Arachnid code collection

Rudolf Cardinal. 1 July 1999.

Introduction

This is a collection of all my Arachnid code. Arachnid, from Cenes Cognition né Paul Fray Ltd, is a extension of BBC BASIC V designed for real-time control applications in the loose sense of the word; it's not a language itself. The two most important features of this collection, as I see it, are

1. Fairly legible code. (Illegible code is bad code, even if it works. Legible code may not be good, but at least you can tell.)
2. **Modular libraries** to save effort.

Libraries are collections of functions that have been written to perform a specific, useful task and have been carefully tested. You can use them without wondering how they work (or have a look under the bonnet if you're so inclined; it's a good way to learn). More importantly, you can be sure they're not going to crash or do something weird, because they've been tested thoroughly and used a lot.

For your own sake, don't modify my libraries. Write your own for extra functions. Otherwise, you'd lose your changes if I altered a library function and gave you the new copy. But I'd appreciate copies of any code you write using this system, plus of course comments and bug reports (though I hope that won't happen). And only put functions in a library if they're useful for a range of programs; if they're specific to a single application they might as well live in the main program.

To **use a library,** place a reference at the start of your program like

```
LIBRARY "UI"
```

for the user interface library called UI. I keep all the libraries in a single place (a directory called ProgLibs) and use an operating system variable to find them, so my references look like this:

```
LIBRARY "<RudolfDir>.ProgLibs.UI"
```

This system is described below (under "Putting an application together").

References

- The Arachnid manual
- *BBC BASIC Reference Manual*, from Acorn. You don't get a proper guide to BASIC with the computers these days.

Disk formats

PCs will not read Arc disks. Arcs will read PC disks, *as long as they do not contain long (Windows 95/NT) filenames.*

Putting an application together

1. Create a directory to hold all your programs and data. Mine's called Rudolf (see Table 1).

2. Copy the **ProgLibs** directory in its entirety into your personal directory. This contains all the libraries.

3. Within your personal directory, create a subdirectory whose name begins with an exclamation mark (!). This tells the computer that it contains an application. An example is the !Evenden directory shown in Table 1.

4. Optionally, create a data directory for the same application (within your main directory). An example is Evenden-Dat in Table 1.

5. Create a **!Run** file within the application directory. An example is given in Table 2. It will probably work without modification.

6. Create a **Start** file in the same place. A sample file is shown in Table 3. You will have to modify the parts in bold to refer to your own personal directory, data directory and BASIC program name. You must not change the phrase “RudolfDir” itself, because the code searches for that. It is important that the ProgLibs directory is in the directory you name in the “*Set RudolfDir *somewhere*” command.

7. Store the program itself in the application directory. **Important:** I keep the programs in a text format so I can edit them on a PC, but they must be stored as a BASIC program to run. To convert between the two, you must use !Edit on the Archimedes. Load the text file into !Edit; click the middle mouse button; choose Misc → Set Type and type in *BASIC*. Press Enter. Now save the program where you want it to live. To convert BASIC into text, simply repeat the process but type *TEXT*.

Table 1: Directory structure on the Archimedes. Directories are underlined.

```
$                                 Root directory of the hard disk.
|
+-- Apps
+-- !Arachnid
+-- !2ndOrdr
+-- ...
+-- Rudolf                        Your personal directory, containing everything.
    |
    +-- !Evenden                  An Arachnid application, containing:
    |       !Run                      Generic !RUN file
    |       Start                     START file that calls the program
    |       Evenden                   The BASIC program itself.
    |
    +-- !TestCBox                 An Arachnid application.
    +-- EvendenDat                Data directory for the !Evenden program.
    +-- ProgLibs                  Contains all the libraries.
    |       Arachnid
    |       BoxConst
    |       DateTime
    |       ...
    |
    +-- ...
```

Table 2: Sample !RUN file

```
| !Run file for Arachnid code
DIR <Obey$Dir>
RMEnsure Arachnid 0 RMLoad :4.$.!Arachnid.ArachMod
RMReInit Arachnid
ECHO |V|C|U
EXEC Start
BACK
```

Table 3: Sample START file. Customize the parts in bold.

```
*BASIC
V.11:P.SPC(26):V.13,11,21
*DIR <Obey$Dir>
INSTALL ":4.$.!Arachnid.ArachLib"
PROCinit
P.CHR$6"Arachnid Issue 1.12 (C) 1991 Paul Fray Ltd"'
*SET RudolfDir :4.$.Rudolf
*DIR <RudolfDir>.EvendenDat
CHAIN"<Obey$Dir>.Evenden"
```

The modular libraries in detail

Library	Purpose	Functions and procedures intended for general use. **Note:** in all cases, there is further documentation in the source code itself.

