if you want to link the object to the scene, even if you already linked it before: this is because the "Ok" gadget accepts your modifications of the objects settings but always unlink the object from the scene. There's no need to explain what the "Cancel" gadget does.

Gadget reference:

Name	The object name.
Internal Name	The object internal name.
Default Shp	This integer points to the shape used to blit the object on the scene. It has no meaning for an Empty Zone.
Inv. Shp	This integer points to the shape used by the Player to store the object in the inventory (the dimensions of this shape must be 32*32 pixels).
Empty/Graphic	Selects the object type.
Add	Links a shape to the objects and adds its name to the lister.
Kill	Removes a shape from the lister.
Status	Modifies the status default value.
Set Zone	Creates the bounding box of an Empty Zone.
Pos	Quits linking the object to the scene.
Ok	Quits without linking the objects to the scene (if the object is alreay linked, then this gadget unlinks it).
Cancel	Quits restoring the previous settings.

1.21 DOOPSI-GS - DOOPSI Coder

****
Doopsi Coder.
****

Introduction.

The Doopsi Coder is the editor that allows you to "program" the objects to react to the user's actions. As far as the Doopsi Player is concerned, objects are only complex data structures without a precise meaning: the task of the Coder is to coordinate the user's actions with the behaviour of every object. The Coder is probably the most important part in the editing of an adventure, because it allows the user to give a "real" appearance to the objects on the scene and allows the player to interact with such objects.

The fundamental idea is very easy: to every action allowed might correspond a reaction from the selected object. "Might correspond" because you need not define a reaction for every action on every object of your adventure. If, for example, the player needs to talk to a door, this door is not compelled to answer (or probably it's a magic door...).

The coding of Doopsi objects is performed with the aid of the Coder that puts at your disposal an easy and quick environment to program. Don't worry if you haven't a sufficient programming skill: you are not expected to write even a single line of code because the Coder does this for you: all you have to know is exactly how an object must react to a given action. All the rest it's easy.

General notions of OOP in Doopsi.

Doopsi means "Dynamic Object Oriented Programming System Interface". Good programmers may note that in Doopsi there isn't an actual OOP interpreter, but the object-code interface is managed by Doopsi itself. For this reason the coding in Doopsi is "dynamical": objects can change shape and features "on the fly". Objects are seen as "beeings", a set of data and characteristics that make them unique. All data contained in an object is private and not directly accessible by the user. Access is granted only by means of the instructions that constitutes the Doopsi language. Many instructions require as a first parameter the name of the object on which they act. This allows also to modify parameters of an object that is not the current one (the one which you are acting on). Every object has a "Status" field, a value that can be modified with the Doopsi language and that can represent whatever the user likes: the Status value is arbitrary and doesn't depend on the special nature of the object, but on the user needs.

The Doopsi language instructions are thoroughly described in Appendix A

Note: the following tutorials suppose that you already know something about Doopsi, like the Object Attributes editor. We refer the reader to the chapter related to such an editor to revise how to create and edit a new object.

Tutorial 1
Tutorial 2

DOOPSI Coder General Description

1.22 DOOPSI-GS - Coder Tutorial 1

Tutorial 1 - "A door that open and close"

To use this example you are supposed to have created an object called (internal name) "door" with two shapes linked to it, the first representing the door closed ("door_closed") and the second representing the door open ("door_open").

- From the main window make the "door" the current object and enter the Doopsi coder.

- Press the "Open" button to code the object reaction to the action "Open". Now execute the following steps:

- Instruction "IfStatus"
- Select the "door" object
- Insert 0 when you are asked for a value
(in the Coder window will appear the line: IfStatus SELF, 00)
- Instruction "ChangeStatus"
- Select the "door" object
- Insert the value 1
(in the Coder window will appear the line: ChangeStatus SELF, 01)
- Instruction "ChangeShape"
- Select the "door" object
- Select the name of the shape representing the door open
(in the Coder window will appear the line: ChangeShape SELF, 01)
- Instruction "Else"
- Instruction "ShowText"
- When asked for a string enter: "The Door is already open"
(in the Coder window will appear the line: ShowText The Door is already open)
- Instruction "EndIf"

At the end of the previous steps you should have in the Coder's window the following code:

```
IfStatus SELF, 00
ChangeStatus SELF, 01
ChangeShape SELF, 01
Else
ShowText The door is already open
EndIf
```

This means (line by line):

If the status of the object itself (SELF -> "door") is 0
Change the status of the object itself (SELF) to 1
Change the shape of the object itself (SELF) to 1 (the door open)
Else
Show the message "The door is already open"
End of the If group

```
Now we are going to write the code relative to the "Close" action. Press
the "Close" button to code such an action, and execute the following steps:

   - Instruction "IfStatus"
   - Select the "door" object
   - Insert the value 1
   - Instruction "ChangeStatus"
   - Select the "door" object
   - Insert the value 0
   - Instruction "ChangeShape"
   - Select the "door" object
   - Select the name of the shape representing the door closed
   - Instruction "Else"
   - Instruction "ShowText"
   - Enter "The door is already closed"
   - Instruction "EndIf"

The following code will appear:

  IfStatus SELF, 01
  ChangeStatus SELF, 00
  ChangeShape SELF, 00
  Else
  ShowText The door is already closed
  EndIf

"Translated" into everyday words this means:

   If the status of the object itself (SELF -> "door") is 1
   Change the status of the object itself (SELF) to 0
   Change the shape of the object itself (SELF) to 0 (the door closed)
   Else
   Show the message "The door is already closed"

At this stage the coding of the "door" object is completed. You can press
the Ok button to quit the Coder and return to the main window.
```

