

# Institute for Empirical Research in Economics University of Zurich

Working Paper Series ISSN 1424-0459

### Working Paper No. 21

# z-Tree – Zurich Toolbox for Readymade Economic Experiments - Experimenter's Manual

Urs Fischbacher

September 1999



# **Experimenter's Manual**

Urs Fischbacher

# **Table of contents**

1	About z-Tree	5
2	Conducting an Experiment	7
2.1	Terms	
2.2		
	2.2.7 Conclusion of the Session with a Questionnaire	10 11 11
2.3	The Clients' Table	11
2.4	Data Windows	12
2.5	How Does a Client Establish Connection with the Server?	13
2.6	Command Line Options for z-Leaf	13
2.7	Crash Protection	14
2.8	What Happens if Subjects Suffer Losses?	17
2.9		
3	Definition of Treatments	19
3.1	Overview	19
3.2	The Stage tree	21
3.3		
3.4		
Э.Т	3.4.1 Concepts	
	3.4.2 Comments	
	3.4.3 Expressions	26
	3.4.4 Table Functions	27
	3.4.5 Restriction of a Table Function to the Members of One's Own Group	
	3.4.6 The Scope Operator	
	3.4.7 Scope Environment.	
	3.4.8 Conditional Evacution of Statements	32

	3.4.9	Arrays	. 32
	3.4.10	do Statement	. 34
	3.4.11	Loops	. 34
	3.4.12	Creation of New Records in the Program	. 35
3.5	Sub	ject and Period-specific Parameters	. 35
	3.5.1	Period Parameters	. 37
	3.5.2	Role Parameters	. 37
	3.5.3	Specific Parameters	. 37
	3.5.4	Group Matching	. 37
3.6	Lay	out	. 38
	3.6.1	Screen	. 38
	3.6.2	Items	. 38
	3.6.3	Buttons	. 40
	3.6.4	General Remarks on Boxes	. 41
	3.6.5	Standard Box	. 42
	3.6.6	Header Box	. 42
	3.6.7	Help Box	. 43
	3.6.8	Container Box	. 43
	3.6.9	Grid Box	. 44
	3.6.10	Calculator Button Box	. 45
	3.6.11	History Box	. 45
	3.6.12	Text-formatting with RTF	. 45
	3.6.13	Insertion of Variables into the Text	.47
3.7	Che	cking of Subjects' Entries	. 48
	3.7.1	Item Checks	. 48
	3.7.2	Checker	. 48
3.8	Cou	rse of the Treatment	. 49
	3.8.1	Stages	. 49
	3.8.2	Time	. 51
	3.8.3	Leaving out Stages	. 51
	3.8.4	Treatments of Indefinite Length	. 52
3.9	Mar	ket Experiments	. 52
	3.9.1	Concepts	. 52
	3.9.2	Auction Stages	. 53
	3.9.3	Contract Creation Box	. 53
	3.9.4	Contract List Box	. 55
	3.9.5	Contract Grid Box	. 56
	3.9.6	Buttons	. 57
	3.9.7	Checker	. 58
	3.9.8	Message Box	. 58
	3.9.9	Programs	. 58

3.10	) Exar	mples	59							
	3.10.1	Public Goods Game	59							
	3.10.2	Ultimatum Game	61							
	3.10.3	One-sided Auction	63							
4	Quest	ionnaires	65							
	4.1	Overview	65							
	4.2	Profit Display	65							
	4.3	Running a Questionnaire	66							
	4.4	Address Form	66							
	4.5	Question forms	66							
	4.6	Questions	66							
	4.7	Rulers	68							
	4.8	Buttons	69							
5	Instal	lations	70							
	5.1	Installation with a File Server	70							
	5.2	Installation without File Server	70							
	5.3	Setting up a Test Environment	70							
	5.4	Command Line Options of z-Tree	70							
6	Reference									
	6.1	Menu Commands	72							
	6.2	Programming	81							
	6.3	The Import Format	88							
7	Tips and Tricks									
	7.1	Calculation of Rank	89							
	7.2	Reduce Time-out Times after the Experiment has Started	89							
	7.3	Leave Profit Display Standing	90							
	7.4	Observer Subject	90							
	7.5	Checking of Exercises	90							
	7.6	Differing Endowments	90							
	7.7	Display Costs Table	91							
	7.8	Displaying the Number of Players that Have not yet Finished	91							
	7.9	Dutch Auction	91							
	7.10	Multiphase Experiment	91							
	7.11	Testing with overlapping z-Leaves	92							
	7.12	Payment of a Random Period	92							
	7.13	Own Records at the Top of a List	93							

### 1 About z-Tree

The z-Tree program was developed at the University of Zürich. It was specially designed to enable the conducting of economic experiments without much prior experience. It consists, on the one hand, of z-Tree, the "Zürich Toolbox for Readymade Experiments", and, on the other hand, of z-Leaf, the program used by the subjects.

In z-Tree, one can define and conduct experiments. One can program a broad range of experiments with z-Tree, including public goods games, structured bargaining experiments, posted-offer-markets or double auctions. The programming of z-Tree requires a certain experience. Thereafter, the effort required for conducting experiments is minimal: An experimenter with a certain amount of experience can program a public goods game in an hour and a double auction in less than a day.

On performance: In Zürich, z-Tree is used for almost all experiments that are conducted with computers. 26 PCs with 486er Processors and 16 MB RAM are connected on an Ethernet. The program always works efficiently in this configuration.

You need not read the whole manual in order to work with z-Tree. Chapter 2, on conducting, is of great importance. Chapters 2.1 and 2.2, especially, are vital. Furthermore, it is advantageous if at least one person at institutes using z-Tree is well acquainted with the remainder of chapter 2.

Those who wish to design treatments must read the introducing chapters 3.1 and 3.2. Those who only wish to design screens may confine their reading to chapters 3.4.1 and 3.6. Within chapter 3.6 all chapters up to 3.6.6 need to be read. The remaining chapters may be consulted when required.

Chapters 3.4.1, 3.4.2, 3.4.3, 3.4.4 and 3.4.5 are essential for programming. Furthermore it is advantageous to read chapter 3.5. Chapter 3.8, on the course of treatments is also important for all who wish to program experiments. The scope operator, explained in 3.4.6, is certainly the most difficult concept. It is, however, extremely important for all more complicated experiments, especially for market experiments.

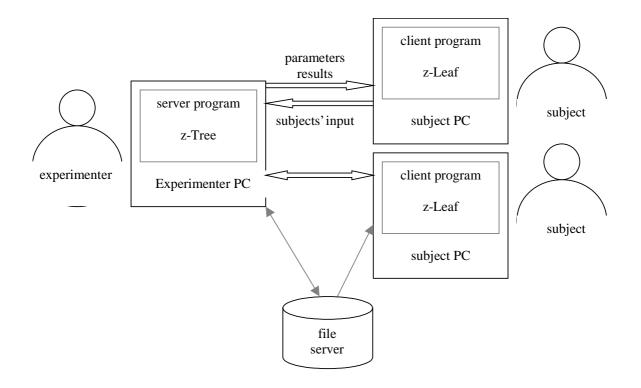
All other chapters may be consulted when the need arises.

Z-Tree is very flexible. Nevertheless, it may occur that you wish to realize something that is not covered by the program. Chapter 7 contains tips and tricks that might help you in this eventuality. Furthermore there is a web site on z-Tree at http://www.iew.unizh.ch/ztree. If you still feel something is missing or if you should find an error in the program, please send an email to ztree@iew.unizh.ch.

I would like to thank the following users of z-Tree for their patience with the program and for their suggestions how to improve the program: Vital Anderhub, Armin Falk, Ernst Fehr, Simon Gächter, Florian Knust, Oliver Kirchkamp, Andreas Laschke, Martin Strobel, Jean Robert Tyran. I would also like to thank Christina Fong, Omar Solanki and Beatrice Zanella for helping me to present the program in this manual. Omar Solanki translated the originally German manual into English.

# 2 Conducting an Experiment

#### 2.1 Terms



In a non-computerized experiment there is one or more **experimenter** and a number of **subjects** who communicate with one another through the experimenter. In a computerized experiment this communication takes place through the computer. The computer operated by the experimenter is called the **experimenter PC**. The computers operated by the subjects are called **subject PCs**. The program with which the experimenter works is called "z-Tree"; it is the **server program** or in short, the server. The program with which the subjects work is called "z-Leaf"; it is the **client program** or in short, the client.

Be careful not to confuse the server program with the file server. The latter is used to save the programs, the data and the results of an experiment. A file server is not absolutely necessary for conducting an experiment. As soon as the clients have established contact with the server, communication between server and client takes place directly, not via the file server. However, a file server does facilitate startup. On the file server, information can be stored centrally so as to be readable by all. Thus the clients

may, for example, find out on which computer the server was started. This makes it easily possible to use different computers as experimenter PCs.

By session we mean the events that occur in the time span between the arrival of the subjects and the moment they receive payment. A set of corresponding sessions constitutes one experiment. A treatment is a part of a session that is stored in a file. How treatments are defined, is explained in the chapter "Definition of Treatments". Each session consists of one or more treatments followed by a set of questionnaires. Questionnaires can also be freely defined. This will be explained in the chapter "Definition of Questionnaires". In this chapter consider treatments and questionnaires as black boxes. We will explain in detail the structure of treatments and questionnaires where necessary.

#### 2.2 Quick Guide

The running of a session consists of the following steps:

- 1. preparation of treatments and questionnaires
- 2. start-up of the experimenter PC
- 3. start-up of the subject PCs
- 4. arrival of subjects
- 5. start of the session and the first treatment
- 6. observing the course of the session
- 7. start further treatments
- 8. conclusion of session with a questionnaire
- 9. payment
- 10. switching-off of machines
- 11. data analysis

While conducting an experiment we mainly work with the menu "Run". The order of commands in the run menu corresponds, on the whole, with the course of the session.

#### 2.2.1 Preparation of Treatments and Questionnaires

Before the session begins all treatments and questionnaires in z-Tree are defined and saved on the file server. How you define treatments and questionnaires is explained in chapter 3 (Definition of Treatments). Treatments are different for different numbers of subjects. Hence, in case you are not sure whether all subjects will show up, you would like to have the possibility of running a session with varying

numbers of subjects - depending on how many show up. In simple cases the number of subjects, which is fixed in every treatment, can be changed at the beginning of the session, i.e., before the start of the first treatment. In cases where changing the number of subjects involves more than simply changing this number<sup>2</sup>, you have the option of preparing treatments for different numbers of subjects.

We recommend that you make a list of the treatments and questionnaires. It should include the names of all of the treatment files for all possible numbers of subjects. Alternatively, you may list how the treatment files are to be adjusted for all possible numbers of subjects.

#### 2.2.2 Start-up of the Experimenter PC

Switch on the experimenter PC, log in<sup>3</sup> and start "z-Tree". Z-tree has a window that displays which clients have started up and established contact. This information is displayed in what is called the clients' table, which will be described in more detail in chapter 2.3 (The Clients' Table). You can open the clients' table with the command "Clients' table" in the Run Menu. The first column of the table lists the name of the client, which is (normally) identical with the host name of the PC.

#### 2.2.3 Start-up of the Subject PCs

Switch on the subject PCs and log in<sup>4</sup>. On the server we can check the clients' table to see which clients are connected. With the command "Sort Clients" from the menu "Run" we can sort the list for control purposes. When we have finished, we can shuffle the lists again with "Shuffle Clients".

#### 2.2.4 Start of the Session and the First Treatment

As soon as the subjects arrive, count them. Next, open the treatments with the correct number of subjects. Alternatively, you may adjust the number of subjects for all treatments that will be started.

If we have started up more clients than necessary we can discard the extra clients. To achieve this we click the clients in question one at a time in the clients' table and choose the command "Discard client" from the run menu.

<sup>&</sup>lt;sup>2</sup> Within a Stranger design, for example, the group matching has to be redone.

<sup>&</sup>lt;sup>3</sup> The name under which one logs in depends on the installation. Typically one will set up a user account for all persons who carry out experiments.

<sup>&</sup>lt;sup>4</sup> The login name depends on the installation. One will set up a user account with highly restricted permissions for the subjects.

Now we open the file containing the first treatment using the open command from the file menu. Generally this is the welcome treatment that bids the subjects welcome. We then begin the treatment with the command "Start Treatment" from the menu run menu. Note that once we start the treatment, the number of subjects is fixed. Thereafter, this number cannot be changed! The treatment starts and the subjects may begin. It is worthwhile to minimize the treatment window after this point, as it now no longer needs to be seen.

#### 2.2.5 Observing the Course of the Session

Treatments have different stages which essentially correspond to subjects' screens. The stage of the experiment each subject has reached is shown in the second column of the clients' table. It is labeled with "state". On the basis of the client state you know which screen the subjects are viewing.

The data of a session is gathered in tables. The commands "time table", "globals table", "subjects table", "contracts table", and "summary table" from the run menu, open windows that show these tables. One important table is the time table where you can see how much time the subjects have at their disposal. You also see when the time for data entry expires. This gives you the possibility to determine whether there is something wrong at that subject's place. In general, the subjects table contains the subjects' entries. It is therefore one of the most interesting tables, if you want to know how subjects behave in this experiment.

#### 2.2.6 Starting Further Treatments

Once a treatment is concluded all subjects are in the state "Ready". You see this in the clients' table. The clients still see the final screen of the last treatment. A new treatment can be started like the first with the command "Start Treatment".

#### 2.2.7 Conclusion of the Session with a Questionnaire

A session is concluded with a questionnaire. A questionnaire consists of a series of screens. In every case such a series of screens includes an address and an end-screen. Start the questionnaire with the command "Start Questionnaire". This command is located at the same place as the command "Start Treatment" when a questionnaire window is active. While the subjects answer the questionnaire, the clients remain in the state "Questionnaire". The window "subjects table" indicates which questionnaire is being answered at the moment.

#### 2.2.8 Payment

Z-Tree writes a payment file as soon as all subjects have supplied their address. This payment file has the suffix ".pay". It contains the names of the subjects, the name of the client (usually the name of the PC) and the amount the subjects have earned. You can open this file from every computer that has access to the file server.

#### 2.2.9 Switching-off of Machines

When all subjects have finished, they are in the state "Ready". Now you may exit the program and switch-off the computers.

#### 2.2.10 Data Analysis

The data is stored in files. The names of these files consist of date and starting time. During a session that starts on February 17, 1999 at 3pm, the following files are created:

990217P0.pay: Contains names and profits of the subjects. This file can be used as a signature form for

payment.

990217P0.adr: The address file contains the addresses of the subjects.

990217P0.xls: The xls file contains all tables that were shown during the session. These tables consist

of information on the definition of treatments and subjects' entries. This is the file we

will need for data analysis.

990217P0.sbj: Answers to the questionnaires and the roles that were played. Without subjects' names.

990217P0.gsf: GameSafe in binary form.

#### 2.3 The Clients' Table

The clients' table shows which clients are connected to the server. It also contains information on the state of the clients, i.e., which screens are currently being displayed to the clients. You may open the clients' table with the first command from the Run Menu.

Each line corresponds to a subject. This subject is seated at a PC on which the client is running (column "Clients"). As long as no treatment is started, the clients can be moved. As soon as you start a treatment, the order of the clients is fixed. After a treatment has started, only those clients that are not taking part in

the current session can be moved. These clients are listed at the end of the table. So long as no treatment has been started, the clients can be sorted and shuffled with the commands "Sort Clients" and "Shuffle Clients" respectively. The clients' table has five columns:

**Client:** Name of the client. In general this is the name of the subject PC. If a client is no longer connected, its name appears in brackets. If a client with the same name should reconnect, it can continue at the same place where the previous subject left off. The caption to the clients' table gives the number of clients currently connected.

**State:** The state in which the subject is at the time. Example: "Ready" when the subject is waiting for the next treatment or for the next questionnaire.

**Profit:** Accumulated income made in the previous treatments. The profit made in the current treatment is *not* included therein. This profit is indicated in monetary units (Fr., DM, Euro, \$), as various treatments may have different exchange rates.

**Show up fee invested:** TRUE if the subject has suffered losses and has declared himself ready to account for these with his/her show-up fee.