`Arachnid`	Uses Arachnid to control switches and sensors. Functions to dispense pellets, flash lights/clickers, calculate the mass of a number of 45 mg pellets, etc. The flashing code is particularly useful.	`FNswitch_number(box%, line%, nboxes%)`
		`FNswitch_number_2(box%, line%, nboxes%)`
		`PROCsingle_pellet(line%)`
		`PROCfast_pellet(line%, numpellets%)`
		`PROCspaced_pellet(line%, numpellets%, timer%, gap%)`
		`PROCold_spaced_pellet(line%, numpellets%, timer%, gap%)`
		`PROCstart_flash_line(line%, timer%, on%, off%)`
		`PROCstop_flash_line(line%, timer%)`
		`PROCold_start_flash_line(line%, timer%, on%, off%)`
		`PROCflash_line_number(line%, timer%, on%, off%, count)`
		`PROCold_flash_line_number(line%, timer%, on%, off%, count)`
		`PROCflash_line_time(line%, timer1%, timer2%, on%, off%, duration%)`
		`PROCuserfunc_start_flash(on_func$, off_func$, timer%, on%, off%)`
		`PROCuserfunc_stop_flash(off_func$, timer%)`
		`PROCuserfunc_flash_time(on_func$, off_func$, timer1%, timer2%, on%, off%, duration%)`
		`FNswitch_on_line(line%, bogus%)`
		`FNswitch_off_line(line%, bogus%)`
		`FNfood_mass(pellets%)`
		`FNliquid_volume(dips%)`

`ASCII`	For sending ASCII output to a file.	`PROCprint_line(channel%, string$)`
		`PROCprint_string(channel%, string$)`
		`PROCprint_string_visibly(channel%, string$)`
		`PROCprint_line_visibly(channel%, string$)`

`BoxConst`	Defines the wiring of all the boxes we use and dimensions appropriate variables.	`PROCcombined_boxes`
		`PROCintermediate_boxes`
		`PROCsucrose_boxes`
		`PROCpellet_boxes`
		`PROCyiaping_boxes`
		`PROCfive_hole_boxes`
		`PROCfive_hole_boxes_yogi`

`DateTime`	Generates a date/time code.	`FNdate_time_code`
`DelayLib`	Specialist library to support delay-of-reinforcement tasks	*The main infrastructure for AdjDelay, Evenden, Richards and DTrain.*

`Filename`	Asks the user for a filename and ensures that it's a safe one to use.	`FNget_filename(prompt$)`
		`FNget_filename_default(prompt$,default$)`
		`FNget_filename_dangerous(prompt$)`

`JP`	Touchscreen code.	*Interfaces to IntaSolve touchscreens. See source code.*

`Random`	Random number handling, and other random mathematical things	`FNrandom_integer(min%,max%)`
		`FNmin(a%, b%)`
		`FNmax(a%, b%)`

`Spool`	Simple functions to control *SPOOL and printer output.	`PROCopen_spool(file$)`
		`PROCclose_spool`
		`PROCprinter_on`
		`PROCprinter_off`
		`PROCscreen_on`
		`PROCscreen_off`
		`PROCpause_spooling`
		`PROCresume_spooling`
		`PROCset_output_state(screen%, printer%, spool%)`

`UI`	User interface code, including routines to ask the user for parameters and check they are within an allowed range.	`FNget_num_param(prompt$, default, min, max)`
		`FNget_str_param(prompt$, default$)`
		`FNask_yes_no(prompt$)`
		`FNget_letter_param(prompt$, valid$, default$)`
		`FNget_letter_param_as_num(prompt$, valid$, default$)`
		`PROCprint_centered(s$)`
		`PROCdefine_colours`

`YOGILIB`	Touchscreen/five-choice code for Yogi.	*Box definitions, touchscreen definitions, screen coordinate handlers.*

