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

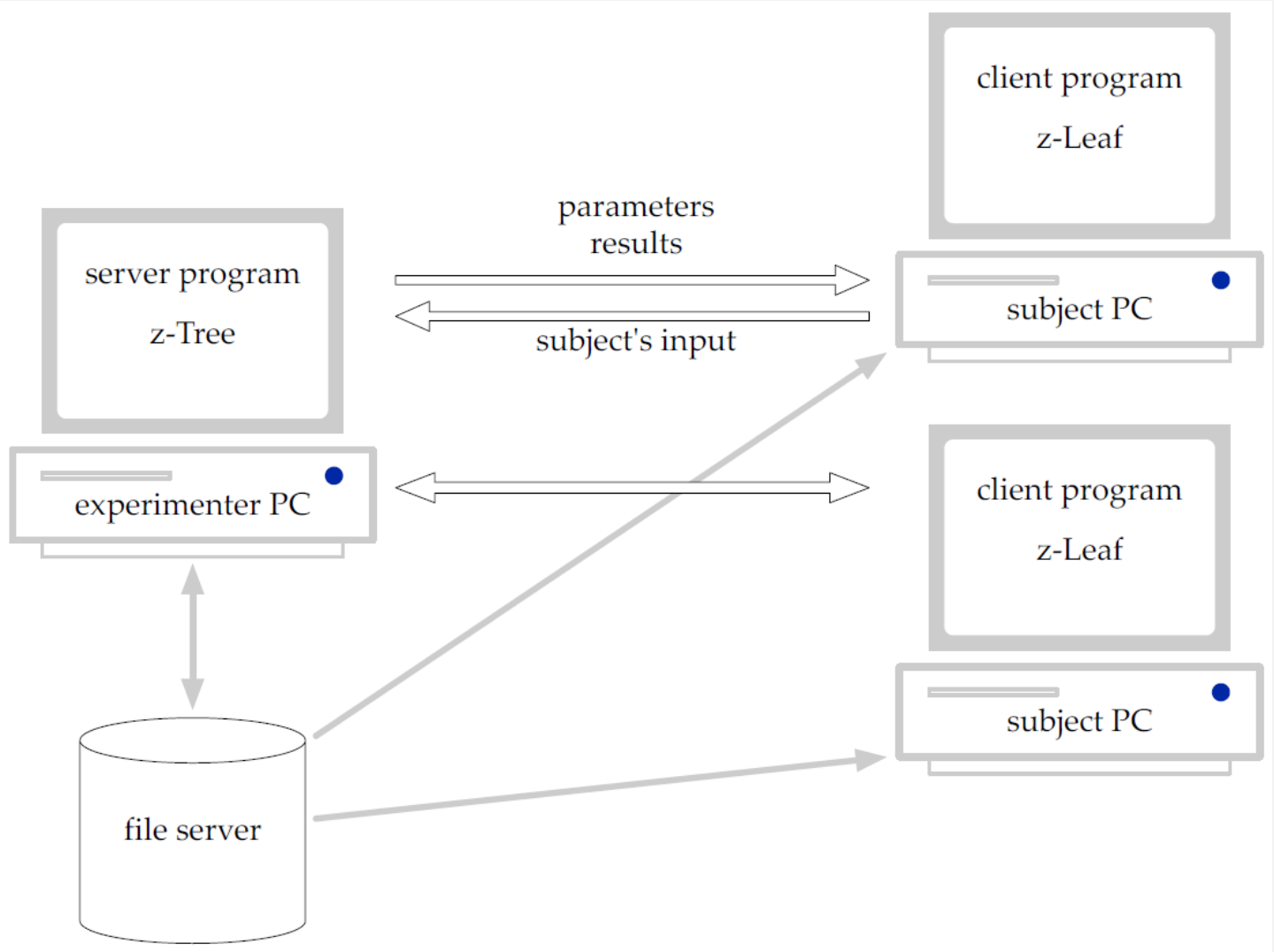
- **Creating multiple leafs on a screen**
- **Example III: 2x2 Normal form game**
  - Laying out Grid matrix
  - Random round payment
- **Example IV: Search Lottery**
  - Array programming and complex loops
  - Programming a Survey
- **Class Exercise II: Jackpot machine (A fair jackpot)**
- **Example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

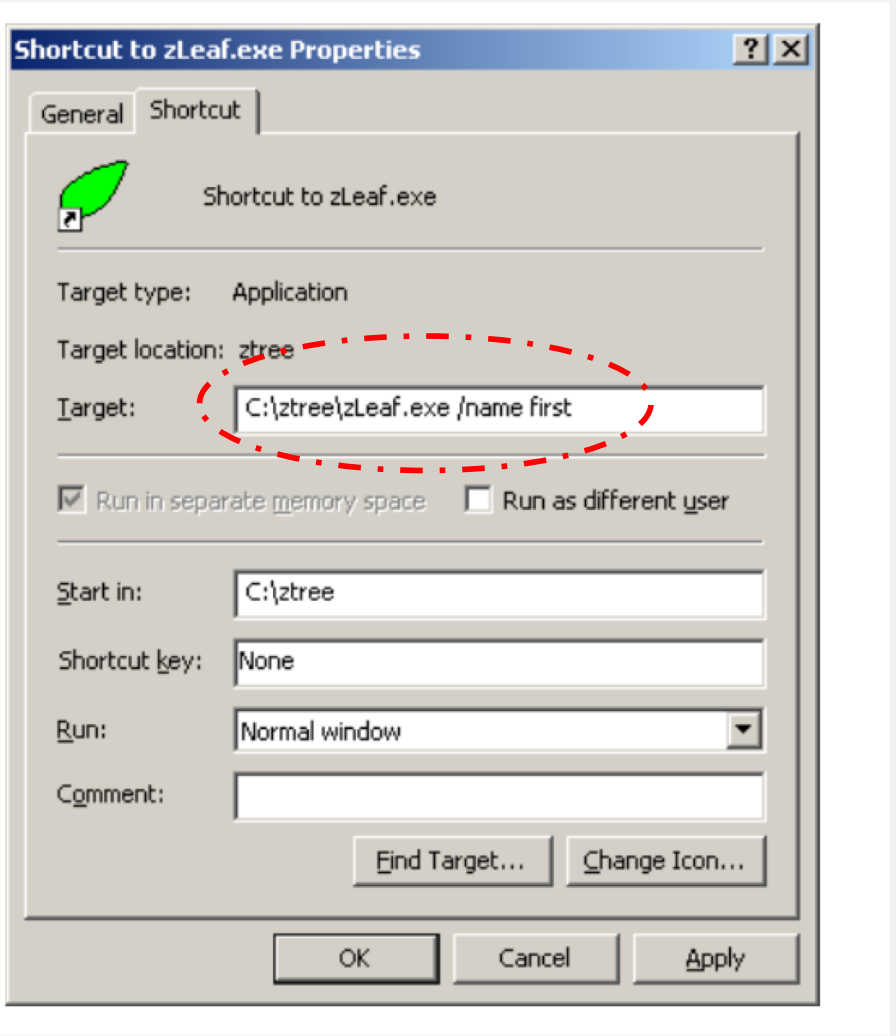


INSTALLING "LEAFS"

- 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 1

THE PUBIC GOODS GAME

EXAMPLE: PUBLIC GOODS GAME

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

globals table

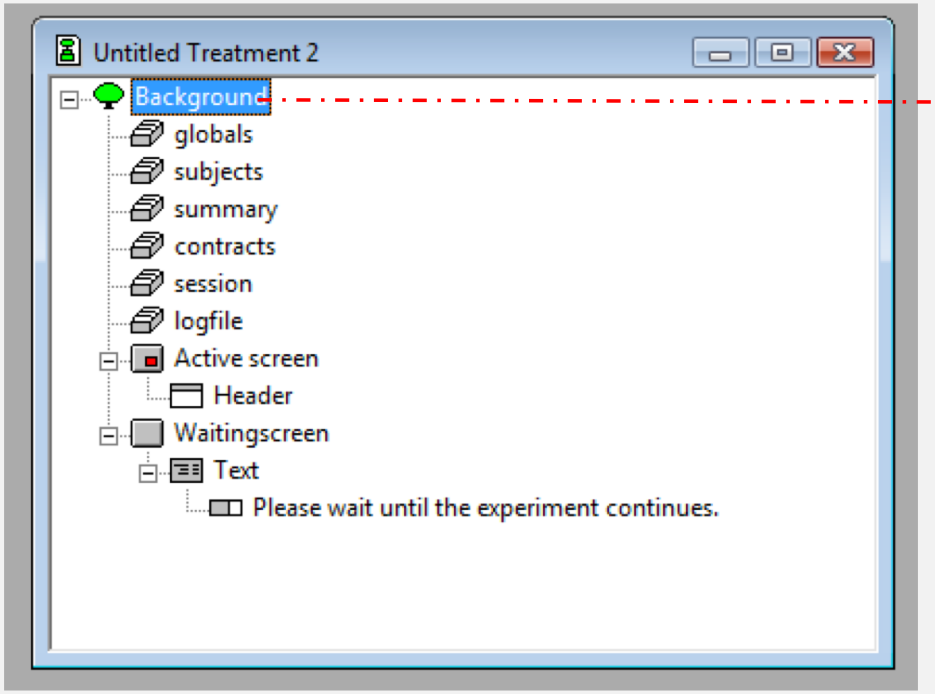
Period	NumPeriods	RepeatTreatment	M	E	N
2	5	0	1.2	10	4

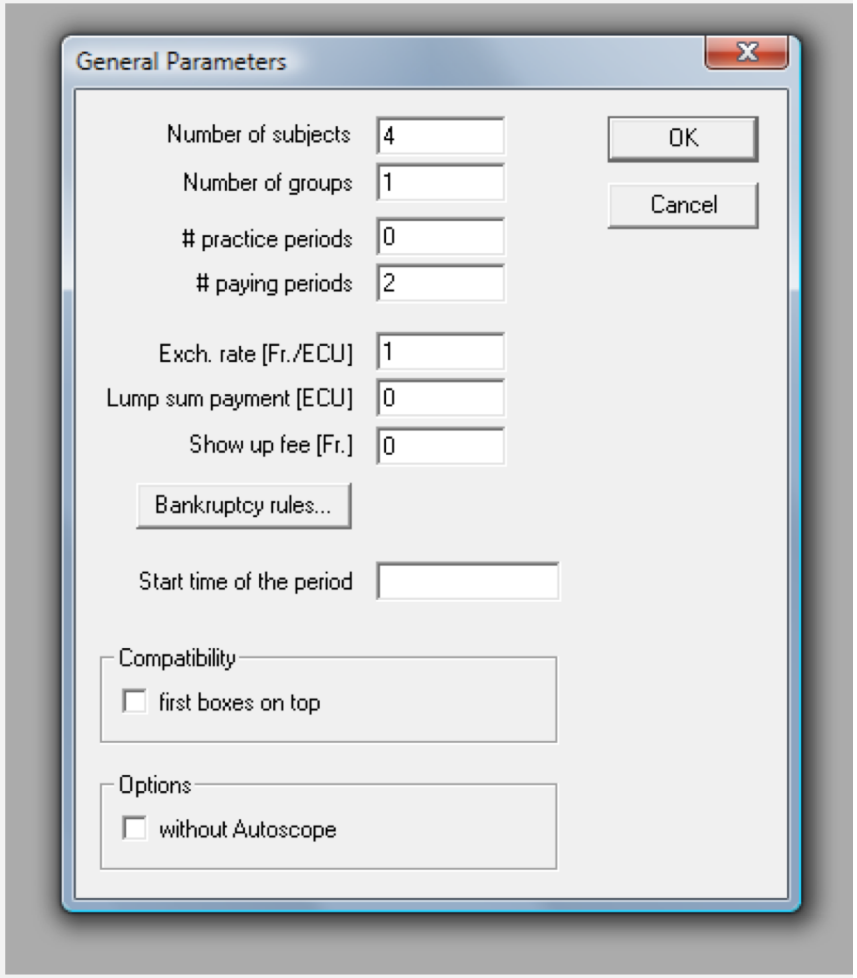
subjects table

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

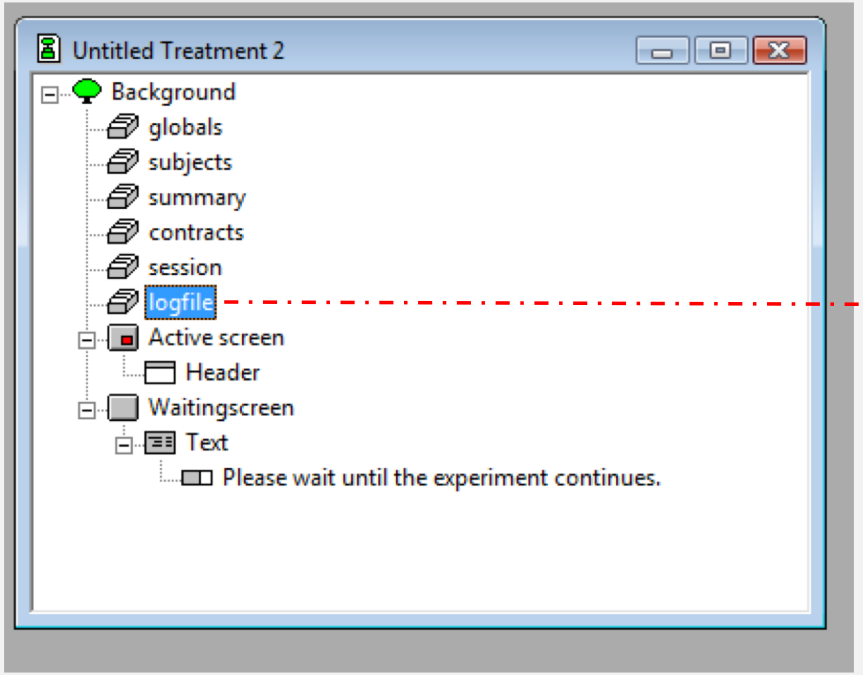


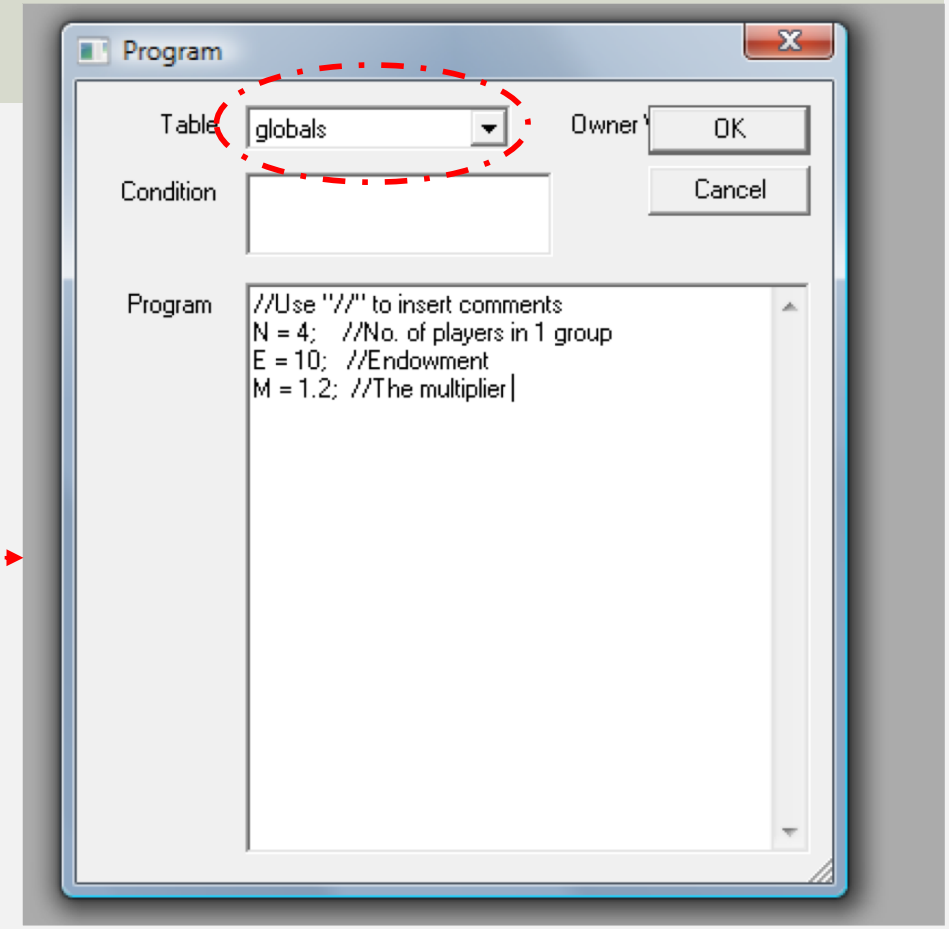


DEFINE INITIAL VALUES (globals table)

