

CHAPTER 3

A CRASH COURSE IN ZTREE

Lawrence Choo, PhD

LESSON PLAN

- **Introduction to z-tree**
 - Ztree architecture
 - How to *setup* your zleafs
- **Example I: The Public goods game**
 - Basic programming
 - Generating Input / Output variables
- **Example II: The Ultimatum Game**
 - Grouping mechanism (more programming)
 - Sequential decision making
 - Rich text format (rtf) coding
- **Class Exercise I: Second Price Auction**

LESSON PLAN

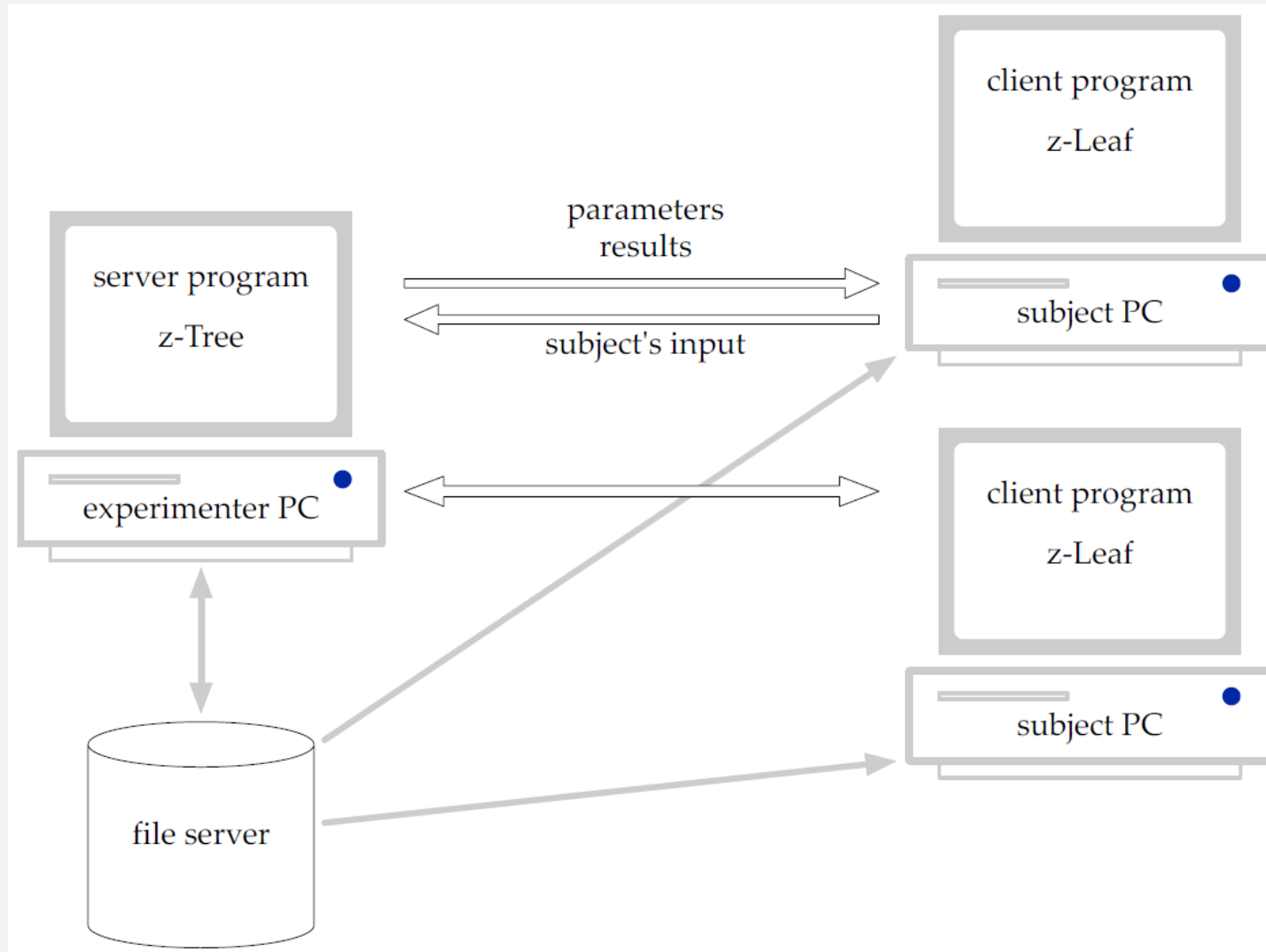
- **Creating multiple leafs on a screen**
- **Example III: 2x2 Normal form game**
 - Laying out Grid matrix
 - Random round payment
- **Example IV: Search Lottery**
 - Array programming and complex loops
 - Programming a Survey
- **Class Exercise II: Jackpot machine (A fair jackpot)**
- **Example V: Dutch Auction**
 - the “later” function
- **Class Exercise III: English Auction**

LESSON PLAN

- **Example VI: Continuous Double Auction**
 - Introduction to the Contract table
- **Example VII: Random Stopping Public Goods Game**
 - Creating *infinite* length games
- **Example VIII: Complex Move games**
 - Inserting Figures / Videos
 - Designing complex sequential move formats
- **Example IX: Chat Box**
- **Example X: 2-Dimesion Graphing**
 - Bars
 - Lines
- **Example XI: Graphing Pie Charts**
- **Exercise IV: Vernon Smith, Gerry Suchanek and Arlington Williams (1988)**
design with Graphed prices.

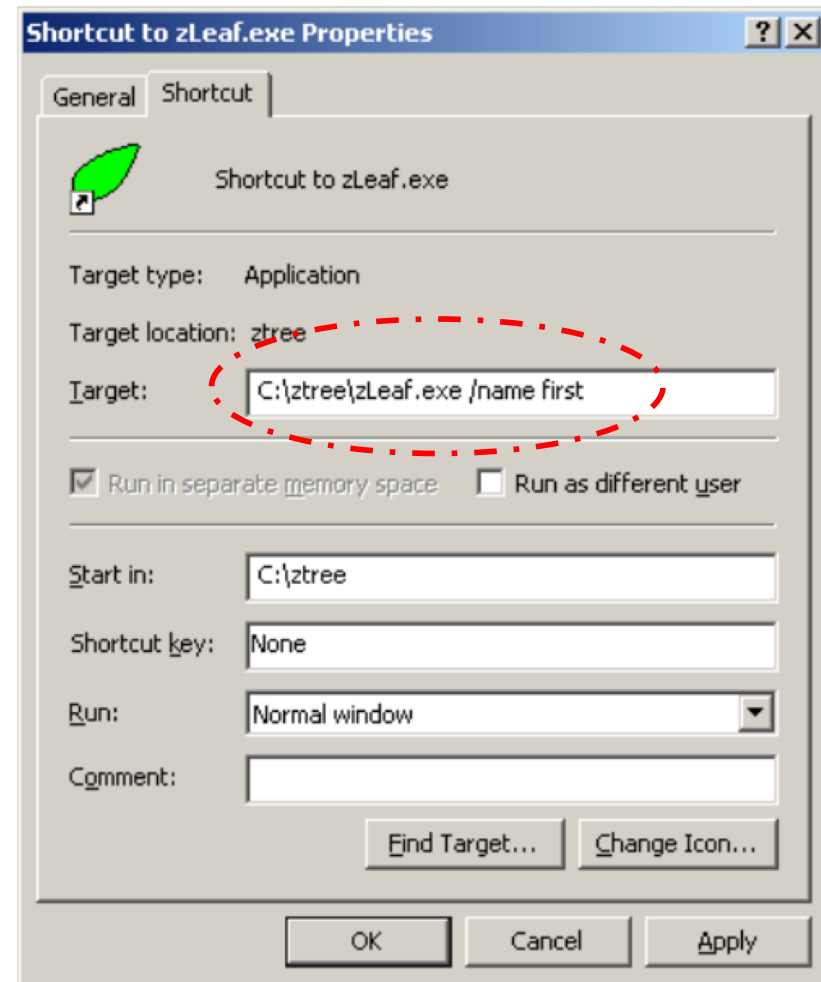
INSTALLING ZTREE
AND A BRIEF
INTRODUCTION

CLIENT-SERVER ARCHITECTURE



INSTALLING “LEAFS”

- Create multiple shortcuts for the zleaf
- Go into the properties of each shortcut leaf click on the properties dialog, click on the shortcut tab and append the
.exe /name Yourleafname
- Do this for every shortcut leafs giving a unique name



WHAT IS IN AN EXPERIMENT

Session

- The whole experiment. This might contain multiple treatments

Treatment

- A specific treatment setup.
- A treatment might contain multiple periods (i.e., rounds)

Period

- A specific period.
- This might contain multiple stages

Stage

- The lowest level, where subjects input / output variables are collected

HOW IS DATA STORED

Data is stored in *numerical* values in “pre-specified *Tables*”.

Name	written	Reset Freq.	Description
global	Every period	Every Period	Input / Output variables that affect ALL subjects
subjects	Every period	Every Period	Input / Output variables that affect a specific subject
contract	Every period	Every Period	Input / Output variables that affect a specific subject within a period
summary	Every period	Every Treatment	Input / Output variables that affect a specific subject over a treatment
session	Every treatment	Every Session	Input / Output variables that affect a specific subject over a Session

ztree also allows for user created tables in addition to the above

EXAMPLE I

**THE PUBIC
GOODS GAME**

EXAMPLE: PUBLIC GOODS GAME

$$u_i = E - x_i + \frac{\sum_i x_i}{N} \times M$$

globals table

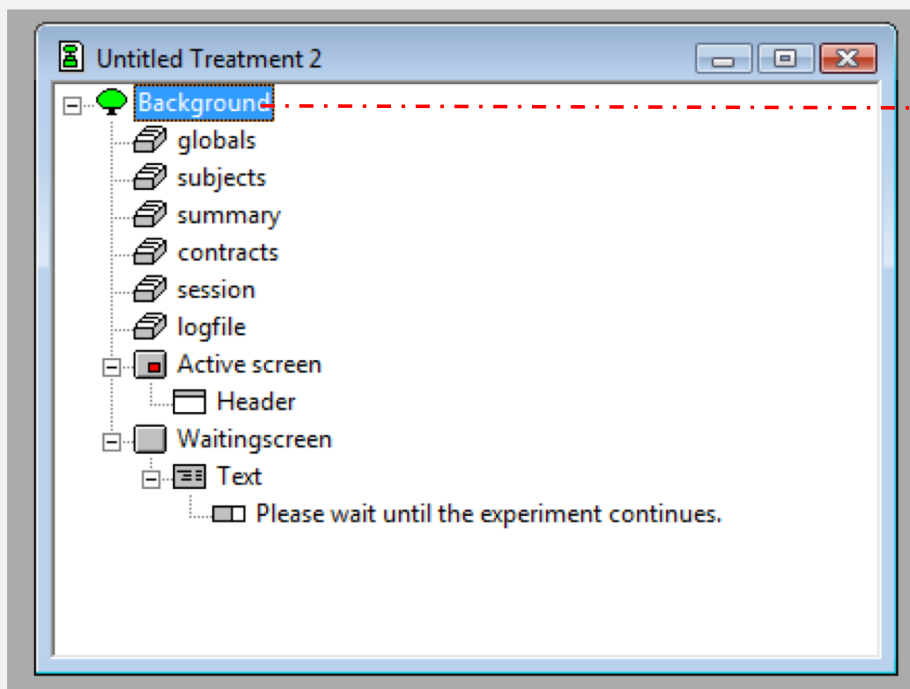
Period	NumPeriods	RepeatTreatment	M	E	N
2	5	0	1.2	10	4

subjects table

Period	Subject	Group	Profit	TotalProfit	Participate	x	sumx	u
2	4	1	12.65	18.65	1	2	15.50	12.65
2	5	2	11	15.45	1	2	10	11
2	6	2	12	20.10	1	1	10	12
2	7	2	9	10.00	1	4	10	9
2	8	2	16	22.12	1	3	10	16

SET BACKGROUND

- 4 Subjects
- All subjects in *same* group
- $t = 2$ periods



General Parameters

Number of subjects	4	OK
Number of groups	1	Cancel
# practice periods	0	
# paying periods	2	
Exch. rate [Fr./ECU]	1	
Lump sum payment [ECU]	0	
Show up fee [Fr.]	0	

Bankruptcy rules...

Start time of the period

Compatibility

first boxes on top

Options

without Autoscope

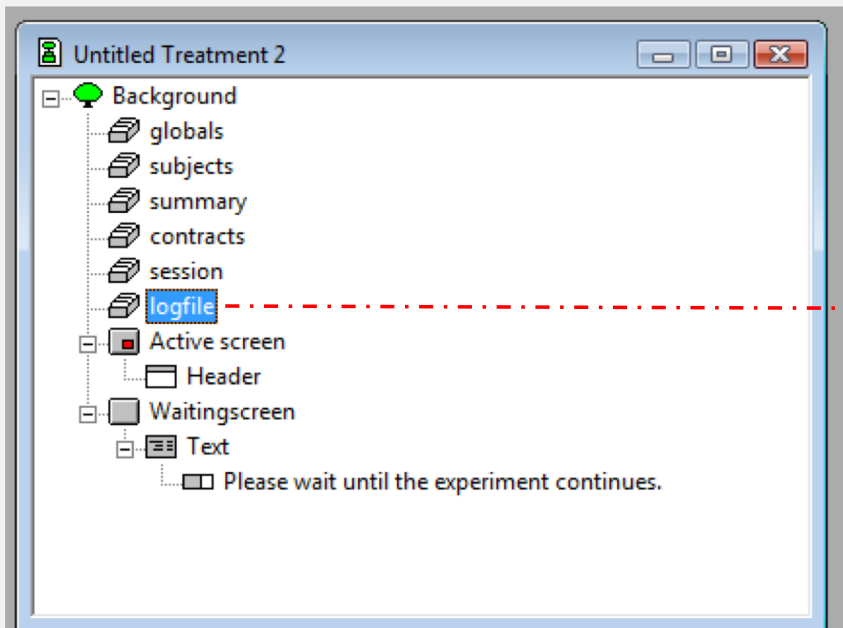
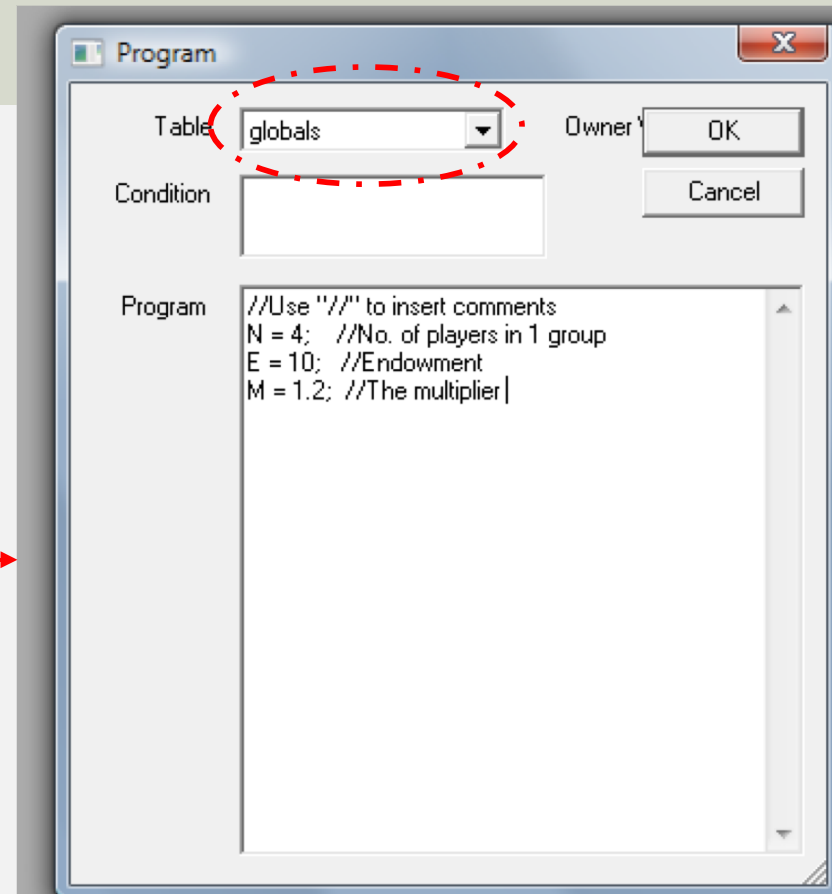
DEFINE INITIAL VALUES (globals table)

> Treatment > New program

```
N = 4; //no. of players in a group
```

```
E = 10; // endowment
```

```
M = 1.2; // multiplier
```



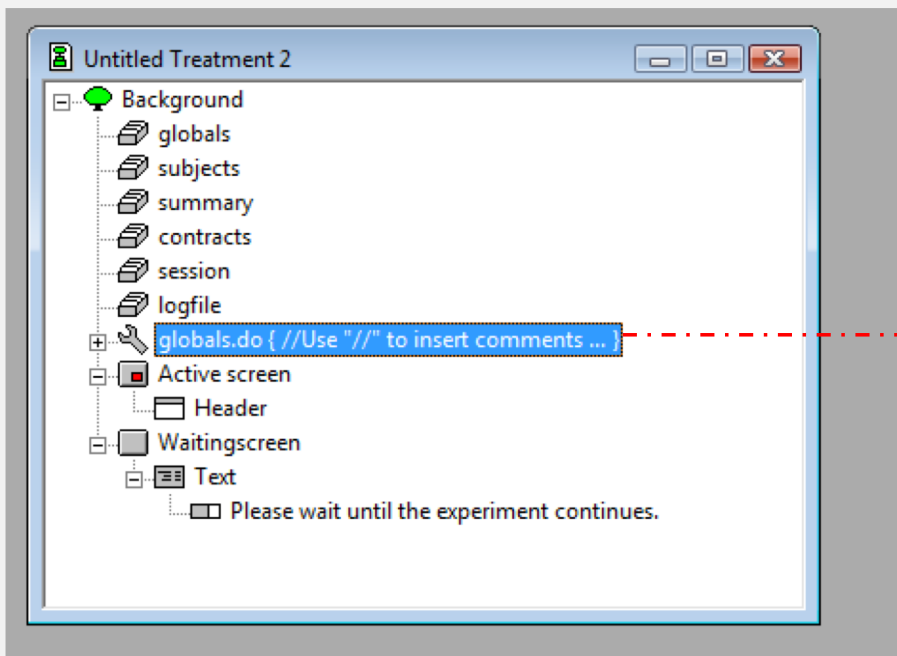
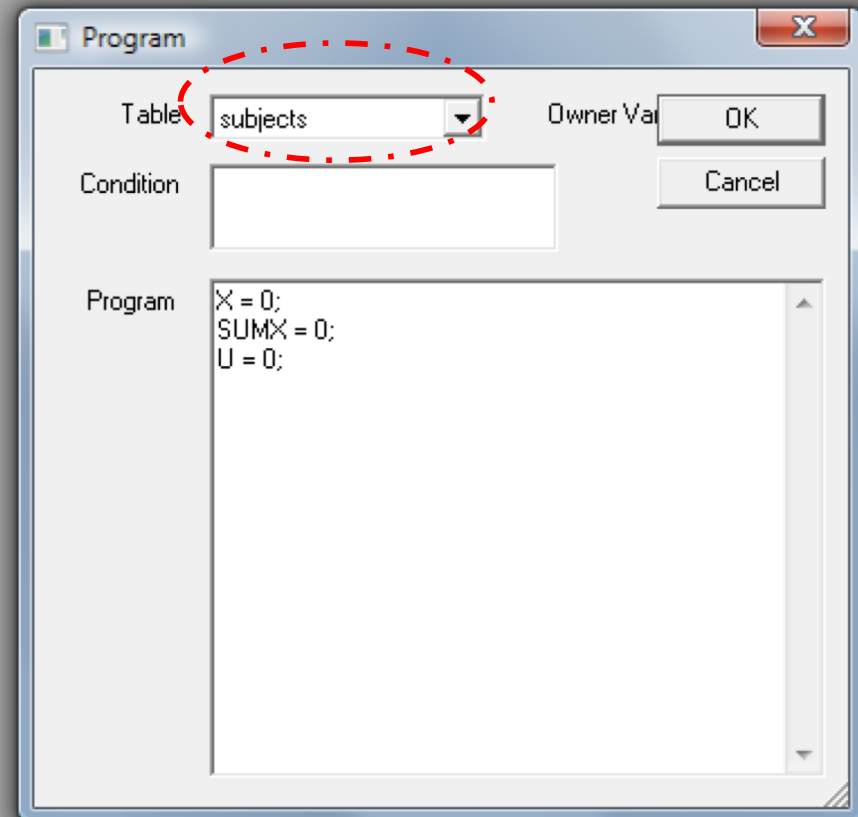
DEFINE INITIAL VALUES (subjects table)