1.23 DOOPSI-GS - Coder Tutorial 2

```
Tutorial 2

In this tutorial we will show you how to modify the characteristics of an
object from another object. You are supposed to have already created two
objects, the first named "lamp" with two shapes (representing respectively
the "light_on" and the "light_off"), and the second named "button" with
only one shape (of a button, of course).

- Select the "button" object and enter the Coder.

- Press the "Push" gadget to edit the code relative to that action.

- Execute the following steps:

   - Instruction "IfStatus"
```

```
   - Select the "lamp" object
   - Insert the value 0
   - Instruction "ChangeStatus"
   - Select the "lamp" object
   - Insert the value 1
   - Instruction "ChangeShape"
   - Select the "lamp" object
   - Select the shape named "light_on"
   - Instruction "Else"
   - Instruction "ChangeStatus"
   - Select the "lamp" object
   - Insert the value 0
   - Instruction "ChangeShape"
   - Select the "lamp" object
   - Select the shape named "light_off"
   - Instruction "EndIf"

In the Coder window will appear the following program:

  IfStatus lamp, 00
  ChangeStatus lamp,01
  ChangeShape lamp,01
  Else
  ChangeStatus lamp,00
  ChangeShape lamp,00
  EndIf

This, "translated", means:

   If the status of the "lamp" object is 0 (-> the lamp is off)
   Change the status of the "lamp" object to 1 (-> the lamp is on)
   Change the shape of the lamp to "light_on"
   Else (the lamp is already on)
   Change the status of the "lamp" object to 0 (-> the lamp is off)
   Change the shape of the "lamp" object to "light_off"

As you can see from the previous code we didn't need to edit the "lamp"
object, but the latter is still able to modify its status (graphical and
not) if the user presses the "button" object. Obviously this tutorial is
extremely easy, but it is possible to create complex links among different
objects, and the result will be a rich and complex game.
```

1.24 DOOPSI-GS - Coder General Description

```
General description.

First of all, you have to select the action you are going to code:  to do
this press one of the nine "Possible Actions" button located at the upper
left corner of the window; the name of the current action will appear in
the field "Action" at the bottom of the window.

To insert any instruction you have to do the following steps:  click on the
name of the needed instruction contained in the lister located at the right
of the "Possible Actions"; then the Editor asks you to fill in any argument
the instruction may have.  After you have satisfied all the requests you
```

can see the line of code appearing in the code area (the biggest rectangle with a line highlighted in it). The bar in the code area is a sort of cursor that highlights the current line. The instruction you are currently editing will be inserted before the highlighted one. You can move the cursor by directly clicking on the chosen line or using the U (Up), D (Down), T (Top) and B (Bottom) gadgets. To kill a line simply click on the "Kill" button.

Gadget reference.

In the upper-left zone there are the nine gadgets to select one of the nine actions to code. To the right of this zone is the lister with all the instructions of the Doopsi language. Other gadgets are:

T	Moves to the top of the code.
B	Moves to the bottom of the code.
U	Moves one line up. (To move on a line you can also click directly on that line.)
D	Moves one line down. (To move on a line you can also click directly on that line.)
Kill	Discards the line of code currently highlighted.

1.25 DOOPSI-GS - Anim Workshop

```
*****************
The AnimWorkshop.
*****************
```

This editor allows you to build animations putting together some shapes in a sequence. The result is that you change an Object in a more complex thing called an "AnimObject", that is to say an object containing an animation. Such an animation will be played when the AnimObject is blitted on the scene.

1.26 DOOPSI-GS - Anim Workshop Tutorial 1

- Create a new object with "Pendulum" as internal name.

- Enter AnimWorkshop.

- Select the "Load Frames" gadget and, using the file requester, select the

shape named "pendulum.000". AnimWorkshop will then load the three shapes called "pendulum.000", "pendulum.001" and "pendulum.002".

- Select frame number 0 and click on the "Add Frame" gadget; select frame number 1 and click on the "Add Frame" gadget; repeat again for the third and last frame.

- Press the "Ping-Pong" gadget to create a Ping-Pong animation (in our case only a frame will be added)

- Now enter 3 in the "Frame Rate" string gadget.

- Press "Play" to view the animation and the right mouse button to stop it.

- Press the "Ok" gadget to exit AnimWorkshop.

1.27 DOOPSI-GS - Anim Workshop Tutorial 2

Here's a more difficul tutorial on "how to build the Man":

- First create a new object using the Object Attributes Editor (I suppose you already know how to do this) and call it (internal name) "ManRight". Now you are ready to enter AnimWorkshop.

- Select the "Load Frames" gadget and, using the file requester, select the shape called "man_walk_right.000": AnimWorkshop will load this frame and all subsequent frames with the same name and increasing number; such frames will be displayed on a new screen. You can cycle through them using the "Prev" and "Next" gadgets in the upper part of the editor window.

- Select frame number 1 (frame 0 is the rest frame and we will consider it later) and click on the "Add Frame" gadget: the number 1 will appear in the Sequence window. Then move to frame number 2 and click again on Add Frame; repeat these actions until the last frame is added.

- Now you can enter the frame rate in the "Frame Rate" string gadget: write 12 and press Return.

- Well, your animation is ok: press on the "Play" gadget to look at it.

- Stop the animation by pressing the right mouse button. Then, exit AnimWorkshop by pressing either the "Ok" or the "Cancel" button depending on whether you are satisfied of your animation or not.

- What about frame number 0? It cannot be included in the animation because it is the rest frame, but the Player has to know which frame to use when the Man is still: so we link it to our Man object and point to it by the "Default Shape" field in the Attributes editor. (Obviously this field has no meaning for a general animation.)

- A last remark: if you enter now the Attributes editor you will see all the shapes linked to our AnimObject with AnimWorkshop. This means that you can link your frames also using the Attributes editor if you like.