> Treatment > New program

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

E = 10; // endowment

M = 1.2; // multiplier
```

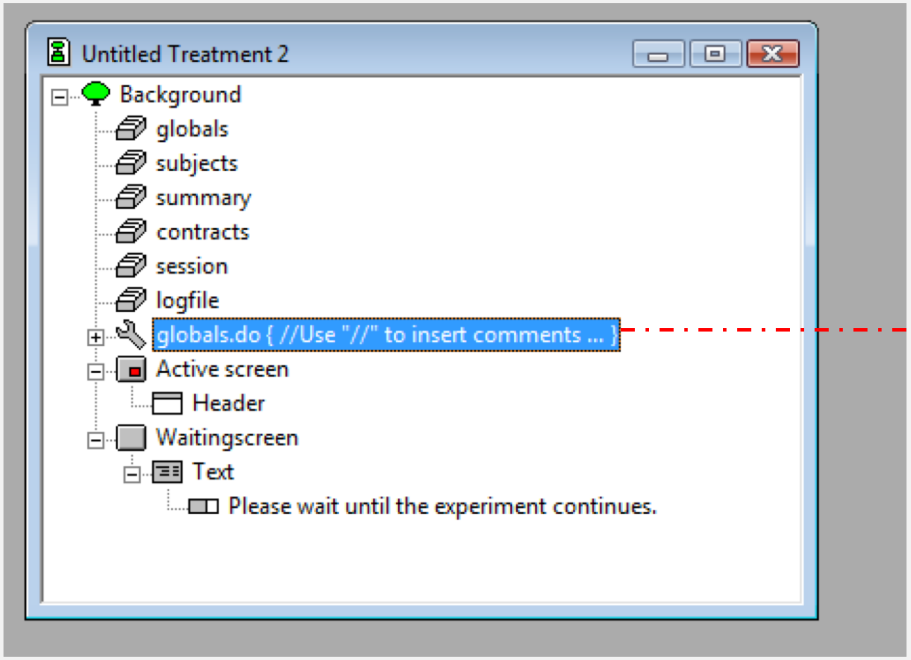


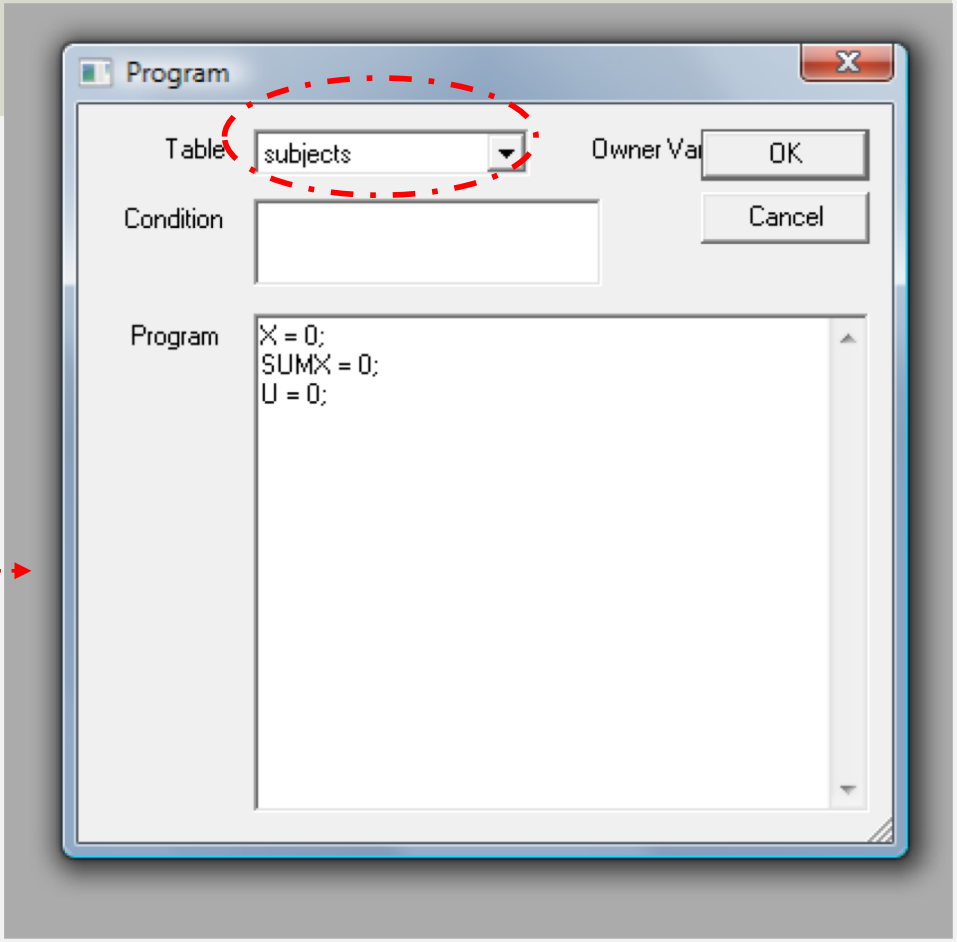


DEFINE INITIAL VALUES (subjects table)

> Treatment > New program

```
X = 0; //define variable

SUMX = 0;