> Treatment > New program

```
X = 0; //define variable
```

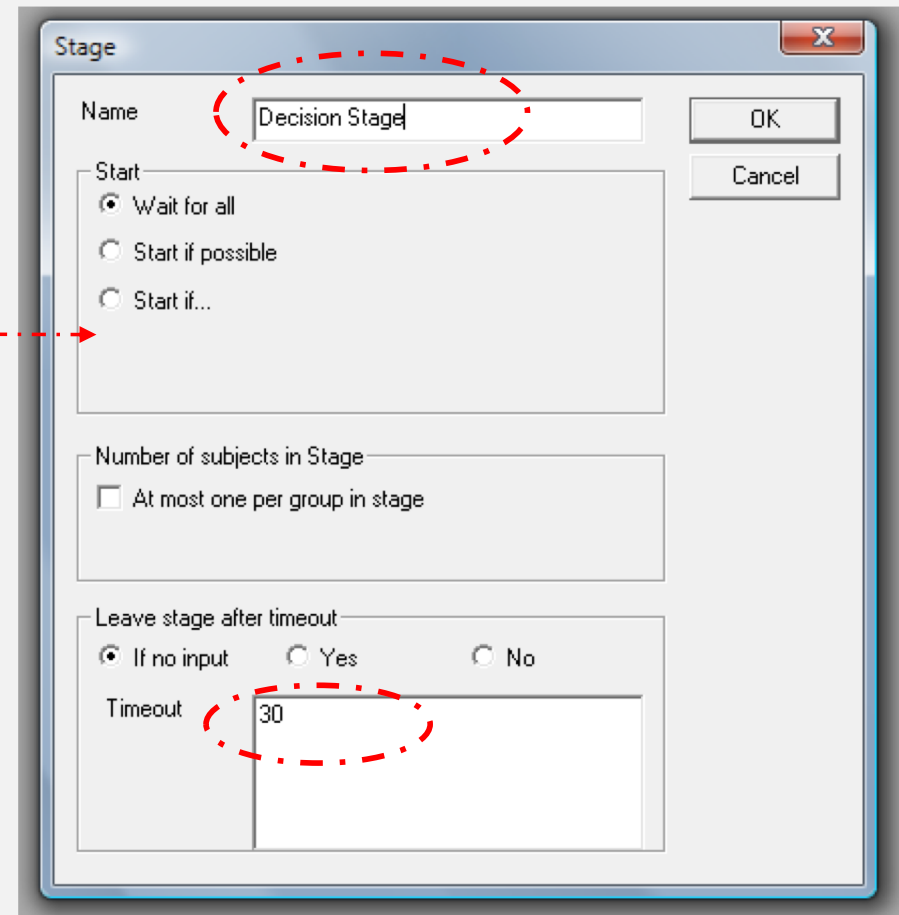
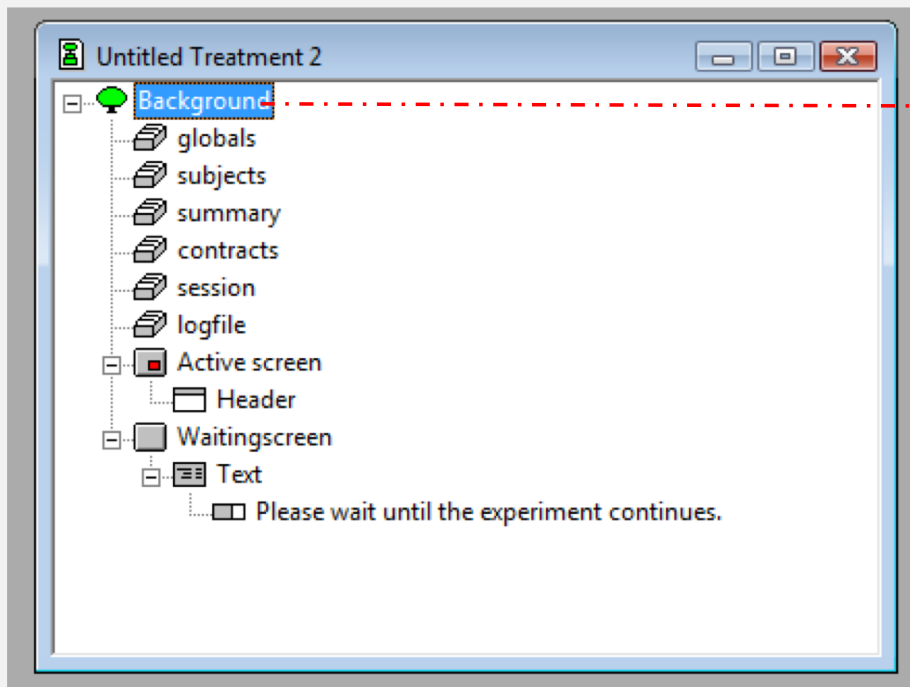
```
SUMX = 0;
```

```
U = 0;
```



CREATE NEW STAGES

- Select the most recent stage
 - > Treatment > New stage
- You can add as many stages as necessary
- Create “**Decision stage**”
- Create “**Results stage**”



DECISION STAGE (create new box)

- **A box contains**
 - Output variables that subjects see
 - Input variables that subjects enter

Active screen > Treatment > New Box > Standard Box

Standard Box

Name: Standard with Frame

Width [p/%)

Height [p/%)

Distance to the margin [p/%)

10%

10%

10%

Adjustment to the remaining box

left top right

bottom

Display condition

Buttons

Position

Arrangement

In rows

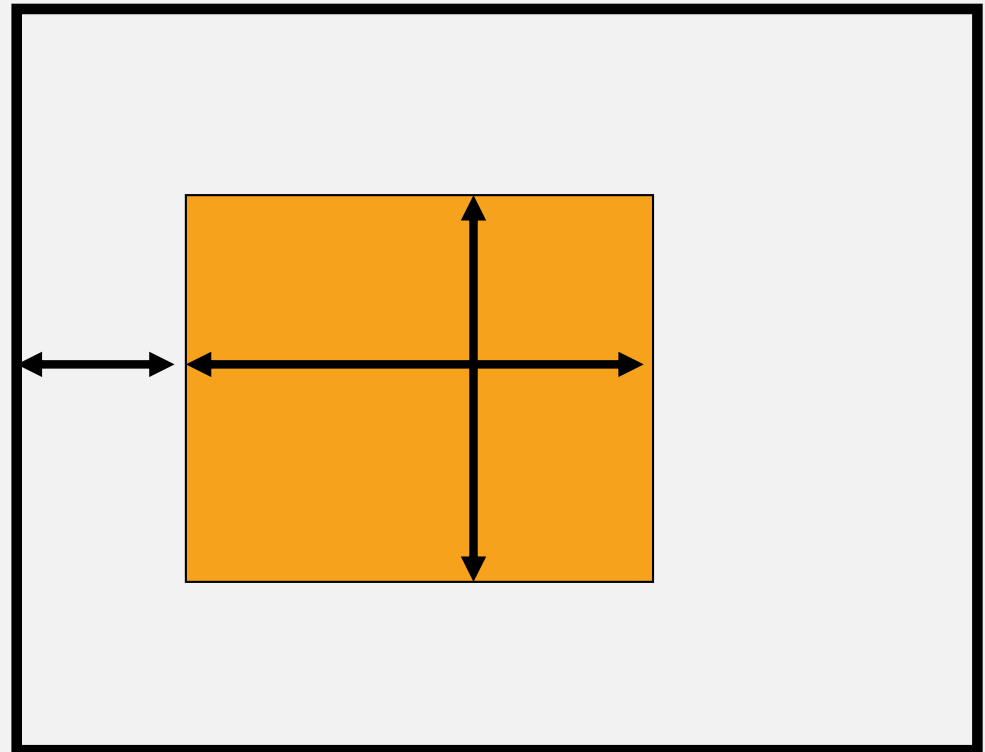
In columns

OK

Cancel

GENERAL (box layout)

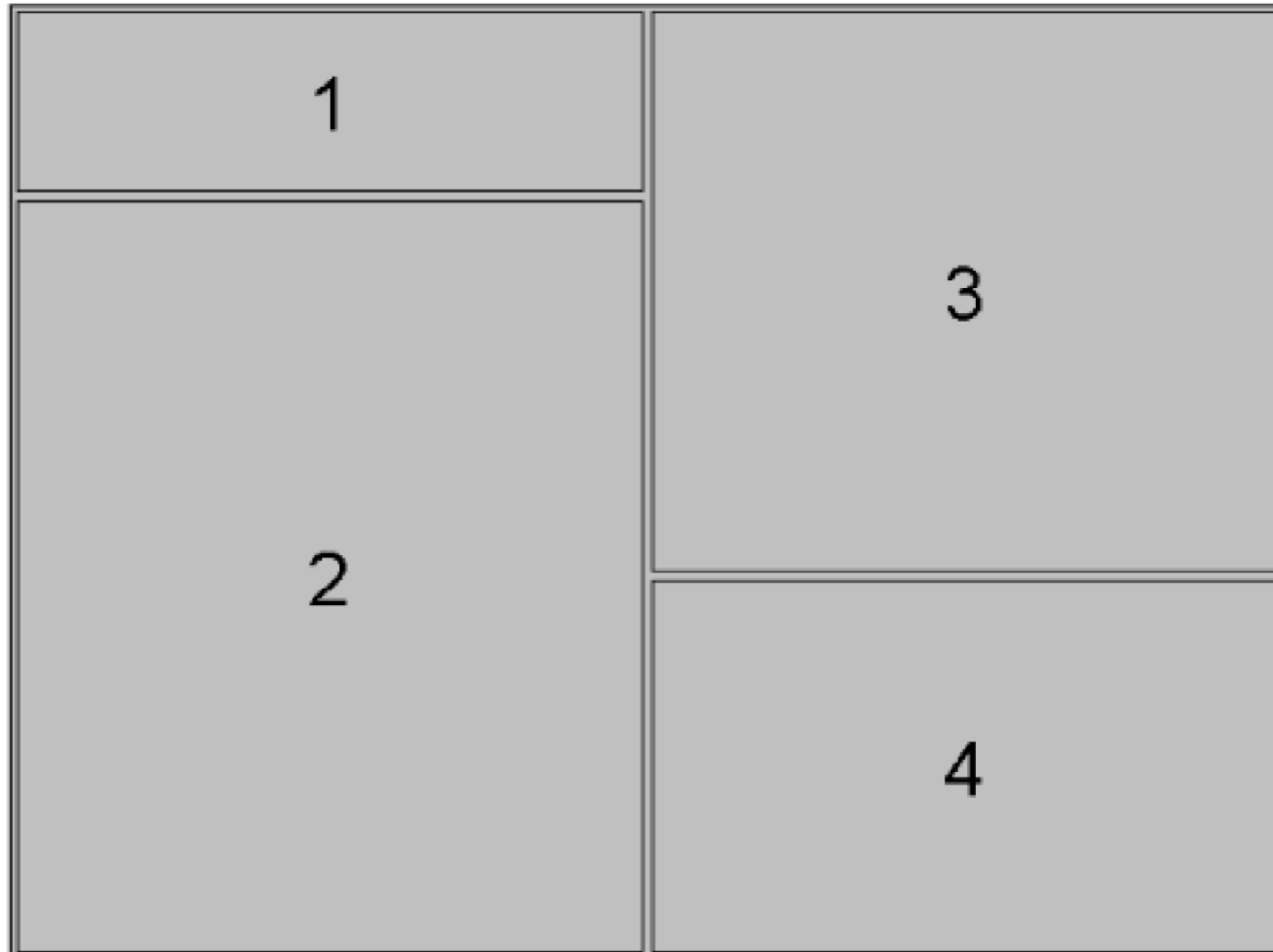
- **Box can be position one after another**
- **Types of boxes:**
 1. **Standard box**
 2. **Grid box**
 3. Header box
 4. Help box
 5. History box
 6. Container box
 7. Calculator button box



GENERAL

(box layout – Container Box)

- Active screen
- Container 12
 - Standard 1
 - Standard 2
- Container 34
 - Standard 3
 - Standard 4



GENERAL

(box layout – Grid Box)

- Grid
- label
 - input var: IN[b]
 - input var: IN[d]
 - shows a: OUT(a)
 - second label
 - input var: IN[c]
 - OK

label	17
<input type="text"/>	second label
X <input type="radio"/> <input type="radio"/> <input type="radio"/> Y	<input type="text"/>
<input type="button" value="OK"/>	

column by column

label	<input type="text"/>
X <input type="radio"/> <input type="radio"/> <input type="radio"/> Y	17
second label	<input type="text"/>
<input type="button" value="OK"/>	

row by row

DECISION STAGE (Putting items in Box)

- **Creating a Output variable**

Standard box > Treatment > New item

Item

Label: Your Endowment:

Variable: E

Layout: .01|

Input

OK

Cancel

Text that subjects see
“Your Endowment”

The output variable “E”

How the output variable is presented:

“I” : no decimal places

“.I” : 1 decimal place

“.0I” : 2 decimal places

DECISION STAGE (Putting items in Box)

- **Creating a input variable**

Standard box > Treatment > New item

Item

Label: How much do you want to contribute?

Variable: X

Layout: .01

Minimum: 0

Maximum: E

Default:

Event time:

Input

Show value (value of variable or default)

Empty allowed

OK

Cancel

The input variable “X”

How subjects input the data

.01 : 2 decimal places, last digit is multiple of 1

10: no decimal, variable is multiple of 10

5: no decimal, variable is multiple of 5

GENERAL

(input / output formats)

Layout	Input variable	Output variable
2	<input type="text" value="6"/>	6
<code>!text: 7 = "seven"; 8 = "eight"; 9 = "nine";</code>	<input type="text" value="seven"/>	seven
<code>!radio: 1 = "86.8"; 24 = "102.8";</code>	<input checked="" type="radio"/> 86.8 <input type="radio"/> 102.8	<input checked="" type="radio"/> 86.8 <input type="radio"/> 102.8
<code>!radioline: 0="zero";5="five"; 6;</code>	zero <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> five	zero <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> five
<code>!radiosequence: 7="seven";8="eight";9="nine";</code>	<input type="radio"/> seven <input type="radio"/> eight <input type="radio"/> nine	<input type="radio"/> seven <input checked="" type="radio"/> eight <input type="radio"/> nine
<code>!slider: 0 = "A"; 100= "B"; 101;</code>	A <input type="range" value="50"/> B	A <input type="range" value="50"/> B
<code>!scrollbar: 0="L";100= "R";101;</code>	L <input type="range" value="50"/> R	L <input type="range" value="50"/> R
<code>!checkbox:1="check me";</code>	<input checked="" type="checkbox"/> check me	<input checked="" type="checkbox"/> check me
<code>!button: 1 = "accept"; 0 = "reject";</code>	<input type="button" value="accept"/> <input type="button" value="reject"/>	accept
<code>!string</code>	<input type="text"/>	
20		Hello World

DECISION STAGE (Putting items in Box)

- **Creating a button**

Standard box > Treatment > New Button

Button

Name

No record created or selected

Clear entry after OK

Leave Stage

Yes

No

Normal (i.e. stage is not left after click if stage is left after timeout and button is contained in contract creation or selection box)

Color

Automatic

Gray

Red

Event time

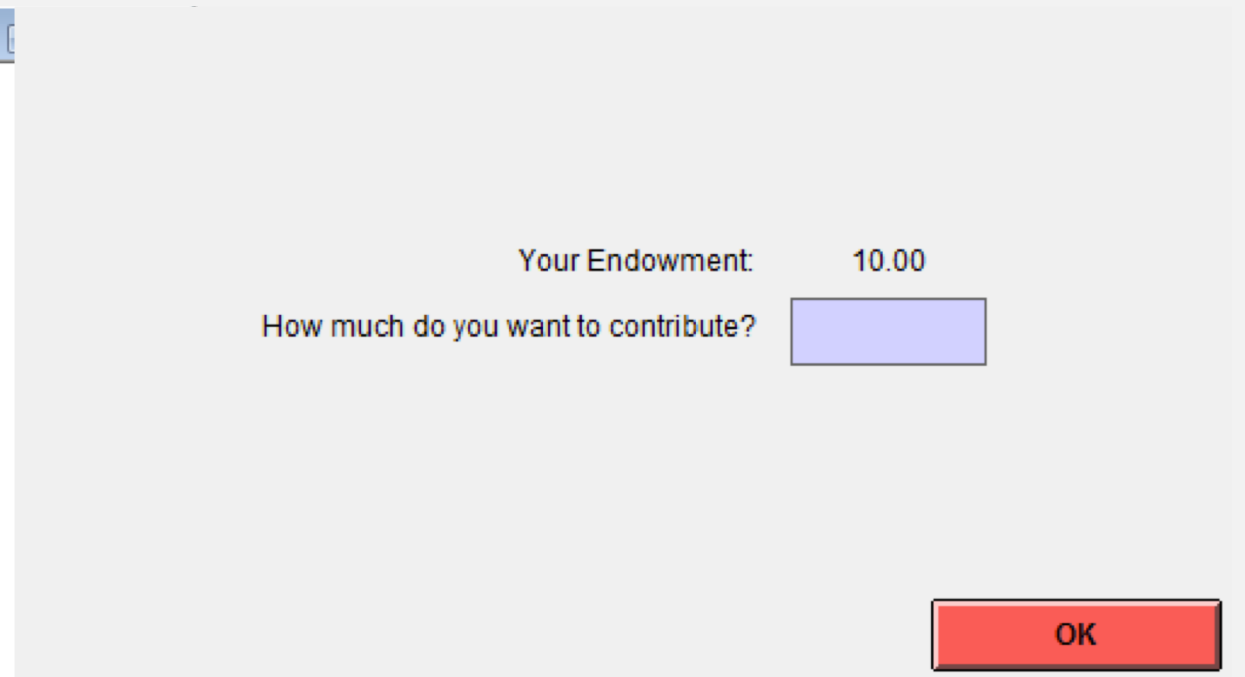
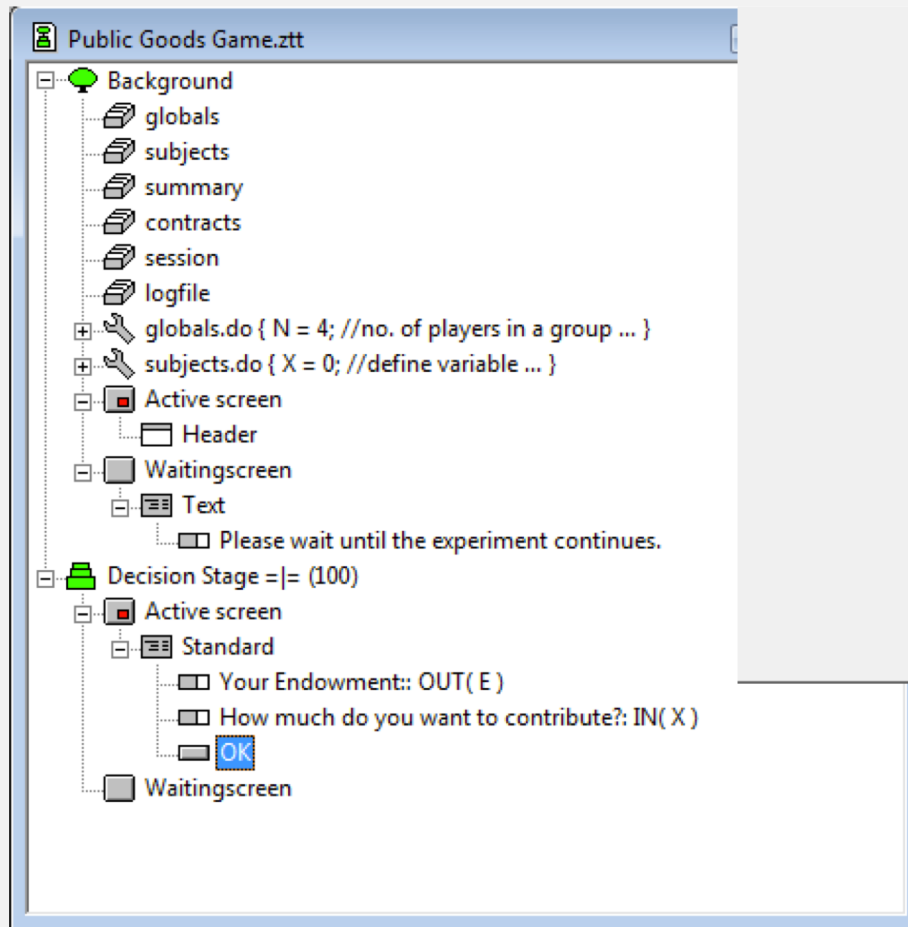
A button tells ztree to collect the data and let the subject leave the stage.

- Easy to forget
- Without a button, subjects get "stuck" on the screen

DECISION STAGE

(trial the Decision Stage)

run > start treatment

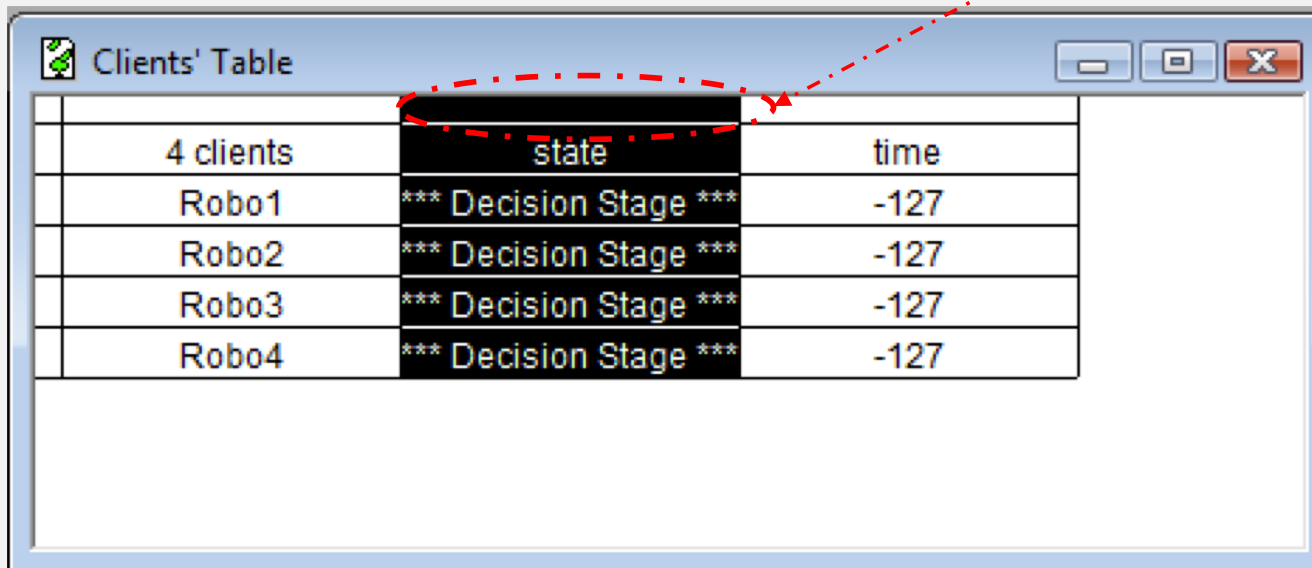


GENERAL

(How to "force subjects to leave a stage"?)

