Plan of the course

Part 1: Getting started with z-Tree

Part 2: Experiments on individual decision making

Part 3: One-shot two-player games

Part 4: Repeated two-player games

Part 5: Multi-player games: auctions and markets

Part 6: Advanced concepts, questionnaires and crashes

Part 7: Recruitment, Text formatting and Multimedia

Part 8: Advanced programming examples

Part I

Getting started

Getting started

Introduction

Installing z-Tree

- on your PC
  - command-line options
- on a file server
- Without a file server
- Test the installation

Structure of a z-Tree treatment

Pros and Cons of computerized experiments

- **Pros:**
  1. speed
  2. reliability and ease of data collection
  3. replicability
  4. **new possibilities**

- **Cons:**
  1. some subjects pools may have problems using computers
  2. need to run the experiment in a room, with the necessary equipment

Structure of a typical session I

1. Subjects arrival
   - check subjects on the list
   - pay and send home supernumerary subjects (if any)
   - randomly assign subjects to workstations

2. Instructions
   - hand out written instructions
   - read instructions out loud
   - question time
   - control questions (possibly computerized)

3. Bathroom break

Structure of a typical session II

4. Trial periods (optional)

5. Treatment(s)

6. Questionnaire (computerized, if possible)

7. Payment
   - must be individual and anonymous
   - check out all the bureaucratic requirements (receipts, copies of the documents, etc.)

How can z-Tree help you?

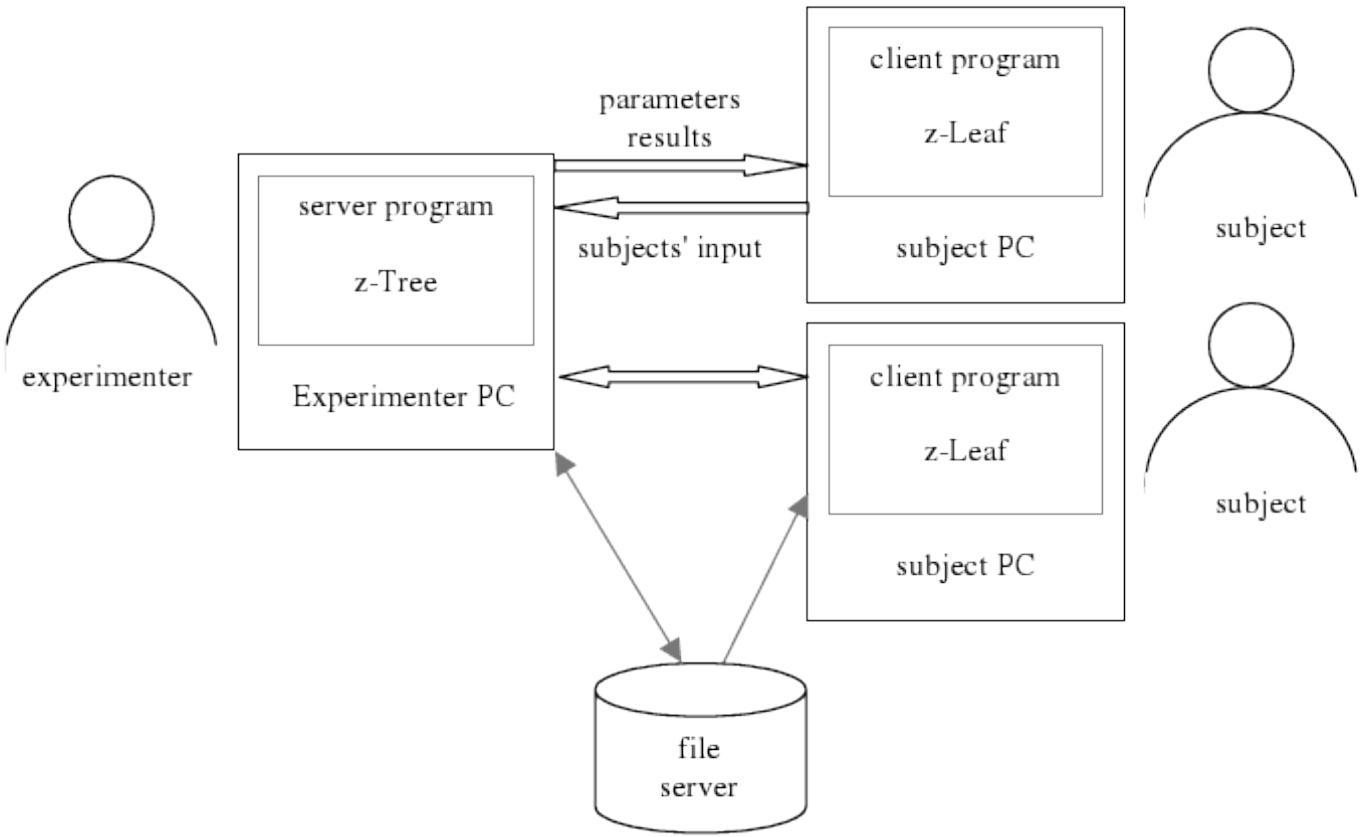
1. **Control questions:**
   - record how many times a subject gives the wrong answer
   - measure how long it takes for a subjects to complete the control question

2. **Treatment(s)**

3. **Questionnaires**

4. **Payment**
   - manage losses
   - add show-up fee
   - round earnings
   - prepare simple receipts

How does z-Tree work?

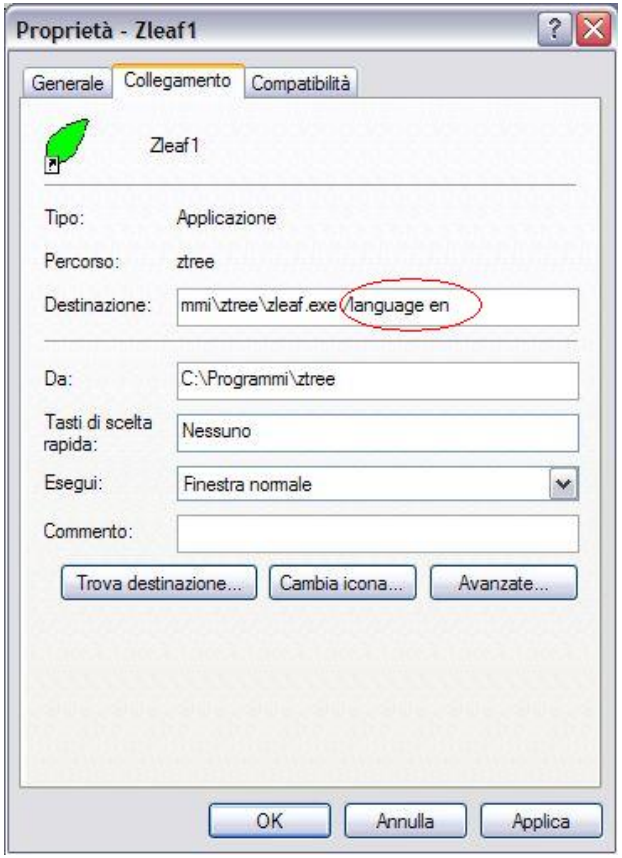


Installing z-Tree on your PC

Simply put zLeaf.exe and zTree.exe in *the same* folder, on your pc.

Command-line options:

1. create a shortcut to zTree.exe or to zLeaf.exe
2. add the desired options in the "target" field.



z-Leaf command-line options

"/name" to create shortcuts to different clients on the same computer (useful for testing).

- `/name zleaf1`
- `/name zleaf2`
- ...

"/language" to change the language used in the clients' interface.

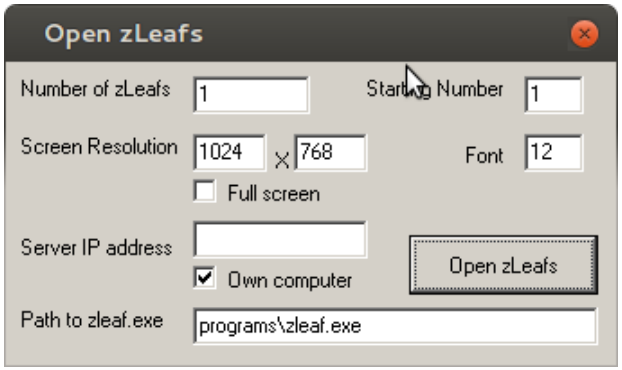
- `/language en`
- `/language de`
- ...

For the full list of options, see the z-Tree reference manual, at pp. 57-59.

Batch files

To make things simpler, you can

- download and extract Ernesto Reuben's zip file: `http://www.ereuben.net/teach/zTreeTestEnvironment.zip`
- place the executables zTree.exe and zLeaf.exe in the "program" sub-folder
- launch z-Tree with the batch file: "openztree.bat"
- launch as many z-Leaves as you need using the file "Open Zleafs.exe".

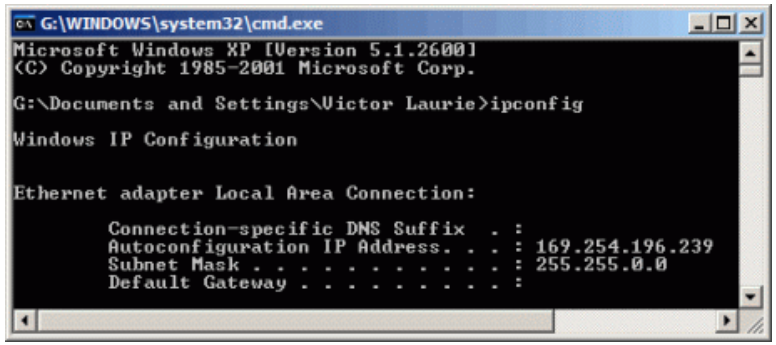


Installing z-Tree on a file server

1. put the files z-Tree.exe and z-Leaf.exe in a single directory on the file server
2. make sure you have **read/write permissions** on this directory **and on the parent directory**
3. make sure clients have read permissions on the directory

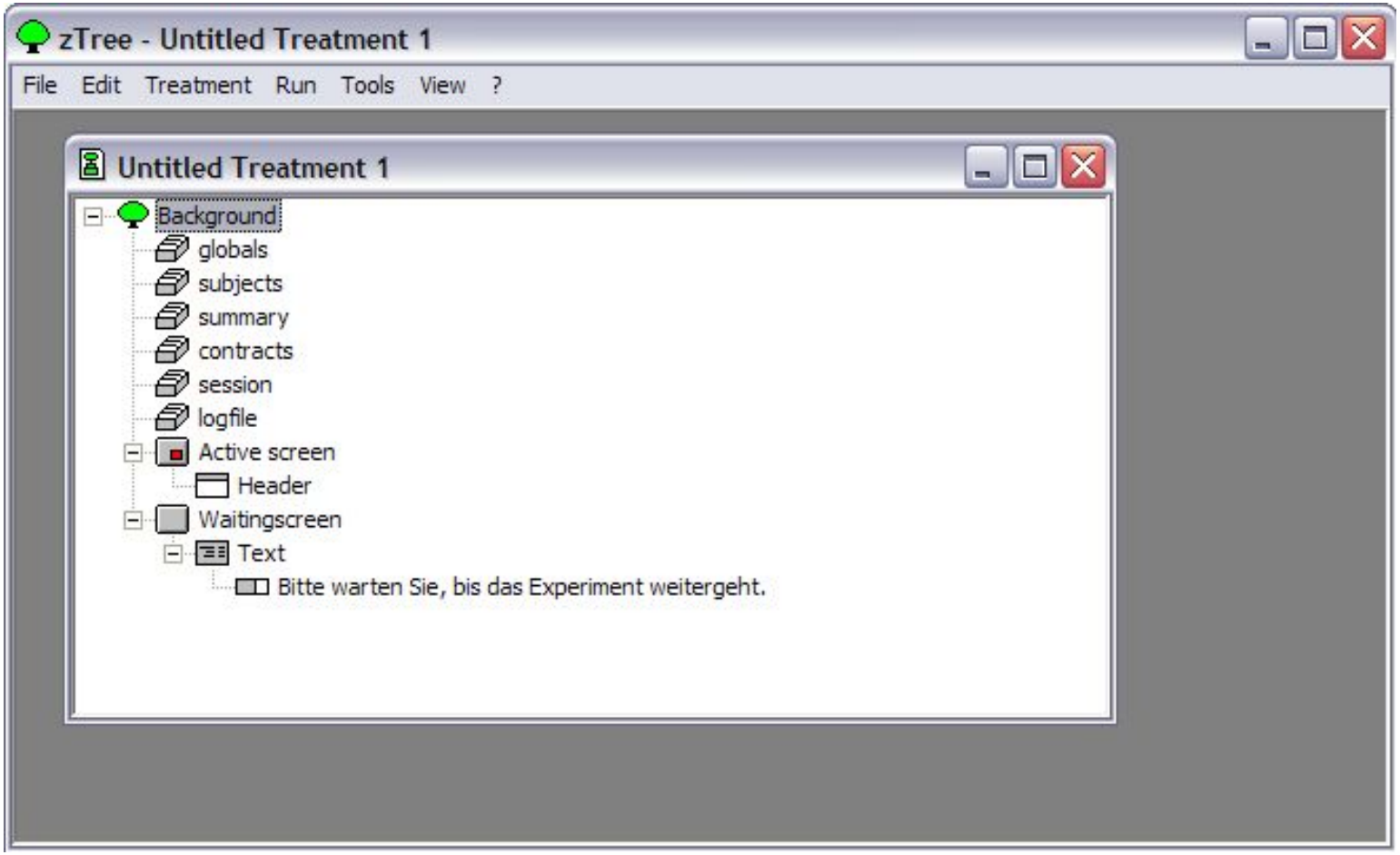
Installing z-Tree without a file server

- Z-Tree and z-Leaf can run on different computers that do not have access to a common file server (e.g. to run experiments over the Internet).
- You need to run zLeaf with the command line option "/server"
  - e.g. `/server 169.252.196.239`
  - where 169.252.196.239 is the ip address of the computer where zTree is running
- To know a computer's ip address open the windows' shell and type `ipconfig`



Testing z-Tree

z-Tree starts with one untitled treatment.



You can change the default language from the menu `Treatments`→`Language` before creating a new treatment.

Testing z-Leaf I

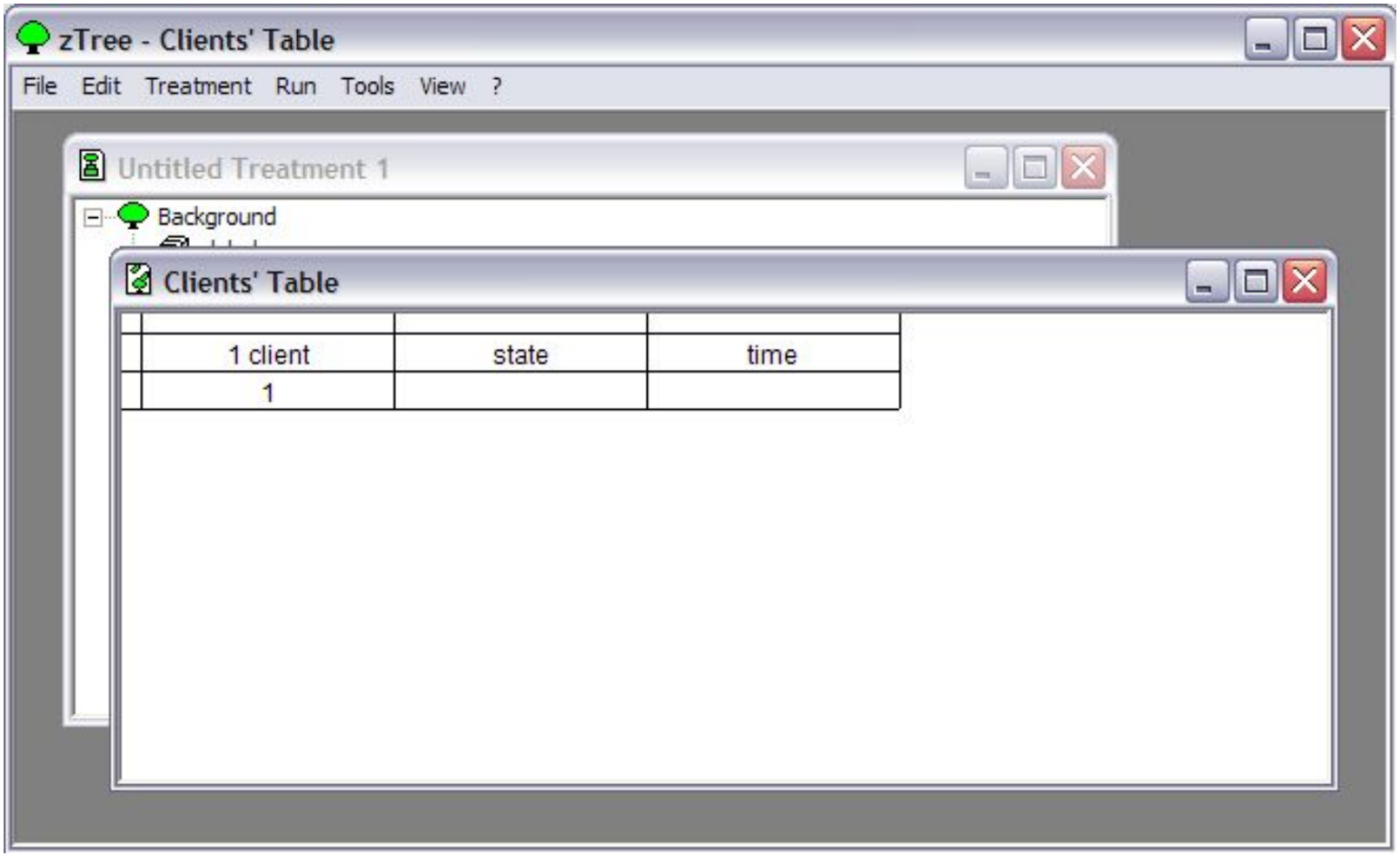
z- Leaf starts only if z-Tree is running.
By default, it starts as a full-screen window.

- Press `Alt+F4` to close it,
- `Alt+Tab` to switch to a different window.

Testing z-Leaf II

When you start a "leaf" (i.e. a new client), you can also see it from the server program z-Tree.

Select `Run`→`Clients Table` from the menu at the top.



Structure of a z-Tree treatment

Background:

- contains the general information about the treatment
  - number of subjects
  - number of periods
  - conversion rate
- lists the **tables** used in the treatment
- defines the basic layout of the **Active Screen** and of the **Waiting screen**
- may contain **programs** that are executed at the beginning

Stages:

- correspond to the "screens" of the treatment
- may contain programs that are executed at the beginning of the stage

Part II

Individual Decision Making

Individual Decision Making

Example: lottery

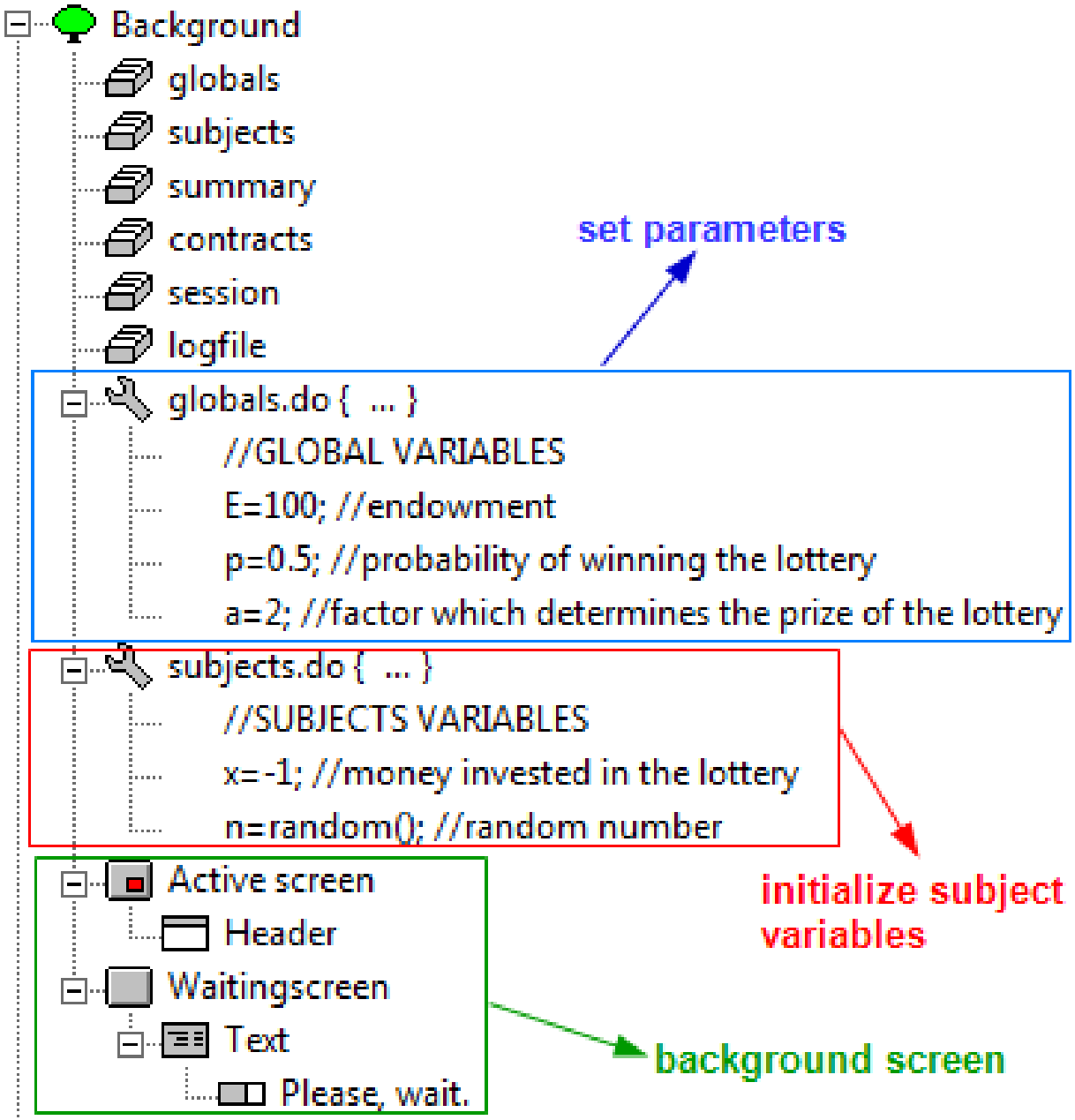
- endowment: E
- the lottery yields:
  - $x(1+a)$ with probability p
  - 0 with probability $1-p$
- task: choose x between 0 and E
- random draw: n between 0 and1
- payoff: $E - x + I[n \leq p]x(1+a)$

see lottery.ztt

Plan of the treatment

1. **Background:** set the value of the variables
   - **global variables:** same value for all subjects $\rightarrow$ E, p, a
   - **"subject" variables:** possibly different values for different subjects $\rightarrow$ x, n
2. **Stage 1:** subjects make their choice $\rightarrow$ x
3. **Stage 2:** results
   - compute subjects' payoffs (in a program, at the beginning of the stage)
   - show each subject his/her payoff on the screen (in the "Active screen" of the stage)

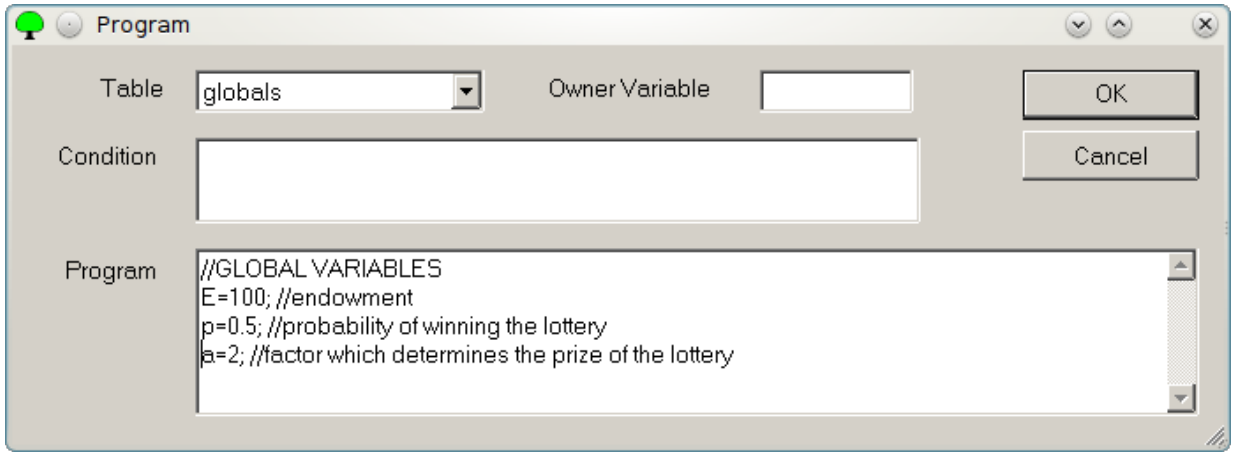
Background


Background
globals
subjects
summary
contracts
session
logfile
globals.do { ... }
//GLOBAL VARIABLES
E=100; //endowment
p=0.5; //probability of winning the lottery
a=2; //factor which determines the prize of the lottery
subjects.do { ... }
//SUBJECTS VARIABLES
x=-1; //money invested in the lottery
n=random(); //random number
Active screen
Header
Waitingscreen
Text
Please, wait.
set parameters
initialize subject variables
background screen

Programs - I

Set the global variables

- select the last table listed in the Background (`logfile`)
- from the menu, choose `treatment`→ `new program` (crtl+alt+p)

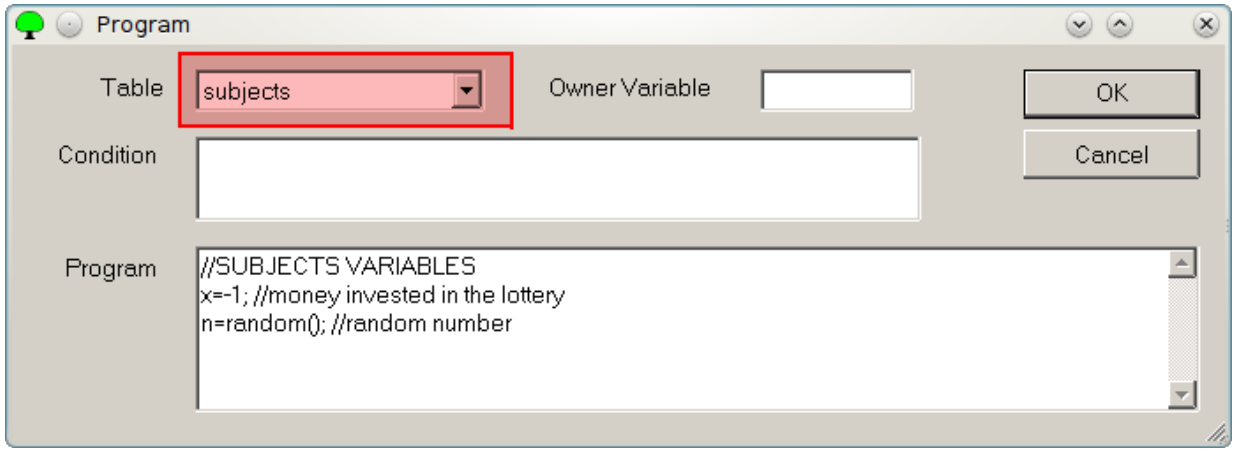


- choose "globals" in the field `Table`
- use // to insert comments

Programs - II

Set the "subject" variables

- Create another program which runs on the subjects table



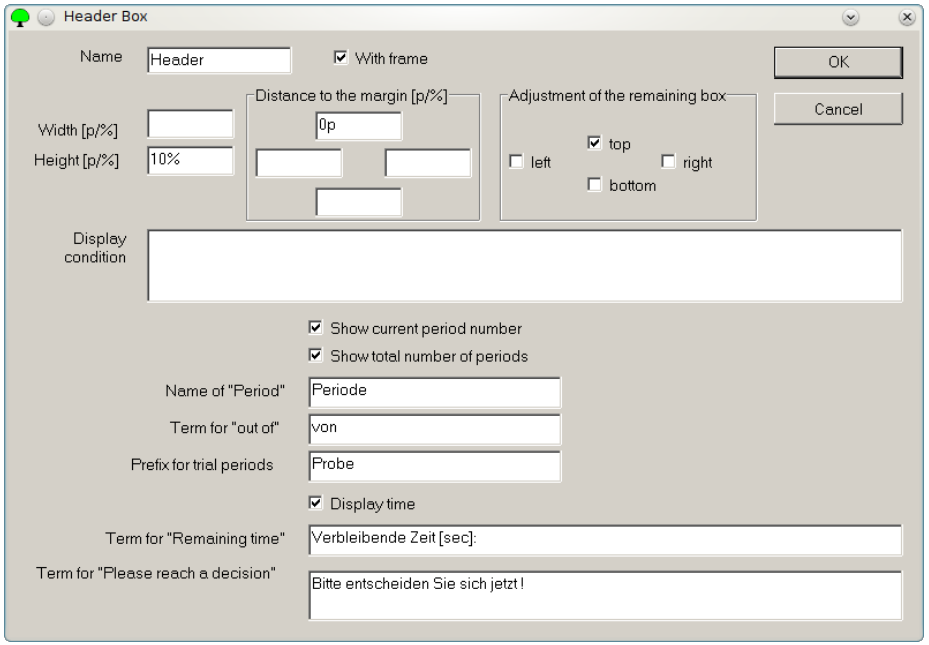
Note: here we use the function `random()`.

- This generates a random number from a uniform distribution between 0 and 1.
- The random number will be *different* for *different subjects*.

Background screen

The `Background` also contains some information about what is displayed on subjects' screen.

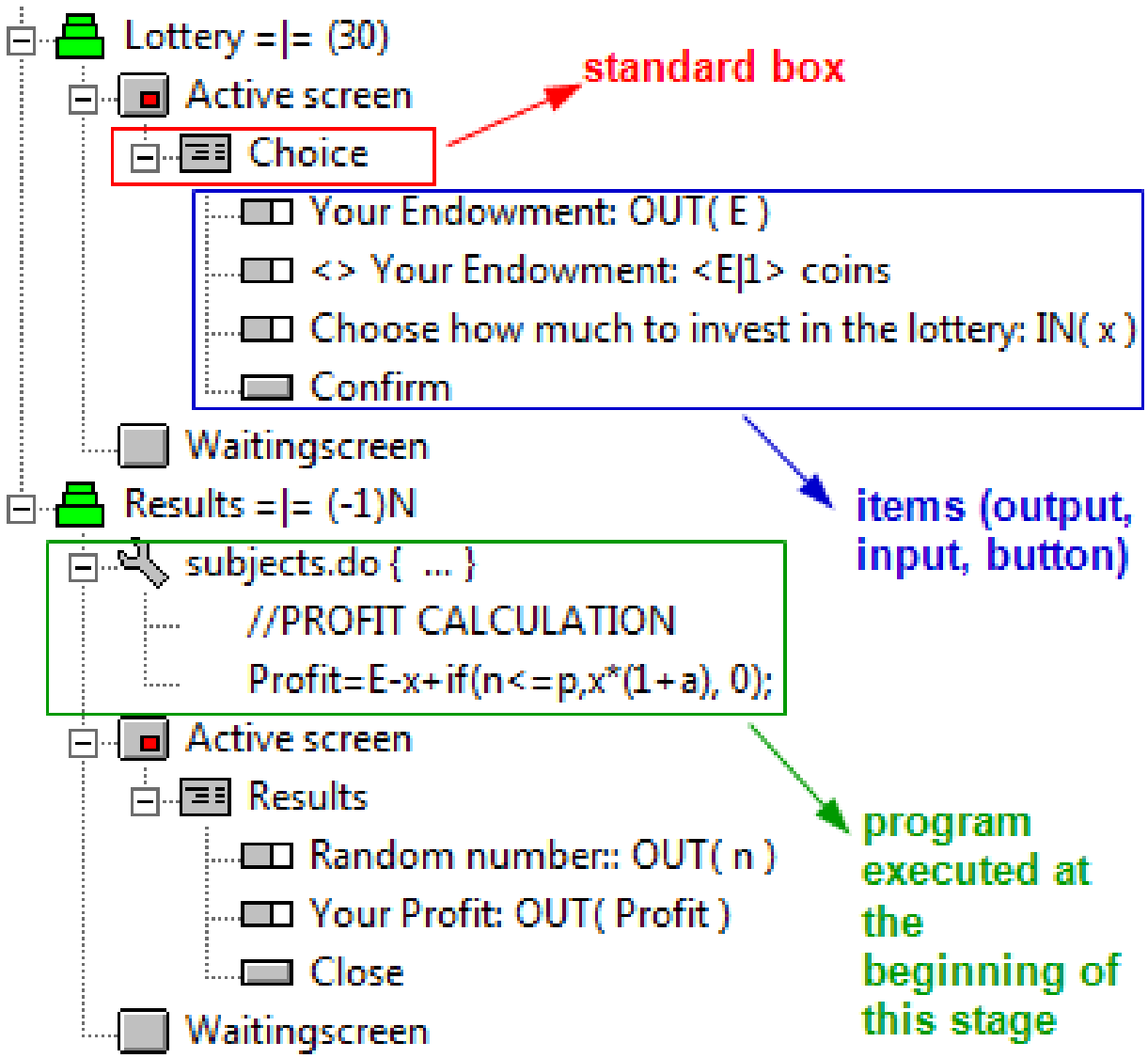
In the **Active screen** a `header box` is placed by default.



- the `header box` defines the top of the screen.
- to remove the header delete the `header box` from the `Active screen` in the `Background`

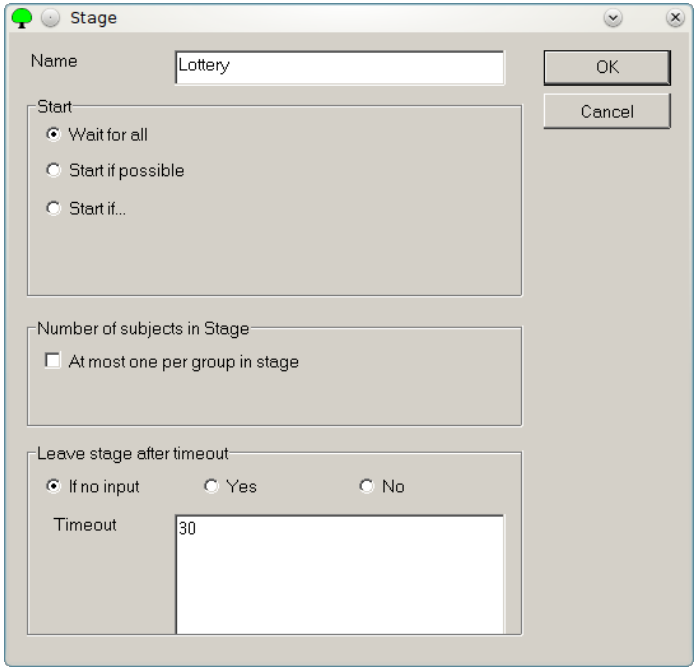
The **Waiting screen** contains the message shown to subjects when they have to wait.

Stages



Create a new stage

Select the `Background` icon, then from the menu, choose: `Treatment` → `New Stage` (ctrl+alt+s)



- Choose a meaningful name for the stage (e.g. "Lottery")
- Set the **timing** (wait for all, start if possible, ...)
- Set the **timeout**: choose "No" if you wish to record the time without forcing subjects to leave the stage

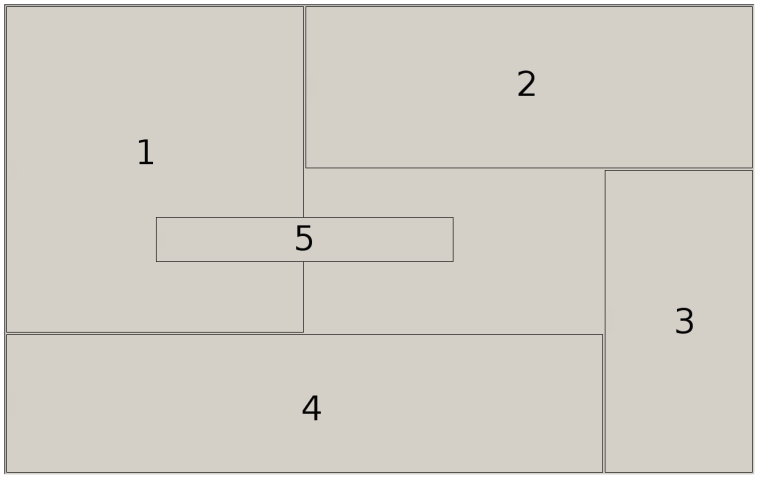
Create a second stage named "Results".

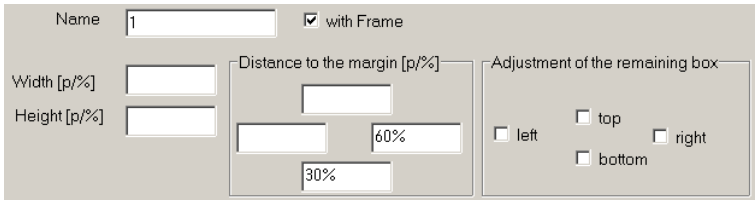
the Active screen

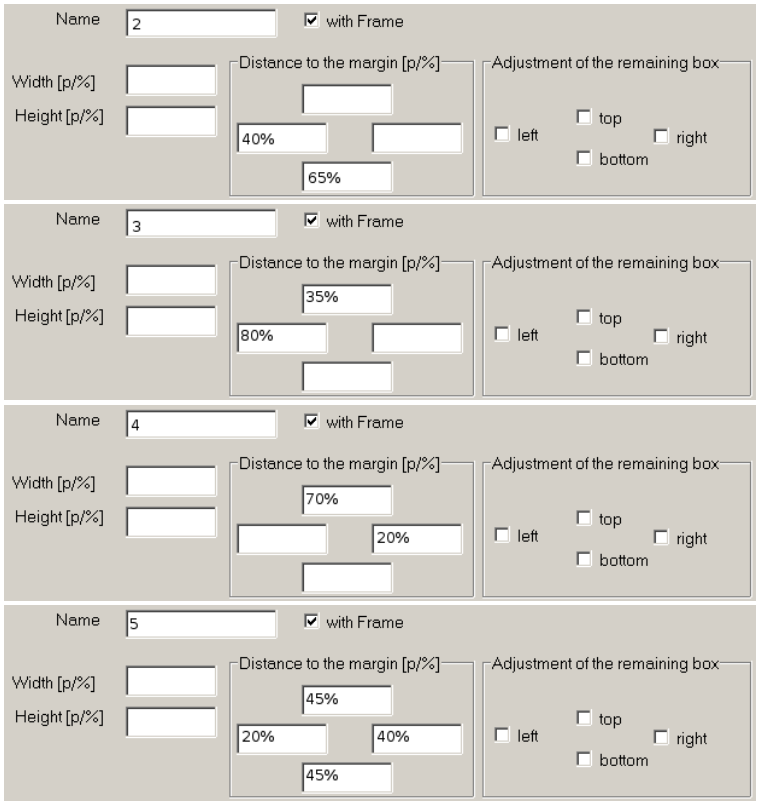
- the `Active screen` of a Stage defines the information displayed on subjects' screens
- If you do not want to use the background screen in some stages, click on `Active screen` and unmark "Background screen is being used"
- the `Active screen` contains `Boxes`
  - i.e. rectangles that organize the space on the screen
- `Boxes` contain `Items`, which can be
  - pieces of information
  - input fields
  - figures
  - etc.

Screen layout

- Set the dimension and position of each box.
- Boxes can overlap

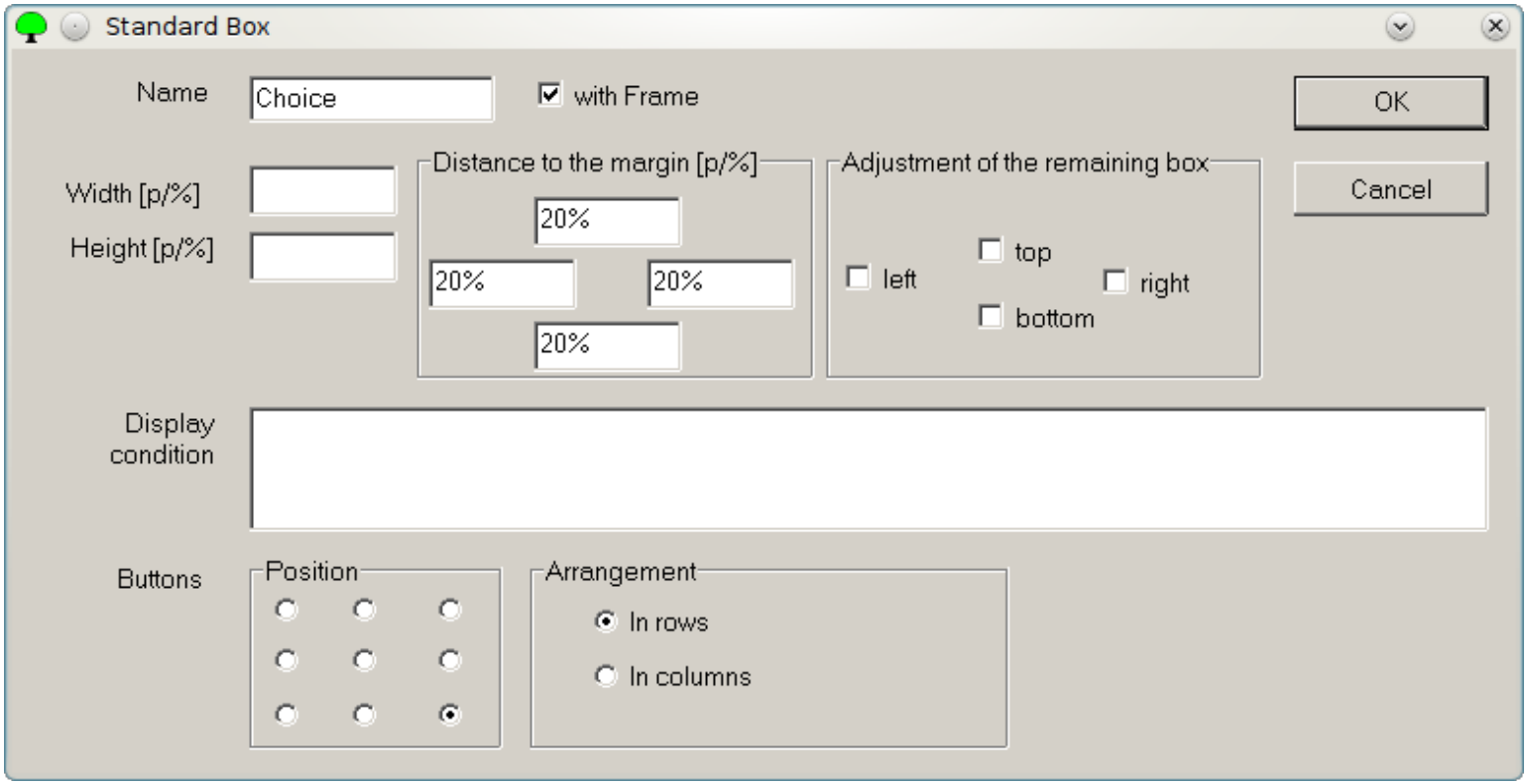






Create a Box

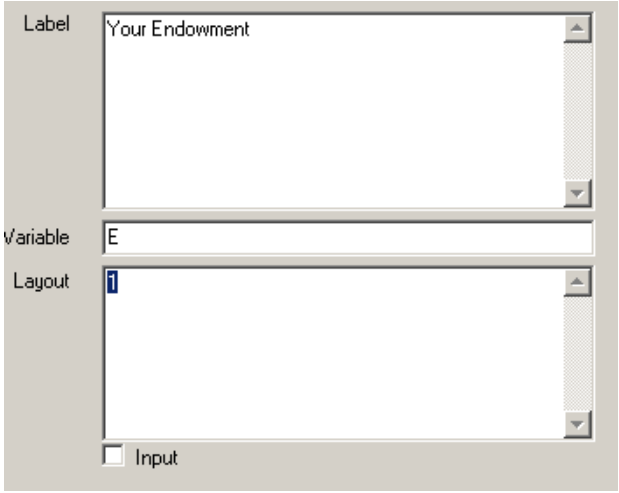
- Select `Active Screen` in the "Lottery" stage.
- From the menu, select `Treatment` → `New Box` → `Standard Box` (ctrl+alt+d)



Output items

Output items: to convey information to the subjects.
Place output items in the "Choice" box

- select the box "Choice" in the "Lottery" stage
- from the menu, select `Treatment`→ `New Item` (ctrl+alt+i)



Output

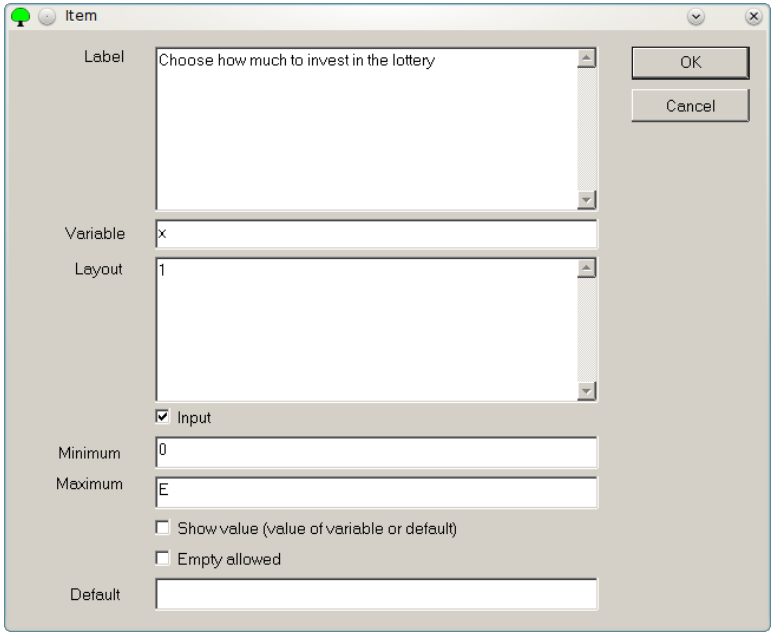
- Label: "Your endowment"
- Variable: E
- Layout: 1 (0.1 for 1 decimal digit...)
- You can use < > to insert variable values into labels:
  Label: "< > Your Endowment: < E|1> coins."