U = 0;
```



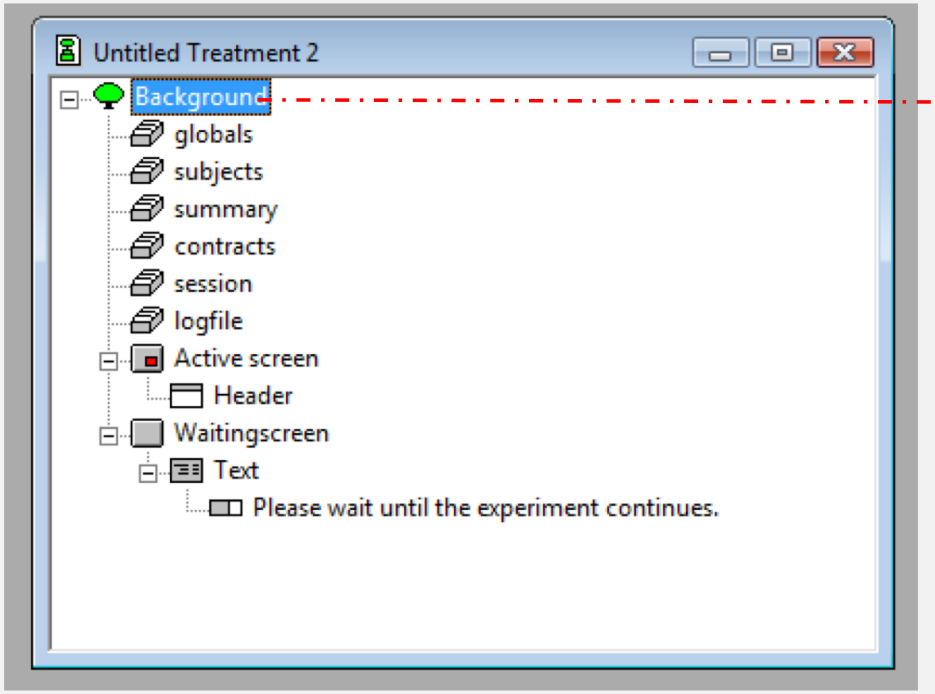


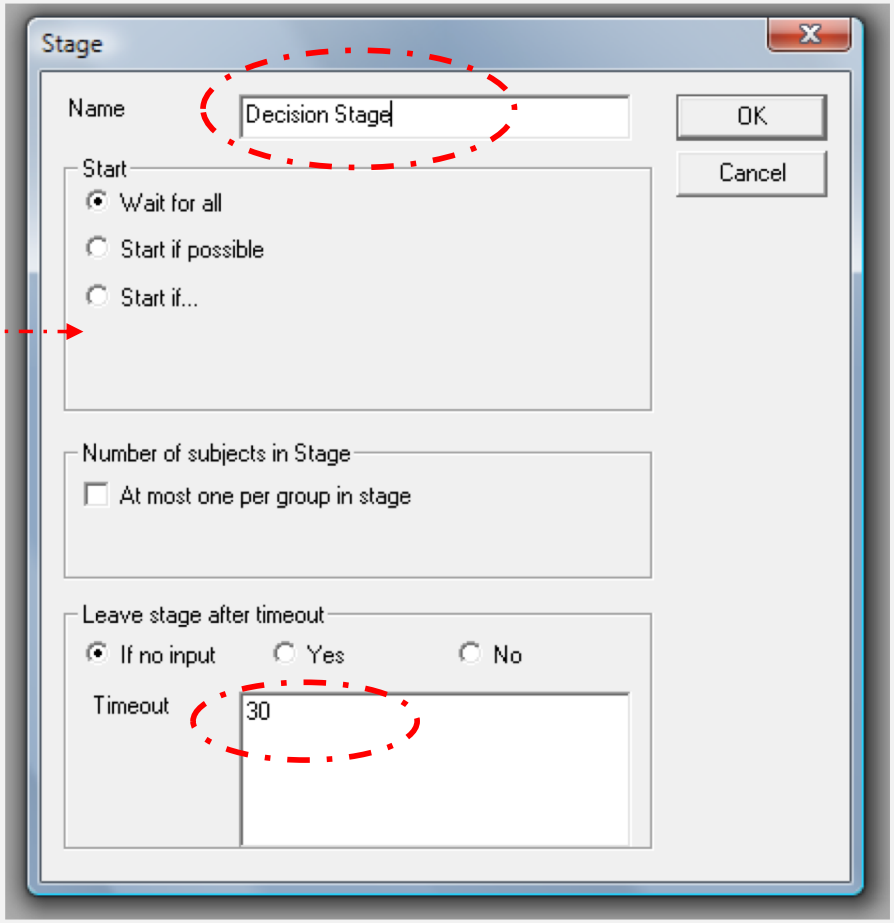
CREATE NEW STAGES

- Select the most recent stage

  > Treatment > New stage

- You can add as many stages as necessary
- Create **"Decision stage"**
- Create **"Results stage"**

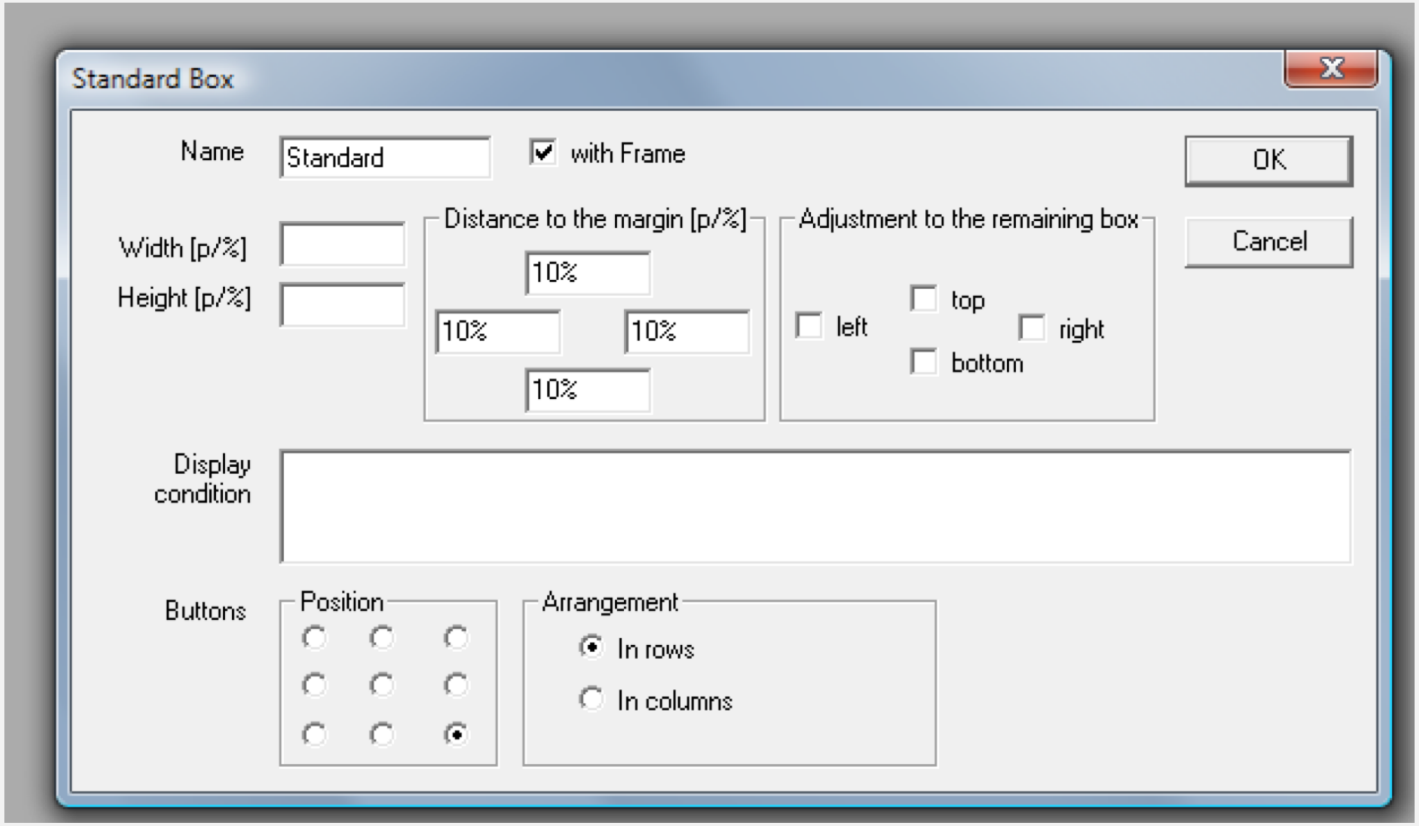




DECISION STAGE
(create new box)

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

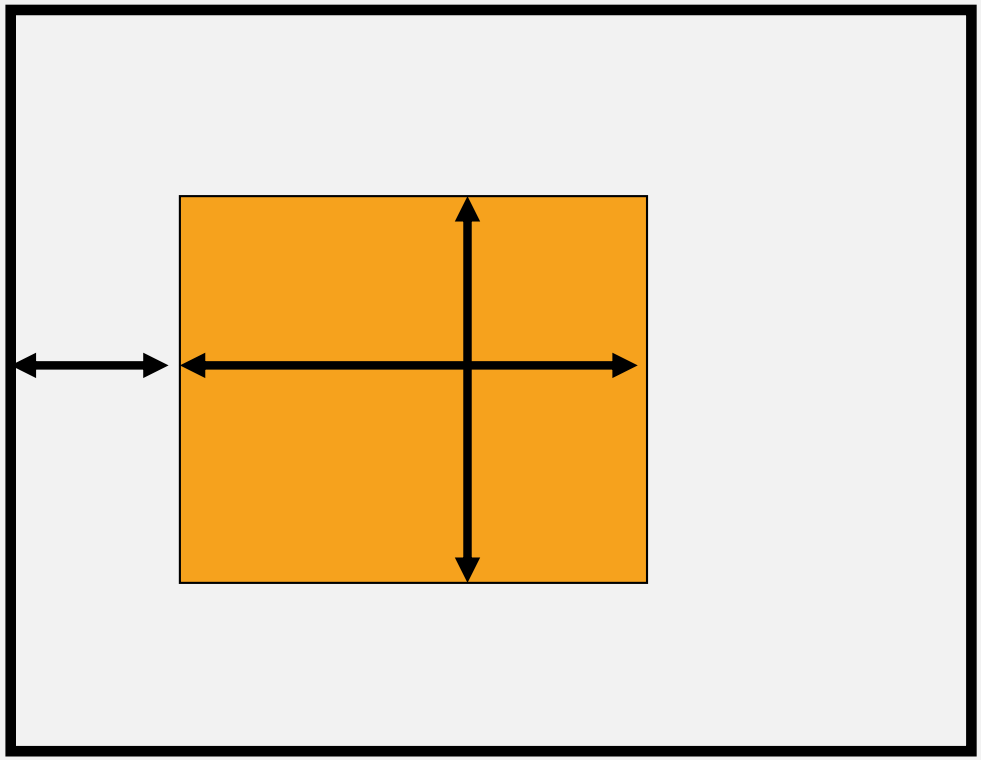
Active screen > Treatment > New Box > Standard Box



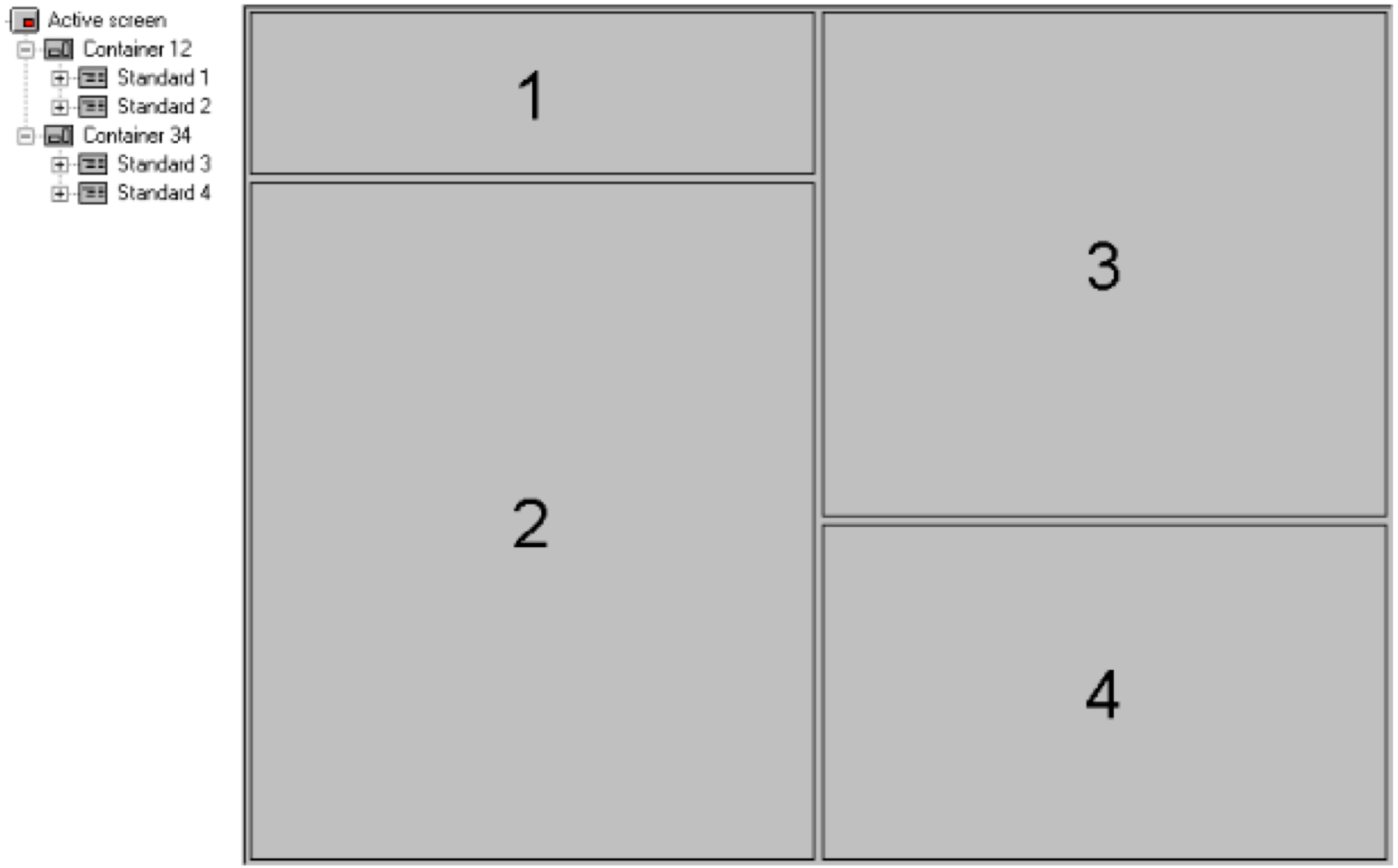
GENERAL
(box layout)

- **Box can be position one after another**
- **Types of boxes:**

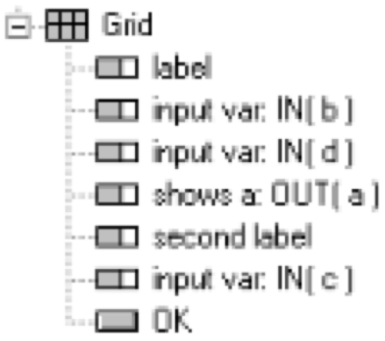
1. Standard box
2. Grid box
3. Header box
4. Help box
5. History box
6. Container box
7. Calculator button box

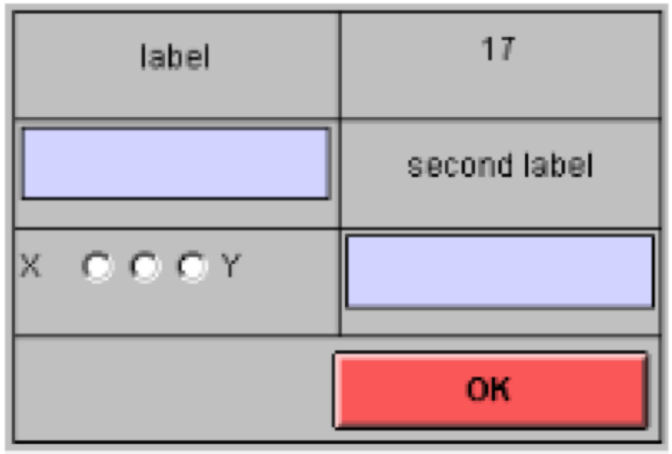


GENERAL
(box layout – Container Box)

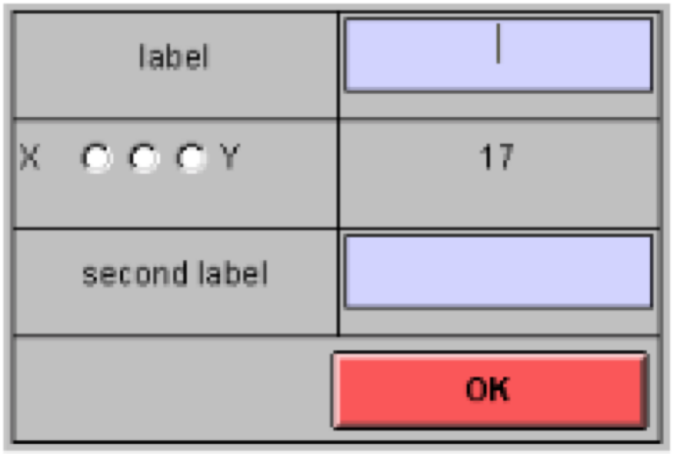


GENERAL
(box layout – Grid Box)





column by column

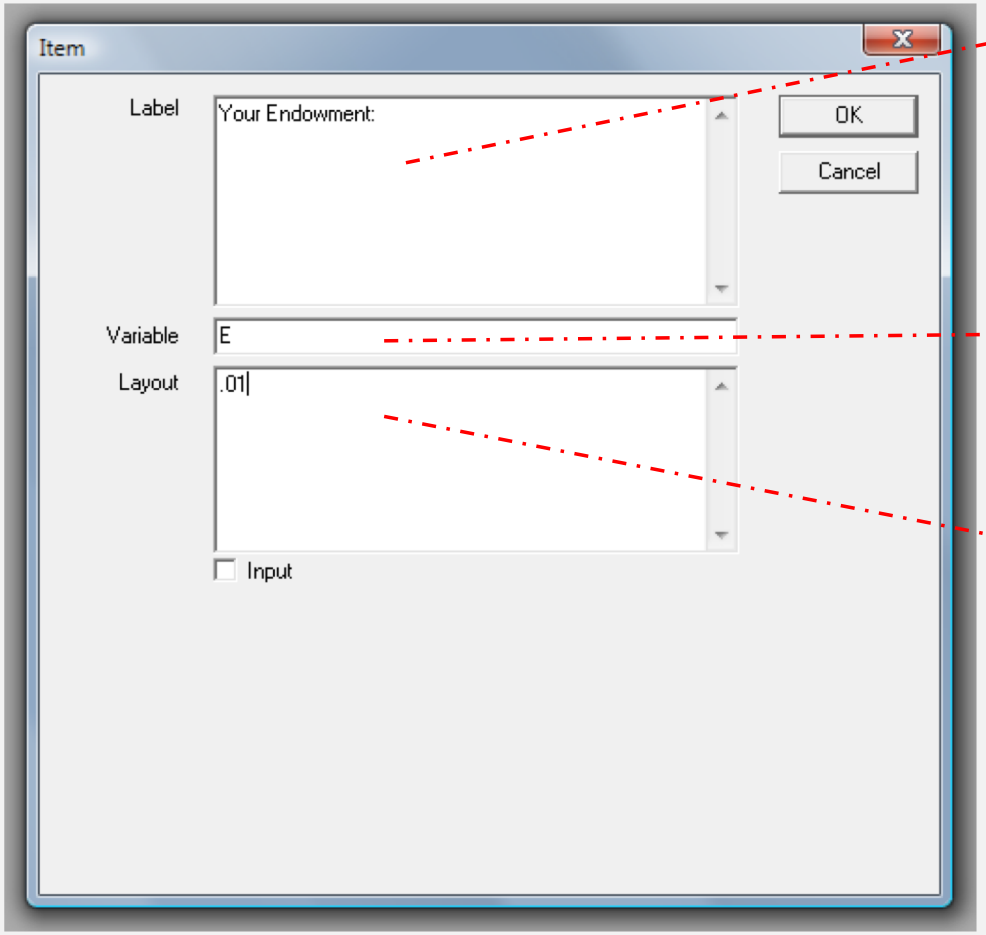


row by row

DECISION STAGE
(Putting items in Box)

- **Creating a Output variable**

Standard box > Treatment > New item



Text that subjects see

"Your Endowment"

The output variable "E"

How the output variable is presented:

"1" : no decimal places

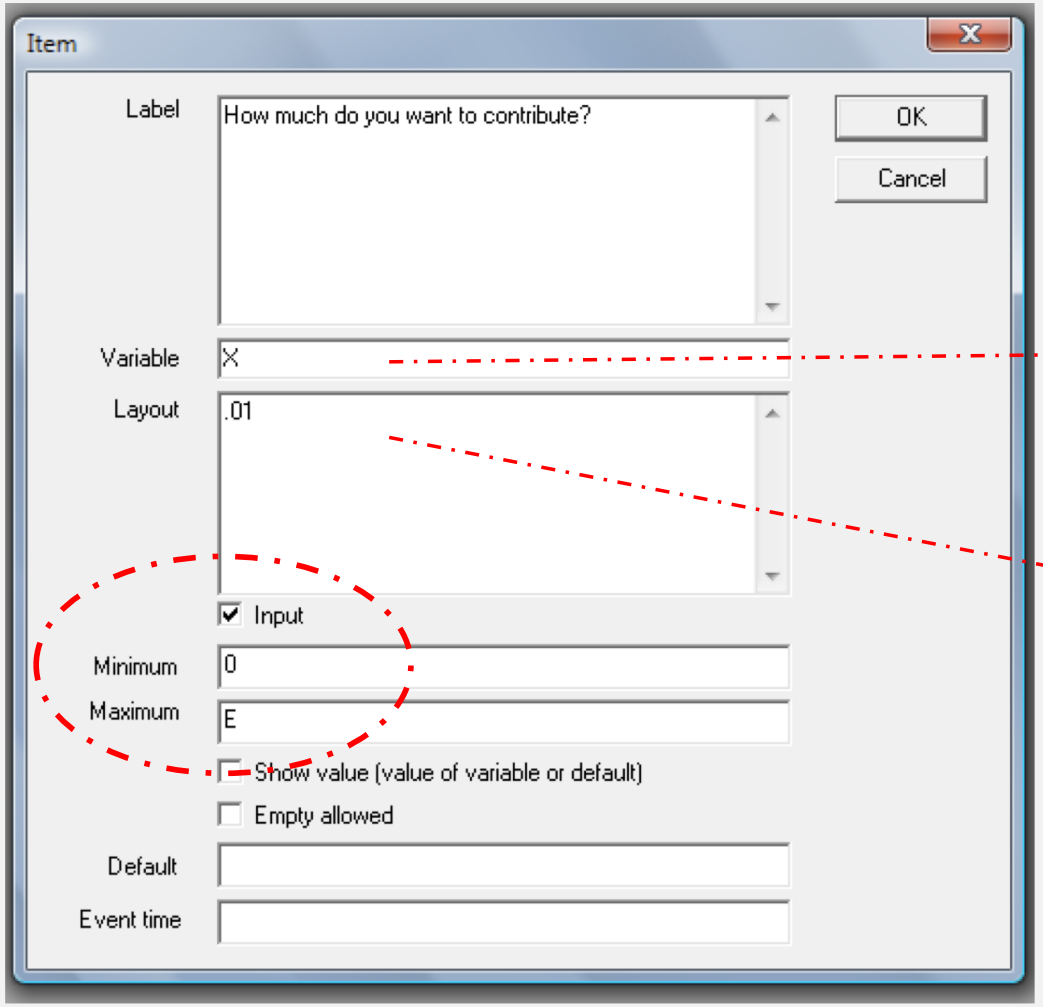
".1" : 1 decimal place

".01" : 2 decimal places

DECISION STAGE
(Putting items in Box)

- **Creating a input variable**

Standard box > Treatment > New item



The input variable "X"

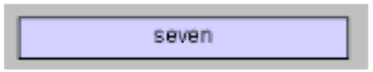
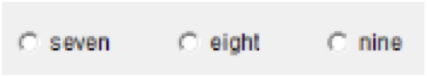
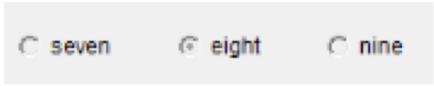
How subjects input the data