- **run > client's table**
- **Double click**
- **run > leave stage**

Double click here



	state	time
4 clients		
Robo1	*** Decision Stage ***	-127
Robo2	*** Decision Stage ***	-127
Robo3	*** Decision Stage ***	-127
Robo4	*** Decision Stage ***	-127

RESULTS STAGE

(collect the data from other subjects)

Results Stage > Treatment > New program
(subjects table)

- Find the contributions of all other players within the same group

```
SUMX = sum(same(Group), X);
```

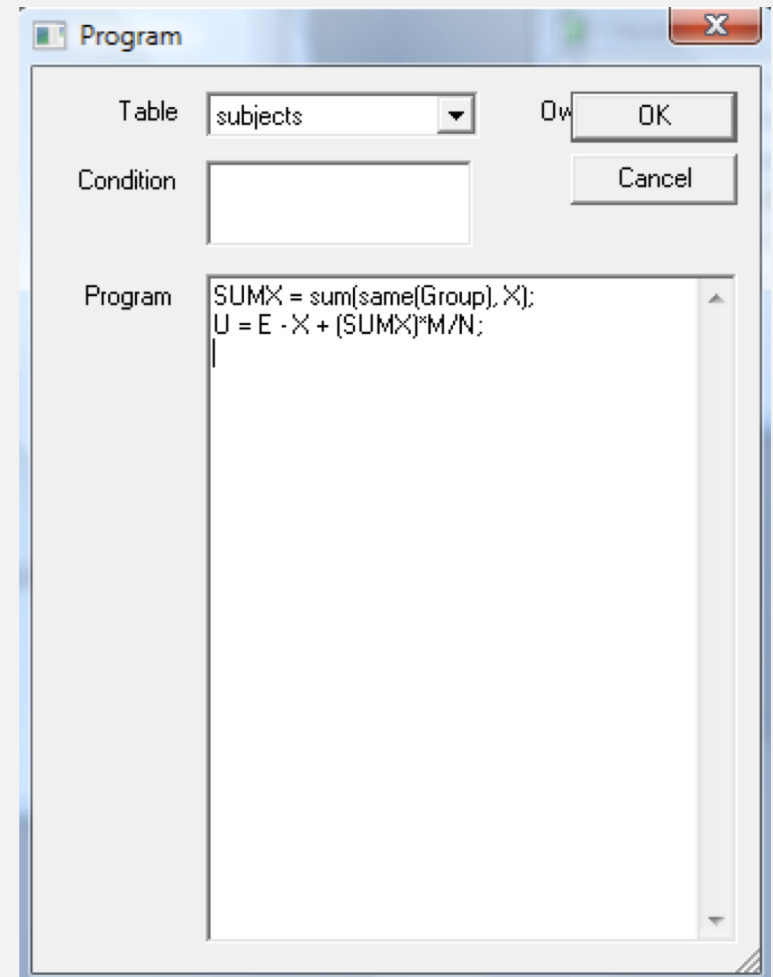
- Compute payoff

```
U = E - X + (SUMX) * M / N;
```

Alternative code

```
SUMX = average(same(Group), X);
```

```
U = E - X + SUMX * M;
```



GENERAL

(how programs run)

g	M	x
5	0	0
12	0	0
7	0	0

```
M = 20;  
x = M - g;
```

g	M	x
5	20	15
12	0	0
7	0	0

g	M	x
5	20	15
12	20	8
7	0	0

g	M	x
5	20	15
12	20	8
7	20	13

GENERAL

(how programs run)

g	M	x	y
5	0	0	0
12	0	0	0
7	0	0	0

```
M = 20;  
x = M - g;  
y = sum(x);
```

↓

g	M	x	y
5	20	15	15
12	0	0	0
7	0	0	0

→

g	M	x	y
5	20	15	15
12	20	8	23
7	0	0	0

→

g	M	x	y
5	20	15	0
12	20	8	23
7	20	13	36

GENERAL

(how programs run)

```
M = 20;  
x = M - g;
```

g	M	x	y
5	20	15	0
12	0	0	0
7	0	0	0

g	M	x	y
5	20	15	0
12	20	8	0
7	0	0	0

g	M	x	y
5	20	15	0
12	20	8	0
7	20	13	0

```
New program  
y = sum(x);
```

g	M	x	y
5	20	15	36
12	20	8	0
7	20	13	0

g	M	x	y
5	20	15	36
12	20	8	36
7	20	13	0

g	M	x	y
5	20	15	36
12	20	8	36
7	20	13	36

GENERAL

(Some use scope operators)

```
Y = sum ( [condition] , variable );
```

```
Y = average ( [condition] , variable );
```

```
Y = minimum ( [condition] , variable );
```

```
Y = maximum ( [condition] , variable );
```

```
Y = median ( [condition] , variable );
```

```
Y = find ( [condition] , variable );
```

```
Y = count ( [condition] );
```

RESULTS STAGE

(Create New Box and Output Variabels)

- **Active screen > Treatment > New Box > Standard Box**
- **Standard box > Treatment > New item**
 - Label: Your Contribution | Variable: X
- **Standard box > Treatment > New item**
 - Label: Total contribution in this period | Variable: SUMX
- **Standard box > Treatment > New item**
 - Label: Your Payoffs | Variable: U
- **Standard box > Treatment > New Button**

You contributed:	3.00
Total Contributions this period:	12.00
Your Payoffs this period:	10.60

OK

EXAMPLE II

**THE ULTIMATUM
GAME**

DESIGN OBJECTIVES

Design Objectives

- 4 Subjects, 2 groups
- 2 period
- At each period, random allocation to Proposer or Responder
- Random grouping
- Pot = 10

DEFINE INITIAL VALUES

> Treatment > New program (global table)

```
POT = 10; //Amount of money to be shared
```

> Treatment > New program (subjects table)

```
TYPE = 0; //1=Proposer, 2=Responder
```

```
OFFER = 0; //Proposer's offer
```

```
RESPOND = 0; //Responder's respond 1=Accept 2=Reject
```

```
U = 0; //Payoffs
```


MATCHING (BRUTE FORCE)

>Treatment > Parameter table

The screenshot displays a software window titled "Untitled Treatment 4:2" containing a parameter table. The table has columns for subjects S1, S2, S3, and S4, and rows for parameters 1 and 2. The cell for parameter 1 under subject S2 is highlighted in black. A "Specific Parameter" dialog box is open, showing details for Subject S 2, Period 1, Group 1, and Program TYPE=1 }.

	S 1	S 2	S 3	S 4
1	1	1	1	1
2	1	1	1	1

Specific Parameter

Subject S 2 Period 1

Group

Name

Program TYPE=1 }

OK Cancel

MATCHING

(a better approach to random grouping)

```
> Treatment > New program (subjects table)
```

```
G = 2; //Number of subjects in a group
```

```
r = random(); //Generate a random number between 0 and 1
```

Create a new program after random variable is created.

```
> Treatment > New program (subjects table)
```

```
RANK = count (r >= :r);
```

```
Group = roundup ( RANK / G, 1);
```

```
> Treatment > New program (subjects table)
```

```
TYPE = count (same (Group) & r >= :r);
```

But there might be ties !!!!

GENERAL (scope operator)

```
Y = count (g>=: g);
```

Subject	g	y
1	5	3
2	12	0
3	7	0

Subject	g	y
1	5	3
2	12	1
3	7	0

Subject	g	y
1	5	3
2	12	1
3	7	2

MATCHING

(a better approach to random grouping)

> Treatment > New program (subjects table)

```
G = 2; //Number of subjects in a group
```

> Treatment > New program (subjects table)

```
Sum_No_Tie = sum(Subject);
```

```
repeat {
```

```
subjects.do {
```

```
r = random();
```

```
}
```

```
subjects.do { RANK = count ( r >= :r); } } while (Sum_No_Tie  
!= sum( RANK ));
```

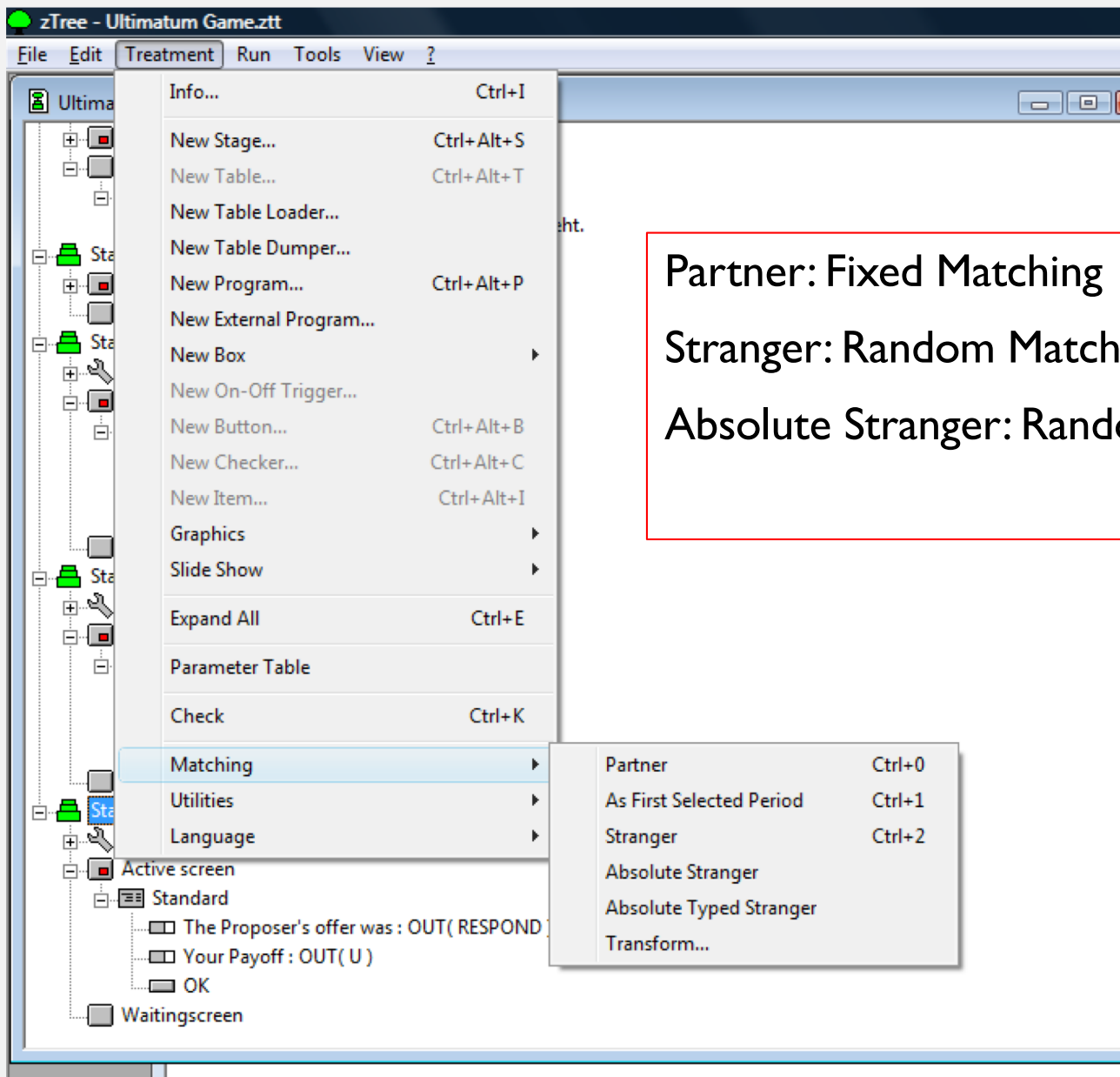
> Treatment > New program (subjects table)

```
Group = roundup ( RANK / G, 1);
```

```
TYPE = count(same(Group) & r>= :r);
```

GENERAL

(ztree build in grouping)



Partner: Fixed Matching

Stranger: Random Matching

Absolute Stranger: Random Matching + New Stranger

GENERAL (useful functions)

```
Y = if ( k < 5 | k >= 10, 1, 10 );
```

```
Y = abs ( c -d );
```

```
Y = round ( a, 0.5 );
```

```
Y = roundup ( a, 0.5 );
```

```
Y = exp ( random() );
```

```
Y = sqrt ( b ^ 2 );
```

```
Y = max ( ln ( x ), log ( y ) );
```

SEQUENTIAL PLAY

STAGE 1

- Subjects learn about their types (Simultaneous)

STAGE 2

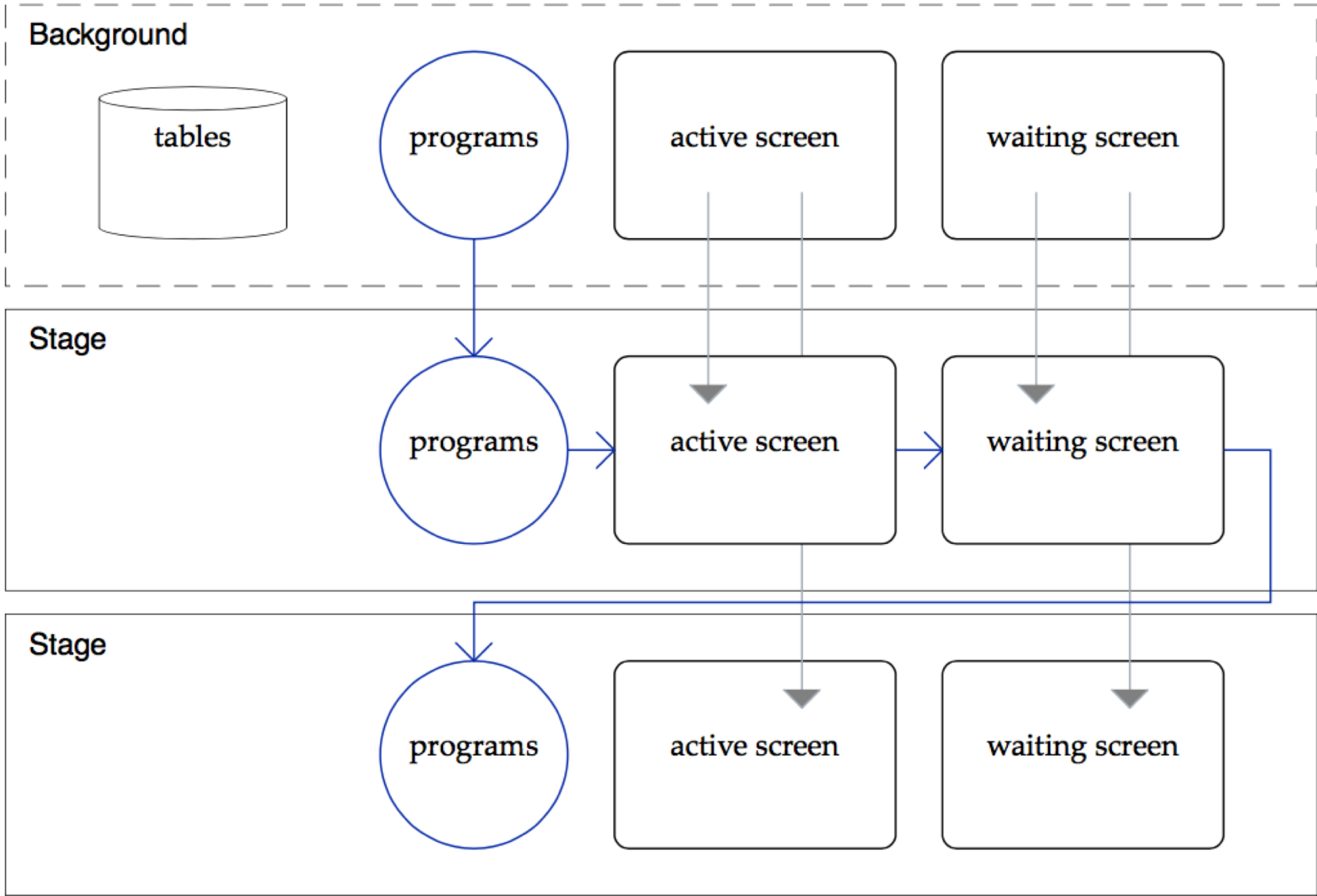
- Proposer: Makes an offer
- Responder: -

STAGE 3

- Proposer: -
- Responder: See Proposer's offer and chooses to accept or reject

STAGE 4

- Payoffs are realised (Simultaneous)



STAGE I

(Subjects learn about their types)

Solution I: Create two “standard box” and input items

- Label: You are the Proposer in this period
- Label: You are the Responder in this period

in *each* of the boxes. Use display condition to determine who sees what

Only subjects with
TYPE=I will see the
items in this box

Standard Box

Name with Frame

Width [p/%]

Height [p/%]

Distance to the margin [p/%]

Adjustment to the remaining box

left top right

bottom

Display condition

Buttons

Position

Arrangement

In rows

In columns

OK

Cancel

STAGE I

(Subjects learn about their types)

Solution II: create a generic box and in the label section of new item, include

```
<>{\rtf\fs20\qc You have been assigned to Group <Group|1>  
and is the <TYPE|!text: 1="Proposer"; 2="Responder"> in  
this period}
```

You have been assigned to Group 2 and is the Proposer in this period

OK

GENERAL (rtf codes)

<code>\tab</code>	tabulator
<code>\par</code>	new paragraph
<code>\line</code>	new line
<code>\bullet</code>	bullet
<code>\ql</code>	aligned to left
<code>\qr</code>	aligned to right
<code>\qc</code>	centered
<code>\b</code>	bold
<code>\b0</code>	not bold
<code>\i</code>	italic
<code>\i0</code>	not italic
<code>\sub</code>	small and inferior numbers (index)
<code>\super</code>	small and superior numbers (exponent)
<code>\strike</code>	crossed through
<code>\ul</code>	underline
<code>\ul0</code>	do not underline
<code>\colortbl</code>	Color table. See examples.
<code>\cfn</code>	Text color. <i>n</i> is the index of the color table which is defined by <code>\colortbl</code> .
<code>\fsn</code>	Font size <i>n</i> in units of half a dot. The font size must be explicitly given, otherwise it is larger (24) than usual in z-Leaf.

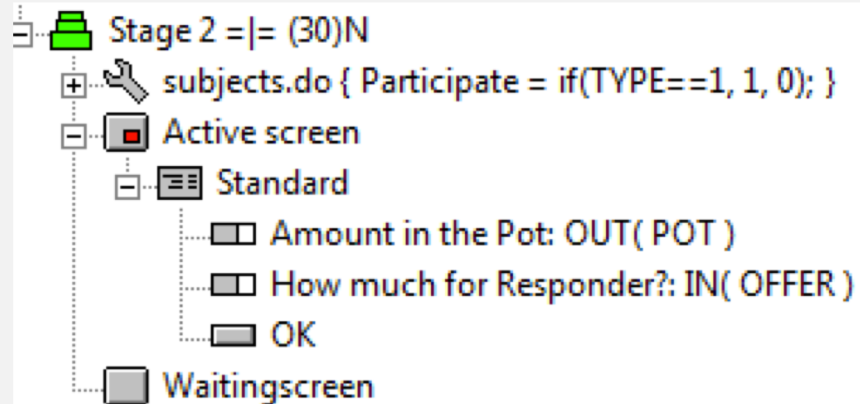
STAGE 2

(Proposer makes offer)

We only want the Proposer to enter stage 2

> Treatment > New program (subjects table)

```
Participate = if(TYPE==1, 1, 0);
```



The screenshot shows a dialog box with the following content:

Amount in the Pot	10.00
How much for Responder?	<input type="text" value=""/>

At the bottom right of the dialog box is a red 'OK' button.