Input items

Input items: to get inputs from subjects.

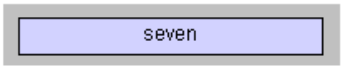
Create a new Item (ctrl+alt+i)

- Label: "Choose how much to invest in the lottery"
- Variable: x
- Layout: 1 (0.1 for 1 decimal digit...)
- **check the** `Input` **box**
- Minimum: 0
- Maximum: E



Note: you can use variables here!

Item layout

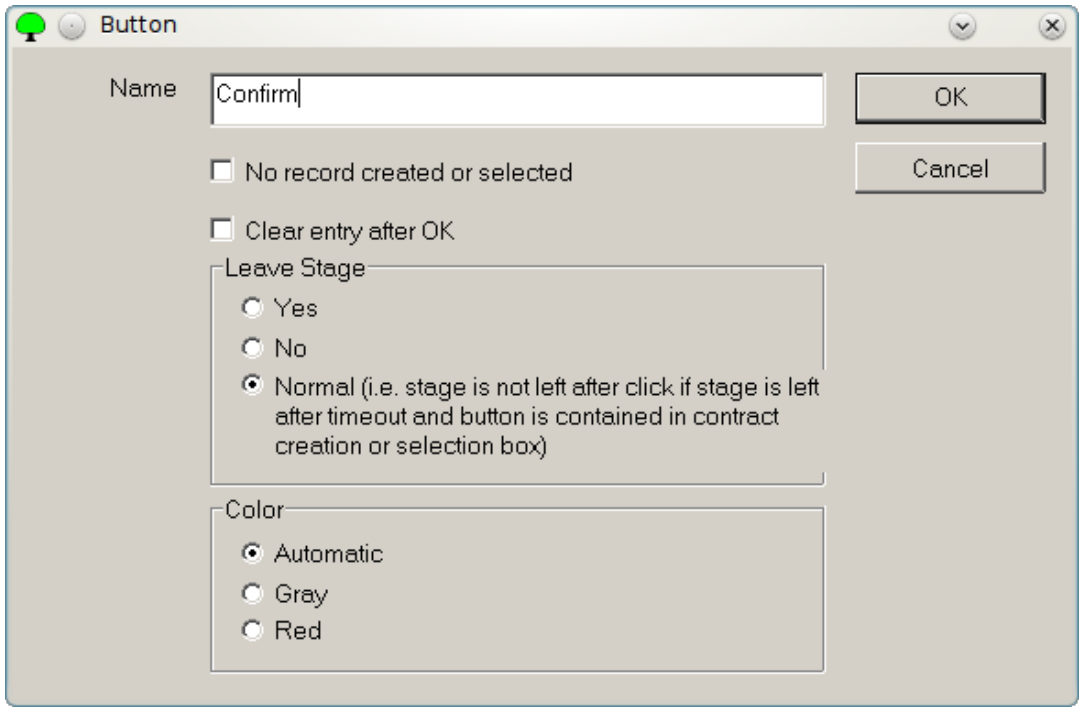
Layout	input variable	output variable
2		
!radio: 1 = "86.8"; 24 = "102.8";		
!radioline: 0="zero";5="five"; 6;		
!slider: 0 ="A"; 100= "B"; 101;		
!scrollbar: 0="L";100= "R";101;		
!checkbox:1="check me";		
!text: 1= "one"; 2 = "two"; 3 = "three"; 4 = "four"; 5 = "five"; 6 = "six"; 7 = "seven"; 8 = "eight"; 9 = "nine"; 10 = "ten";		
!button: 1 = "accept"; 0 = "reject";		

See the Reference Manual at page 30.

Confirm an input

Inputs are confirmed by clicking on a button.

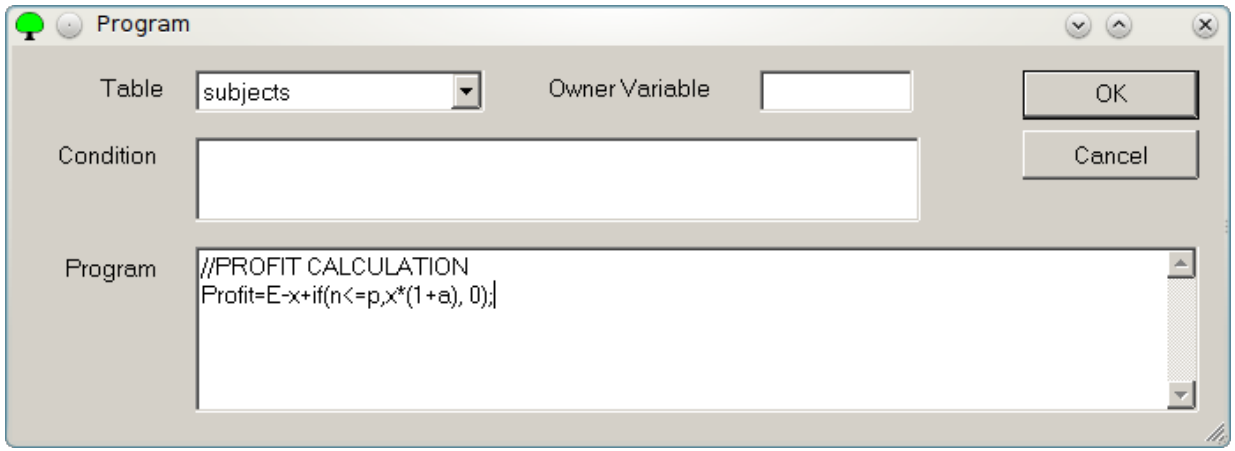
To **add a button** in a box, from the menu, select `Treatment`→ `New Button` (ctrl+alt+b)



Profit calculation

Once the subjects' choices have been collected, the program should compute the profits.

Select the "Results" stage, and create a new program.



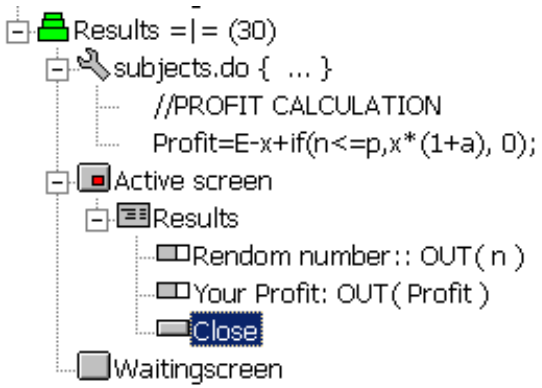
Note: here we use the "if" function:
`if(condition,T,F)` , which returns value

- *T* if the "condition" is true
- *F* if the "condition" is false

A complete list of available functions at page 45 of the Reference Manual.

Results screen

In the `Active Screen` of the "Results" stage, create a new `Standard Box`, and name it "Results".

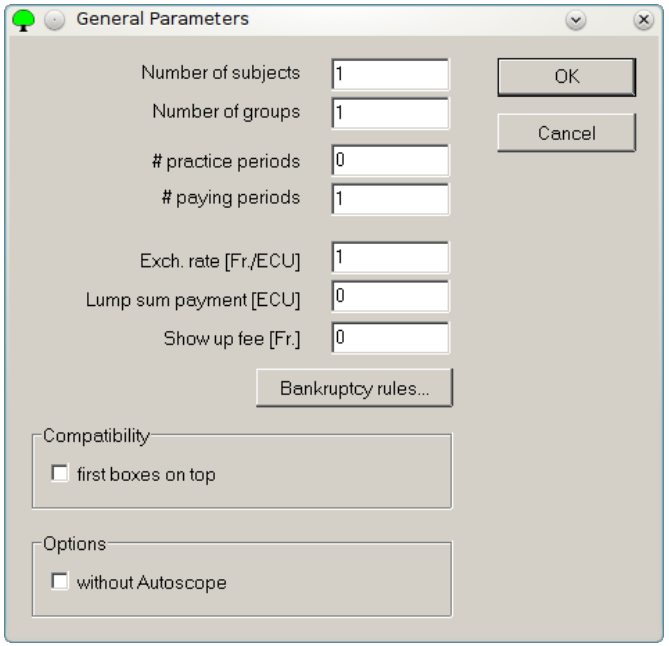


- In this box, add an `Item` to display the **random number** *n*, with 2 decimal digits.
- Add a second Item to display the subject's **profit**.
- Add a Button to end the treatment.

Test the treatment I

To set the **general parameters**:

- Double-click on the `Background` icon (the very first line of the program) to set the general parameters.
- **Exchange rate** $= 1/n$ if n points equal 1 Euro.

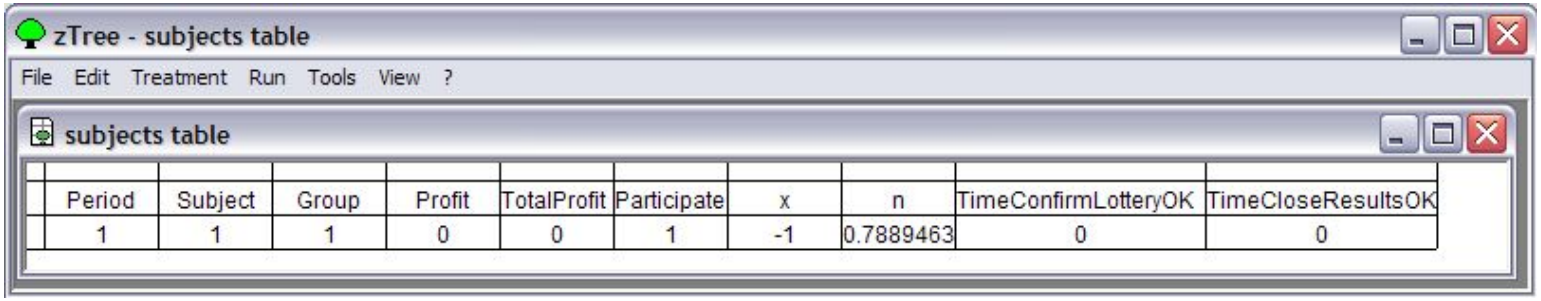


To start the test, launch z-Leaf.exe and check the connection from z-Tree, in the Clients Table.

Test the treatment II

From the z-Tree menu,

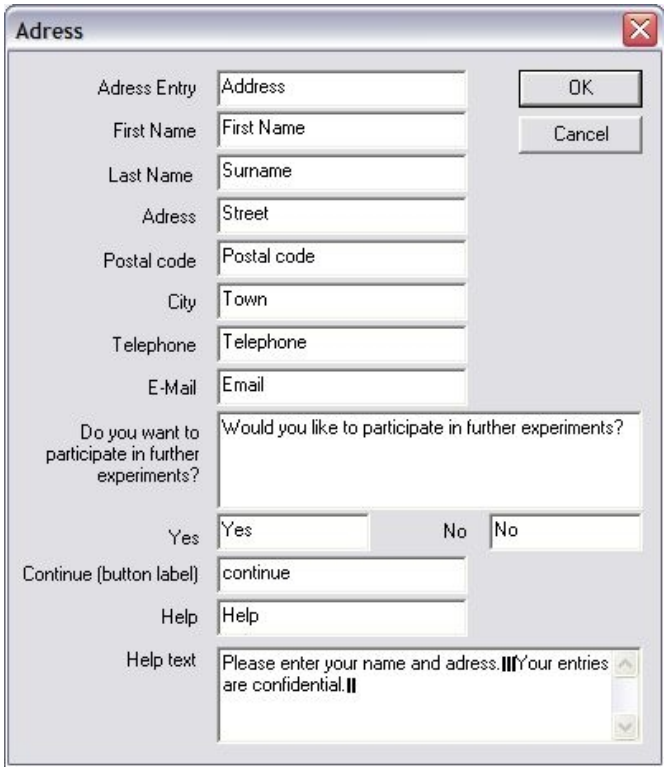
- select `Run`→ `Start treatment` (or press F5)
- select `Run`→ `Stop clock` (or press F12) to stop the time from running,
- select `Run`→ `Restart clock` (or press Shift+F12) to restart the time,
- select `Run`→ `subjects table` to monitor the subjects activity.



Questionnaires

A questionnaire is needed to generate `payment file`.

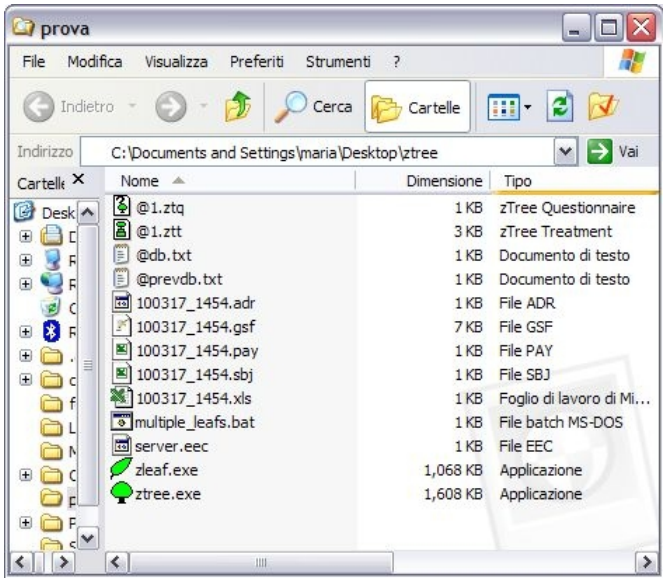
- select `File`→ `New questionnaire`
- select `Questionnaire`→ `Language` and select the language of your choice
- select `Questionnaire`→ `New address form`
- select `Questionnaire`→ `New question form` and name it "Empty"



The `Address Form` is needed, but if you leave the fields `First Name` and `Last Name` empty the Address form will not be displayed. To start the questionnaire, select `Run`→ `Start questionnaire` (or press F5)

Files saved by z-Tree

Session data are saved in the directory containing z-Tree.exe:



.pay: the payment file, which lists the subjects' final profits including the show-up fee

.adr: subjects' addresses (from the Questionnaire)

.sbj: answers to questionnaire's questions, without subjects' names

.gsf: backup file, in case a crash occurs

.xls: contains all tables used in a session (subjects table, globals table, etc)

.pay and .xls files

The **payment file** (.pay):

```
Subject Computer        Interested      Name            Profit          Signature
1       zleaf1          OK              Bigoni, Maria     9.00
2       zleaf2          OK              Doe, John        19.00
3       zleaf3          OK              Smith, James      2.50
4       zleaf4          OK              Johnson, Sarah   16.50
Experiment      Z:\ztree\examples\100319_1112.pay        46.50
```

The **data file** (.xls):

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	100319_1112	1	globals	Period	NumPeriods	RepeatTreatment	E	p	a				
2	100319_1112	1	globals	1	1	0	100	0.5	2				
3	100319_1112	1	subjects	Period	Subject	Group	Profit	TotalProfit	Participate	x	n	TimeConfirmLotteryOK	TimeCloseResultsOK
4	100319_1112	1	subjects	1	1	1	90	90	1	10	0.67	7	99999
5	100319_1112	1	subjects	1	2	1	190	190	1	45	0.48	19	99999
6	100319_1112	1	subjects	1	3	1	25	25	1	75	0.97	12	99999
7	100319_1112	1	subjects	1	4	1	160	160	1	30	0.23	27	99999
8	100319_1112	1	summary	Period									
9	100319_1112	1	summary	1									
10	100319_1112	1	session	Subject	FinalProfit	ShowUpFee	ShowUpFeeInvested	MoneyAdded	MoneyToPay	MoneyEarned	Participate		
11	100319_1112	1	session	1	9	0	0	0	9	9	1		
12	100319_1112	1	session	2	19	0	0	0	19	19	1		
13	100319_1112	1	session	3	2.5	0	0	0	2.5	2.5	1		
14	100319_1112	1	session	4	16	0	0	0	16	16	1		

How to use z-Tree data

Option 1: From the menu select `Tools`→ `Separate tables`. Select the .xls file generated by z-Tree and select `Open`. This generates one .xls file per each of the tables used in the session.

Option 2: for those who use **Stata**, Kan Takeuchi developed an .ado file to import z-Tree data directly into Stata.
`www.econ.hit-u.ac.jp/~kan/research/ztree2stata/index.html`

Option 3: for those who use **R**, Oliver Kirchkamp developed a utility to import data from z-Tree into R. download.
`www.kirchkamp.de/lab/zTree.html`

Exercise 1: guess

see z-Tree Tutorial, page 25.

Subjects have to make a calculation.
They have to calculate the sine function for a randomly determined value: sin(C*A/B)
Subjects are paid according to the precision of their guess:

- Profit (in points):
  $100 * (1 - |sin(C * A/B) - guess|)$
- A and B are random integer numbers between 1 and 10. They take different values for different subjects.
- C is equal to π (3.14...)

Exercise 1: details

- Subjects cannot make negative profits (set the profit function accordingly).
- Do not include the header, nor the alert frame.
- Layout of the input variable: slider.
- Functions you need to use:
  - `abs()`: absolute value
  - `max(x,y)`: returns the maximum value between x and y
  - `pi()`: returns 3.1415...
  - `roundup(x,y)`: returns the smallest multiple of y that is greater or equal to x.
  - `sin(x)`: returns the sine of x.

Solution: exercize_1.ztt

Exercise 1 - layout

Part III

One-shot Two-player Games

One-shot Two-player Games

One shot games

- Matching
- Table functions
- Programming tools
  - The scope operator
  - `if` as function/as statement

Exercise 2

Interactive experiments

The subject's payoff also depends on the decisions of other subjects.
Example: **Prisoner's dilemma**

	cooperate	**defect**
cooperate	3,3	0,5
defect	5,0	1,1

Hint: use **parameters** to make your program more flexible.

	cooperate	**defect**
cooperate	r,r	s,t
defect	t,s	p,p

PD – steps

1. create pairs
2. set parameters: r, t, s, p
3. initialize variables:
   - partner
   - choice
   - partnerchoice
4. subjects' choice
5. profit calculation
6. results

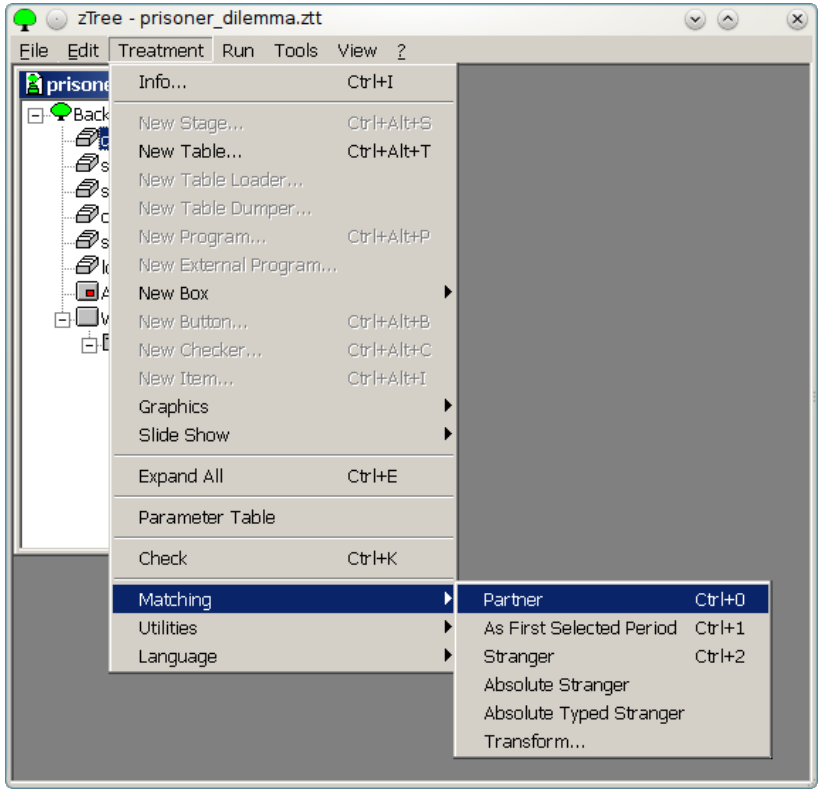
See prisoner_dilemma.ztt

1. Create pairs - a simple matching procedure

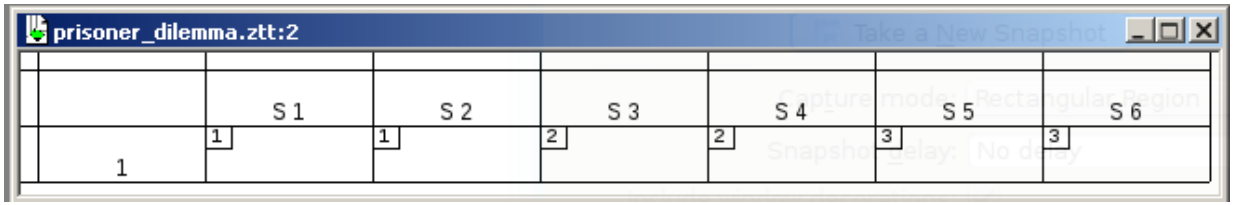
In the `Background` set:

- number of subjects=10
- number of groups=5

Then from the menu select `Treatment` → `Matching` → `Partner`.



To **check the matching**, from the menu select `Treatment`→ `Parameter table`.



2. and 3. Parameters and variables

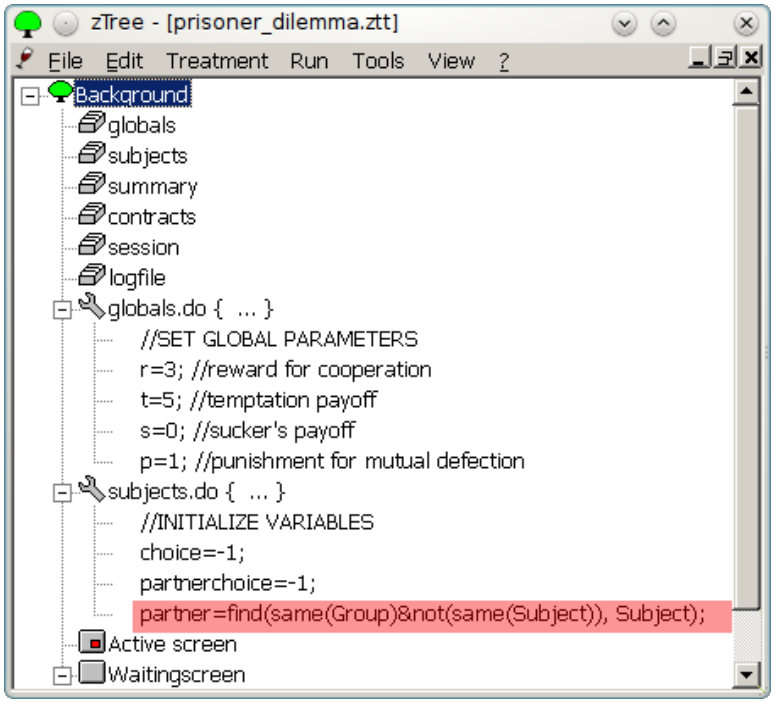


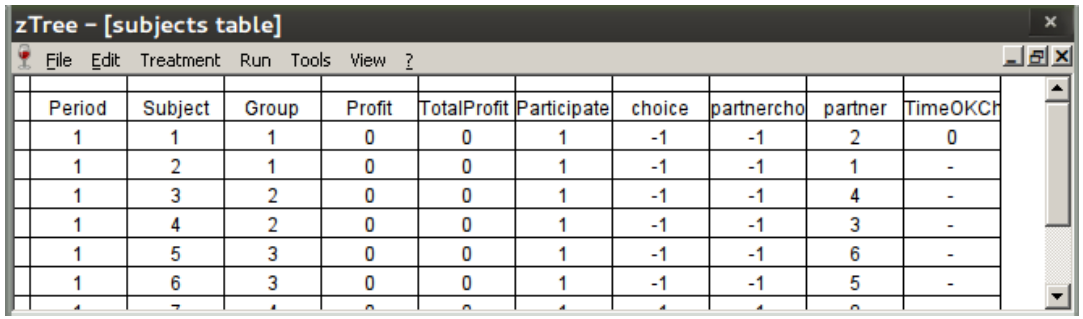
Table functions

The function `find()` is a table function, i.e. a function which does not only refer to a single record of a table, but runs over the **whole table**.

A complete list of table functions can be found at page 46 of the Reference Manual.

Find the partner

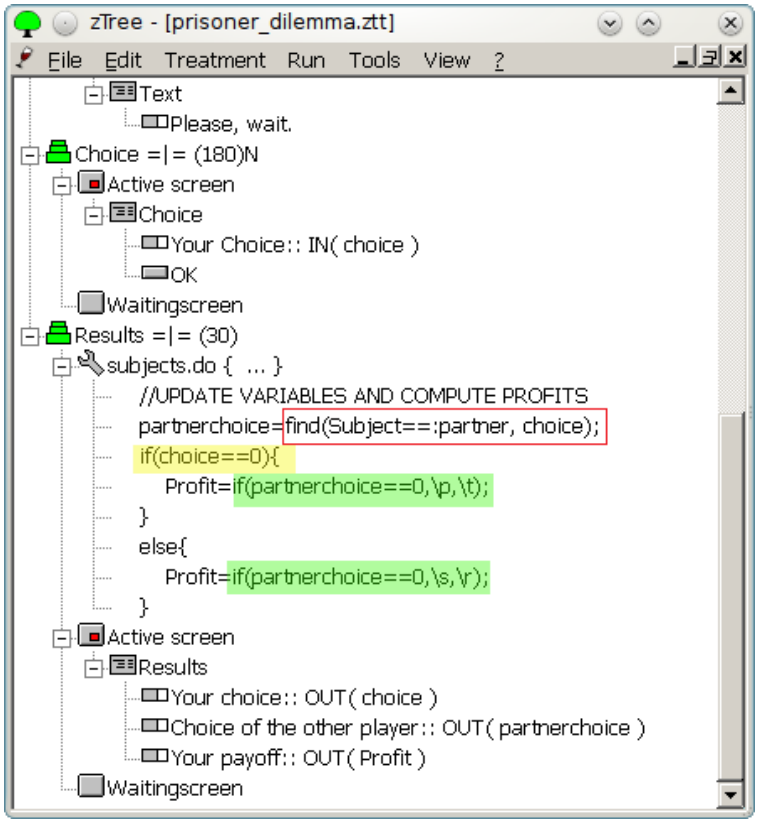
```
partner=find(same(Group)
             &not(same(Subject)),Subject)
```



Period	Subject	Group	Profit	TotalProfit	Participate	choice	partnercho	partner	TimeOKCh
1	1	1	0	0	1	-1	-1	2	0
1	2	1	0	0	1	-1	-1	1	-
1	3	2	0	0	1	-1	-1	4	-
1	4	2	0	0	1	-1	-1	3	-
1	5	3	0	0	1	-1	-1	6	-
1	6	3	0	0	1	-1	-1	5	-

- `find(condition, x)` looks at all the records in a table, from top to bottom, and returns the value of variable `x` of the first record in which condition is TRUE.
- `same(x)` is TRUE for all records in the table, where the variable (or expression) `x` takes the same value it has in the "reference record".
- `not(same(x))` is TRUE when `same(x)` is FALSE and vice versa.

4. 5. and 6. Choices and results



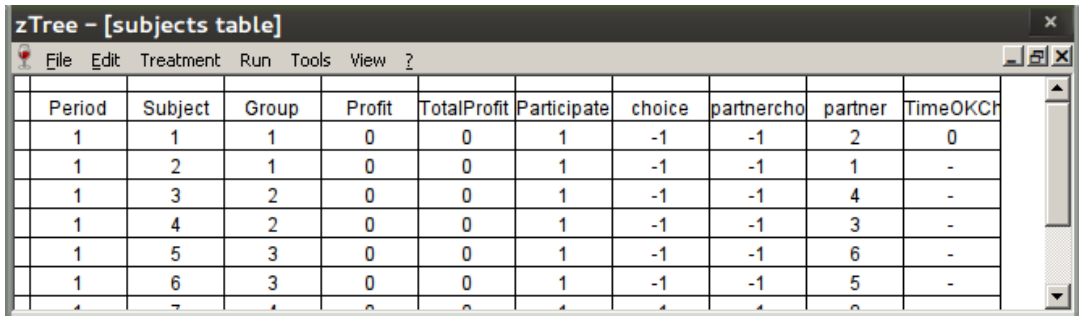
4. **subjects' choice:** with radiobuttons: `!radio:0="D";1="C";`
5. **profit calculation:** scope operator (:) and if as a function and as a statement. See page 44 of the Reference manual.
6. **display results:** layout `!text:0="D";1="C";`

The scope operator

see page 27 of the z-Tree Tutorial

```
partnerchoice=find(Subject==:partner, choice)
```

The scope operator ":" indicates that the variable `partner` belongs to the record in which the cell `partnerchoice` lies, not to the record where `Subject` lies.



Period	Subject	Group	Profit	TotalProfit	Participate	choice	partnercho	partner	TimeOKCh
1	1	1	0	0	1	-1	-1	2	0
1	2	1	0	0	1	-1	-1	1	-
1	3	2	0	0	1	-1	-1	4	-
1	4	2	0	0	1	-1	-1	3	-
1	5	3	0	0	1	-1	-1	6	-
1	6	3	0	0	1	-1	-1	5	-

if as function/as statement

```
if(condition a){c1}\\
elsif(condition b1){c2}\\
elsif(condition b2){c3}...\\
else{c4}
```

if as a statement

- If condition a is TRUE, then command(s) c1 is (are) executed;
- if condition a is FALSE and condition b1 is TRUE, then command(s) c2 is (are) executed; (etc.)
- if conditions a, b1 and b2 are FALSE, then command(s) c4 is (are) executed;

if as a function

```
if(condition a, x, y)
```

returns x if condition a is TRUE, returns y otherwise.

Exercise 2: the Traveler's dilemma

The Traveler's dilemma is a simple two-players game.

- each player can choose a number **between 2 and 100**
- the choice is simultaneous
- the player's **profit** is:
  - equal to the number he chose, if this number is *equal* to the number chosen by his partner
  - equal to the number he chose $+2$, if this number is *lower* than the number chosen by his partner
  - equal to the number he chose -2, if this number is *higher* than the number chosen by his partner

Part IV

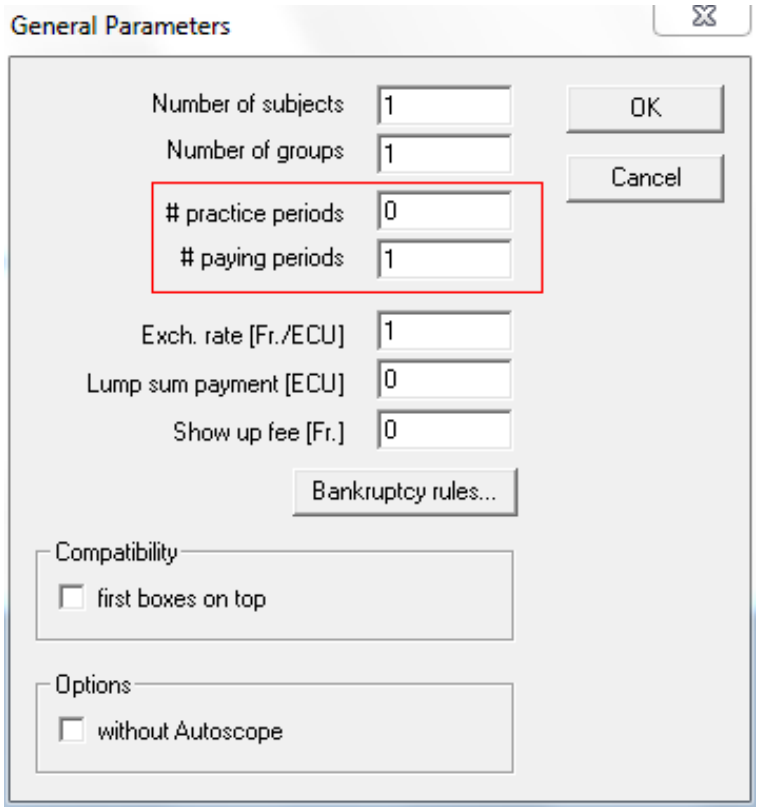
Repeated Two-player Games

Repeated Two-player Games

Symmetric games with finite horizon

Increase the number of periods:

- double-click on `Background`
- set the number of periods and trial periods



See prisoner_dilemma_2.ztt

Matching in repeated games - I

When you have more than 1 group and multiple periods, there are several possible matching procedures.

Option 1: from the menu, choose `Treatment` → `Matching`:

- partner: groups remain the same throughout the whole treatment
- stranger: groups are randomly formed at the beginning of each period
- absolute stranger: players never meet more than once in the treatment (not always possible)

To **check the matching**, from the menu select `Treatment`→ `Parameter table`.

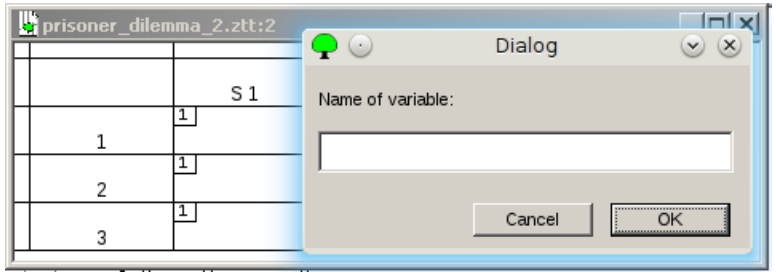
Matching in repeated games - II

Option 2: load the matching from an external table.

1. With MS Excel (or similar) create a tab-separated .txt file ("save as" → "other formats").

	A	B	C	D
1	1	1	2	2
2	1	2	1	2
3	1	2	2	1

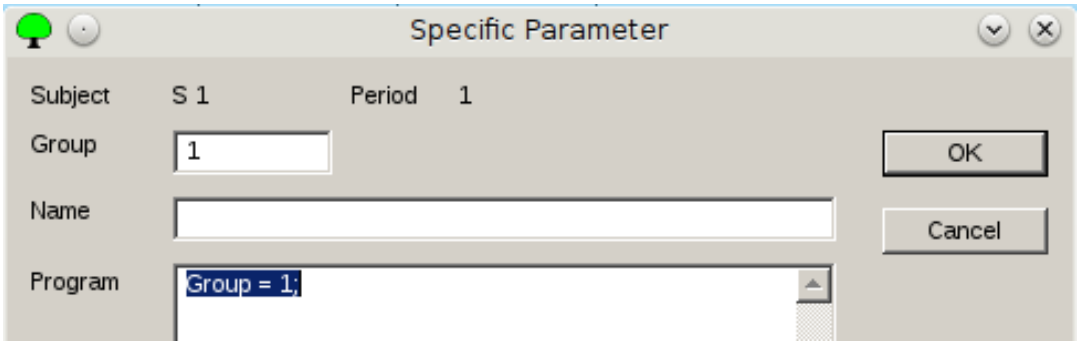
2. Then, from z-tree, open the `Parameter Table`
3. Select `Treatment` → `Import Variable Table`.
4. Set `Name of variable = Group`.



5. Press OK and select the matching table (tab separated .txt file).

Matching in repeated games - III

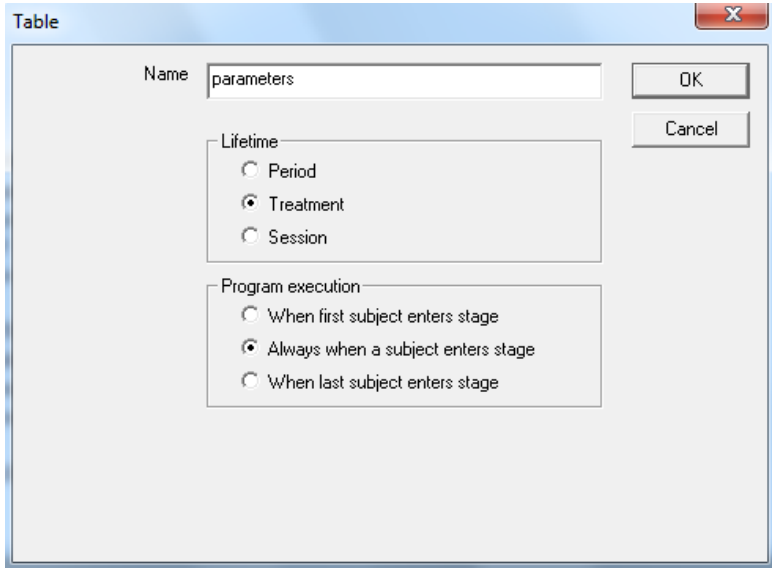
If you double-click on one of the cells of the parameter table, you'll see a program defining the `Group` variable, for a specific subject in a given period.



Programs in the parameter table are executed *after* those in the background.

Changing parameters across periods -1

1. in z-Tree, select the `Background` and choose `Treatment`$\rightarrow$ `New table`.



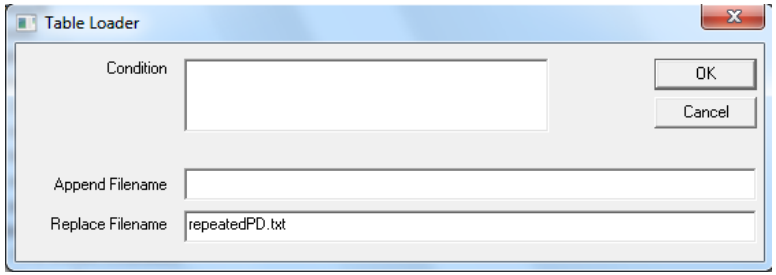
2. create a new table (with Excel, or similar programs)

	A	B	C	D	E	F
1	parameters	Period	r	t	s	p
2	parameters	1	3	5	0	1
3	parameters	2	3	6	0	1
4	parameters	3	4	5	0	1

3. save the table as a *tab delimited* .txt file
4. the first column reports the name of the table you want to load.

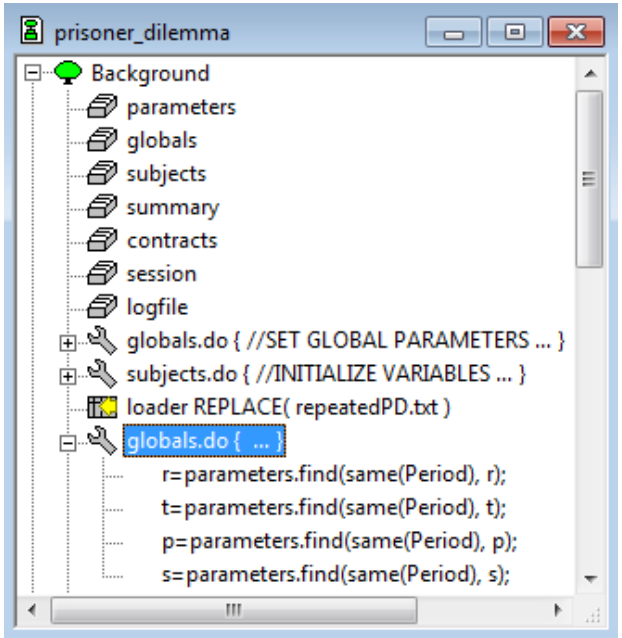
Changing parameters across periods - 2

5. in z-Tree, select the `Background` and choose `Treatment`$\rightarrow$ `New table loader`.



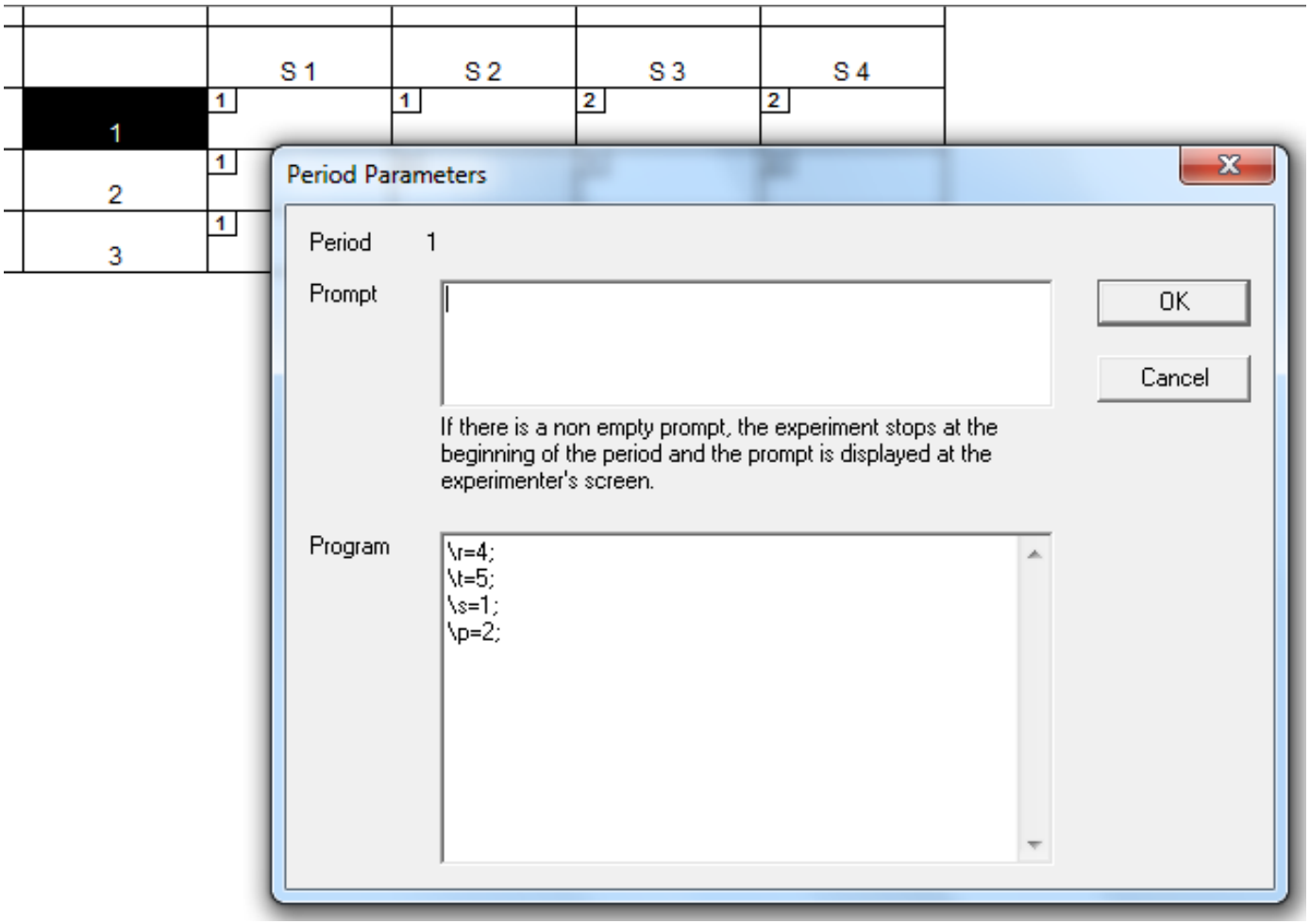
6. Append/Replace filename: name of your .txt file

7. in the Background, after the `Table Loader` create a new `Program`, to load the new parameters.



Changing parameters across periods - 3

An alternative possibility is setting parameters manually, adding programs in the `Parameter Table`.



Grid boxes and Container boxes - I

Now the payoff matrix of the game changes every period. You might want to show it on the subjects' screens.