.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

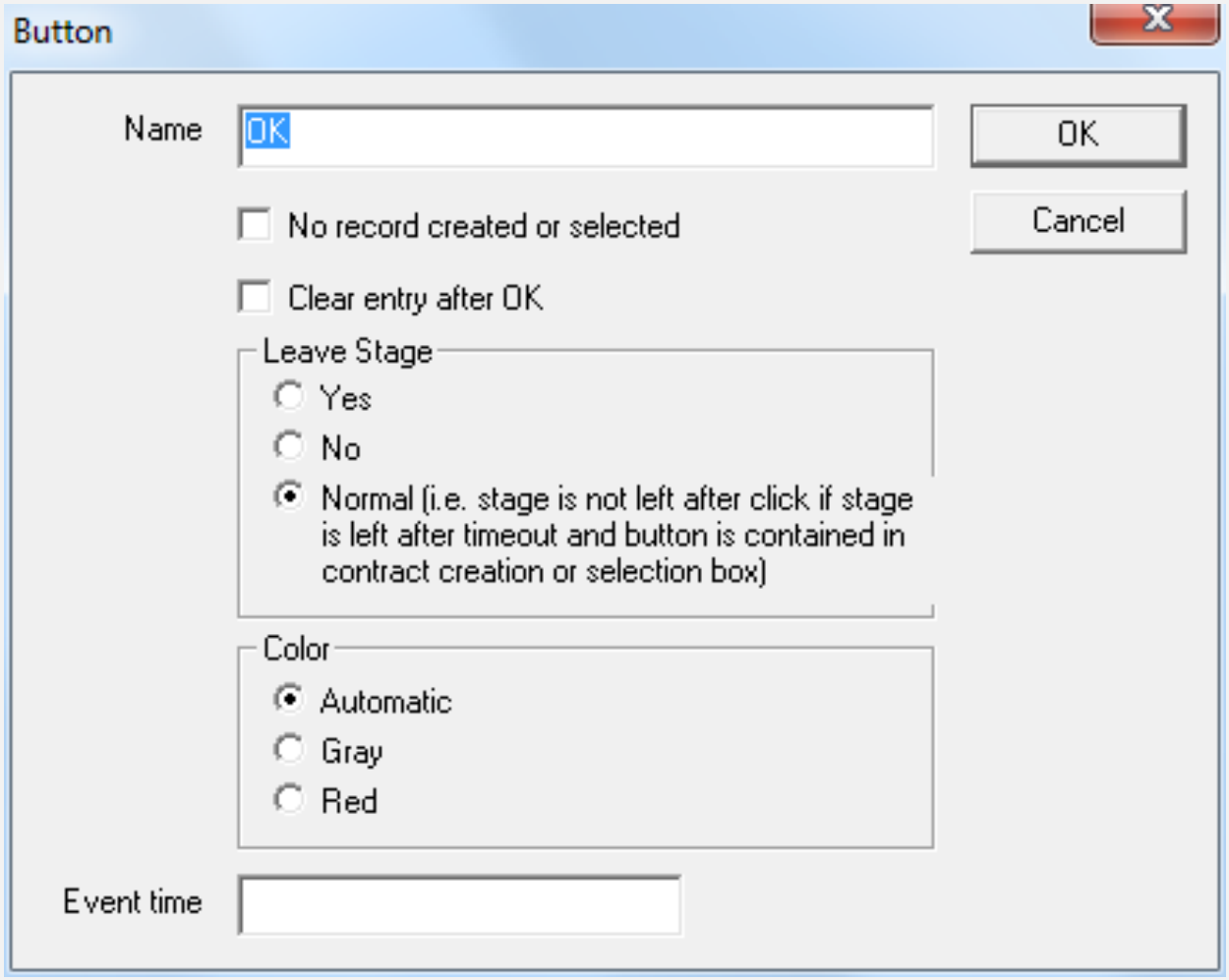
GENERAL
(input / output formats)

Layout	Input variable	Output variable
2		
!text: 7 = "seven"; 8 = "eight"; 9 = "nine";		
!radio: 1 = "86.8"; 24 = "102.8";		
!radioline: 0="zero";5="five"; 6;		
!radiosequence: 7="seven";8="eight";9="nine";		
!slider: 0 ="A"; 100= "B"; 101;		
!scrollbar: 0="L";100= "R";101;		
!checkbox:1="check me";		
!button: 1 = "accept"; 0 = "reject";		
!string		
20		

DECISION STAGE
(Putting items in Box)

- **Creating a button**

Standard box > Treatment > New Button

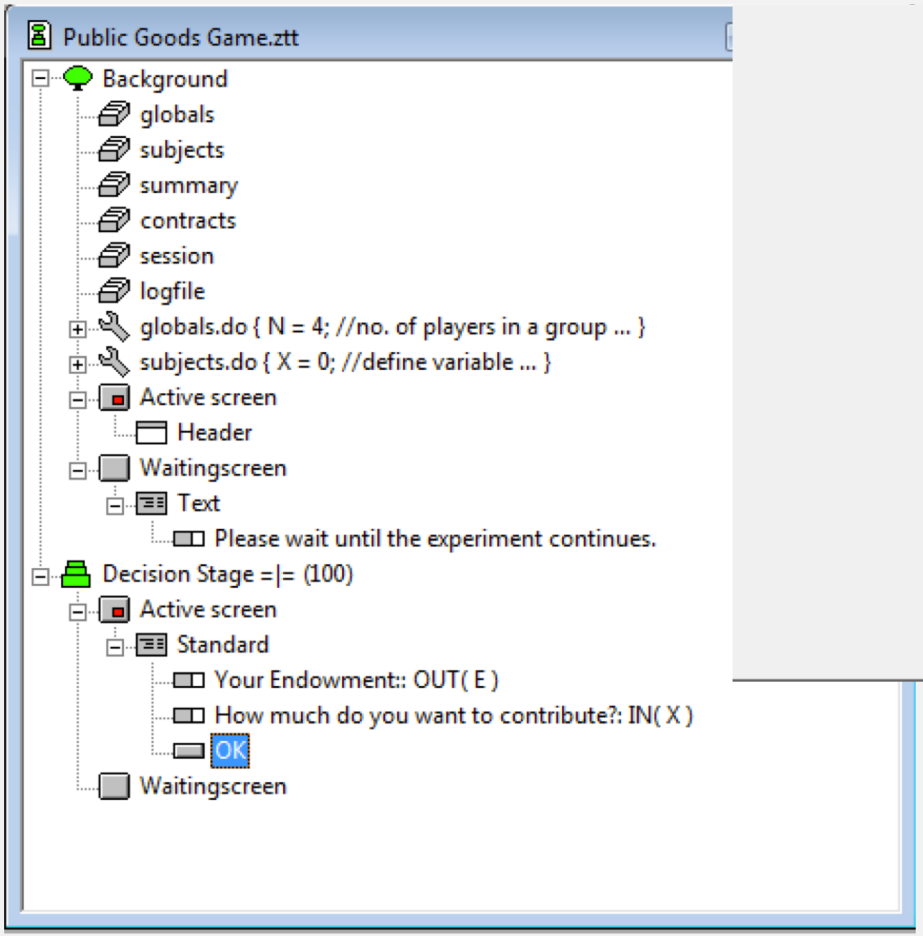


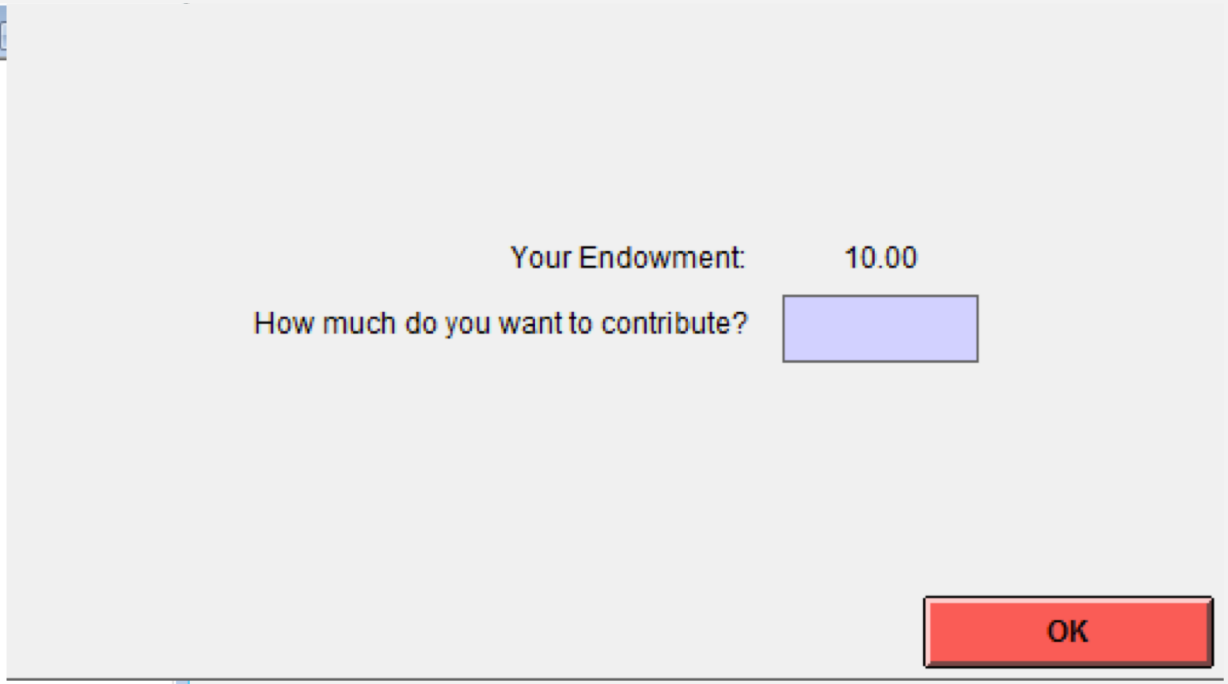
A button tells ztree to collect the data and let the subject leave the stage.

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

DECISION STAGE
(trial the Decision Stage)

run > start treatment

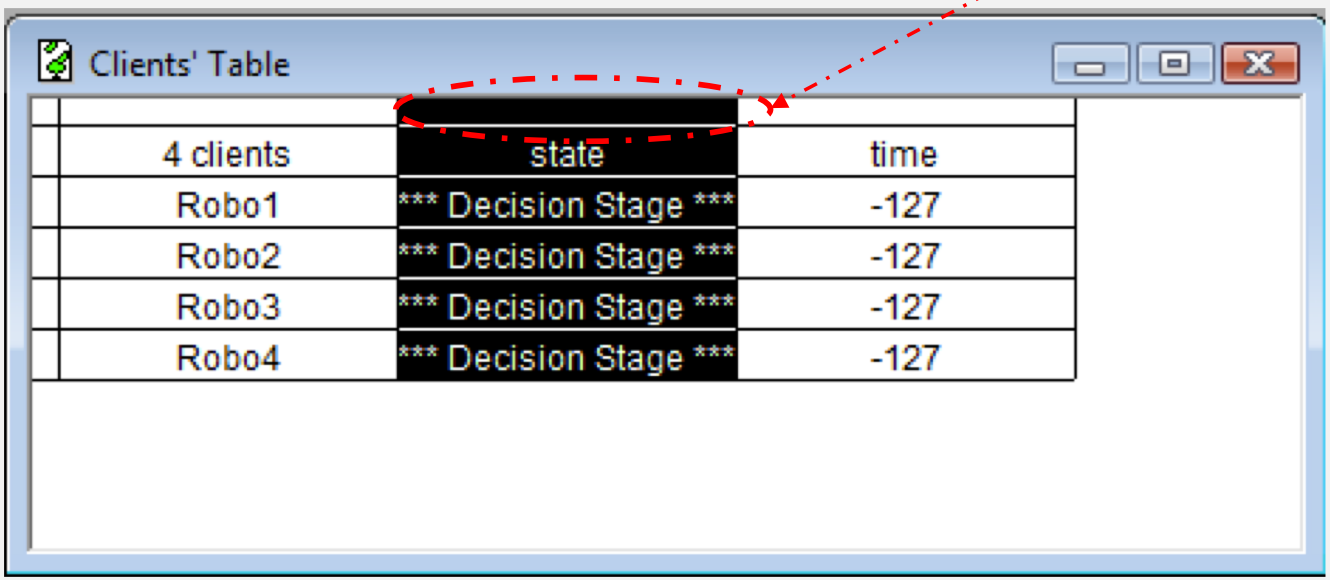




GENERAL
(How to ”force subjects to leave a stage”?)

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

Double click here



RESULTS STAGE
(collect the data from other subjects)

Results Stage> Treatment > New program

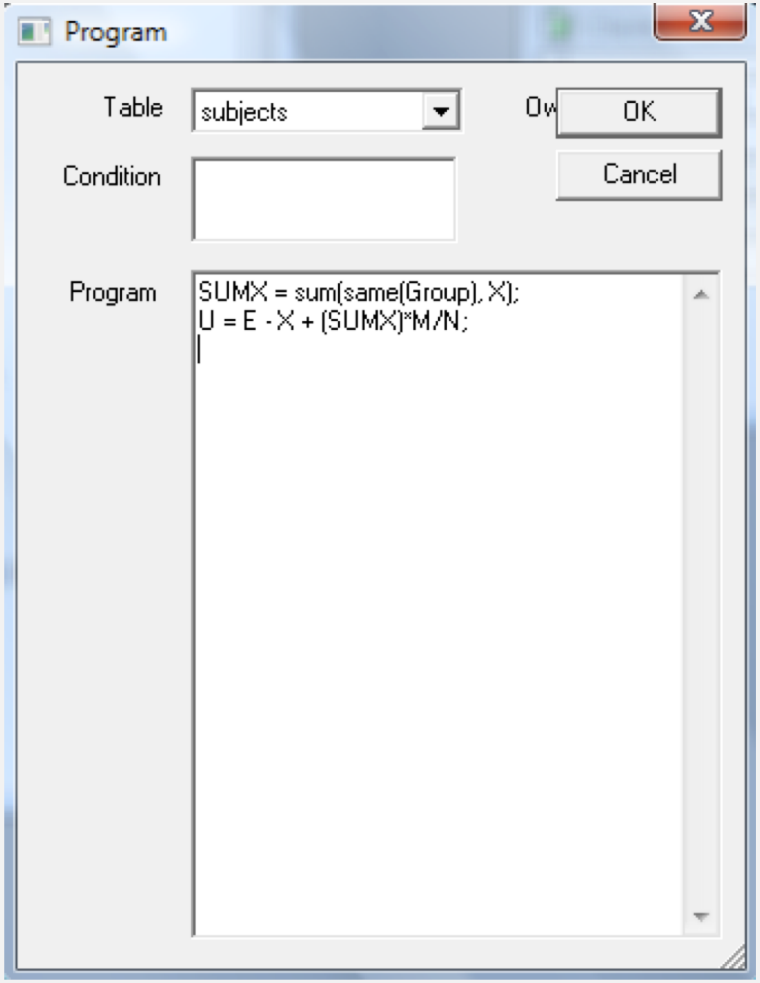
(subjects table)

- 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

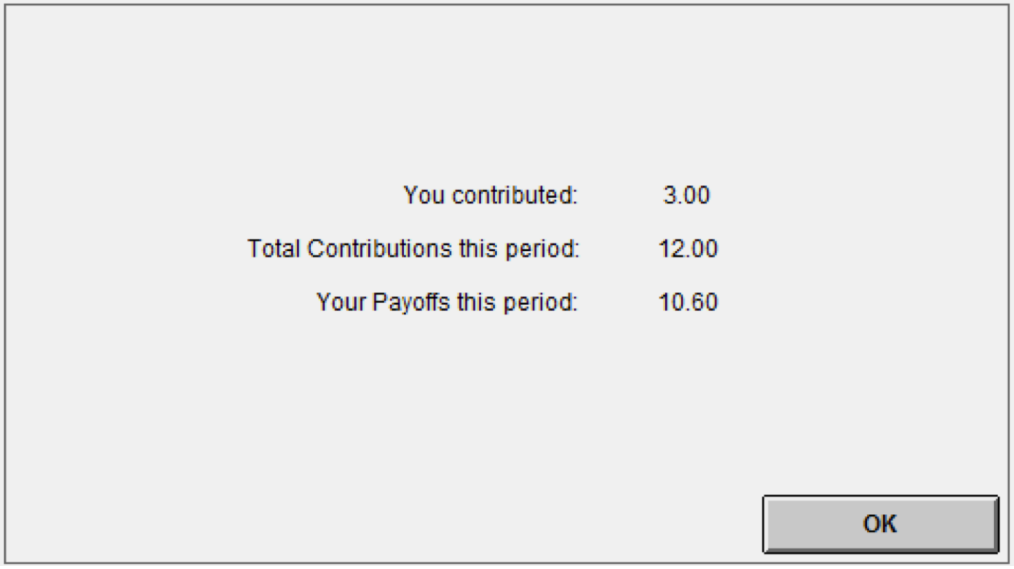
g	M	x	y
5	20	15	36
12	20	8	36
7	20	13	36

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**



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)

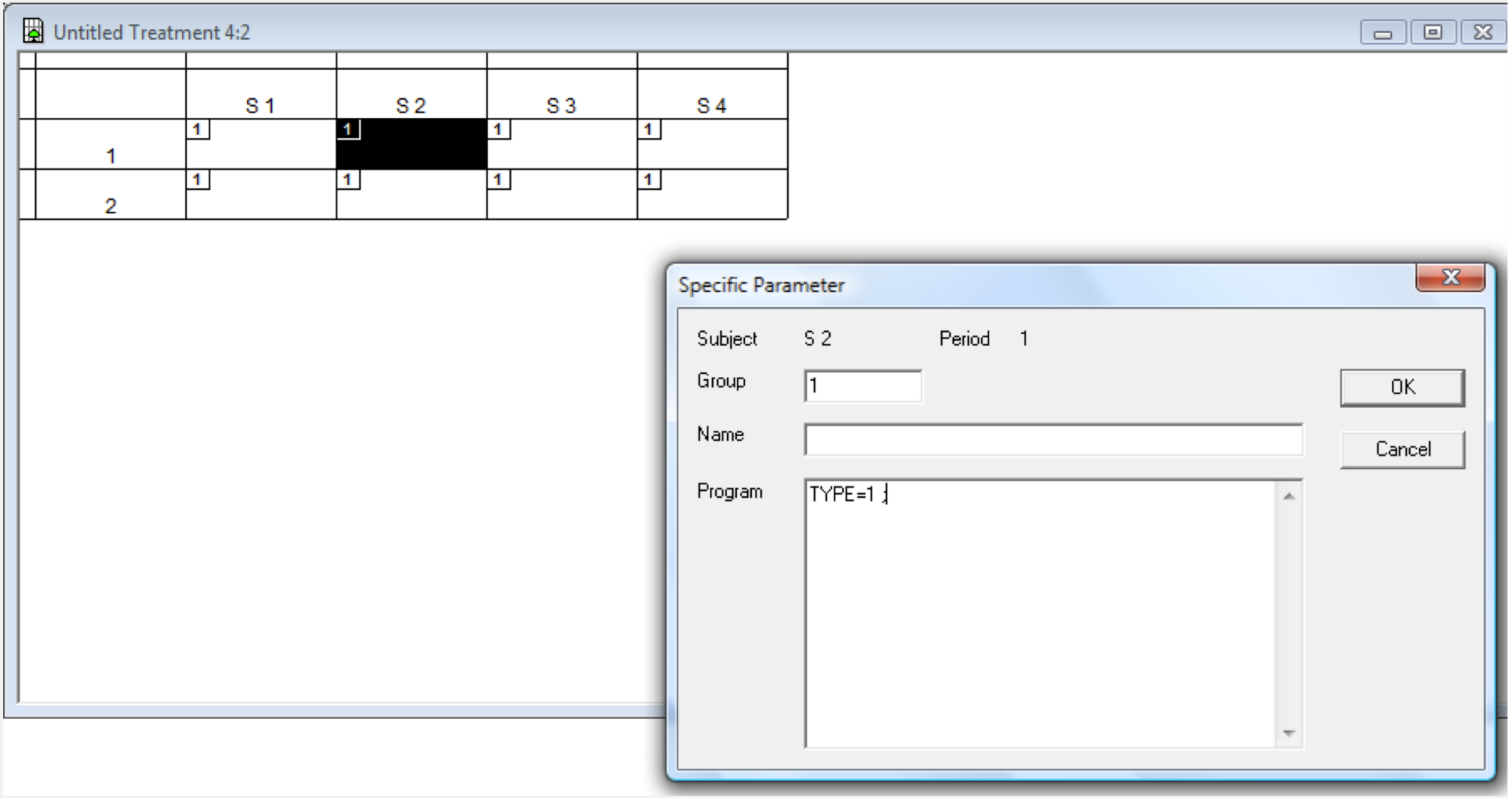
```
POT = 10; //Amount of money to be shared
```

> 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



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
```