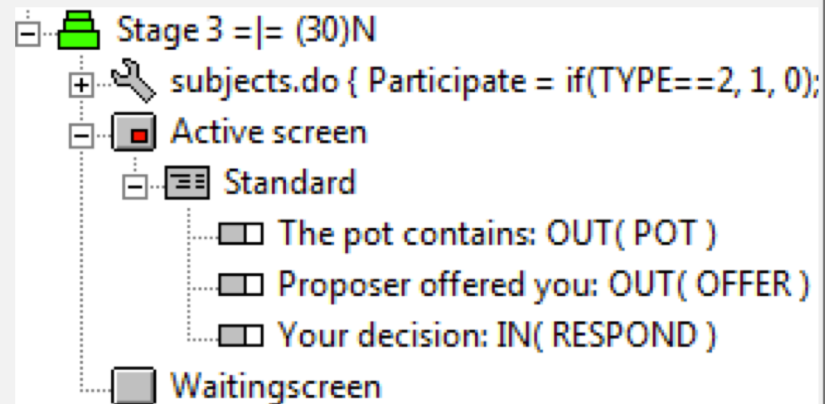
STAGE 3 (Responder Decides)

We only want the Responder to enter stage 3

> Treatment > New program (subjects table)

```
Participate = if(TYPE==2, 1, 0);
```

```
OFFER = find(same(Group) & TYPE==1, OFFER);
```



The pot contains 10.00

Proposer offered you 3.00

Your decision

Accept

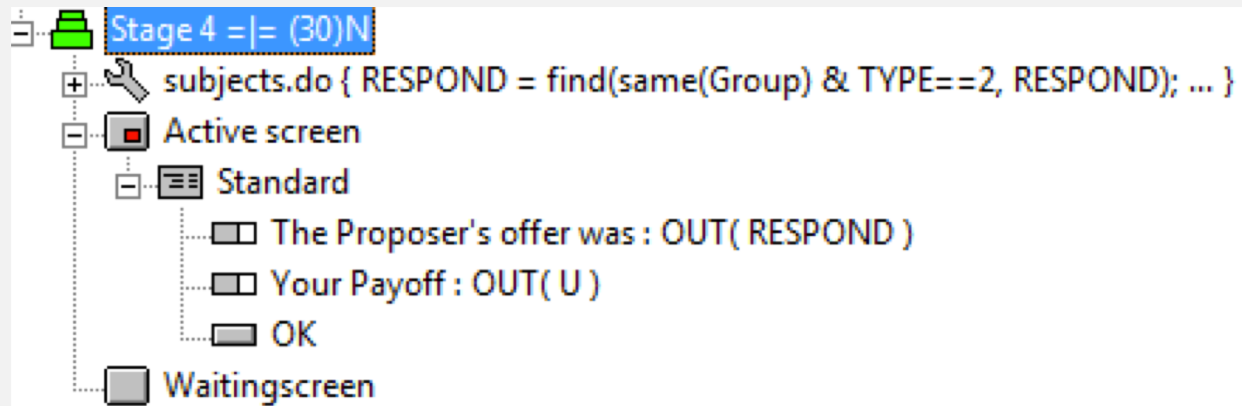
Reject

STAGE 4 (Compute payoff)

> Treatment > New program (subjects table)

```
RESPOND = find(same(Group) & TYPE==2, RESPOND);  
if (RESPOND == 2) {U=0;}  
elseif (RESPOND == 1)  
{  
    if (TYPE==2) {U=OFFER;}  
    elseif (TYPE==1) {U=POT-OFFER;}  
}
```

STAGE 4 (Compute payoff)



The Proposer's offer was Accepted

Your Payoff 5.00

OK

CLASS EXERCISE I

**SECOND PRICE
AUCTION**

TASK

- N=4 bidders
- Valuations between [0,100] uniform
- Bidders are endowed with $E = 200$
- 2nd price auction
- In the event of a tie, random allocation amongst all claimants

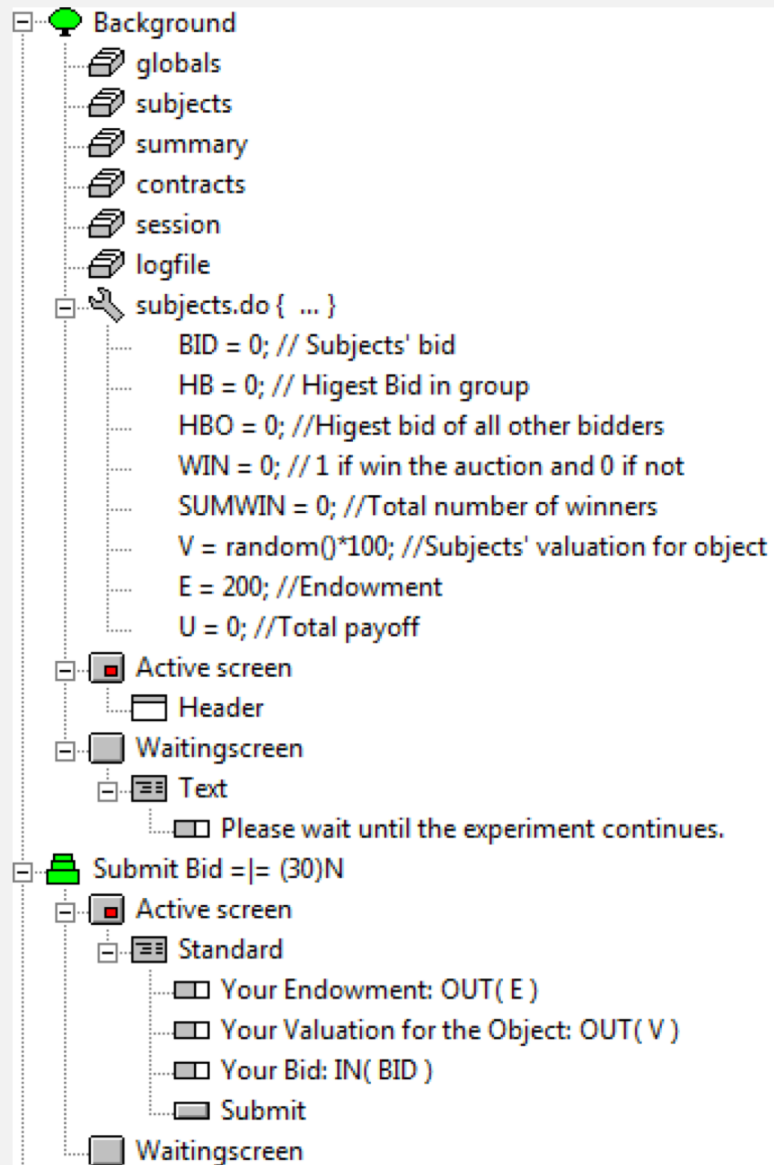
Some useful scope operators

```
Y = maximum (same (Group) , X) ;
```

```
Y = maximum (same (Group) & not (same (Subject)) , X) ;
```

```
Y = sum (same (Group) , X) ;
```

POTENTIAL SOLUTION



Your Endowment	200.00
Your Valuation for the Object	91.81
Your Bid	<input type="text" value=""/>

POTENTIAL SOLUTION

```
Results =|= (30)N
subjects.do { ... }
  HB = maximum(same(Group), BID);
  HBO = maximum(same(Group) & not(same(Subject)), BID);
subjects.do { ... }
  WIN = if(BID == HB, 1,0);
subjects.do { ... }
  SUMWIN = sum(same(Group), WIN);
subjects.do { ... }
  if(SUMWIN>1) { if(WIN==1) {T = random(); }}
subjects.do { ... }
  if(SUMWIN>1){if(WIN ==1) {TRANK = count(same(Group) & WIN==1 & T<=:T);}}
subjects.do { ... }
  if(SUMWIN>1){if(WIN==1 & TRANK!=1){WIN=0;}}
subjects.do { ... }
  U = if(WIN==1, E-HBO+V, E);
Active screen
  Standard
    Your Bid: OUT( BID )
    Highest Bid: OUT( HB )
    Total Number of Winners : OUT( SUMWIN )
    Did you win the auction (1=Yes, 0=No);: OUT( WIN )
    Your Payoff: OUT( U )
    OK
  Waitingscreen
```

Your Bid	4.00
Highest Bid	4.00
Total Number of Winners	1
Did you win the auction (1=Yes, 0=No);	1
Your Payoff	246.36

OK

LESSON PLAN

Day II

- **Creating multiple leafs on a screen**
- **Example III: 2x2 Normal form game**
 - Laying out Grid matrix
 - Random round payment
- **Example IV: Search Lottery**
 - Array programming and complex loops
 - Programming a Survey
- **Class Exercise II: Jackpot machine (A fair jackpot)**
- **Example V: Dutch Auction**
 - the “later” function
- **Class Exercise III: English Auction**



CREATING
MULTIPLE LEAFs
ON A SCREEN

MULTIPLE LEAFS

Welcome to



z-Leaf 3.6.7
The client software of z-Tree



Zurich
Toolbox for
Readymade
Economic
Experiments

Design: Urs Fischbacher

Programming: Urs Fischbacher
Stefan Schmid

Copyright © 1998-2016
University of Zurich
Department of Economics
Schoenberggasse 1
CH-8001 Zurich

<http://www.ztree.uzh.ch/>
ztree@econ.uzh.ch

Welcome to



z-Leaf 3.6.7
The client software of z-Tree



Zurich
Toolbox for
Readymade
Economic
Experiments

Design: Urs Fischbacher

Programming: Urs Fischbacher
Stefan Schmid

Copyright © 1998-2016
University of Zurich
Department of Economics
Schoenberggasse 1
CH-8001 Zurich

<http://www.ztree.uzh.ch/>
ztree@econ.uzh.ch

Welcome to



z-Leaf 3.6.7
The client software of z-Tree



Zurich
Toolbox for
Readymade
Economic
Experiments

Design: Urs Fischbacher

Programming: Urs Fischbacher
Stefan Schmid

Copyright © 1998-2016
University of Zurich
Department of Economics
Schoenberggasse 1
CH-8001 Zurich

<http://www.ztree.uzh.ch/>
ztree@econ.uzh.ch

Welcome to



z-Leaf 3.6.7
The client software of z-Tree



Zurich
Toolbox for
Readymade
Economic
Experiments

Design: Urs Fischbacher

Programming: Urs Fischbacher
Stefan Schmid

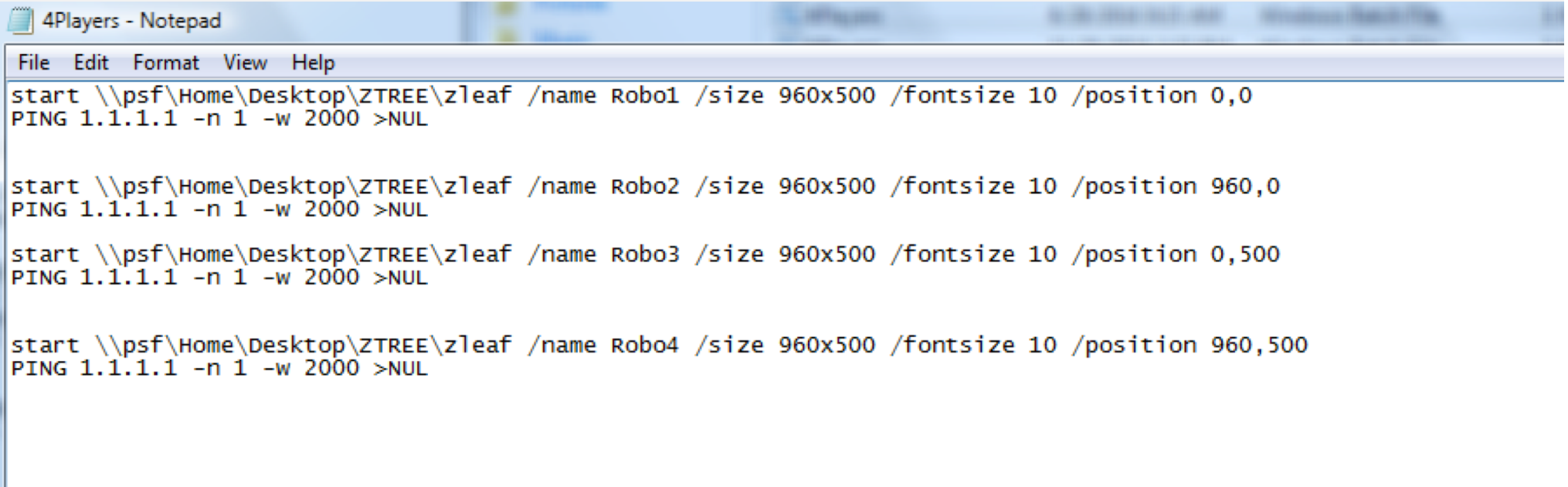
Copyright © 1998-2016
University of Zurich
Department of Economics
Schoenberggasse 1
CH-8001 Zurich

<http://www.ztree.uzh.ch/>
ztree@econ.uzh.ch

GENERAL

(Creating multiple leafs)

- Open "notepad"
- Write command lines
- Save file with suffix **.bat** (e.g, P4.bat)
- Open ztree and execute bat file



```
4Players - Notepad
File Edit Format View Help
start \\psf\Home\Desktop\ZTREE\zleaf /name Robo1 /size 960x500 /fontsize 10 /position 0,0
PING 1.1.1.1 -n 1 -w 2000 >NUL

start \\psf\Home\Desktop\ZTREE\zleaf /name Robo2 /size 960x500 /fontsize 10 /position 960,0
PING 1.1.1.1 -n 1 -w 2000 >NUL

start \\psf\Home\Desktop\ZTREE\zleaf /name Robo3 /size 960x500 /fontsize 10 /position 0,500
PING 1.1.1.1 -n 1 -w 2000 >NUL

start \\psf\Home\Desktop\ZTREE\zleaf /name Robo4 /size 960x500 /fontsize 10 /position 960,500
PING 1.1.1.1 -n 1 -w 2000 >NUL
```

EXAMPLE III

**2X2 NORMAL
FORM GAME**

DESIGN

	Today	Tomorrow
Today	200, 200	400, 0
Tomorrow	0, 400	R, R

- R can be either 300, 350, 400, ..., 800 with equal probability
- Subjects play 3 periods.
- Control question before starting the experiment
- Random period payment

** Note: Payoffs are *symmetric*, thus we don't have to worry about types.

INITIAL VALUES (globals table)

> Treatment > New Program (globals)

```
Outcome1 = 0;
```

```
Outcome2 = 200;
```


```
Outcome3 = 400;
```

```
Rand = random();
```

```
R = roundup(Rand*11, 1)*50 + 250;
```

```
Outcome4 = R;
```

```
Last_Period = 3;
```



Rand = random();	Y = Rand*11	X = roundup(Y , 1)	R = X*50+250
0.13425	1.4767	2	350
0.85932	9.4523	10	750
0.002	0.022	1	300

INITIAL VALUES (subjects table)

> Treatment > New Program (subjects)

```
X = 0; //Own decision
```

```
XO = 0; //Decision of other group player
```

```
U = 0; //Payoff for the period
```

```
if(Period==1)
```

```
{
```

```
    rr = random();
```

```
    Pay_Period = roundup(rr*Last_Period, 1)+0;
```

```
    Pay_Amount = 0;
```

```
}
```

```
elseif(Period>1)
```

```
{
```

```
    Pay_Period = OLDsubjects.find(same(Subject), Pay_Period);
```

```
    Pay_Amount = OLDsubjects.find(same(Subject), Pay_Amount);
```

```
}
```

GENERAL

(accessing data from previous period)

- The “lifespan” of the *subjects table* is only 1 period – reset at start of each period
- The command “**OLDsubjects**” accesses the *subjects table* in the immediate previous period – older periods are not accessible.

```
Y = OLDsubjects.find(same(Subject), X);
```

Period	Subject	Group	X	Y
1	1	1	3	0
1	2	1	6	0
2	1	1	2	3
2	2	1	9	6
3	1	1	5	2
3	2	1	2	9
4	1	1	3	5
4	2	1	5	2

PROCESS FLOW

STAGE 1

- Control Questions (Period 1 only)

STAGE 2

- See R
- Make Decision

STAGE 3

- Realise Payoff

STAGE 4

- See random chosen round and payment from that round (Period = 3 only)

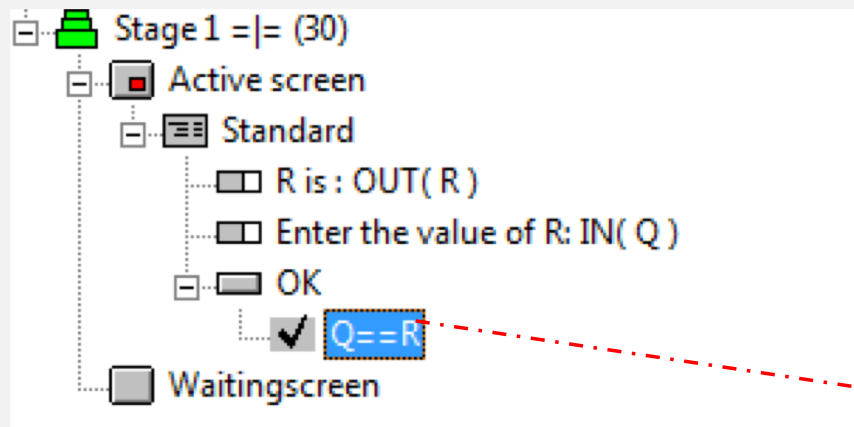
STAGE I

> Treatment > New Program (subjects)

```
Participate = if(Period==1, 1,0);
```

> Treatment > New Checker

The true condition that has to be met.



Checker

Condition: Q==R

Message (This message appears if the condition is not satisfied.): Please enter the correct R

"yes"-Button:

"no"-Button: OK

If there is only a "yes"-button, then the message appears and after pressing this button the input is accepted.

If there is only a "no"-button, then the message appears and after pressing this button the input is rejected.

If there is a "yes" AND a "no"-button, then the message should contain a question. Pressing the YES button accepts the input pressing NO rejects it.

Dialog

Please enter the correct R

OK

STAGE 2

Grid Box to show
2x2 matrix

Standard Box for subject's
input

> Treatment > New Box > Grid Box

Grid Box

Name: Grid with Frame

Width [p/]: Distance to the margin [p/]: 0% 50%

Height [p/]: Adjustment to the remaining box: left top right bottom

Display condition:

Num. Rows: 3 Num. Columns: 3

Input row-by-row (tab through rows)
 Input column-by-column (tab through columns)

First row contains labels Height [%]: 100
 First column contains labels Width [%]: 100

Separate labels by lines
 Separate rows by lines
 Separate columns by lines

Buttons: Position:

Sequence of buttons: In rows In column

OK Cancel

STAGE 2

Stage 2 =|= (30)N

- Active screen
 - Grid
 - Today
 - Tom.
 - Today
 - <>{\rtf\fs20\qc <Outcome2 |1>, <Outcome2 |1>}
 - <>{\rtf\fs20\qc <Outcome3 |1>, <Outcome1 |1>}
 - Tom.
 - <>{\rtf\fs20\qc <Outcome1 |1>, <Outcome3 |1>}
 - <>{\rtf\fs20\qc <Outcome4 |1>, <Outcome4 |1>}
 - Standard
 - You are the Row Player
 - Please enter your choice: IN(X)
- Waitingsscreen

	Today	Tom.
Today	200, 200	400, 0
Tom.	0, 400	700, 700

You are the Row Player

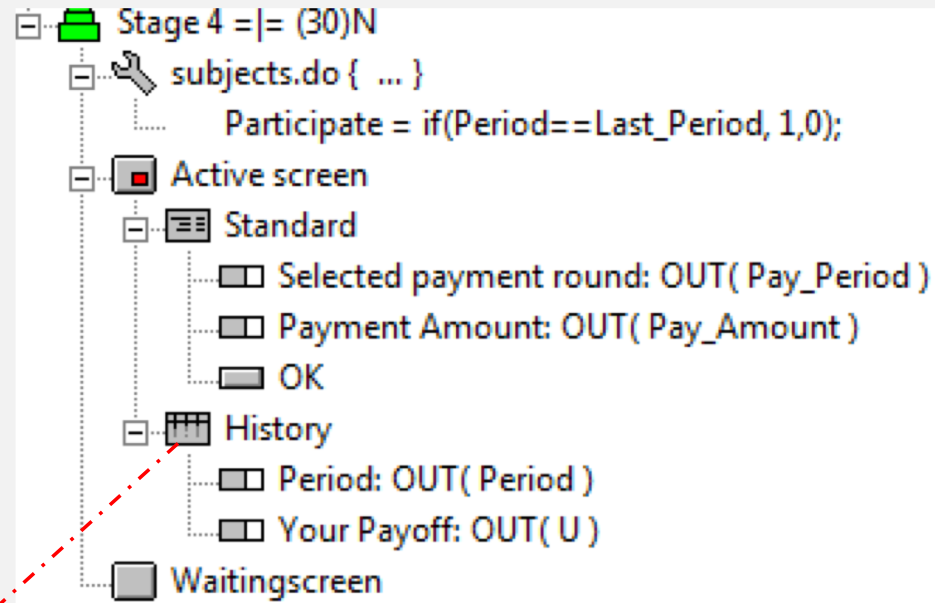
Please enter your choice

STAGE 3

```
Stage 3 =| (30)N
  subjects.do { ... }
    XO = find(same(Group) & not(same(Subject)), X);
  subjects.do { ... }
    if(X==1 & XO==1) {U=Outcome2;}
    elseif(X==1 & XO==2) {U=Outcome3;}
    elseif(X==2 & XO==1) {U=Outcome1;}
    elseif(X==2 & XO==2) {U=Outcome4;}
  Active screen
  Standard
    Your Payoffs for this round: OUT( U )
    OK
      subjects.do { ... }
        if(Period == Pay_Period)
        {
          Pay_Amount = U;
        }
  Waiting screen
```

Only update the **Pay_Amount** if the period is exactly that of the pre-determined payment period.

STAGE 4



Period	Your Payoff
1	0
2	0
3	200

Selected payment round	3
Payment Amount	200

OK

EXAMPLE IV

**SEARCH
LOTTERY**

DESIGN

- Search for an “Object” by putting in some **effort level** $\{0, 5, 10, 15, \dots, 100\}$, which denotes the probability of finding a Prize (worth \$50)
- Greater effort corresponds to greater cost.
- Run Survey after session
- *Note:* When a subject chooses an effort level , he gets to observe the corresponding cost first to which he has to confirm – he is able to revise his decision.

EFFORT	0	5	10	15	20	25	30	35	100
COST	0	2	4	6	8	10	12	14	40

SOME CONSIDERATIONS

The simple approach

```
// Effort is the input parameter  
  
if(Effort == 0) {Cost = 0;}  
elseif(Effort == 5) {Cost = 2;}  
elseif(Effort == 10) {Cost = 4;}  
...  
elseif(Effort == 100) {Cost = 40;}
```

Can we do this more efficiently?

```
// Effort is the input parameter  
  
Cost = Effort/5*2;
```

However, this is because this example's parameters are convenient – Lets think about this for the more general case.