**Money added:** Extra money that the subject has injected into the game after losses in order to be able to continue. This sum is of importance for the calculation of losses. The profit in the payment file is always the net income the subject has gained from the experiment. At the hour of payment, injected money must therefore be added to the amount indicated in the payment file.

#### 2.4 Data Windows

The window "time table" shows the time as it appears for the subjects. If the subjects have different times appearing on their screens (for example, in a posted offer), the individual times each appear on one line. The order is the same as in the clients' table. In order to see for which subject time is running out, it is best to position the two tables beside each other.

In the windows "globals table", "subjects table", "contracts table" and "summary table" the content of the appropriate table is displayed. In these tables the sizes of lines and columns can be changed. These tables are automatically exported to the xls file.

Fischbacher/ z-Tree/ Mar. 29, 99 - 12 -

#### 2.5 How Does a Client Establish Connection with the Server?

A session begins when the experimenter starts the server program z-Tree on the experimenter PC. After this the client program z-Leaf is started up on the subject PCs. Z-Leaf establishes contact with z-Tree on the experimenter PC.

#### How does the client know the server's address?

Z-Leaf can determine the server's TCP/IP address with command line option /server ipaddress. If, for instance, z-Leaf is started with the command

Zleaf /server 100.20.10.233

then z-Tree must run on a PC with the TCP/IP address 100.20.10.233. This method is very useful if you want to run an experiment without a file server. If z-Tree is located on a file server, however, there is a more comfortable way: Z-tree automatically writes it's TCP/IP address into the current directory in the file "server.eec". If z-Leaf is started from the same directory (on the same PC or on another) it reads this file. If z-Leaf does not find this file, it can look for it in the directory "c:\expecon\conf". If it is unable to find either, it can take the TCP/IP address of the local machine. It can now seek the server at this address. If the server is not found here either, the person who starts the client is asked if he or she would like to try again.

#### How to fix the name of a client?

If z-Leaf is started up by means of a name on the command line (e.g., z-Leaf /name T1), this will be the name of the client (e.g., T1). Otherwise the name will first be sought in the file "name.eec" in the local directory. In this way you can run several clients on one PC. If the file "name.eec" does not exist, the host name of the PC is used. The host name is entered in the network control panel under TCP/IP.

#### 2.6 Command Line Options for z-Leaf

Z-Leaf can be started with options. Example:

zleaf /language en /name test2

starts the client in English with the name test2.

Fischbacher/ z-Tree/ Mar. 29, 99 - 13 -

About the individual options:

/language lan lan can have the value "en" English and "de" for German. This option

concerns only a few general error messages like "In the field xx the highest

possible entry is yy". All other texts are defined in treatments or

questionnaires and can be changed there.

/server adr adr contains the IP address of experimenter PC.

/channel ch Determines the channel through which contact is established. The actual

channel is 700+ch.

/name name name will be the name the client has in the clients' table.

/size widthxheight Size of the client's window for the format widthxheight. In this way you

can test on a large screen how the layout will look like on a small screen.

For example, the VGA format is 640x480.

#### 2.7 Crash Protection

All data is stored in various files in readable form as soon as it is complete. Besides this, all messages exchanged between server and clients are stored in a (non-text) file, the GameSafe. With the help of these files it is generally possible to continue the experiment after the crash of a computer.

#### What to do if a subject PC crashes or the client's program is blocked?

If the PC is still working, it can simply be restarted. It then automatically receives again at accelerated speed all messages it had received before. After this, it is in exactly the same state it would be in, had the crash not occurred. Of course this also means that it can be further than before: If in the meantime the time set for a stage has passed, the treatment moves ahead by a stage, even if not all the clients are connected.

There are several possible reasons why a subject PC may not go on. It is possible that the server is very busy.<sup>5</sup> In this case you simply have to wait a little. However, if the server is in a wrong state due to a programming error, the server has to be restarted.

- 14 -

<sup>&</sup>lt;sup>5</sup> In particular, this can happen when the server is replaying data from GameSafe to the client (see below).

If the subject PC is out of order, you can start a new PC. Discard the old PC from the clients' table with the command "Discard Client" from the Run Menu. As soon as the new client appears, it can be manually assigned to the subject who was working on the old computer by selecting the client field of the new client and moving it over the field of the old client.

#### What to do if the experimenter PC crashes?

Two mechanisms serve to continue the session after a crash of the experimenter PC. Z-Tree stores every GameEvent, i.e., every piece of information that is exchanged between server and clients, in GameSafe. Thanks to this an experiment may be continued as if nothing had happened. However, as this is relatively time-consuming, particularly important intermediate results are specially saved.

Z-Tree stores the assignment of clients and the profits accrued in the course of a session in temporary files: At the beginning of the first treatment the assignment of clients is stored in the file "@lastclt.txt". At the end of each period the current total profit of the treatment is written into the file "@profit.txt". At the end of a treatment the total profit is written in the file "@tprofit.txt" and the file "@profit.txt" is deleted. This is why the profit can be read in again after a crash. In this way a session can be played to an end from the last unfinished period on.

The files "@lastclt.txt" and "@tprofit.txt" are deleted at the end of the session so that there is no confusion in the following session.

#### Start without GameSafe

In order to start the server from the last unfinished period on, you need to do the following:

- 1. Restart the clients from z-Tree with the command "Restart client" from the Run Menu. In the clients' table, you can check which clients are again connected.
- 2. At the beginning of the first treatment a message appears asking: "Same assignment of computers as in last session?" This question must be answered with "yes". By doing this the profits made so far are also taken into account.
- 3. If you cannot start with the first period of a treatment, you need to do the following: If, for example, 3 out of 10 periods have already been played, you need to define a new treatment with number of periods = 10 and trial periods = -4 (minus 4). Now 7 periods will be played, starting with 4. If the number of trial periods is negatively defined, z-Tree adds the profit from "@profit.txt" to the total

profit in the treatment when the first period starts. If you should begin a treatment with the same assignment as in the last session, z-Tree adds the profits from "@tprofit.txt" to the total profit of the session. What is lost in this kind of restart, however, is the history displayed to the subjects. If you should change the number of periods in a treatment, you should be careful to properly place the parameters which differ from period to period.

#### Start with GameSafe

The second method consists of restarting the server with GameSafe. This method has the advantage that one attains the same state as before the crash. However, this can be quite time-consuming.

- First you start the server empty, i.e., without a document.
- You select the command "Replay old session..." from the menu "Run".
- A dialog appears. In this dialog the file containing GameSafe, that is to be replayed, can be selected.
   Generally, you need to select the second lowest one, because at the beginning of the server program a new GameSafe is created.
- Now you need to indicate how many messages should be read. It is particularly useful not to read all
  messages if the server has crashed in a reproducible fashion, i.e., if it has relapsed into the previous,
  unsatisfactory state after a new loading of GameSafe.
- When the messages are read in, all dialogs that appeared during the experiment appear again. These dialogs need to be completed the same as before. In the beginning a dialog may appear which had not appeared during the experiment: "Same assignment of computers as in last experiment?" If this dialog had already appeared in the original experiment, it needs to be answered in the same way. If it has not appeared before, it needs to be answered with "no".
- Finally the clients may be restarted with the command "Restart Clients".

#### What to do if the file server crashes?

No special measures have been designed for this scenario. The file server needs to be restarted. After this the experimenter PC may be restarted as explained above. It depends on your configuration of the file server how much data will get lost in the case of a file server crash.

#### 2.8 What Happens if Subjects Suffer Losses?

Losses by subjects are covered by the following sources:

- 1. Previous profits.
- 2. Lump-sum payment.
- 3. Show-up fee.
- 4. Money injected during the session.

When a subject suffers losses, messages appear on the screen. The text of these messages is fixed in the treatment (in the background).

If the first two sources cannot cover the losses, but the show-up fee can, a message appears on the subject's screen. It informs the subject that he or she can now choose either to use the show-up fee or drop out of the game. If the subject uses the show-up fee, he or she may simply play on.

If, on the other hand, the subject chooses not to use the show-up fee to cover his/her losses, he or she reaches the state "BancruptShowupNo" and you have to release the subject from the server. If the show-up fee is used and exhausted, a message appears on the subject's screen informing him/her that he or she has made a loss. This message can be concluded by a question. If the subject answers this question with "yes", he or she arrives at the state BancruptMoreYes. By answering "no", he or she arrives at the state BancruptMoreNo. In this case also the experimenter has to release the subject from the server. A dialog appears by double-clicking the "State" field. There are three possibilities:

- 1. You allow the subject to continue. In this case you need to type a number into the field "Amount injected" that is higher than the current loss. You either make the subject pay this amount or, alternatively, consider it to be the credit limit. You take it into account when ascertaining whether the subject is in the minus. However, it is not shown together with the subject's profit, nor is it considered in the payment file. This means that you need to add the amounts injected to the amount indicated in the payment file. In this way, the payment file lists the net amounts paid out to the subjects.
- 2. Another subject takes on the role of the subject released. In this case all profits are reset to zero and one is able to continue work on the PC.
- 3. The subject drops out. You need to be careful about this as it may change group sizes and the information on the active subjects may therefore no longer be correct.

Fischbacher/ z-Tree/ Mar. 29, 99 - 17 -

#### 2.9 The Files Generated by the Server

The following files are created in the course of the experiment. They are distinguished by their suffixes.

- pay Payment file. Name of subject and profit.
- adr Addresses of the subjects.
- sbj Questionnaire responses. Without subjects' names.
- *xls* Contains all tables shown in the course of the session. As the tables are stored chronologically, you need some preparation to work with them. The first column contains the treatment number, the second the name of the table and the third column contains the period variable. It is a good idea to sort the whole file by these three columns. Then the tables can be copied into separate tables.
- gsf GameSafe in binary form. It can be exported into a readable file but this file will be gigantic.

The starting time of the session is used as file name. The format is YYMMDDHM. YY means year, MM month, DD day, H hour and M minutes. In this way the start of the session can easily be reproduced. The hours correspond to the following table:

Code	A	В	C	D	Е	F	G	Н	I	J	K	L	M	N	О	P	Q	R	S	T	U	V	W	X
Hour	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23

#### The minutes correspond to the following table

Code	0	1	2	3	4	5	6	7	8	9	A	В	С	D	Е	F	G	Н	I	J	K	L	M	N	О	P	Q	R	S	T
Hour	00	02	04	06	08	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40	42	44	46	48	50	52	54	56	58

Fischbacher/ z-Tree/ Mar. 29, 99 - 18 -

# 3 Definition of Treatments

While planning an experiment, the first question we need to answer is, in what way the experiment is defined as a game: What are the action sets? How are the profit functions defined? What is the move structure like? What are the information conditions? If the experiment is computerized, we additionally need to consider how the move structure can be organized in a sensible way and what the layout of the screens is to be like.

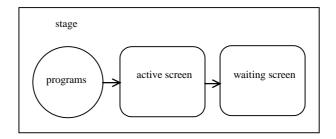
Information on how to implement action sets and profit functions in z-Tree can be found in the chapter "Programming". Information conditions are defined by the screen layout. This is discussed in detail in the chapter "Layout". There is a chapter on "Move Structure". The final chapter, "Market Experiments", discusses the programming of market experiments. Firstly, in the chapter "Overview", the most important concepts are introduced.

#### 3.1 Overview

As previously explained in the chapter "Conducting an Experiment", a session consists of several treatments. The chapter you are now reading deals with the definition of treatments. Each treatment is stored in a file with the suffix ".ztt".

A treatment consists of a number of **periods**. This number is fixed. In every period a number of **stages** are gone through. On the first screen of each stage the subjects can make their entries. This is called the **active screen**. When the data has been entered or time has run out, the second screen that belongs to this stage appears, the **waiting screen**. Normally, one progresses on to the next stage when all subjects have reached the waiting screen.

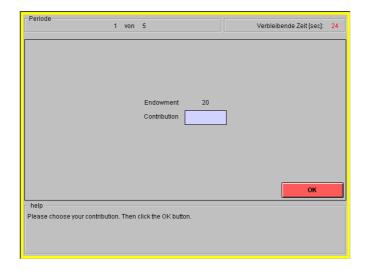
At the beginning of each stage calculations are carried out. These calculations are defined in programs.



#### **Example of a simple Public Goods Game**

In this treatment the subjects are matched into groups. They decide how many out of 20 points they want to contribute to a public good. This amount is called g. Their profit is made up of two parts: The first part is the amount they keep: 20-g. The second part is the income from the public good: All contributions are pooled, multiplied by 1.6 and distributed among all subjects in the group.

The treatment consists of two stages: contribution entry and profit display. In the program of the "Background" parameters are set: The endowment M, which is shown on the active screen of the input stage, and the parameter eff. The active screen of the contribution entry stage displays the endowment and contains a field for the entry of the contribution. On the waiting screen all that appears is the message "Please wait". The profit is calculated in the program of the profit display stage and the active screen displays the profit, among other things. On the waiting screen we again only have the "Please wait" message.

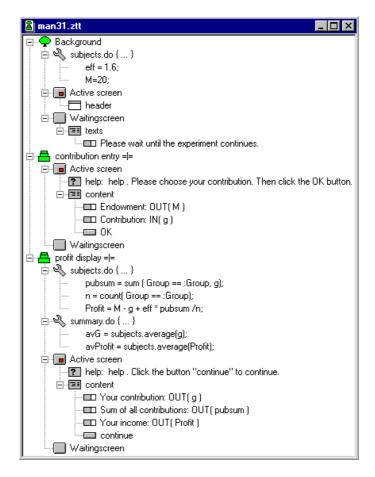


The above screen dump shows the active screen of the contribution entry stage of the public goods game. It consists of various **boxes**. The **header box** at the top gives the period number as well as the time remaining. The **help box** at the bottom displays instructions for the next step. The **standard box** lies between these two. In the standard box parameters, input fields and a **button** appear with which input can be concluded.

We call a unit of information that appears in the standard box an item. In the case above we have two items in the standard box: The endowment and the contribution. Input is concluded by clicking a button labeled OK.

#### 3.2 The Stage tree

The stages of a treatment are depicted in the shape of a tree diagram. The figure below shows the stage tree of a public goods game:



All elements of the treatment are arranged hierarchically in this figure: Stages contain programs and screens, screens contain boxes, and boxes contain items. In order to have an overview, you can fade out and fade in the lower hierarchy levels at will. By double-clicking you can view and change the parameters of the elements. Besides this, the elements may be moved and copied. You can insert new elements with menu commands.

#### 3.3 The background

The common elements for all stages may be inserted and defined in the "Background". In our example, all active screens have a header window and all waiting screens a text "Please wait until the game continues". The programs of the background are run at the beginning of a period. They are used for

defining constants. In the background itself, i.e., when you double-click it, some central parameters of the treatments can be viewed and changed.

**Number of subjects:** This number is constant for a whole session. Therefore all treatments must have the same number of subjects.

**Number of groups:** Number g of groups to which the subjects are assigned. This means that the subjects belong to a group with an ID between 1 and g. Group matching is a variable like any other. There is also the possibility of empty groups and it is possible to form "new" groups, i.e., groups with ID greater than g. The input in this field is of particular importance for automatic group matching.

Number of trial periods: Number of periods in which no profit is paid out. The number entered can be negative. In this case the period counter starts later. If, for instance, the number of trial periods is -2 and the number of paying periods is 10, then the treatment extends from period 3 to period 10.

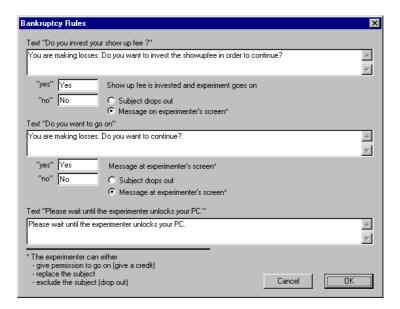
Number of paying periods: Number of periods in which the profit is paid out.

Exchange rate [Fr. / ECU]: How many Franks (\$, DM, £) are paid out per "experimental currency unit" [ECU]

**Lump sum payment** [ECU]: Amount in internal unit that is credited at the beginning of period 1 *of the treatment*. This amount figures in the variable TotalProfit.