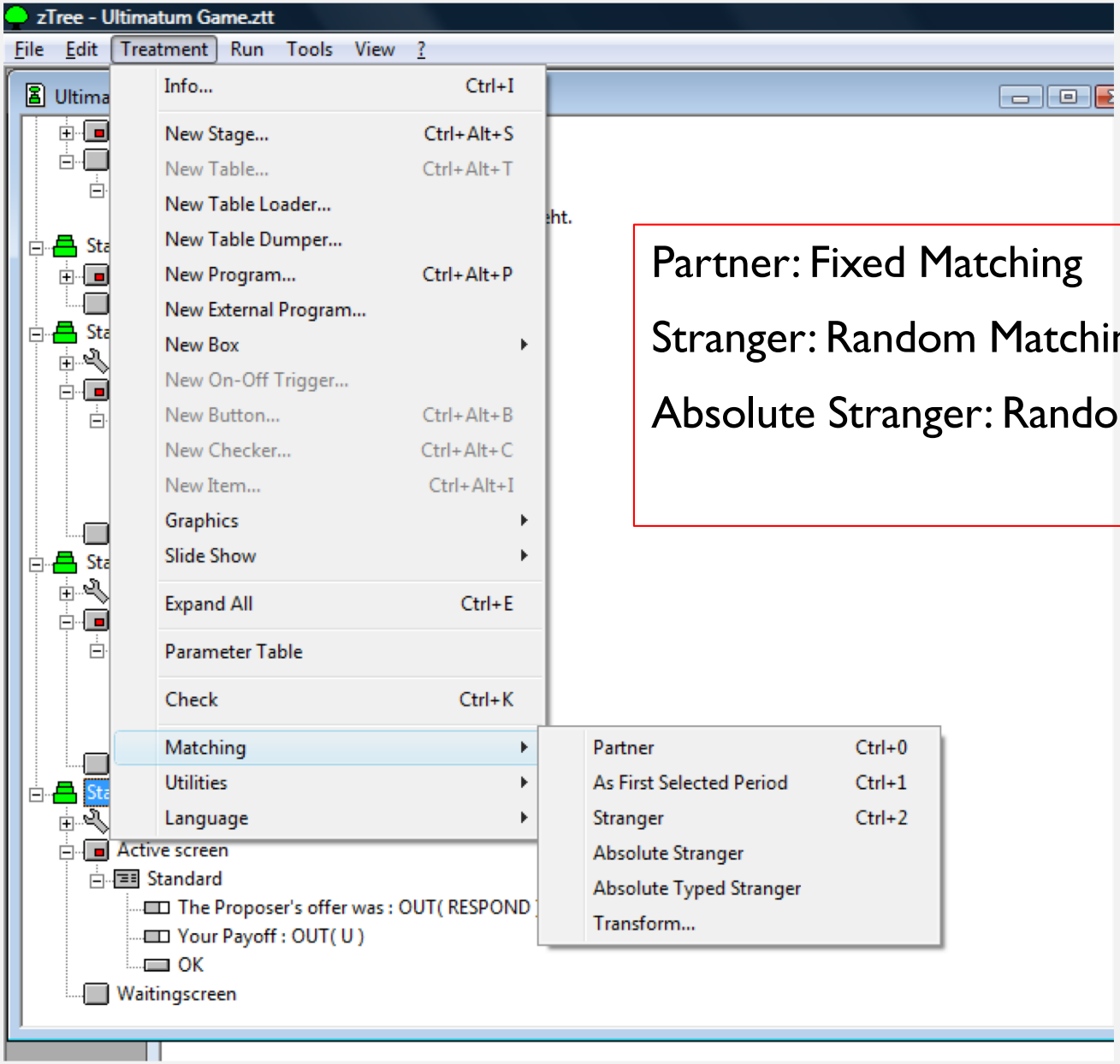
> Treatment > New program (subjects table)

```
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

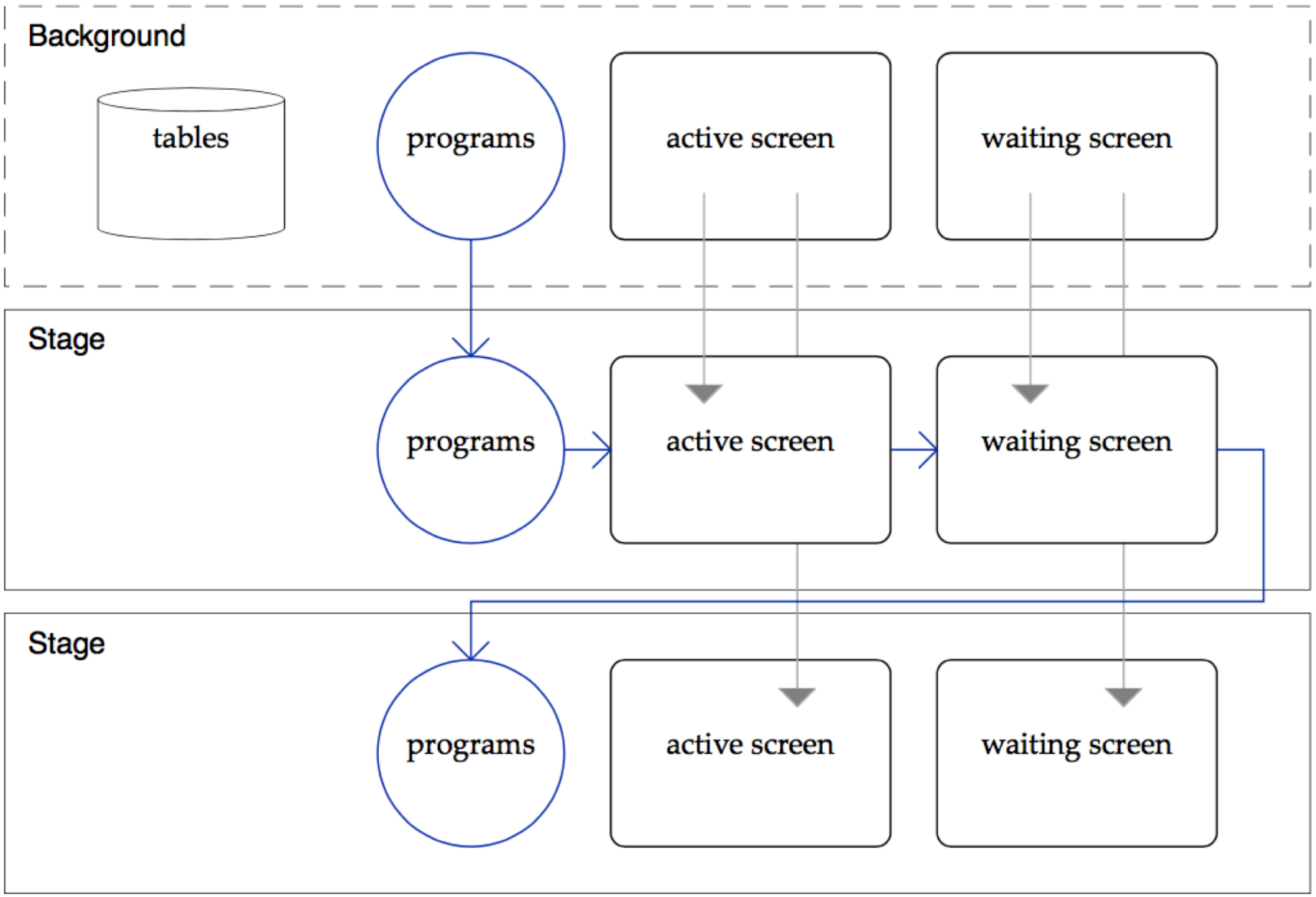
- Proposer: Makes an offer
- Responder: -

STAGE 3

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

STAGE 4

- Payoffs are realised (Simultaneous)


Background
tables
programs
active screen
waiting screen
Stage
programs
active screen
waiting screen
Stage
programs
active screen
waiting screen

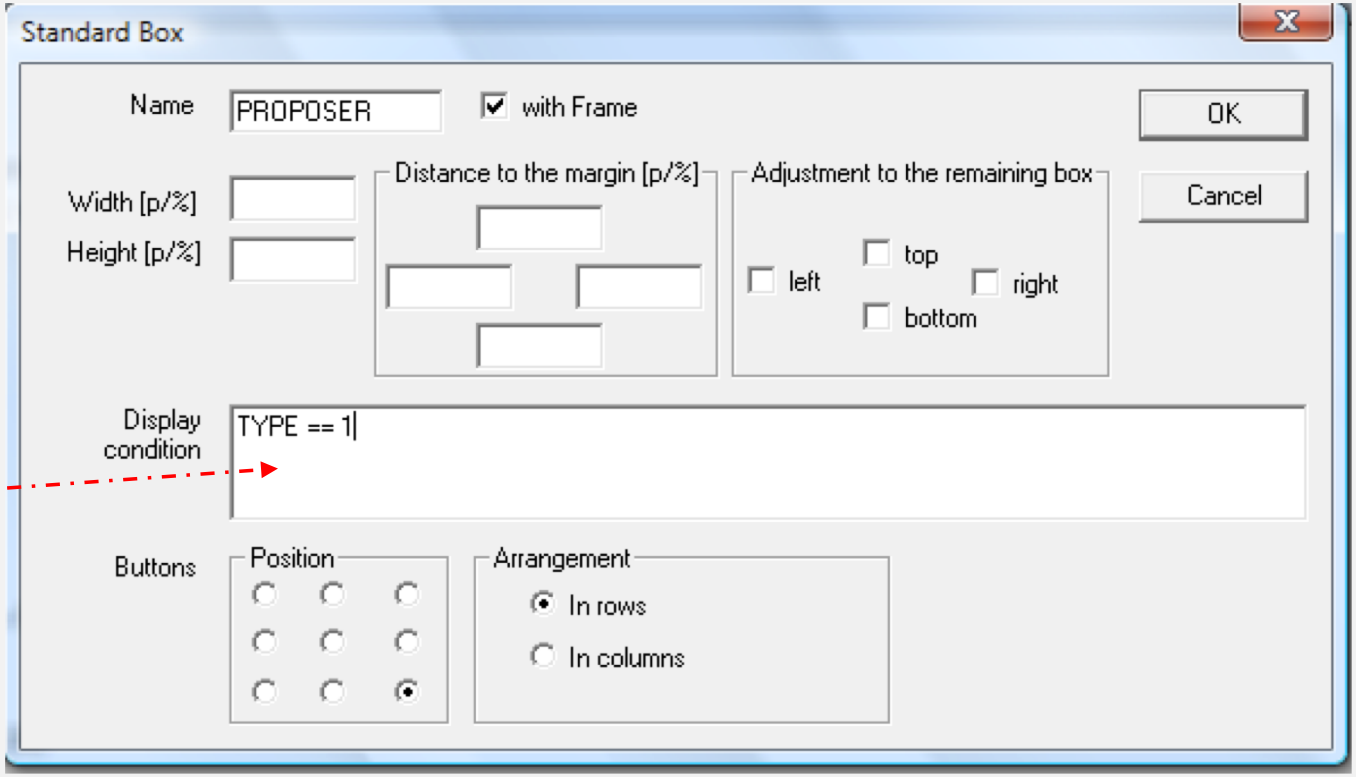
STAGE 1
(Subjects learn about their types)

Solution 1: 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

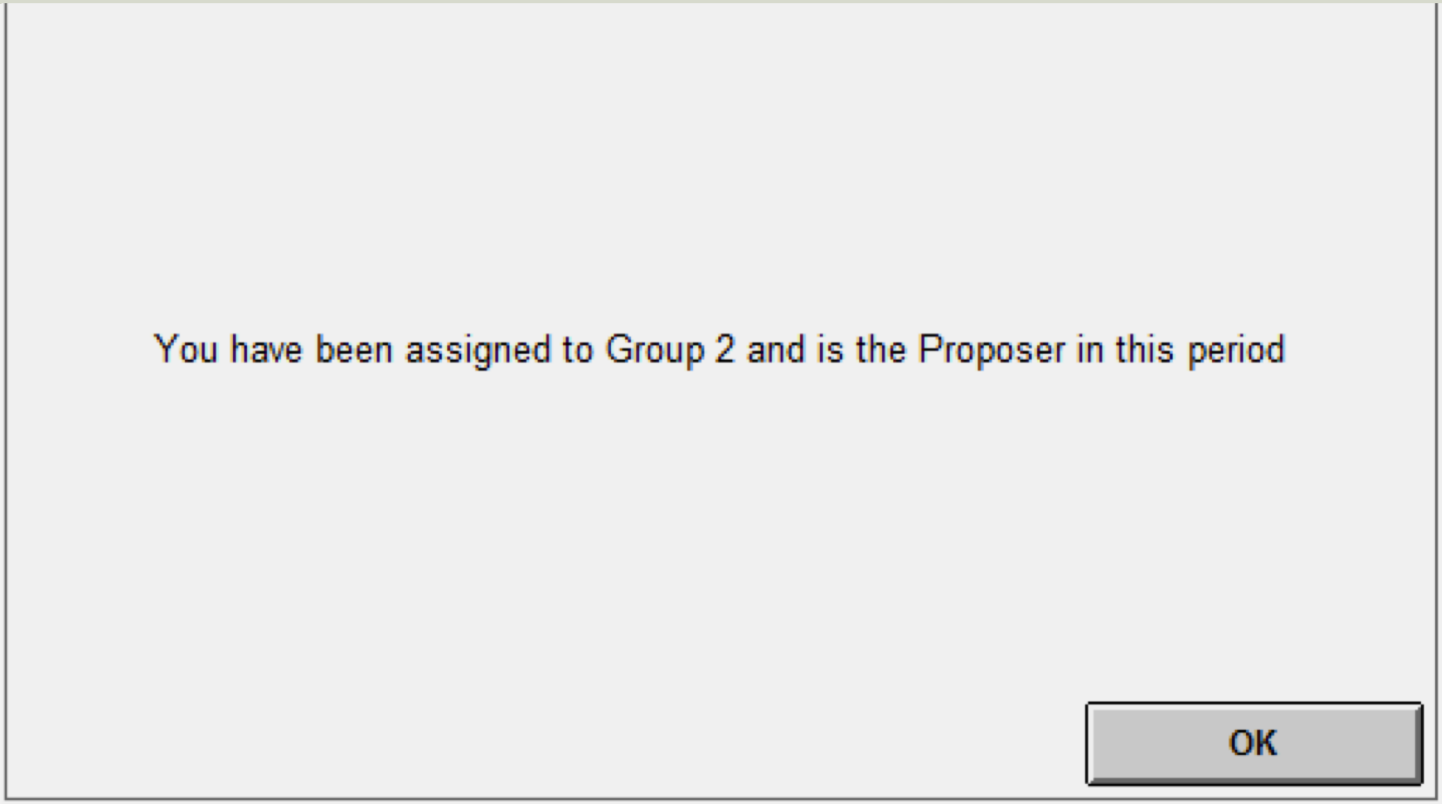
Only subjects with TYPE=1 will see the items in this box



STAGE 1
(Subjects learn about their types)

Solution 11: 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}
```