GENERAL (the Array Parameter)

defines an array with indices from 1 to n
`arrayvar[n];`

defines an array with indices from x to y
`array arrayvar[x, y];`

defines an array with indices from x to y with distance d.
`array arrayvar[x, y, d];`

USING THE ARRAY

```
Cost = 0;  
array C[0,20]; // define the array  
  
//Input variables into the array  
C[0] = 0;  
C[1] = 2;  
C[2] = 4;  
...  
C[20] = 40;  
  
//Now match the effort to the C array  
Cost = C[Effort/5];
```

Suppose that we are too "lazy" to input $C[0], C[1], \dots, C[20]$

GENERAL (generating loops)

Basic Loop

```
if ( condition ) { statements if condition is true;}  
elseif ( condition) {statements if condition is true;}
```

While Loop

```
while( condition ) {statements if condition is true; }
```

Repeat Loop

```
repeat { statements } while ( condition );
```

Iterate Loops

```
iterator( varname, y ) //runs from 1 to y  
iterator( varname, x, y ) //runs from x to y  
iterator( varname, x, y, d )  
//runs from x to y with steps of d.
```


GENERAL (generating loops)

Calculating: $Y = 1+4+9+16+25 = 55$

```
Y = 0;  
iterator(i,5).do {  
  :Y = :Y + i * i;  
}
```

i	Y
1	1
2	
3	
4	
5	

i	Y
1	1
2	5
3	
4	
5	

i	Y
1	1
2	5
3	14
4	30
5	25

i	Y
5	25

INITIAL VALUES

```
// Globals
```

```
Prize = 50;
```

```
array C[0,20];
```

```
iterator(i,21).do {  
    C[i-1] = (i-1)*2;  
}
```

```
//Subjects
```

```
Effort = 0;
```

```
Cost = 0;
```

```
Box = 0;
```

```
U = 0;
```

```
Find = 0;
```

STAGE 1

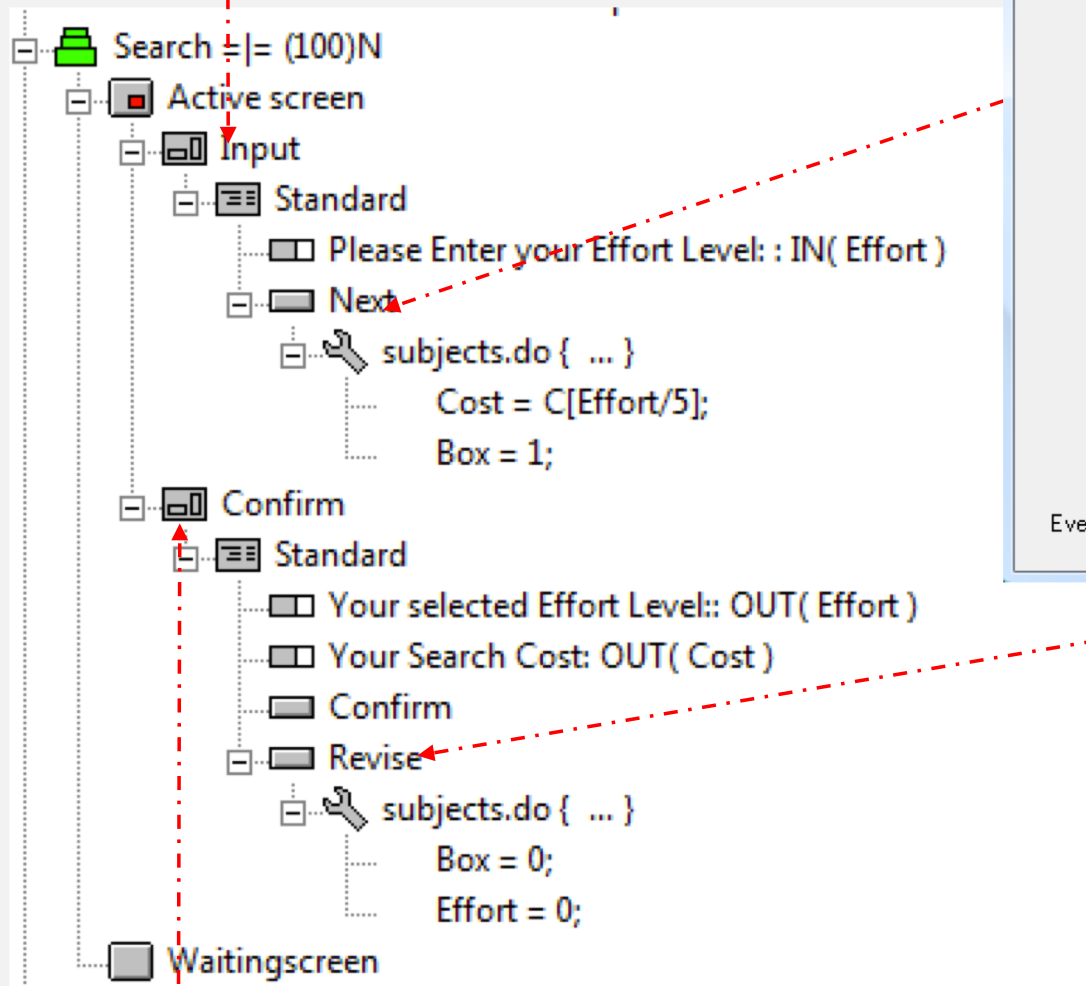
- Choose Effort (and see Cost)

STAGE 2

- See Search Outcome

STAGE I

If Box == 0



Button

Name:

No record created or selected

Clear entry after OK

Leave Stage

Yes

No

Normal (i.e. stage is not left after click if stage is left after timeout and button is contained in contract creation or selection box)

Color

Automatic

Gray

Red

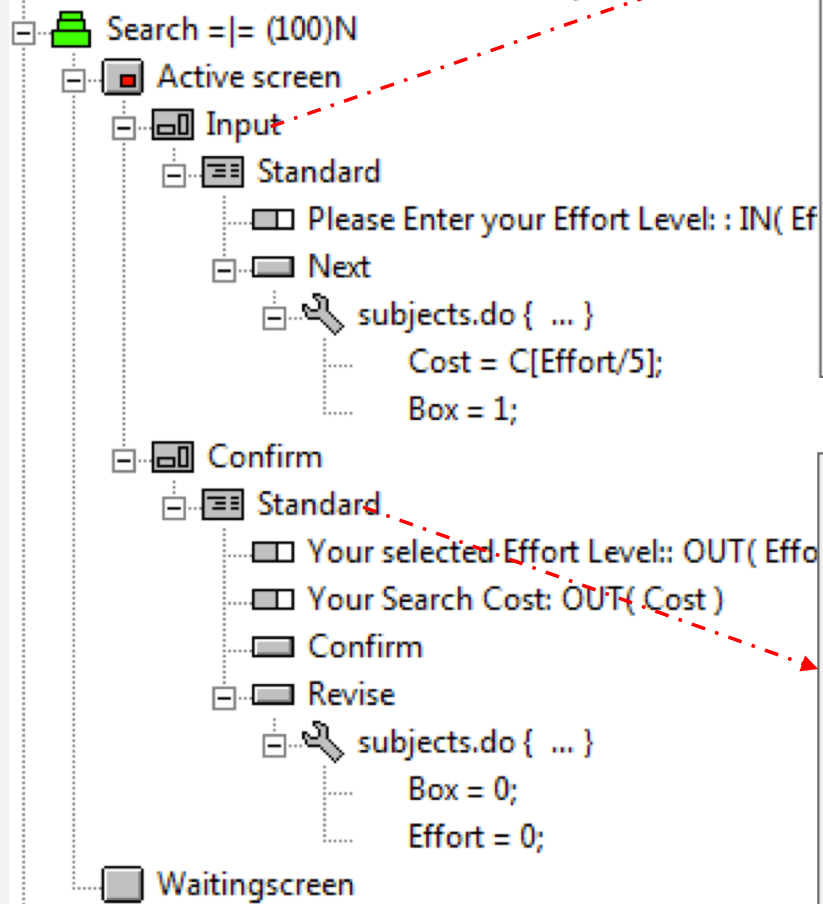
Event time:

OK

Cancel

If Box == 1

STAGE I



Please Enter your Effort Level:

Next

Your selected Effort Level: 30
Your Search Cost 12.00

Revise

Confirm

STAGE 2

```
Result =|= (30)
subjects.do { ... }
  rr = random()*100;
  if(Effort >= rr) {Find=1;}

  if(Find==0)
  {
  U = - Cost;
  }
  elsif(Find==1)
  {
  U = - Cost + Prize;
  }

Active screen
  Standard
    Did you find the prize? (1=Yes, 0 =No): OUT( Find )
    Your Payoff: OUT( U )
    Survey
      session.do { ... }
      U = :U;

Waitingscreen
```

Did you find the prize? (1=Yes, 0 =No) 0

Your Payoff -20.00

Survey

Write into the "session table"

GENERAL (session table)

- One row per subject

Subject	FinalProfit	ShowUpFee	ShowUpFeel nvested	MoneyAdded	MoneyToP ay	MoneyEarne d	X
1	12.65	0	0	0	0	0	3
2	11	0	0	0	0	0	4
3	12	0	0	0	0	0	56
4	9	0	0	0	0	0	8
5	16	0	0	0	0	0	9

SURVEYS

The survey design always starts with an “Address form”

> Questionnaire > New Address Form

A tree view of a survey design. The 'Address' node is highlighted with a blue selection box. Below it are two main sections: 'Demographics:' and 'Bye:'. Under 'Demographics:', there are three items: 'What is your age: IN(Age)', 'Explain your behaviour: IN(Explain)', and 'Next'. Under 'Bye:', there is one item: 'Thank you and bye'. Each item has a small icon to its left, and the 'Demographics:' and 'Bye:' sections have a '??' icon.

Leave the details “empty” if you want to skip the address form

A screenshot of the 'Address' form. The form has a title bar 'Address' and a close button. It contains several input fields: 'Address Entry', 'First Name', 'Last Name', 'Address', 'Postal code', 'City', 'Telephone', and 'E-Mail'. There are 'OK' and 'Cancel' buttons on the right. Below the input fields is a section for 'Do you want to participate in further experiments?' with 'Yes' and 'No' radio buttons. There is a 'Continue (button label)' field with the value 'next', a 'Help' field, and a 'Help text' field with a scrollable area.

SURVEYS

> Questionnaire > New Question Form

A screenshot of a survey form structure. The form is organized into sections, each with a collapse icon (minus sign) and a status icon (question mark). The sections are:

- Address** (status icon: list icon)
- Demographics** (status icon: question mark)
 - What is your age: IN(Age) (input type: IN)
 - Explain your behaviour: IN(Explain) (input type: IN)
 - Next (input type: button)
- Bye:** (status icon: question mark)
 - Thank you and bye (input type: button)

Red dashed arrows point from the text boxes on the right to the 'Demographics' section and the 'Next' button.

> Questionnaire > New Button

SURVEYS

The image shows two side-by-side 'Question' dialog boxes. The left dialog is for a question labeled 'Explain your behaviour' with variable 'Explain'. It is configured as a 'Text' type, with 'Wide' and 'Input' checked, and 'Empty allowed' unchecked. The number of rows is set to 20. The right dialog is for a question labeled 'What is your age' with variable 'Age'. It is configured as a 'Number' type, with 'Input' checked and 'Wide' and 'Empty allowed' unchecked. The minimum value is 0, the maximum is 100, and the resolution is 1. Both dialogs have 'Cancel' and 'OK' buttons at the bottom.

Question 1 (Left):

- Label: Explain your behaviour
- Variable: Explain
- Type: Text
- Wide:
- Input:
- Empty allowed:
- Num. rows: 20

Question 2 (Right):

- Label: What is your age
- Variable: Age
- Type: Number
- Wide:
- Input:
- Empty allowed:
- Minimum: 0
- Maximum: 100
- Resolutor: 1

SURVEYS

What is your age

Explain your behaviour

Next

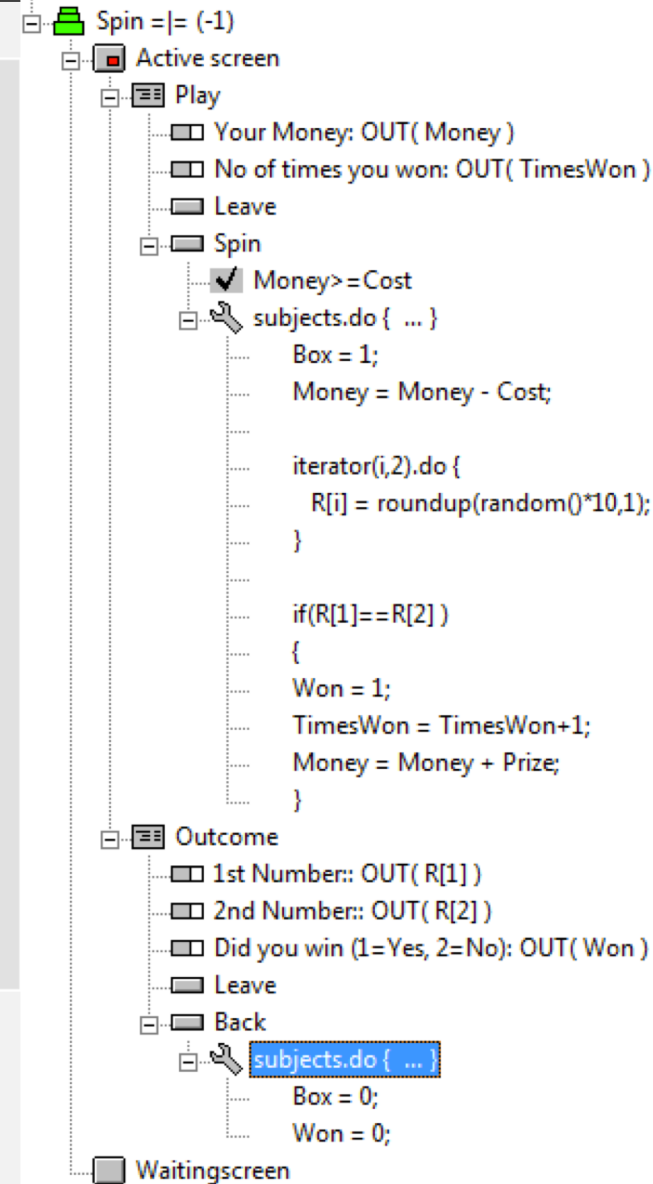
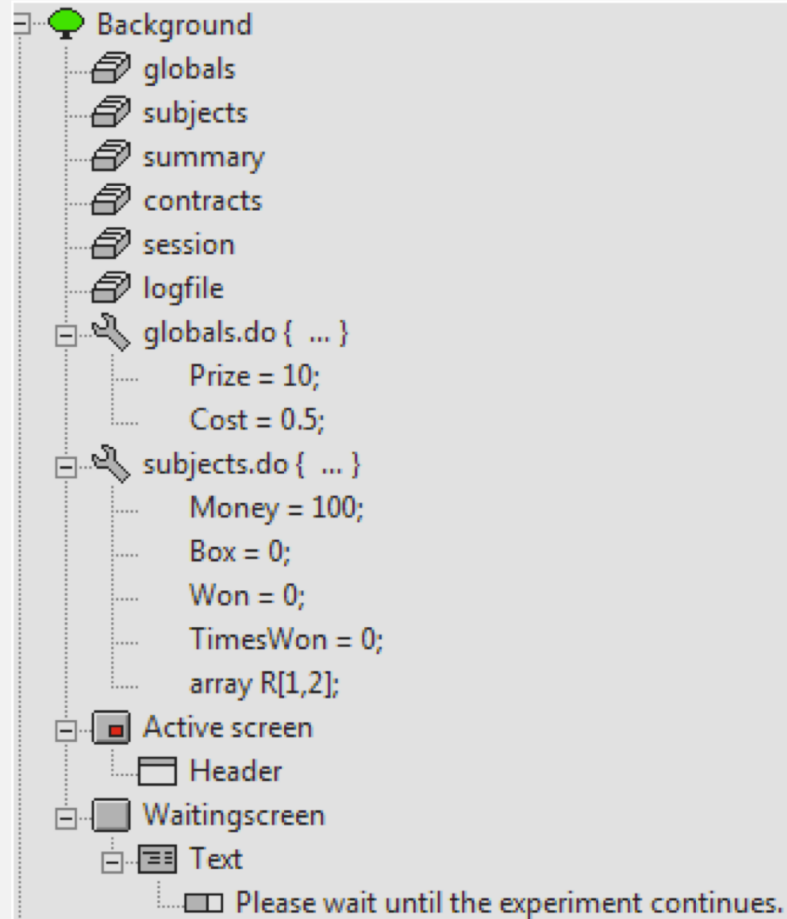
CLASS EXERCISE II

**JACKPOT
MACHINE**

TASK

- Do a simple jackpot machine consisting of two numbers (1,2,...,10).
- Subject wins a prize if the two number are identical.
- Subject gets to "spin" the jackpot as many times as he wants – subjected to budget constraint.
- For each spin:
 - Some money gets deducted (Tokens cost)
 - New random numbers (1,2,...,10) are generated
 - Prize money is added if subject wins
- Subject can also decide to leave the jackpot and cash out

POSSIBLE SOLUTION



POSSIBLE SOLUTION

Your Money 100.00
No of times you won 0

Spin

Leave

1st Number: 8
2nd Number: 1
Did you win (1=Yes, 2=No) 0

Back

Leave

EXAMPLE V

**DUTCH
AUCTION**

DESIGN

- There is **1 object** that is to be sold between **4 bidders**
- The auction starts at the Price of **\$150**.
- **Every 3 seconds**, the Price **reduces by \$10**.
- A Bidder buys the object at the stated price by clicking the “**Buy button**”
- The auction ends for everyone in the group once someone in the group buys the object.

GENERAL (the "later" function)

```
later( expression ) repeat { statements }
```

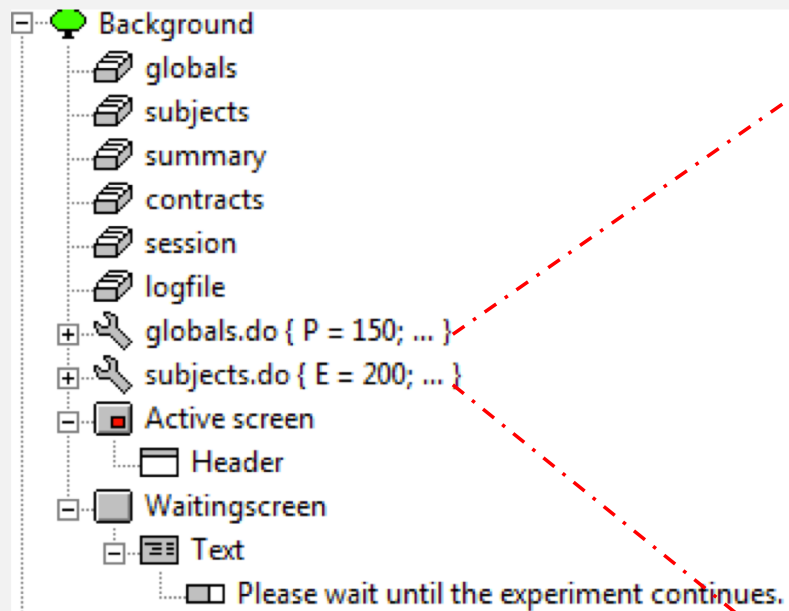
Note that the function does not have a build-in "while" condition.

```
Background > Treatment > New Program > Globals
```

```
P = 150; //Starting price $150  
later(3) repeat  
{  
  P = P-10; //Every 3 seconds reduce price by $10  
}
```

Prices can go below 0!