1.28 DOOPSI-GS - Anim Workshop General Description

General description.

At this stage I suppose you already know how to create a new object. If you need to convert this object into an AnimObject then enter AnimWorkshop. If the user already linked some shapes to this object using the Object-Attributes editor then these shapes will be loaded and displayed immediately; if one or more shapes couldn't be found an error is displayed and the missing shapes will be left blank. Alternatively, you can load your shapes using AnimWorkshop itself: but remember, in this case the shapes must consitute a sequence on the disk, that is to say they all must share the same name followed by a dot and a three (three, not four or two!) digit number, e.g. "foo.000". Thus, when the user selects the Load Frames gadget AnimWorkshop will open a file requester: here the user has to choose the first shape of the sequence (it need not be the one numbered "foo.000"; you can start from shape "foo.154" if you like to), and then the editor will proceed loading all the shapes it finds with the same name and with increasing index number (e.g. "foo.154", "foo.155", and so on) until it reaches the last one or it has no room to put more shapes in the current object (the number of shapes an object can have is of course limited).

After you have loaded the shapes you are ready to build the sequence of frames. First of all, there are two gadgets in the upper part of the window, named "Prev" and "Next", to cycle through the frames, and between them is printed the sequential number of the displayed frame: you can enter it manually. So you have to operate as follows: select the frame to be included in the sequence and then press then "Add Frame" gadget. Easy, isn't it? The frame will be inserted right after the current position, and you always know which is the current position because it is hilighted: you can also move this sort of cursor using the "Prev" and "Next" gadgets at the bottom of the window or using the cursor keys. By the way, if you need to insert a frame just before the first one you move the cursor on the first frame and then once more on the left: the cursor disappears out of the Sequence window and you are now ready to click on the "Add Frame" gadget. Well, I can tell you more: if you double click on a frame number in the Sequence window then such a frame will be displayed. To remove a frame is even easier: move on the frame position in the Sequence and then click the "Remove Frame" gadget.

Now we come to the description of the remaining gadgets. An effortless way to create ping-pong animations is to build the "ping" part as described previously and then click on the "Make Ping-Pong" gadget: AnimWorkshop will add to your sequence the "ping" part flipped (without the first and the last frame) to create the "pong" part. Then, if you are not satisfied with your Sequence you can kill it with only a mouse click on the "Clear Sequence" gadget. You can tell AnimWorkshop the frame rate of your animation just by writing it in the (guess it!) "Frame Rate" gadget. Last but not least, there come the "Play" gadget. I know you know what the function of this is, but let me tell you another thing: Doopsi will loop-play all animations, that is to say the Sequence is scanned from left to right until the end, and then again from the beginning until you stop it.

Gadget reference:

```
Load Frames    Loads a sequence of frames. A sequence is composed of files
               with the same name but with increasing sequential number
               (e.g.: foo.000, foo.001, ...). You select the first frame of
               the list.

Prev           Moves to the previous frame in the list.

Next           Moves to the next frame in the list.

Add Frame      Adds the currently displayed frame to the sequence.

Rem Frame      Removes the selected frame from the sequence. You select the
               frame by moving the cursor on it.

Make Ping-Pong Creates a ping-pong animation by modifing your sequence of
               frames.

Frame Rate     Changes the animation frame rate.

Clear Sequence Kills the sequence of frames.

Play           Plays the animation.

Prev           Goes to previous frame in the sequence (you can also use the
               cursor keys).

Next           Goes to next frame in the sequence.
```

1.29 DOOPSI-GS - Dialog Editor

```
**************
Dialog Editor.
**************

Introduction.

In every good adventure, dialogs between characters have to be present.
Better the dialogs are, brighter the conversations, more funny is the
adventure.  To create dialogs is not an easy task:  you have to consider a
lot of elements and foresee a different answer for every possible sentence.
That's why we strongly suggest to plan your dialogs and to sketch on a
sheet of paper real flow charts where "to see" the dialogs' structure.

Dialogs, in DOOPSI, can be structured in a handy and powerful way and it is
possible to write dialogs calling other dialogs or jumping from one part to
the other to get back later.

Dialog Editor is a easy and functional tool that will allow you to create
complex dialogs in a minute:  be careful, anyway, because, as we have
already said before, if you don't plan your dialog you can loose, just to
say, the dialog's "thread".


What is a dialog:
```

In Doopsi, a Dialog is composed by a sequence of pages. As happens for every other Doopsi element, even Dialog's pages have their "internal name" that identify them.

A page is made by an "answer", which is the sentence that the speaker will say to the Man and one or more sentences the user can select as "answer" to the "answer"... hmmm, said in this way it may seem to be confused; let's try to explain it better, with an example: let's start with the tutorial.

Tutorial 1

Dialog Editor General Description

1.30 DOOPSI-GS - Dialog Editor Tutorial 1

Tutorial 1 - A Brief Dialog

To enter the Dialog Editor, simply press the "Dialog Editor" button in the Main Window.

- Press the "New" button to create a new dialog page.
  A string request will pop up asking for the page's internal name.
  We want this page to be the starting point, so write "Start_Dialog".
  Now you can start to write dialog phrases.

- Press the "Add" button (bottom left) to add a new phrase. The "New Phrase" string will appear in the string gadget called "Phrase:".
  Delete "New Phrase" and insert: "Hello! How are you?"

- Repeat step 2, inserting these messages: "How are you!" and "I should go now".

- Now we have to create a segond page, press "New" and place "How_Are_You" as internal name, so we'll remember this page is the answer to the "Hello, how are you?" question.