GENERAL
(rtf codes)

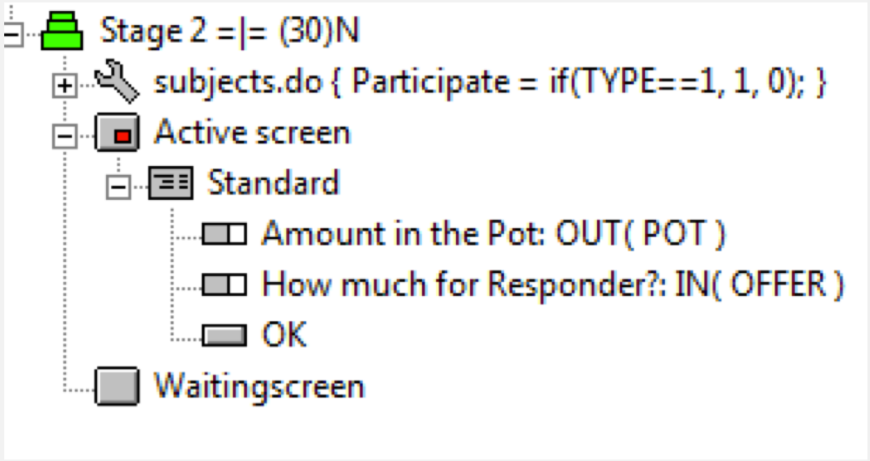
`\tab`	tabulator
`\par`	new paragraph
`\line`	new line
`\bullet`	bullet
`\ql`	aligned to left
`\qr`	aligned to right
`\qc`	centered
`\b`	bold
`\b0`	not bold
`\i`	italic
`\i0`	not italic
`\sub`	small and inferior numbers (index)
`\super`	small and superior numbers (exponent)
`\strike`	crossed through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

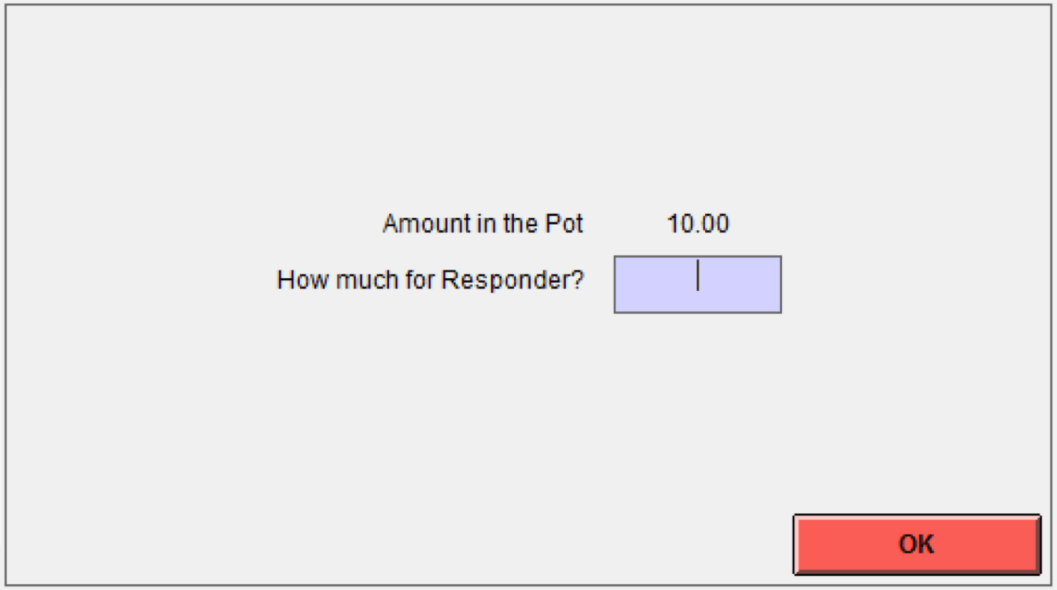
STAGE 2
(Proposer makes offer)

We only want the Proposer to enter stage 2

> Treatment > New program (subjects table)

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



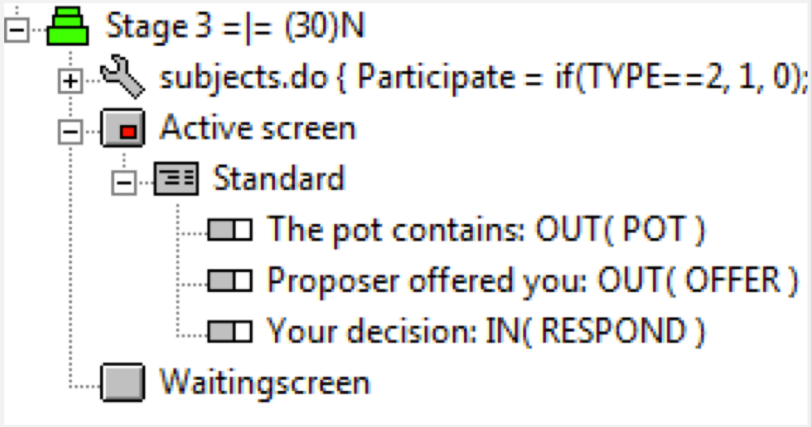


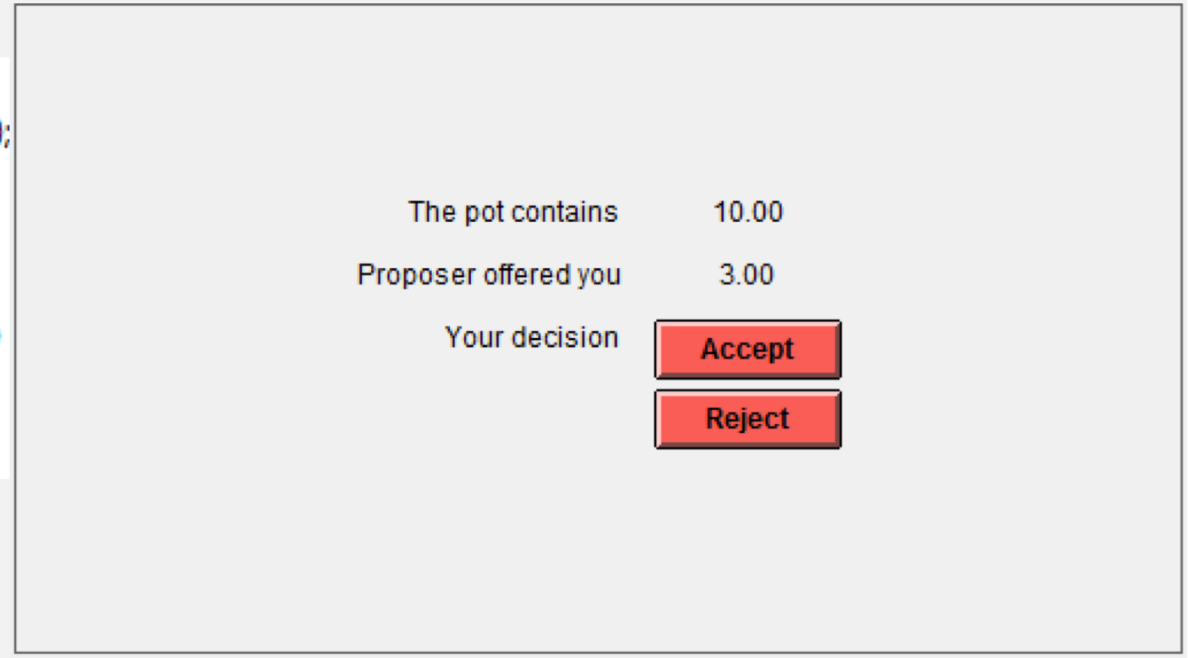
STAGE 3
(Responder Decides)