BACKGROUND (INITIAL VALUES)



```
P = 150;  
R = 80;  
later ( 3 ) repeat  
{  
    P = P - 10;  
    if(P<R)  
    {  
        P = R;  
    }  
}
```

```
E = 200;  
V = random() * 100;  
U = 0;  
Buy = 0;  
Final_Price = 0;
```

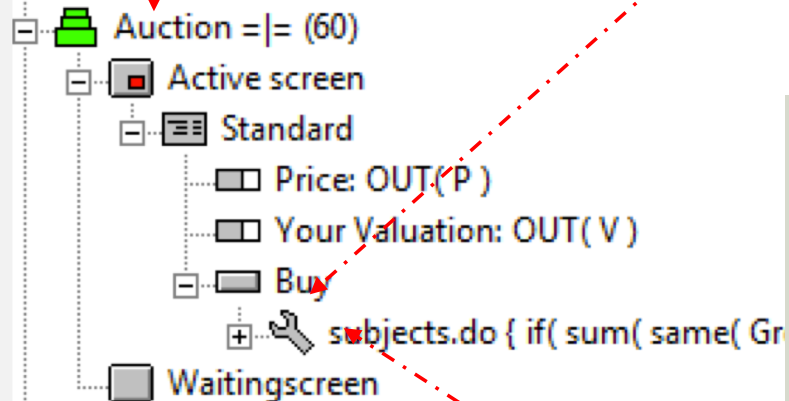
AUCTION STAGE

Leave stage after timeout

- If no input Yes No

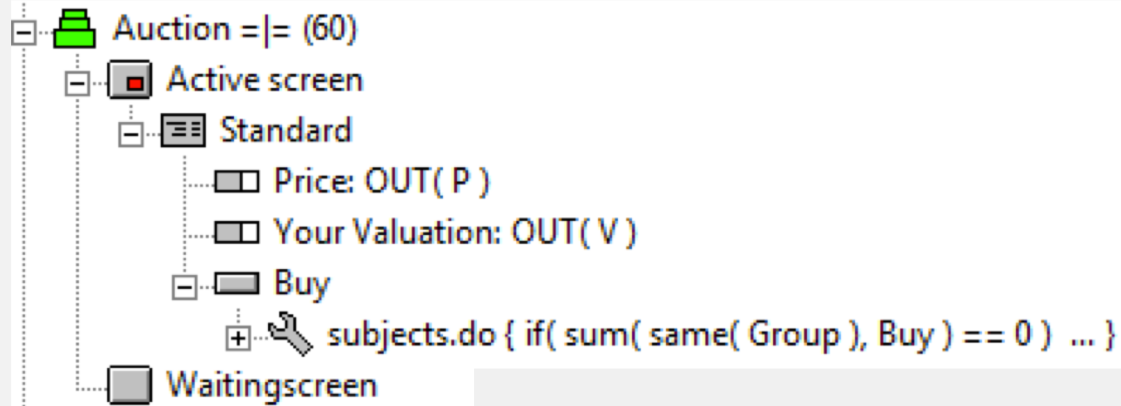
Leave Stage

- Yes
 No
 Normal (i.e. stage is not left after click if stage is left after timeout and button is contained in contract creation or selection box)



```
if (sum (same (Group, Buy) ==0)
{
  Buy = 1;
  subjects.do{
    if( same( Group ) )
    {
      LeaveStage = 1;
      Final_Price = P;
    }
  }
}
```

AUCTION STAGE



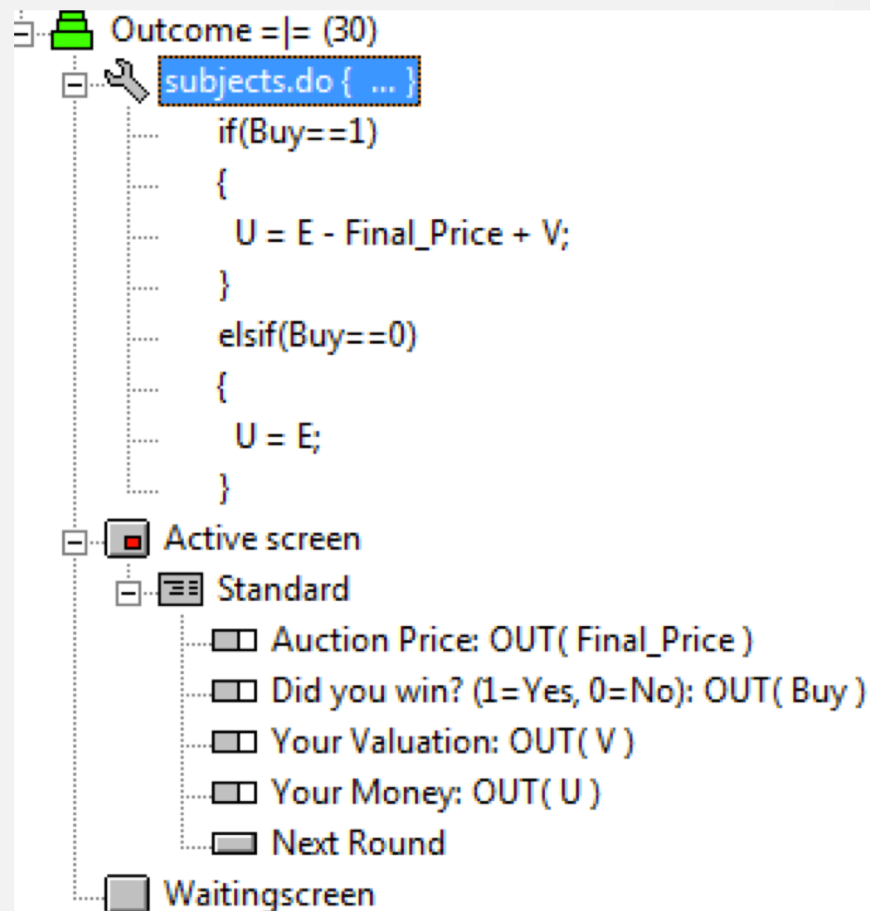
Price	120.00
Your Valuation	85.88

Buy

RESULT STAGE

Auction Price	110.00
Did you win? (1=Yes, 0=No)	1
Your Valuation	85.88
Your Money	165.88

Next Round



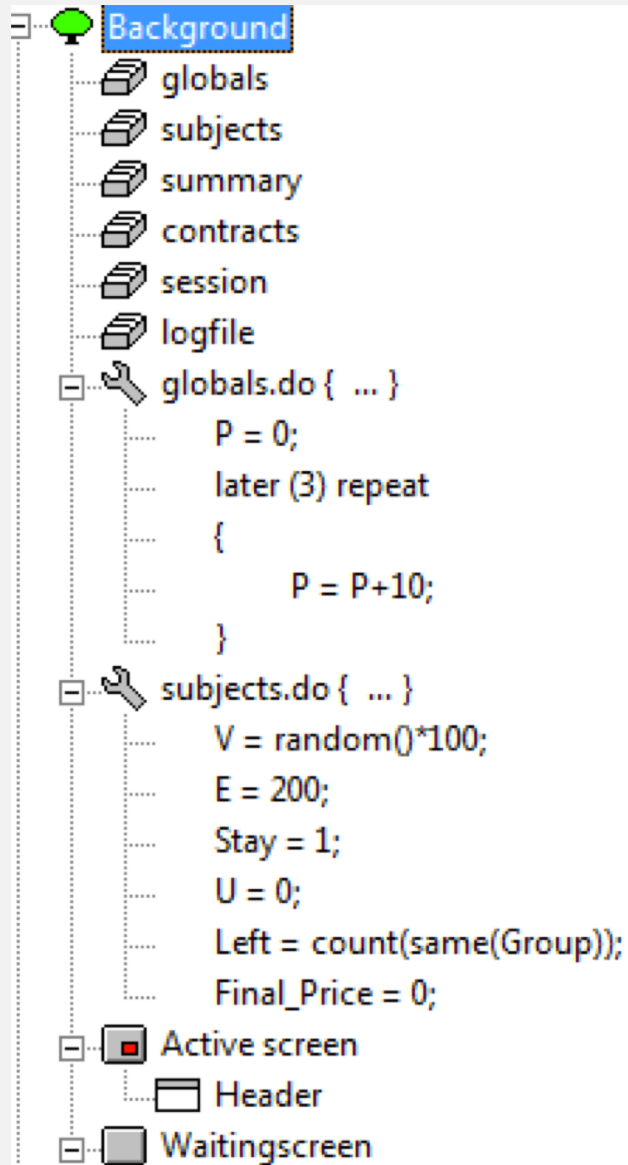
CLASS EXERCISE
III

**ENGLISH
AUCTION**

DESIGN

- There is 1 object that is to be sold between 4 bidders
- The auction starts at the Price of \$0.
- Every 3 seconds, the Price reduces by \$10.
- A Bidder in the auction can choose to leave the auction.
 - Each time someone leaves, all other bidder sees the total number of remaining bidders
- The auction ends for everyone in the group once there is only 1 bidder left in the auction – auction price determined.

POSSIBLE SOLUTION (BACKGROUND STAGE)



POSSIBLE SOLUTION (AUCTION STAGE)

```
Auction Stage =|= (30)
├── Active screen
│   ├── Standard
│   │   ├── Price: OUT( P )
│   │   ├── Your Valuation:: OUT( V )
│   │   ├── No. of Bidders Left: OUT( Left )
│   │   └── Leave Auction
│   │       └── subjects.do { ... }
│   │           ├── if(count(same(Group) & Stay==1) > 1)
│   │           │   ├── {
│   │           │   │   ├── Stay = 0;
│   │           │   │   ├── subjects.do{
│   │           │   │   │   ├── if(same(Group))
│   │           │   │   │   │   ├── {
│   │           │   │   │   │   │   ├── Left = Left -1;
│   │           │   │   │   │   │   └── }
│   │           │   │   │   │   └── }
│   │           │   │   └── }
│   │           │   └── }
│   │           ├── subjects.do{
│   │           │   ├── if(count(same(Group) & Stay==1) == 1)
│   │           │   │   ├── {
│   │           │   │   │   ├── subjects.do{
│   │           │   │   │   │   ├── if(same(Group))
│   │           │   │   │   │   │   ├── {
│   │           │   │   │   │   │   │   ├── LeaveStage = 1;
│   │           │   │   │   │   │   │   ├── Final_Price = P;
│   │           │   │   │   │   │   └── }
│   │           │   │   │   │   └── }
│   │           │   │   └── }
│   │           │   └── }
│   │           └── }
└── Waitingscreen
```

Price	10.00
Your Valuation:	28.27
No. of Bidders Left	4

Leave Auction

POSSIBLE SOLUTION (RESULT STAGE)

```
Results =| (-1)N
subjects.do { ... }
  if(Stay==1)
  {
    U = E - Final_Price + V;
  }
  elseif(Stay ==0)
  {
    U = E;
  }
Active screen
  Standard
    Final Price: OUT( Final_Price )
    Did you win? (1=yes, 2=No): OUT( Stay )
    Your Money: OUT( U )
    Next Period
  Waitingscreen
```

Final Price	70.00
Did you win? (1=yes, 2=No)	0
Your Money	200.00

Next Period

LESSON PLAN

Day III

- **Example VI: Continuous Double Auction**
 - Introduction to the Contract table
- **Example VII: Random Stopping Public Goods Game**
 - Creating *infinite* length games
- **Example VIII: Complex Move games**
 - Inserting Figures / Videos
 - Designing complex sequential move formats
- **Example IX: Chat Box**
- **Example X: 2-Dimesion Graphing**
 - Bars
 - Lines
- **Example XI: Graphing Pie Charts**
- **Exercise IV:** Vernon Smith, Gerry Suchanek and Arlington Williams (1988) design with Graphed prices.

EXAMPLE VI

**CONTINUOUS
DOUBLE AUCTION
MARKET**

DESIGN

- One-Period market involving $N=4$ traders
- Each trader endowed with \$1000 and 10 assets
- Trade facilitated through continuous double auction

Period		1 of 1		Remaining time [sec]: 112	
Your Money: 1250.00					
Your Stock: 8					
	Market Ask Price	Market Price	Market Bid Price		
	100.00	50.00	50.00		
		100.00	60.00		
		100.00			
		100.00			
Your Ask	<input type="text" value="100"/>			Your Bid	<input type="text" value="50"/>
ASK	BUY		Sell	BID	

Ask Prices of everyone

inventory

Bid Prices of Everyone

Period: 1 of 1 Remaining time [sec]: 112

Your Money: 1250.00
Your Stock: 8

Market Ask Price	Market Price	Market Bid Price
100.00	50.00	50.00
	100.00	60.00
	100.00	
	100.00	

Your Ask: Your Bid:

ASK BUY Sell BID

Market transaction prices

Ask Price (i.e., How much you want to sell a stock at)

Bid Price (i.e., How much you want to buy a stock at)

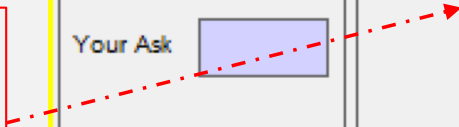
Period	Seller	Buyer	Maker	P	Traded	contractID	tradeID
1	3	-1	3	50	0	1	0
1	3	-1	3	45	0	2	0
1	4	-1	4	60	0	3	0
1	4	-1	4	35	0	4	0

Period: 1 of 1 Remaining time [sec]: 190

Your Money: 1000.00
Your Stock: 10

	Market Ask Price	Market Price	Market Bid Price	
	60.00 50.00 45.00 35.00			
Your Ask <input style="width: 50px;" type="text"/>				Your Bid <input style="width: 50px;" type="text"/>
<input type="button" value="ASK"/>	<input type="button" value="BUY"/>		<input type="button" value="Sell"/>	<input type="button" value="BID"/>

Lowest Ask price below



Period	Seller	Buyer	Maker	P	Traded	contractID	tradeID
1	3	-1	3	50	0	1	0
1	3	-1	3	45	0	2	0
1	4	-2	4	60	0	3	0
1	4	1	4	35	1	4	1

Period: 1 of 1 Remaining time [sec]: 178

Your Money: 985.00
Your Stock: 11

	Market Ask Price	Market Price	Market Bid Price	
	50.00	35.00		
	45.00			
Your Ask	<input type="text"/>			Your Bid
<input type="button" value="ASK"/>	<input type="button" value="BUY"/>		<input type="button" value="Sell"/>	<input type="button" value="BID"/>

Transaction Price Updated
Subjects' inventory updated

Ask price from Seller 4 removed

TYPES OF CONTRACT BOXES

Contract List Box (Output)

Standard Box

Period: 1 of 1 Remaining time [sec]: 112

Your Money: 1250.00
Your Stock: 8

Market Ask Price	Market Price	Market Bid Price
100.00	50.00	50.00
	100.00	60.00
	100.00	
	100.00	

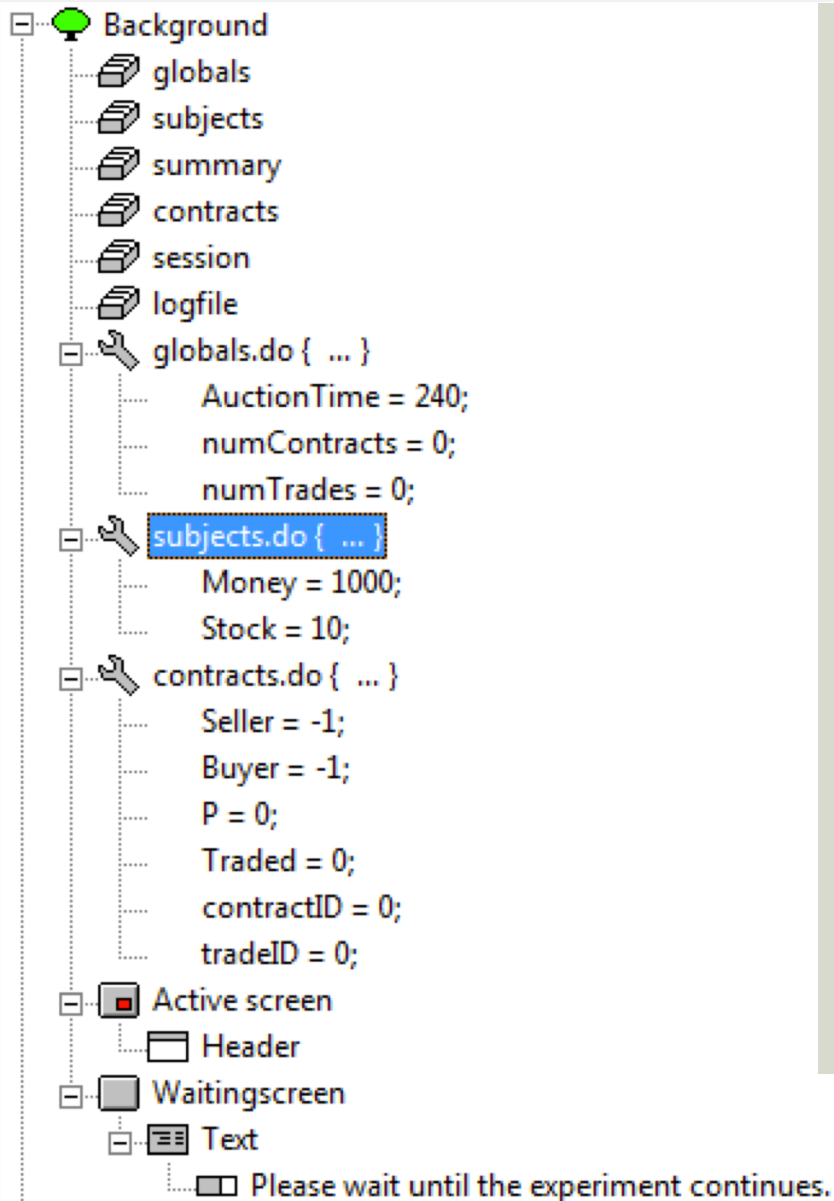
Your Ask:

Your Bid:

Contract creation box (input)

BACKGROUND STAGE

(initial values)



Globals

```
AuctionTime = 240;
numContracts = 0;
numTrades = 0;
```

Subjects

```
Money = 1000;
Stock = 10;
```

Contracts

```
Seller = -1;
Buyer = -1;
P=0;
Traded = 0;
contractID = 0
tradeID = 0
```

AUCTION STAGE (initial values)

- Market =|=(AuctionTime)A
 - Active screen
 - Inventory
 - Your Money:: OUT(Money)
 - Your Stock: OUT(Stock)
 - make: Ask: contracts
 - To Buy: contracts(Buyer == -1), sorted by: -P; -contractID
 - Contract list: contracts((Buyer > 0) & (Seller>0)), sorted by: tradeID
 - To Buy: contracts(Seller == -1), sorted by: P; -contractID
 - make: bid: contracts
 - Waitingscreen

Period: 1 of 1 Remaining time [sec]: 112

Your Money: 1250.00
Your Stock: 8

	Market Ask Price	Market Price	Market Bid Price	
	100.00		50.00 100.00 100.00 100.00	
Your Ask <input style="width: 40px;" type="text" value="100"/>				Your Bid <input style="width: 40px;" type="text" value="50"/>
<input type="button" value="ASK"/>	<input type="button" value="BUY"/>		<input type="button" value="Sell"/>	<input type="button" value="BID"/>

AUCTION STAGE (initial values)

- Market =|=(AuctionTime)A
 - Active screen
 - Inventory
 - Your Money:: OUT(Money)
 - Your Stock: OUT(Stock)
 - make: Ask: contracts
 - To Buy: contracts(Buyer == -1), sorted by: -P; -contractID
 - Contract list: contracts((Buyer > 0) & (Seller > 0)), sorted by: tradeID
 - To Buy: contracts(Seller == -1), sorted by: P; -contractID
 - make: bid: contracts
 - Waitingscreen

Period: 1 of 1 Remaining time [sec]: 112

Your Money: 1250.00
Your Stock: 8

	Market Ask Price	Market Price	Market Bid Price	
	100.00		50.00	
		100.00	60.00	
		100.00		
		100.00		
Your Ask <input style="width: 40px;" type="text" value="100"/>				Your Bid <input style="width: 40px;" type="text" value="50"/>
<input type="button" value="ASK"/>	<input type="button" value="BUY"/>		<input type="button" value="Sell"/>	<input type="button" value="BID"/>

Market =|= (AuctionTime)A

Active screen

Inventory

make: Ask: contracts

Your Ask: IN(P)

ASK

Stock>0

contracts.do { ... }

Seller = :Subject;

Buyer = -1;

Maker = :Subject;

Traded = 0;

\numContracts = \numContracts + 1;

contractID = \numContracts;

To Buy: contracts(Buyer == -1), sorted by: -P; -contractID

Contract list: contracts(Buyer > 0) & (Seller > 0), sorted by: tradeID

To Buy: contracts(Seller == -1), sorted by: P; -contractID

make: bid: contracts

Waitingscreen

Period 1 of 1 Remaining time [sec]: 112

Your Money: 1250.00
Your Stock: 8

Market Ask Price	Market Price	Market Bid Price
100.00	50.00	50.00
	100.00	60.00
	100.00	
	100.00	

Your Ask: 100

Your Bid: 50

ASK BUY Sell BID

>treatment > New Box > Contract creation box

```

To Buy: contracts( Buyer == -1 ), sorted by: -P; -contractID
  Market Ask Price: OUT( P )
  BUY
    Seller != :Subject
    Money >= P
    contracts.do { ... }
      Buyer = :Subject;

      :Money = :Money - P;
      :Stock = :Stock + 1;
      Traded = 1;
      \numTrades = \numTrades+1;
      tradeID = \numTrades;

      subjects.do{
        if(:Seller == Subject)
        {
          Money = Money + P;
          Stock = Stock - 1;
        }
      }

```

Period 1 of 1 Remaining time [sec]: 112

Your Money: 1250.00
Your Stock: 8

Market Ask Price	Market Price	Market Bid Price
100.00	50.00	50.00
	100.00	60.00
	100.00	
	100.00	

Your Ask: 100
Your Bid: 50

ASK BUY Sell BID

>treatment > New Box > Contract List box

- If condition (Buyer==1)
- Sort (-P; -contractID;)

```

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

```

Removes seller's other ask prices from "Market Ask Price"

Contract list: contracts((Buyer > 0) & (Seller>0)), sorted by: tradeID

Market Price: OUT(P)

To Buy: contracts(Seller == -1), sorted by: P; -contractID

Market Bid Price: OUT(P)

Sell

Buyer != :Subject

Stock > 0

contracts.do { ... }

Seller = :Subject;

:Money = :Money + P;

:Stock = :Stock - 1;

Traded = 1;

\numTrades = \numTrades+1;

tradeID = \numTrades;

subjects.do{

if(:Buyer == Subject)

{

Money = Money - P;

Stock = Stock + 1;

}

}

contracts.do{

if (Buyer == :Buyer & Seller == -1)

{

Seller = -2;

}

if (Seller == :Seller & Buyer == -1)

{

Buyer = -2;

}

}

Period 1 of 1 Remaining time [sec]: 112

Your Money: 1250.00
Your Stock: 8

Market Ask Price	Market Price	Market Bid Price
100.00	50.00	50.00
	100.00	60.00
	100.00	
	100.00	

Your Ask: 100

Your Bid: 50

ASK BUY Sell BID

```

make: bid: contracts
├── Your Bid: IN( P )
├── BID
│   ├── Money >= P
│   └── contracts.do { ... }
│       ├── Seller = -1;
│       ├── Buyer = :Subject;
│       ├── Maker = :Subject;
│       ├── Traded = 0;
│       ├── \numContracts = \numContracts + 1;
│       └── contractID = \numContracts;

```

Period 1 of 1 Remaining time [sec]: 112

Your Money: 1250.00
Your Stock: 8

	Market Ask Price	Market Price	Market Bid Price	
	100.00	50.00	50.00	
		100.00	60.00	
		100.00		
		100.00		