The applications

For Rudolf

Program	Description
`2-Discap`	Two-stimulus discriminated approach task.
`2-DA-Om`	Two-stimulus discriminated approach task with an omission contingency.
`2-CRf`	Two-stimulus conditioned reinforcement (acquisition of new response)
`AdjDelay`	Adjusting-delay task where delay can vary after every choice trial. (Never used.)
`Bradshaw`	Original (RDR/ND/RNC) version of Mazur/Bradshaw's adjusting-delay task.
`CapTrain`	Train on an FR1 schedule for a capped number of reinforcers.
`CRf`	Conditioned reinforcement. CR lever causes delivery of an abbreviated form of the conditioned reinforcer from DiscAp; NCR lever has no programmed consequence. Also contains a timer to assist with intracerebral infusion.
`DiscAp`	Discriminated approach. Light/sound CS predicts sucrose.
`DTrain`	Trains a schedule in which nosepokes initiate forced-choice trials.
`Evenden`	One pellet now, or four pellets later? Systematic variation of delay across a session. Delays are 0, 10, 20, 40, 60s for a session length of 100 min.
`EvInfuse`	For intracerebral infusions. Delays are 0, 20, 60s for a session length of 60 min.
`EvNoDel`	Evenden task with no delays, but still 100 s trial period. (Can be performed with the main !Evenden application, but this simplifies use.)
`FVRIP`	FR, VR, FI, VI and probabilistic simple schedules. *Note:* the variable schedules are not random interval (RI) or random ratio (RR) schedules; they pick an interval or ratio from a range. There are some problems with this approach.
`Infuse`	Simple timer to assist with intracerebral infusion.
`Lardy`	Gives free pellets at high speed.
`PIT-mult`	Tests ‘specific’ and ‘general’ Pavlovian-instrumental transfer. Contains several training and testing stages.
`PIT-simp`	Simple Pavlovian-instrumental transfer.
`RDiscrim`	Discrete-trials reinforcer discrimination task. INCOMPLETE.
`Richards`	Adjusting-magnitude task.
`SP-PIT`	Sensory preconditioning, using appetitive simple PIT as the measure of conditioning.
`TestCBox`	Box test program for combined sucrose/pellet boxes (currently in use).
`TestIBox`	Box test program for ‘intermediate’ boxes (no longer in use).
`TestPBox`	Box test program for pellet-only boxes (no longer in use).
`TestSBox`	Box test program for sucrose-only boxes (no longer in use).
`YokedCRf`	Destined to be a yoked conditioned reinforcement application. INCOMPLETE.

For Anastasia

`5choice`	Five-choice task, designed to mimic the behaviour of the Paul Fray original but to provide better output (including recording each response with its time and location). IN PROGRESS, AC MODIFYING. ALSO NEEDS "TESSDIR" REFERENCES CHANGED TO "RUDOLFDIR". YOGI NEEDS A COPY.
`5cb-test`	Box test program for 5-choice boxes.

To use Yogi's applications (e.g. !AttMem), it'd be necessary to change the call to PROCfive_hole_boxes_yogi to PROCfive_hole_boxes.

For Yogita

`AttMem`	Two-stage five-choice task with attentional/mnemonic demands. I'M MISSING THE LATEST VERSION.
`YScrTest`	Tests multiple VDU output system (LCD displays for touchscreens)
`YTchTest`	Tests touchscreen apparatus (i.e. conventional Arachnid inputs/outputs, plus touchscreen sensors and interface)

To use other five-choice applications (e.g. !5choice, !5cb-test), it's necessary to change the call to PROCfive_hole_boxes (Filippo/Anastasia) to PROCfive_hole_boxes_yogi.

For Felicity

`TestExt`	Records responses on two levers in extinction. For cocaine/sucrose devaluation experiments. (Self-contained program; does not use libraries.)

For Yia-Ping

`TestYBox`	Box test program for Yia-Ping's boxes.

To use other applications (e.g. !Evenden), it's necessary to change the call to PROCcombined_boxes (Rudolf) to PROCyiaping_boxes.

For JP

`TestTch`	Tests marmoset touchscreen apparatus.

Bugs

- Potential for division by zero in DiscAp and 2-DiscAp if the rat never pokes in the VI? (PROCfinal_output; possibly elsewhere).
-