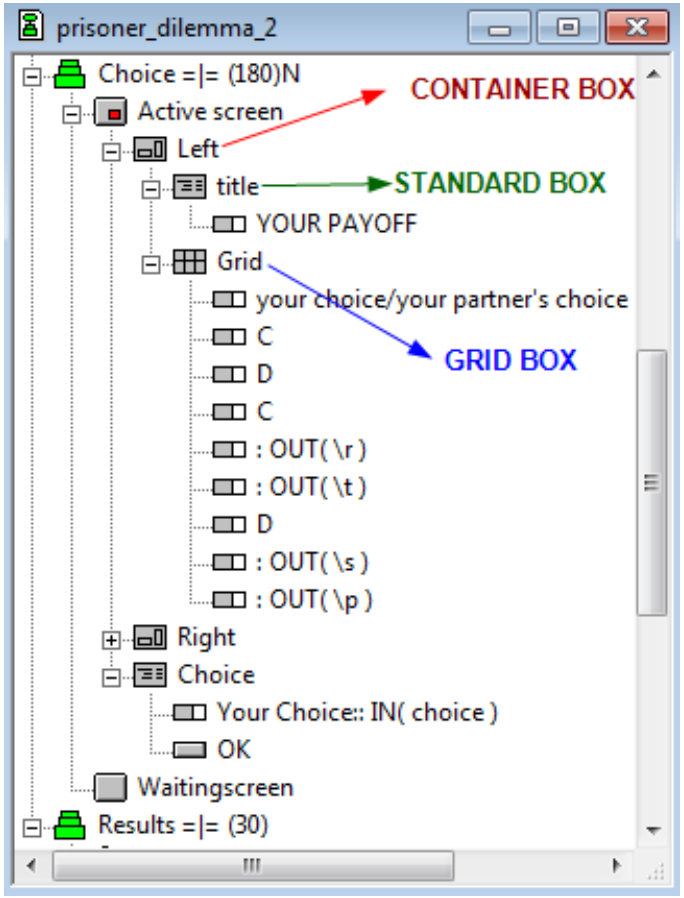
YOUR PAYOFF		
your choice/your partner's choice	C	D
C	3	0
D	5	1

Your Choice: D C

OK

YOUR PARTNER'S PAYOFF		
your choice/your partner's choice	C	D
C	3	5
D	0	1

Grid boxes and Container boxes - II



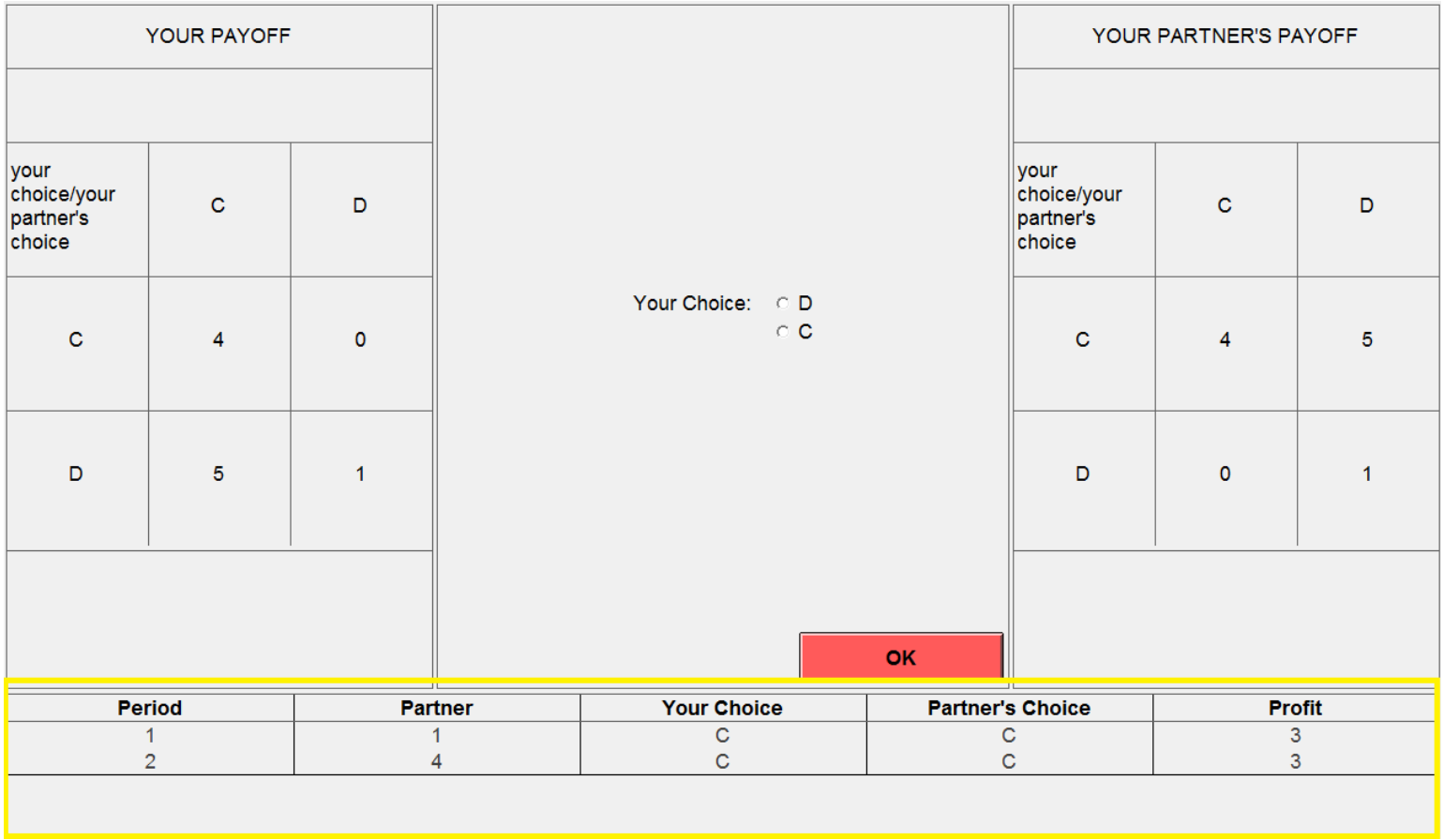
A `container box` is a box that can contain other boxes.

Note: the *relative measures* of the boxes it contains are defined *w.r.t. the container box*, not to the whole screen.

A `grid box` presents items in a tabular form.

History box - I

To remind subjects about what happened in past periods, you can insert a `history box`.



YOUR PAYOFF

your choice/your partner's choice	C	D
C	4	0
D	5	1

Your Choice: D C

OK

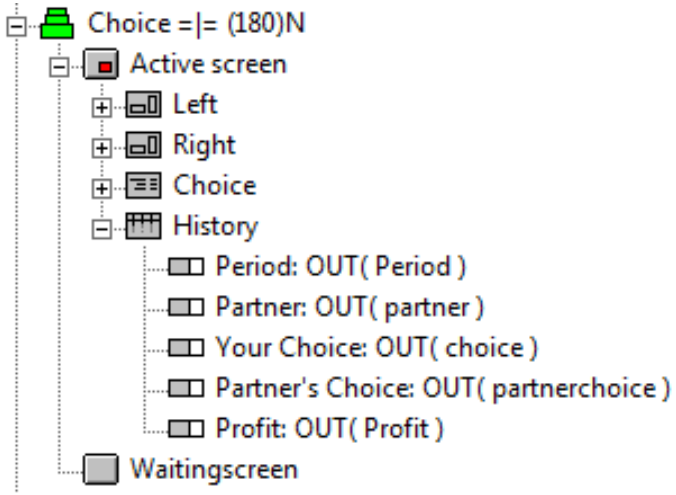
YOUR PARTNER'S PAYOFF

your choice/your partner's choice	C	D
C	4	5
D	0	1

Period	Partner	Your Choice	Partner's Choice	Profit
1	1	C	C	3
2	4	C	C	3

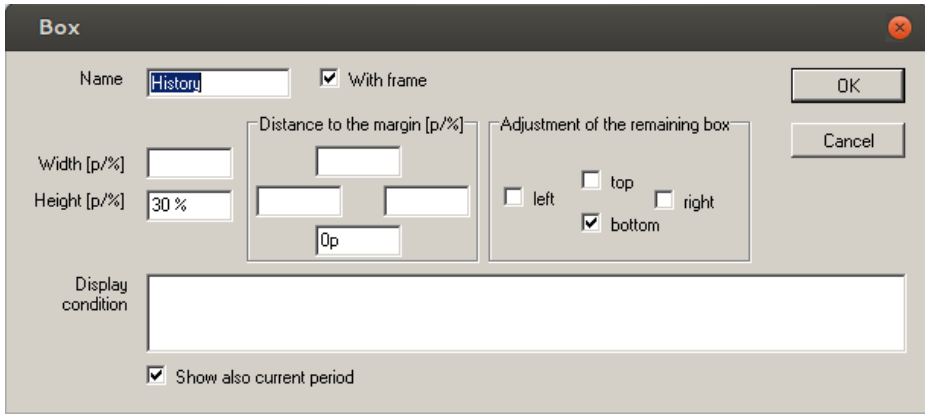
History box - II

see the Reference Manual, page 18.



- lists results from previous periods
- only takes variables form the `subjects` table
- a label row contains the labels

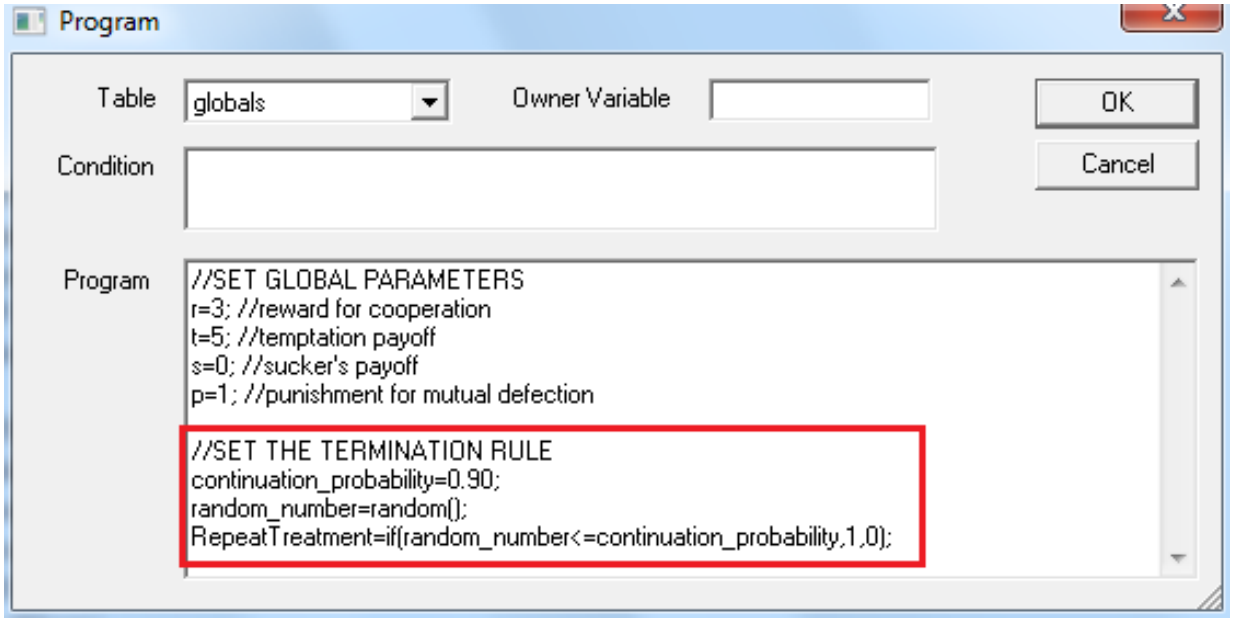
- if the table is too long, a scroll-bar appears



- Option: showing/not showing the current period.

Indefinite number of periods

1. Define a treatment with a single period.
2. In the `Background`, create a new program which runs on the `globals` table. In this program, you can set the variable `RepeatTreatment`, as follows:



Random matching in programs - I

Hard to use the `parameter table` or to import an external matching table.

$\Rightarrow$ Matching **in a program** in the `Background`, running on the `globals` table (it runs only once, at the beginning of each period).

- generate a different random number (rand) for each subject;
- generate the variable rank, which sorts subjects according to the variable rand;
- group together subject having consecutive ranks.

subjects table

Period	Subject	Group	Profit	TotalProfit	Participate	i	rand	rank
1	1	2	0	0	1	2	0.3314207	3
1	2	2	0	0	1	2	0.6126384	4
1	3	1	0	0	1	2	0.2980133	2
1	4	1	0	0	1	2	0.1228111	1

Random matching in programs - II

```
//CREATE GROUPS
i=1;
repeat{
  i=i+1;
  subjects.do{
    rand=random();
  }
  subjects.do{
    rank=subjects.count(rand<=:rand);
  }
}
while(subjects.sum(Subject)!=subjects.sum(rank)&i<10);
subjects.do{
  Group=roundup(rank/2,1);
}
```

Example: random_matching_program.ztt

do statement and loops

`do` **statement**:
see page 62 of the Tutorial

With `do{commands}` the commands are executed *for all records* in the current table.

For example, with `subjects.do{commands}`
we specify that the commands should be executed for all records in the `subjects` table.

`repeat{commands} while {condition}` **statement**

The commands are executed. *Then* it is checked whether the condition is TRUE, and as long as it is the commands are repeated.

Loops can be left with the key combination Ctrl+Alt+F5

Different games in different periods I

When you don't know in advance the length of a treatment, you can:

1. **randomize**:

```
//LOAD PARAMETERS FROM THE IMPORTED TABLE
n=max(1,roundup(3*random(),1));
//random number equal to 1, 2 or 3.
r=parameters.find(Period==\n, r);
t=parameters.find(Period==\n, t);
p=parameters.find(Period==\n, p);
s=parameters.find(Period==\n, s);
```

Different games in different periods II

2. **rotate:**

```
//LOAD PARAMETERS FROM THE IMPORTED TABLE
n=if(mod(Period,3)==0,3,mod(Period,3));
//number taking value 1, 2 or 3, in turn.
r=parameters.find(Period== \n, r);
...
```

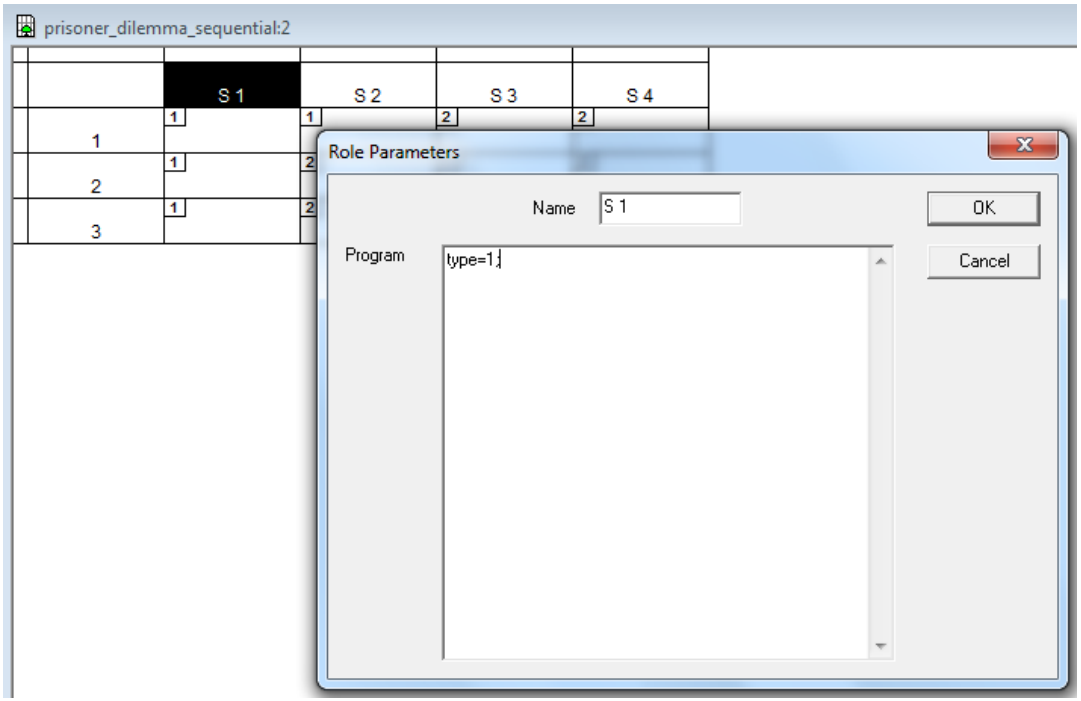
Note: setting parameters in the Parameter Table is less flexible.

Games with asymmetric players - I

Suppose now you want to let subjects play the prisoner dilemma *sequentially*.

2 types of players: 1 (first mover) and 2 (second mover)$\Rightarrow$ `type` is a subject variable.

If types remain fixed across periods, you can set them in the `Parameter Table`.

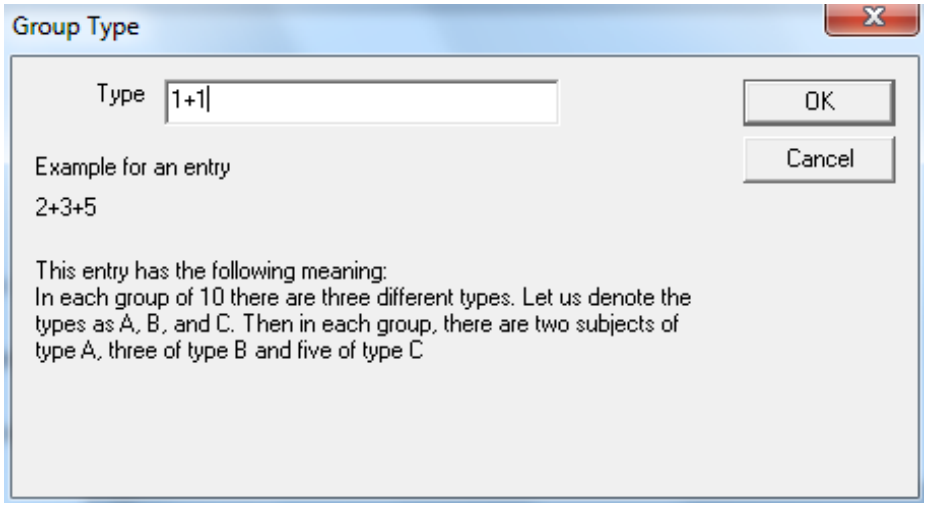


Games with asymmetric players - II

You can also assign types with a **program** in the `Background` (on the `globals` table).

```
numsubjects=subjects.maximum(Subject);
subjects.do{
  type=if(Subject<=\numsubjects/2,1,2);
}
```

With types, you can use the `Absolute typed strangers` matching procedure (*Ref.Man., page 32*).
From the menu, select `Treatment` → `Matching`

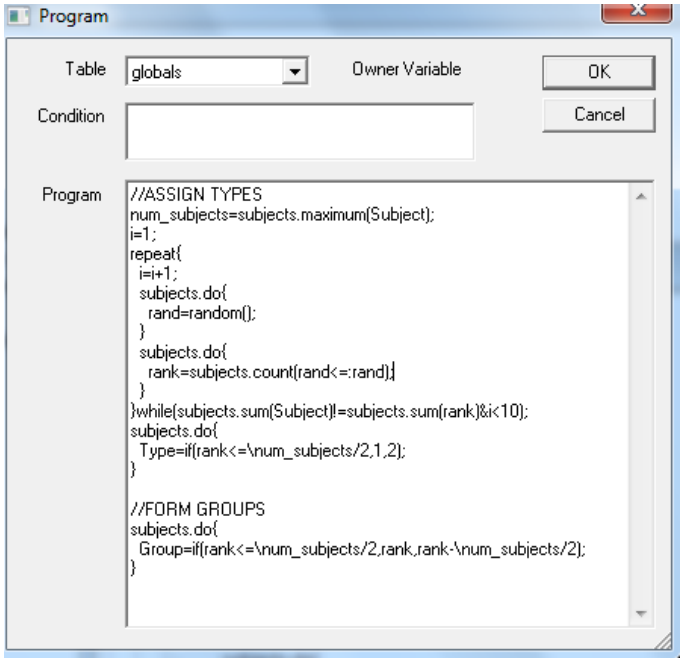


Assign types and groups randomly

See assign_types.ztt

It is possible to set types randomly at the beginning of each period, with a program in the `Background`.

1. rank subjects according to some random variable
2. set `type=1` for the first half of the subjects, `type=2` for the others
3. form groups picking the first subject from the first half, and the second subject from the second half.

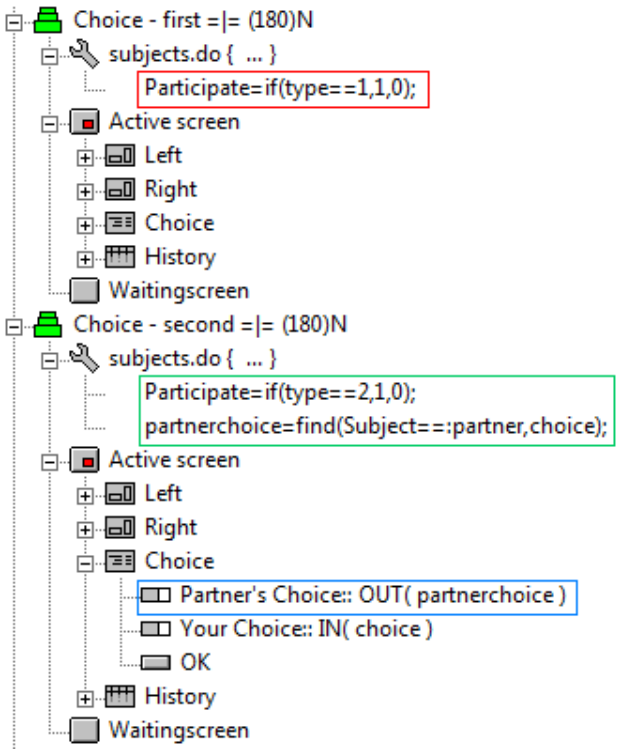


subjects table

Period	Subject	Group	Profit	TotalProfit	Participate	rand	rank	Type
3	1	1	0	0	1	0.7331797	3	2
3	2	1	0	0	1	0.2467885	1	1
3	3	2	0	0	1	0.4047647	2	1
3	4	2	0	0	1	0.9164408	4	2

The Participate variable

Once you have defined the types, you can modify the structure of the program. The `Participate` variable in the `subjects table` determines whether a subjects enters a stage or not.



- If this variable has value 1 then the subjects enters the stage, i.e., the corresponding `Active Screen` appears on the subjects computer screen.
- Programs in the stage are executed *for all* subjects.

See prisoner_dilemma_sequential.ztt

Program execution

At the beginning of each period, **programs** are executed in the following **order**:

1. Standard variables (Subject, Period, etc.) are set.
2. Programs in `Background`, in the order they appear in the .ztt file
3. Programs in the `Parameter Table`

cells	Subject programs (in current period) in the subjects table
top row	Role program in subjects table
first column	Period program in globals table

4. Programs of the first stage

Exercise - battle of the sexes

The "battle of the sexes" is a two-player asymmetric game.

Rob (row player) and Clara (column player) want to go out together tonight. She prefers the box, he is a fan of ballet.

	box	**ballet**
box	3,5	0,0
ballet	0,0	5,3

They will go out together every Tuesday, as long as their engagement lasts.

	cooperate	**defect**
cooperate	m, hc	l, l
defect	l, l	hr, m

Exercise - details - I

- **Indefinitely repeated** game. There is a 10% probability that Rob and Clara break up before next Tuesday.
- **Partner matching** (couples remain the same throughout the treatment). Types are also fixed across periods.
- $l = 0$, $m = 3$, while *hc* **and** *hr* **parameters vary** randomly across periods:
  - sometimes the box match is particularly important: hc=8, hr=5;
  - sometimes the ballet company is very good: hc=5, hr=8;
  - otherwise hc=5, hr=5;

Exercise - details - II

- Simultaneous moves.
- Show a `history box` and two `grid boxes` to display the payoff matrices.
- You can use the file prisoner_dilemma_sequential.ztt as a starting point.

Exercise - screen-shot

YOUR PAYOFF		
your choice/Clara's choice	box	ballet
box	3	0
ballet	0	5

Hi Rob, make your choice: ballet / box

OK

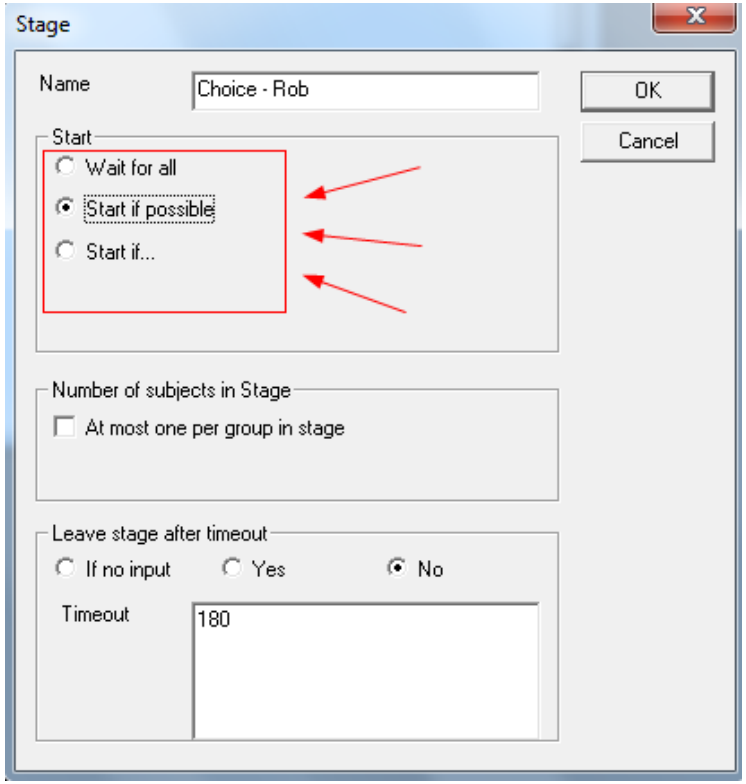
CLARA'S PAYOFF		
your choice/Claras's choice	box	D
box	8	0
ballet	0	3

Period	Partner	Your Choice	Clara's Choice	Profit
1	4	ballet	ballet	8
2	4	ballet	ballet	8
3	4	ballet	box	0
4	4	box	box	3

Exercise - hint

Create 3 stages:

1. Rob's choice stage
2. Clara's choice stage
3. Results



To let the two players play simultaneously, set Rob's choice stage and Clara's choice stage to "`start if possible`".

Solution: battle_of_sexes.ztt

Part V

Multi-player Games: Auctions and Markets

Multi-player Games: Auctions and Markets

Second price sealed-bid auctions

- `checkers`
- `while` statement
- variables into labels

Dutch Auctions

- The `later` statement
- Programs into buttons

Posted-offer markets

- the `contracts table`
  - the `contracts creation box`
  - the `contracts list box`

Double Auctions

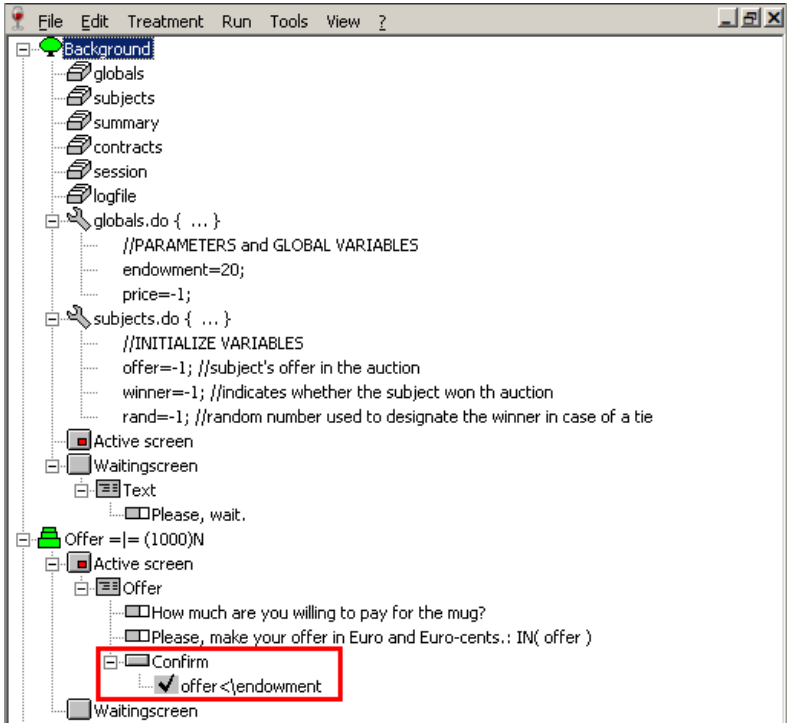
Second price sealed-bid auctions - I

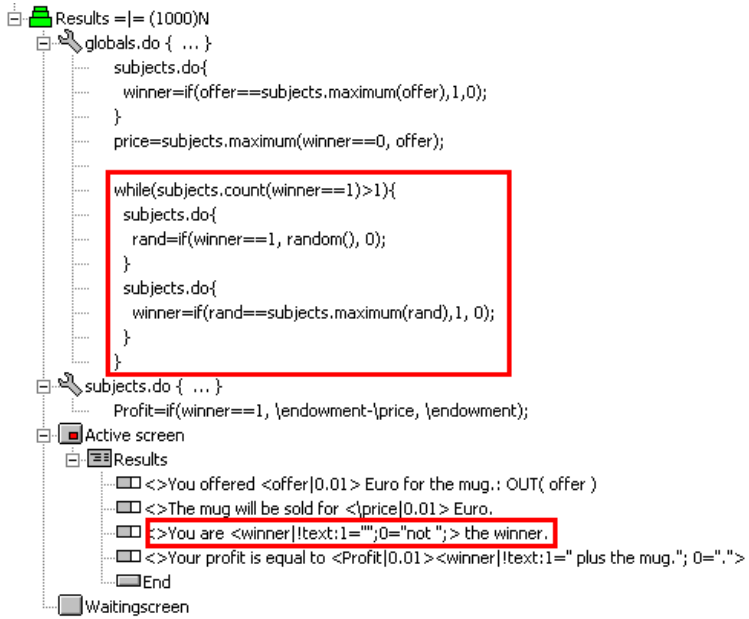
Purpose: elicit the true value of a good (e.g. a mug) for each of the subjects.

- each subject receives an initial endowment (20 euros)
- each subject makes an offer for the mug (possibly even above 20 euros). The offers are secret and simultaneous.
- the subject who makes the highest bid wins the mug and have to pay a price equal to the second highest bid.
- in case of a tie, the winner is drawn at random among the bidders who submitted the highest bid.

See second_price_auction.ztt

Second price sealed-bid auctions - II

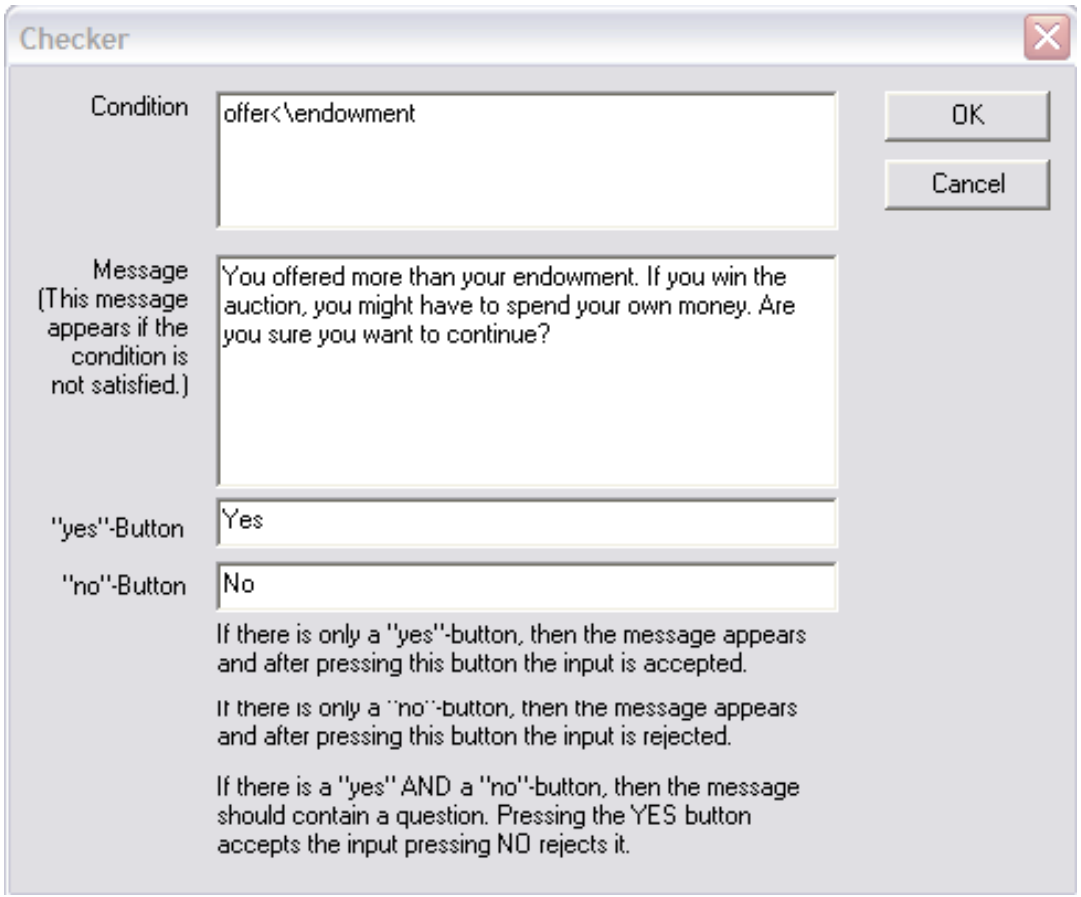




checkers

`checkers` are used to **verify the validity of an input**.

To create a checker, select the button you need to "check", then, from the menu:
`Treatment→New checker`



while statement - I

To sort the winner at random in case of a tie, we use the `while` statement:

```
while(subjects.count(winner==1)>1){
 subjects.do{
  rand=if(winner==1, random(), 0);
 }
 subjects.do{
  winner=if(rand==subjects.maximum(rand),1, 0);
 }
}
```

`while` statement - II

The general use is:

```
while(condition){
  program;
}
```

While the `condition` is TRUE, the program is executed.

Reminder: loops can be left with the key combination Ctrl+Alt+F5.

Variables into labels

In the `Active screen` of the `Results` stage, we tell each participants whether he is the winner.

In the input **label** we write: `<>You are <winner|!text:1="";0="not ";> the winner.`

This becomes:

- "You are the winner", if the variable `winner` is equal to 1
- "You are not the winner", if the variable `winner` is equal to 0

Do not forget the <> sign at the beginning.

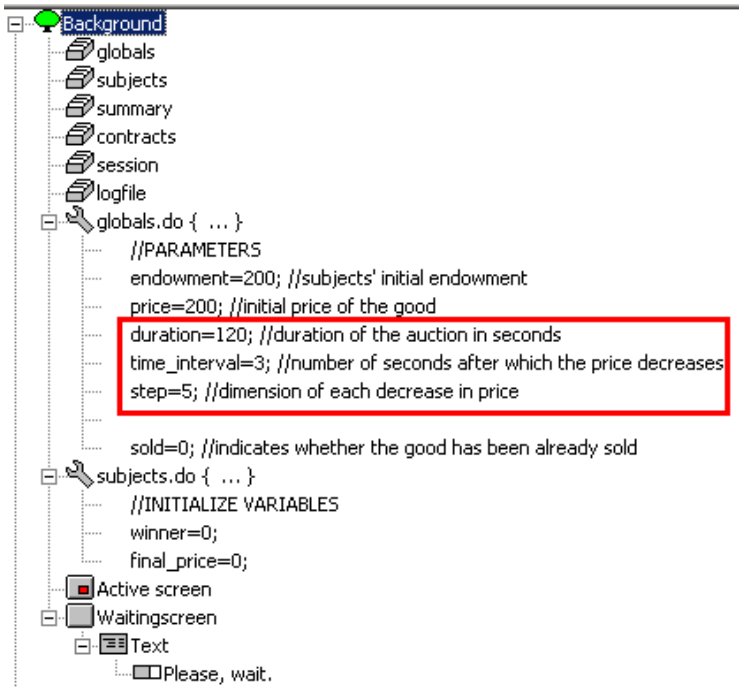
Dutch Auctions - I

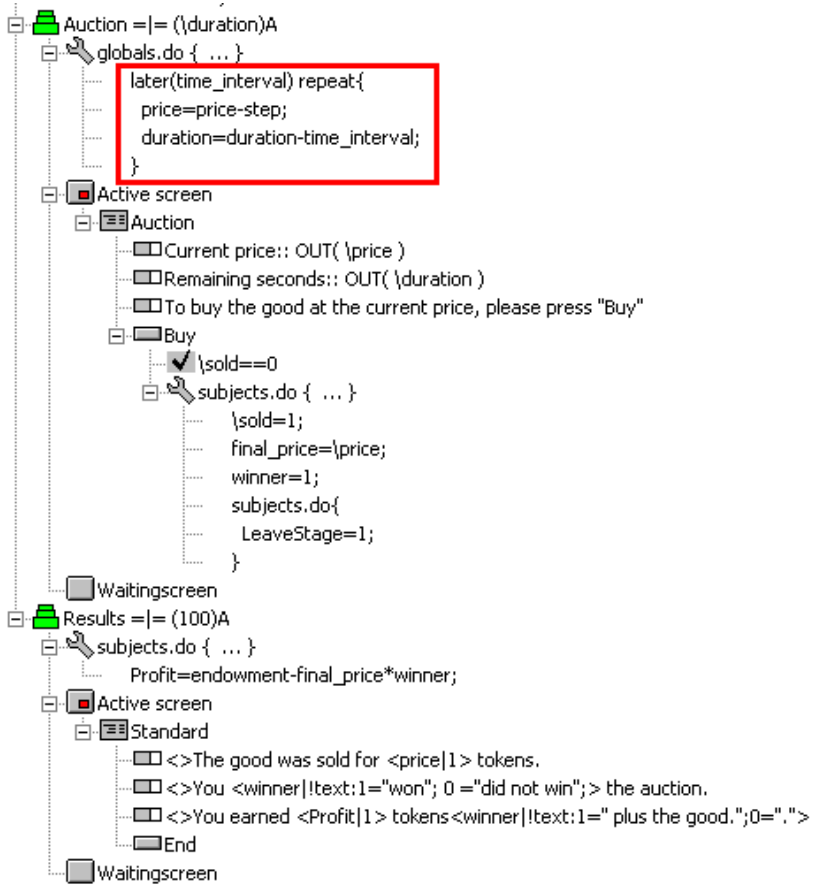
A Dutch auction is an auction in which the auctioneer begins with a very high asking price, which is progressively lowered until some participant accepts the auctioneer's price, or until a predetermined time is over.

- all subject are buyers
- `global` variables:
  - initial asking price
  - duration of the auction (in seconds) $\rightarrow$ duration
  - step of decrease of the price $\rightarrow$ step
  - frequency of decrease of the price (in seconds) $\rightarrow$ time_interval

See dutch_auction.ztt

Dutch Auctions - II





The `later` statement

In the `Auction stage`, we run the following program:

```
later(time_interval) repeat{
  price=price-step;
  duration=duration-time_interval;
}
```

The **general form** is:

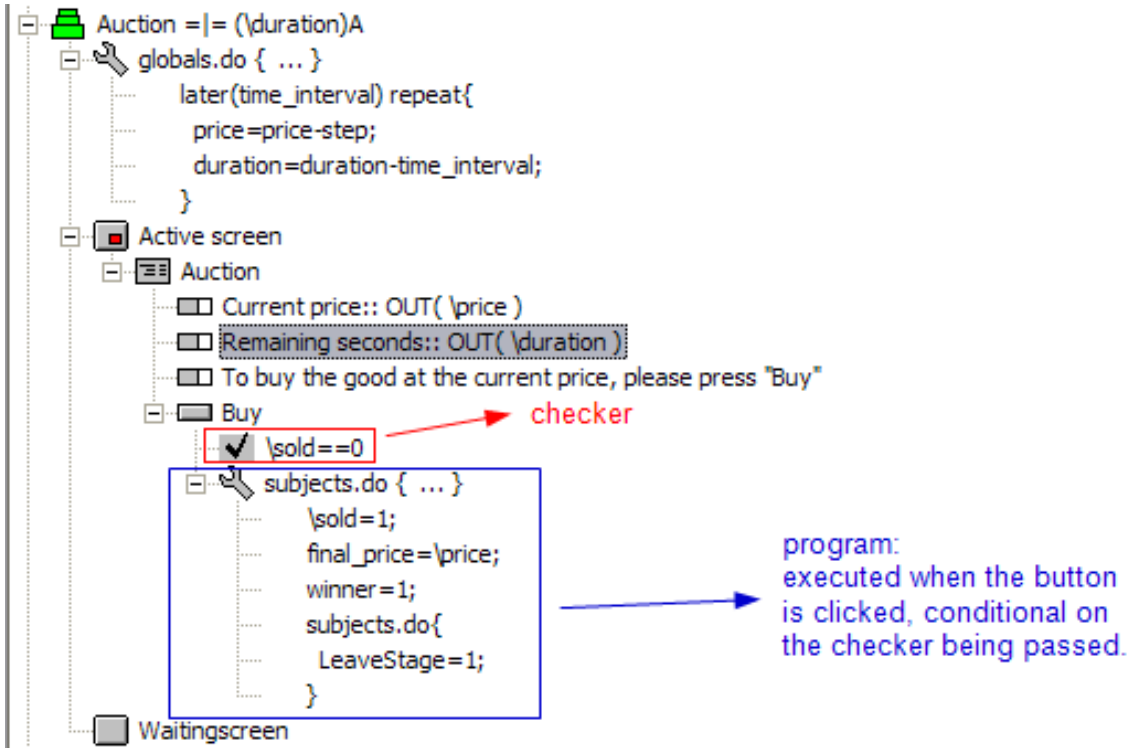
```
later(a) repeat{
  program;
}
```

Expression `a` is calculated. The resulting number of seconds later, the `program` is executed.