Your Ask: 100 Your Bid: 50

ASK BUY Sell BID

EXAMPLE VII

**RANDOM
STOPPING PUBLIC
GOODS GAME**

DESIGN

- Publics Good game session which stops at the period with probability $\frac{1}{2}$.

```
globals.do { ... }  
..... N = 4; //no. of players in a group  
..... E = 10; // endowment  
..... M = 1.2; // multiplier  
..... rr = random();  
..... if(rr<=0.5)  
..... {  
..... RepeatTreatment = 0;  
..... }  
..... elseif(rr>0.5)  
..... {  
..... RepeatTreatment = 1;  
..... }
```

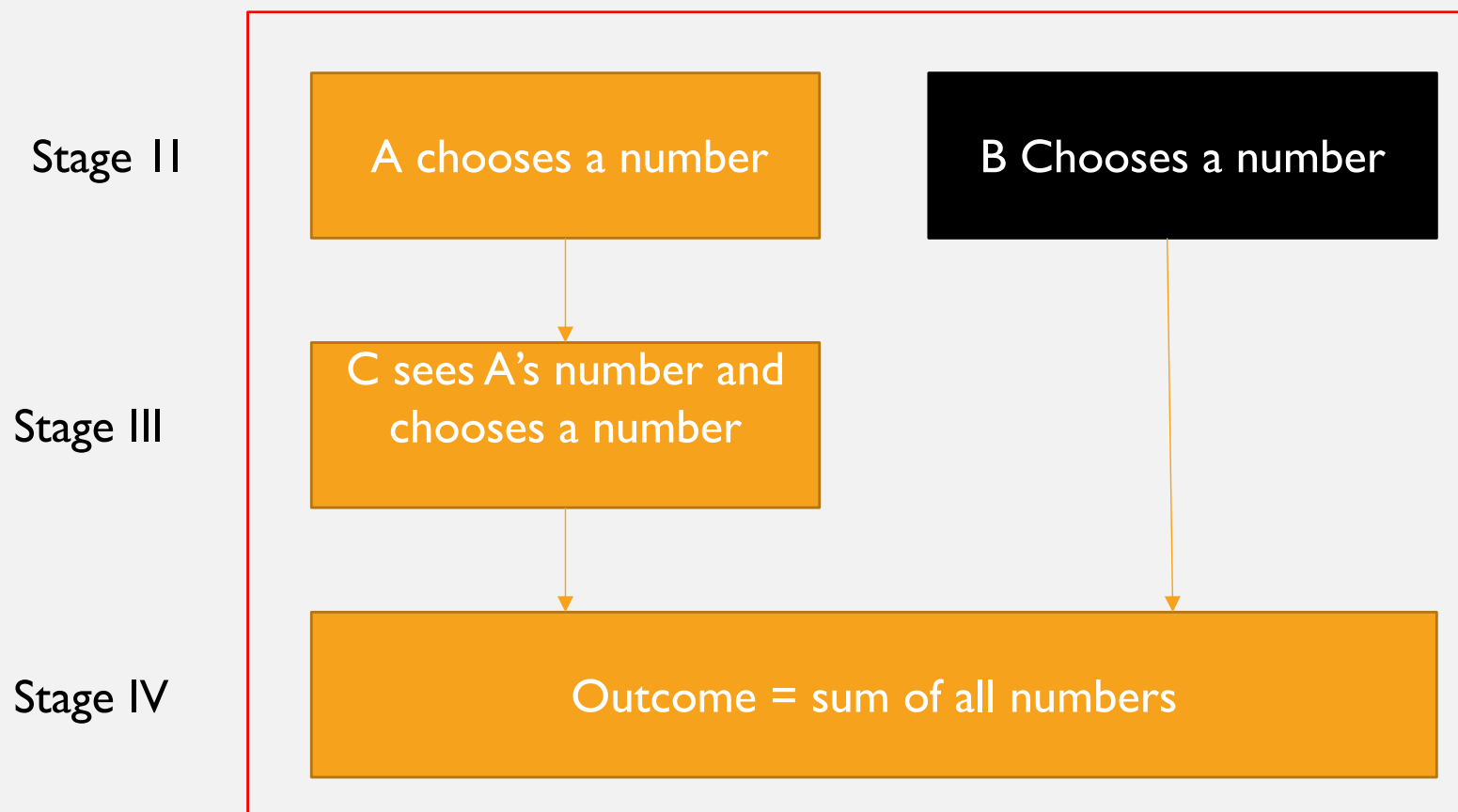
Globals table

```
RepeatTreatment = 1 or 0;
```

EXAMPLE VIII
COMPLEX MOVES

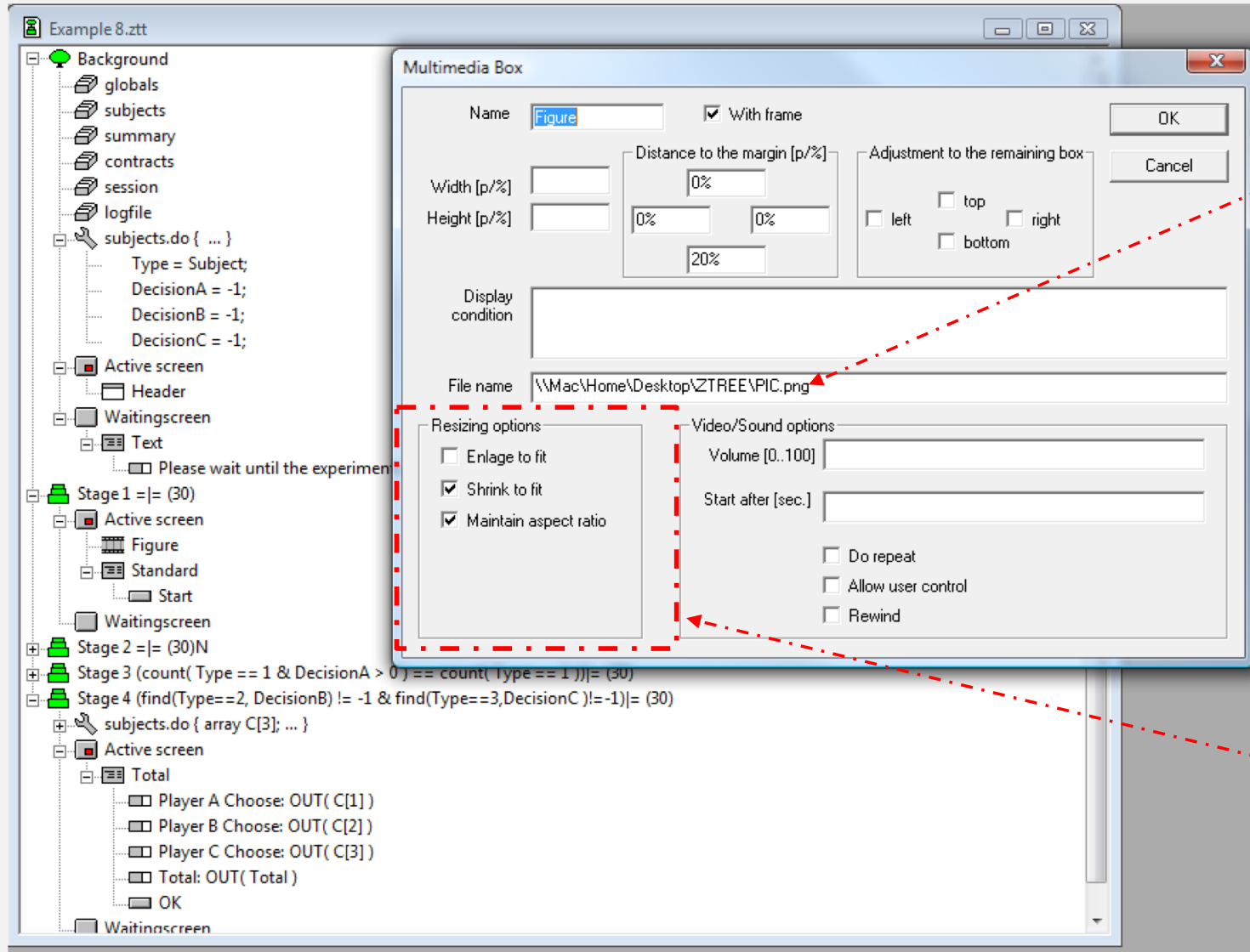
DESIGN

- Suppose that numbers are between 0-3
- Assume B's number is difficult to determine.
- We thus want C to start once A has chosen his number
- We also want to show subjects the below graph – Stage I.



STAGE I

>treatment > New Box > New Multimedia box



Path location of file

Manage distortions

STAGE I

A chooses a number

B Chooses a number

C sees A's number and
chooses a number

Outcome = sum of all numbers

Start

STAGE II

As per normal

The screenshot displays a software interface for configuring stages. On the left, a tree view shows the following structure:

- Example 8.ztt
 - Background
 - Stage 1 =| (30)
 - Stage 2 =| (30)N
 - subjects.do { Participate = if(Type==1 | Type
 - Active screen
 - Player A
 - Enter a Number: IN(DecisionA)
 - OK
 - Player B
 - Enter a Number: IN(DecisionB)
 - OK
 - Waitingscreen
 - Stage 3 (count(Type == 1 & DecisionA > 0) ==
 - Stage 4 (find(Type==2, DecisionB) != -1 & find(
 - subjects.do { array C[3]; ... }
 - Active screen
 - Total
 - Player A Choose: OUT(C[1])
 - Player B Choose: OUT(C[2])
 - Player C Choose: OUT(C[3])
 - Total: OUT(Total)
 - OK
 - Waitingscreen

On the right, a 'Stage' dialog box is open, showing the configuration for 'Stage 2':

- Name: Stage 2
- Start:
 - Wait for all
 - Start if possible
 - Start if...
- Number of subjects in Stage:
 - At most one per group in stage
- Leave stage after timeout:
 - If no input
 - Yes
 - No
- Timeout: 30

STAGE III

```
count( Type == 1 & DecisionA > 0 ) == count( Type == 1 )
```

The screenshot shows a hierarchical tree of stages and screens. The tree is as follows:

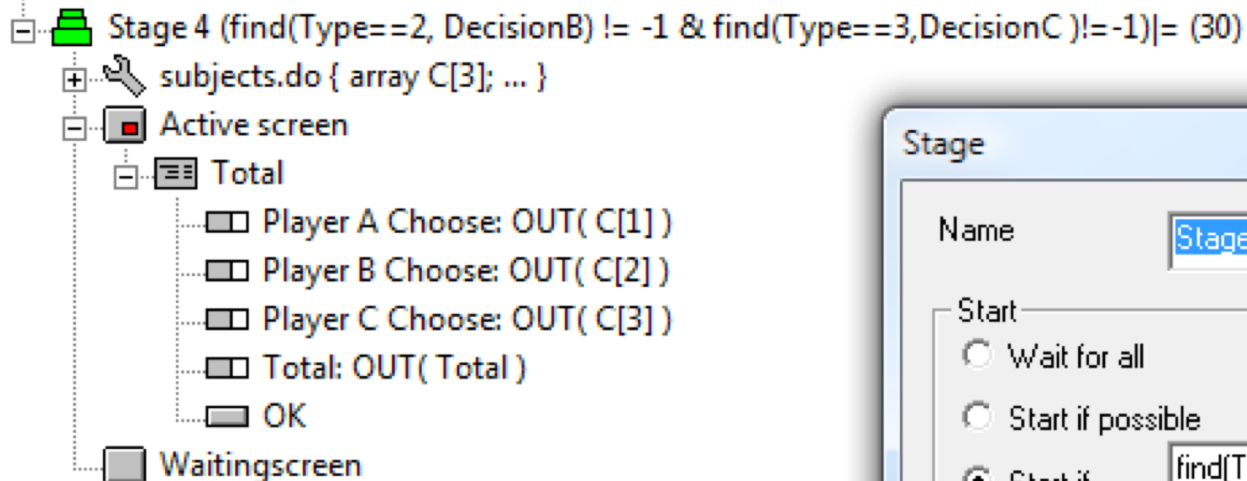
- Stage 3 (count(Type == 1 & DecisionA > 0) == count(Type == 1)) = (30)
 - subjects.do { Participate = if(Type==3, 1,0); ... }
 - Active screen
 - Player C
 - Player A Choose: OUT(DecisionA)
 - Enter a Number: IN(DecisionC)
 - OK
 - Waitingscreen
- Stage 4 (find(Type==2, DecisionB) != -1 & find(Type=...
- subjects.do { array C[3]; ... }
- Active screen
 - Total
 - Player A Choose: OUT(C[1])
 - Player B Choose: OUT(C[2])
 - Player C Choose: OUT(C[3])
 - Total: OUT(Total)
 - OK
 - Waitingscreen

The screenshot shows a 'Stage' dialog box with the following configuration options:

- Name: Stage 3
- Start:
 - Wait for all
 - Start if possible
 - Start if... count(Type == 1 & DecisionA > 0) == c
- Number of subjects in Stage:
 - At most one per group in stage
- Leave stage after timeout:
 - If no input Yes No
 - Timeout: 30

STAGE IV

```
find(Type==2, DecisionB) != -1 & find(Type==3, DecisionC) != -1
```



Stage

Name: Stage 4

Start:

- Wait for all
- Start if possible
- Start if... find(Type==2, DecisionB) != -1 & find(Ty

Number of subjects in Stage:

- At most one per group in stage

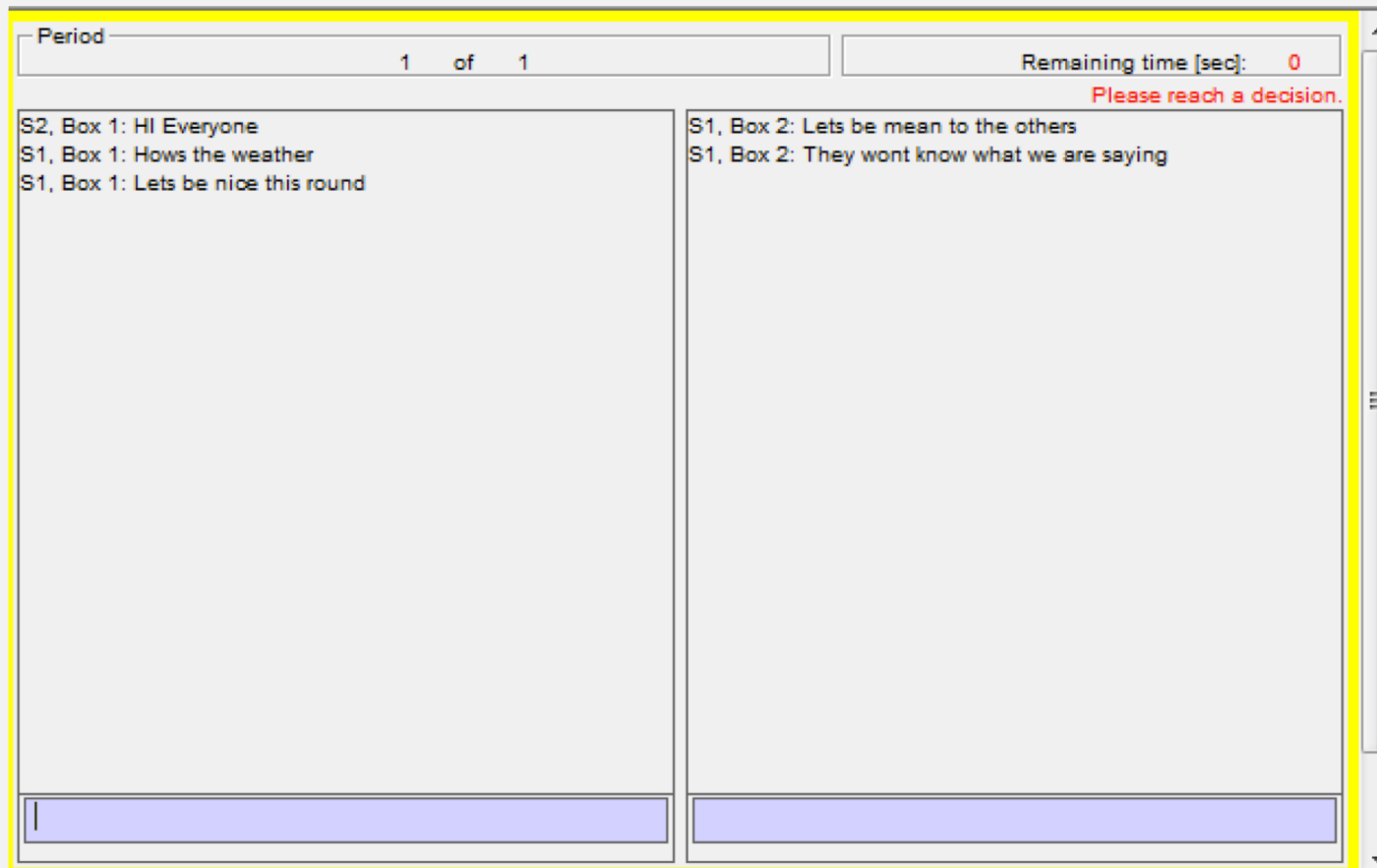
OK Cancel

EXAMPLE IX

CHAT BOX

DESIGN

- N=4 players are separated into 2 groups.
- They have two chat boxes
 - Box 1 (Left): Sends message to everyone
 - Box 2 (Right): Sends message only to same group members



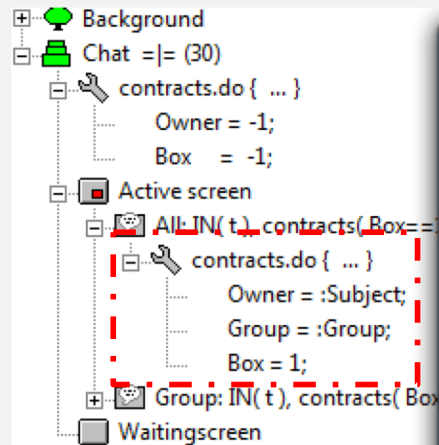
CONSIDERATIONS

- We use the contracts table.
- This is how the data looks like

Period	Owner	Box	t	Group	TimeChatC
1	2	1	"Hi Everyone "	1	22
1	1	1	"Hows the weather "	1	12
1	1	2	"Lets be mean to the others"	1	0
1	1	2	"They wont know what we are saying"	1	-12
1	1	1	"Lets be nice this round"	1	-22

CHAT STAGE

- We first program the Box I
- >treatment >new box >New Chat



Input variable is "t"

Only "t" associated with Box==1 is listed

Chat Box

Name: All With frame

Width [p/]: 50%

Height [p/]:

Distance to the margin [p/]: 0% (top, left)

Adjustment to the remaining box: left, bottom

Display condition:

Table: contracts

Input var.: t

Condition: Box==1

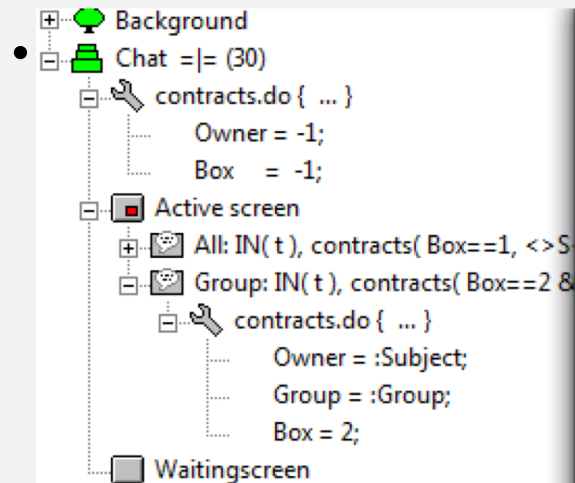
Output text: <>S<Owner|1>, Box <Box|1>: <t|-1>

Wrap text Output text centered

<>S<Owner|1>, Box <Box|1>: <t|-1>

CHAT STAGE

- Now we program the Box 2



Chat Box

Name: With frame

Width [p/%]: Distance to the margin [p/%]: Adjustment to the remaining box: left top right bottom

Height [p/%]:

Display condition:

Table:

Input var: Number of characters: Number of lines:

Condition:

Output text:

Wrap text Output text centered

OK Cancel

EXAMPLE X

**2 DIMENSIONAL
GRAPHING**

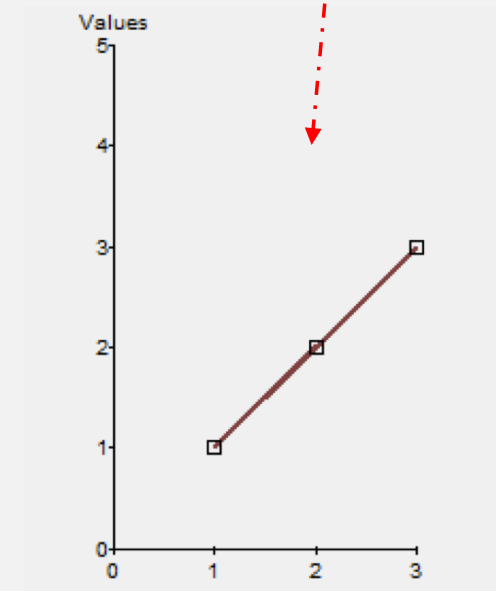
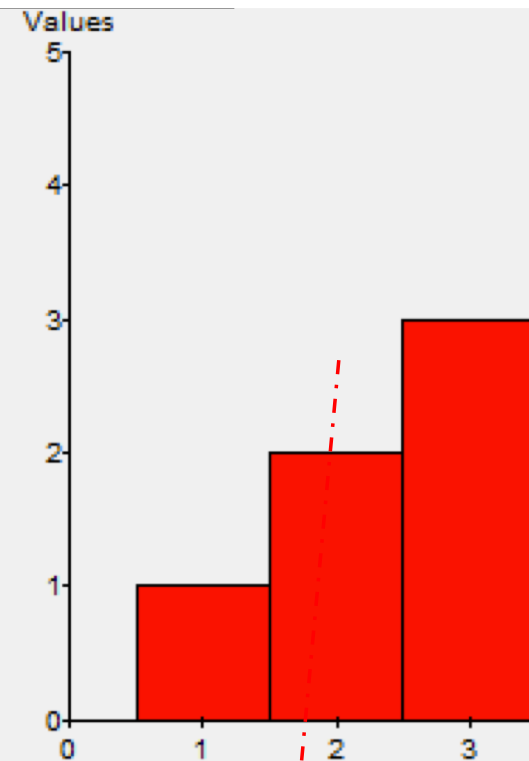
DESIGN