**Show-up fee [Fr.]**: Amount credited to the subjects at the start of the session. Only the show-up fee of the first treatment played is decisive.

**Bankruptcy rules:** By clicking this button the parameter of the bankruptcy rules may be adjusted.



Fischbacher/ z-Tree/ Mar. 29, 99 - 22 -

The first question appears when the subject has suffered losses that can be covered by the Show-up fee.

The second message appears when the losses suffered by the subject exceed the show-up fee.

The fields near "yes" and "no" contain the labeling of buttons. If both buttons are not empty, the user has

a choice. If one of the two buttons is empty, the user can only confirm the given answer. Depending on

what button the user has clicked, the action stated beside the button is executed.

Show-up fee is injected and the game continues: The show-up fee is used to cover the losses and the

game continues.

The player drops out automatically (OUT): By putting the subject into the state OUT, he or she is

taken out of the game. This is not wholly unproblematic - the programs need to be equipped for this

process. You can, for example, set an input variable to an impossible value, in order to be able to discern

that a subject has dropped out.

**Message to the experimenter:** This case is explained in chapter 2.8.

3.4 **Programming** 

3.4.1 **Concepts** 

Information on the state of the experiment is stored in a database. This database consists of tables. The

lines of a table are called **records**, the columns **variables** and the individual entries **cells**. In z-Tree, the

cells can only contain numbers, not text. The following tables exist:

subjects: Table that contains a record for each subject. This is the main table. It is freshly set up for

each period. The subjects table of the preceding period is available under the name OLDsubjects. Values

from earlier periods than the immediately preceding one are not accessible.

In the subjects table, the variables Period, Subject, Group, Profit, TotalProfit and Participate are always

defined. Period is the number of the current period, starting with 1 in the first paying period. Subject is

the ID of the subject, starting with 1. Group is the group number. Profit is the amount won that

determines payment. If subjects are paid, this variable has to be set in the treatment. TotalProfit is the

sum of all profits in the treatment. This variable should not be changed; it is automatically calculated.

Participate is explained in the chapter "Course of the Treatment".

**globals:** Table that contains a single record. In it, values are stored that are the same for all subjects. It is freshly set up for each period. The globals table of the preceding period is available under the name OLDglobals. In the globals table the variable Period is always defined.

**summary:** Table with one entry per period. This table gives the experimenter an overview over the current goings on in the game. In the summary table the variable Period is always defined.

**contracts:** This table is used mainly for market experiments. New records can be added to this table and existing records can be changed. The chapter "Market Experiments" explains the procedure in detail.

If you want access to a certain field in the database in order to read it or to change it, you need the following information:

- the table
- the record
- the variable

Variables are accessed in programs. Programs always run in a particular record in a particular table. This record is called the current record. Variables are defined by their names. When the variable of a record is fixed, the cell is fixed with it. In tables every record has the same variables. A program for a particular record can therefore be calculated for every record of this table.

Let us take a program that is located at the beginning of a stage. It is run when the subjects reach this stage. In other words, a program for the subjects table is executed for the record of those subjects that have reached the stage in question. The program for the summary table is run for the current period when *all* subjects have reached the stage in question.

The name of a variable can be any word. To be precise, a variable can be composed of letters, numbers, and the underscore character "\_"; it must start with a letter. Not allowed are diacritical marks, blanks and punctuation marks. Capital letters and small letters are distinguished. Therefore hallo, Hallo and hAllo are three different variables. You can give long names to variables. This makes programs easier to understand. If you want to give a variable a name made up of several words, you can separate the words with underscores or you begin each word with a capital letter.

Examples of allowed names of variables:

A contribution23 v\_2\_13 Buyers\_offer BuyersOffer

Not allowed as name of variable:

starts with a number
2 13 contains spaces
1.0 contains dots
Buyer'sOffer contains apostrophe

Variables are created when needed: Either through input from the subject or through definition in programs. Input from subjects is defined with the help of items. This is explained in the chapter "Items". In programs variables are defined by the syntax:

This statement is called an assignment because the expression on the right side of the equation is calculated and assigned to the variable on the left side. In case the variable does not already exist, it is defined by this statement. Note that every statement is concluded with a semi-colon.

Example (g was already defined, M and prv are to be defined):

M = 20; prv = M-g; g M prv -> g M 5 20

	_				_							
g		g	M	prv		g	M	prv		þ	M	prv
5		5			->	5	20	15		5	20	15
12	->	12	20	8		12	20	8		12	20	8
7		7				7			->	7	20	13

This example shows what happens to the table when the program is consecutively run for the second, the first and the third Record. The empty spaces signify undefined values. One should not use them. When a program is executed for *all* records of a table (which is generally the case), then there are eventually no undefined fields left.

#### 3.4.2 Comments

In order to be able to easily understand a program days and weeks after you worked on it, you insert comments. Every text between /\* and \*/ as well as between // and the end of the line is a comment and of no consequence for the actual running of the program, i.e., when the program runs, this text is simply omitted. We will use comments in our example to explain the reasons why a certain statement has a particular form.

#### Examples:

```
a=1; // initialize
b = sum( /*cos(x*x+) */ a);
// there is an error in the expression in the second line;
// therefore we put questionable parts into a comment to
// localize the error
```

Note: Comments with /\* and \*/ may not be nested: After /\*, the first occurrence of \*/ terminates the comment. The following program is therefore illegal:

```
/* discard the following lines by putting them into a comment
a /* first variable */ =1;
b = 2;
*/
```

You may also use comments in error searches. To find a wrong statement, you can turn doubtful passages into comment until no error message appears anymore. Later you can take them out of comment again.

#### 3.4.3 Expressions

All basic kinds of calculation are permitted in an expression. As usual, multiplication and division are calculated before addition and subtraction. In all other cases you calculate from left to right. Thus 2+3\*4 makes 14 and not 20, 10-5-2 makes 3 and not 7. A series of functions is furthermore available, for example min, max, sin, random, round.

Besides "normal" expressions there are also **conditions**. These are expressions that represent either true (TRUE) or false (FALSE). An example of a condition is g>=h. This condition results in TRUE if and only if the variable g is at least as great as the variable h. The condition m==n results in TRUE if and

only if the variables m and n are equal. Note: In contrast to the assigning equals sign the comparing equals sign consists of two marks of equality.

Conditions may not be directly assigned to variables, however they may be used in expressions. With

a conditional calculation can be carried out. Here, x and y are normal expressions, a is a condition. The following expression calculates a step function:

$$y = if (x<0, 0, if (x<=5, 1, if (x<=10, 2, 3)));$$

If you wish to calculate a condition and keep it for future use, you have to store it in a normal variable. In this case you use the standard 0=FALSE and 1=TRUE.

The reference section includes a complete list of all operators and functions that an expression can contain.

#### 3.4.4 Table Functions

In the public goods example, we need to calculate the sum of all contributions made by all members of a group. However, the expressions described so far always refer only to the current record. In this example, we need to carry out calculations over the whole table. We call such calculations table functions. For instance, if g is the variable of the contribution, then

$$S = sum(g);$$

defines a new variable S as the sum of the contributions of all subjects. The variable g that appears in this expression now no longer belongs to the same record as S. If i is the number of the current record, then the expression above mathematically means

$$S_i = \sum_j g_j$$

Hereby j loops over all records.

Of course, the argument of a table function may again be an expression. This expression is then calculated for every record of the table and these results are added in the case of the sum table function. In this way the program

```
x = sum (cos(a * b));
```

corresponds to the mathematical expression

$$x_i = \sum_{j} \cos(a_j * b_j)$$

In every table function you can insert a condition as a first argument. This condition is checked for every record and the table function only applies to the records that satisfy this condition. Example:

```
z = average (x>0, y);
```

Here the average y is calculated for records having positive x.

Table functions can also be evaluated in other tables. In order to do this you simply put the name of the table followed by a dot before the table function. For example, if you wish to calculate the average profit of all subjects and put it into the summary table, you write:

```
avProfit = subjects.average ( Profit );
```

The find function is a special table function. With it you gain access to a cell in another table or record.

```
x = find( Subject == 12, Profit );
```

Here a record is sought for which the variable Subject has the value 12. For this record you choose the variable Profit and assign it to x (x in the *current* record).

#### 3.4.5 Restriction of a Table Function to the Members of One's Own Group

One can perform table functions on the members of one's own group with the function same:

```
totalSum = sum( x ) ;
groupSum = sum( same( Group ) , x) ;
```

In the first line the sum of all x is assigned to the variable totalSum. In the second line, the x values of all members of one's own group is calculated. The members of ones own group are characterized by the fact that in their record the variable Group has the same value (as I have). Adding the condition same (Group) to a table function is like carrying out the table function separately for each group.

This explanation contains all the basic instructions. The function same is based on the concept of the scope operator, which is explained in the following chapter.

#### 3.4.6 The Scope Operator

Let us suppose you want to calculate the expression

$$x_i = \sum_{j} \cos(a_i * b_j)$$

It is the same expression as in chapter 3.4.4 except for the fact that we wish to use, in every summand of the sum, the variable a of the record in which the cell  $x_i$  lies, and not the a of the record of  $b_j$ . In order to express this, the variable a is preceded by a colon. This colon is called the **scope operator**.

```
x = sum(:a * b));
```

Let us look at a table with three records in which the variable a has the values 2, 4 and 6 and the variable b has the values 5, 12 and 7. After the execution of the following program, the table contains the values as shown in the table below.

```
c = sum ( a * b);
d = sum ( :a * b);
e = sum ( :a * :b);
```

a	b	c=sum(a*b);	d=sum(:a*b);	e=sum(:a*:b);
2	5	10+48+56=114	10+24+14= 48	10+10+10=30
4	12	10+48+56=114	20+48+28= 96	48+48+48=144
8	7	10+48+56=114	40+96+56=192	56+56+56=168

As the concept of the scope operator is very important for various applications let us undertake a further interpretation. When we calculate an expression, we fix a current record. When we execute a table function, we calculate an expression for every record of a table. With every step another record becomes the current record. With the scope operator we may go back to the variables of the "old" current record, the record that lies *outside* the table function. With this respect, the scope operator gives us access to a wider scope of cells.

When a table function is carried out within another table function, this results in an expression of this table function with three records that are accessible. Let us suppose the index of the current record is i, the (invisible) index of the first table function is j and the running index of the table function k. Using the scope operator :x gives us the field in record j in this table function; Only by doubling the scope

operator, ::x, do we reach the field in records i. In order to see clearly what is happening, it is best to carry out table functions in another table. Suppose A, B and C are tables that all contain a variable v. The following expression is an expression in table A. The line underneath the expression specifies which v is being used at a particular place.

```
x = v + B.sum (v * : v - C.product (v - : v - : : v))

A

B

A

C

B

A
```

In this example the variable v occurs in all tables. However, sometimes it may happen that a variable only occurs in one table. In this case you may omit the scope operator. Let us suppose that the variable a occurs only in the table A, b only in the table B and c only in C. Then the following expressions are equivalent:

```
x = a + B.sum (b * :a - C.product(c - :b - ::a))

x = a + B.sum (b * a - C.product(c - b - a))
```

Note, however, that not using the scope operator can be dangerous. If at some time a variable a had been defined in table C, then the product of the second expression no longer goes back to the a in table A but to the a in table C.

A common use of the scope operator consists of calculating table functions of members of the subject's own group. The following expression calculates the sum of all g in the subject's own group.

```
s = sum(Group == : Group , g) ;
```

If we know that, at most, groups 1, 2 and 3 exist, we can calculate s without the Scope operator:

```
s = if( Group == 1, sum(Group == 1, g),
    if( Group == 2, sum(Group == 2, g),
    if( Group == 3, sum(Group == 3, g))));
```

However, this expression is more complex, more susceptible to error and less general.

As performing table functions on one's own group is something very common, the function **same** was specially introduced for such calculations. The expression same(x) is an abbreviation of x == :x. The expression above may therefore be written in the following way:

```
s = sum ( same ( Group ) , g ) ;
```

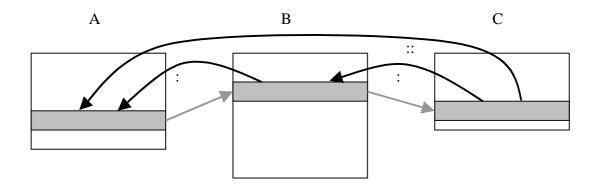
Thanks to this operation the scope operator becomes invisible. It is not necessary to understand the scope operator in detail in order to understand a program intuitively. However, if you want to write your own programs, a deeper understanding of the scope operator is crucial.

#### 3.4.7 Scope Environment

Let us consider the expression c-:b-::a in the assignment

```
x = a + B.sum (b * :a - C.product (c - :b - ::a))
```

It is possible to imagine that the expression c-:b-::a is calculated in a scope environment. This environment consists of an immediate environment, the record in the table C, and an environment that is somewhat further away: the record of table B. The latter can be accessed by the scope operator. The record of table A is even further away. Having access to it requires a double application of the scope operator. The graph below illustrates the situation. The gray arrows correspond to the call up of the table function and the bent arrows to the application of the scope operator.



We could imagine that the current record in table A is not the beginning of this chain. Instead, starting from the current record in A, it should be possible to fall back on more fundamental tables with the help of the scope operator. This possibility is realized in z-Tree. Each program call up is constructed in such a way that the (only) record of the globals table is placed at the beginning of this chain. Thanks to this you can access a variable x in the globals table in the following ways: globals.find(x), x and, if the variable x does not occur in the current table, simply x. In market experiments one utilizes this facility more widely.

The maximal scope operator is defined by  $\setminus$ . It corresponds with the maximum possible number of colons. As the globals table always constitutes the most far-reaching scope surrounding,  $\setminus x$  always corresponds to the variable x in the globals table.

#### 3.4.8 Conditional Execution of Statements

If you wish to carry out something only under certain conditions, you can utilize the if-instruction. It may be utilized in the following forms:

```
if ( condition ) { true_statements }
if ( condition ) { true_statements } else { false_statements }
```

The condition is calculated first. If it should result in TRUE, the instructions in the true\_statements are carried out. If the condition should result in FALSE, nothing is done in the first case, and in the second case the instructions in the false\_statements are carried out.

Example:

```
if ( type == FIRM ) {
    Profit = v * e - w;
    othersProfit = w - c;
}
else { // type == WORKER
    Profit = w - c;
    othersProfit = v * e - w;
}
```

You may also carry out conditional calculations by using the if *function*. However, carrying out many calculations in this way requires constant repetition of the condition. This is inefficient and can become confusing.

Note: The if-statement makes it possible to calculate a variable for certain records only, not for all records. This can result in undefined cells.

#### **3.4.9 Arrays**

Let us consider a treatment, in which we wish to use the strategy method for a simple game. There are two types of players: A and B. The A players select a number a between 1 and 10, the B players select a

number b for every possible a, i.e., they select b1, b2, b3,...b10. Let the profit be the absolute difference between the two numbers a and b. The formula for the calculation of profits looks something like this:

This is very long-winded.

In such cases it is wisest to define an array, i.e., an indexed variable. As index set every finite, equidistant subset of numbers is allowed. The index set needs to be defined by an array instruction before the first use of the array. The array instruction has the following forms:

```
array arrayvar[];  // defines an array with indices from 1 to number of subjects
array arrayvar[n];  // defines an array with indices from 1 to n
array arrayvar[x, y];  // defines an array with indices from x to y
array arrayvar[x, y, d];  // defines an array with indices from x to y with distance d.
```

The access to array elements is carried out with arrayvar[indexvalue]. The expression indexvalue is rounded to the nearest possible index and the corresponding value in the array is fetched or filled.

Example:

The example introduced at the beginning of this sub-chapter would now look like this:

#### 3.4.10 do Statement

With the do statement calculations can be carried out in all records of a table. With

```
do { statements }
```

the program statements is executed for all records in the current table.

As in the table function, it is possible to precede the do-statement with a table name. By doing this, calculations are carried out in the table in question. As in the table functions, you may also use the scope operator here.