Programs into buttons

When a subject tries to buy the good:

- check that the good has not been sold so far
- run a program to assign the good to the subject

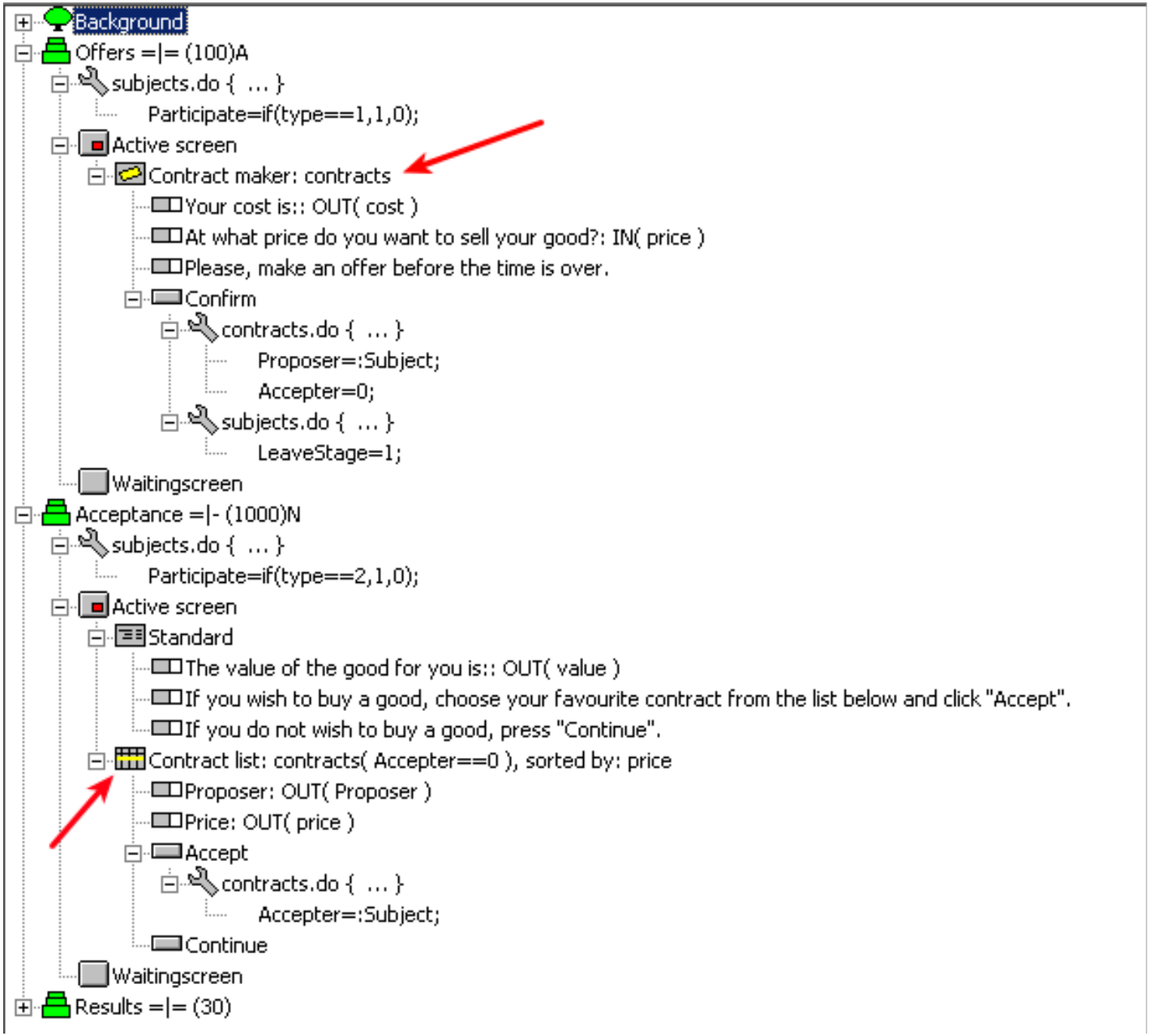


Posted offer markets - I

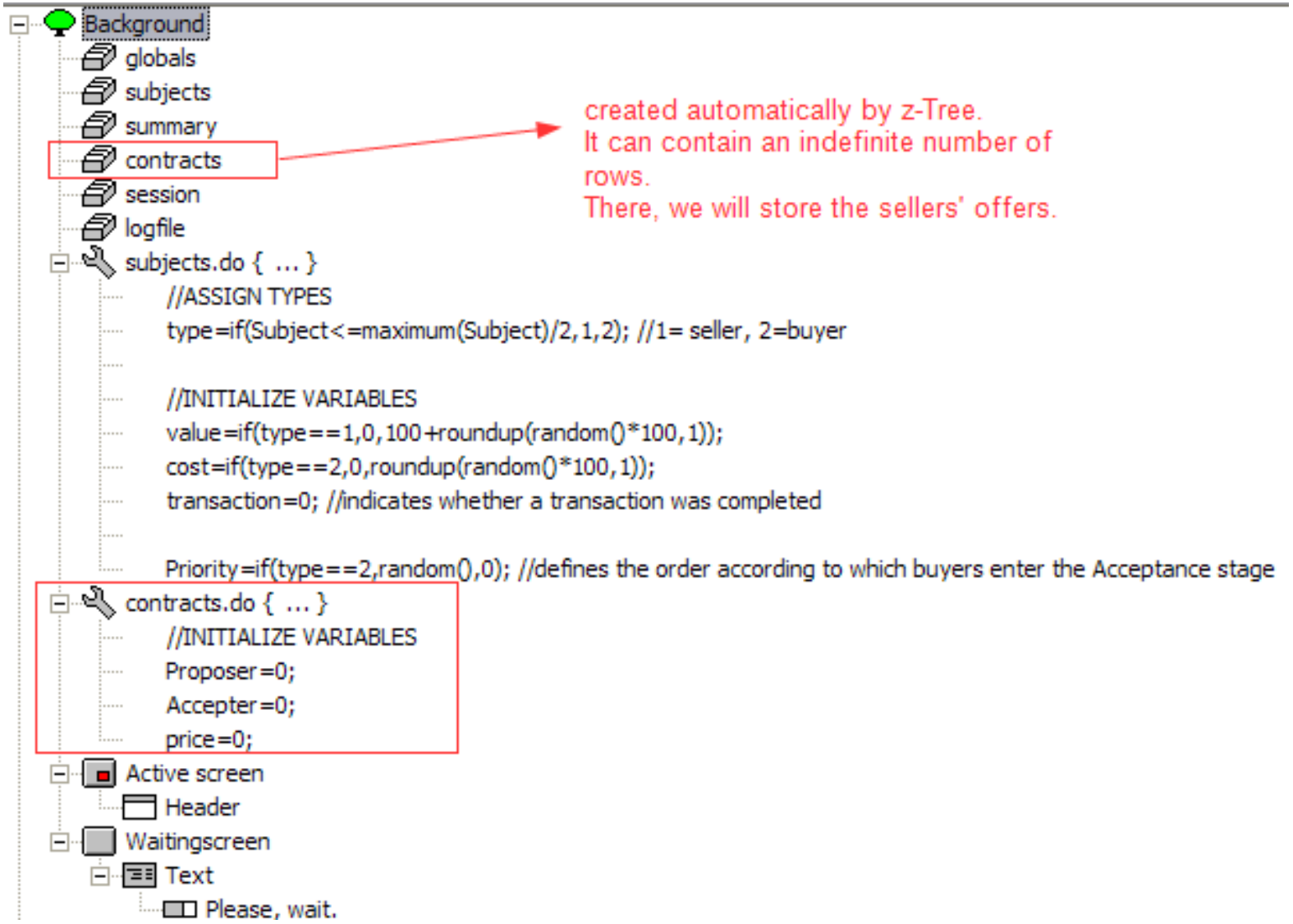
- subjects in the role of *buyers and sellers*
- each seller makes an offer, without knowing the offers made by other buyers.
- buyers act *sequentially*, in random order
- they can see all the sellers' offers that are still open, and choose which one to accept, if any.
- accepted offers are not visible anymore to subsequent buyers.

See posted_offers_markets.ztt

Posted offer markets - II

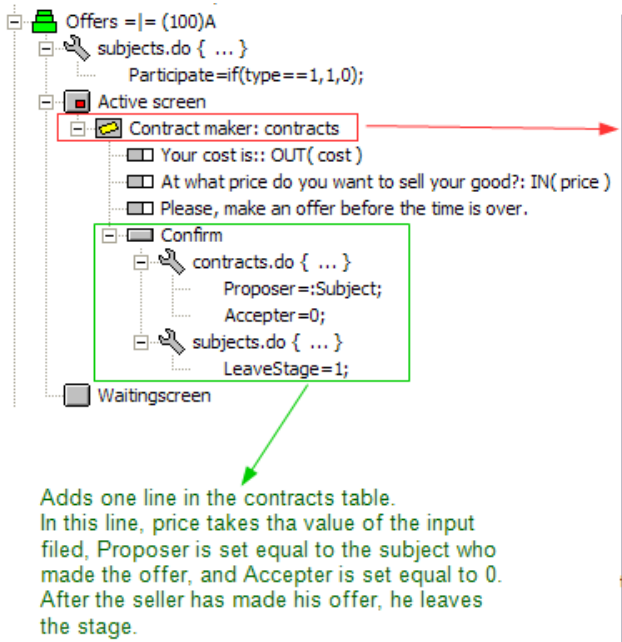


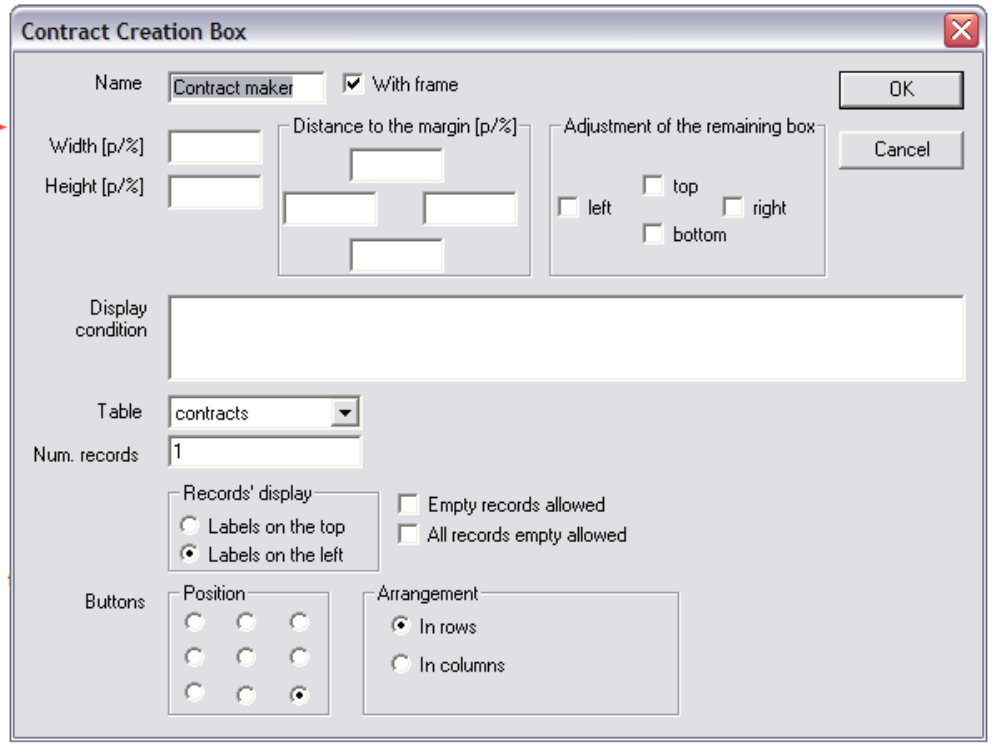
The contracts table



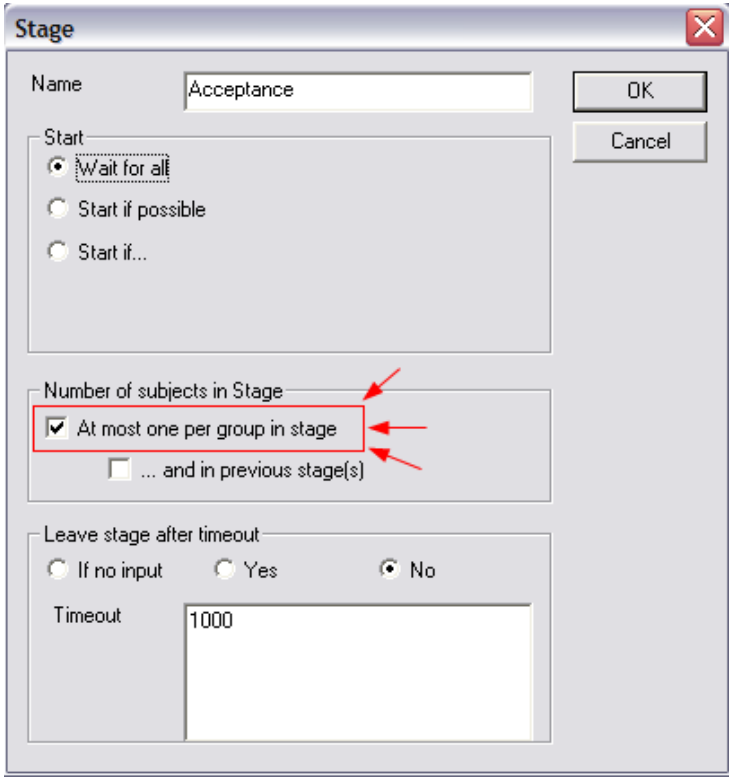
The contracts creation box

Records in the Contracts table are created via inputs in the Contract Creation Box.





Buyers enter the Acceptance stage one by one



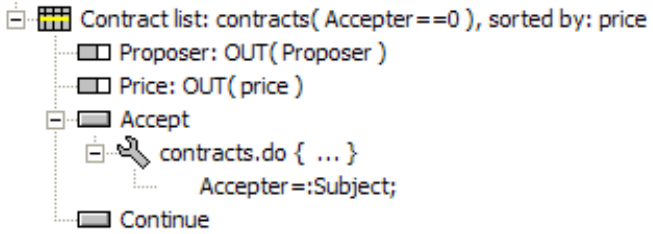
- the variable `Priority`, if set, defines the order according to which subjects enter the stage.
- if `Priority` is not set, subjects enter the stage in random order.

The contracts list box

Available offers are displayed in a Contract list box.

Proposer	Price
3	100
2	106
1	142

Continue | Accept



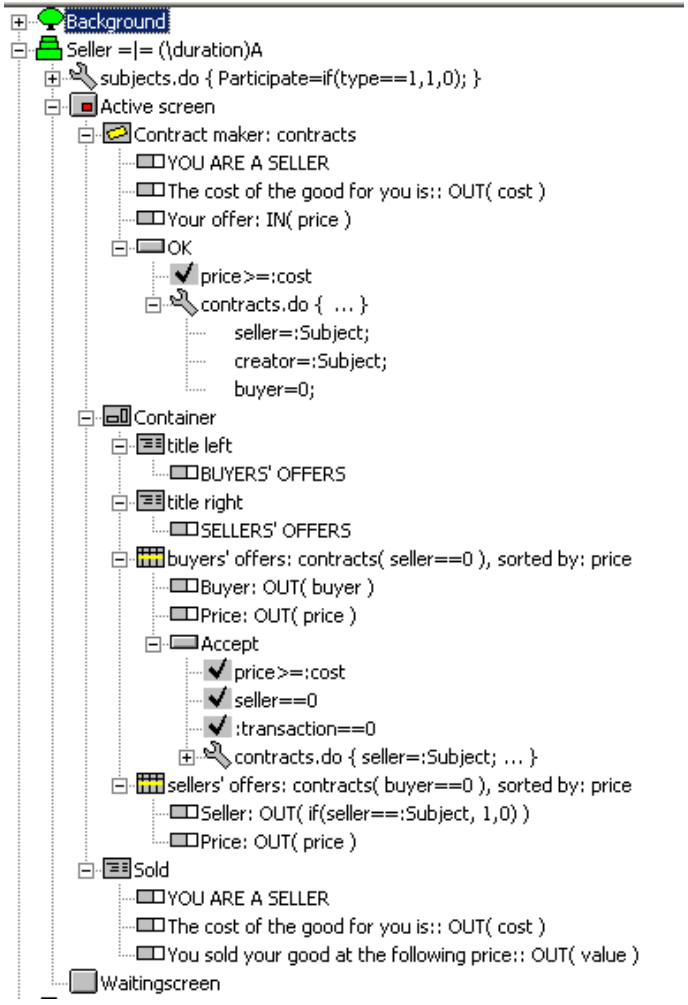
Double auctions - I

see the z-Tree Tutorial, pages: 57-66

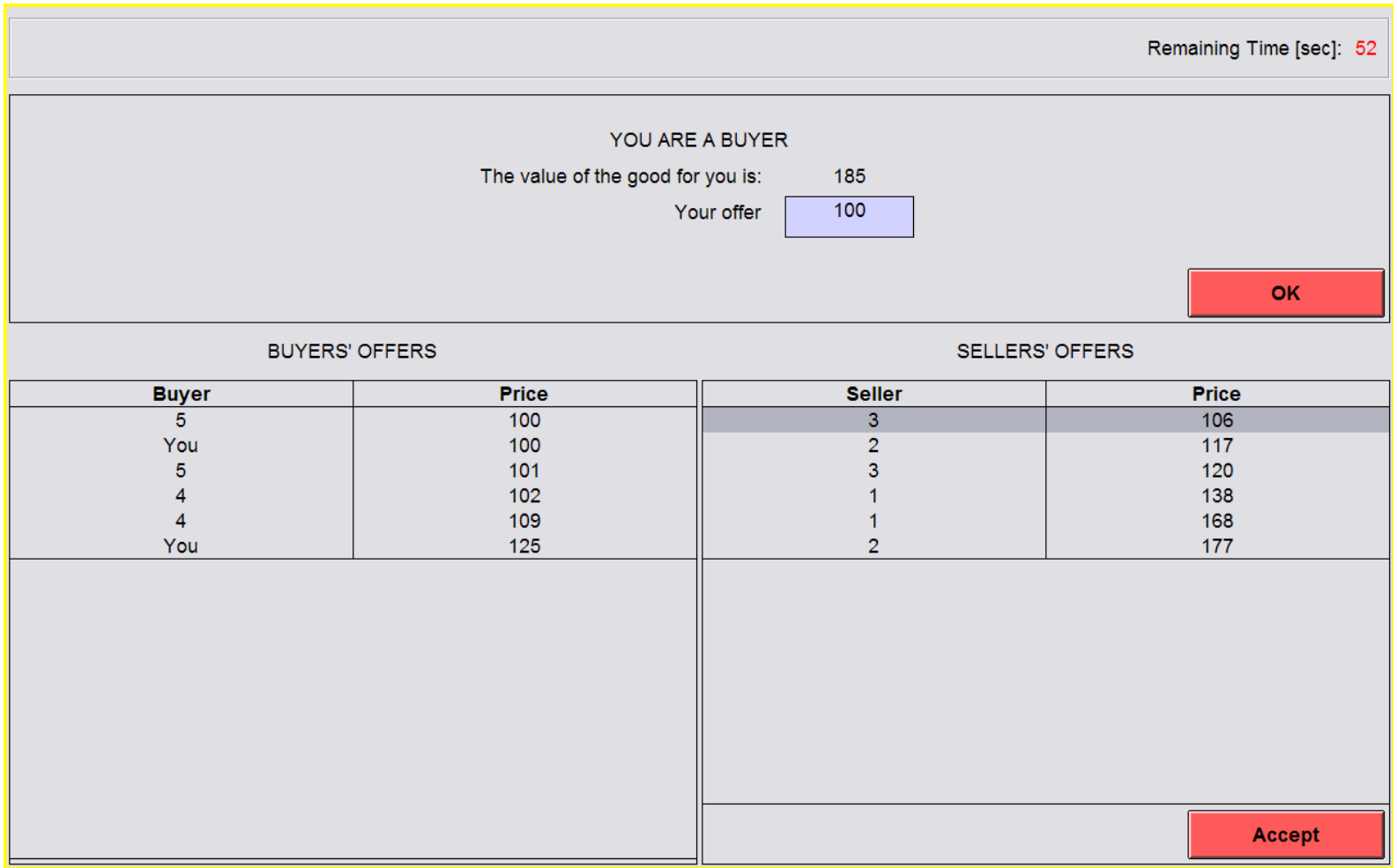
- subjects in the role of buyers and sellers.
- *both sellers and buyers* can make offers, *at the same time*.
- each seller and each buyer can make *more than one offer*.
- sellers and buyers can see *all* the open offers (made by buyers and by sellers)
- each seller can accept *only one* of the buyers' offers.
- each buyer can accept *only one* of the sellers' offers.

See double_auction.ztt

Double auctions - II



Double auctions - screenshot



Remaining Time [sec]: 52

YOU ARE A BUYER

The value of the good for you is: 185

Your offer 100

OK

BUYERS' OFFERS

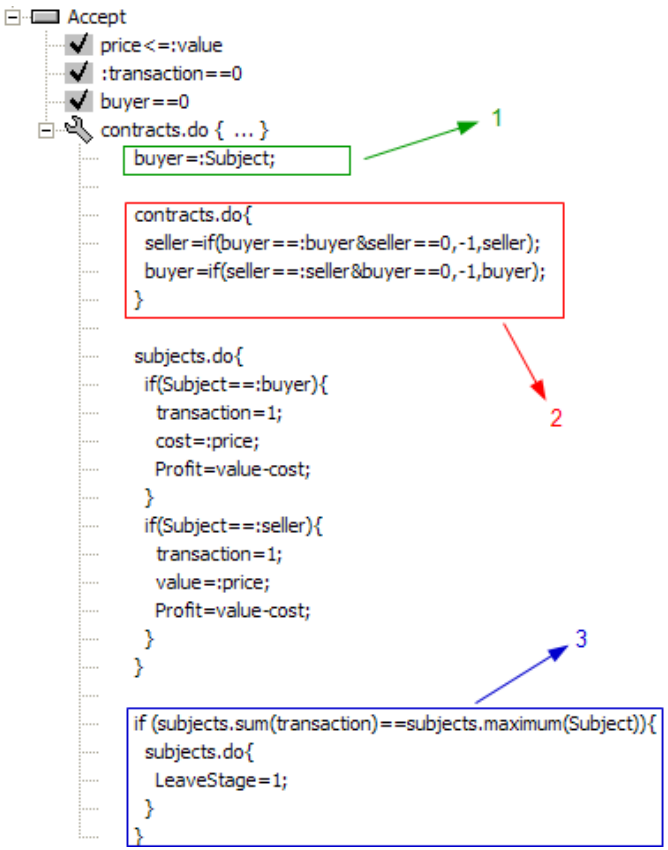
Buyer	Price
5	100
You	100
5	101
4	102
4	109
You	125

SELLERS' OFFERS

Seller	Price
3	106
2	117
3	120
1	138
1	168
2	177

Accept

Double auctions - accepting an offer



1. set the ID of the accepter
2. close all other offers by the same proposer and by the same accepter
3. if all subjects have signed a contract, leave the stage

Note: the variable `LeaveStage` is preset in z-Tree.

Part VI

Advanced concepts, questionnaires and crashes

Advanced concepts, questionnaires and crashes

Advanced concepts

- Arrays
- `iterator`
- Data from previous periods
- Bankruptcy rules
- Chats

Summary of z-Tree tables

Questionnaires

Handling crashes

- Crash of a client PC
- Crash of the server PC

Arrays

Arrays are *indexed variables*. In an index set, every finite, equidistant subset of numbers is allowed (e.g *1,2,3,...*; but also *5, 10, 15, ...*).

The index set needs to be defined *before* the first use of the array.
This is done with one of the following instructions:

- `array arrayname[]` $\rightarrow$ defines an array with indices from 1 to the number of subjects
- `array arrayname[n]` $\rightarrow$ defines an array with indices from 1 to n
- `array arrayname[x,y]` $\rightarrow$ defines an array with indices from x to y
- `array arrayname[x,y,z]` $\rightarrow$ defines an array with indices x, x+z, x+2z,..., y.

Arrays - an example

```
//the index set of array p is 10, 15, 20
array p[10, 20, 5];

p[10]=1;
p[15]=5;
p[20]=2;

p[30]=3; //sets p[20] to 3
x = p[10] + p[20]; // x=4
```

`arrayname[indexvalue]` reports the element of array `arrayname`, corresponding to the index `indexvalue`.

Note: the expression `indexvalue` is rounded to the nearest possible index.

iterator

Loops can be programmed with the statements `repeat` and `while`.

Another way of programming loops is by using `iterators`.

An iterator creates a *small temporary table* that contains only one variable.

When a table function or a do-statement is applied to an iterator, it corresponds to a **loop over the values contained in the table**.

iterator - example

This example provides two possible ways of calculating the sum of the square of the first five natural numbers:

```
SquareSum= 1 + 4 + 9 + 16 + 25 = 55
```

First way:

```
SquareSum = iterator(i, 1, 5).sum(power(i,2));
```

Second way:

```
SquareSum = 0;
iterator(i,5).do{
 :SquareSum = :SquareSum + power(i,2);
}
```

Note the use of the **scope operator** to refer to a variable that is in the table *which contains* the table on which the program is running.

iterator: syntax

An iterator has the following **syntax**:

- `iterator (varname)` → variable `varname` runs from 1 to the number of subjects.
- `iterator (varname, n)` → variable `varname` runs from 1 to n.
- `iterator (varname, x, y)` → variable `varname` runs from x to y.
- `iterator (varname, x, y, z)` → variable `varname` runs from x to y with steps of z.

Retrieving data from previous periods

In z-Tree, the subjects database is set up freshly in every period, so the information about earlier periods is not available directly.

However, the tables of the previous periods can be accessed with the prefix OLD.

So, if you want to copy some `variable` from the previous period, you can write:

```
if (Period >1) {
 variable=OLDsubjects.find(same(subject),variable);
}
```

This way, you can only access data from the previous period. To access data from earlier periods, a different procedure should be followed.

Data from previous periods - example

Suppose you want to run an experiment with the following structure:

1. N periods
2. in every period, each subject earns a certain amount of points
3. the subject's profit is determined at the end of period N:
   - three of the periods are drawn at random by the subject
   - the profit is set equal to the sum of the points earned by the subject in these three periods

Example: values_from_previous_periods.ztt

Data from previous periods - I

YOUR PONITS IN THE PAST 10 PERIODS

Period	Points
1	1
2	7
3	6
4	9
5	9
6	5
7	2
8	9
9	3
10	5

click the button to select one period

draw a number

click the button to select one period

draw a number

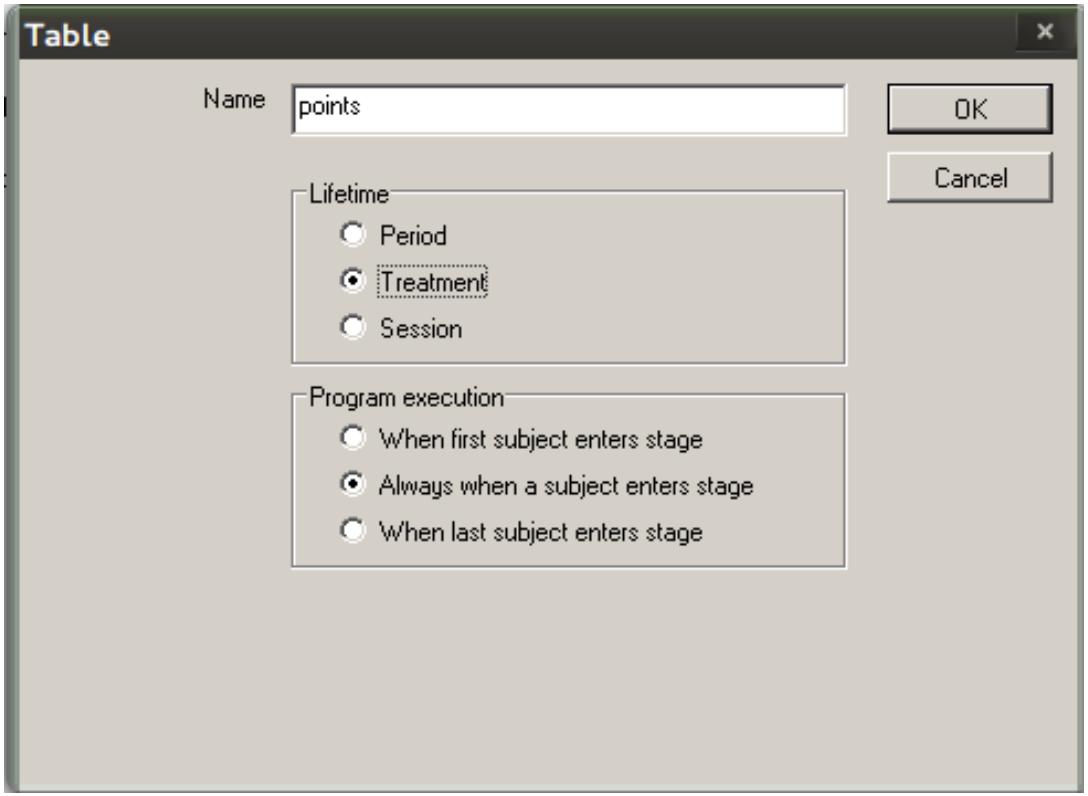
click the button to select one period

draw a number

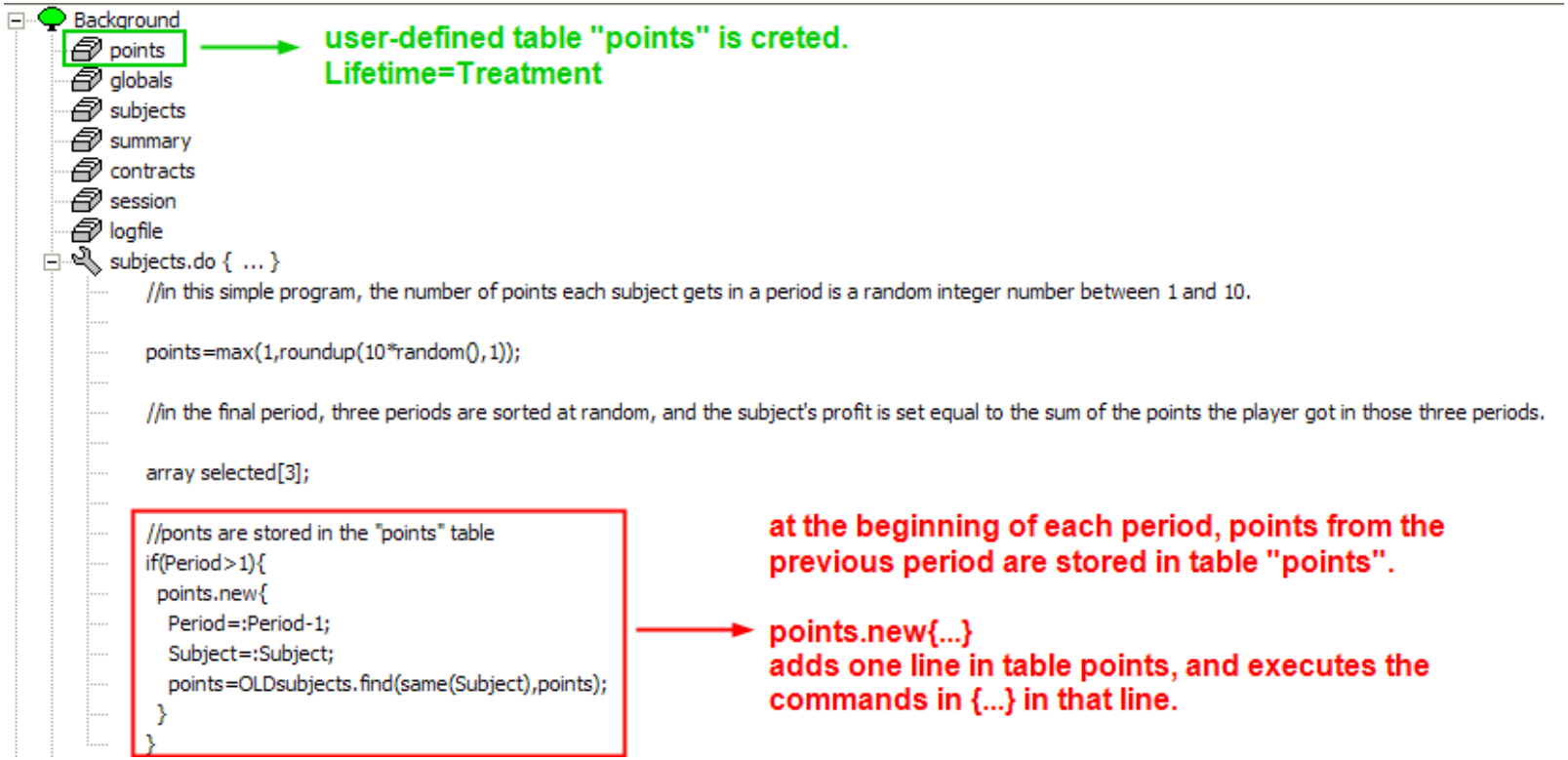
Data from previous periods - II

To retrieve data from all earlier periods, these data have to be stored in a **user-defined table** with `lifetime` set to `Treatment` or to `Session`.

From the menu, select `Treatment`→`New table`

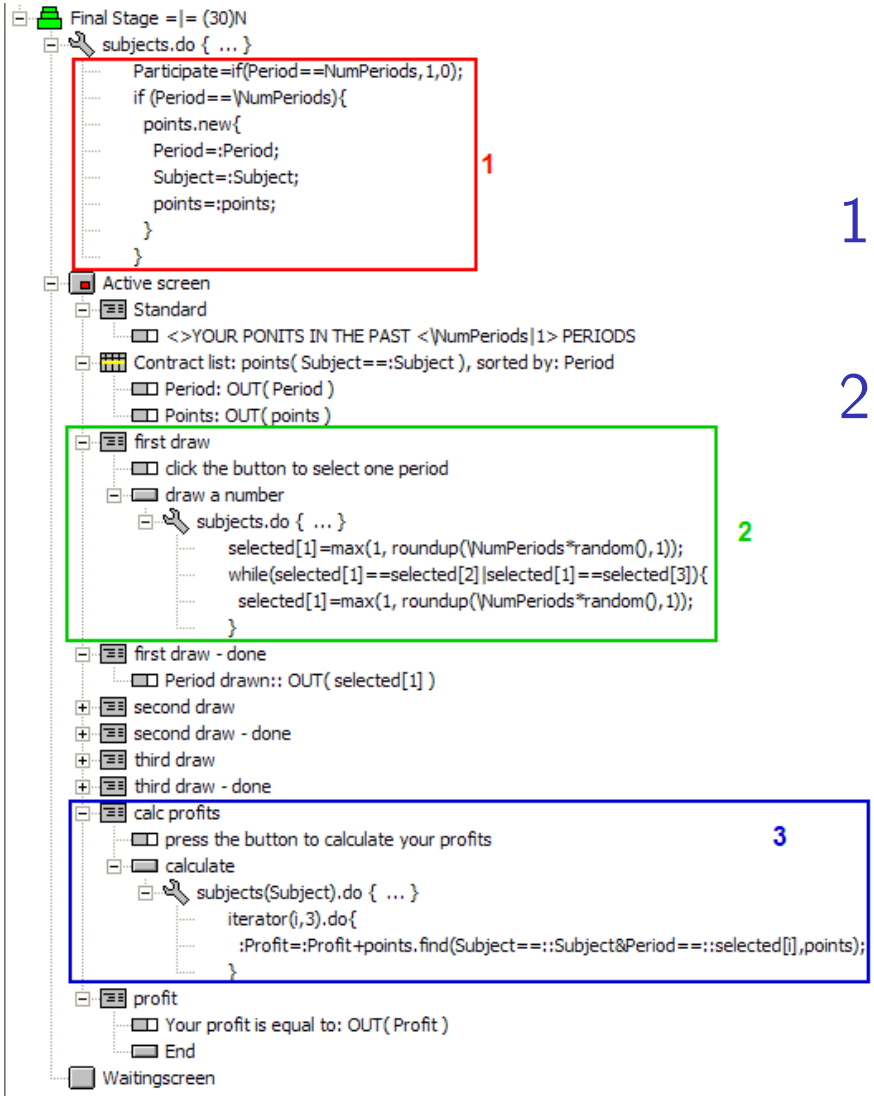


Data from previous periods - III



Note: again, here we use the `scope operator` to refer to variables in the `subjects table` from the `points table`

Data from previous periods - IV



1. store points form the last period
2. draw a random period NB. this box is shown only if the draw is yet to be done. `Display condition: selected[1]==0`

3. calculate the profit, using the iterator.

Bankruptcy - I

Tip: try to design the experiment in order to avoid possible losses.

If losses occur, can be covered by the following sources:

- profits from previous periods and treatments
- show-up fee
- money injected by the experimenter during the session

Example: bankruptcy.ztt

Bankruptcy - II

If previous profits are not enough to cover all losses, but the **show-up fee** is, a message appears on the subject's screen.
This message informs the subject that he can choose *either to use the show-up fee, or to drop out of the experiment*.

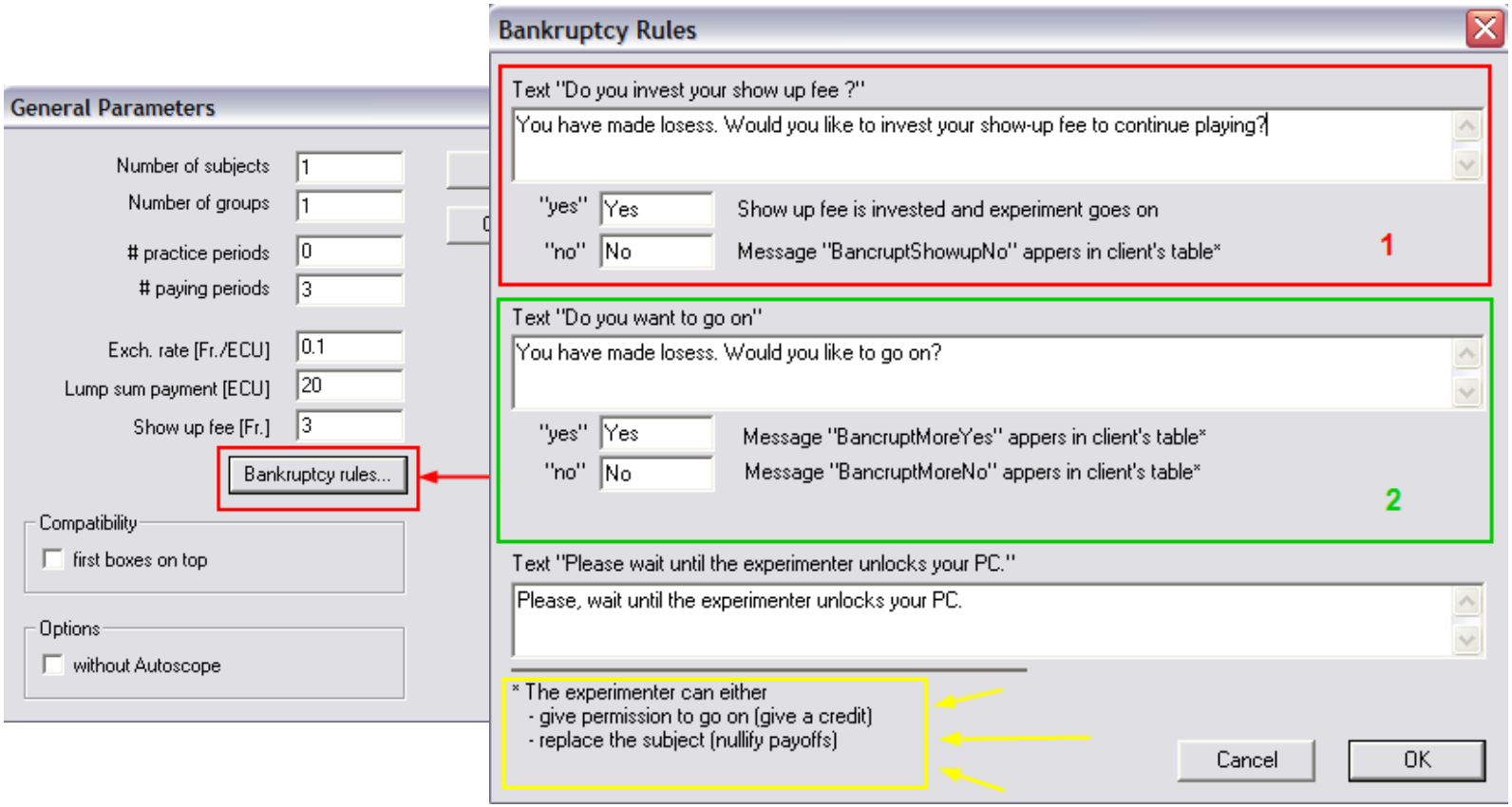
- if the subject chooses to use the show-up fee, he may simply play on.
- otherwise, the subject reaches the state `BankruptShowupNo` (which you can read in the clients table), and the experimenter has to release him manually from the server.

Bankruptcy - III

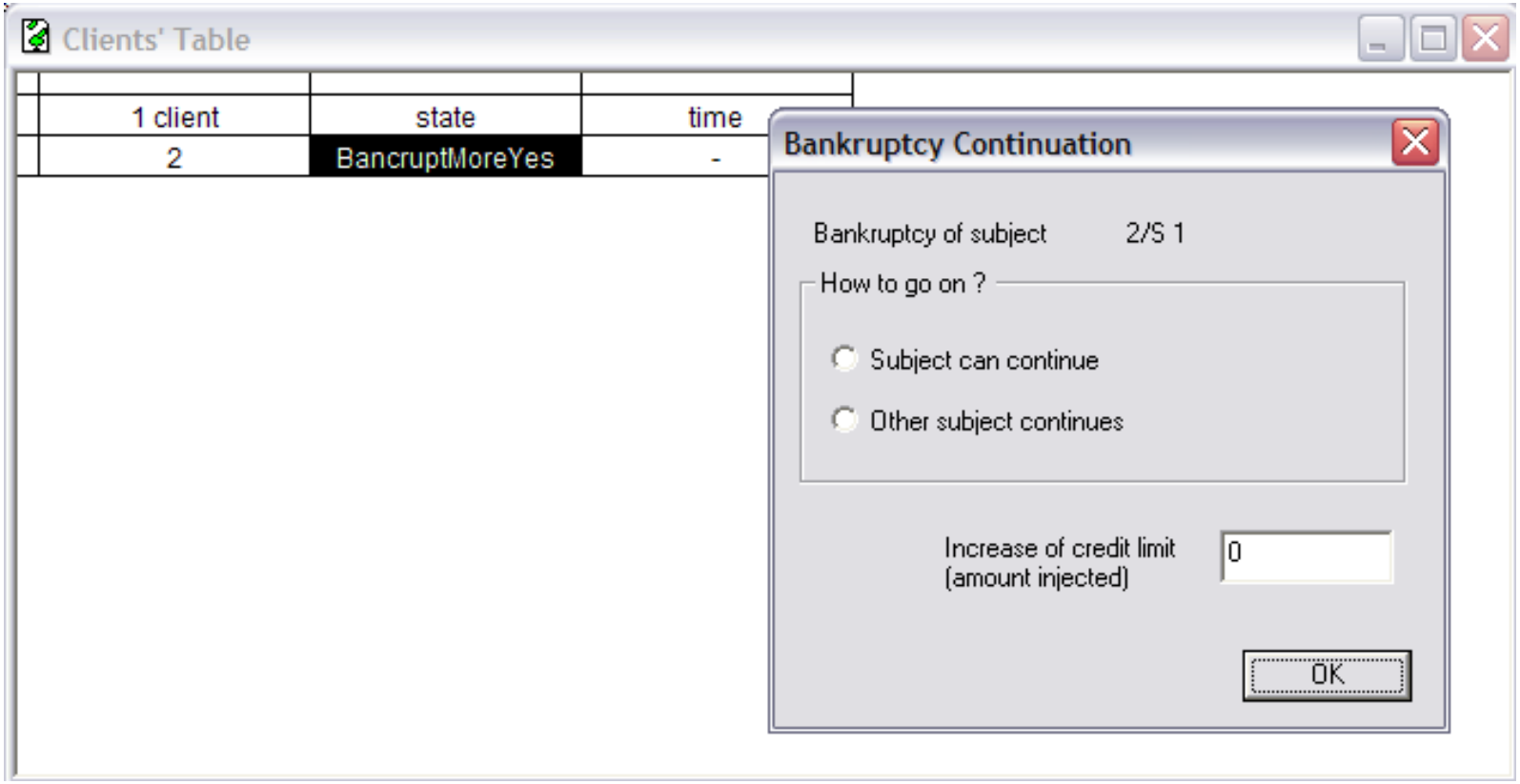
If the **show-up fee cannot cover the losses** (or has already been exhausted) a message informs the subject that he has incurred a loss.
This message can be concluded with a question.