- In the "Answer:" gadget (top of the window), write "Well!",
  this is the answer that our friend will give us.
  To write something inside this field is not stricly required, but it is always better that the other says something... otherwise our Dialog could become a "monologue".
  The only place in which we can omit an answer is in the first dialog's page, when we begin to talk to someone.

- Now repeat setp 2, and write "Now I go, bye".

- Now we have to "link" these two pages together.
  To do so, return to the first page. Press the "Select" gadget: a Lister containing dialog pages' names will appear. In our case there'll be two names: "Start_Dialog" and "How_Are_You", select "Start_Dialog".
  Dialog Editor's window will show the first page again.

- Now press on the phrase, "Hello, how are you?" that will appear in

```
     the string gadget (if you wish, you should also edit the phrase)
     and then select the "Jump" button, the same Lister we have seen
     previously will appear again, this time select "How_Are_You"
     name: the page's name will appear in the string gadget "Jump:".
     That's all! Now, when, during the game, user will select phrase
     "Hello! How are you?" the other will say "Well!".
     That's easy, isn't it?

   - Just for fun, we'll link also the "How Are You!" phrase to
     "How_Are_You" page.  Press "How are you!" and then on "Jump" and
     select "How_Are_You".

NOTE: a phrase without a jump means "END OF DIALOG". When the user will
      select one of these phrases, the Dialog ends.
      In our example, we have two phrases of this kind: the first is
      "I should go now" and the other is "Now I go, bye".

NOTE2: as a convention, the exiting phrase is the last in the list.
       Where possible, please keep this convention.
```

1.31 DOOPSI-GS - Dialog Editor General Description

```
General description.

The Dialog Editor is used to create dialogs between the Man and another
Object on the scene.  As we said when introducing this editor, a complete
dialog is made of a number of pages linked together; any one af the pages
can be the first page:  the name of the first page will be passed as
argument to the ShowDialog instruction in the Doopsi Coder.  All the other
pages of the dialog follows because they are linked together by means of
the "Jump" field.

Now let's describe the contents of a page.  In every page there is an
"Answer" field that contains the sentence the Object says when it's its
turn to speak:  this is the first field the Player looks for when
displaying a dialog.  This implies that if the first page has the "Answer"
field not empty, then the conversation is started by the Object.  When the
Object has finished to speak (that is to say, when the user, during the
game, clicked on the mouse button to clear the text that appears on the
scene), then all the sentences contained in the big lister under the
"Answer" field will be displayed in the space reserved for the console.
The Player waits for the user to select one of the sentences and then
displays it on the scene (waiting again for the user to clear the text).
Then the Player looks at the contents of the "Jump" field:  if it is not
empty, the Player searches for a page with a name equal to the string in
the "Jump" field.  Eventually, if this page has been found the process
restarts:  the "Answer" is displayed, then the choices under it, and so on.
A dialog meets its end when the "Jump" field of the selected sentence is
empty or the linked page has not been found.

To build a dialog you first have to create and edit some pages.  The "New"
button on the right creates a new page and asks you the name to give to
that page:  this name is the identifier that Doopsi uses to address the
page, so you should have care not to duplicate such a name.  To edit the
answer, simply enter it in the "Answer" string gadget.  To add a new
```

sentence you have to press the "Add" button: the "New Phrase" text appears in the lister and in the string gadget under it. Now you can edit this sentence by entering it in the "Phrase" string gadget: the new text will replace the old one in the lister. In the "Jump" field there goes the name of the page to jump to when the current sentence is selected during the game: you can write such a name by entering it in the string gadget or you can press the "Jump" button; the latter opens a lister with all the names of the pages created up to now: click on the desired name and then on the "Ok" gadget.

Now we describe the remaining gadgets in the bottom row. The "Memorize" button stores the contents of the "Phrase" field in the lister (it's the same as pressing the Return key in the "Phrase" field). The "Select" gadget allows you to move through pages: when you press it a lister appears with the names of the pages created so far, and you can select the desired one. The "Restore" gadget undoes the latest modification of the "Phrase" field. The "Del" button discards the currently selected sentence.

The gadgets on the right column have the following meanings. The "New" gadget creates a new page. The "Save" button saves all the pages created so far, and the "Load" gadget loads a set of pages saved with the previous button. The "Page Name" gadget allows you to change the name of the current page. The "Kill Page" discards only the current page and the "Kill All" discards all pages created.

Gadget reference:

Let's start with the right gadgets:

New Creates a new page

Load Loads a set of pages.

Save Saves a set of pages.

Page Name Modify the page internal name.

Kill Page Erase current page.

Kill All Erase all created pages.

And here there come the bottom gadgets:

Add Add a new phrase.

Memorize Memorize the current phrase.

Select Select a dialog's page.

Restore Restore old phrase in the current line.

Del Erase current page.

1.32 DOOPSI-GS - Preferences

```
************
Preferences.
************

Doopsi is a complex and flexible program and some of its features can be
adapted to the programmer's needs.  In Doopsi your preferred settings are
entered via the Preferences window, divided in four parts.

Path Selection.

In this section it is possible to choose dirs containing all files of a
specific kind.  Creating complex adventures, it is better to divide
graphics, sounds and all the rest in different dirs.  With these paths,
every time a file requester will appear (for example to load a background)
the directory will be set to the preferences' one.  In case you do not want
any default dir, just place an empty string in the string requester.

The second group of default paths is related to the Player:  the Player
needs them to find the files that are vital to start the game, that is to
say the file containing the console (default:  "Doopsi:Data/panel.iff"),
the file "player.dat" (default:  "Doopsi:Data/player.dat") and (optional)
the catalog containing a language different from english (default:
"Doopsi:Data/player.catalog").  In these string gadget you have to insert
the file name together with the complete path.  This allows, for example,
to use a different console for each of your adventures.

The last string gadget, named "Player", contains the complete path together
with the name of the program "DoopsiPlayer":  this one is needed by the
Editor to locate the Player when you click on the "Test Game" button in the
Main Window.

Next to the string gadget there is a gadget called "R" (research), which
will open a path requester you can use to search the directory you want to
set as default.

Palette Editing.

In this section it is possible to load, edit and save Doopsi's colors.  By
pressing "Load" you'll be able to load up the first four colors of any IFF
picture, with "Save" you can save the palette and with "Edit" it will be
opened a standard Palette Requester which allows you to directly edit
colors.

Setting Default Names.

This is the most important secion of the preferences window.  As you should
already know, Doopsi uses for the Man some default names, used by the
Player to find the right animations when needed.  The explanation on how
this works are contained in the section "Some notes on how the Player finds
the Man" located right after the Main Window section of this manual.  Here
are the names and the meanings of the fields:


NAME         DEF. VALUE   DESCRIPTION
```