#### Example:

```
subjects.do { receivedFromMe = :myAmount ; }
```

equates, in all records of the subjects table, the variable receivedFromMe with the variable myAmount in the current record.<sup>1</sup>

#### **3.4.11 Loops**

In connection with arrays, one would like to be able to carry out calculations on all array elements. To this end one may use **iterators**. An iterator creates a new small table containing one variable. When a table function or a do-statement is applied to such an iterator, this corresponds to a loop over the values contained in the table.

An iterator has the syntax:

This syntax defines a table. This table has a variable varname and records for which the variable varname is filled with, in the last case, for instance, the values x, x+d, ... y.

#### Example:

```
squareSum = iterator( i, 1, 5).sum( i*i );
// squareSum = 1+4+9+16+25 = 55
```

#### 3.4.12 Creation of New Records in the Program

In the contracts table, new records can be added with the "new" statement. The syntax is

```
table.new{ statements }
```

The contracts table is the only table where new records may be entered, as the other tables have fixed records. The statements in curly brackets can contain initializations of the record. The following example is taken from the program of the subjects table of a certain stage. When a user reaches this stage, a record is added in the contracts table. In this record, the variables Seller, Buyer and Price are fixed. (The value Buyer is the Subject ID of the subject for which the program is run.)

```
contracts.new {
    Seller = -1;
    Buyer = :Subject;
    Price = 25;
}
```

#### 3.5 Subject and Period-specific Parameters

You can define variables in subject or period-specific terms by using the variables Period or Subject.

```
if ( Period== 1 & Subject ==1 ) {
     p=11;
}
if ( Period== 1 & Subject ==2 ) {
     p=12;
}
...
```

There is also a parameter table with which the subject-specific variables may be managed quite easily. In this table periods are given in the lines and subjects in the columns.

- 35 -

<sup>&</sup>lt;sup>1</sup> This program serves for illustration purposes only. If the program is executed, the variable receivedFromMe will have the value of myAmount of the last subject.

9	Untitled Treatment 1:2								
Е									
l		81	S 2	83	84				
П		2	1	2	1				
ш	Trial 1								
Г		1	2	2	1				
ш	1								
П		2	1	2	1				
ш	2								
П		1	2	2	1				
ш	3								

The subjects are named S1, etc. and the periods are numbered. Trial periods are preceded by the term "trial". The role parameters define whatever remains the same (with regard to role) for any one subject within one treatment. These parameters are entered or changed in the space marked T1. The fields with the small numbers in the upper left corners contain the **specific parameters** (specific for subjects and periods). The small number is the group number. The period parameters on the left side define whatever holds for all subjects in any given period.

Role parameters, period parameters and specific parameters can be viewed and changed with the command "Info..." from the menu "Treatment" or by double-clicking the field in question. Parameters may be copied either with Copy/Paste or by selecting and then pulling over from one field to another.

At the beginning of a period, the database is set up as follows:

- 1. Setting of standard variables
- 2. Running of programs in background
- 3. Running of subject program (in current period) in the subjects table
- 4. Running of role program in subjects table
- 5. Running of period program in globals table
- 6. Running of programs of first stage

Variables inserted in the parameter table need to have been predefined in a program in the background.

Example: Let us suppose that there are different types of subjects, e.g., type 0 and type 1. In this case you first define a default value for the variable in the background; e.g., "type=0;". Then you double-click in the fields in which you wish to change the value. You then enter the changed value as a program.

When a variable takes on many values, it becomes impractical to double-click the field in question and to adapt the program for every value. Instead you can set up a table with the variable values in a word

processing program and import it from the "Treatment" menu with the command "Import Variable

Table...". This command is only available if the parameter table is the active window.

3.5.1 **Period Parameters** 

**Prompt:** If a text is entered here, it is displayed on the server before the start of the period. The

treatment will only continue when OK is clicked on the server.

**Program:** This program is run in the globals table after the programs in the background, the specific

parameter program and the role program have been run.

3.5.2 **Role Parameters** 

Name: of no consequence

**Preferred work station:** At the start of the session this role is assigned to the subject whose machine has

announced itself under this name. This option is only of significance for the first treatment of a session.

**Program:** This program is run in the subjects table, after the programs in the background and the specific

parameter program have been run.

3.5.3 **Specific Parameters** 

**Group:** Group number of the subject in this period.

Name: of no consequence

**Program:** this program is run in the subjects table after the programs in the background have been run.

3.5.4 **Group Matching** 

Group matching is entered directly in the specific parameters and not by means of a variable.

Furthermore the menu "Treatment" contains an item matching where the groups can be set up in partner

or stranger designs. It is also possible to redefine the variable Group within a program, because, at the

creation of the database, the standard variables are set first of all. The variable Group is one of these

standard variables.

3.6 Layout

The screen constitutes the whole region that is visible to the subject. Boxes are rectangular parts of the

screen. There are different kinds of boxes. We will present them starting with chapter 3.6.4. Some kinds

of boxes can contain items and buttons which will be explained in chapter 3.6.2 and 3.6.3.

3.6.1 Screen

Only one option is available for screens: "Background screen is being used". If this field is marked, the

windows of the background are taken over and placed first.

3.6.2 **Items** 

Items are used for displaying and reading in variables. An item contains the name of the variable and

information on how to display it. We call an item "input item", if the checkbox "Input" is checked. In

this case subjects have to make an entry. We call an item "output item", if the checkbox "Input" is not

checked. In this case a variable will be displayed. Items can be contained in standard boxes, grid boxes,

history boxes and all contract boxes.

**Label:** Name of the variable displayed to the subjects. This text may also be empty.

Variable: Variable that is displayed or read in. In the case of input items only variables or array variable

with a number as index are allowed. (Array variable may not contain spaces and the index must be

written with the number of digits given by the increment of the array.) For output items any expression is

allowed. If the variable is empty, the item only displays the label. If the variable contains only an

underscore ("\_"), the layout is the same, as when a variable has to be entered, except that no value for the

variable is displayed.

Layout: If a number, a variable or an expression is entered here, the value of that field determines how

the variable is rounded when it is displayed. If for instance 0.2 is entered in layout, and the variable

contains the value 12.34, then 12.4 is displayed. If it is an input item, then the number is used to check

whether the number entered by the subject is a multiple of this number.

An exclamation mark makes further formatting options available:

```
!text: value1 = label1 ; value2 = label2 ; ...
```

In the case of an output item the text <code>labeli</code> appears when the variable has the value <code>valuei</code>. In the case of an input item it is the other way round, i.e. if a <code>texti</code> is entered, the variable gets the value <code>valuei</code>. If text different from all <code>textis</code> is entered, an error message appears.

```
!radio: value1 = label1 ; value2 = label2 ;...
```

Labeled radio buttons appear. The value of the variables again corresponds to the values, the labels of the buttons are the corresponding labels.

```
!radioline: leftvalue = leftlabel; rightvalue = rightlabel; number
```

A row of radio buttons appears. They may be accompanied by labels to the left and to the right. Leftvalue is the value of the button at the left margin. Rightvalue is the value of the button at the right margin. The value of number determined the number of buttons.

```
!slider: leftvalue = leftlabel; rightvalue = rightlabel; number
```

A slider appears. The slider may be accompanied by a text on the left and on the right. Leftvalue is the value of the variables at the left margin. Rightvalue is the value of the variables at the right margin. The value of number determines the resolution, i.e., the number of possible slider positions.

```
!scrollbar: leftvalue = leftlabel; rightvalue = rightlabel; number
```

A scrollbar appears. The scrollbar may be accompanied by a text on the left and on the right. Leftvalue is the value of the variables at the left margin. Rightvalue is the value of the variables at the right margin. The number sets the resolution, i.e., the number of possible scrollbar positions.

```
!button: value1 = label1 ; value2 = label2 ;...
```

Buttons appear, arranged similarly to radio buttons. By clicking a button the variable is set to the corresponding value. Besides this, each button also has the same effect as the OK button. Hence, at most one variable per screen can have the button option. For output items, the button options yields the same display as the text option.

```
!checkbox: 1 = text;
```

A checkbox appears. This checkbox is labeled with text. If the box is marked, this corresponds to the value 1. If it is not marked, this corresponds to the value 0.

**Input:** If this checkbox is marked, the subject has to enter a variable. Otherwise the variable is displayed.

**Show Default:** This option appears only with input variables. Normally an input field is empty in the beginning. If this option is chosen, the input field is filled with the current value from the database.

**Empty allowed:** If this option is marked, the variable may be left empty.

**Minimum:** The minimum permitted value for input variables.

**Maximum:** The maximum permitted value for input variables

## **Examples of item layouts**

Layout	input variable	output variable
2	6	6
!radio: 1 = "86.8"; 24 = "102.8";	<b>⊙</b> 86.8 <b>○</b> 102.8	© 86.8 © 102.8
!radioline: 0="zero";5="five"; 6;	zero CCCCC five	zero CCCCC five
!slider: 0 ="A"; 100= "B"; 101;	А ; В	А
!scrollbar: 0="L";100= "R";101;	LIPR	L F R
!checkbox:1="check me";		✓ check me
!text: 1= "one"; 2 = "two"; 3 = "three"; 4 = "four"; 5 = "five"; 6 = "six"; 7 = "seven"; 8 = "eight"; 9 = "nine"; 10 = "ten";	seven	seven
!button: 1 = "accept"; 0 = "reject";	accept reject	accept

#### **3.6.3 Buttons**

Input is concluded by clicking a button. The labeling of the button can be defined. If the text consists of an underscore ("\_"), no button appears. However, space for a button is kept free. If the button label contains the "&"-character, the next character is underlined. If you want to insert an "&"-character into the button label, you have to duplicate it.

Buttons can contain checkers and programs. Checkers are used for checking subjects' entries. Programs in buttons are executed when the button is clicked and when all checks are OK.

Standard boxes, grid boxes and all contract boxes can contain buttons.

#### 3.6.4 General Remarks on Boxes

The subjects' screens are made up of boxes. These are rectangular parts of the screens. The boxes are positioned within the "remaining box" in the order in which they appear in the stage tree. In the beginning, the "remaining box" constitutes the whole screen. Later the "remaining box" is adjusted according to the definition of the boxes. The dialog for a box is explained below.

Name	any box with Fra	me
Width [p/%]		Distance to the margin [p/%]
Height [p/%]		
	ment of the naining box	☐ left ☐ top ☐ right ☐ bottom

**Name:** Name of the box. Used for documentation only.

With frame: Is there a frame around the box or not.

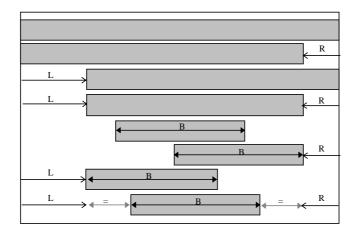
**Width:** Width of the remaining box in pixels [p] or percent [%]; optional.

**Height:** Height of the remaining box in pixels or in percent; optional.

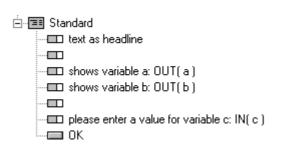
**Distance to the margin:** Indicates how far away the margin is from the margin of the remaining box; in pixels or in percent of the remaining box; optional.

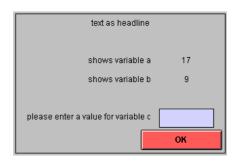
Adjustment of the remaining box: Where is the remaining box cut-off, if at all?

Width, height and distance to the margin are optional. Depending on which fields have been filled, the box is placed within the remaining box. The graph below shows all cases for the fields width (W), distance to left margin (L) and distance to right margin (R).



#### 3.6.5 Standard Box





Variables of the subjects table may be displayed or entered. The items are displayed from top to bottom. The window is divided into a label column (left) and a variable column (right). The variables are always displayed in the variable column. The label always appears in the left column if the variable is defined or if it consists only of an underscore ("\_"). If the variable is empty, the label is regarded as title and written centered over the whole window. If the label consists of more than one line, it is aligned to the left.

**Button position:** Buttons can be placed into corners or centered at a margin.

**Button arrangement:** If more than one button is defined in this box, the further buttons can be placed either in rows or in columns. If they are placed "in rows", z-Tree first tries to fill a row starting from the corner given by the button position. If the first row is full, a next row is filled starting from the same corner.

#### 3.6.6 Header Box

Period 1 of 1 Remaining time 5

Period number and time are displayed in a header box. The following options can be adjusted:

**Show current period number:** Period number is displayed or not.

Show total number of periods: Total number of Periods is displayed or not.

**Name of "period":** Name in front of the period number. "Period" is used as standard. However, "Trading day" or "Round", for example, are also possible.

Term for "out of": At "3 out of 10" (Periods).

**Prefix for trial periods:** "Trial" is standard.

**Display time:** Is time being displayed to the subjects? If not, not even a time message is generated.

**Term for "Remaining time":** "Remaining time [sec]:" is standard. If this field is empty, no time is displayed. In this way you can do without displaying the time and yet ask the subjects to continue when time has run out.

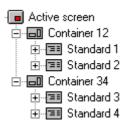
"Please make your decision now": Text that appears when the allotted time has run out.

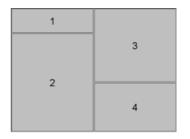
## **3.6.7** Help Box

Help text label
A help text can explain what the subjects have to do and how a particular action can be carried out.

Box with a label that contains a text. A label and a help text can be entered. If the help text is too long to appear in the area reserved for the help box, a scrollbar appears.

#### 3.6.8 Container Box





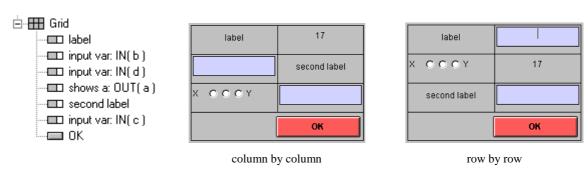
If you want to directly place the boxes in the figure above, it becomes necessary to choose the appropriate sizes. By defining container boxes you can easily make sure that the boxes always match one another. Container boxes are boxes that can contain other boxes.

Fischbacher/ z-Tree/ Mar. 29, 99 - 43 -

In the example we define six boxes, the boxes 1,2,3, and 4, as well as two container boxes, 12 and 34. We define the width in the container box 12 and cut-off the remaining box at left. Now container box 34 fills the region on the right. In boxes 1 and 3 we define the heights and cut the remaining box off at the top. In this way we have not entered a size specification at more than one place, and even if we should change something, the appearance of the screen remains the same.

If we move a box from one place to another inside the stage tree, it is moved *after* the box where the mouse button is released. By moving a box over the *icon* of a container box it is moved *into* the container box and positioned at its beginning.

## **3.6.9 Grid Box**



The items are displayed in a table. Each item belongs to one field. If the item contains a variable, this variable is displayed. During checks the label is used to convey to the subject in which field he or she has made a mistake. The label is only displayed if the item does not contain a variable.

The individual options are:

Num. rows: Number of rows of the grid, including label row if it exists.

Num. columns: Number of columns of the grid, including label column if it exists.

**Input row-by-row:** The items are filled into the grid row-by-row. By pressing the tab key the subject can make the entry row-by-row.

**Input column-by-column:** The items are filled into the grid column-by-column. By pressing the tab key the subject can make the entry column-by-column.

**First row contains labels:** First row is label row. Only used for "Separate labels by lines".

**Height [of the first line]:** Height of the first line in % of the other lines.

First column contains labels: First column is a label column. Only used for "Separate labels by lines".

Width [of the first column]: Width of the first column in % of the other columns.

**Separate labels by lines:** If the first row is a label row, a dividing line is drawn between the first and the second row. The same procedure applies for the first column.

**Separate rows by lines:** Lines are drawn between the rows.

**Separate columns by lines:** Lines are drawn between the columns.

**Button position:** Buttons can be placed into corners or centered at a margin.

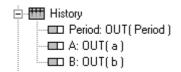
**Button arrangement:** If more than one button is defined in this box the further buttons can be placed either in rows or in columns. If they are placed in rows, z-Tree first tries to fill a row starting from the corner given by the button position. If the first row is full, a next row is filled starting from the same corner.

#### 3.6.10 Calculator Button Box



Defines a button by which the Windows NT Calculator can be called up. It serves as a substitute for subjects who have forgotten their calculator.