- By answering "no", the subject enters the state `BankruptMoreNo`. In this case, the experimenter has to release the subject from the server.
- If the subject answers this question with "yes", he enters the state `BankruptMoreYes`. In this case, there are 3 possibilities:
  1. Allow the subject to continue, by injecting into his account an amount of money higher than his current losses
  2. Another subject takes the role of the subject released
  3. The subject drops out

Bankruptcy - IV



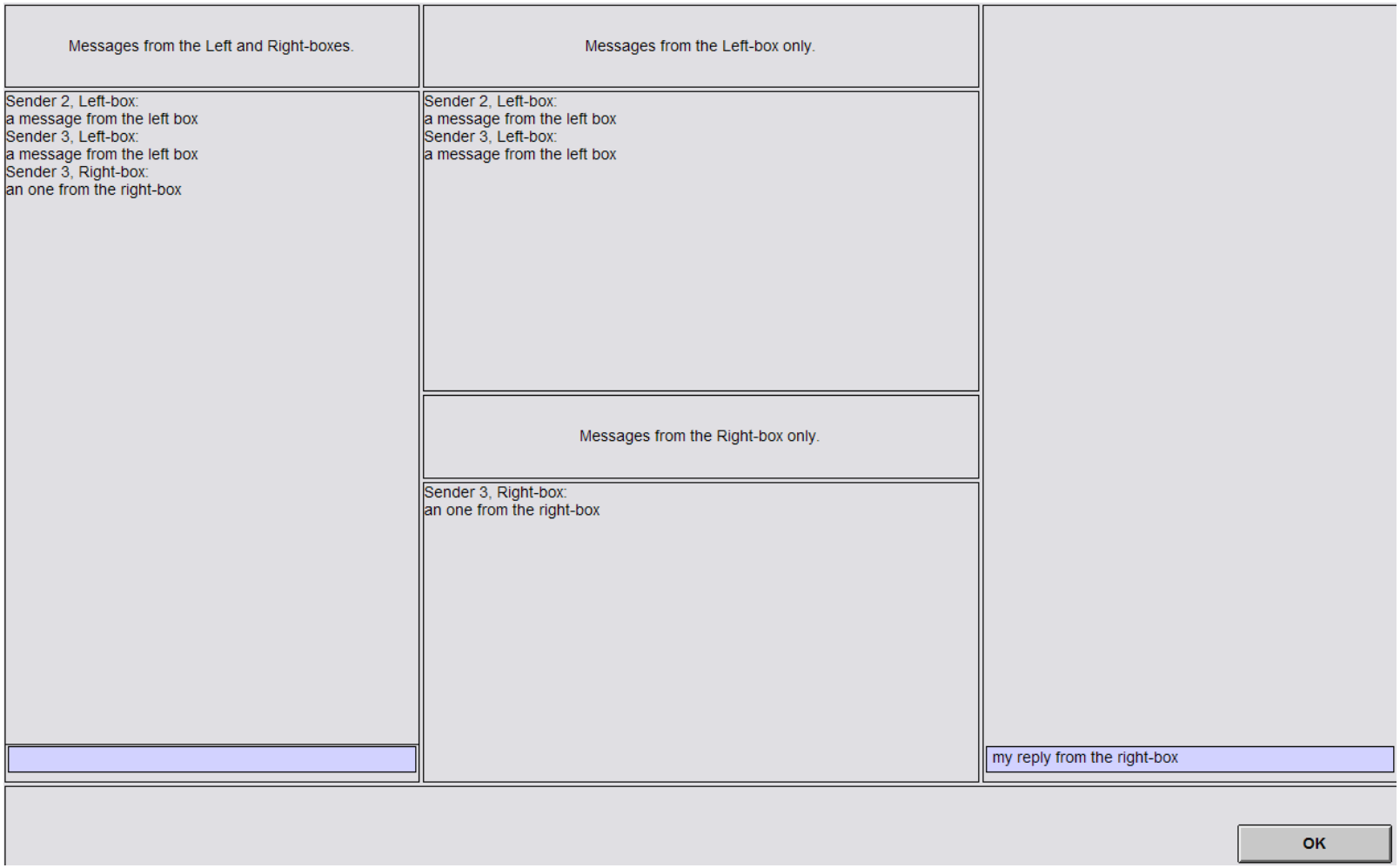
Bankruptcy - V



session table

Subject	FinalProfit	ShowUpFee	ShowUpFeeInvested	MoneyAdded	MoneyToPay	MoneyEarned
1	-4.7000000000	3	1	3	1.2999999999999	-1.7000000000000

Chats

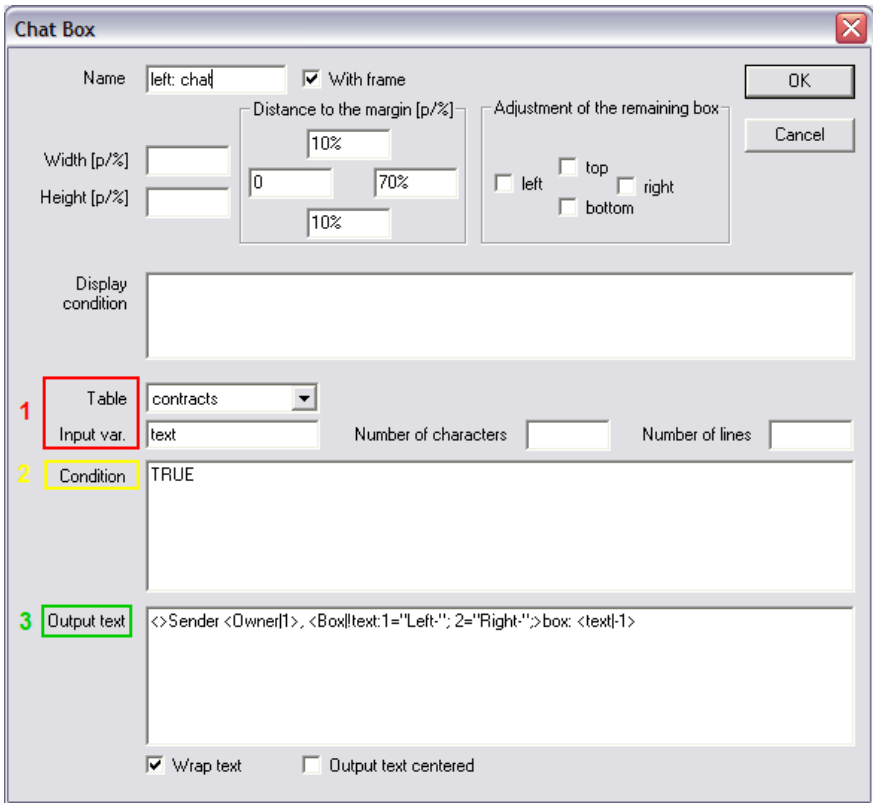


Example: chat.ztt

Chat Box

In a stage, you can add a chat box by clicking on:
`Treatment` → `New Box` → `New Chat Box`

1. specify the table where you want to store the chat data contracts table user-defined table



2. specify what records you want to display on the subject's screen
3. specify the output text

Chat data

zTree - [contracts table]

File Edit Treatment Run Tools View ?

	Period	Owner	Box	text		TimeStage
	1	2	1	"message from the left-box"	-	99999
	1	1	2	"message from the right-box"	-	99999
	1	1	1	"another message from the left"	-	99999
	1	2	2	"and another from the right"	-	99999

- messages are saved in the `contracts table`, or in the user-defined table specified in the Chat-box dialogue.
- The `contracts table` and the user-defined tables are the *only tables* in which text-variables can be saved.
- To **display** text variables on the screen, write "-1" in the layout filed.

The `subjects table`

Contains *one record per subject*. It is freshly set up for each period. The subjects table of the previous period is available under the name `OLDsubjects`.

In the `subjects table`, the following variables are always defined by z-Tree:

- Period
- Subject
- Group
- Profit
- TotalProfit - this is calculated automatically. It should not be changed manually.
- Participate
- LeaveStage

The `globals table`

This table contains *a single record*, i.e. a single line. It stores values that are the same for all subjects. It is freshly set up for each period. The `globals table` of the previous period is available under the name `OLDglobals`.

In the `globals table`, the following variables are always defined by z-Tree:

- Period
- NumPeriods
- RepeatTreatment

The summary and contracts tables

The summary table

This table contains *one record per period*. It is not destroyed at the end of each period, as the subjects and globals table, but at the end of each treatment. This table is most useful for observing aggregate data of the treatment.

In the summary table, the only variable defined by z-Tree is the variable Period.

The contracts table

This table is used mainly for market experiments and chats. New records can be added to this table, and existing records can be changed.

In the contracts table, the only variable defined by z-Tree is the variable Period.

The `session` table

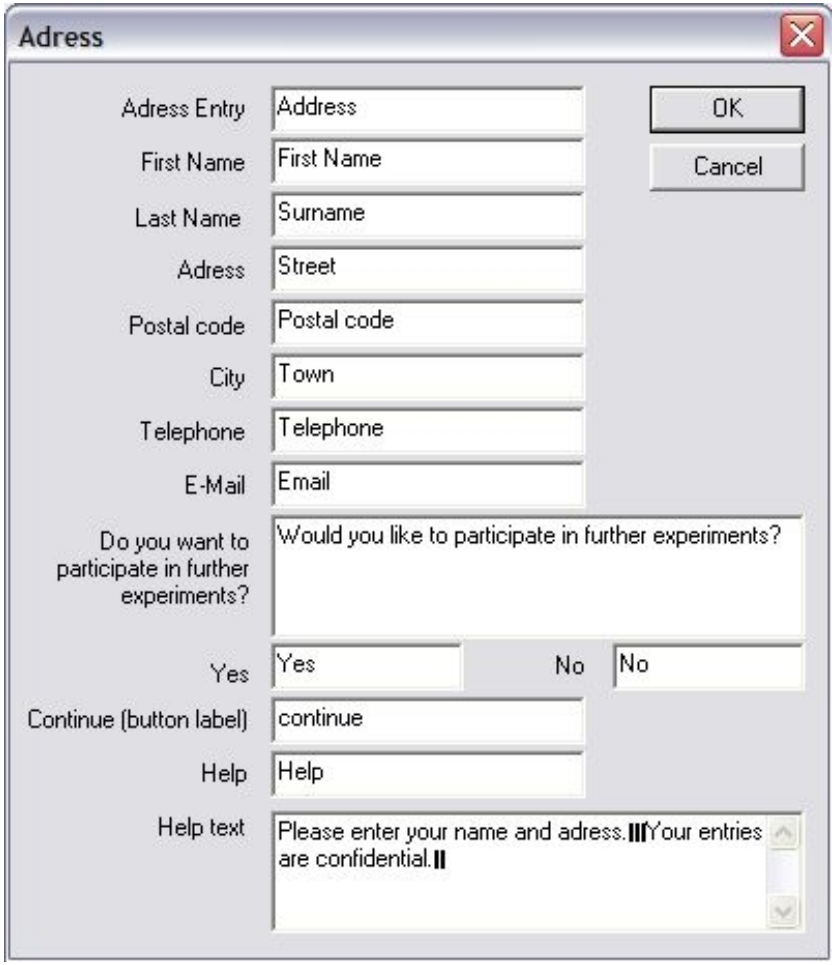
This table contains *one record per subject*. This table is never destroyed during a session, as it stores the aggregate profits earned by subjects in earlier treatments. It can also be used to *exchange information across treatments*, or *from treatments to questionnaires*.

In the `subjects table`, the following variables are always defined and automatically calculated by z-Tree:

- Subject
- Final Profit
- ShowUpFee
- ShowUpFeeInvested
- MoneyAdded
- MoneyToPay
- MoneyEarned

Questionnaires: Address form

Example: questionnaire_complete.ztq

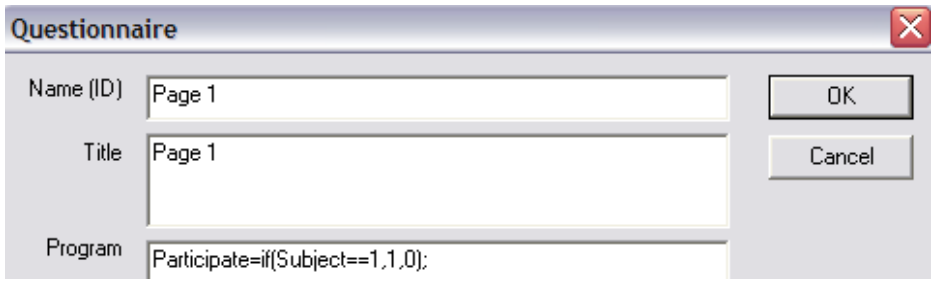


The `address form`:

- usually goes first
- may/may not contain the name and the address of the subject
- as soon as it is filled in by all subjects, the **payment file** is generated by z-Tree.

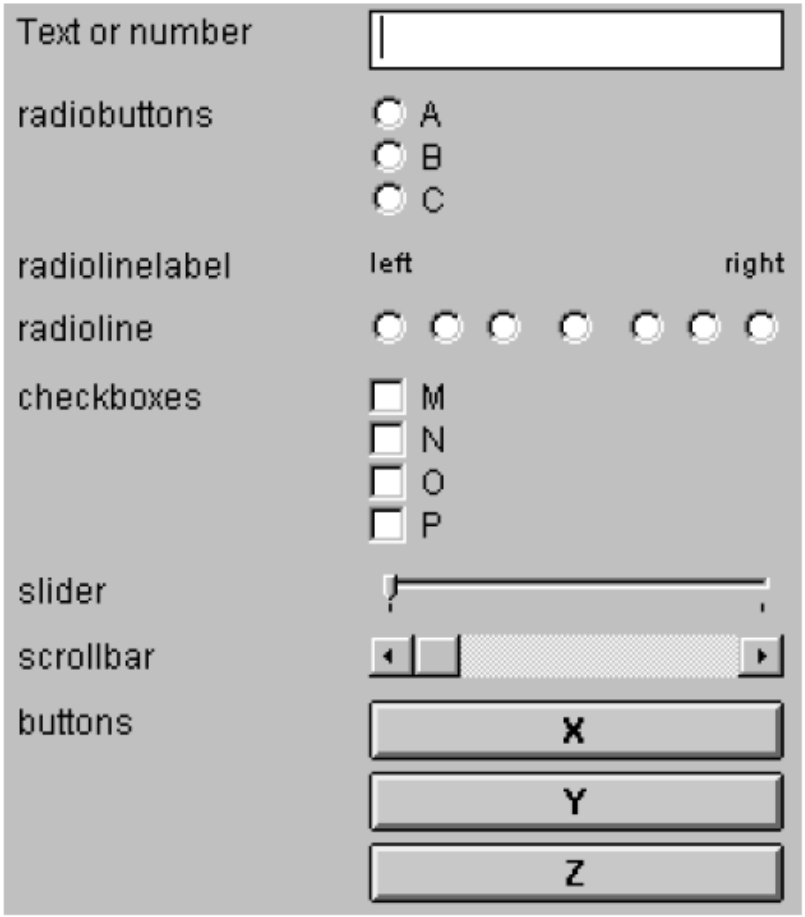
Question forms(s)

- contain several questions with different possible layouts.
- answers in questionnaires are of no consequences $\Rightarrow$ no earnings can be made in questionnaires.
- answers are saved as text and cannot be used in programs.
- the last question form of a questionnaire remains on the users' screen until you continue (with a new treatment, a new questionnaire, or simply shutting down z-leaf).
- Therefore, the final question form *cannot contain a button*.

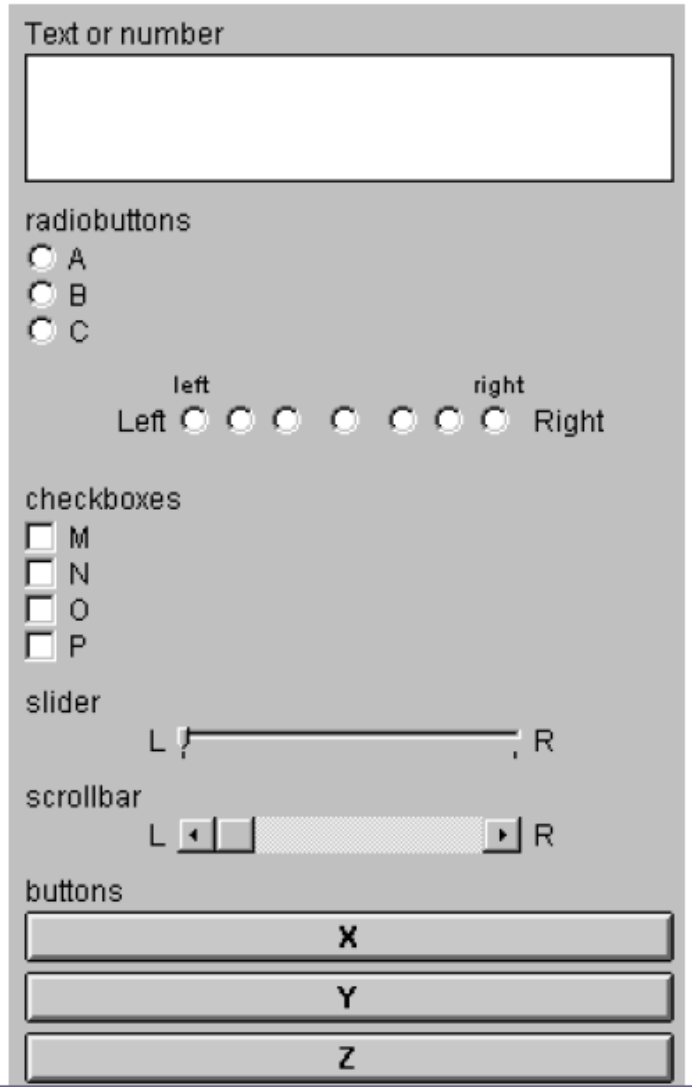


Layout of the questions

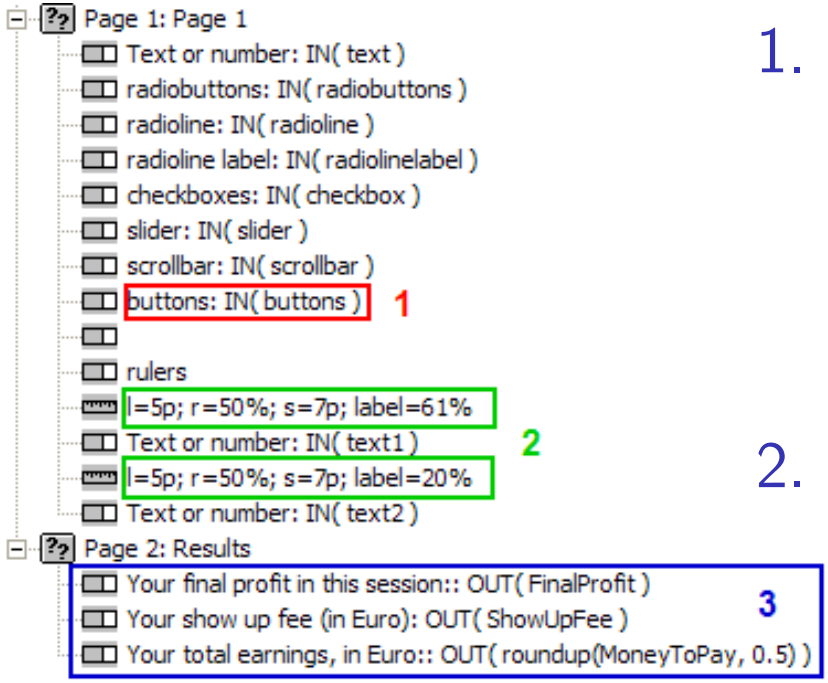
Example of normal layout (not wide):



Example of wide layout:



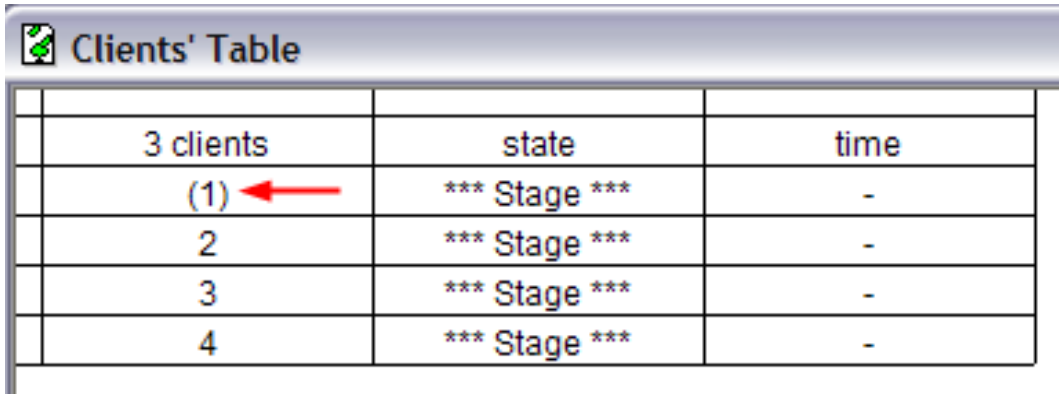
Buttons, rules, and variables



1. buttons: maximum one per question form.
   $\Rightarrow$ clicking a button closes the question form, if possible
2. rules: are used to set the regions where labels and questions are positioned.
3. In questionnaires, only variables from the session table can be retrieved. If you need to retrieve a specific variable from a treatment, you have to store it in the session table with a command in the treatment program (.ztt).

Crash of a client PC - I

When a subject's PC disconnects from the server, the clients appears in parentheses in the clients table.

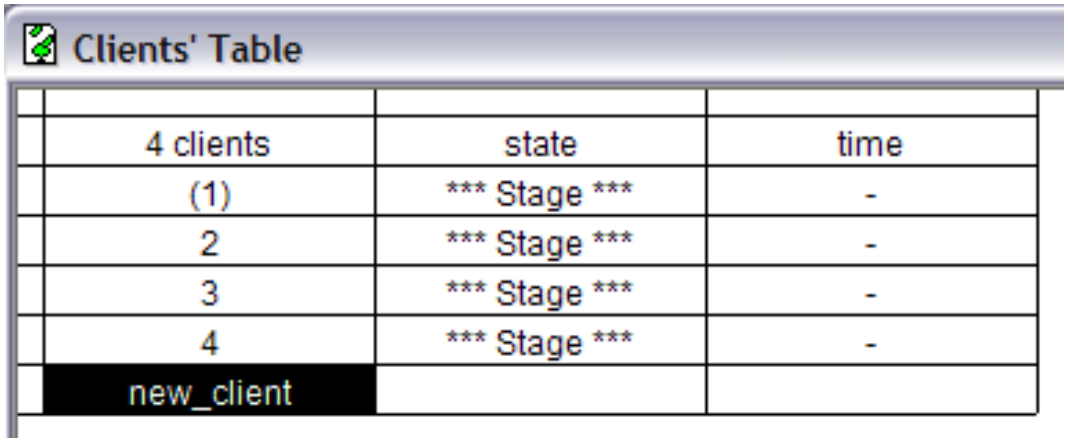


First solution:
Try to **close** the Leaf program on the subject's PC, and to **restart** it.

Crash of a client PC - II

If the first solution does not work, you can try to **move the subject to a different computer**.

1. Start a new PC and launch z-Leaf from there.
2. From the client table, **drag the name of the new client on the name of the client that does not work**. This will release the old client, and replace the old with the new client.

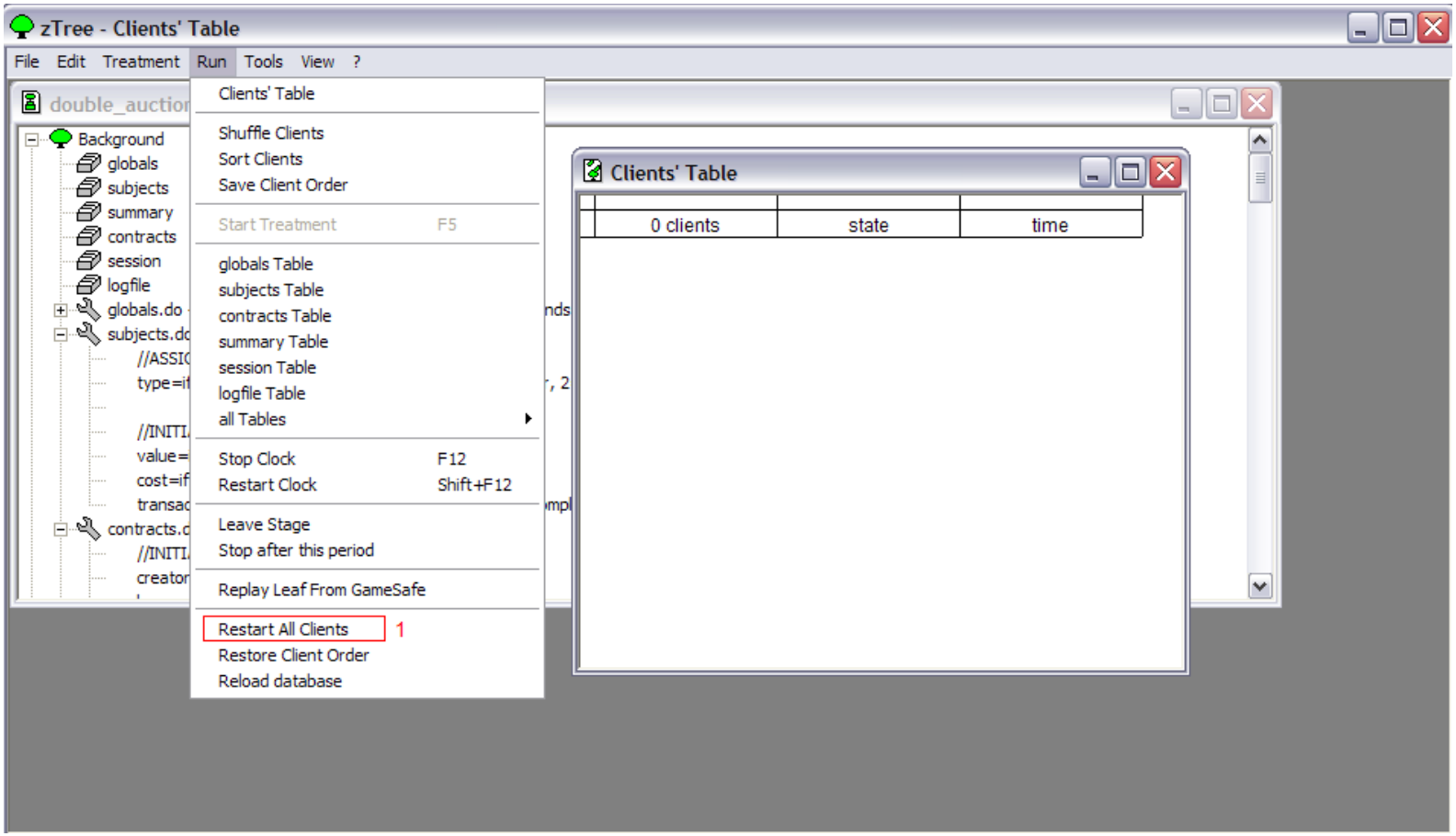


Crash of the server PC - I

After a **crash of z-Tree**, you have to follow this procedure carefully:

- restart z-tree
- open the client's table
- restart all clients with the menu `Run` $\rightarrow$ `Restart all clients`.
- If no clients connect you have different options
  - try to restart the clients manually
  - wait (up to 4 minutes) and try again
  - shut down and restart the experimenter's PC, then follow the previous steps
  - start z-Tree on a different computer

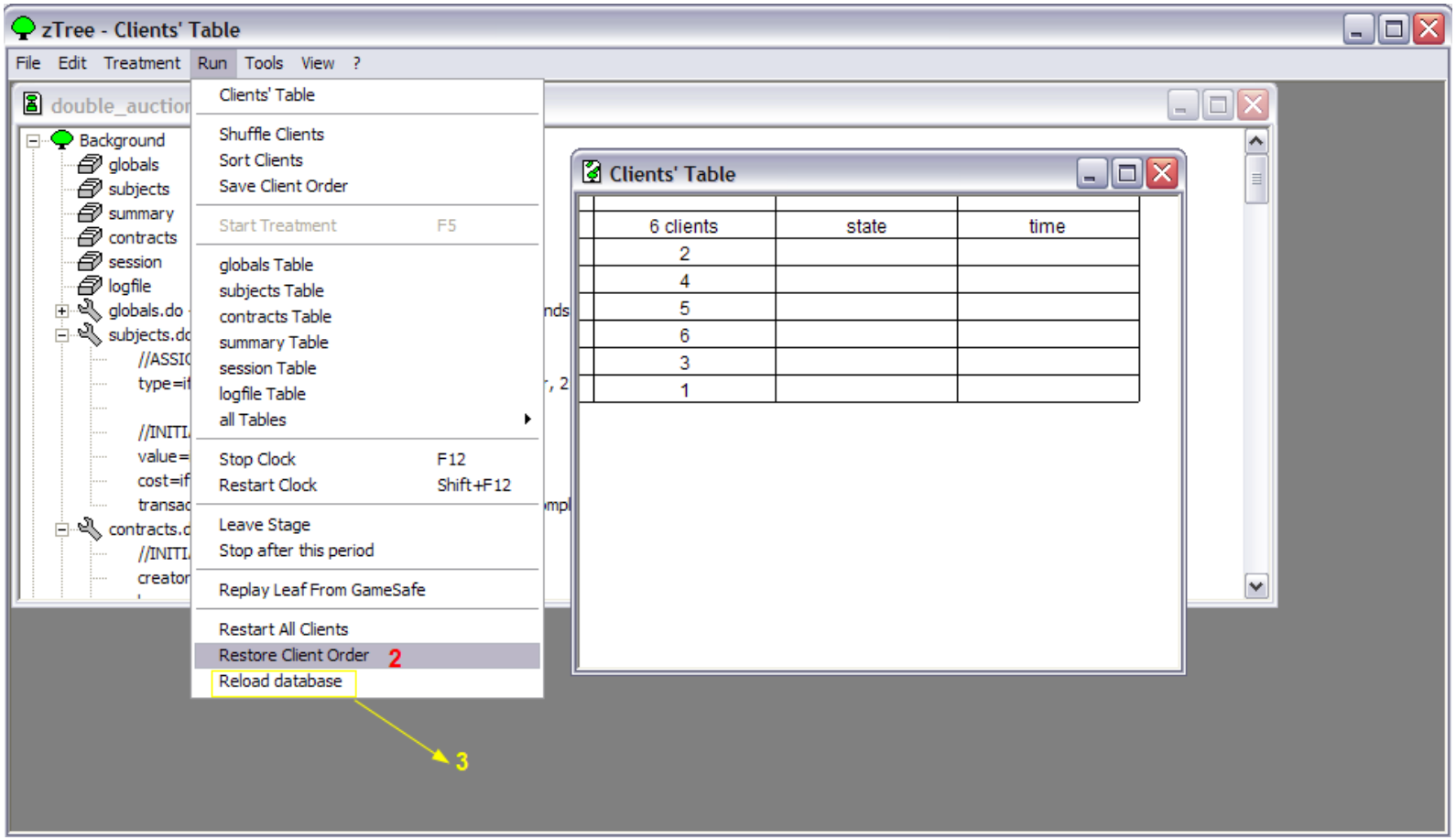
Crash of the server PC - II



Crash of the server PC - III

When you manage to restart all the clients,

- select `Run` → `Restore Client Order` (2).
- then `Run` → `Reload Database` (3).



Crash of the server PC - IV

- check **how many periods have been played** (e.g., from the subjects table)
- open the treatment that was running when the crash took place, and **set the number of practice periods to -n** (where n is the number of periods already played)
- select `Run` $\rightarrow$ `start treatment`.

Part VII

Recruitment, Text formatting and Multimedia

Recruitment, Text formatting and Multimedia

Recruitment: ORSEE

Text formatting

Plots and vectorial graphics

```
plot box
plot items
plot inputs
```

Multimedia, slide-shows and external programs

External programs

Recruitment: ORSEE

ORSEE is a web-based Online Recruitment System, specifically designed for organizing economic experiments. It's key features are:

- multiple experimenter/laboratory/subject-pool/experiment-classes/language support,
- random recruitment,
- public and internal experiment calendar,
- reputation system,
- automated mailing,
- pdf output,
- experimenter rights management.

ORSEE comes with a complete documentation.

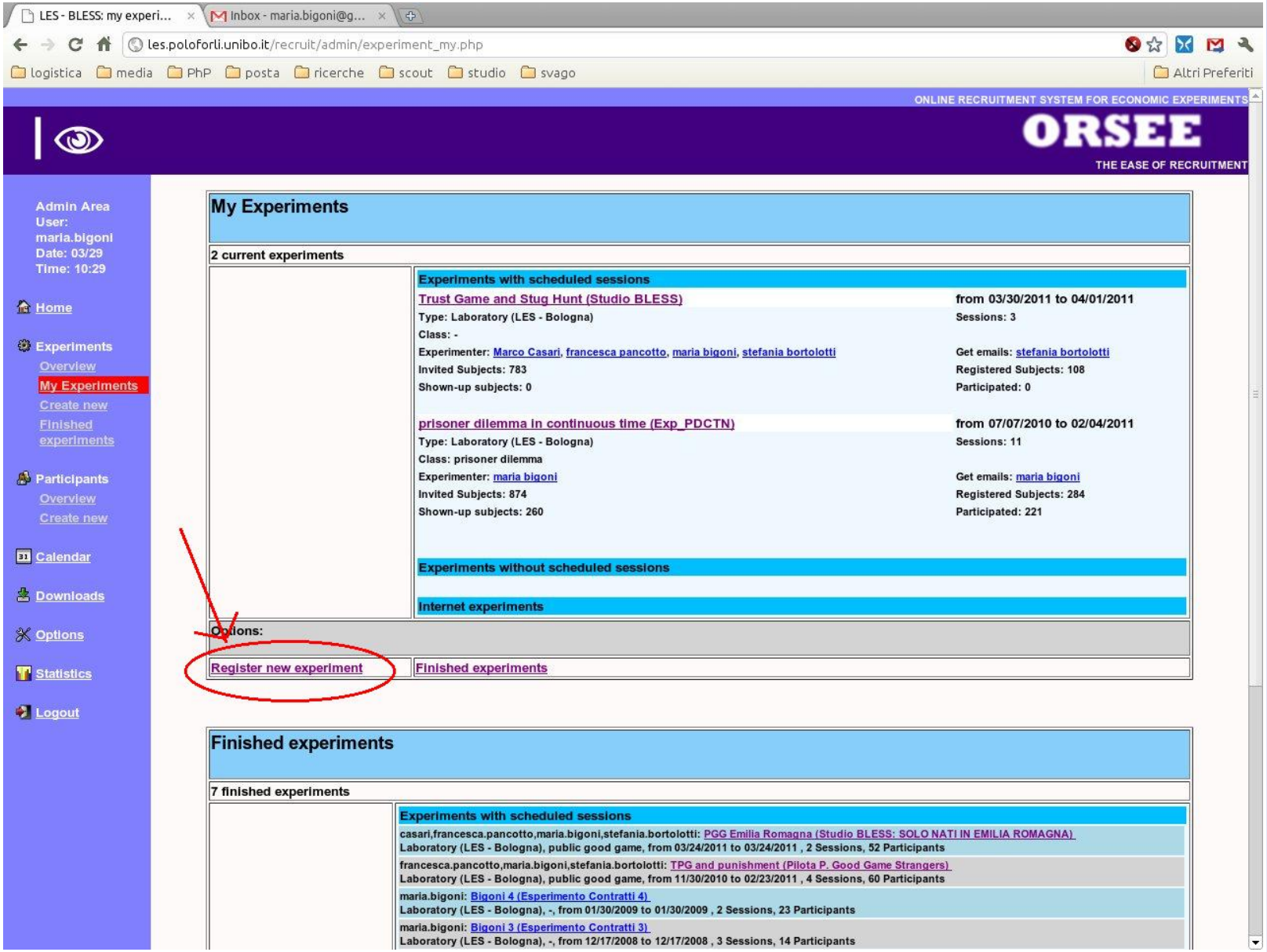
Recruitment with ORSEE - in a nutshell

You need IT support to install ORSEE on your university server.

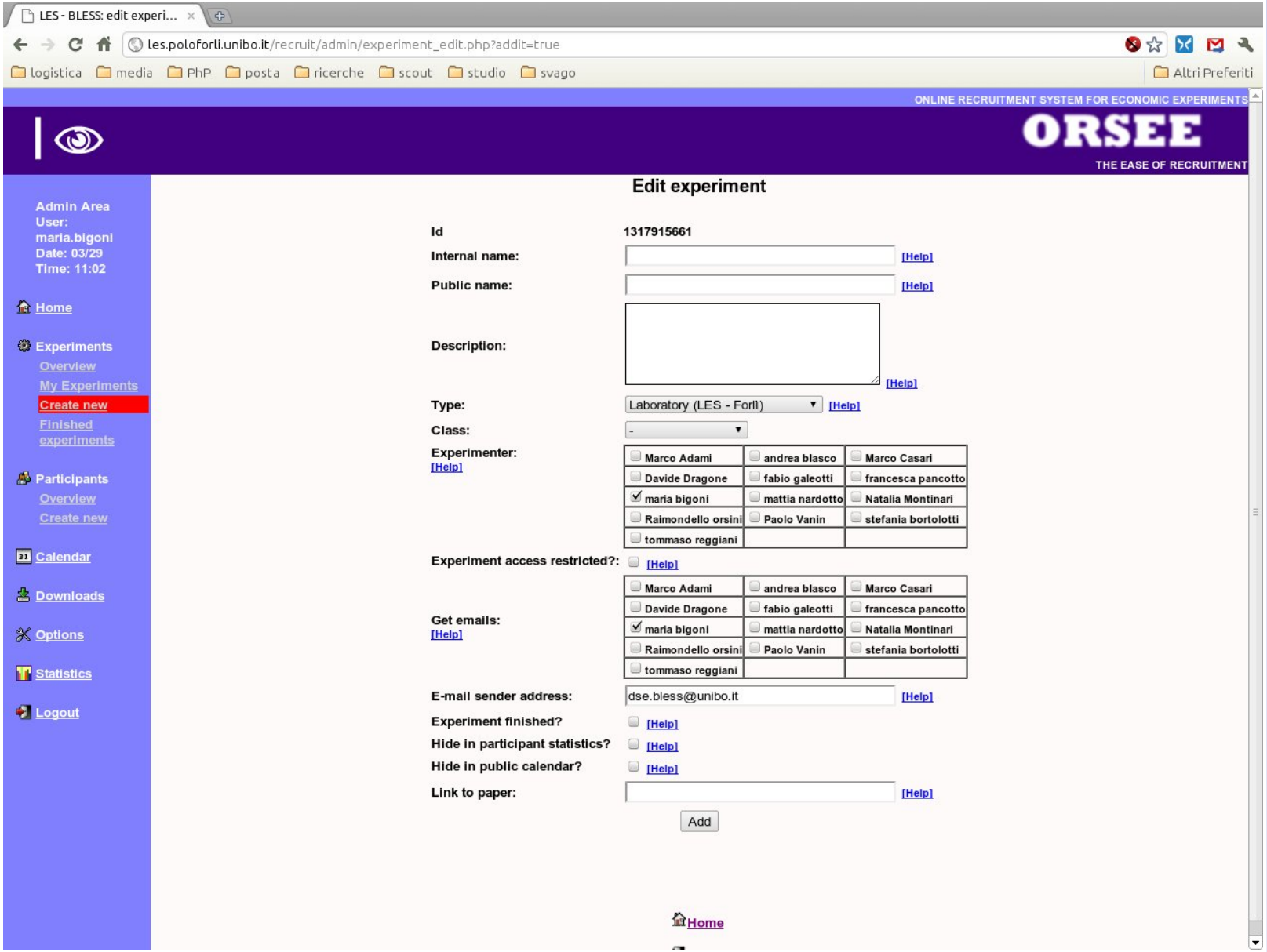
Once ORSEE is installed and you have an account, you can/should:

- create a **new experiment**
- **select subjects** for the experiment
- create **sessions** within an experiment
- send **e-mail invitations** and check who subscribed
- after each session, **record who participated** and close the session
- at the end of the last session, **close the experiment**