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);
```



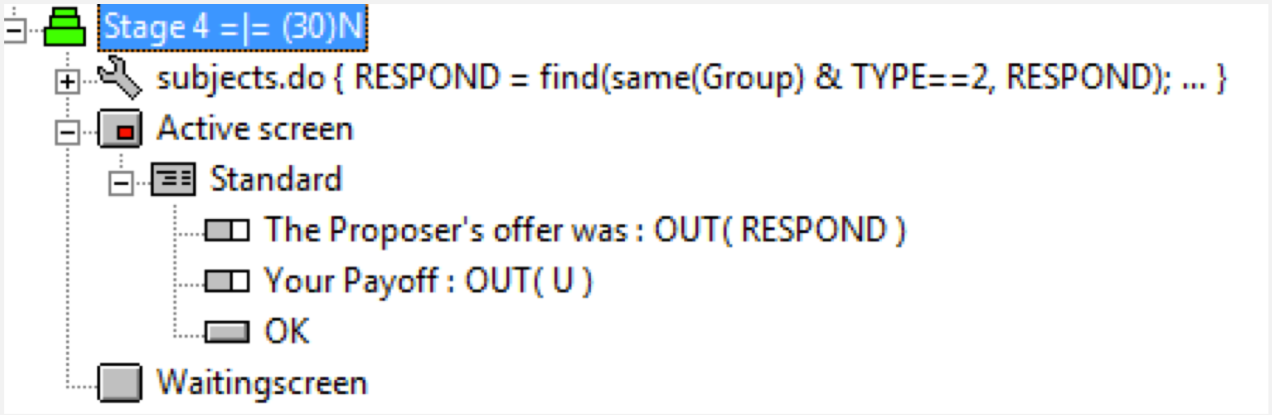


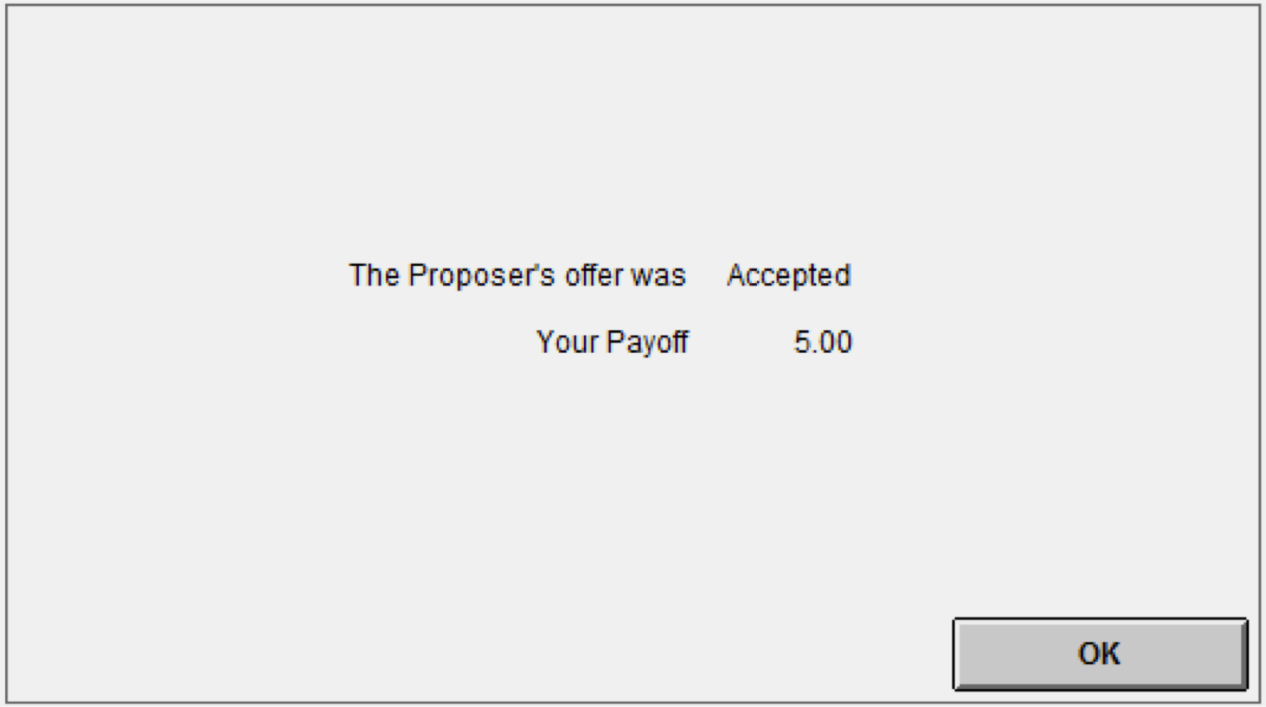
STAGE 4
(Compute payoff)

> Treatment > New program (subjects table)

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

STAGE 4
(Compute payoff)





CLASS EXERCISE I

SECOND PRICE AUCTION

TASK

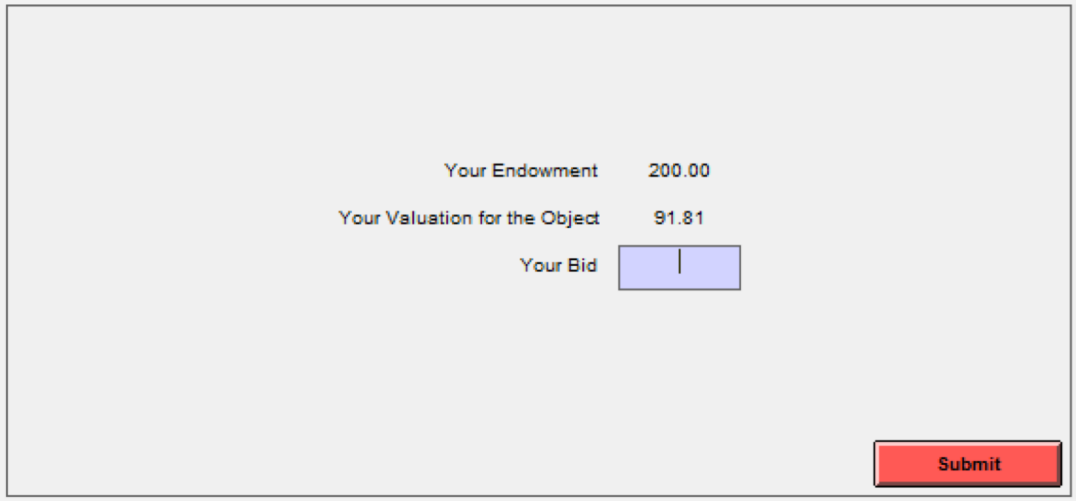
- N=4 bidders
- Valuations between [0,100] uniform
- Bidders are endowed with E = 200
- 2nd price auction
- In the event of a tie, random allocation amongst all claimants

Some useful scope operators

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

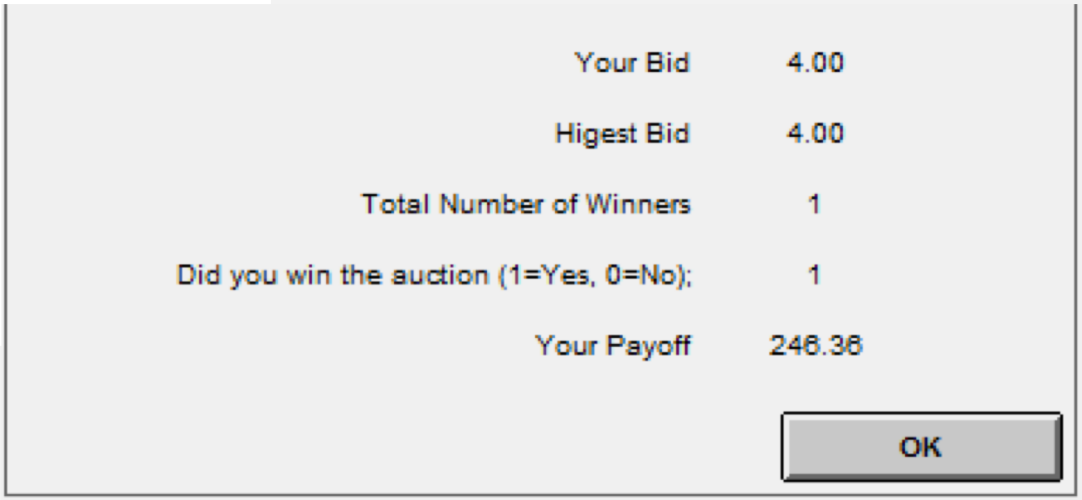
POTENTIAL SOLUTION

- Background
  - globals
  - subjects
  - summary
  - contracts
  - session
  - logfile
  - subjects.do { ... }
    - BID = 0; // Subjects' bid
    - HB = 0; // Higest Bid in group
    - HBO = 0; //Higest bid of all other bidders
    - WIN = 0; // 1 if win the auction and 0 if not
    - SUMWIN = 0; //Total number of winners
    - V = random()*100; //Subjects' valuation for object
    - E = 200; //Endowment
    - U = 0; //Total payoff
  - Active screen
    - Header
  - Waitingscreen
    - Text
      - Please wait until the experiment continues.
- Submit Bid =|= (30)N
  - Active screen
    - Standard
      - Your Endowment: OUT(E)
      - Your Valuation for the Object: OUT(V)
      - Your Bid: IN(BID)
      - Submit
  - Waitingscreen



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 )
      Higest 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
```



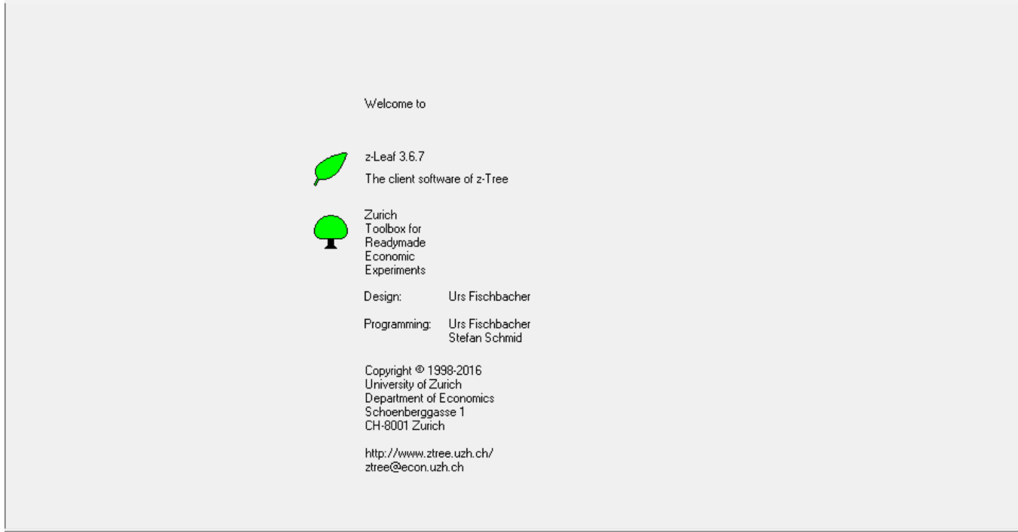
LESSON PLAN

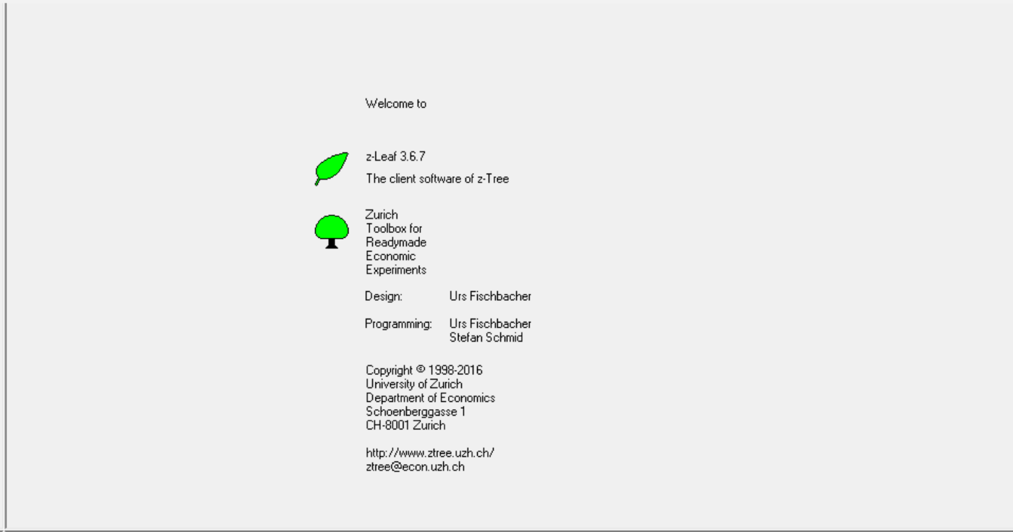
Day 11

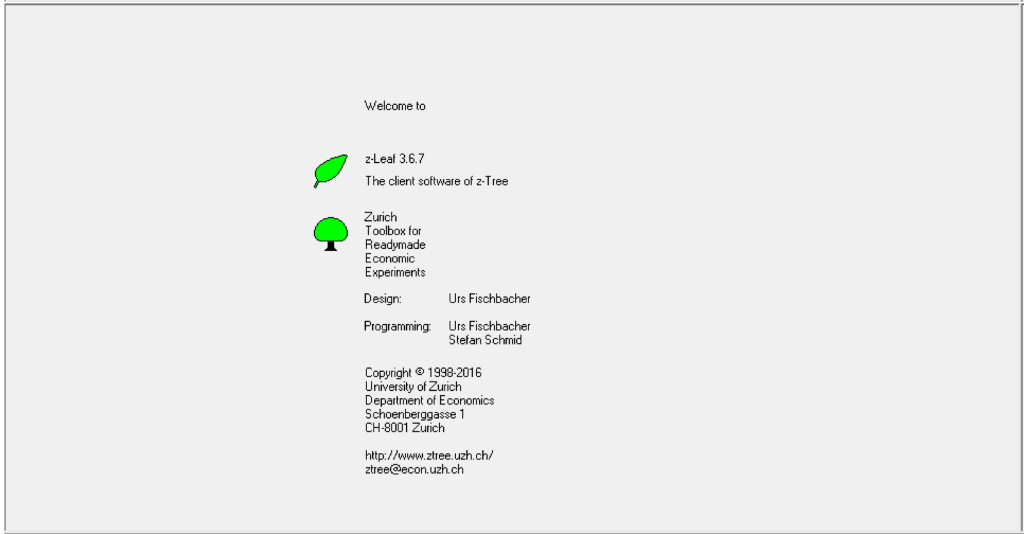
- **Creating multiple leafs on a screen**
- **Example III: 2x2 Normal form game**
  - Laying out Grid matrix
  - Random round payment
- **Example IV: Search Lottery**
  - Array programming and complex loops
  - Programming a Survey
- **Class Exercise II: Jackpot machine (A fair jackpot)**
- **Example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