Root	Man	It is the root name, from which the Man's animations names are built.
Left Right Up Down	L R U D	These four values set the animations of the Man in all the four directions. For example ManU is "Man Up", that is: "animation of the man walking forward".
Talk	Talk	This is a kind of "second root" and it is the suffix that is added DIRECTLY to the real Root to define the animations of the Man while it speaks. To the Root+Talk string are then appended the usual suffixes Left, Right, Up or Down. For example: the Man speaking to the left results in: ManTalkL.

These names, being just for "internal" use of the Player, should remain "rigid" and not modifiable, but we preferred to allow modification, to let everything be more readable. Infact, you can change these names, giving them a more exaustive meaning. Here there is an example.

```
Root = Man
Talk = _talking
L    = _left
R    = _right
```

the old ManTalkL now becomes: "Man_talking_Left", more elegant. Be careful, anyway: in Doopsi, internal names have a maximum length of 30 characters.

Starting Scene.

With this last part of the Preferences' window, you can define the name of the first scene that will be shown at the beginning of the adventure.

"USE/LOAD/SAVE/CANCEL" Gadgets

These gadgets allows you to do the following things: "USE" to use the current configuration, "LOAD" and "SAVE" to load a previous saved configuration and to save the current one, and "CANCEL" to abort settings.

Note: preferences are ALWAYS saved along with the complete Doopsi file, to allow the handling of different adventures at the same time with different settings.

TOOLTYPES:

Starting from V1.10, DOOPSI support some tooltypes. At the moment they are:

SCREENMODE=$xxxxx	To select the ScreenMode you want. (Ex. $8000 = Hires NoLace)
PREFS=ON|OFF	By setting this flag to ON or OFF you can enable or disable the setting of colours of the interface.

1.33 DOOPSI-GS - Appendix A

```
*********************************
Appendix A:  Doopsi instructions.
*********************************

In the following list, objects and scenes are identified by their internal
names, so Object represents the Object internal name, and so on.


AddStatus Object, Value

Adds Value to the status field of the selected Object.  The status field
can contain a value ranging from -32767 to +32768.


ChangeDescription Object, Text

Changes the name field of the selected Object to Text.  This name is the
one that appears on the console display line when the object is selected on
the scene.


ChangeScene Scene

If Scene has been found, this instruction exits the current scene and
displays the new scene.


ChangeShape Object, Shape

Changes the current shape of the selected Object.  If the object is on
scene the new shape will be blitted and it will replace the old one.  If
the object is in the inventory this command affects the inventory shape
(and reblits it).  If the object is not displayed then this instruction
modifies only the current shape field of the selected object.  This
instruction doesn't apply to AnimObjects or to Empty Zones (for obvious
reasons) when they are displayed on the scene, but their inventory shapes
can still be changed.  In ChangeShape the current Object shape is erased
because the new shape is reblitted immediately over it:  the drawback is
that the two shapes must have the same dimensions.


ChangeShapeBG Object, Shape

This is similar to ChangeShape, but before drawing the new shape the Player
erases the Object current shape reblitting the background.  This removes
the limit (present in ChangeShape) that the new image must have dimensions
greater or equal to the old one.


ChangeStatus Object, Value
```

Puts Value in the status field of the selected Object. The status may be 0 or 1 to represent a door closed or open respectively, and so on... If the Object is a coin this field can be used to store the number of coins the Man possesses. The value the status field of an object can contain ranges from -32767 to 32768.

Close Object

See the Open instruction.

ConsoleHide

Switches off the console.

Console Show

Switches on the console.

Drop Object

See the Open instruction.

Else

Else is used in an If group: if the condition is False then the player will execute all instructions following the Else statement until the next EndIf (or, if not present, the end of the program).

EndIf

This instruction ends an If group. You can have up to 10 If groups nested.

EraseInventory Object

Removes Object from the inventory list and deletes its inventory shape from the console.

EraseObject Object

Removes Object from the scene. This instruction doesn’t apply to AnimObjects: in this case the user has to call StopAnim.

GetInventory Object

Adds Object to the inventory list and displays its current inventory shape on the console. This instruction fails if the inventory list is full, because such a list can contain only a limited number of objects.

IfOnScene Object

Returns True if Object is blitted on the scene.

IfSceneIs Scene

Returns True if Man is currently in scene Scene.

IfStatus Object, Status

Returns True if the status field of the selected object is equal to Status.

IfUsedWith Object

Returns True if the current object is used with Object. This means that you first selected the Use button and clicked once on Object, then, after the text "Use Object with..." appeared on the console, you clicked twice on the current object: this activates the Use action program of the current object where this instruction is intended to be. The text "with..." appears and the Player waits for the user to select another object only if UseWith is the first instruction of the Use action program of Object.