## 3.6.11 History Box



A	В	
36	12	•
45	24	
51	21	
37	11	
63	28	_
	45 51 37	36 12 45 24 51 21 37 11

The results from previous periods are listed in a table. A label row contains the labels. If the table is too long, a scrollbar appears. The current period appears at the end of the table. The scrollbar is adjusted in such a way as to make this line visible at the beginning.

## 3.6.12 Text-formatting with RTF

In labels of items, in help boxes and in message boxes texts formatted with RTF can also be entered. The RTF format begins with "{\rtf " (with a blank space at the end) and ends with "}". In between is the text

to which formatting instructions can be added. Formatting instructions begin with "\" and end with a blank space. If a formatting is only supposed to apply to a certain range, you can place this range in curly brackets.

Not the whole RTF is supported. The most important formatting instructions that are supported are:

\tab tabulator

\par new paragraph

\line new line \bullet thick dot•

\ql aligned to left

\qr aligned to right

\qc centered

\b bold

\b0 not bold

\i italic

\i0 not italic

\sub small and inferior numbers (index)

\super small and superior numbers (exponent)

\strike crossed through

\ul underline

\ul0 do not underline

Text color. n is the index of the color table which is defined by colortbl. See example.

\fsn Font size in units of half a dot. The font size must be explicitly given, otherwise it is larger

(24) than usual in z-Leaf.

### **Examples**

normal font size, **bold**, no longer bold {\rtf \fs18 normal font size, \b bold,

\b0 no longer bold }

Text italic no longer italic {\rtf \fs18 Text {\i italic} no longer

new line italic \par new line}

One word in ochre, rest in black {\rtf

{\colortbl;\red0\green\blue0;\red255\gree n100 \blue0;} \fs18 One word in \cf2

```
ochre\cf0 , the rest in black.}
```

For more complex operations it is best to format the text in a word processor and then export it as RTF. However, if you make the RTF code by hand, it will be shorter and easier to read.

#### 3.6.13 Insertion of Variables into the Text

Variables can also be inserted in the label and in the format field of items, as well as in help and message boxes. Thanks to this you can display a formula with the values inserted.

Example: We want to write "income = 23.5 points" where 23.5 is the value of the variable Profit. As the variable does not stand alone on the right hand side, it needs to be integrated into the text. The example above is entered in the following manner:

```
<>income = < Profit | 0.1 > points
```

The string "<>" at the beginning of the text indicates that there might be variables in the text. The variable and its layout appear in square brackets, separated by a vertical line. In our example, 0.1 is the layout. The value of profit is thereby given to one decimal place.

"!text:" is also allowed as layout. This works in exactly the same way as with output items. Example:

```
<>Your income is < profit | !text: 0="small"; 80 = "large";>.
```

Depending on whether the profit is nearer to 0 or to 80, either "Your income is small" or "Your income is large" is displayed.

Variables can also be inserted into the strings of the "text" option. There you need not begin the string with a second "<>". If you want to define a layout in which, for negative values, a text is written but, for positive values, the number (to two decimal places), you write

```
<>!text: -1 = "negative"; 1 = "<Profit 0.01>";
```

This layout has the disadvantage that you need also to write the name of the variable into the format field. However, you can also omit the name of the variable. In this case the variable of the *item* is automatically shown:

```
<>!text: -1 = "negative"; 1 = "< |0.01>";
```

Warning: Be careful when you use this option for items that can change their value (in chapter 3.9 on market games we explain treatments in which that happens). The width of an item is calculated at the start of a session and it is not modified afterwards. So, if the value of a variable changes from 1 to 20, only 2 might be displayed. You are responsible to avoid it. You can choose a sufficiently wide first value or place the item into a box with other items that are wider.

This also works with variables inserted in the text:

```
<>Your profit is < Profit | !text: -1 = "negative"; 1 = "< |0.01>">
```

If you wish to insert the symbol < in the text, the symbol should be duplicated so that it is not confused with a variable expression. Note that variables in labels are never updated

The insertion of variables is carried out *before* the interpretation of RTF. In other words, conditional formatting is possible: When the variable BOLD is 1, "hallo" should be shown in boldface but otherwise normally:

```
<> {\rtf <BOLD |!text: 0="";1="\b ";>hallo}
```

# 3.7 Checking of Subjects' Entries

#### 3.7.1 Item Checks

First of all we always check if each field is filled. The subject needs to enter a value into each field, unless it is explicitly stated in the item that the field may be empty.

A minimum value and a maximum value need to be entered in the definition of items. If a number has to be entered, the resolution is also given in the format. The subjects' entries must lie between the minimum and the maximum value and it must be a multiple of the resolution. If this is not the case, the entry is not accepted. If the label is not empty, the subject receives a message telling him/her which error has occurred in which field. If the label is empty, it bleeps and the field containing the error is marked. No message appears.

## 3.7.2 Checker

One can create a Checker for carrying out more complicated checks. This checker is placed inside a button.

You have the following options:

**Condition:** logical expression.

**Message:** This message appears when the condition is not met.

If, for example, the sum of two values a and b should not exceed a given value M, you write for the condition

```
a + b < = M /* without semicolon */
```

and for the message:

 $\rightarrow$  The sum of a and b must not be more than  $\rightarrow$  M|1>.

As you can see it is also possible to integrate variables into messages.

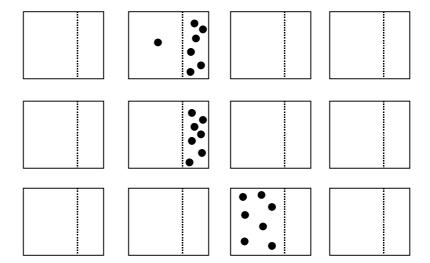
#### 3.8 **Course of the Treatment**

#### 3.8.1 **Stages**

In each period the subjects go through all stages, one stage after the other. In each stage they arrive at the active state of the stage. In this state the subjects see the active screen of this stage. In the clients' window the active state of a stage is designated as "\*\*\* stage name \*\*\*". The active screen is left by means of an OK button or a time-out. When this happens, the waiting screen appears and the subject arrives at the waiting state of that stage which is designated as "- stage name -". Whether the subjects can begin with the next stage depends on how the option "Start" is set in the next stage. The following graphs show the various possibilities. The rectangles are stages. The black spots are subjects in a particular state. If these spots are located to the left of the dashed line, this means that the subjects are in the active state. Otherwise the subjects are in the waiting state. In the first case a sequence is shown. In the following cases the situation that results when half the subjects reach the third stage is shown.

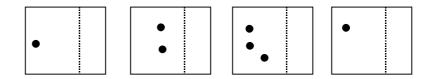
#### wait for all; start together = | =

All have to wait until all subjects have reached the waiting state of the previous stage. Once this has been achieved all subjects enter this stage simultaneously.



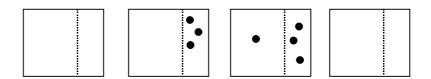
#### individual start -=

As soon as a subject has reached the waiting state of the previous stage, he or she can begin this stage, irrespective of what the others are doing at the time.



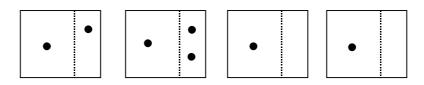
## wait for all, then start separately = | -

You first wait until all subjects have reached the waiting state of the previous stage. Once this is achieved, one per group may begin this stage. As soon as one subject in the group has finished the stage, the next may begin it.



## as soon as possible, start separately - | -

As soon as a subject has reached the waiting state of the previous stage, he or she may begin this stage, as long as no one from his or her group is already in it.



In the last two cases, two things need to be added. "Start separately" means that there is never more than one subject *from each group* in the active state of a particular stage. Group membership is defined by the value of the variable Group. The order in which the subjects enter the next stage is determined by the variable **Priority**. Of the subjects that can continue, the subject with the smallest value of Priority is chosen. If this variable does not exist, i.e., if this variable is not fixed in the entire treatment, a subject is chosen randomly.

#### 3.8.2 Time

For each stage in the time-out field, the time available to the subjects per stage is fixed. The expression entered is calculated in the globals table. If different stages should have the same time-out times, define a variable in the globals table and enter the name of this variable in the time-out field.

After the time set for a stage has run out, the continuation of the game depends on whether entries have to be made in this stage or not. If the subject does not have to make any entries on this screen, the game continues when the time set for the stage has run out. The subject arrives at the waiting state of the stage and may go on from there. If entries have to be made, the time displayed is simply a guideline. When the time has run out, a message saying "Please make your decision now" appears. However, the game does not continue until the entries are made. If you enter a negative number, no time-out will occur. The subjects have an unlimited amount of time at their disposal. If the time-out is set to zero, the subjects will arrive at the waiting screen straight away.

## 3.8.3 Leaving out Stages

Example: In the ultimatum game, the proposers move first, making a decision on their offers. After this the responders decide whether or not they want to accept this proposal or not. Therefore we define a Proposer Decision stage and a Responder Decision stage. Only the proposers go through the former, and only the responders go through the latter. To this end we write the following line in a program of the subjects table in the stage Proposer Decision:

```
Participate = if ( type == Proposer, 1, 0);
```

This line sets the variable Participate. If it is zero, the subject does not go through this stage. He or she directly reaches the waiting state if that stage, without the display changing. To the subject it looks as if he or she were still in a previous stage. In this way you can build in pure calculation stages not seen by any of the subjects. No screens need to be defined in such a calculation stage.

Fischbacher/ z-Tree/ Mar. 29, 99 - 51 -

## 3.8.4 Treatments of Indefinite Length

If you want to apply a random stopping rule, the treatment has no definite length. This can be implemented in z-Tree with the variable **RepeatTreatment**. You define a one period treatment. At the end of the period you decide whether to continue the treatment or to stop. This decision is written into the variable RepeatTreatment in the globals table. If the variable is not defined, the treatment is terminated. This is the normal case. If you define the variable RepeatTreatment, it depends on the value of the variable whether the treatment is terminated or whether it is repeated once again. If the value is smaller or equal to zero, the treatment stops. If the value is greater than zero, the treatment is repeated.

If the treatment is repeated then the period counter is incremented as if there was one treatment. This is implemented by modifying the number of (trial) periods. So if you repeat a treatment with three periods, the first treatment has 3 Periods and no trial periods. In the second run of the treatment, the treatment's number of periods is changed into 6 and the number of trial periods is changed to -3.

## 3.9 Market Experiments

## 3.9.1 Concepts

With the concepts described so far you can program any kind of treatment in which the subjects make only one decision in every stage. In markets, however, people can make offers, revise them and accept offers from other people. Such offers are stored as records in a new table, namely, in the contracts table. A record in the contracts table is called a contract. A contract of this kind should not be conceived of as a bilateral, signed contract, but rather as an offer or as a contract as yet signed by one side only. Furthermore, there are boxes in which the subjects can enter new contracts, boxes in which they can accept contracts and boxes in which contracts can be viewed and modified:

- Contract creation boxes are used by subjects to *create* contracts..
- Contract list boxes are used to select contracts.
- Contract grid boxes are used to view and *modify* a part of the contracts table.

Normally, a stage is automatically concluded when a subject has made his/her entries. He or she then reaches the waiting screen and from there may proceed to further stages, depending on what "Start" option was chosen. Auctions are not concluded by individual entries. They end after all possible contracts have been realized or when the allotted time has run out. Therefore, a separate start type has been defined for auctions.

Fischbacher/ z-Tree/ Mar. 29, 99 - 52 -

In auctions, certain calculations must be carried out immediately after a subject's entry. When a subject accepts an offer in a double auction, all other open offers from this subject need to be deactivated. This cannot wait until the next stage begins. Therefore, in addition to the programs at the beginning of the stages, programs also need to be placed in buttons. These programs are then literally run at the press of a button.

Before the clicking of a button is accepted, all checks located in the button are carried out. The entry is only accepted and the programs are only run if none of these checks fails.

## 3.9.2 Auction Stages

We did not explain the "Start" options "Auction" and "Auction Continuation" in the chapter "Course of the treatment". This will be done now: An auction stage is started and ended simultaneously with all following auction continuation stages. Before the stages start, all programs of these stages are executed. This is necessary in order to determine which subjects will participate in which stages.

Example: Though buyers and sellers in a double auction take part in the same auction they are shown different screens. To this end we define two stages: The first, an auction stage for the sellers and the second an auction continuation stage for the buyers. These two stages are started simultaneously, the first for the sellers, the second for the buyers. For that we insert the following program lines into the two stages:

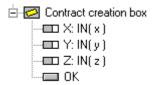
Generally, auction stages are concluded when the time has run out. There are two exceptions, however:

- If the variable AuctionStop is set to 1 in the globals table, the auction is terminated immediately. This is useful, for example, when all possible transactions are exhausted.
- If the variable AuctionNoStop is set to 1 in the globals table, the auction is not concluded even if the time is up.

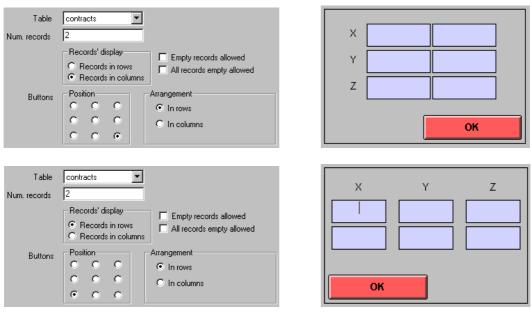
#### 3.9.3 Contract Creation Box

We show in an example how a subject can enter two new contracts in a contract creation box. A contract should contain the values x, y, and z. Therefore, we put three input items, x, y and z as well as an OK button into the contract creation box. In the stage tree the contract creation box looks like this:

Fischbacher/ z-Tree/ Mar. 29, 99 - 53 -



How the contract creation box looks like in z-Leaf can be modified in the box'dialog:



option in z-Tree box in z-Leaf

When you click the button, z-Tree checks if all entries have been made. Next the checkers in the button are checked, and if these are OK, two new contracts are added to the contracts table. Finally, the programs of the button are run. If we are in an auction stage, further entries can be made, otherwise the stage is concluded. You may specify the following options:

**Table:** All contract boxes can be linked to any one of the tables. This paves the way for future extensions. At the moment only the contracts table is feasible.

**Num.** records: Number of records that have to be entered. Only one record is feasible in an auction stage. You can also enter a variable from the subjects table here, for example a constant, numContracts. In this way you can define treatments in which subjects have to enter different numbers of contracts.

**Records' display:** The records can be displayed in columns or in rows. In the above example, we show the two options.

**Empty records allowed:** If this option is marked, the subject can fill out less records than are displayed. For each record, however, either all fields must be filled or none. In this way you can allow the subjects to offer less than the maximum possible number of contracts.

Fischbacher/ z-Tree/ Mar. 29, 99 - 54 -

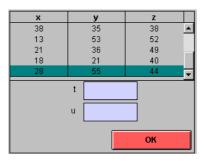
All records empty allowed: All records can be left empty which means that no record is created.

**Button position:** Position of the button (the buttons) in the box.

**Button arrangement:** When the first button lies in the lower right hand corner, horizontal means that further buttons are added on the left, until one has to begin a second row. The first button in the second row lies above the first button. The other cases are analogous.

#### 3.9.4 Contract List Box





In a contract list box a selection of records is listed. Which records appear as well as the order of the records in the list can be set. Contract list boxes are continually updated. Therefore, whenever a subject effects a change in the contracts table, this is immediately registered and displayed in the contract list box.

The contract list box is composed as follows: The values of the output items are entered in one list, each record on a separate row. If there is not enough space for all contracts in the list, a scrollbar appears. If there are input items, these appear at the bottom of the list in the same layout as in a standard box.

When a contract list box contains a button, the subject can select a record and fill in the input items. When the button is clicked and the checks are OK, the variables of the input items are set and the programs are run for the selected record.

It is useful for the button of a contract list box to contain a program. Only with the help of a program can the contracts table be changed in such a way as to show that the contract has been selected and who has selected it. Such a program could be the following:

Accepter is a variable of the selected records. ":Subject" is the variable Subject in its own record in the subjects table. (For details see chapter on programs below).