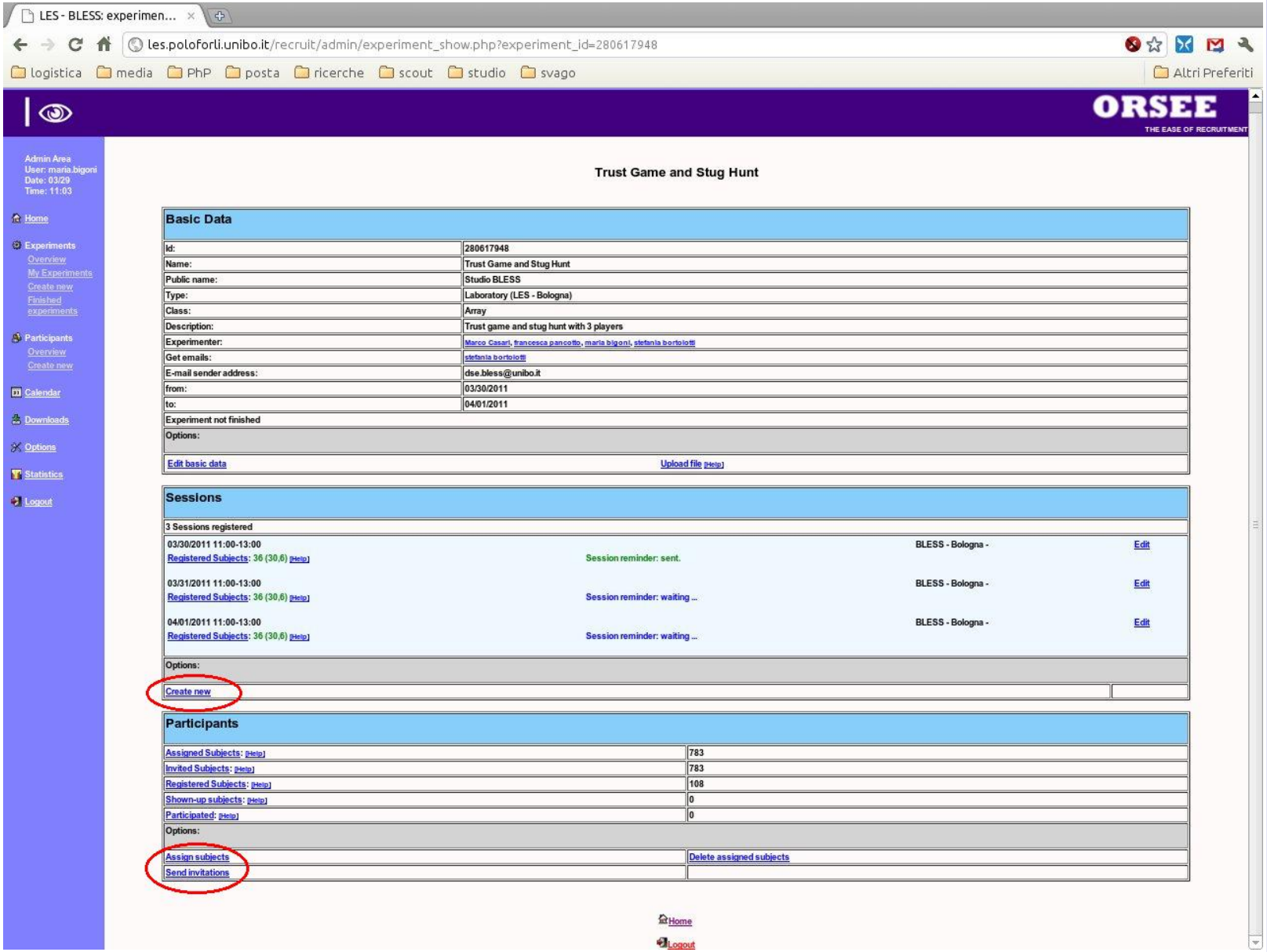
Create a new experiment



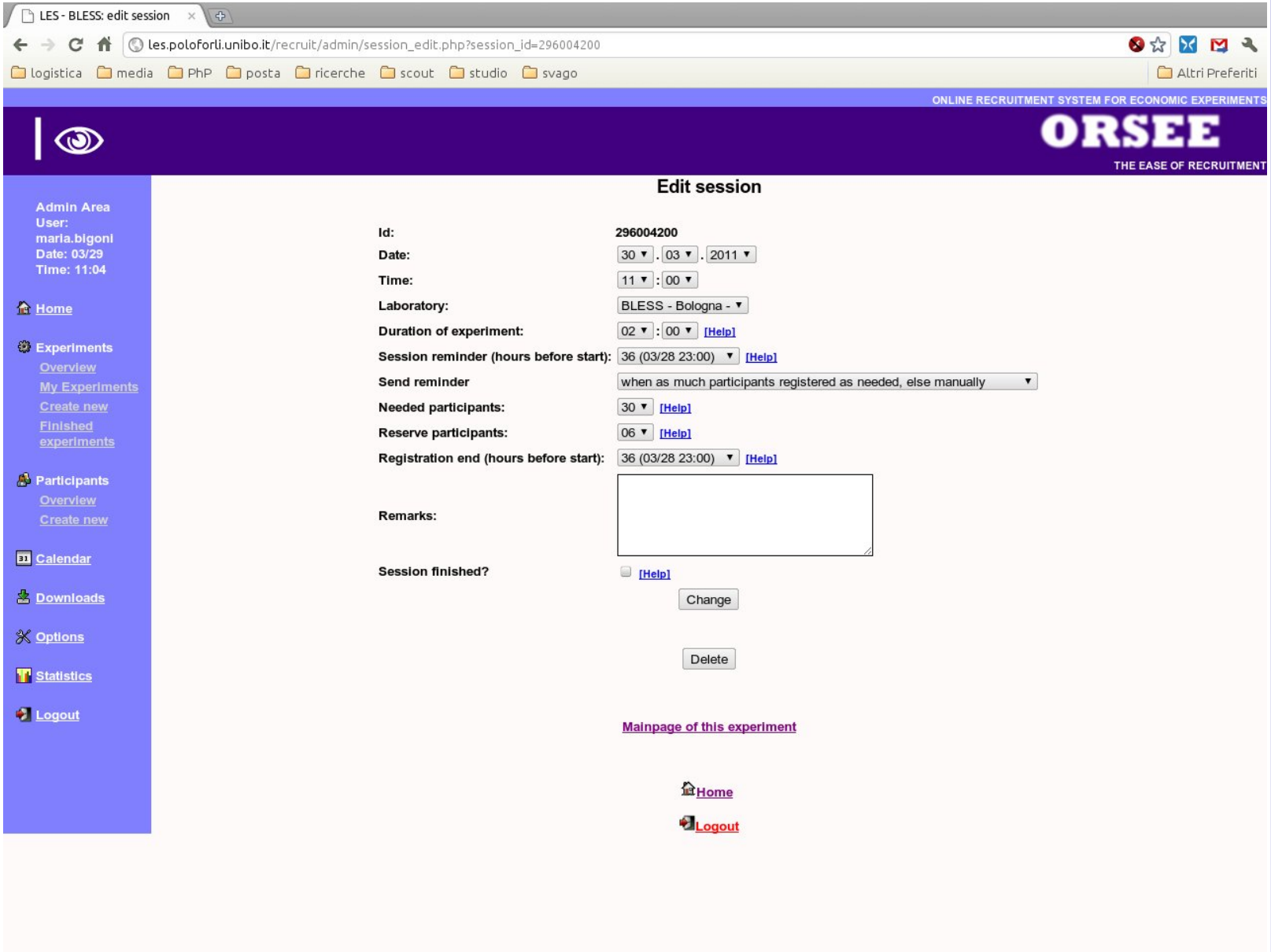
Set the details of the experiment



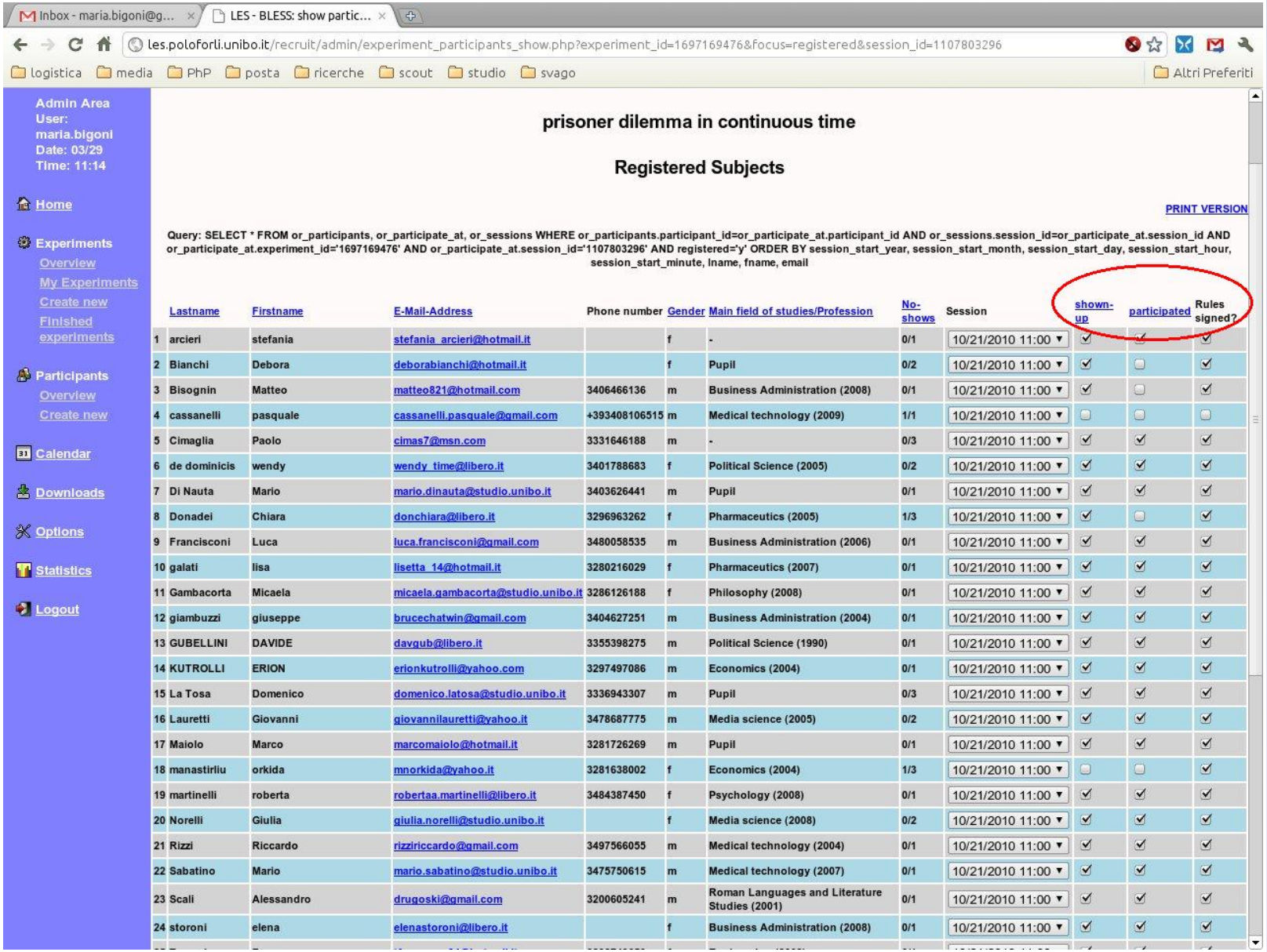
Assign subjects and create sessions



Set the details of the session

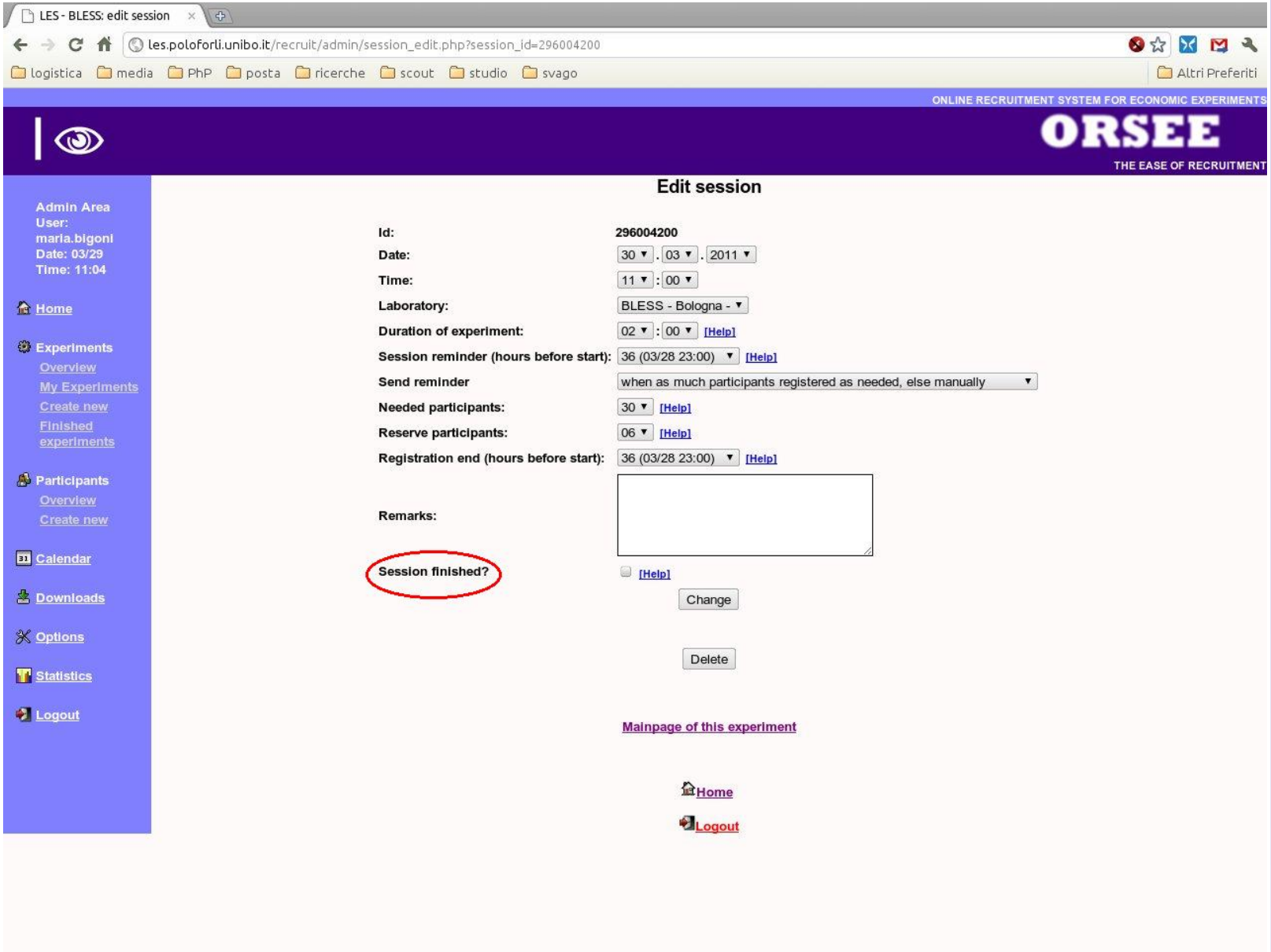


Record subjects' participation

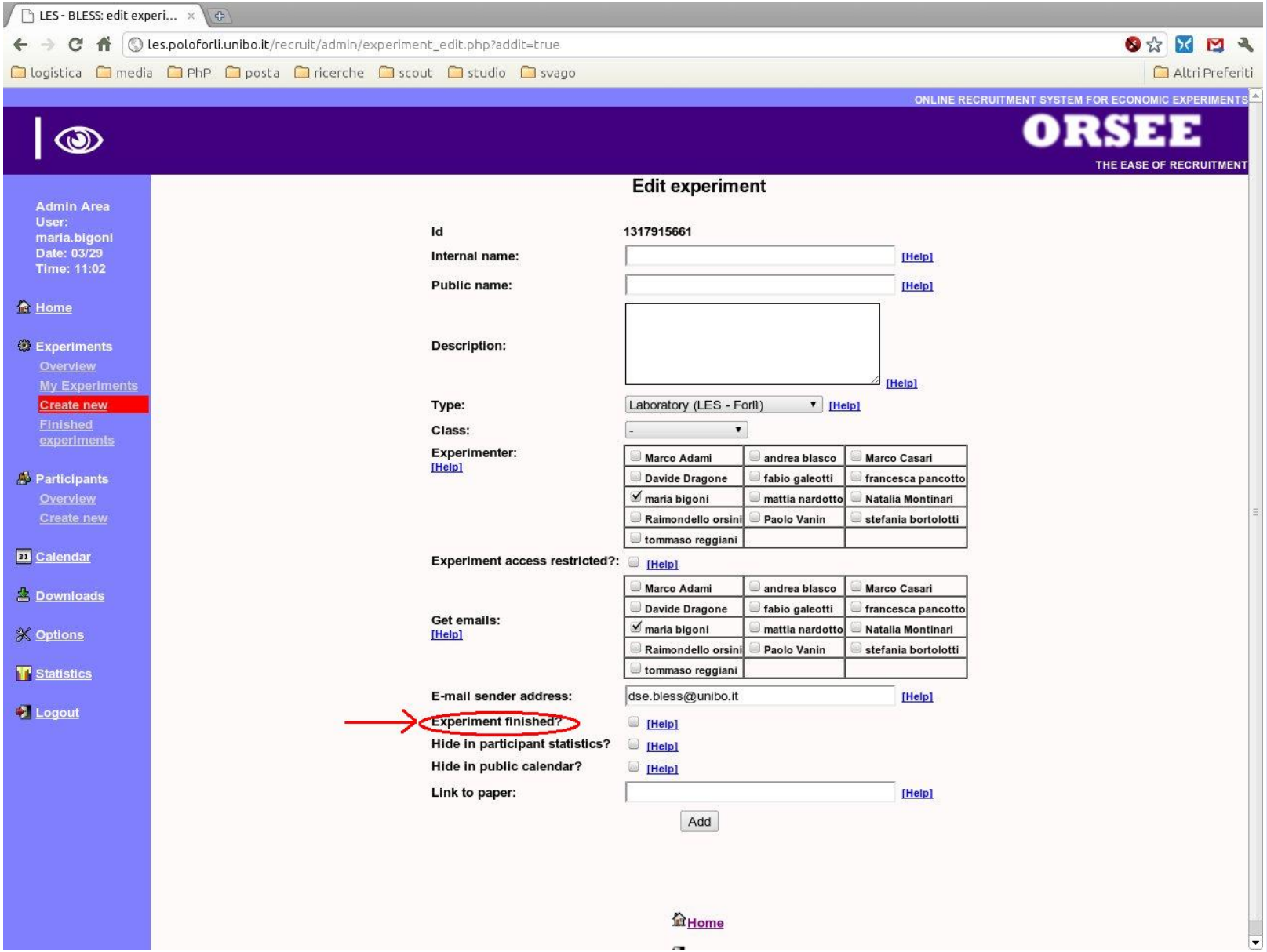


	Lastname	Firstname	E-Mail-Address	Phone number	Gender	Main field of studies/Profession	No-shows	Session	shown-up	participated	Rules signed?
1	arcieri	stefania	stefania_arcieri@hotmail.it		f	-	0/1	10/21/2010 11:00	☑	☑	☑
2	Bianchi	Debora	deborabianchi@hotmail.it		f	Pupil	0/2	10/21/2010 11:00	☑	☐	☑
3	Bisognin	Matteo	matteo821@hotmail.com	3406466136	m	Business Administration (2008)	0/1	10/21/2010 11:00	☑	☐	☑
4	cassanelli	pasquale	cassanelli.pasquale@gmail.com	+393408106515	m	Medical technology (2009)	1/1	10/21/2010 11:00	☐	☐	☐
5	Cimaglia	Paolo	cimas7@msn.com	3331646188	m	-	0/3	10/21/2010 11:00	☑	☑	☑
6	de dominicis	wendy	wendy_time@libero.it	3401788683	f	Political Science (2005)	0/2	10/21/2010 11:00	☑	☑	☑
7	Di Nauta	Mario	mario.dinauta@studio.unibo.it	3403626441	m	Pupil	0/1	10/21/2010 11:00	☑	☑	☑
8	Donadei	Chiara	donchiara@libero.it	3296963262	f	Pharmaceutics (2005)	1/3	10/21/2010 11:00	☑	☐	☑
9	Francisconi	Luca	luca.francisconi@gmail.com	3480058535	m	Business Administration (2006)	0/1	10/21/2010 11:00	☑	☑	☑
10	galati	lisa	lisetta_14@hotmail.it	3280216029	f	Pharmaceutics (2007)	0/1	10/21/2010 11:00	☑	☑	☑
11	Gambacorta	Micaela	micaela.gambacorta@studio.unibo.it	3286126188	f	Philosophy (2008)	0/1	10/21/2010 11:00	☑	☑	☑
12	giambuzzi	giuseppe	brucechatwin@gmail.com	3404627251	m	Business Administration (2004)	0/1	10/21/2010 11:00	☑	☑	☑
13	GUBELLINI	DAVIDE	davgub@libero.it	3355398275	m	Political Science (1990)	0/1	10/21/2010 11:00	☑	☑	☑
14	KUTROLLI	ERION	erionkutrolli@yahoo.com	3297497086	m	Economics (2004)	0/1	10/21/2010 11:00	☑	☑	☑
15	La Tosa	Domenico	domenico.latosa@studio.unibo.it	3336943307	m	Pupil	0/3	10/21/2010 11:00	☑	☑	☑
16	Lauretti	Giovanni	giovannilauretti@yahoo.it	3478687775	m	Media science (2005)	0/2	10/21/2010 11:00	☑	☑	☑
17	Maiolo	Marco	marcomaiolo@hotmail.it	3281726269	m	Pupil	0/1	10/21/2010 11:00	☑	☑	☑
18	manastirliu	orkida	mnorkida@yahoo.it	3281638002	f	Economics (2004)	1/3	10/21/2010 11:00	☐	☐	☑
19	martinelli	roberta	robertaa.martinelli@libero.it	3484387450	f	Psychology (2008)	0/1	10/21/2010 11:00	☑	☑	☑
20	Norelli	Giulia	giulia.norelli@studio.unibo.it		f	Media science (2008)	0/2	10/21/2010 11:00	☑	☑	☑
21	Rizzi	Riccardo	rizziriccardo@gmail.com	3497566055	m	Medical technology (2004)	0/1	10/21/2010 11:00	☑	☑	☑
22	Sabatino	Mario	mario.sabatino@studio.unibo.it	3475750615	m	Medical technology (2007)	0/1	10/21/2010 11:00	☑	☑	☑
23	Scali	Alessandro	drugoski@gmail.com	3200605241	m	Roman Languages and Literature Studies (2001)	0/1	10/21/2010 11:00	☑	☑	☑
24	storoni	elena	elenastoroni@libero.it		f	Business Administration (2008)	0/1	10/21/2010 11:00	☑	☑	☑

Close the session



Close the experiment



Text formatting - 1

This paragraph is not formatted.

This paragraph is formatted.

This paragraph is separated by tabs.

This paragraph is bold.
This is a new paragraph.
This paragraph is italic.

This paragraph is centered.

This one is aligned to the left.

And this one is aligned to the right.

"\line" starts a new
line.

This is a list:
- first bullet point;
- second bullet point;
- third bullet point;

This is a subscript and this is a superscript.

this is underlined. ~~this is deleted.~~

You can also change the size of the font, making it bigger.

Finally, ou can also change the color of the text, making it RED, GREEN, or BLUE, or highlighting it in YELLOW.

Examples:

- text_formatting.ztt: source code for the above example,
- formatting_exercize.ztt: colored text, dynamically changing size,
- timer.ztt: a big, colorful timer.

Text formatting - 2

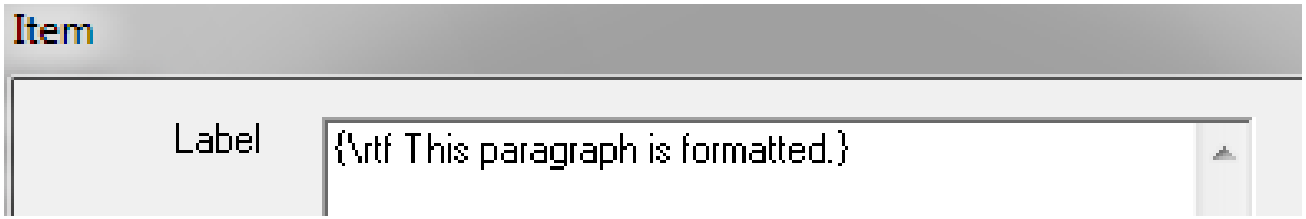
see the Reference manual, at page 55.

Text can be formatted in:

- standard boxes
- grid boxes
- contract creation boxes
- contract grid boxes

The RTF format begins with {\rtf (with a blank space at the end), and ends with }.

Example:



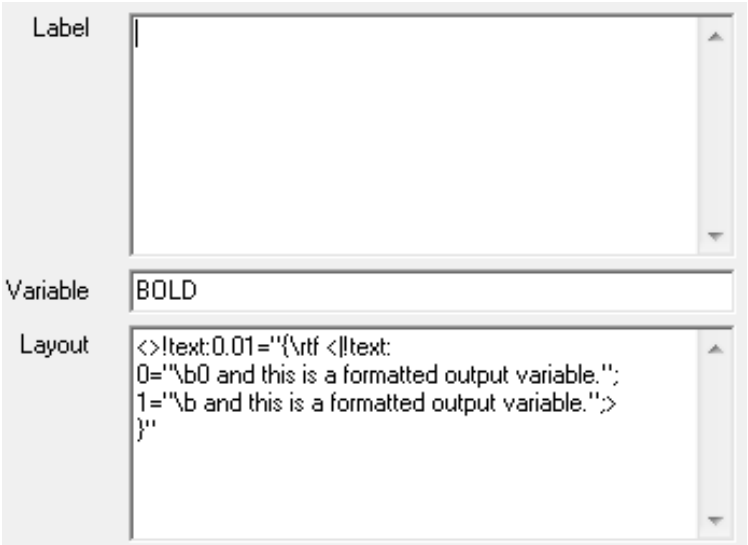
RTF formatting instructions

\tab	tabulator
\line	new line
\ql	aligned to the left
\qc	centred
\qr	aligned to the right
\strike	crossed through
\ul(\ul0)	underlined (not underlined)
\par	new paragraph
\bullet	thick dot (for lists)
\b (\b0)	bold (not bold any more)
\i(\i0)	italics (not italics any more)
\sub	subscript
\super	superscript
\fsn	font size in units of half a dot

Conditional Formatting

The insertion of variables is carried out *before* the interpretation of RTF instructions. This make conditional formatting possible, as in the following example: when the `BOLD` variable is 1, "hallo" should be shown in boldface, otherwise it is shown in plain text.

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



N.B. Labels do not change dynamically when the value of the variable changes.
→put formatting in the layout field.
Example:
conditional_formatting.ztt

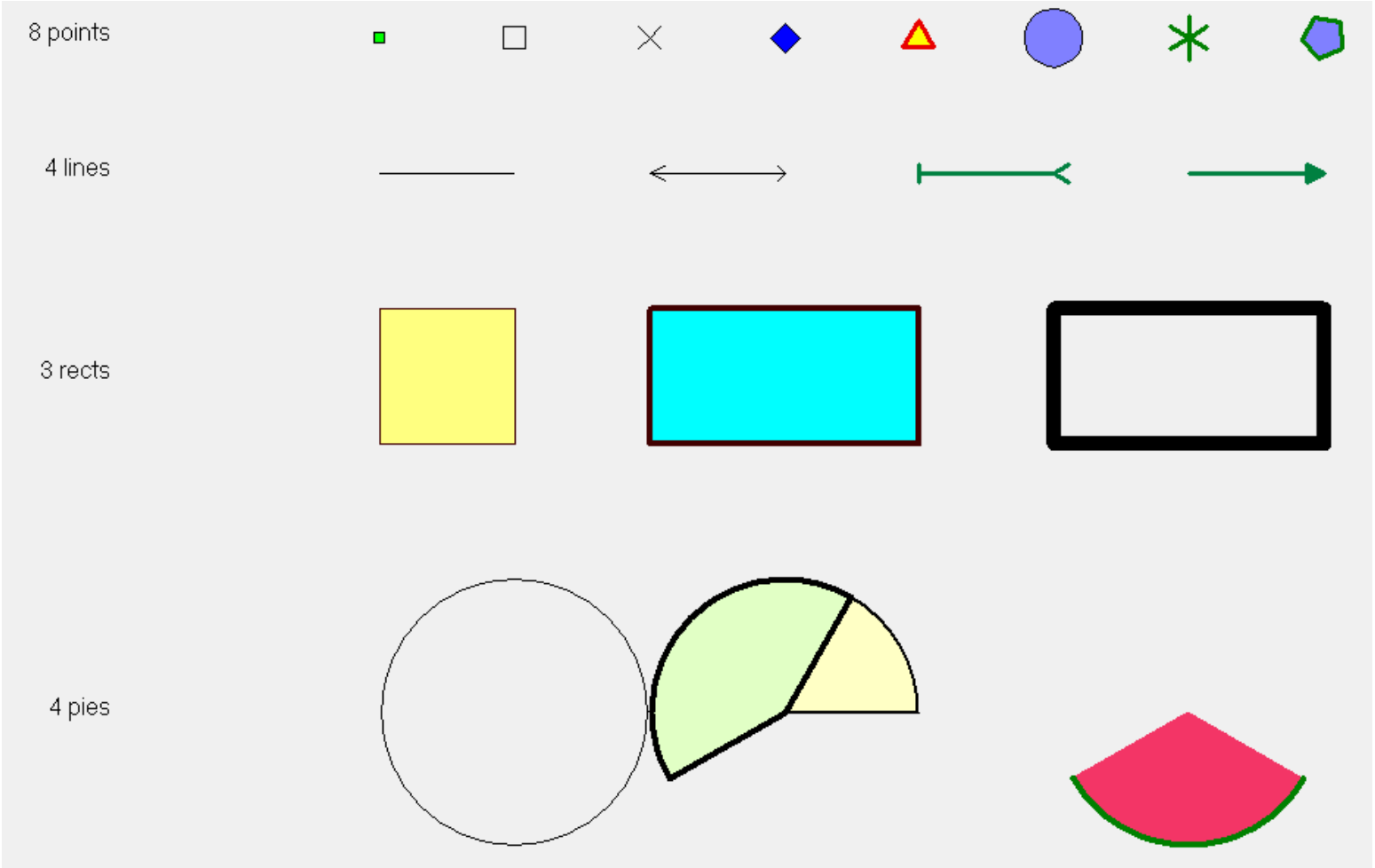
Text Color

With RTF formatting, you have to define first the colors you are going to use, with the instruction `\colortbl`.

```
{\rtf {\colortbl;
  \red0\green0\blue255;
  \red255\green0\blue0;
  \red0\green255\blue0;
  \red255\green255\blue255;
  }
\cf1 this is BLUE
\cf2 this is RED
\cf3 this is GREEN
\cf4 this is WHITE
}
```

Plots and Vectorial Graphics

See the z-Tree wiki



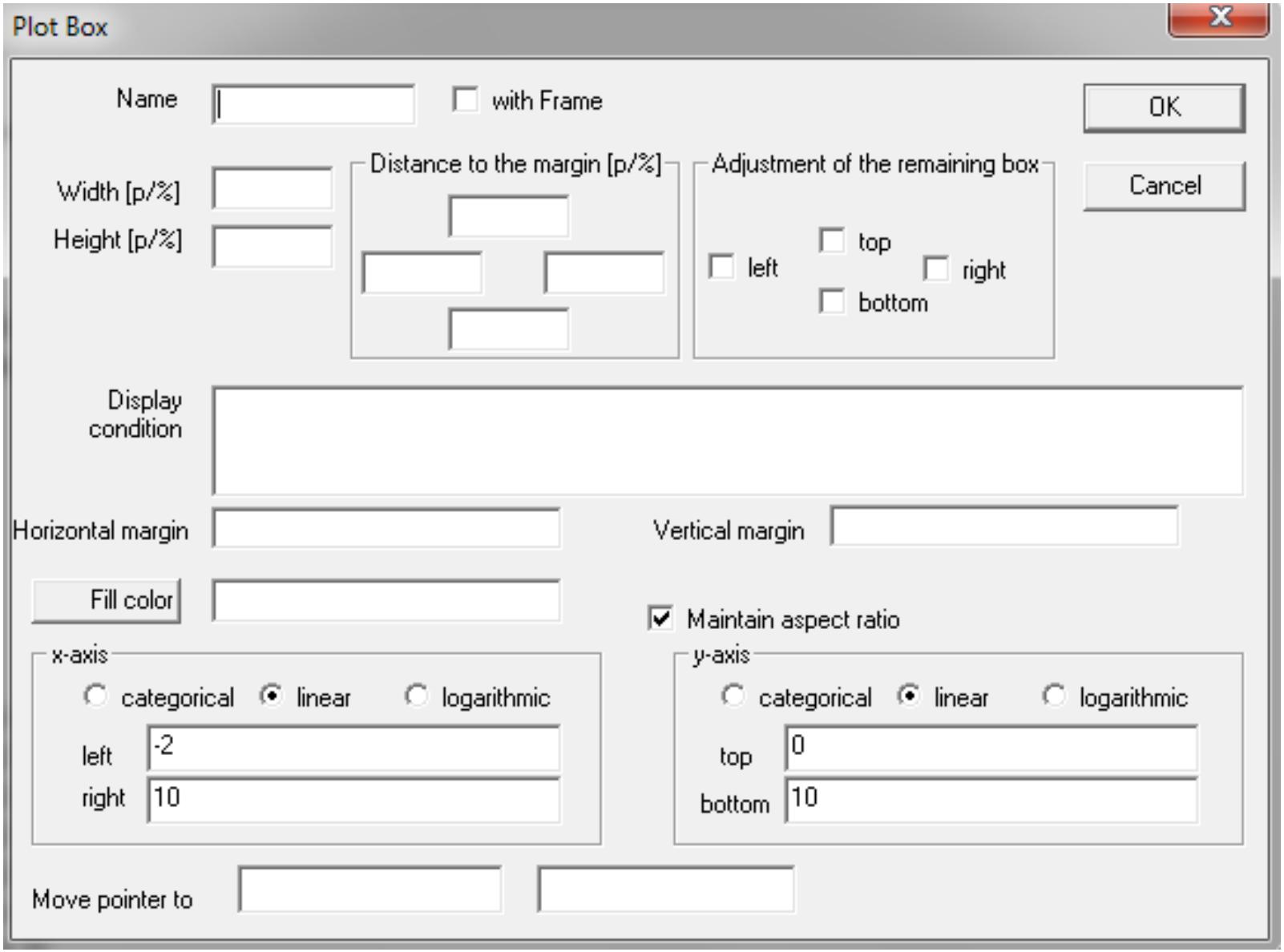
Example: plotitems.ztt

Plot Box - I

- The `Plot Box` sets up a **coordinate system** and allows to display Plot Items.
- Plot items are drawn **sequentially**. So items more down in the list of items **can cover** items more up in the list.
- The `Plot box` is **opaque**. It covers all boxes below.
- The `Plot box` is a **Box**. It can be contained in Screens and Container Boxes. It can contain Plot Items.

To create a new Plot Box, select `Treatment`→`New Box`→ `New Plot Box`

Plot Box - II

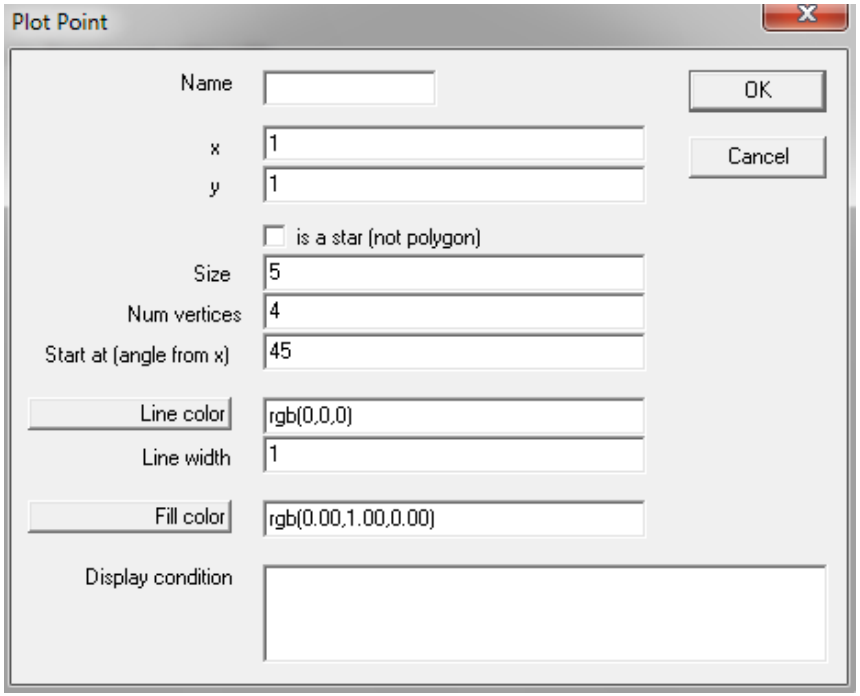

Plot Box
Name
with Frame
OK
Cancel
Width [p/%]
Height [p/%]
Distance to the margin [p/%]
Adjustment of the remaining box
left
top
right
bottom
Display condition
Horizontal margin
Vertical margin
Fill color
Maintain aspect ratio
x-axis
categorical
linear
logarithmic
left
-2
right
10
y-axis
categorical
linear
logarithmic
top
0
bottom
10
Move pointer to

Plot items

- In a plot box different types of graphical elements (points, lines, rectangles,...) can be displayed. These elements are called **plot items**.
- They can be placed into the **coordinate system** that is set up in the plot box.
- The **position** of the plot items is thus defined with reference to this coordinate system.
- Furthermore, these elements have **features**, for instance the lines can be thicker or thinner and drawn in different colors.
- The size used to describe these features is always in screen **pixels**.

Points

To add a Point, select `Treatment`→`Graphics`→ `New Point`. This displays a point either as a regular polygon or as a star consisting of line elements.



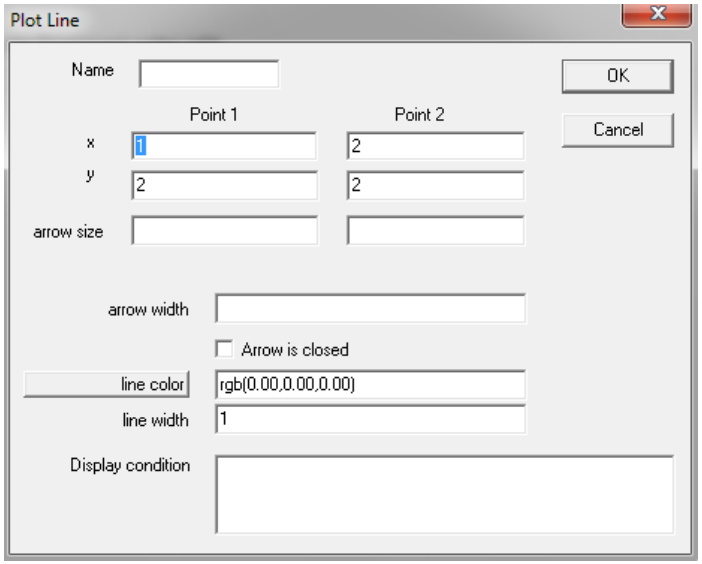
`Plot points are plot items. They can be` **contained**

- in `plot boxes`
- and in `plot graphs`.

`Start at (angle from x):` Angle where the first point is drawn. The angle is measured from the x-axis in counter clockwise direction.

Lines

To add a Line, select `Treatment`→`Graphics`→ `New Line`. This displays a **connection** between two points.



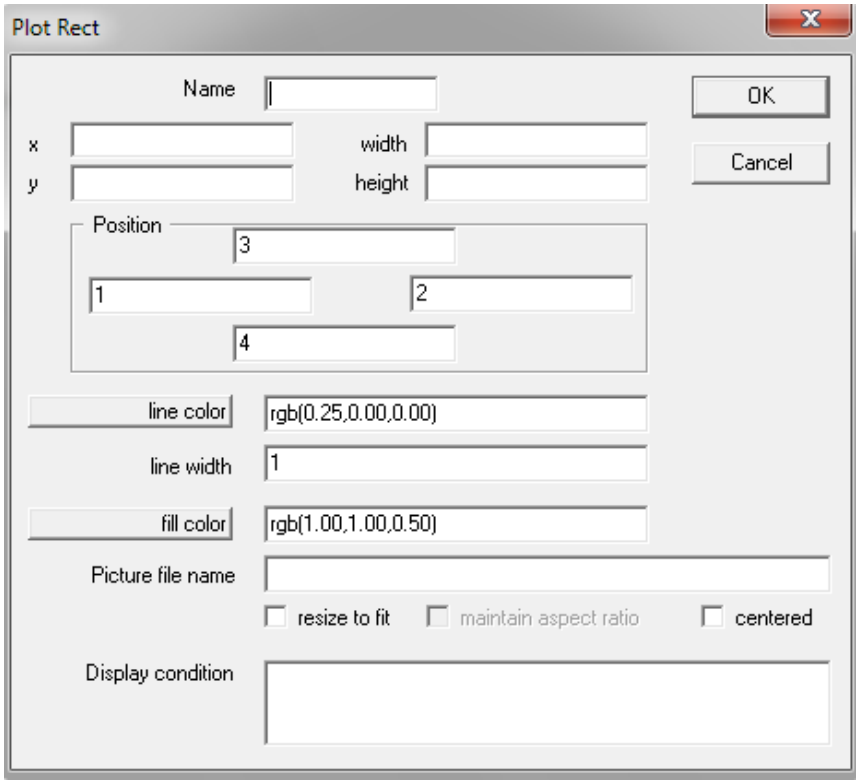
It can be a straight line or an one-sided or two-sided arrow. `Plot lines` are `plot items`. They can be **contained**

- in `plot boxes`
- and in `plot graphs`.

`Arrow size:` Size of the arrow in the direction of the line. The arrow size can be negative, in which case the arrow points from the outside to the line.
`Arrow width:` Total size of the arrow orthogonal to direction of the line.
`Arrow is closed:` If not checked: arrow consists of two lines. If checked, the arrow is a filled triangle.

Rectangles

To add a Rectangle, select `Treatment`→`Graphics`→ `New Rectangle`.



`Plot rectangles` are `plot items`. They can be **contained**

- in `plot boxes`
- and in `plot graphs`.

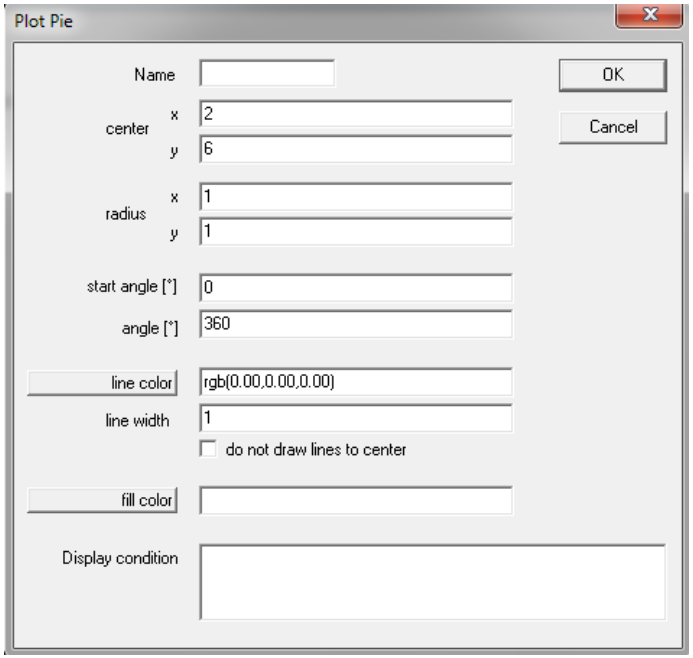
`x/y:` coordinates of the center of the rectangle.

`width/height and position:` defined with respect to the coordinate system that is set up in the plot box.

Pies

To add a Pie, select `Treatment`→`Graphics`→ `New Pie`. `Plot pies` are `plot items`. They can be **contained**

- in `plot boxes`
- and in `plot graphs`.



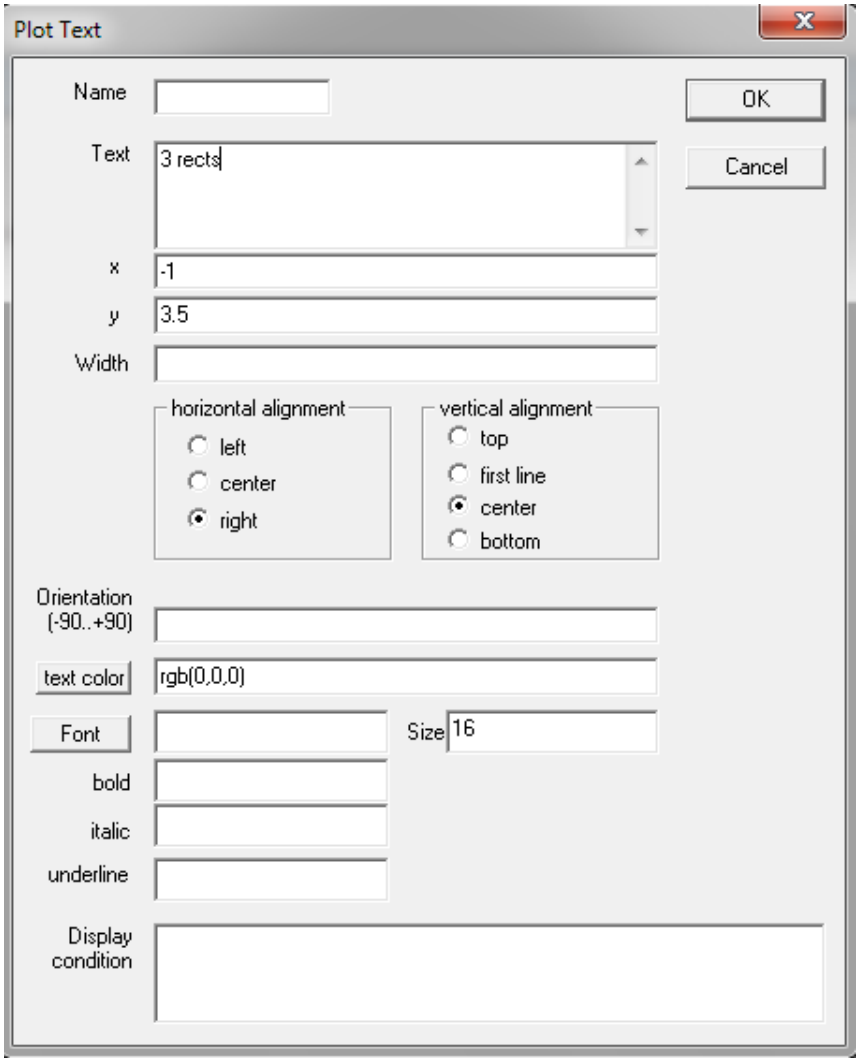
`start angle`: Angle where the segment starts, measured in degrees (the full circle has 360). The angle is measured from the x-axis in counter clockwise direction.

`angle`: Size of the segment in degrees. For a full circle, you enter 360. The segment goes in counter clockwise direction from the start angle.

Example: spin_the_wheel.ztt: a rotating pie, representing the outcome of a lottery.

Text

To add a text to a plot, select `Treatment→Graphics→ New Text`.



`Plot texts are plot items. They can be` **contained**

- in `plot boxes`
- and in `plot graphs`.

`x/y:` position of the text.

`Horizontal/vertical alignment:` Defines which border of the text is determined by the text position.

`Orientation:` Does not yet work.

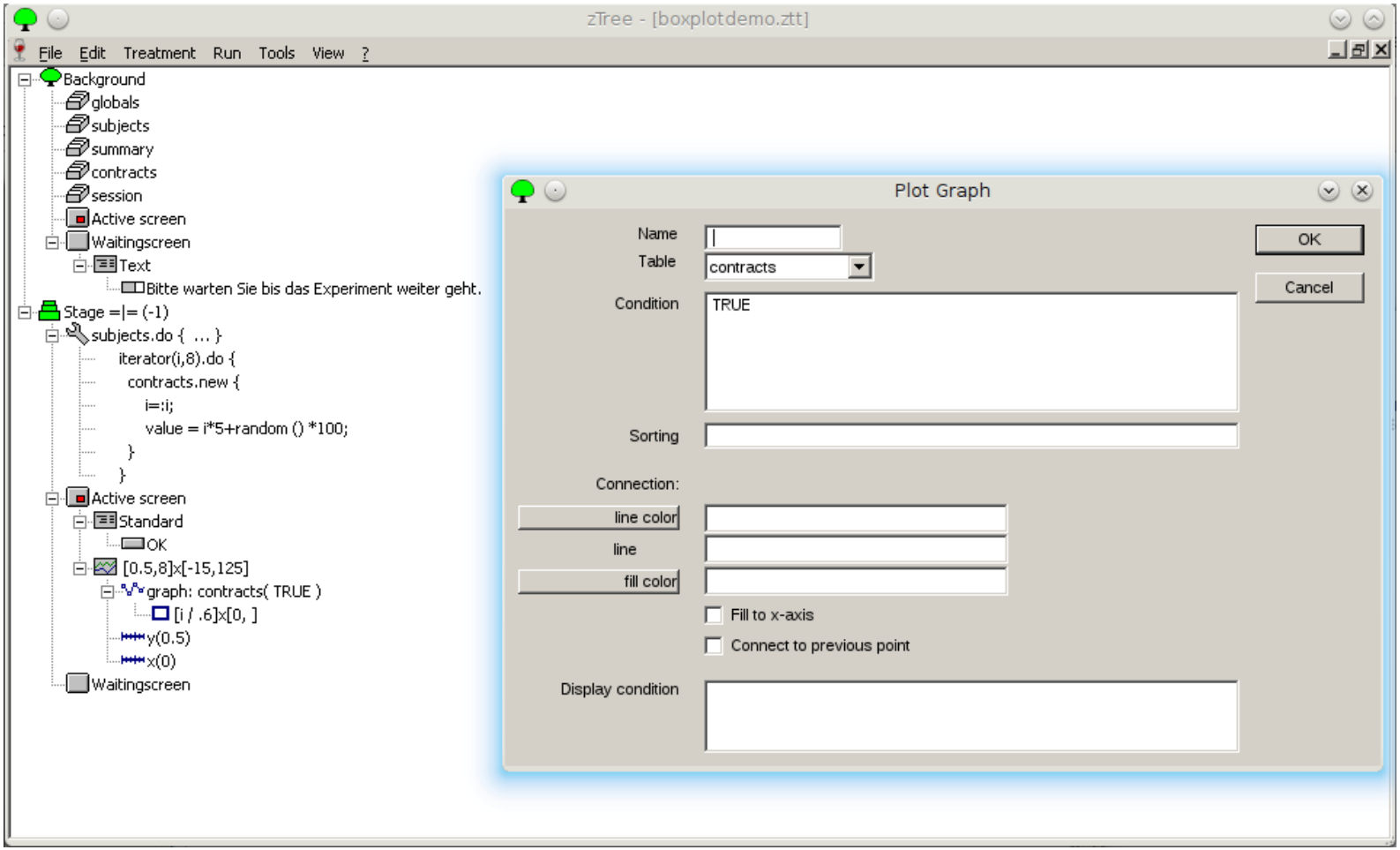
Graphs

To add a text to a plot, select
`Treatment`→`Graphics`→ `New Plot Graph`.

- The `plot graph` is used to display **polygons or series of data**.
- The `plot graph` displays series of **records in a table**. Plot items that are placed into the plot graph are drawn **for every record** and the variables used in these items are evaluated in this record.
- `Plot graphs` **can be nested**, i.e., placed into each other. If they are nested the data from different records can be accessed with the scope operator.
- `Plot graphs` are `plot items`: they can be contained in `plot boxes` and in `plot graphs`.
- They can contain `plot items`.