InvToScene Object

Picks up the selected Object from the inventory and puts it on the scene at the internal coordinates of Object: such coordinates are initialized when the Object is positioned on the scene using the Editor. If Object is an AnimObject then the animation will be automatically played.

InvToSpot Object, Spot

Similar to InvToScene, but the Object will be drawn at the Spot coordinates.

Look Object

See the Open instruction.

ManDirection Extension

Forces the Man to face in the selected direction: Extension is one of the four extensions representing the four directions the user can modify using the Preferences editor.

ManTalkColour Value

With this instruction the user selects the colour register to be used for

the text in all subsequent ShowText or ShowDialog instructions when the Man is talking.

MoveMan Spot

This instruction allows the user to force the Man walk to a desired Spot on the scene.

MoveObject Object, Spot

Moves Object from its current position on the scene to the location specified by Spot position. During the movement an AnimObject is substituted to the Object current shape if it has been previously specified with the SetMoveAnim instruction.

MoveObjectOnPath Object, Spot

This is similar to MoveObject with the difference that the selected Object will move following the current scene path and will stop at the path node nearest to the Spot specified. Another slight difference is that the hot-spot of the object (or, if present, of the animation that replaces it) is considered to be at the lower left-hand corner.

MoveObjectToObject Object1, Object2

This is similar to MoveObject, and it moves Object1 to the location of Object2.

MoveObjToObjOnPath Object1, Object2

This is similar to MoveObjectOnPath, and it moves Object1 to the location of Object2.

MusicPause

Momentarily stops the music. To restart it use this command again.

MusicPlay File

Starts playing the contents of File. File must be a standard tracker module.

MusicStop

Stops all musics.

ObjTalkColour Value

This is similar to ManTalkColour, but allows you to select the colour of the text related to the Object the Man is talking to during a ShowDialog instruction.

```
Open  Object
Close Object
Look  Object
Take  Object
Drop  Object
Push  Object
Pull  Object
Talk  Object
Use   Object
```

These instructions execute the Doopsi code contained in the corresponding action of Object. They are treated just like subprograms, and you can nest at most 10 of such subprograms.

Pull Object

See the Open instruction.

Push Object

See the Open instruction.

PutObject Object, Spot

Blits Object at the Spot location. Object may be also an EmptyZone.

SceneToInv Object

Picks up the selected Object from the scene and puts it in the inventory. If Object is an AnimObject the animation will be automatically stopped.

SetManName Object

This instruction is used to change the Man's animations: the root name (that is to say, after having removed any extension) of Object is used to build the Man's names until SetManName is called again or it is issued by a path node.

SetMoveAnim AnimObject

The selected AnimObject will be used as animation for the following MoveObject or similar instruction (MoveObjectOnPath, MoveObjectToObject, MoveObjToObjOnPath). If the AnimObject name ends with one of the four extensions (indicating the four directions) then the Player is allowed to choose the needed direction before moving the object: if such a direction is not supplied, then the object current shape will be used, thus the

animation selection will be totally ignored.

SetTalkAnim AnimObject

This instruction tells the Player that, when the next ShowDialog will be executed, it must use AnimObject to show that Object is talking. If the user ends the AnimObject internal name with one of the the four extensions indicating a direction (such extensions can be changed using the Preferences editor), then the Player removes this extension from the internal name and chooses the appropriate direction (depending on the Object position relative to the Man's position). Then it appends the correct extension indicating the chosen direction to the AnimObject internal name and searches for the resulting AnimObject. If it is found, then the animation will be played when Object is speaking.

ShowAnim AnimObject

This instruction plays the animation contained in AnimObject. The animation will be displayed at the internal coordinates of AnimObject: such coordinates are initialized when the AnimObject is positioned on the scene using the Editor (and are kept in memory even if the AnimObject is subsequently removed from the scene). The number of animations the Player can handle at the same time is limited.

ShowAnimAtSpot AnimObject, Spot

Plays the animation contained in AnimObject at the Spot coordinates.

ShowDialog Dialog

This starts a conversation between the Man and the currently selected object. Dialog is the first page of the conversation. If the animation showing the Man talking is found then it is played; you can choose an animation showing Object talking with SetTalkAnim. The Man text colour and the object text colour are selected using calls to ManTalkColour and ObjTalkColour respectively.

ShowPicture File

Momentarily hides the scene and the console to display the contents of File. File must be an Iff ILBM picture. If the picture is bigger than the screen area you can scroll it by moving the mouse to the edges of the screen. To return to the previous state press the left mouse button.

ShowText Text

Displays Text. The position and line breaks will be determined by the Man's position in the scene and by the width of the scene itself. The Man's direction determines which one of four animations will be played, if it is found, to make the Man speak; if the animation is not present the Player won't complain and it only leaves the Man still. The colour of the

text is selected using the ManTalkColour instruction. The user has to
click once on the left mouse button to quit the text.

SoundPlay File

Plays the contents of File. File must be a standard Iff 8SVX sample.

StopAnim AnimObject

Stops the animation contained in AnimObject and removes it from the scene.

Take Object

See the Open instruction.

Talk Object

See the Open instruction.

Use Object

See the Open instruction.

UseWith

This instruction must be the first one in the Use action program of an
object: it tells the Player that the current object is to be Used in
conjunction with another one; so the Player displays the text "Use Object
with ..." and waits for the user to select another object. See also
IfUsedWith.