- Players enter 3 numbers
 - Each number between 0-5
- The 3 numbers are graphed as bars
- The 3 numbers are graphed as line chart

Input Num 1:

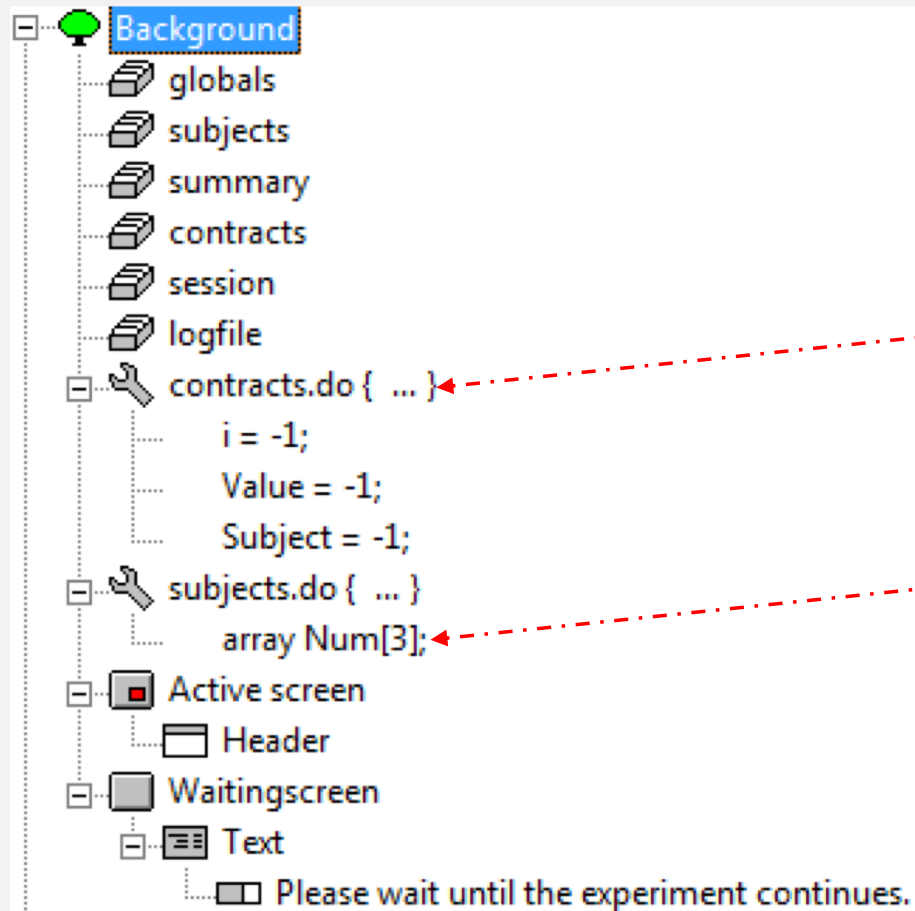
Input Num 2:

Input Num 3:



- We are going to utilise the contracts table

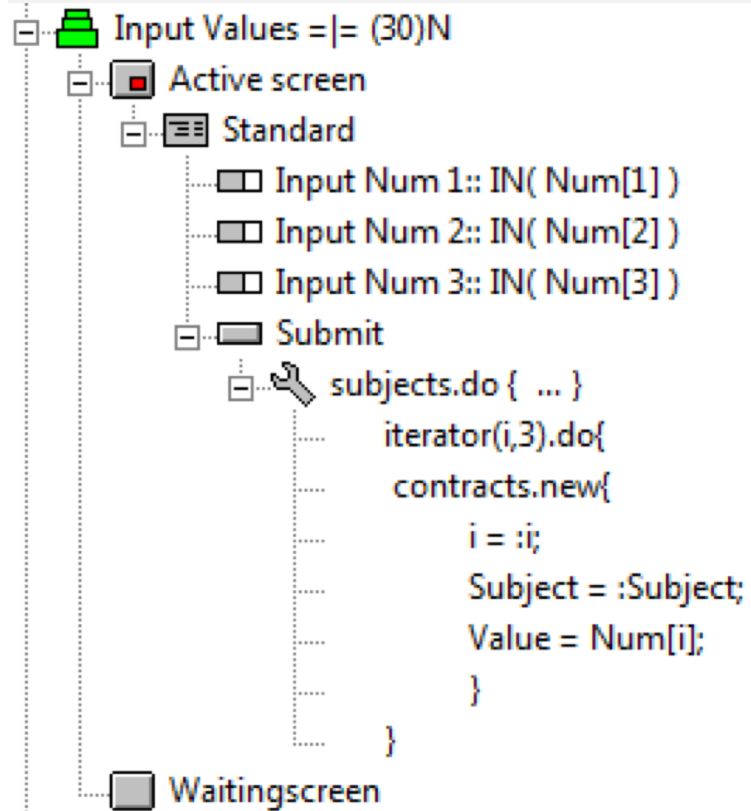
BACKGROUND



Numbers will be sorted in the contracts table

Subjects enter 3 numbers

STAGE I

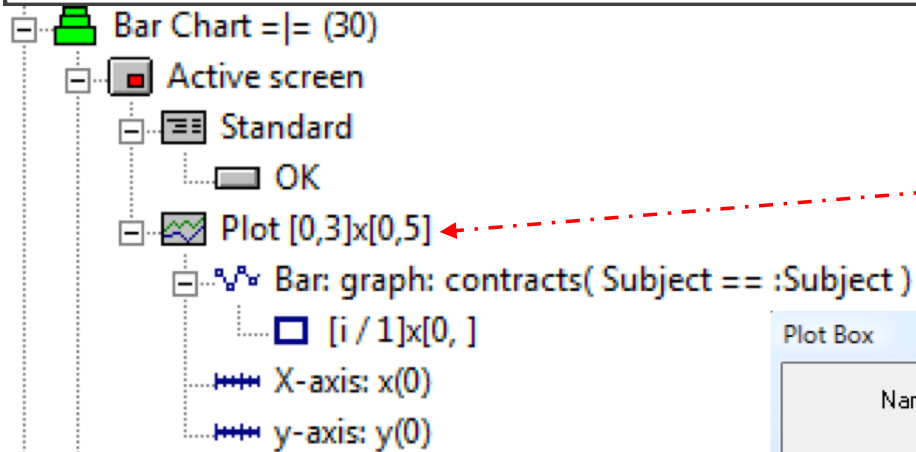


Input Num 1:

Input Num 2:

Input Num 3:

STAGE II (BAR CHART)



>treatment >New Box >New Plot box

Plot Box

Name: with Frame

Width [p/%)

Height [p/%)

Distance to the margin [p/%)

Adjustment to the remaining box

left top right

bottom

Display condition

Horizontal margin Vertical margin

Fill color

Maintain aspect ratio

x-axis

categorical linear logarithmic

left right

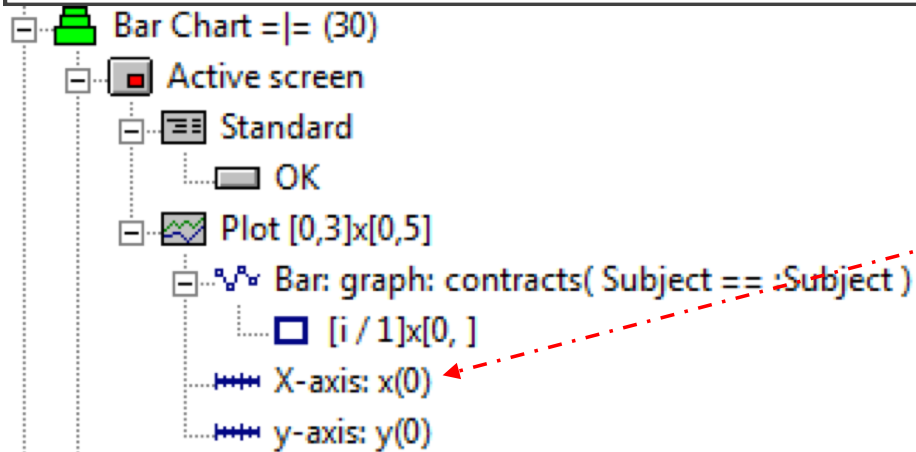
y-axis

categorical linear logarithmic

top bottom

Move pointer to

STAGE II (BAR CHART) DEFINE THE AXIS

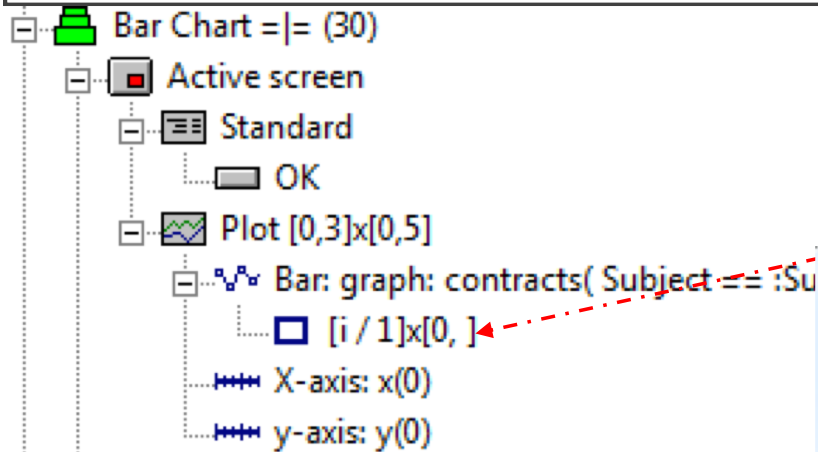


>treatment >Graphics >New Axis

Plot Axis

Name	X-axis	OK
	<input checked="" type="radio"/> x-axis <input type="radio"/> y-axis	Cancel
position	0	
from	0	
to	3	
tick distance	1	
grid from		
to		
increment		
data label distance	1	
layout	1	
caption	Num1 - Num3	
line color	rgb(0.00,0.00,0.00)	
line width	1	
Display condition		

STAGE II (BAR CHART) CONNECT DATA TO PLOT



>treatment >Graphics >New Rect

Plot Rect

Name

OK

Cancel

x width

y height

Position

line color

line width

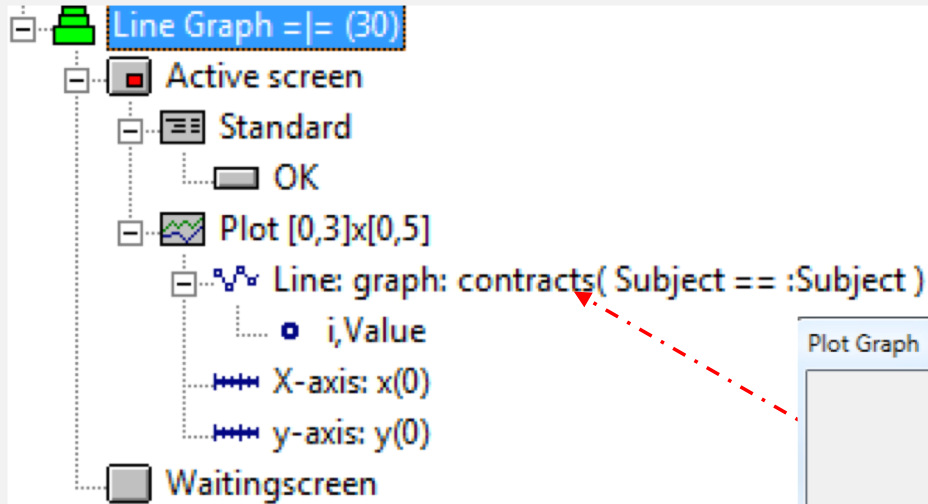
fill color

Picture file name

resize to fit maintain aspect ratio centered

Display condition

STAGE III (LINE) SETUP



Plot Graph

Name:

Table:

Condition:

Sorting:

Connection:

line color:

line width:

fill color:

Fill to x-axis

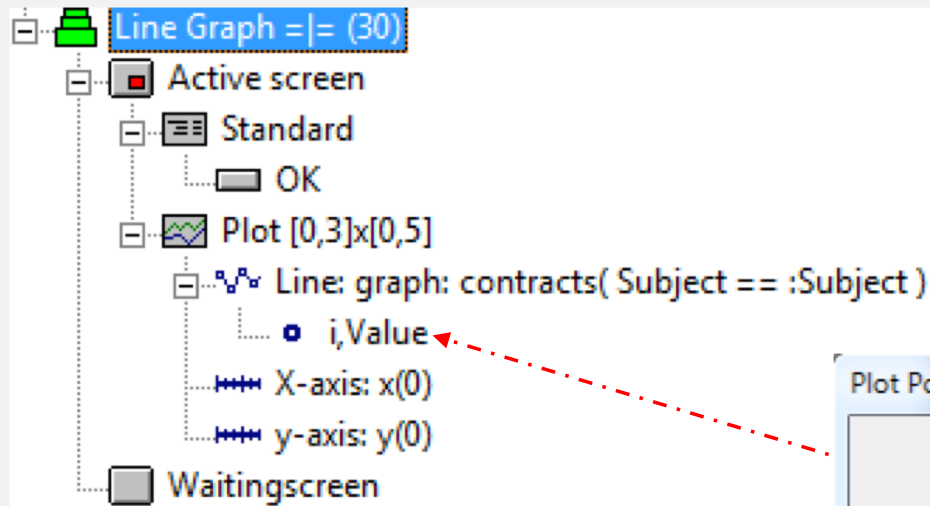
Connect to previous point

Display condition:

OK

Cancel

STAGE III (LINE) CONNECT DATA TO PLOT



Plot Point

Name	<input type="text"/>	OK
x	<input type="text"/>	Cancel
y	Value	
	<input type="checkbox"/> is a star (not polygon)	
Size	5	
Num vertices	4	
Start at (angle from x)	45	
Line color	rgb(0.00,0.00,0.00)	
Line width	1	
Fill color	<input type="text"/>	
Display condition	<input type="text"/>	

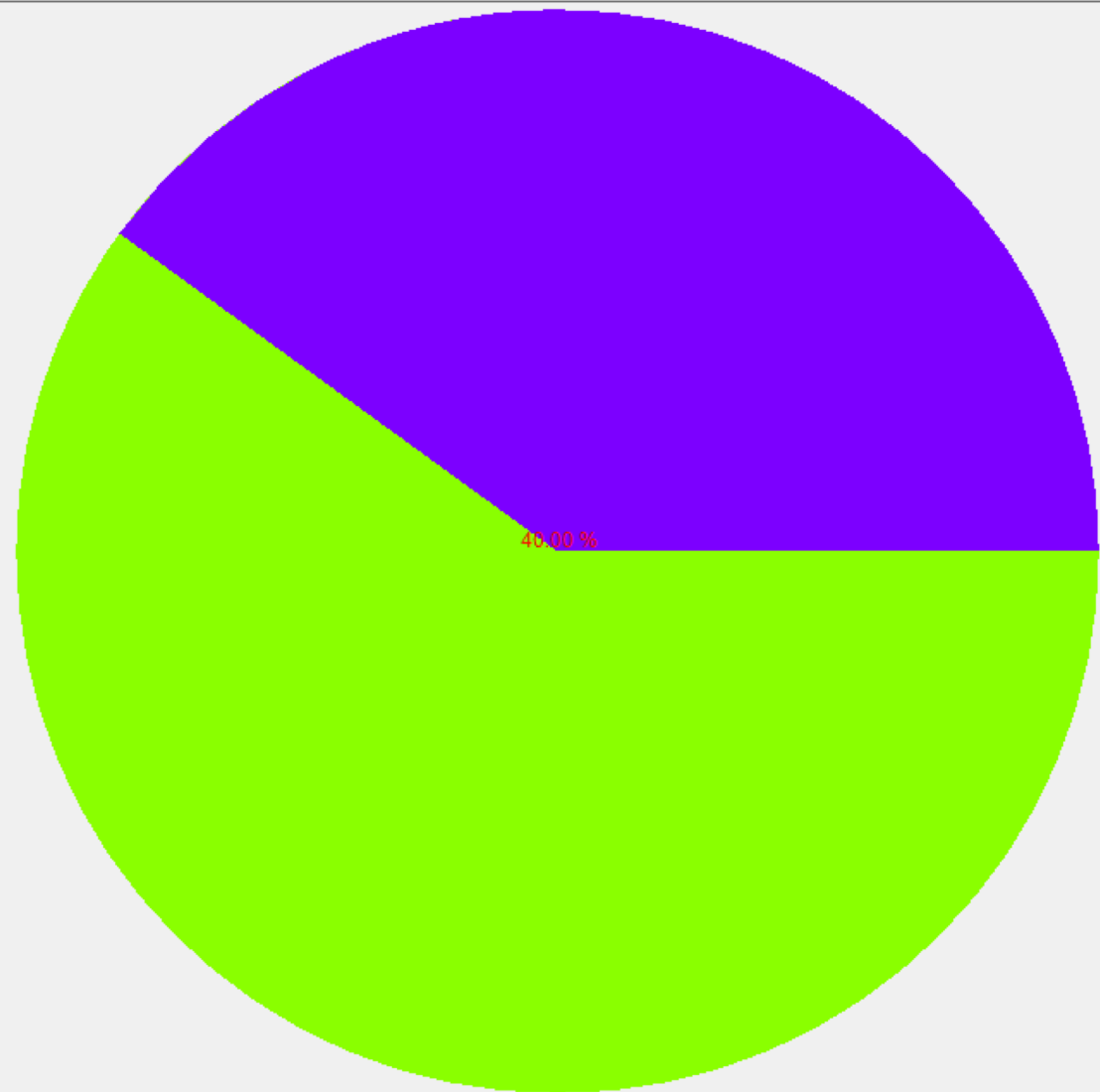
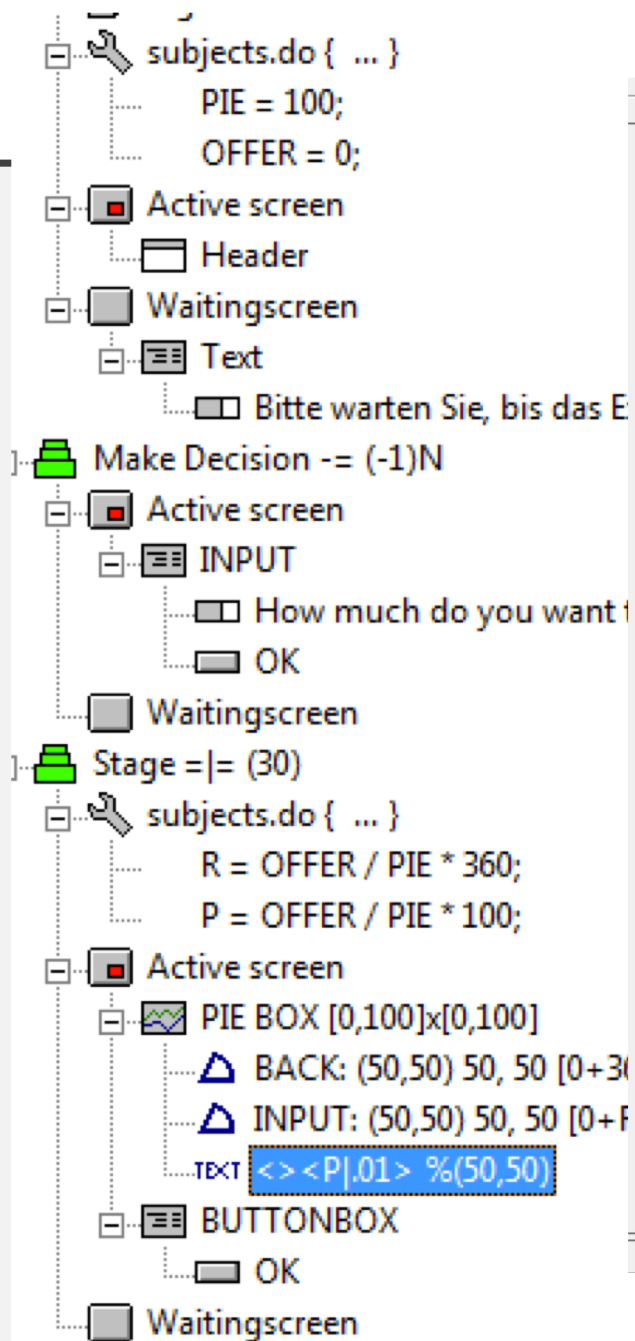
EXAMPLE XI

**GRAPHING PIE
CHARTS**

DESIGN

- There is a PIE of money (e.g., \$100)
- Player chooses how much to offer to the Other (between 0 and 100)
- Player sees the offer in a
 - Pie Chart
 - % is plot

IGN



EXERCISE IV
SSW MARKETS

DESIGN

- $N > 2$ Traders each endowed
 - 6 assets
 - 1000 cash
- Trade takes place over 3 periods (inventory are carry forwarded at each period)
- Assets pay dividend 0, 20, 40 or 60 with equal probabilities
 - Realised only at the end of the period
- CDA market trade where plot are prices is presented to subjects
 - X-axis time
 - Y-axis transacted price