On the options:

**Table:** All contract boxes can be linked to any one table. This paves the way for future extension. At the moment, however, only the contracts table is feasible here.

Owner var.: Owner variable is a variable of the contracts table. The records for which the value of this variable equals the variable Subject are shown blue for each subject. Example: Seller is the variable in the contracts table in which the subject number of the seller is written. When all sales offers are displayed on the auction screens of the sellers, one can show one's "own" contracts in blue. These are the contracts for which the variable seller has the value Subject.

**Condition:** Condition for the records displayed.

**Sorting:** Expressions according to which sorting is carried out, separated by semi-colons. When sorting is to be undertaken in descending order, the expression is preceded by a minus sign.

Example: a; -b; -c; sorts the columns in the following way:

a	1	3	6	6	6	7	8	8	8	10	10
b	9	2	7	5	5	4	4	4	2	6	3
С	7	2	4	5	3	2	8	4	5	3	3

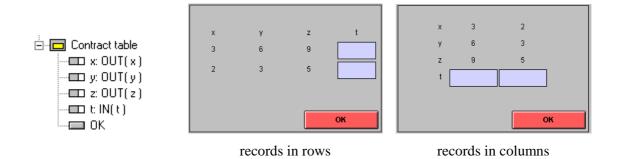
**Scrolling:** After each change to the list scrolling to the beginning or to the end takes place.

Mark best foreign contract: If no other contract is selected, the contract in the direction of scrolling is automatically selected, i.e., if scrolling has taken place towards the end, the last foreign contract is chosen. A contract is considered as an *own* contract if the owner variable has the value of the subject ID of this subject. All other contracts are foreign contracts. If there is no owner variable, all contracts are considered as foreign.

**Button position:** Position of button (buttons) inside the box.

**Button arrangement:** When the first button lies in the lower right hand corner, horizontal means that further buttons are added at left until one has to begin a second row. The other cases are analogous.

#### 3.9.5 Contract Grid Box



Fischbacher/ z-Tree/ Mar. 29, 99 - 56 -

The selected records are shown in a table. Values of the output items and input fields for the input items appear in this table. There must be space for all contracts in the table.

This box works similarly to the standard box, except that variables from the contracts table, not from the subjects table can be viewed and changed in it. The images above show contract grid boxes. To the left, the records are in the rows and, to the right, in the columns.

On the options:

**Table:** All contract boxes can be linked to any one table. This paves the way for future extensions of z-Tree. At the moment, however, only the contracts table is feasible here.

**Owner variable:** The owner variable is a variable in the contracts table. For each subject, the records for which this variable has the same value as the variable Subject are shown in blue.

**Condition:** Condition for the records that are displayed. The expression in this field is evaluated for each record. Those records are shown for which the value of the condition is TRUE.

**Sorting:** Expressions according to which sorting is carried out, separated by a semi-colon. If sorting is to be carried out in descending order, the expression is preceded by a minus.

**Records' display:** As in the contract creation box.

**Empty records allowed:** If this option is marked, the subject can fill out less records than are displayed. For each record, however, either all fields must be filled or none.

All records empty allowed: All records can be left empty.

**Button Position:** Position of button (buttons) inside the box.

**Button Arrangement:** When the first button lies in the lower right hand corner, horizontal means that further buttons are added at left until one has to begin a second row. The other cases are analogous.

### **3.9.6 Buttons**

A button always lies inside a box. It can contain checkers and programs.

**Name:** Label of the button. If this consists only of an Underscore ("\_"), no button appears. However, the space for the button is kept free.

**Type:** OK buttons conclude entry and transfer the data to the server. In the server, the checkers are run and, if successful, the data is integrated in the database and the programs are run. The cancel button does

not transfer data but checks and program call-ups take place regardless. Furthermore, cancel buttons do not conclude stages.

**Clear Entry after OK:** After the button has been pressed and the input has passed all checks, the input fields in the box where this button is located are cleared. This is particularly useful for buttons in contract boxes where the stage is not terminated by the button (see market experiments)

### 3.9.7 Checker

Checkers are used for more complicated input checks. In contract creation boxes, checkers are run for *each new* record. In contract list boxes, they are run for the *selected* record. In all other boxes, the check is carried out in the subjects table.

If a condition is not met, a message appears on the subject's screen. If there is a message box, the message appears in this box. Otherwise, a dialog appears that needs to be confirmed by the subject.

## 3.9.8 Message Box

Box in which the messages generated by the checkers are displayed. As soon as an entry is made for which all checkers were successful, the text in the message box is deleted. The "first message" appears when the screen is set up.

## 3.9.9 Programs

Programs are used for carrying out calculations and for managing users' entries. For which Records a program is run depends on the context. The following table lists the different possibilities.

Fischbacher/ z-Tree/ Mar. 29, 99 - 58 -

table	Program lies in	Owner var.	Cond.	Records for which the program is run	Records accessible through scope operator
globals			without	Record of globals table	-
			with	If cond. is met, record of globals table	-
subjects			without	All records of subjects table	globals
			with	Records of subjects table for which cond. is met	globals
summary			without	Record of current period	globals
			with	Record of current period, if cond. is met	globals
contracts	stage	without	without	All records of contracts table	globals
		without	with	All records of contracts table that meet cond.	globals
		with	without	All records of contracts table. The records are worked through subject for subject. The records of a subject are those records for which the Owner variable equals the Subjects variable.	Record of owner in subjects table, globals
		with	with	Same as above, except that the record of cond. also needs to be met	Record of owner in subjects table, globals
contracts	standard box, grid box, contract grid box		without	All records of contracts table	Record in subjects table of subject who has clicked the button globals
			with	All records of contracts table that meet cond.	ditto
contracts	contract creation box, contract list box		without	All new records of contracts table in the case of the contract creation box, and the selected record in the case of the contract list box. (To run the program for all records cond. has to be set to TRUE)	ditto
			with	All records of contracts table that meet cond.	ditto

# 3.10 Examples

## 3.10.1 Public Goods Game

The subjects are matched into groups of four people. Each subject selects an amount g between 0 and 20 as his or her contribution to a public good. When the sum of the contributions of all subjects of one's own group equals s, the profit is 20-g+1.6/4\*s.

The treatment consists of two stages: The entry of the contribution and the profit display.

First of all, we define certain parameters. To this end, we add a program in the background for the subjects table that contains the following program:

```
eff = 1.6; // social return of a contribution to the public good M = 20; // endowment
```

With this we define constants which we can now use in the stages. It is useful to define parameters for a treatment at this point. If we wish to run a similar treatment in which only these parameters are different, we can do this here centrally.

Furthermore, we add a program that is run in the globals table.

```
TimeoutEntry = 45;
TimeoutDisplay = 30;
```

Here we define constants for the amount of time that the subjects have at their disposal for entry and profit display. In the first period we give them more time. For this we go into the parameter table. There we open the period parameter of the first period and write into the program:

```
TimeoutEntry = 90;
```

Now we define the stage "contribution entry". In order to do this, we create a new stage: We click the background and select the menu command "New stage". A dialog, to be filled with information, appears. For the name of the stage we enter "contribution entry". The name is used as the name of the state the subject is in. Hence, the stage name must be unique. (You can guarantee the uniqueness of the stage name by using the command "Make Stage Names Unique" in the Menu "Treatment/Utilities".) As the stage is only started when all subjects are ready and then all start simultaneously, we select for start the option "Wait for all; Start together".

The time-out is set:

```
TimeoutEntry
```

This stage has no program. We only make the entries.

The active screen consists of the header box, that was already defined in the background, a help box and a standard box. In the help box, we enter the help text: "Please choose your contribution. Click OK after making your entry". Furthermore, we may adopt the settings just as they are proposed by the program. In the standard box, we can adopt all settings. We only need to add the items and the button. We wish to display the amount M that is to be divided. After this the subject is supposed to decide how much to contribute. We call this contribution g.

The order of the boxes is important. The help box appears first. It is automatically positioned in the lower part of the screen and the remaining screen is cut-off so that the standard box can occupy the middle region of the screen.

The waiting screen consists only of the waiting screen from the background.

The "Profit display" stage has the same settings as the contribution entry stage. Here, we first define the profit function in a program.

```
oeff = sum ( same( Group), g);
n = count( same ( Group ));
Profit = M - g + eff * oeff / n;
```

The variable oeff is the sum of the contributions to the public good. Naturally, the sum only extends over the members of the group. The number of members in a group is generally constant. In order to make the treatment as universal as possible, we re-count the members at this stage and arrive at the number of group members in n. Profit means the profit in this period.

The active screen again consists of a header box, a help box and a standard box. In the standard box we create items for the information we wish to display to the subjects. First, as a reminder, we display the contribution g, then the sum of the contributions (oeff) and last the income (Profit). If we wish to structure the display a little, we can build in empty rows. Finally, we add an OK button labeled "continue".

Now the treatment is defined. We can now program variants: Better help texts, other information conditions, another profit function, individual endowments, etc. Furthermore, we can carry out calculations in the summary table so that we can better follow the experiment as experimenters.

## 3.10.2 Ultimatum Game

Player 1 (the proposer) can propose a division for an amount of 100. Let the proposal be (share1, share2). Player 2 (the responder) can accept or reject this division. If player 2 should accept this proposal, both players receive their share. If player 2 rejects, both receive nothing.

For this treatment we define four stages: The decision of the proposers, the decision of the responders, the profit display of the proposers and the profit display of the responders.

First we define the constant M, the amount to be divided, and the variable type which has the value 1 for the proposer and the value 2 for the responder. The variable type needs to be set in the parameter table

for all subjects. Alternatively, we may also calculate the type in the first stage. To make the programs in the treatment easier to read we define the constants PROPOSER and RESPONDER in the globals table:

```
PROPOSER = 1;
RESPONDER = 2;
```

Only the proposers take part in the first stage, called the proposer entry. We therefore set in the program of this stage:

```
Participate = if ( type == PROPOSER, 1,0);
```

Here, the offer is the only entry that the proposer makes. We call it "offer". In the second stage the responders make their entries. As the offer is only recorded in the individual record of the proposer, we need to transfer the offer of the proposer to the record of the responder. To this end we define a new variable Share:

Only responders go through this stage. This is expressed by:

```
Participate = if ( type == RESPONDER, 1,0);
```

The entry whether the offer is accepted or not is carried out by buttons. For this reason we do not require a button in the standard box. For this layout the following entry needs to be made in the format field of the item of the variable Accept:

```
!button: 1="yes"; 0="no";
```

The exclamation mark signifies that this is not an ordinary input field. "Button" signifies the fact that buttons for making entries appear. "1" or "0" are the values that the variable can have, "yes" or "no", the texts that appear in the buttons.

The profit is calculated in the profit display stage:

```
accept = find ( same( Group) & type == RESPONDER, accept);
Profit = if( accept == 0, 0, share );
```

The profit display takes place separately for proposers and responders, as they each have a different display. In order to achieve this, we place the stages one after the other, first proposers' profit display then the responders'. In the two stages we set the Participate variable in such a way that only the proposers enter the proposers' profit display and only the responders enter the responders' profit display.

Finally, we change the start option in the responders' profit display stage to "individual start". If we omit doing this, the responders only receive the profit display after the proposers have been shown the profit. This creates an unnecessary wait.

#### 3.10.3 One-sided Auction

In this experiment there are sellers and buyers. The sellers can make sales offers, and the buyers can accept or reject them. When a seller's offer is accepted, the offers he or she made up to this point are deleted. Both - buyers and sellers - see the existing offers on the screen.

We implement this market by writing all offers into the contracts table. The following variables exist there:

Seller: Seller's subject ID.

Buyer: Buyer's subject ID. Initially this is -1. This means: "not yet accepted offer". Deleted sellers' offers have -2.

Price: Price offered.

The sellers can make offers in a contract creation box. They enter the price in this box. When a seller makes an offer and clicks the button "offer", a record is created in the contracts table, in which the variable Price is set at the amount entered. The variables Seller and Buyer are set in a program that is placed in the "offer" button.

```
Seller = :Subject; // access the "own" record with the scope operator Buyer = -1;
```

The buyer sees a list with the offers. He or she sees the offers that have not yet been accepted and that were also not yet deactivated. These are the offers that meet the condition "Buyer ==-1". The offers are sorted in descending order according to the prices. Therefore "-Price" must be entered as sorting.

In this list the buyer can click offers and accept an offer with the button "Accept". The variable Buyer needs to be set for the offer to be accepted:

```
Buyer = :Subject;
```

Furthermore, we need to deactivate all of the sellers' offers. We can achieve this with the following program:

Fischbacher/ z-Tree/ Mar. 29, 99 - 63 -

```
contracts.do {
    if ( Seller == :Seller & Buyer == -1 ) {
        Buyer = -2;
    }
}
```

4 Questionnaires

4.1 Overview

Each session ends with a questionnaire. Generally, the address is asked first so that the payment file can

be written. Next, additional questionnaires are inserted and finally a good-bye screen appears which also

displays the profit.

A questionnaire consists of a series of question forms. One question form displays the individual

questions in the same way as the items in the standard box of a treatment. However, the questions do not

appear vertically centered and adjusted to the screen size. Therefore, question forms may be larger than

the screen. If this is the case, a scrollbar appears.

The answers in questionnaires are of no consequence. In particular, no earnings can be made in

questionnaires. Furthermore, the answers are only saved as text.

The question forms are worked through one after the other. When all subjects have answered all question

forms they are in the state "ready" and you can start a further treatment or a further questionnaire. It is

generally impractical to start several questionnaires one after the other because this leads to unnecessary

waits. It is better to integrate all forms into one questionnaire.

The last question form remains on the users' screens until you continue, i.e., until you select a new

treatment or a new questionnaire or until you shut-down the computer. Therefore, the final question form

must not contain a button.

4.2 Profit Display

If you wish to display the earnings made in the session to the subject, you use a question form where you

can also show the following variables:

FinalProfit Sum of all earnings from the treatments

Money Added Money injected by the subject. (see bankruptcy procedure)

ShowUpFee Amount show-up fee.

MoneyToPay Showupfee + FinalProfit + MoneyAdded.

MoneyEarned Showupfee + FinalProfit.

Fischbacher/ z-Tree/ Mar. 29, 99 - 65 -

## 4.3 Running a Questionnaire

At the end of a session, the experimenter always starts a questionnaire with the command "Start Questionnaire" which is described below. This questionnaire must include a prompt for the subject's address (address form). When all subjects have entered their addresses, z-Tree writes the payment file. You may also insert a questionnaire at another point in the session. However, as questionnaires do not contain information regarding the number of subjects, they can be started at the earliest point after the first treatment (e.g., the welcome treatment). If more than one questionnaire contains an address form, the address only needs to be entered in the first questionnaire. In subsequent questionnaires only the payment file is updated.

A questionnaire is started with the command "Start questionnaire". This command is located at the place where the command "Start treatment" is located when a treatment window is open. While subjects are answering the questionnaires the clients remain in the state "questionnaire". The window "subjects table" shows which questionnaire is currently being answered.

## 4.4 Address Form

The address prompt asks for the subject's address. The fields "name", "first name" and "button" are compulsory if the address form should appear. The other fields are optional. If the question texts in the optional fields are left empty, the question is not sent to the clients. If the fields "name" and "first name" are left empty, there no form appears. The payment file is written whenever the last subject has passed an address form.

## 4.5 Question forms

Question forms consist of a list of questions. The layout of the question is adjusted with rulers. All question forms except the last must contain a button.

## 4.6 Questions

Questions in questionnaires correspond to items in treatments. In items, display options are defined in the field format. Here they can be set with radio buttons. The fields are as follows:

**Label:** Name of the question as it is shown to the subjects. If no variable is set, the label appears as text over the whole width of the screen.

**Variable:** If it is an input variable, this is the name of the question as it should appear in the data file. If input is not set, it may only be one of the variables described in the chapter "Profit Display".

**Type:** Format of the question.

*Text, number:* A text field appears into which input is entered. In the case of the number option there is a check to see if it is really a number that has been entered. Furthermore, the check determines whether the number is in the valid range. If the option "Wide" is selected for a text field, the entry field may consist of several lines. The number of lines can be selected.