1.34 DOOPSI-GS - Console

****
Appendix B: The console.
****

The outline of the console included in the Doopsi package shows the
different zones sensed by the Player: there are the nine action buttons on
the left, two vertical arrows to scroll the inventory and, between them, a
small button used to enter the disk requester; the rightmost part is
occupied by the inventory and, last but not least, the topmost rectangle is
the display line (that is to say the line where all the messages will
appear). By the way: each of the inventory squares measures 32 * 32
pixels, so the user has to take into account this dimensions when he's
designing inventory shapes for his objects. Below the console, on the same
screen, is outlined the disk requester: the widest area on the left is
where the file list will appear and under it there is the string gadget
where the user has to insert the search path; then there are two vertical

arrows to scroll the file list and five buttons.

The console+disk screen must be a lowres non interlaced screen and the user can paint everything he likes on it, but he must remember to observe the dimensions of the sensing zones. Remember also that the inventory squares, the display line and the file lister will be filled by the Player with the background color 0. Another hint: all inventory shapes must have the same palette, the console one, because they will appear on the console and this is fixed throughout the game; on the contrary, the other shapes can have different palettes because you are allowed to change colours passing from one scene to another.

The disk requester.

It's just like all the other requesters you are used to: the search path is entered in the string gadget at the bottom of the screen and the file names are listed in the wide area above it. To load a file the user has to click once on its name and then click on the Load gadget on the right of the screen. To save a file the user has to click once on a name and then click on the Save gadget: a cursor will appear to allow the user to change the name of the game; when it's all right he has to press Return to save it. To save a new game just click on on the Save gadget as before, without selecting any name.

The names in the lister and the search path are all limited to a lenght of 25 characters (I think it's enough for the majority of the applications).

The five buttons on the right are named, in order: Load, Save, Play, Workbench and Quit. The Load and Save buttons have already been described. The Play button returns to the console screen and continues the game. The Workbench button activates multitasking: the Player disappears and the user can return to his game by selecting the Doopsi menu item in the Tools menu of the Workbench. The Quit button..., well I leave you to imagine the purpose of that.

1.35 DOOPSI-GS - DOOPSI Insights

```
*****************************
Appendix C:  Doopsi insights.
*****************************
```

In the following I will use the word Doopsi when I'm describing how routines common to the Editor and the Player work.

Doopsi object structure.

To correctly understand how Doopsi acts and to get the best from it you have to know at least the basics about how Doopsi objects are structured.

Let's start with a Scene:

```
   internal name
```

background file name

list of linked Objects (internal names)

The internal name is a private name the user gives to the Scene and that is used by Doopsi to recognize the Scene. The background name is the complete (path included) name of the Iff file containing the picture to be displayed. Then it comes a list of the internal names of the Objects that have to be blitted on the Scene.

Next we come to an Object. There are three types of object, but they all share the same structure:

a) Objects tout-court are simply objects with the usual meaning and their shapes will be blitted on the scene.

b) AnimObjects contain an animation that will be played on the scene.

c) Empty Zones are made of a rectangle (that won't be displayed) without shapes in it.

Here is the general Object structure:

name

internal name

X position

Y position

Width

Height

status

list of shapes

nine action programs

The name field is what appears on the Player display line when the pointer is on the object. The internal name is a private (possibly unique) name the user gives to the object and it is used by Doopsi to recognize this object: if the user duplicates an internal name the Editor won't complain but you may obtain unpredictable results (usually Doopsi stops at the first object with the requested internal name). X and Y positions specify where to put the object on the screen. Width and Height are the dimension of the object. The status field contains a general purpose integer value: it can be used to store the On-Off status of a light, or the Opened-Closed status of a door, or the number of coins the Man possesses, and so on... Then there is a list of all the shapes (filename with full path of the Iff file containing the image) the user has linked to this Object. Eventually come nine programs (containing some Doopsi code), one for each action button on the Player console.

An AnimObject differs from the previous description in that it has a string

of integers specifing the sequence in which shapes will be played. The Width and Height fields will contain the maximum width and height respectively of all the shapes selected for the animation.

An Empty Zone differs in that it won't be blitted on the screen, but its position will still be sensed by Doopsi. The list of shapes may not be empty, because the user may want, for example, to put this Object in the inventory (and so he needs an inventory shape), and so on.

1.36 DOOPSI-GS - The Player

****
The Player.
****

The startup sequence.

When the Player starts it searches, before anything else, for the Doopsi main file and, from this time on, all the actions are determined by the data it finds in such a file. From the Doopsi file it gets the path where there are the console (the bottom half of the screen where the buttons and the inventory stay), contained in the file "panel.iff", and the file "player.dat". If it doesn't find anyone of the above mentioned pieces then it complains and quits.

Supposing it has successfully accomplished the previous steps, the Player now searches the Doopsi file (the main file with all the game structure in it) to find the Man's animations: there must be four animations, one for each direction, with the default names the user has chosen. Because this data is vital (how could you play without seeing where you are?) the Player stops and quits if it hasn't found all it was searching for. This means that it must, first of all, find the four Objects containing such four animations, and then it must find all the shapes needed. By the way, if the user supplies also the animations to make the Man speak they will be loaded: in this case it's not necessary to supply four animations if the user need not to, because such animations will be used only if they are found. But remember, if the user tells the Player to use an animation then this animation must be complete, that is to say the Player must be able to find all the shapes needed. This is a general rule.

And now we come to the next step: up to now the Player has been able to display the console, but the top half of the screen is still awfully black. So it looks in the Doopsi file for a scene you selected as the first scene using the Preferences Editor: if the user has forgotten to include at least one scene with such a name then the Player has nothing else to do, because it doesn't know from which scene to begin the game, and quits ("again!" you could say, but I don't think this behaviour is due to touchiness). Obviously it quits also if the scene has been found but there is no Iff file containing the background.

From now on the Player will only complain without stopping if it doesn't find something. Common errors are "Object not found", if the object the user requested is not in the list, or "File not found" if it doesn't find the file containing a shape or a background, etc... In this cases nothing happens and the game continues undisturbed. Error messages are displayed

on the console, in the same line where appears the "Go To" text; to quit the message and continue the game the user has to click the right mouse button.