Example: boxplotdemo.ztt

Graphs: example



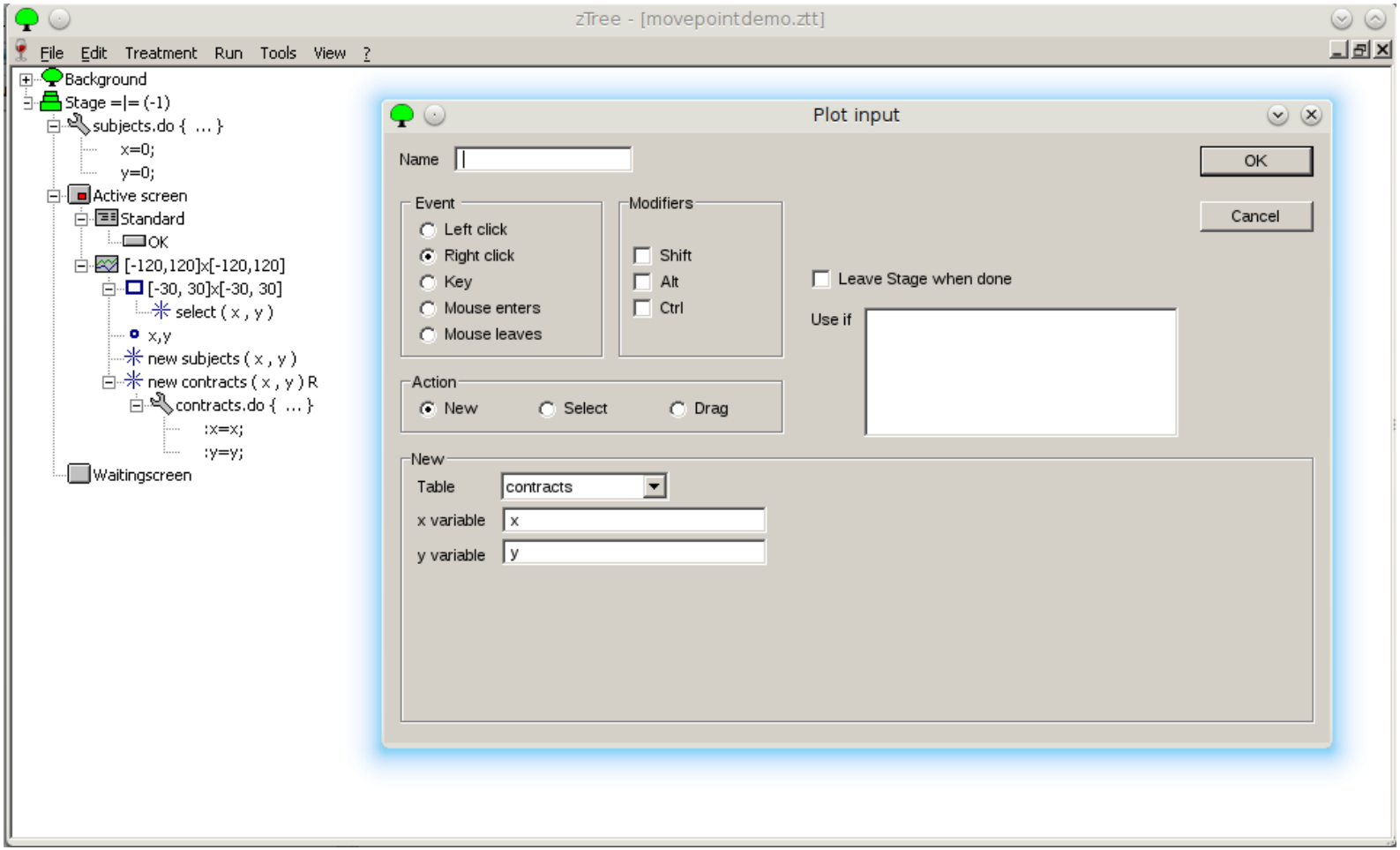
Plot inputs

To add a text to a plot, select
`Treatment`→`Graphics`→ `New Plot Input`.

- `Plot inputs` can be placed into `plot boxes` and `plot items`.
- They can contain `checkers` and `programs`.
- With `plot input` item, you define how **subjects interact** in a `plot box`;
- Subjects can **click** at a new position, they can **select** one of several objects on the screen, and they can even **drag** objects around.

Example: movepointdemo.ztt

Plot inputs: example



Multimedia - I

See the z-Tree wiki. The multimedia box can contain **images**, **movies** or **sounds**.

- The content, i.e. the image, movie or sound of the multimedia box is stored in an **external file**.
- This file must be accessible at the client's computer.
- The best way to achieve this is to put the files **on a server share** and map this share to the same drive letter on each of the client computers.
- Movies and sound are played as long as the box is visible. This allows to **turn them on and off**.

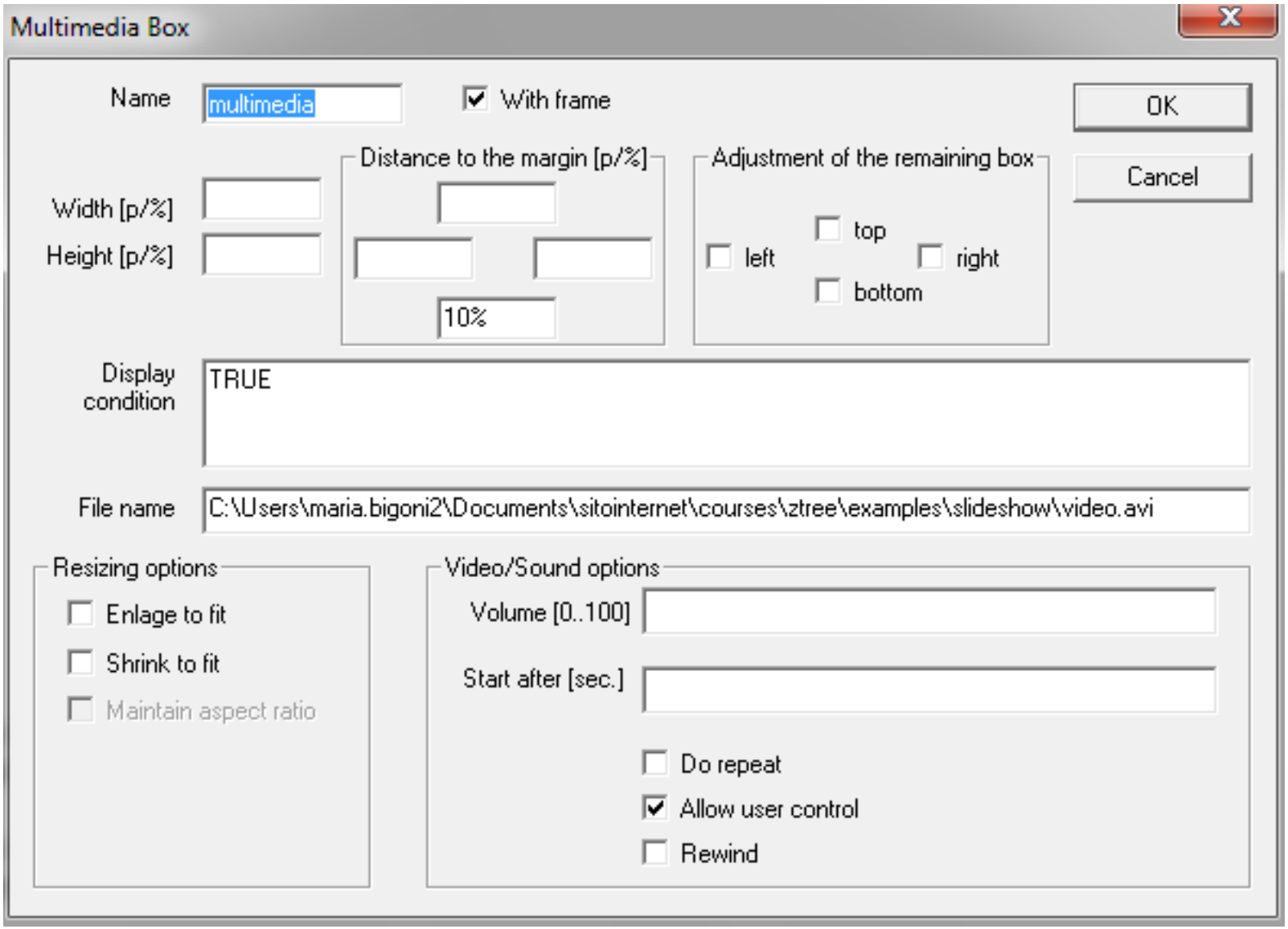
Multimedia - II

The following **formats** should work on a standard installation:

- Image: jpg, gif, png, bmp.
- Movie: mpg, avi.
- Sound: wav, mp3.

The `multimedia box` is a Box. It can be contained in `Screens` and `Container Boxes`. It cannot contain other elements.

Multimedia Box



Slide-show - I

From z-Tree version 3.3.0 it is possible to include in the treatment a slide-show, i.e. a **set of pictures displayed in sequence**.

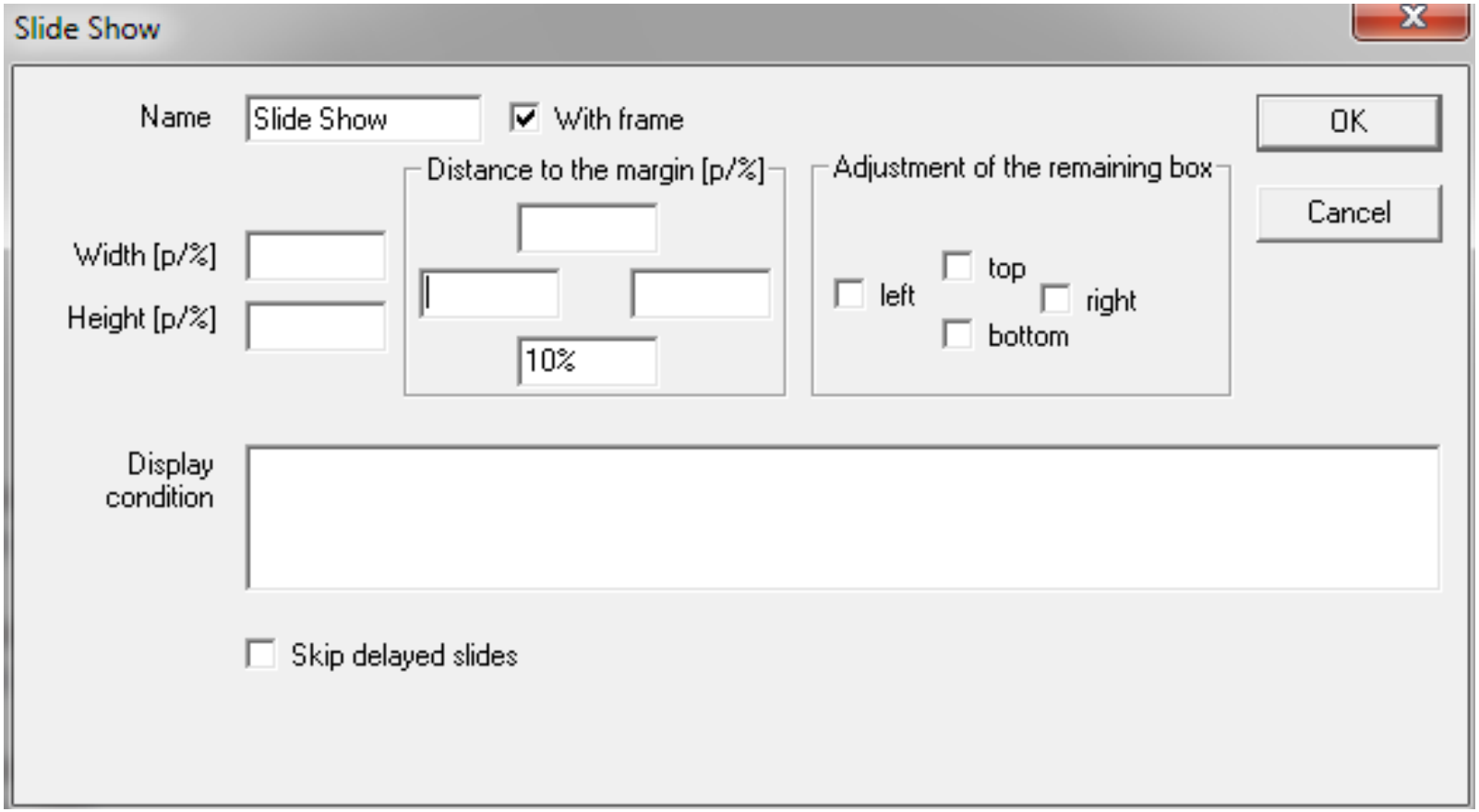
Example: slideshow.ztt

To create a slide-show, the following **procedure** should be followed:

1. in the `Background`, create a new `Table` and name it "slides" (or as you like)
2. in the `Background`, create a new `Program`, running on the globals table.
   - In this program, write:
     `slides.new{}`
   - this simply generates a new empty record in the table "slides".

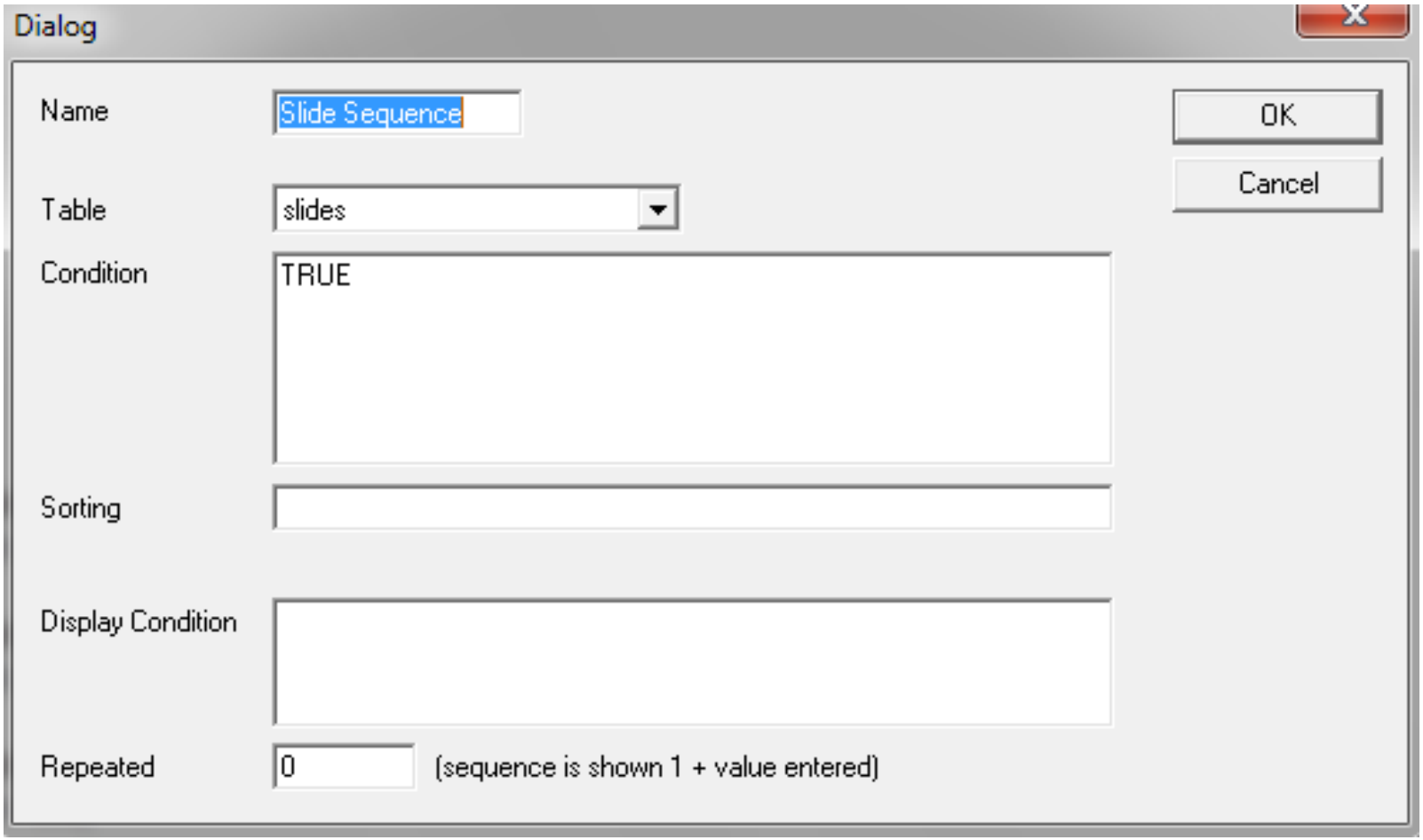
Slide-show - II

3. in the `Active Screen` of a stage, click on `Treatment` → `New Box` → `New Slide Show`.



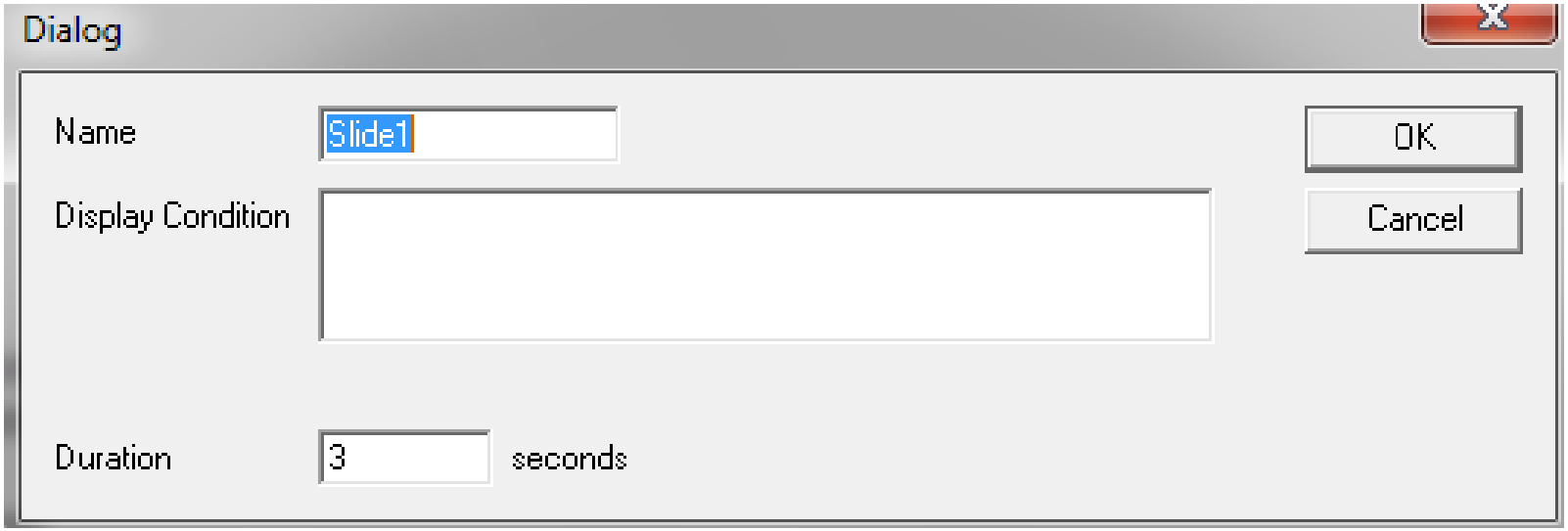
Slide-show - III

4. Within the slide-show, click on `Treatment` → `Slide show` → `New Slide Sequence`.

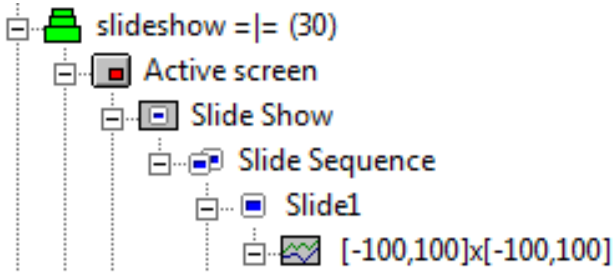


Slide-show - IV

5. Within the slide sequence, click on `Treatment` → `Slide show` → `New Slide`.

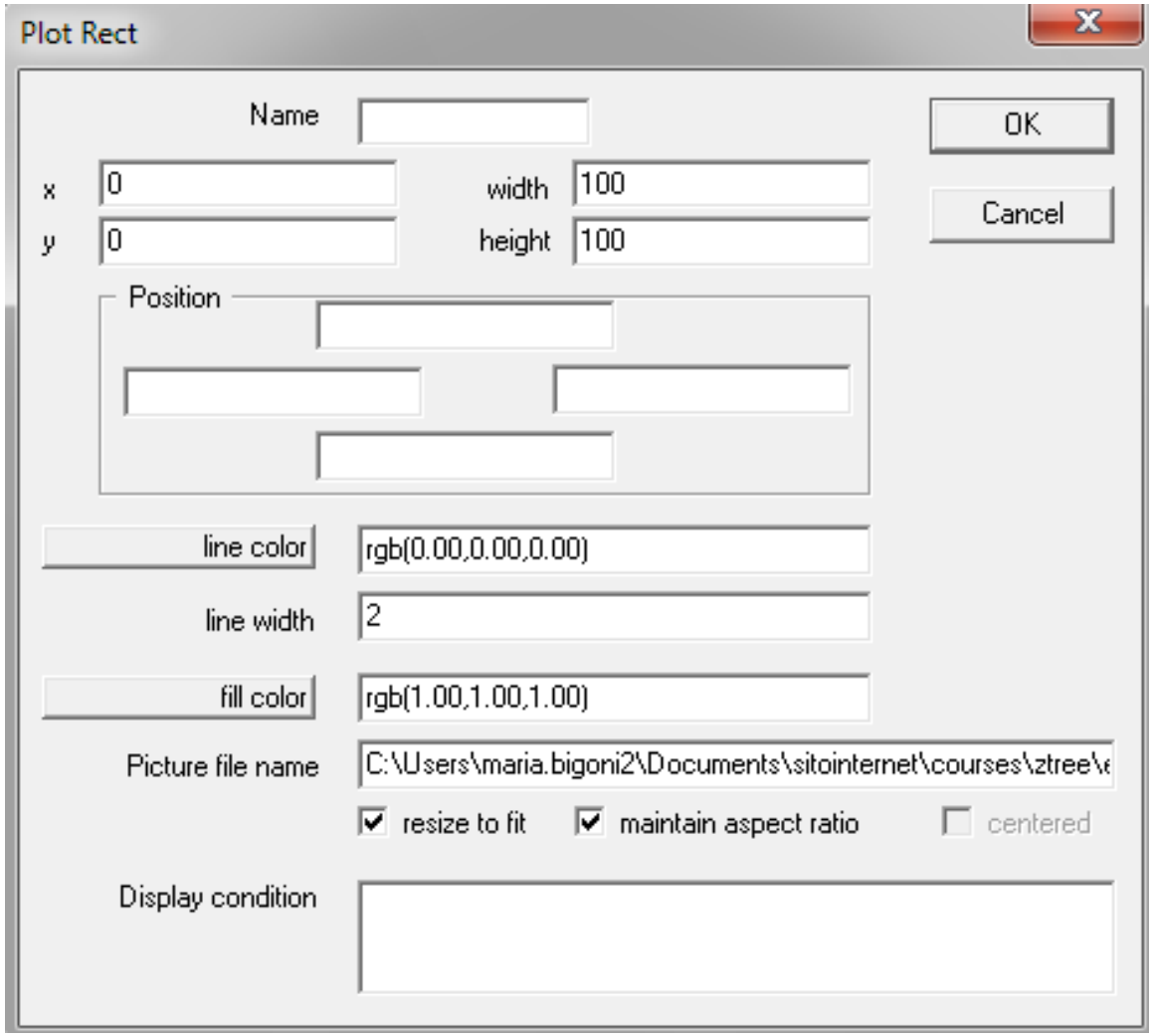


6. Within the slide, click on `Treatment` → `New box` → `New plot box`.



Slide-show - V

7. Within the plot-box, click on `Treatment` → `Graphics` → `New rect`.



External programs

From z-Tree version 3.3.0 it is possible to call external programs, from z-Tree or from z-Leaf.

In the Background, or within a stage, click on `Treatment` → `New External Program`

Part VIII

Advanced programming examples

Advanced programming examples

Dutch auction – fully graphical version

- the background
- the graphical clock
- other elements

Exercise: graphical version of an English auction

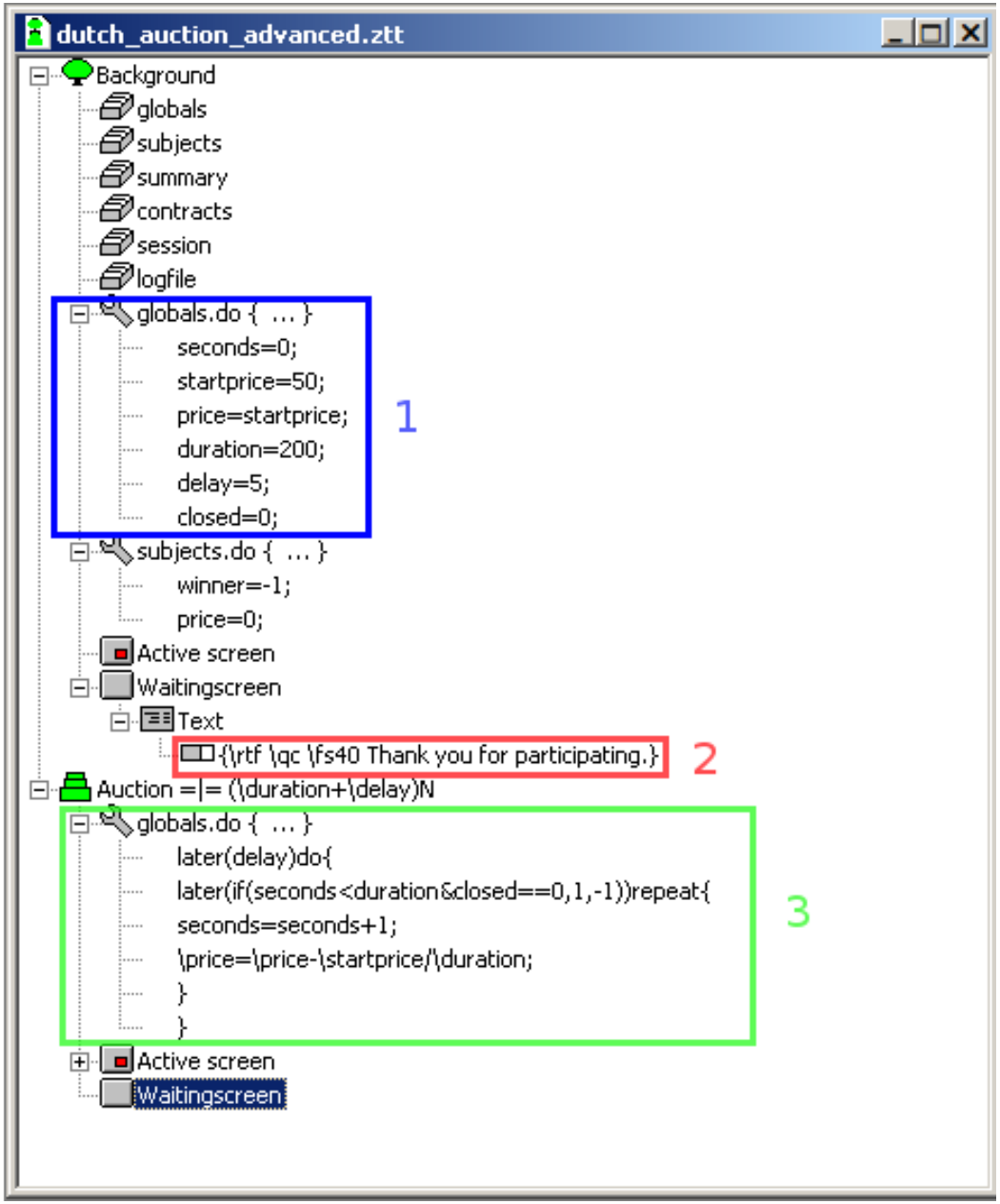
A real effort task: pick the right color

- background
- main stage

Dutch auction - screenshot

Example: dutch_auction_advanced.ztt

Dutch auction - the background



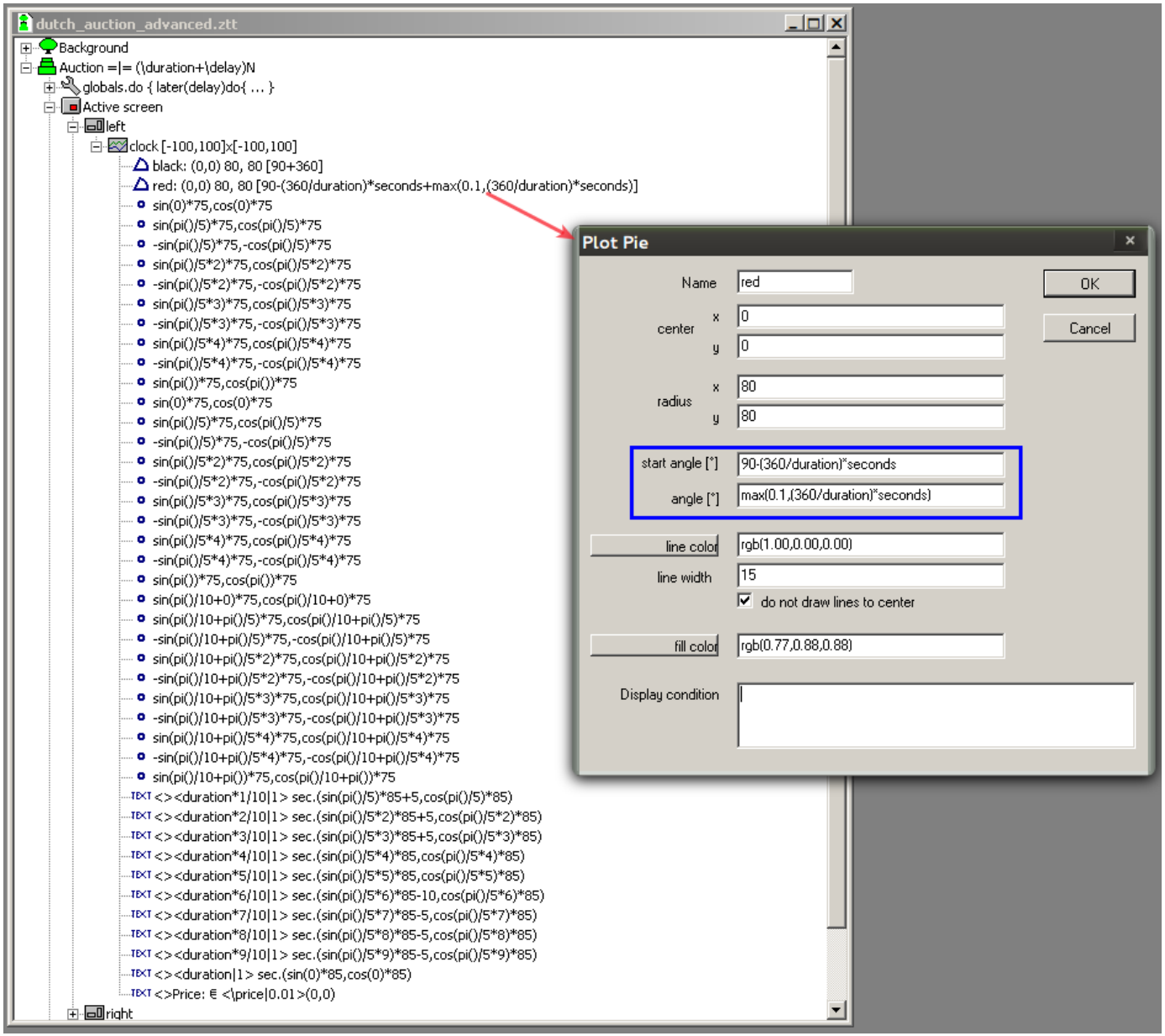
Dutch auction - the background

1. define the **global variables**

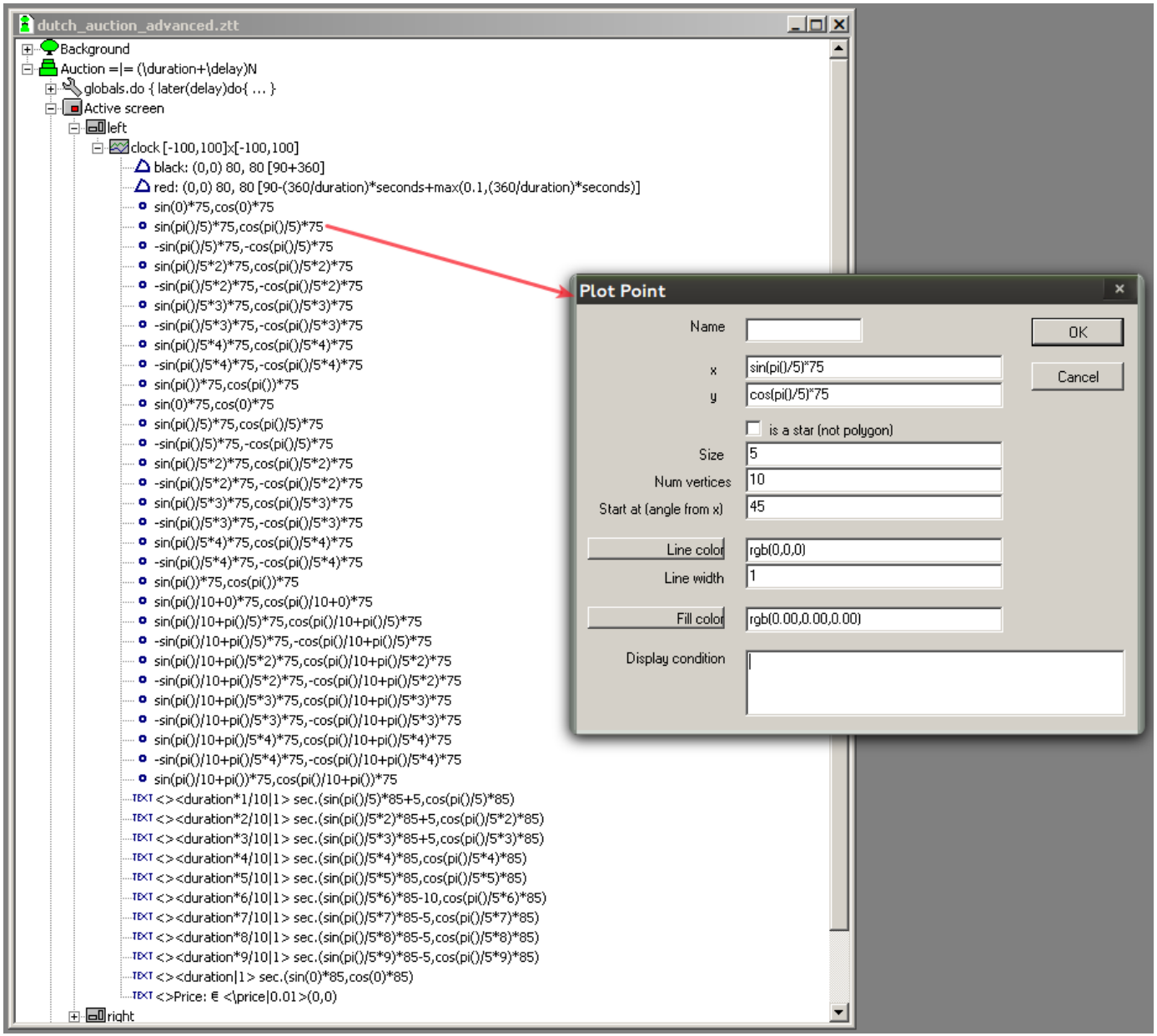
2. set the message for the **waiting screen**

3. let the **clock** run, using
   - the `later()do` statement
   - and the `later()repeat` statement

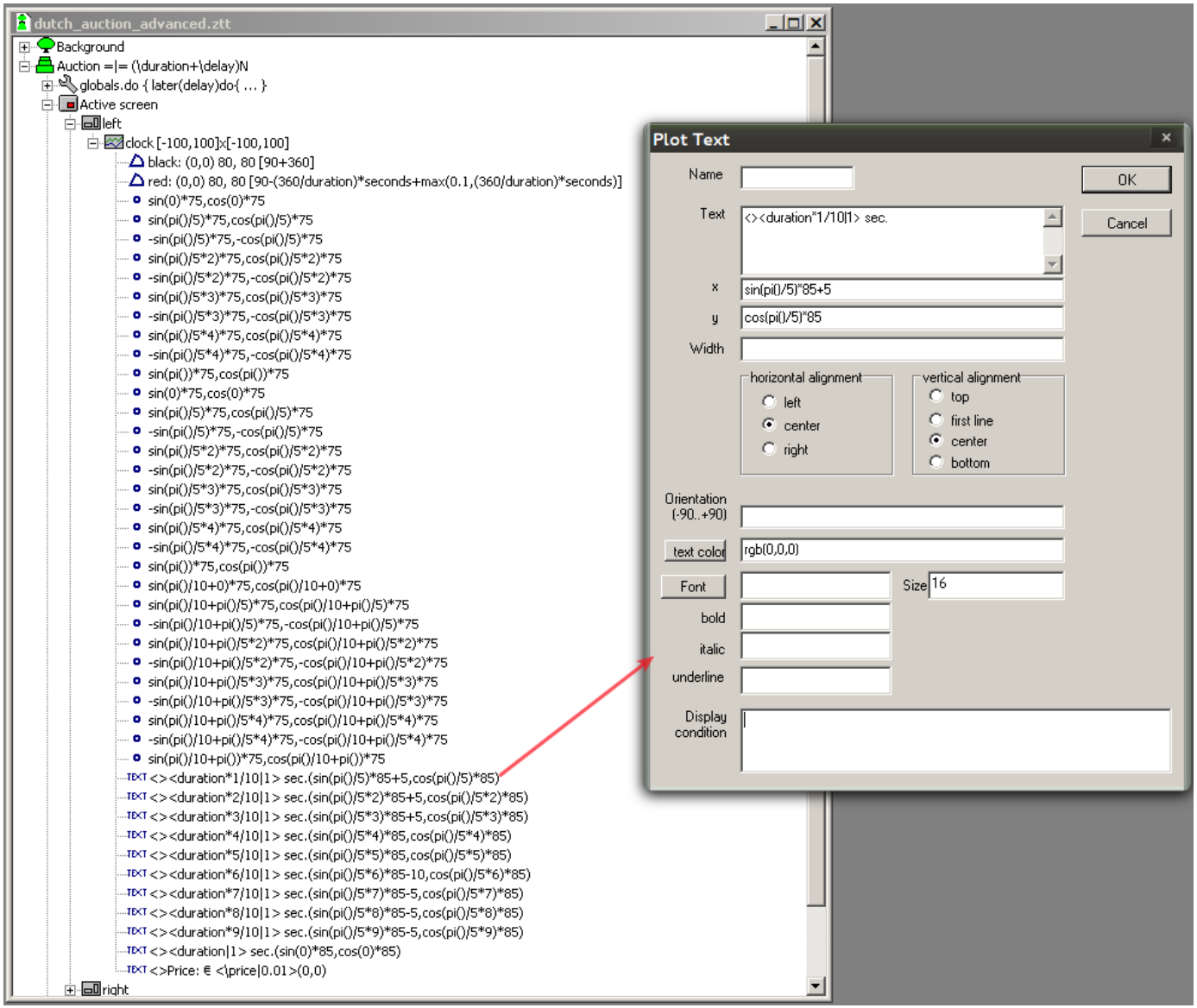
Dutch auction - the graphical clock I



Dutch auction - the graphical clock II

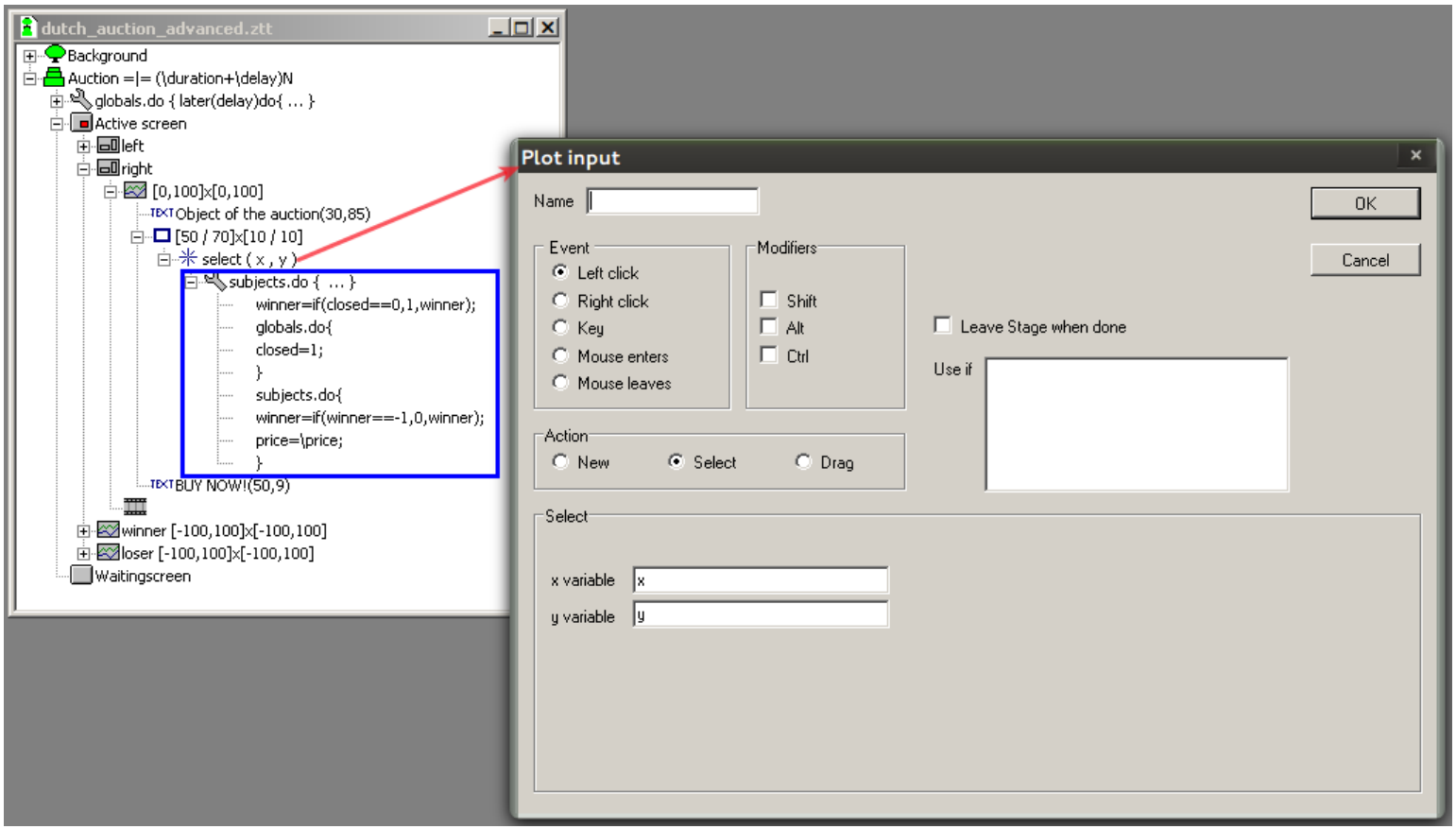


Dutch auction - the graphical clock III



Dutch auction - the input

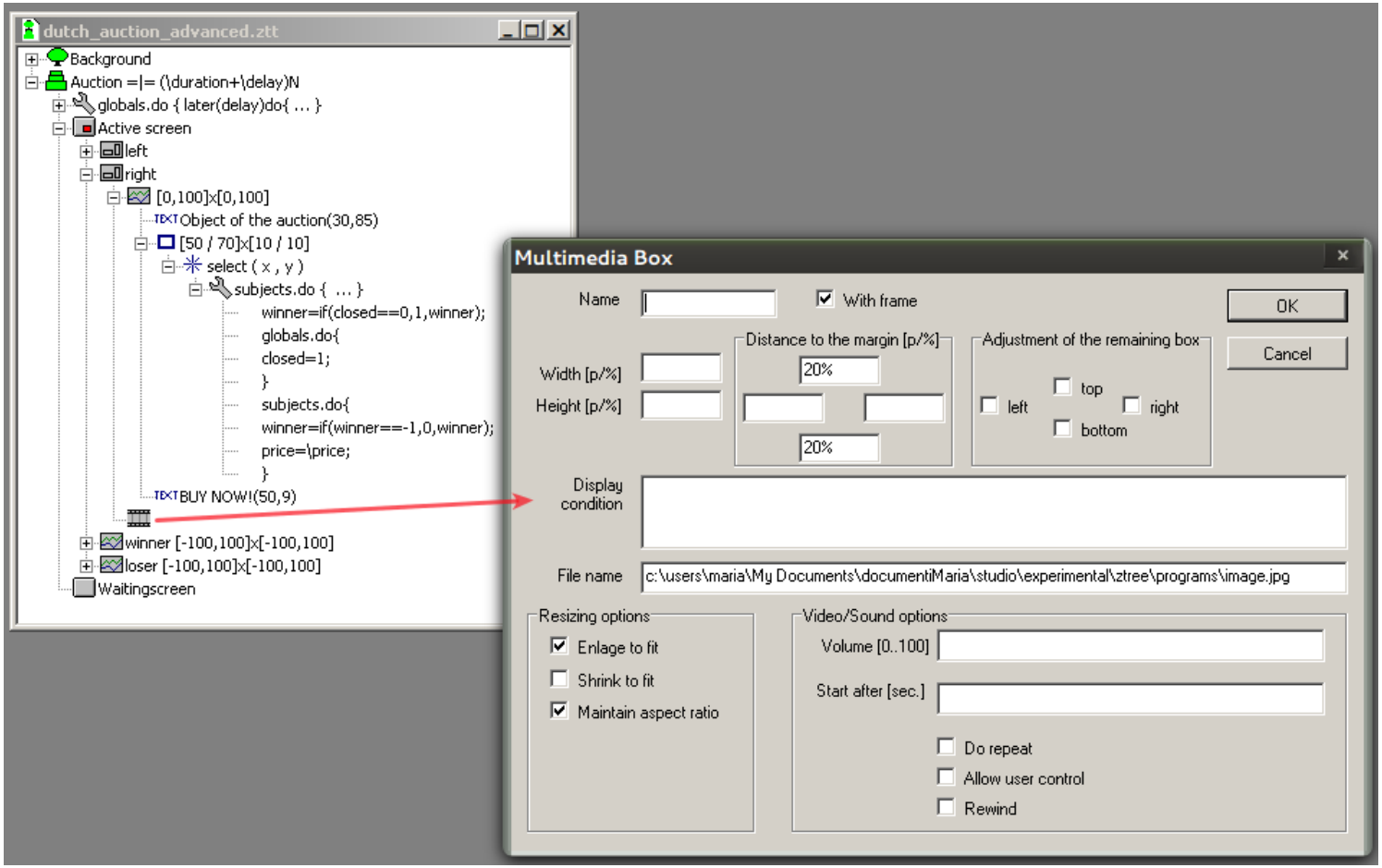
1. Transform a `Rectangle` in a button, by adding a `plot input`.
2. Add a `program` to trigger the consequences of the subject's action.