*Buttons:* A button is created for every line in the field Options. Of course, only one question with this option is permitted per question form.

*Radio buttons:* A radio button is created for every line in the field options. These buttons are arranged vertically one on top of the other.

*Slider, Scrollbar:* The slider and scrollbar make a quasi continuous entry possible. Minimum, maximum, and resolution need to be entered. In the wide format, a label can be placed at the margins. These labels are entered in the field options. The maximum is always the value at the right border, the minimum is the value at the left border. Maximum may therefore be smaller than minimum.

*Radioline:* A radio button appears for all possible answers determined by minimum, maximum and resolution. In the wide format, a label can be placed at the margins. These labels are entered in the field Options. The middle button can be separated from the others by entering a number greater than zero for "distance of central button". The maximum is always the value at the right most button, the minimum is the value at the left most button. Maximum may therefore be smaller than minimum.

*Radiolinelabel:* Label line for radio button lines. Labels can be placed above the buttons at the margins. The texts for this action are in the options field.

*Checkbox:* A checkbox is created for every line in the options field. These checkboxes are arranged vertically one on top of the other. The resulting value is a list of all options checked.

**Wide:** Wide layout format: In the wide layout, the entry region for the question appears under the label over the whole width and does not only appear to the right of the label in a second column.

**Input:** This field determines whether a question is presented to the subjects (input is set) or whether a variable is only shown (input is not set).

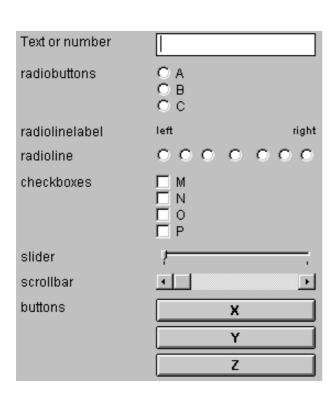
**Empty allowed:** This is set when the question does not need to be answered.

**Minimum, maximum, resolution:** When numbers have to be entered, these values are used for checking. With sliders, scrollbars and radio lines these values are used for converting the position into a number.

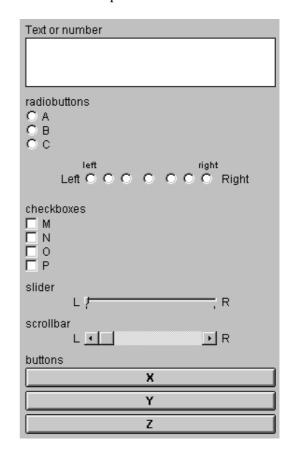
**Num. rows:** In the wide format for text entries, the number of rows that can be entered.

**Options:** The labels of buttons, radio buttons and, in the wide format, of sliders, scrollbars and radio lines.

Example of normal format (not wide):



Example of wide format:



## 4.7 Rulers

Rulers are used to set the regions where labels and questions are positioned.

**Distance to left margin:** The position can be given in points or in percentage of the screen width.

**Distance to the right margin:** The position can be given in points or in percentage of the screen width.

Fischbacher/ z-Tree/ Mar. 29, 99 - 68 -

**Distance between label and item:** This is the horizontal distance between the label and the question. Half of this distance is also kept to the left and right margins, which are defined by the options defined above.

**Size of label:** This is the maximum width of the label. When the display is given in percentage, this refers to the width between left and right margin and not to the width of the whole screen.

## 4.8 Buttons

Buttons conclude question forms. You cannot define more than one button per form. However, this button can be positioned anywhere on the form. The label of a button can be selected at random.

# 5 Installations

## 5.1 Installation with a File Server

The programs z-Tree and z-Leaf are best stored in a common directory. The server must have both read and write access to the directory while the client should only have read access.

## 5.2 Installation without File Server

Z-Tree and z-Leaf can run on different computers that do not have access to a common file server. This option is needed if you want to run experiments over the internet. As opposed to an installation with file server there is no easy method for z-Leaf to know where Z-Tree is running. Therefore, z-Leaf must have the file server.eec in it's directory or in the directory c:\expecon\conf. Z-Tree's IP address can also be set with the option/server when the client starts up.

If z-Tree and z-Leaf do not run from a common directory, the function "Restart all clients" does not work. In case there is a problem, the clients have to be started up manually.

## **5.3** Setting up a Test Environment

You do not need several computers to test programs. You can start a z-Tree and more than one z-Leaf on one computer. However, you have to give the z-Leaves different names. For this you use the command line option /name. For example, you can start z-Leaf twice, once with "zLeaf /name A" and once with "zLeaf /name B".

# **5.4** Command Line Options of z-Tree

Z-Tree can be started with command line options. The following options are available:

/language lan lan can have the value "en" English and "de" for German. This option sets the

language as the menu command "language". The default value is German. If you

want to change this in your environment, you can create a shortcut for z-Tree that

initializes the language to your choice.

/channel ch Determines the channel through which the z-Leaves establish contact to z-Tree.

The actual channel is 700+ch. If you want to run more than one z-Tree on one

computer they must work with different channels, i.e., you set the channel to values greater than one.

Fischbacher/ z-Tree/ Mar. 29, 99 - 71 -

# 6 Reference

## **6.1** Menu Commands

#### File

#### **New Treatment**

A new treatment is created. It contains the background and the standard boxes.

#### **New Questionnaire**

A new questionnaire is created. It is completely empty.

#### Open...

Opens a treatment or a questionnaire. A dialog, in which the treatments are shown, appears. In order to open a questionnaire, you have to select the corresponding option.

#### Save

Saves a treatment or a questionnaire.

#### Save As...

Saves a treatment or a questionnaire under a new name or at another place.

## Export Treatment...

Treatment is written in textual form in a file. A treatment exported in this way may also be read in again.

## Export Questionnaire...

Questionnaire is written in textual form in a file. A questionnaire exported in this way may also be read in again.

#### Export GameSafe

A GameSafe is translated from an unreadable binary format into a gigantic ASCII file.

#### Export Table

A table is written in a tab-separated file.

## **Import**

Imports a treatment or a questionnaire.

## Import Table

A table (tab-separated file) is read into the current window, if it is a table window.

## Page Setup

(Not yet implemented)

#### Print...

(Not yet implemented)

#### Previous files

Opens a treatment or questionnaire previously used.

#### Quit

Quits the program. A warning message appears if the session is still running, i.e., if a treatment is running, if a questionnaire is running or if the payment file is not yet written.

#### Edit

### Undo

(Not yet implemented)

#### Cut

Deletes the marked branch in the stage tree or in the questionnaire and places it onto the clipboard.

## Copy

Copies onto the clipboard the marked area of a table or the selected branch in the stage tree or in the questionnaire.

#### Paste

In the parameter table: Copies a table from the clipboard and pastes it at the selected position.

In the stage tree or in the questionnaire: Copies the branch from the clipboard and pastes it at the selected position.

## Copy groups

Copies the groups of the selected area of the parameter table onto the clipboard.

## Paste groups

Copies the groups from the clipboard and pastes them at the selected position in the parameter table.

#### Insert cells

Adds more rows or columns immediately after the last row or column selected. If more than one row or column are selected, a corresponding number of rows or columns is added.

#### Remove cells

Removes complete rows or columns. Complete rows are selected by clicking at the small cell at the left border of the parameter table. Complete columns are selected by clicking at the cell at the top of the parameter table.

#### **Treatment**

#### Info...

Opens a dialog in which the parameters of the selected element can be viewed and changed.

## New Stage

Creates a stage.

## **New Program**

Creates a program.

#### New header box

Creates a header box.

#### New standard box

Creates a standard box.

#### New calculator button box

Creates a button box.

## New history box

Creates a history box.

Creates a contract list box.
New contract grid box
Creates a contract grid box.
New message box
Creates a message box.
New button
Creates a button.
New checker
Creates a checker.
New item
Creates an item.
Stage Tree
Opens the window with the stage tree.
Parameter table
Opens the window with the parameter table.

- 75 -

New help box

New grid box

Creates a grid box.

Creates a help box.

New container box

Creates a container box.

New contract creation box

Creates a contract creation box.

New contract list box

#### Check

Checks the programs and parameters for errors.

## Import variable table...

You give the name v of a variable and choose a text file that contains a table. Then the line

v = w;

is added into the program of the specific parameters. Here w stands for the value from the table. If the format of the table does not correspond with the format of the parameter table, the specific parameters are filled where there is a value in the table.

#### Show variable

Shows the value of a variable in the parameter table.

## Matching: Partner

A partner design for group matching is selected: The first subjects constitute group 1, the next, group 2, etc., for each period. If a range of the parameter table is selected, group matching is only carried out for this region.

## Matching: As First Period

Groups are matched for all periods in the same way as they are now matched for the first period. If a region of the parameter table is selected, group matching is only carried out for this period. The first period of the selection counts as the first period.

## Matching: Stranger

Group matching is carried out at random for each period. If a region of the parameter table is selected, group matching is only carried out for this region.

## Matching: Absolute Stranger

Group matching is carried out in such a way that each subject is never in the same group with any other subject more than once. This is only possible for a limited number of periods. In the periods in which such a group matching is no longer possible, 1 is set as group number for all subjects. If a region of the parameter table is selected, group matching is only carried out for this region.

Fischbacher/ z-Tree/ Mar. 29, 99 - 76 -

Caution: The execution of this command can take up some time. Depending on the given number it can require milliseconds, minutes or days. Before the command is executed, a warning appears and you get the possibility to cancel the execution of the command.

## Matching: Absolute Typed Stranger

This creates an absolute stranger matching for heterogeneous groups: If, for example, a group consists of a company and two employees, we often do not want the same subject to have different roles. This means that only those groups that contain a company and two employees are allowed.

Caution: The execution of this command can take up some time. Depending on the given number it can require milliseconds, minutes or days. Before the command is executed, a warning appears and you get the possibility to cancel the execution of the command.

In the following table the number of periods for which we have calculated absolute typed stranger is listed. The values for which it is known that it is the highest possible value are given in boldface.

	_															
num	grou	ups			1	2	3	4	5	6	7	8	9	10	11	12
subj. per type																
			2		1	3	5	7	9	11	13	15	17	19	21	23
		1	1		1	2	3	4	5	6	7	8	9	10	11	12
			3		1	1	4	4	7	7	7	7				
		1	2		1	1	3	4	5	6	7	8				
	1	1	1		1	1	3	4	5	5	7	6				
	•	•	4		1	1	1	5	5	5						
		1	3		1	1	1	4	5	5						
		2	2		1	1	1	4	5	6						
	1	1	2		1	1	1	4	5	6						
	1				•	-	-		5							
1	1	1	1		1	1	1	4		5						
			5		1	1	1	1_	6							
		1	4		1	1	1	1	5		Х	>=X	:			
		2	3		1	1	1	1	5		X	=X				
	1	1	3		1	1	1	1	5							
	1	2	2		1	1	1	1	5			quic	k ca	lcula	tion	
1	1	1	2		1	1	1	1	5			avai				
1 1	1	1	1		1	1	1	1	5			Avai	l. on	file	(pro	v.)

## Matching: Transform

Allows to make transformations on the matching. There are three options:

**Add:** You can add a fixed number to the group ID.

Multiply: You can multiply the group ID with a fixed number.

**Shuffle:** You can shuffle the group IDs within the period.

With this command one can for instance easily define matching groups. Using matching groups is an important experimental technique because in a stranger matching there is generally only one independent observation per session. With matching groups on can define a stranger matching with more than one independent observation. For that, the subjects are divided into two (or more) sub-populations. Subjects

are only matched within these sub-populations. Therefore these sub-population constitute independent observations.

Example: You have 24 subjects with groups of 3 and you want to make two matching groups of 12 subjects each. You have to do the following:

- 1. Set the number of groups to 4.
- 2. Select the first 12 subjects in the parameter table and choose "Matching Stranger".
- 3. Select the last 12 subjects in the parameter table and choose "Matching Stranger" again.
- 4. Choose (the last 12 subjects are still selected) "Matching Transform". In the dialog that appears select the radio button "add" and enter 4 into the value field.
- 5. Set the number of groups to 8.

## Utilities: Make Stage Names Unique

Changes the names of the stages by adding numbers in such a way that the names of the stages are unique. This is important because the state of the clients is defined by the stage name.

## Language

Sets the language for elements that are newly created as header boxes etc. All texts within these elements can be modified later, i.e. they can also be translated later.

### Questionnaire

#### Info...

Opens a dialog in which the parameters of the selected element can be viewed and changed.

#### **New Address Form**

Adds a new address form.

## **New Question Form**

Adds a new question form.

#### **New Ruler**

Adds a new ruler

#### **New Question**

Adds a new question.

#### **New Button**

Adds a new button.

## Check

Checks the questionnaire.

#### Language

Sets the language for elements that are newly created as address forms. All texts within these elements can be modified later, i.e. they can also be translated later.

#### Run

#### Clients' Table

Opens the clients' table.

#### Shuffle Clients

Shuffles the clients at random.

#### Sort Clients

Sorts the clients. Should names have the same beginning and a number at the end, they are sorted according to the value of the number at the end, i.e., b2 comes before b11.

#### Start Treatment

Starts the treatment as soon as possible. If it is the first treatment, it is first checked whether the correct number of clients is connected. If not, a warning appears and the start can be canceled. If the treatment is allowed to start, z-Tree waits until at least as many clients are connected with the server as are needed to run the treatment. Then the treatment is started. If it is not the first treatment, the treatment starts immediately.

#### Start Questionnaire

Starts the questionnaire, if this is possible. A treatment needs to have been started beforehand and the clients need to be in the state Ready.

#### Time Table

Opens a window in which the time displayed to the subjects is given. If different times are displayed to different subjects, the individual times of the subjects are listed in the column "ptime".

## globals Table

Opens the window showing the globals table or brings it into the foreground.

## subjects Table

Opens the window showing the subjects table or brings it into the foreground.

#### contracts Table

Opens the window showing the contracts table or brings it into the foreground.

#### summary Table

Opens the window showing the summary table or brings it into the foreground.

#### Stop Clock

Stops the clock. However, subjects can still make entries. If, for instance, all subjects click "continue", the session continues regardless of the stopped clock.

#### Restart Clock

The clock is restarted.

#### Discard a client

The client selected is discarded. It can be replaced by a new client. In case of a defect in a subject PC, the following needs to be done: The client in question is clicked in the clients' window. Then the menu "Discard" is chosen. With this the client is discarded. The new PC is started. The client then appears in the lowermost row of the clients' window. It first has to be selected and then moved over the old client. In doing this the mouse has to be released for an instant. If the mouse is not released this procedure only selects the space between the new and the old client.

#### Restart all clients

All clients are restarted. This is implemented by writing a file "@ararkiri.txt" in the current directory. This file is sought by the z-Leaves every few seconds and when it is newly available z-Leaf is restarted.

## Replay old session...

The server writes every message that takes place between itself and the clients into the file called GameSafe. This makes it possible to continue the session even after a crash of the experimenter PC. This is accomplished by restarting the server and selecting "Replay old session". Next the GameSafe from where the session needs to be restarted may be selected. As the whole course of the session and not

Fischbacher/ z-Tree/ Mar. 29, 99

just the end state is stored, all questions that were answered at the experimenter PC during the session need to be re-answered.

#### **Windows**

## Treatment and questionnaire files

All treatments and questionnaires opened in z-Tree can be brought into the foreground in the windows menu.

#### Toolbar

Opens and closes the toolbar.

## Status Bar

Opens and closes the status bar.

?

(unimplemented help function)

## 6.2 Programming

## **Constants**

**TRUE** 

**FALSE** 

## **Operators**

Mathematical operators

- + addition
- subtraction
- \* multiplication
- / division

## Relational operators:

- < smaller
- <= smaller or equal
- == equals

```
>= greater or equal
```

> greater

Logical operators:

& logical and

logical or

Scope operators:

: next higher scope

highest possible scope. This is always the record of the globals table.