If a scene has been successfully loaded then its background will be displayed and all the objects linked to it will be blitted on the screen; AnimObjects are automatically recognized and the animations contained in them are played. Supposing you already know (from the Editor manual) what a Doopsi path is, I tell you that the Player first searches for a node in the path of the current scene with the same name of the scene where it comes from, then, if this fails, it searches for a node named as the current scene; finally it blits the Man (always facing the user) next to the node found.

There is also something a little bit technical you ought to know about nodes: these allow you to supply a custom animation for the Man and to decide where to change scene; besides, the scene path might be influenced by conditions set in its nodes. The Player reads the custom animation name, the conditions and the changes of scene from a node when the Man has its feet placed on that node: the custom animation will be displayed only when the Man walks from the current node to the next one (thus, if the Man stops on the current node you cannot see such an animation and you have to supply a rest frame to the previous one), but the conditions and the changes of scene are read and executed immediately when the Man arrives on the current node. To sum up: custom animations in a node are executed when the Man departs from that node, but conditions and changes of scene are executed when the Man arrives at that node.

At last the game starts and you become the chief character.

Errors.

Here is a brief description of the errors. Error messages appear in the display line and have to be quitted by pressing the right mouse button. If the error is one of the type "... not found ..." the Player prints also the name of the object (or the file, etc.) that it doesn't find.

Object not found: the object is not in the main list.
Scene not found: the scene you asked for is not in the main list.
File not found: I think this needs no explanation.
Spot not found: the user referenced an inexistent spot.
Inventory full: the inventory list is big but limited, so if the user puts too many objects in it, it could overflow.
Inventory empty: the user tried to act on the inventory even if this is empty.
No room for anim: the number of AnimObjects the Player can handle at the same time is limited.
Dialog not found: the user asked for a dialog page that doesn't exist.
Directory not found: this message is from the file lister: the user specified a path that doesn't exist.
No room for another object: the number of objects that a scene can contain is limited.
Too many bitplanes: the user tried to put on the scene an object with a depth greater than that of the scene.
EndIf without If: in a Doopsi program there is an EndIf instruction without its corresponding If command.

```
Not a valid Iff file:   you have to supply a standard Iff file when
                        explicitly asked, e.g.: in the ShowPicture or
                        SoundPlay instructions.
Stack full:          only up to 10 nested subprograms are allowed: if you
                     put in one more the stack will overfull.
```

1.37 DOOPSI-GS - Hardcoded Limits

```
*****************
Hardcoded limits.
*****************

Here are collected the current limitations of Doopsi:  the "Shareware" and
"Special" fields show the limitations for the two demo versions.


Editor:

Type                    Max number
~~~~                    ~~~~~~~~~~
scenes                  200         (Shareware: 3;  Special: 5)

objects                 500         (Shareware: 30; Special: 40)

dialog pages            100         (Shareware: 5;  Special: 8)

lines per dialog page   12          (Shareware: 3;  Special: 4)

shapes per object       15

objects on scene        20
 at the same time

internal name lenght    30



Player: the same as the Editor plus

Type                    Max number
~~~~                    ~~~~~~~~~~
animobjects on scene    10
 at the same time

objects in inventory    500         (Shareware: 30; Special: 40)

nested subprograms      10

nested If groups        10
```

1.38 DOOPSI-GS - Shareware

REGISTRATION FORM.

Use the followin part of this doc to register to DOOPSI-GS.

At the moment, you can send us money in these ways:

Money and international checks (and stop).

We'll send your registered copy of DOOPSI when we'll be sure of your payment. If you paid by check, you can speed things up by sending us a copy of your payment doc, with the form below compiled in every part.

Along with the registered version of DOOPSI-GS you'll receive the complete documentation in all the electronic formats you selected.

-------- CUT HERE ------- CUT HERE -------- CUT HERE ------------------

I'd like to register to your program.
I'll send you the money amount of US$ 35, (UKP £15, Lit. 35000, DM 35)

Check if needed:

 __
|__| plus the postal charges of US$ 3 (UKP £2, Lit. 3000, DM 3) because I want the disk via Snail-Mail.

 __
|__| plus US$10 (UKP £5, Lit. 10000, DM 10) because I want the next 3 updates by Snail-Mail (NOTE: if you can receive them via e-mail, they are free).

 __
|__| I want to receive info about how much does the printed manual cost.

Prog Version : Doopsi-GS V1.00

Name   :

E-Mail   :

Address   :

Telephone   :

Computer   :

I'd like to have the free electronic documentation in the following formats (check):

 __
|__| PostScript.
 __

```
|__| LaTeX.


-------- CUT HERE ------- CUT HERE -------- CUT HERE ------------------


Send to:

      Fabio Rotondo                    Andrea Galimberti
      c.so Vercelli, 9                 via Villoresi, 87
      28100 Novara                     20029 Turbigo (Mi)
      Italy                            Italy
      E-Mail: fsoft@intercom.it
```

1.39 DOOPSI GS - Player Royalty

```
ROYALTY

DOOPSI Player may be freely distributed along with your adventures
as long as you do not make any profit by it.

If you include it in a Shareware package, you have to send us a registered
version of it.

If you include in a commercial program, you have to pay us the nominal fee of
US$50 and ask us a written permission before shipping the player.

For more info, contact:

    Fabio Rotondo      e-mail: fsoft@intercom.it
```

1.40 DOOPSI-GS - Copyrights

```
DOOPSI-GS and DOOPSI-Player are (C)Copyright by Brigthing Brain Brothers

ReqTools is Copyright (c) Nico François and Magnus Holmgren.
```