Dutch auction - the figure

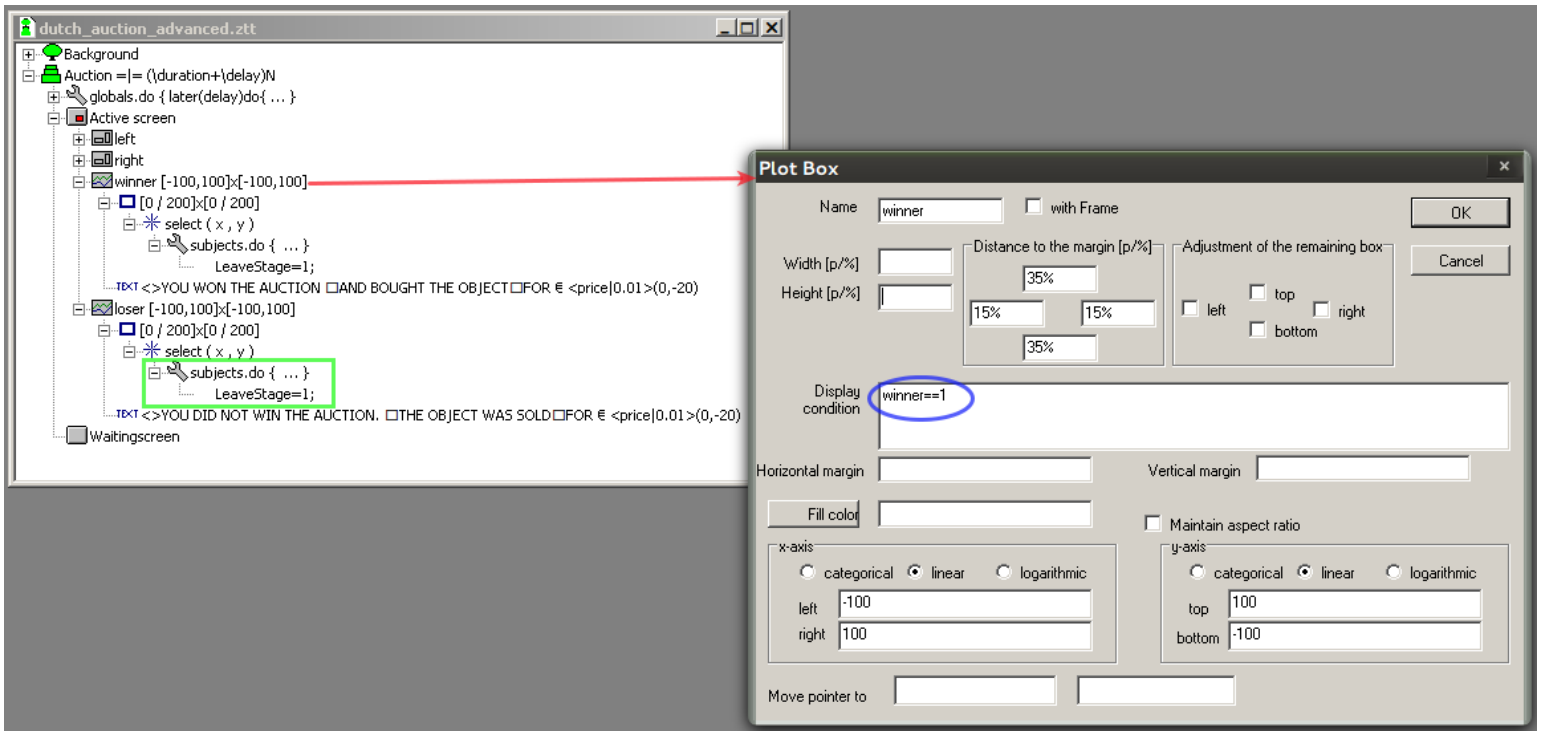
Use a `Multimedia box` to insert the picture of the object of the auction.

Remember to write the full file path.



Dutch auction - the final message

1. Use the `Display condition` to show different messages to the winner of the auction and to the other subjects.
2. With a `program` within a `plot input`, you let the subjects leave the stage when they click on the final message.



Exercise: English auction

solution: english_auction_advanced.ztt

Exercise: what changes?

1. price **increases** in time

2. subject's action: **leave** the auction

3. **plot** on the right, showing the number of remaining participants, and the time when each of the others left the auction ⇒ how to implement it?
   Suggestions:
   - save the number of remaining subjects at each point in time in a `user defined table` or in the `contracts table`
   - plot the content of this table using a `graph`, which is one of the `plot items`

A real effort task: pick the right color

The **task**:

1. correctly answer as many questions as possible
2. in a given time interval (60 seconds)

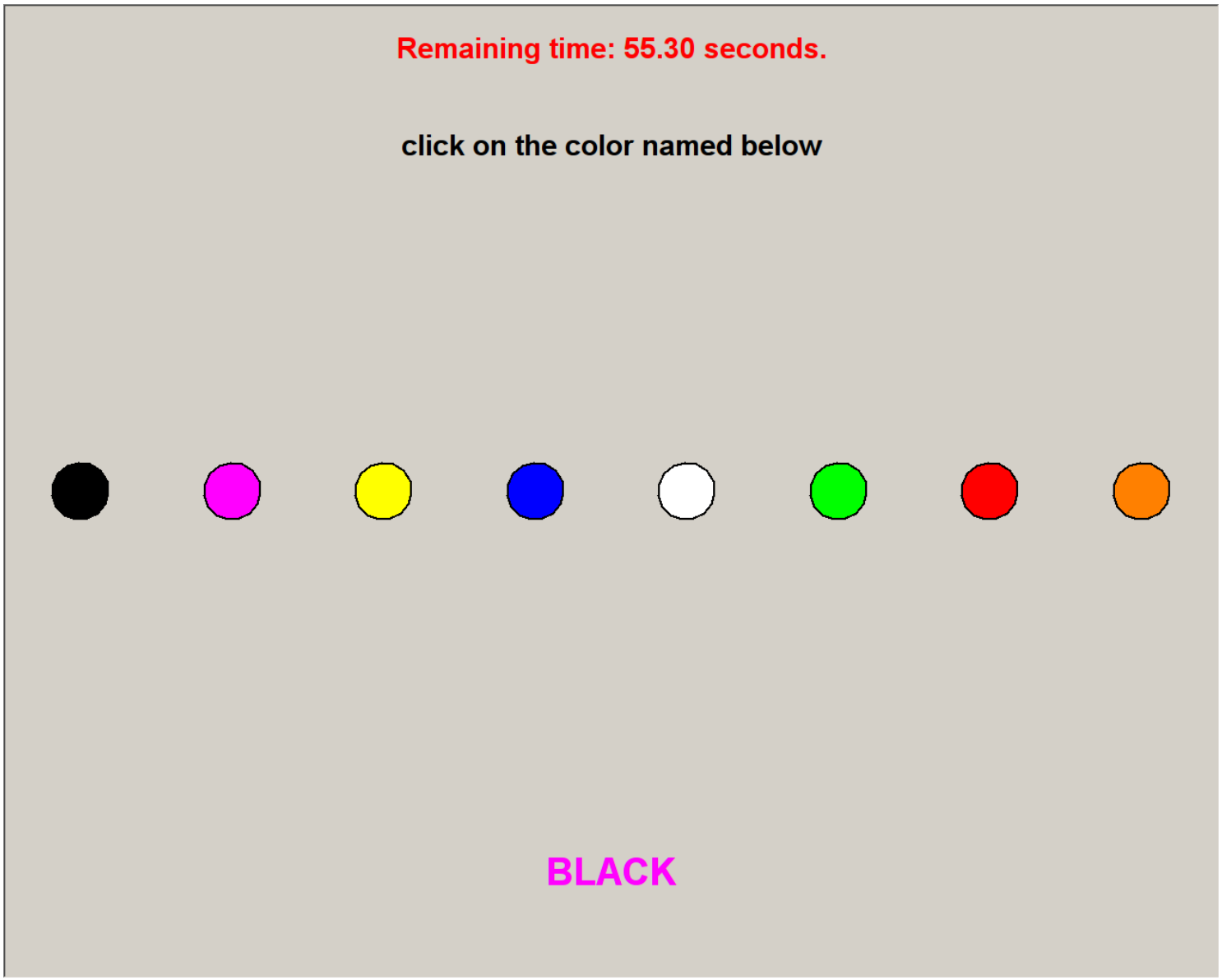
Two different **questions**:

1. click on the color of the word written on the screen
2. click on the color corresponding to the word written on the screen

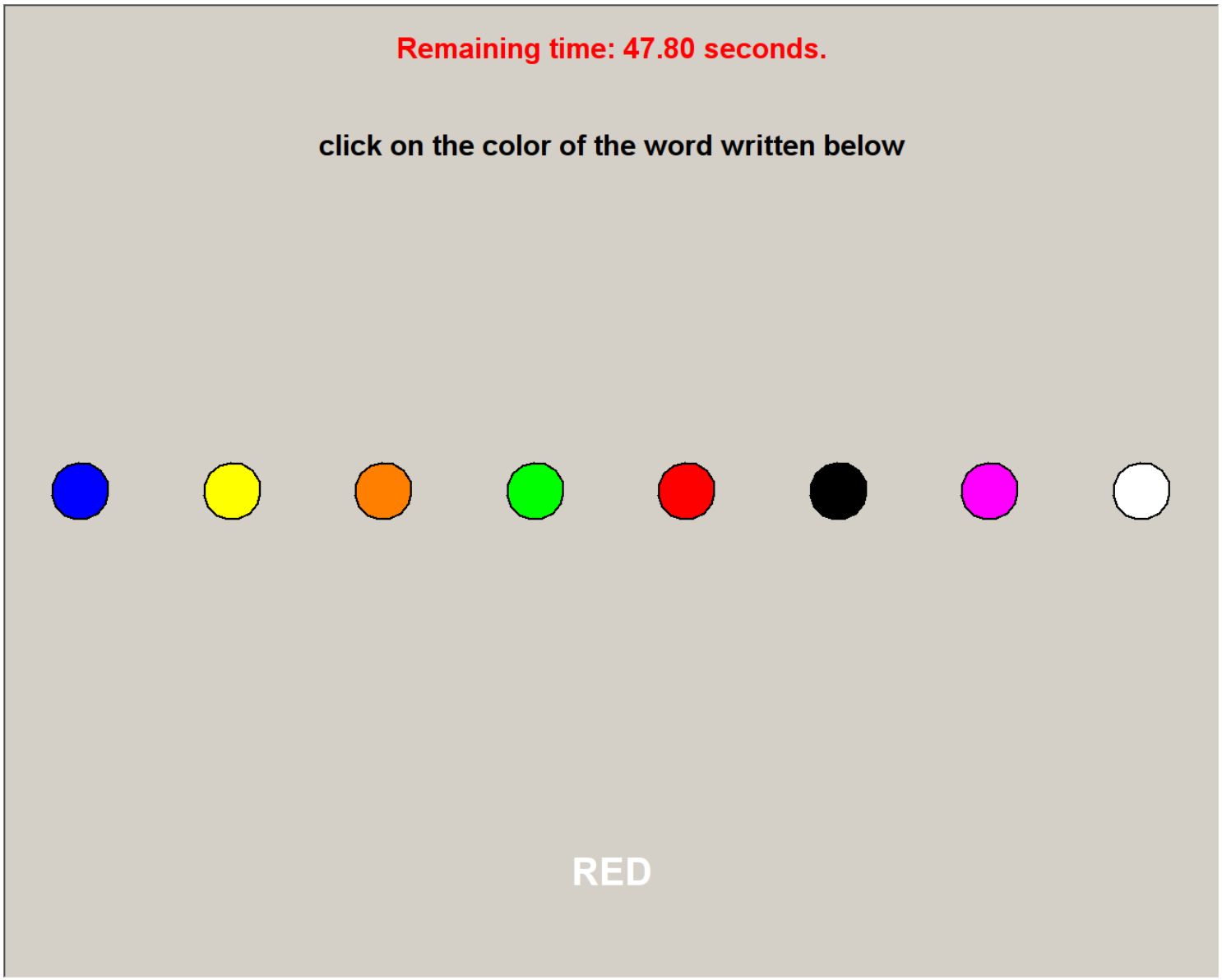
To **answer**, the subject must click on one of eight alternative colors, presented in random order on the screen.

Example: Colors.ztt

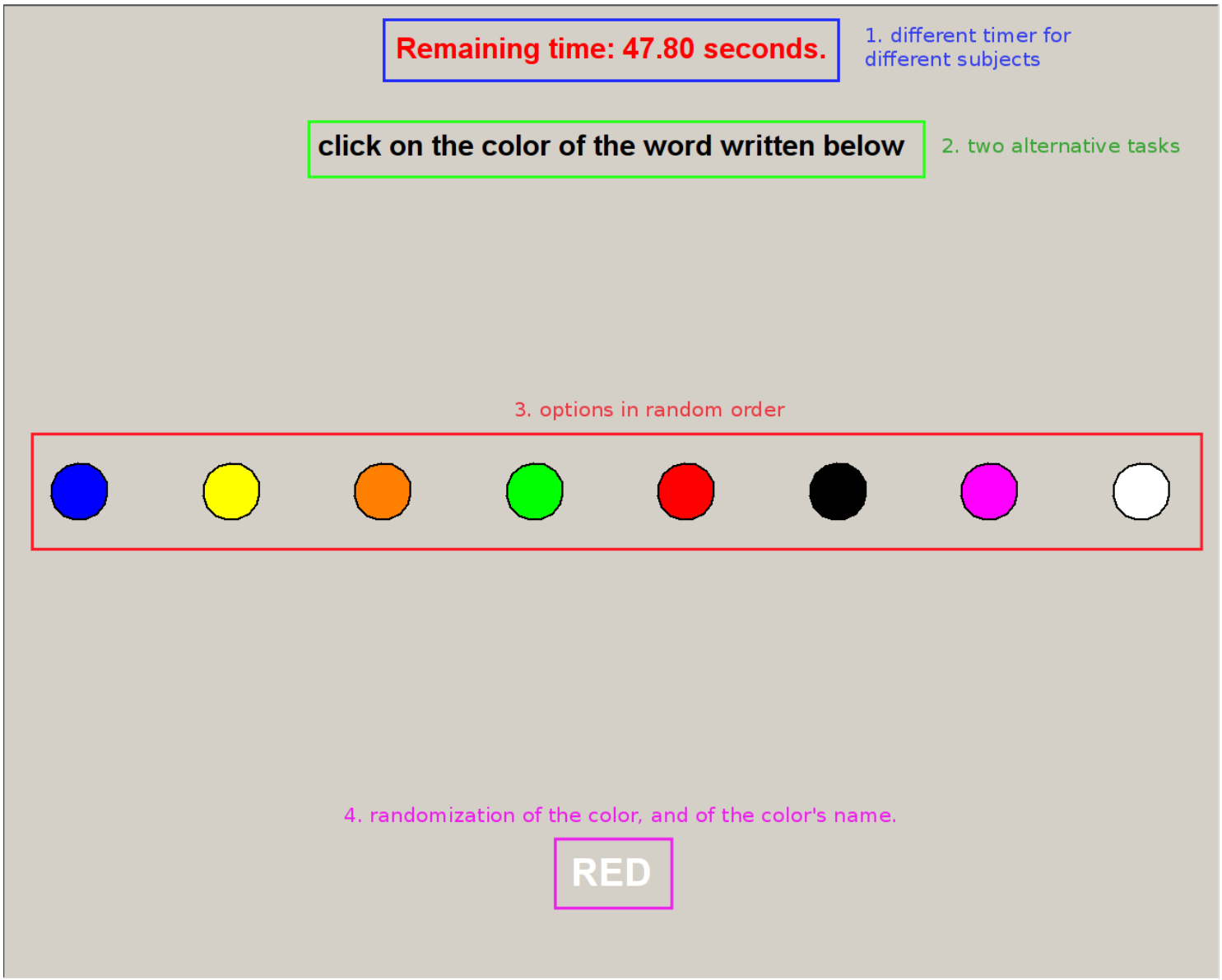
Screenshot - I



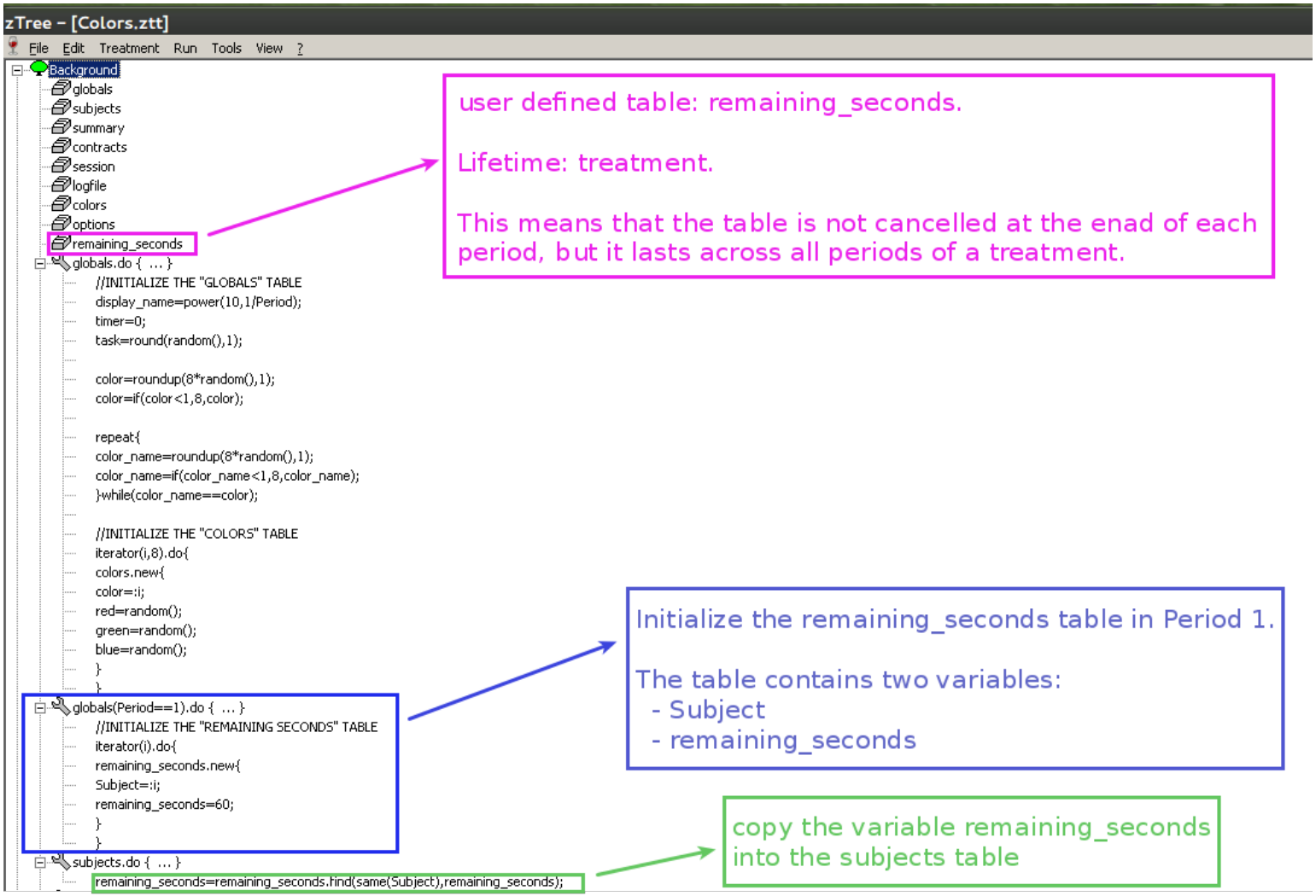
Screenshot - II



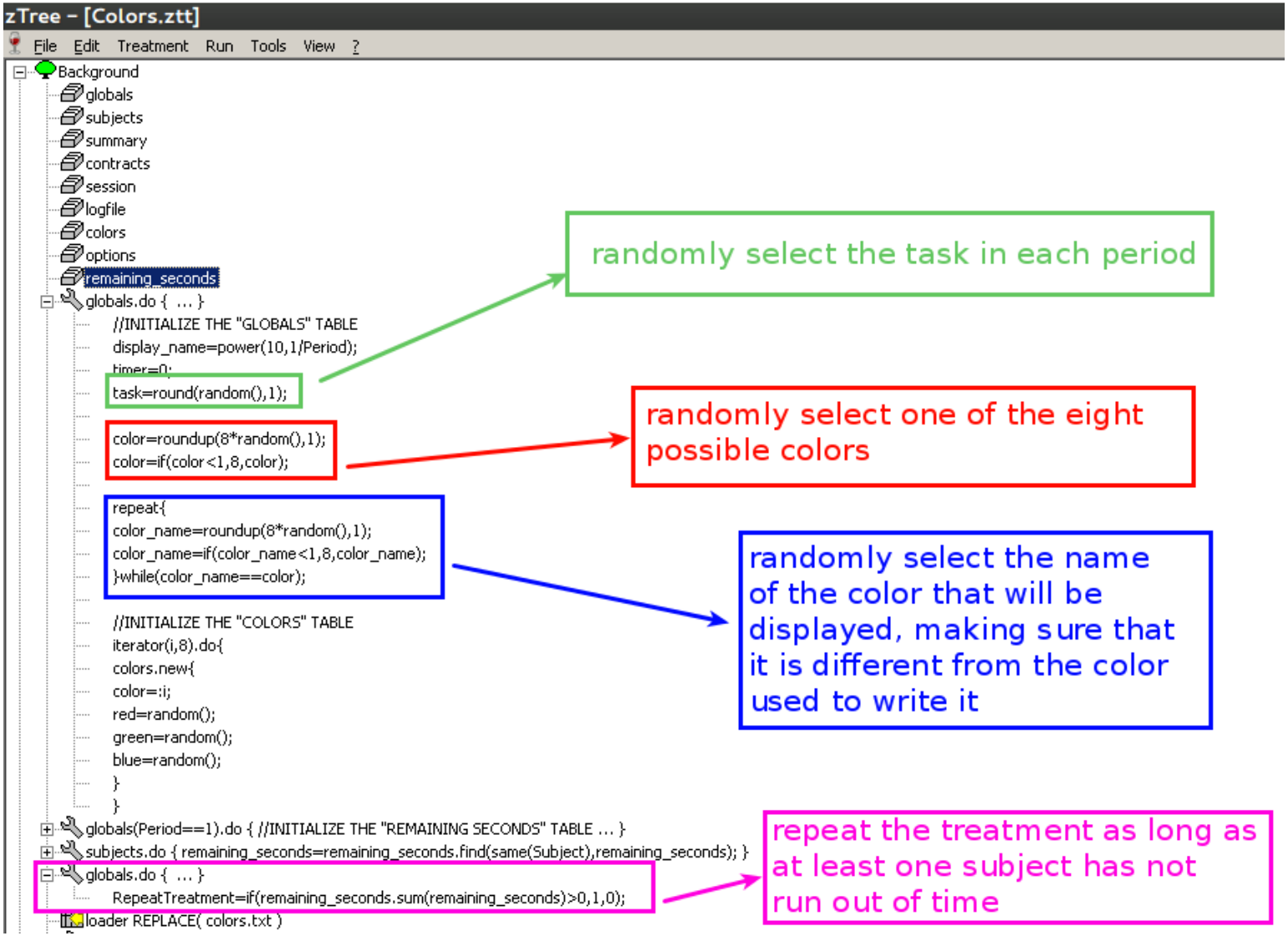
Screenshot - III



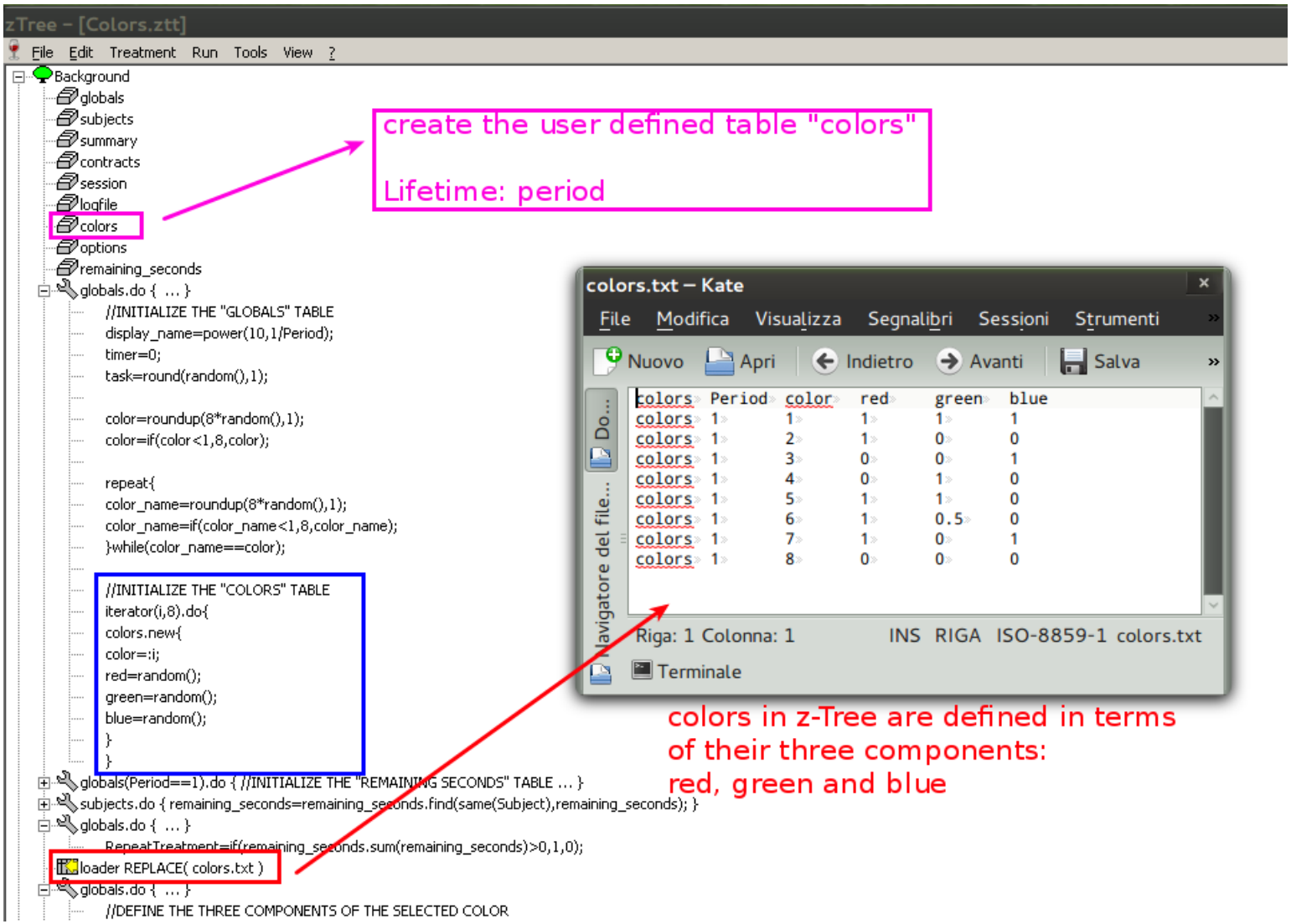
Remaining seconds



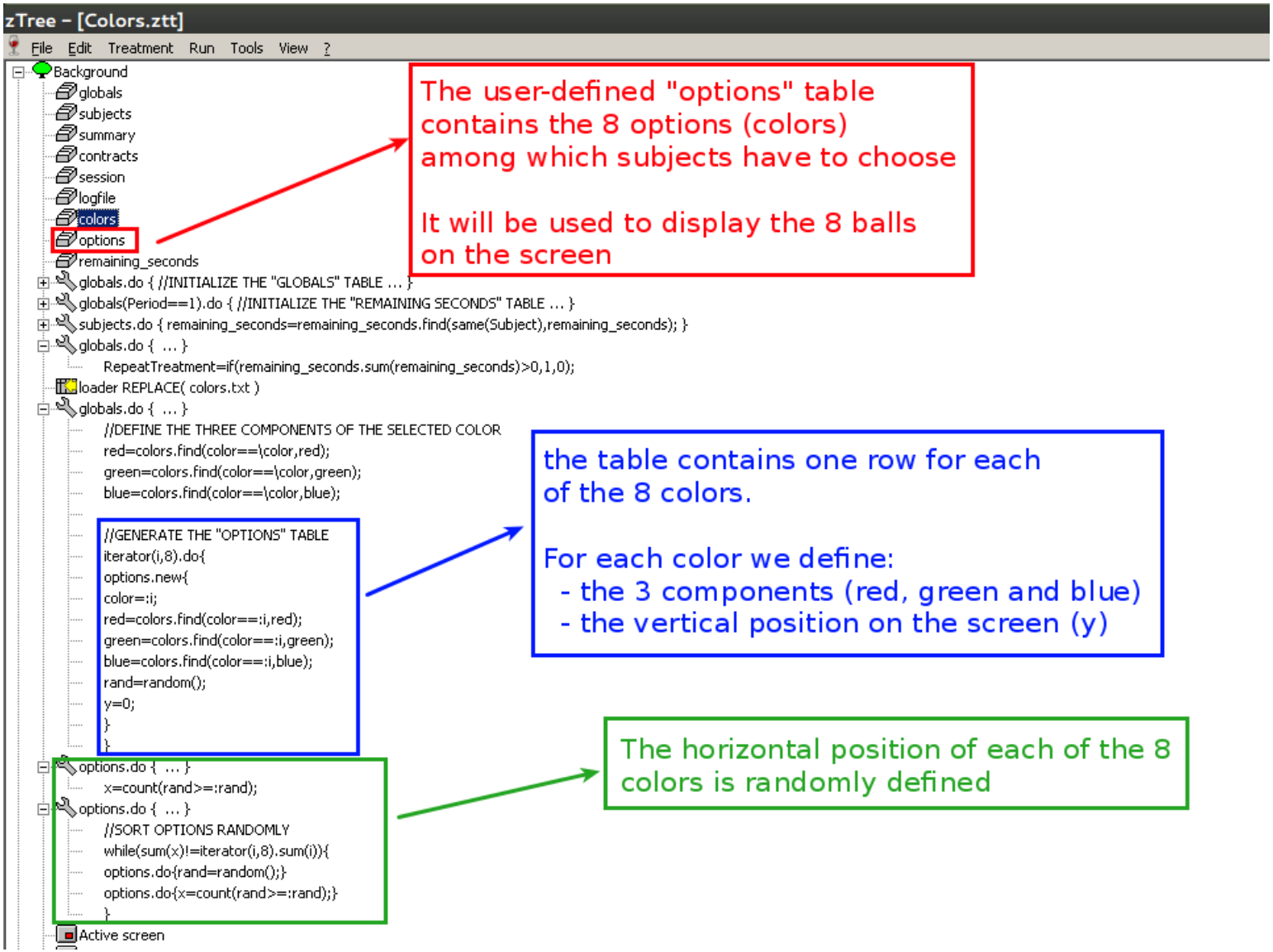
Randomize the task



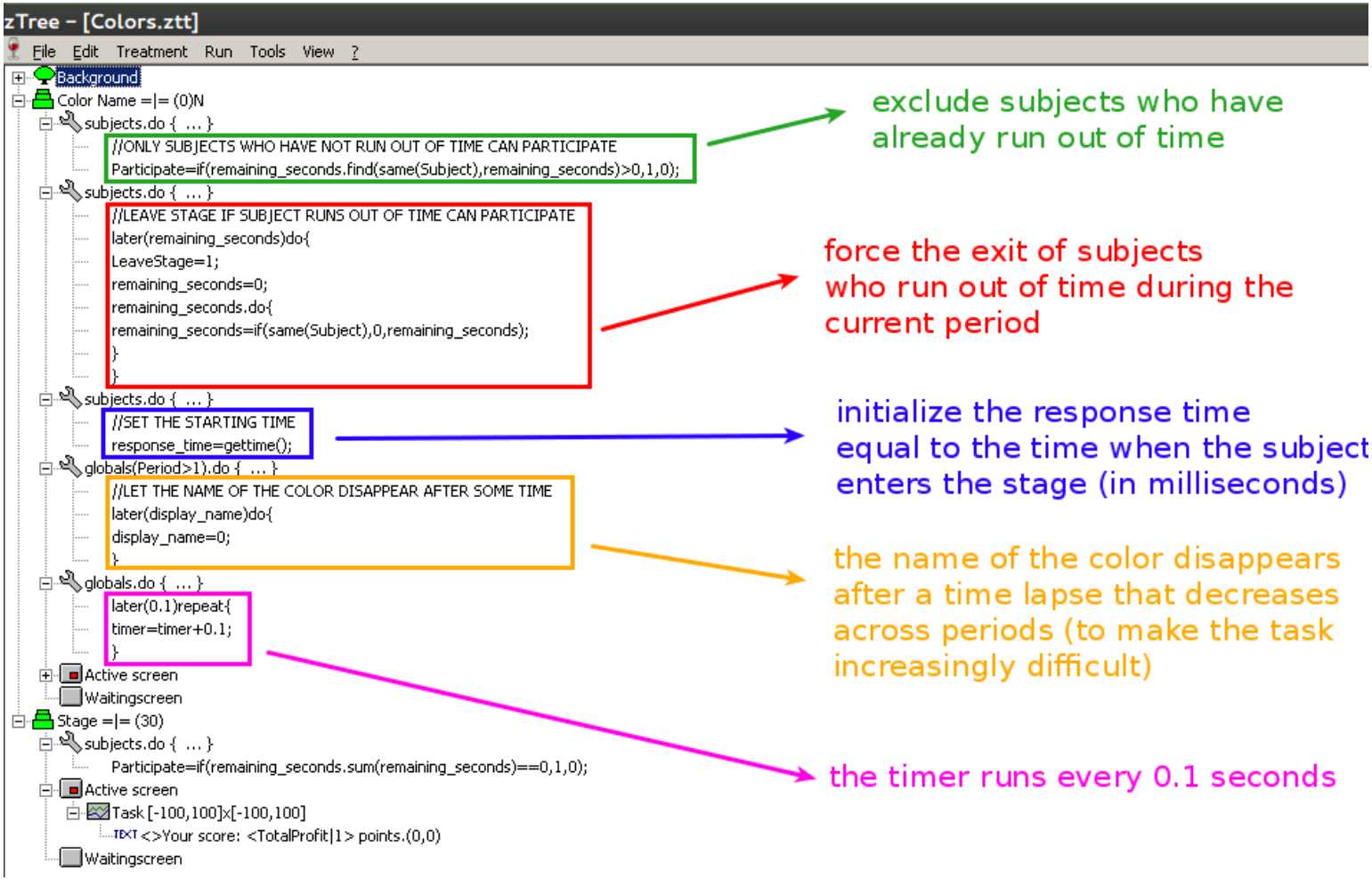
Define the colors



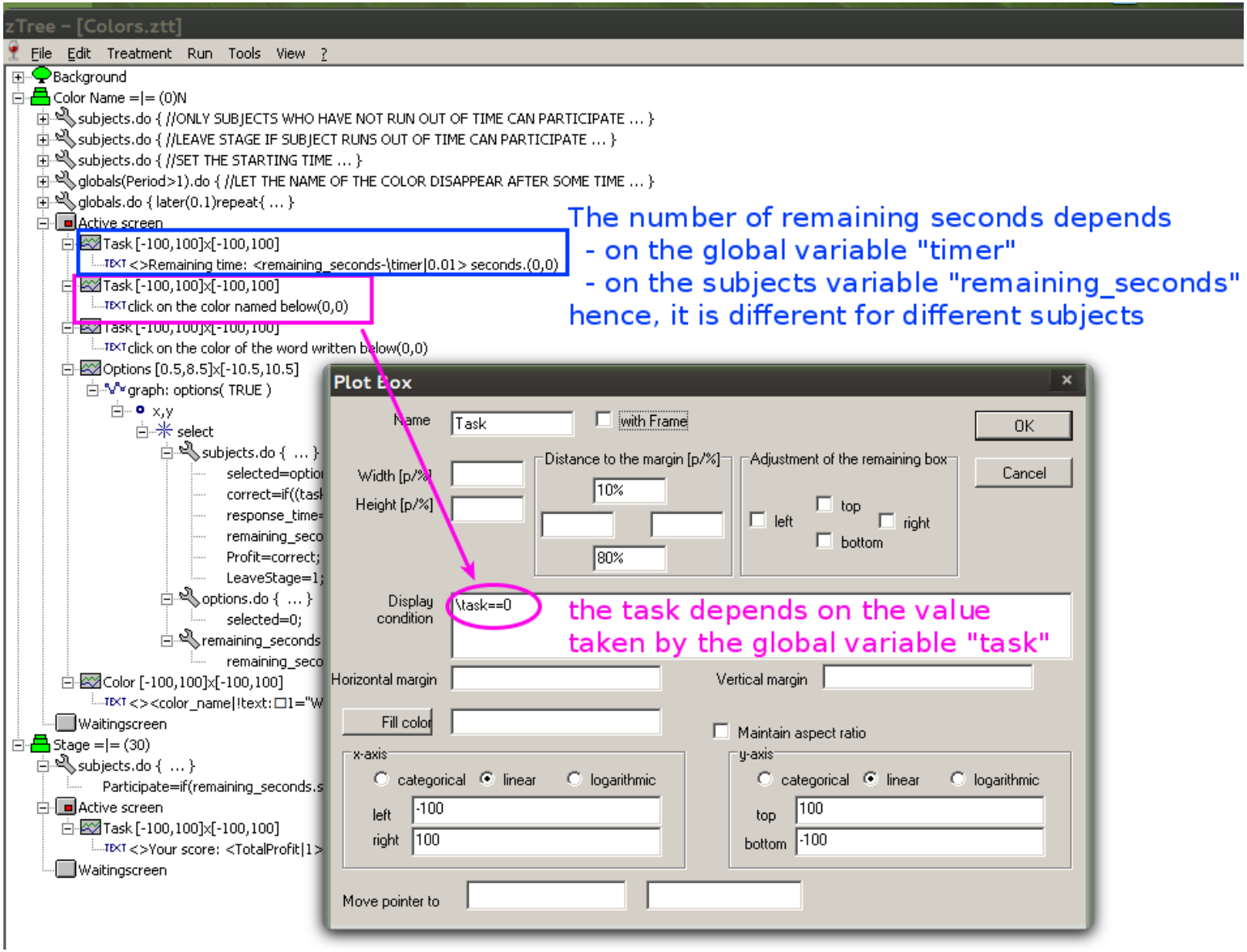
Define the options



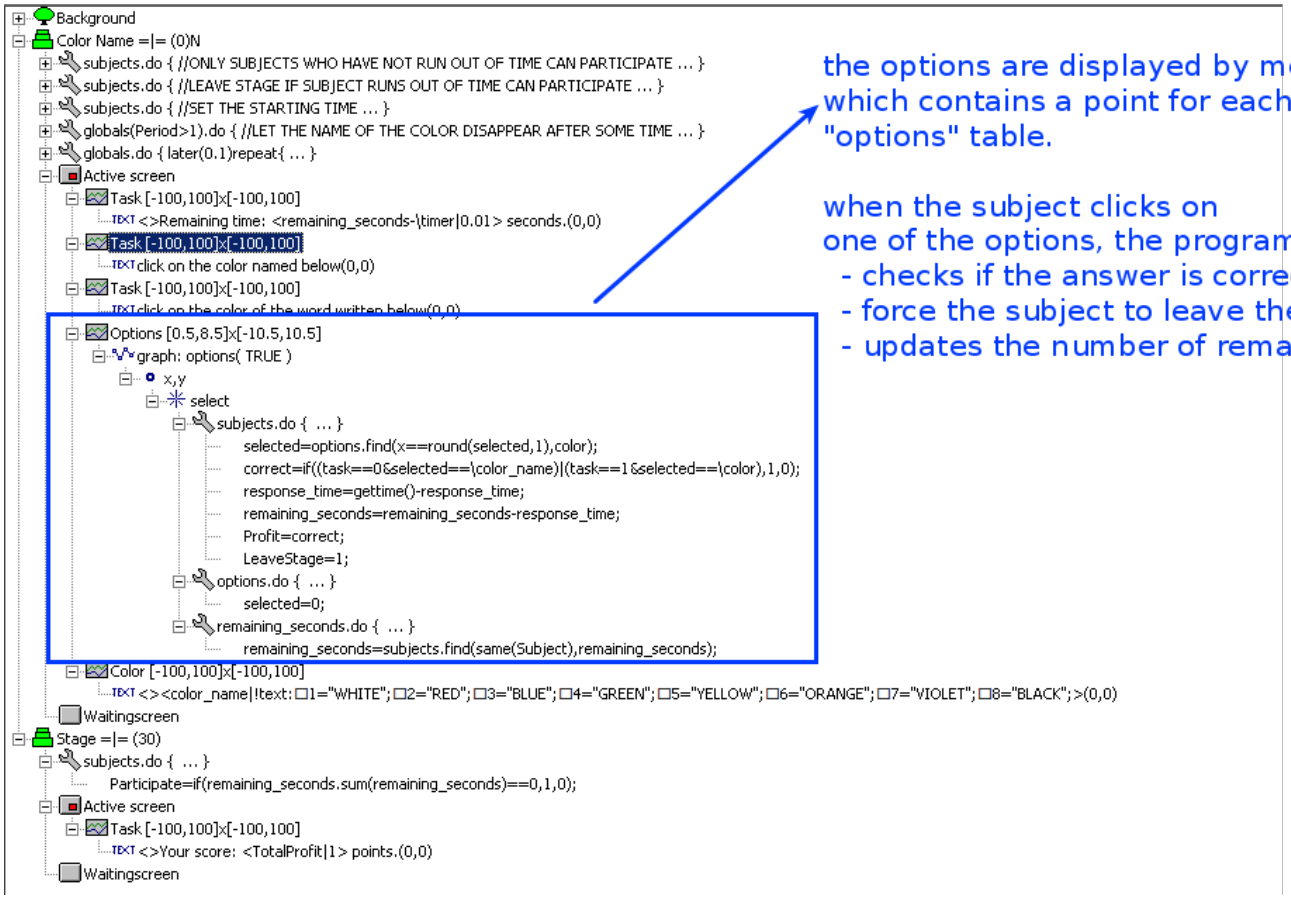
Programs


zTree – [Colors.ztt]
File Edit Treatment Run Tools View ?
Background
Color Name =|= (0)N
subjects.do { ... }
//ONLY SUBJECTS WHO HAVE NOT RUN OUT OF TIME CAN PARTICIPATE
Participate=if(remaining_seconds.find(same(Subject),remaining_seconds)>0,1,0);
subjects.do { ... }
//LEAVE STAGE IF SUBJECT RUNS OUT OF TIME CAN PARTICIPATE
later(remaining_seconds)do{
LeaveStage=1;
remaining_seconds=0;
remaining_seconds.do{
remaining_seconds=if(same(Subject),0,remaining_seconds);
}
}
subjects.do { ... }
//SET THE STARTING TIME
response_time=gettime();
globals(Period>1).do { ... }
//LET THE NAME OF THE COLOR DISAPPEAR AFTER SOME TIME
later(display_name)do{
display_name=0;
}
globals.do { ... }
later(0.1)repeat{
timer=timer+0.1;
}
Active screen
Waitingscreen
Stage =|= (30)
subjects.do { ... }
Participate=if(remaining_seconds.sum(remaining_seconds)==0,1,0);
Active screen
Task [-100,100]x[-100,100]
TEXT <>Your score: <TotalProfit|1> points.(0,0)
Waitingscreen
exclude subjects who have already run out of time
force the exit of subjects who run out of time during the current period
initialize the response time equal to the time when the subject enters the stage (in milliseconds)
the name of the color disappears after a time lapse that decreases across periods (to make the task increasingly difficult)
the timer runs every 0.1 seconds

Information to be displayed



Subjects' actions



the options are displayed by means of a graph which contains a point for each element of the "options" table.

when the subject clicks on one of the options, the program
- checks if the answer is correct
- force the subject to leave the stage
- updates the number of remaining seconds