## **Statements**

In the following, a and b are logical expressions, x, y, and z numeric expressions, v is the name of a variable, t is the name of a table, s is one statement, ss1 and ss2 are lists of statements.

x = yi	Assigns the expression y to the variable x.
if (a) {ss1}	If a returns TRUE, the statements ss1 are executed
if (a) $\{ss1\}$ else $\{ss2\}$	If a returns TRUE, the statements ss1 are executed; otherwise the
	statements ss2 are executed.
<pre>t.new{ ss }</pre>	in the table t, a new record is generated and in this record the
	statements ss1 are executed
array $v[x, y, z];$	defines an array, i.e., an indexed variable. Array variable can be
	accessed with $v[x]$ .

## **Functions**

In the following, a and b are logical expressions, x and y numeric expressions.

abs(x)	Absolute value of x.
and(a,b)	TRUE if and only if a and b are true.
atan ( x )	Arctangent of x.
cos( x )	Cosine function of x.
exp(x)	Exponential of x; e <sup>x</sup>
if( a,x,y )	If a, then the value of the function is $x$ , otherwise $y$ .
In( x )	Natural logarithm of x.
log(x)	Base -10 logarithm of x.
max( x,y )	Maximum of x and y.

min( x,y )	Minimum of x and y.
mod(x,y)	Remainder after x is divided by y.
not(a)	TRUE if and only if a is <i>not</i> true.
or( a,b)	TRUE if and only if a or b is true.
pi()	3.1415
power( x,y )	x <sup>y</sup> . x must be positive
random()	Uniformly distributed random number between 0 and 1.
randomgauss()	Normally distributed random number with average 0 and standard
	deviation 1.
randompoisson( x )	Poisson distributed random number with average x (the result is a whole
	number).
round( x,y )	Rounds $x$ to a multiple of $y$ , i.e., supplies the multiple of $y$ that is closest
	to x.
	Examples:
	round( 345.67, 20) == 340, round( 345.67, 0.1) == 345.7, round( 2.5, 1)
	== 3, round( -2.5, 1) == -3.
rounddown( x,y)	Rounds x down to a multiple of y, i.e., it returns the greatest multiple of y
	that is smaller or equal to $\mathbf{x}$ . This definition is also employed for negative
	numbers. Example: rounddown( $-3.6,1$ ) == $-4$ .
roundup( x,y )	Rounds x up to a multiple of y, i.e., it returns the smallest multiple of y
	that is greater than or equal to x. This definition is also employed for
	negative numbers. Example: roundup( $-3.6,1$ ) == $-3$ .
same( x )	Short form for $x == :x$
	Therefore, this function is only feasible in table functions. x may also be
	an expression.
sin( x )	Sine of x.
sqrt( x )	Square root of x.

## Table functions

average(x), $average(a,x)$	Average of the (found) values.
<pre>count(), count(a)</pre>	Number of records in the table or number of found records.
find(x), $find(a,x)$	The first value of x (where a is satisfied).
maximum(x), $maximum(a,x)$	Maximum of the (found) values.
<pre>median(x), median( a,x)</pre>	Median of the (found) values.

```
\begin{array}{ll} \mbox{minimum}(x)\,,\,\,\mbox{minimum}(\,a\,,x) & \mbox{Minimum of the (found) values.} \\ \mbox{product}(x)\,,\,\,\mbox{product}(\,a\,,x) & \mbox{Product of the (found) values.} \\ \mbox{regressionslope}(\,\,x\,,y) & \mbox{Gradient of a linear regression through the (found) points}\,(x,y). \\ \mbox{regressionslope}(\,\,a\,,x\,,y) & \mbox{Standard deviation of the (found) values.} \\ \mbox{Standard deviation of the (found) values.} \\ \mbox{Sum}(x)\,,\,\,\mbox{sum}(a\,,x) & \mbox{Sum of the (found) values.} \\ \end{array}
```

#### **Iterator**

```
\begin{array}{lll} \text{iterator}(v) & \text{creates a small table with the variable v. (see the} \\ \text{iterator}(v, x) & \text{chapter on iterator}) \\ \text{iterator}(v, x, y) & \\ \text{iterator}(v, x, y, z) & \end{array}
```

## Syntax diagram of the programming language

The following syntax description is intended to serve as a reference to experts.

Syntax used to express the grammar

```
a : b; : a can be replaced by b
a b : a followed by b
a | b : a or b
"x" : exactly that text
a* : zero, one or more repetitions of a
(a) : a
'x' : the character x
```

#### **Definitions**

#### **Terminal elements**

```
NAME : letter (letter|digit)*;

NUMBER : real;

BOOLVALUE : "TRUE" | "FALSE";

FUNCTIONNAME0 : "pi" | "random" | "randomgauss";

FUNCTIONNAME1 : "abs" | "atan" | "cos" | "exp" | "ln" | "log" | "sin" |

"sqrt" | "randompoisson";
```

```
"roundup";
BOOLFUNCTIONNAME1 : "not";
BOOLFUNCTIONNAME2: "and" | "or";
TABLEFUNCTION0
                      : "count";
                        : "sum" | "product" | "average" | "stddev" | "minimum" | "maximum" | "median" | "find";
TABLEFUNCTION1
                        : "pearsoncorr" | "spearmancorr" | "regressionslope";
: "&" | "|";
: "<" | "<=" | "!=" | "!=" | ">=" | ">";
TABLEFUNCTION2
BOOLOPERATOR2
COMPOPERATOR
                        SCOPEOPERATORS
Grammar
                         : statementlist
program
statementlist
                         | statement statementlist
                         : ';'
statement
                            lvalue '=' expression ';'
                          "do" '{' statementlist '}'
| tableidentifier '.' "do" '{' statementlist '}'
| "new" '{' statementlist '}'
| tableidentifier '.' "new" '{' statementlist '}'
| "if" '(' condition ')' '{' statementlist '}'
| "if '(' condition ')' '{' statementlist '}'
| "else" '{' statementlist '}'
                            "else" '{' statementlist '}'
                           "array" arraydeclaration ';'
lvalue
                         : variable
arraydeclaration : NAME '[' ']'
                          | NAME '[' expression ']
                          NAME '[' expression ',' expression ']
NAME '[' expression ',' expression ',' expression ']'
expression
                         : summand
                            '+' expression
                            '-' expression
                           expression '+' expression
                           expression '-' expression
summand
                         : factor
                           summand '*' summand
                           summand '/' summand
factor
                         : NUMBER
                          NAME
                           '(' expression ')'
FUNCTIONNAME0 '(' ')'
                          FUNCTIONNAME1 '(' expression ')'
FUNCTIONNAME2 '(' expression ',' expression ')'
```

: "max" | "min" | "mod" | "power" | "round" | "rounddown" |

FUNCTIONNAME2

```
"if" '(' condition ',' expression ',' expression ')'
                     tablefunction
variable
                   : NAME
                   arrayvariable
                    SCOPEOPERATOR variable
                   : NAME '[' expression ']'
arrayvariable
tablefunction
                   : puretablefunction
                   | tableidentifier '.' puretablefunction
puretablefunction : TABLEFUNCTION0 '(' ')'
                     TABLEFUNCTION0 '(' condition ')'
TABLEFUNCTION1 '(' expression ')'
                     TABLEFUNCTION1 '(' condition ',' expression ')'
                    TABLEFUNCTION2 '(' expression ',' expression ')'
TABLEFUNCTION2 '(' condition ','
                                          expression ',' expression ')'
tableidentifier
                   : NAME
                     "iterator" '(' NAME ')'
                     "iterator" '(' NAME ',' expression ')'
                     "iterator" '(' NAME ',' expression ',' expression ')'
                     "iterator" '(' NAME ',' expression ','
                                               expression ',' expression ')
condition
                   : boolatom
                   | boolatom BOOLOPERATOR condition
boolatom
                   : BOOLFUNO
                     expression COMPOPERATOR expression
                      '(' condition ')'
                     BOOLFUNCTIONNAME1 '(' condition ')'
                     BOOLFUNCTIONNAME2 '(' condition ',' condition')'
                     "same" '(' expression ')'
```

#### The tables and its standard fields

## The subjects table

Table containing subjects' data from the current period.

**Period:** Number of the period. The first paying period is the period 1. Thus, if there are 3 trial periods, the first period has the number -2.

**Subject:** Number of subject; starts at 1.

**Group:** Group number as it is displayed in the parameter table (possibly modified by a program).

**Profit:** Profit made in this period (in experimental currency units). Needs to be calculated; default is zero. Profit is the relevant variable for payment.

**TotalProfit:** Total profit made in this treatment. This is calculated automatically. It should not be changed.

**Participate:** Indicates whether the subject is taking part in the current stage or not. If Participate equals 1, he or she is taking part, if 0, he or she is not taking part. Default is 1.

## The subjects table of the previous period "OLDsubjects"

Table containing subjects' data from the previous period.

## The globals table

Table with global variables.

**Period:** As in subjects table.

NumPeriods: ID of the last period.

**RepeatTreatment:** If this variable is greater that zero after the last period, the treatment is run again with the same number of periods. The period counter is incremented as if all would be done in one treatment.

## The summary table

The summary table contains one record per period. This record is not destroyed when the period has been finished. The whole table is destroyed after the treatment has been finished. This table is most useful for observing aggregate data of the treatment.

**Period:** Number of period as in subjects table.

## What can be contained in stage tree elements?

In the following we list for all stage tree elements what kind of other elements they can contain. We use the notation "a < b c" if the element a contains the element b followed by c. A star after an element means zero or one or more instances one of this kind of element. The first line therefore signifies that the stage tree contains one background followed by some stages (zero or more). The notation " $a = b \mid c$ " means that the term a is one of b or c. This notation is used to define the term box.

stagetree < background stage\*

background < program\* activescreen waitingscreen stage < program\* activescreen waitingscreen

```
activescreen
                     < box*
waitingscreen
                     < box*
box
                     = standardbox
                     headerbox
                     helpbox
                     containerbox
                     gridbox
                     | calculatorbuttonbox
                     historybox
                     | contractcreationbox
                     | contractlistbox
                     contractgridbox
                     < item* button*
standardbox
headerbox
                     < -
helpbox
                     < -
containerbox
                    < box*
gridbox
                     < item* button*
calculatorbuttonbox < -
historybox
                    < item*
contractcreationbox < item* button*</pre>
contractlistbox
                    < item* button*
contractgridbox
                    < item* button*
                     < -
item
button
                     < checker* program*
```

## **6.3** The Import Format

Treatments and questionnaires can be brought into a readable format with "Export". This format may also be read in again. This format is not described in detail here. Generally, however, the elements of the stage tree are given a form such as the following:

```
element name {
    option1 = value1;
    option2 = value2;
    subelement1
    subelement2
}
```

The sub elements have the same structure. Exporting a treatment or a questionnaire supplies the options.

# 7 Tips and Tricks

## 7.1 **Calculation of Rank**

If you wish to determine for a record at which place the value of a variable x is situated relative to the others, you can define a formula. The record with the greatest x shall be given a 1.

If all x are different or we accept that several x may have the same rank:

```
Rank = count ( x >= :x);
```

If x is an integer and we wish to have a different rank for each x, we add a random number to x and calculate the rank of this number. We may not thereby integrate the random function into the sum formula, as we would then get an inconsistent ranking order.

```
x1 = x + random() / 2;
//---- in a new program
Rank = count ( x1 >= :x1);
```

Note: When we write these two lines into one program, the variable x1 of the others is not yet calculated when we use it in the rank formula. A simple trick consists of filling the random number into a field at the very beginning of the procedure.

We thus set

```
random1 = random();
```

in a program of the background

Now we can define the rank:

```
Rank = = count ( x + random1 >= :x + :random1);
```

## 7.2 Reduce Time-out Times after the Experiment has Started

One defines the time-out in the parameters for each stage. This *formula* can no longer be changed in the experiment. Therefore, you define Timeout = Timeout0; . The *variable* Timeout0 can be changed in the period parameters. It is also possible to do this during the experiment for the periods not yet started.

## 7.3 Leave Profit Display Standing

You can also show the same information as on the active screen on the waiting screen. You only have to remove the "continue" button.

## 7.4 Observer Subject

You can define a subject who sees the results of the experiment. For this you define a variable Observer which you set at 0. In all stages we add in a program in the subjects table the line

```
Participate = 1- Observer;
```

Next we define an additional stage in which the variables that are of interest here are shown on the active as well as on the waiting screen. In the program you write lines such as the following:

```
AvOffer = average(Observer ==0, Offer);
```

Now we define a subject with

Observer =1;

With this an interested party may follow these variables at a PC different from the experimenter PC.

## 7.5 Checking of Exercises

You can leave the checking of exercises to the computer. To this end you define a treatment that only contains the exercises. Checkers check whether the values are correct. Common errors can be caught by specialized checkers. These must stand at the beginning.

Example:

Add 3 and -4

Checker: result != 7 / Note that the numbers are not both positive.

Checker: result == -1 / There is an error in your calculation.

The check questions for the first treatment are attached to the welcome treatment so that the subjects need not wait until all have clicked "continue" on the welcome screen to begin answering the check questions.

## **7.6** Differing Endowments

You define a variable

```
Lumpsum = 0;
```

This variable can be changed in the specific parameters.

Under certain circumstances the profit should not contain the lump-sum payment. This means that you define and set a variable ShownProfit

```
Profit = if( Period == 1, ShownProfit + Lumpsum, ShownProfit);
ShownsTotalProfit = TotalProfit - Lumpsum;
```

## 7.7 Display Costs Table

If you wish to show the subjects a table with values, you make a grid box. It is easiest to define an array that contains the function values.

## 7.8 Displaying the Number of Players that Have not yet Finished

First you define a variable in the globals table, e.g., NumPlayersWorking, and set the variable at the number of players that still have entries to make. If all players still have entries to make, you write:

```
NumPlayersWorking = subjects.count();
```

Next you add the following program for the globals table to the OK button.

```
NumPlayersWorking = NumPlayersWorking -1;
```

NumPlayersWorking is automatically updated and may be displayed anywhere.

## 7.9 Dutch Auction

In the Dutch auction a price clock that gradually runs backwards is shown. As soon as a buyer says stop, he or she receives the goods at the price displayed at that moment.

You program a Dutch auction in z-Tree by defining a subject as auctioneer. This subject can bid down prices with a button as explained in the chapter 3.9 on market experiments. Of course, this role is played by an experimenter.

## 7.10 Multiphase Experiment

If a period in an experiment consists of several phases, you define the appropriate number of stages for each phase. If a phase should be very complex, it is generally easier to define each phase as one period.

You define a variable Phase. This variable is calculated at the beginning of a period. If, for example, we have 2 periods

```
Phase = mod(Period-1, 2) +1
```

calculates the phase and

```
UserPeriod = rounddown( Period/2 );
```

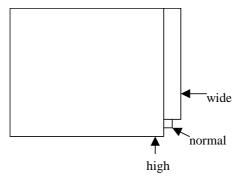
calculates the period as it is supposed to be shown to the subjects. This variable cannot appear in a normal header box. However, you may insert it in a normal standard box (this also holds for the variable Period).

## 7.11 Testing with overlapping z-Leaves

If you want to test a program with more than one client on one computer, it is sometimes difficult to find out which z-Leaf belongs to which client. A good idea is to start the clients with slightly different sizes. This works particularly well if you can run a test on a screen that is larger than the screens of the clients. You can, for instance, start three z-Leaves in the following way:

```
zleaf /name normal /size 640x480
zleaf /name wide /size 650x470
zleaf /name high /size 630x490
```

In this way the z-Leaves are arranged as in the figure below and you can fetch each of the windows with the mouse.



## 7.12 Payment of a Random Period

An experimental technique to avoid income effects consists in paying one randomly selected period. To do this with z-Tree you have to store the profits of all the periods in the summary table.

You need the following program in the summary table:

The variable SelectedPeriod can be set at a subject's screen after a dice has determined the paid period.

## 7.13 Own Records at the Top of a List

If you want to place the own records of a subject at the top of a list, you can insert an expression into the sorting field of the contract list box. Suppose you want to start with the records where you are the seller. In these records, the variable Seller has the value of your variable Subject. This can be expressed with the following expression:

```
if( Seller == :Subject , 0, 1)
```

Please note the use of the scope operator: It accesses the record of the subject that is in front of the screen that is set up.