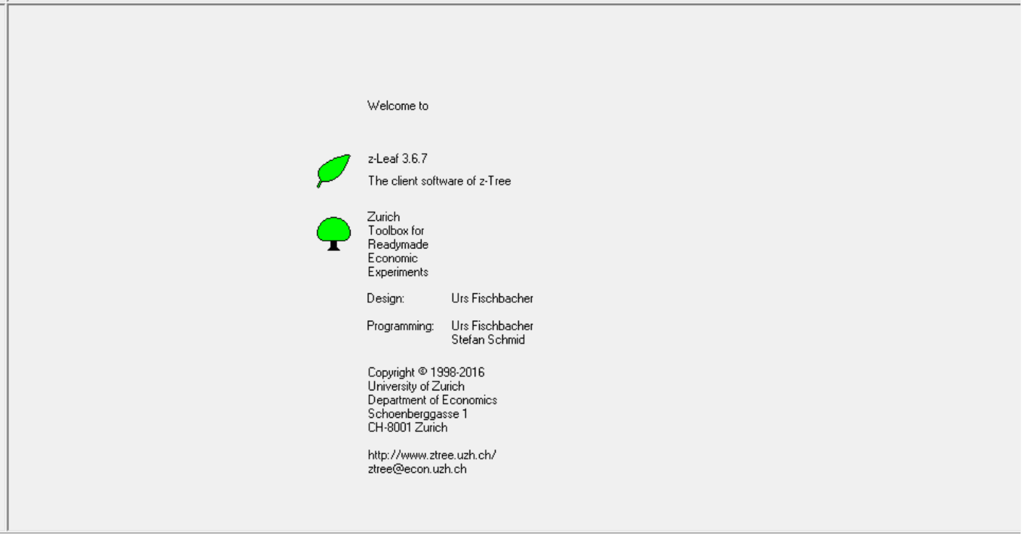
CREATING MULTIPLE LEAFS ON A SCREEN

MULTIPLE LEAFS



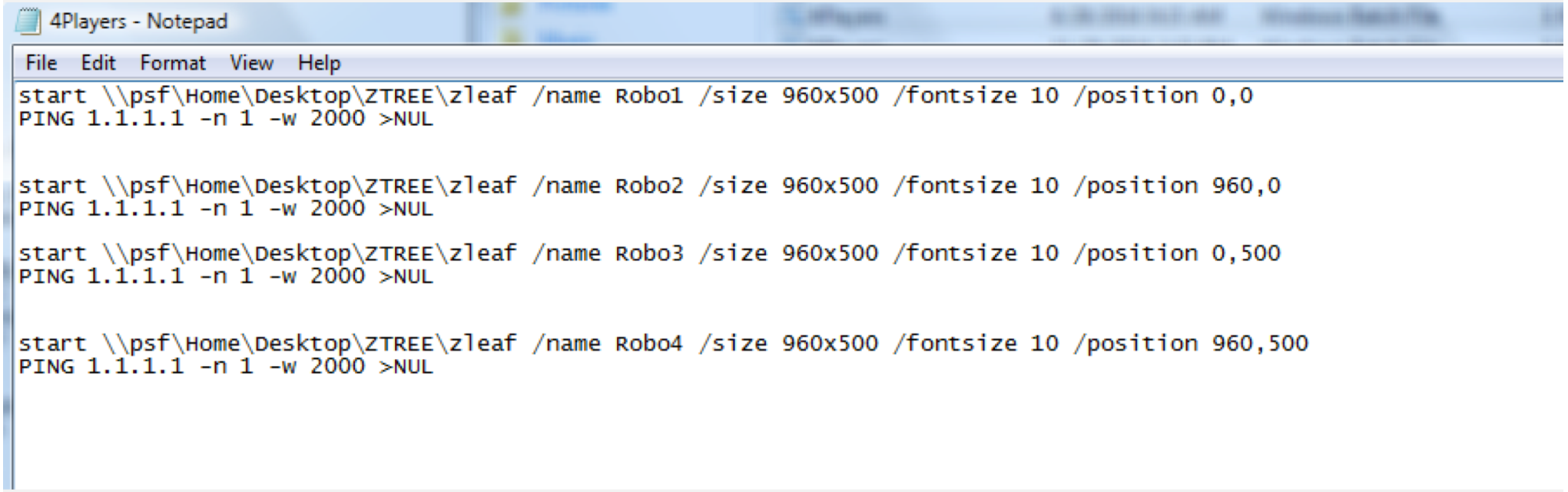






GENERAL
(Creating multiple leafs)

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



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;
}
elsif(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)

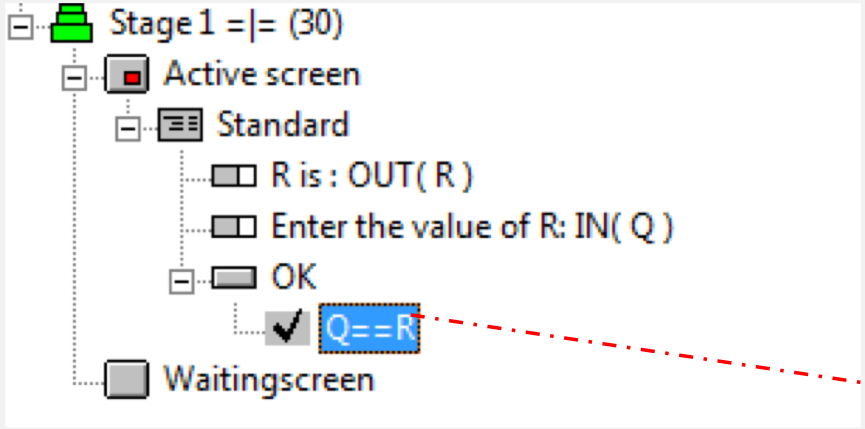
STAGE 1

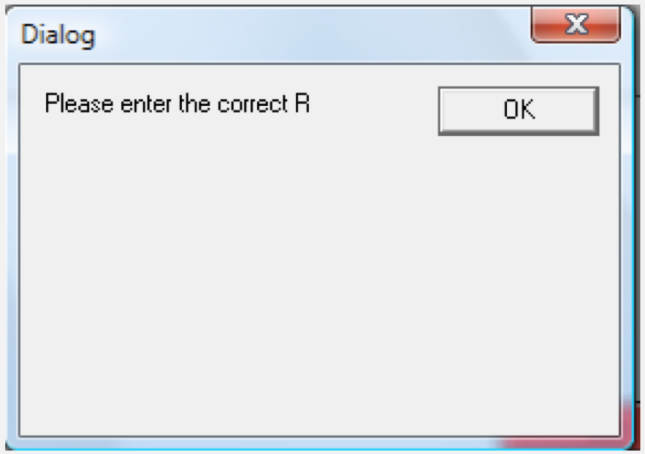
> Treatment > New Program (subjects)

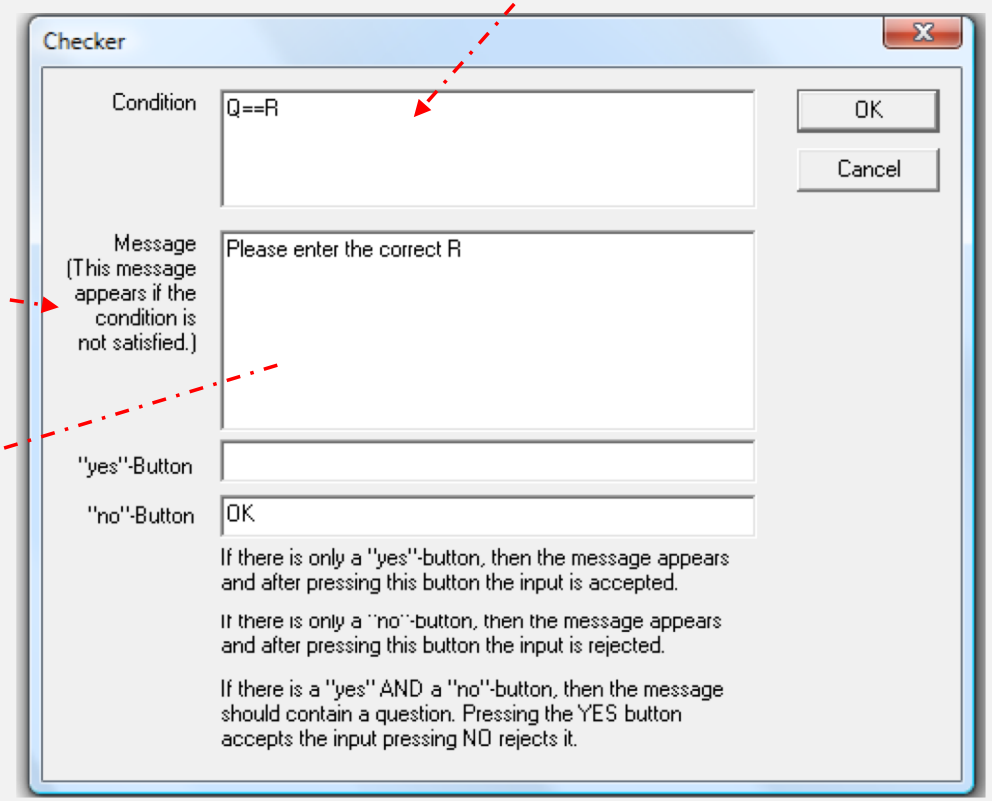
```
Participate = if(Period==1, 1,0);
```

> Treatment > New Checker

The true condition that has to be met.





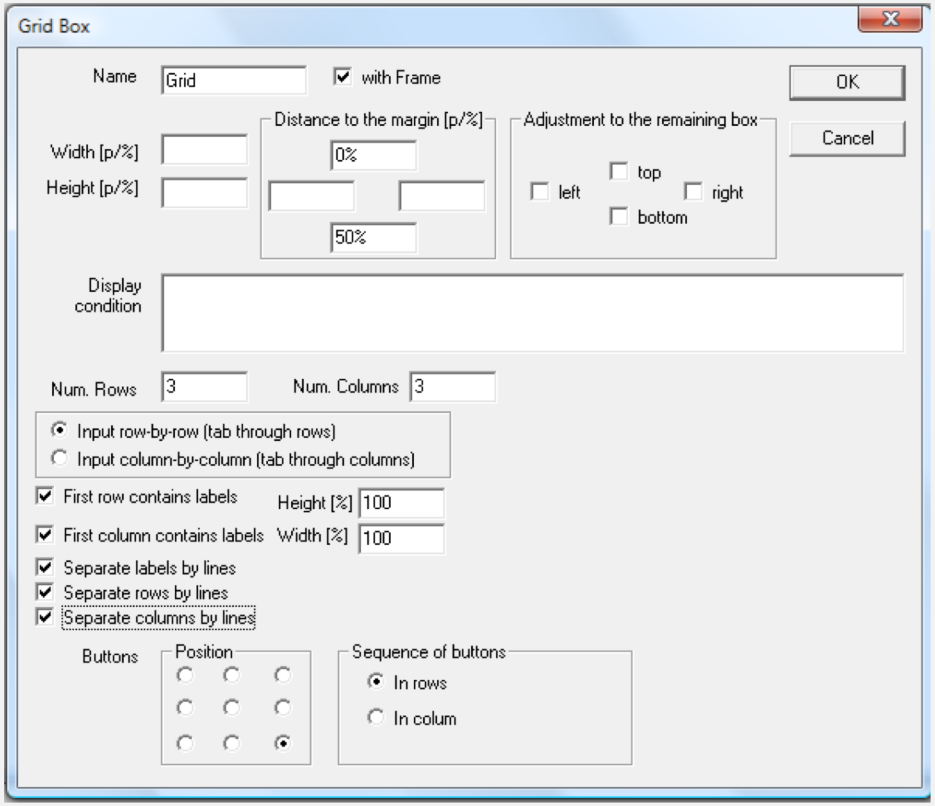


STAGE 2

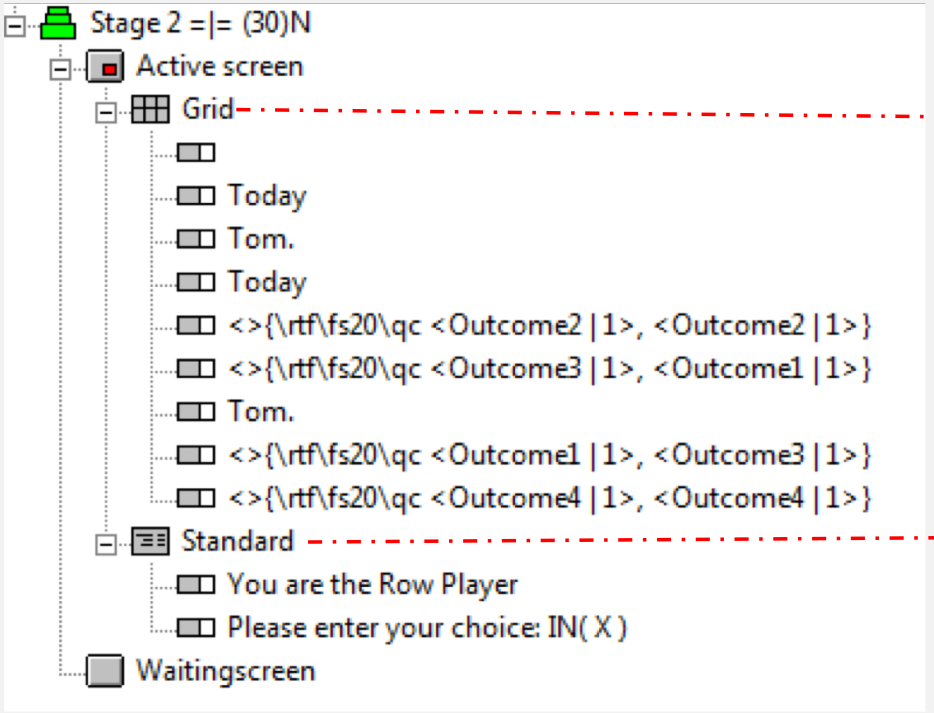
Grid Box to show 2x2 matrix

Standard Box for subject's input

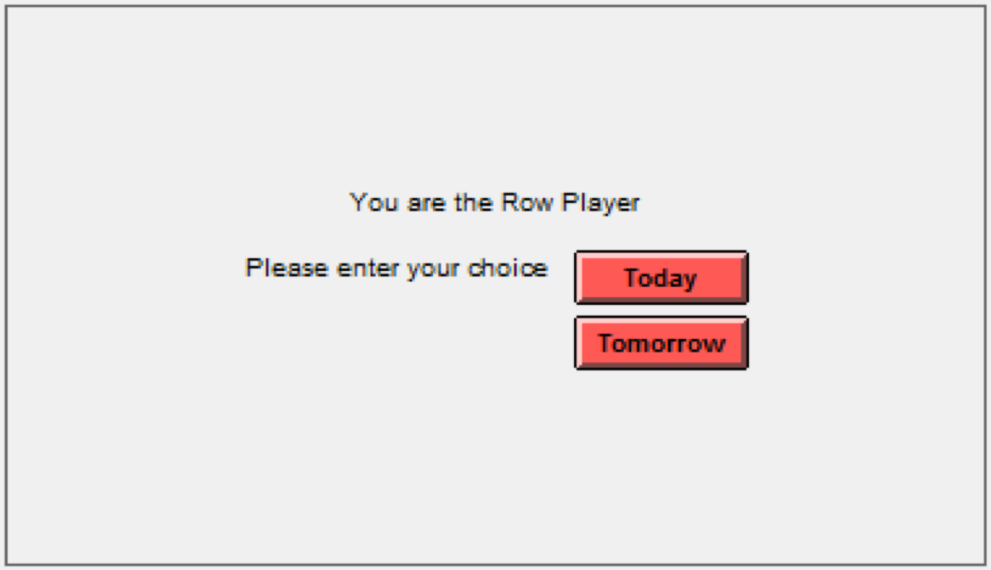
> Treatment > New Box > Grid Box



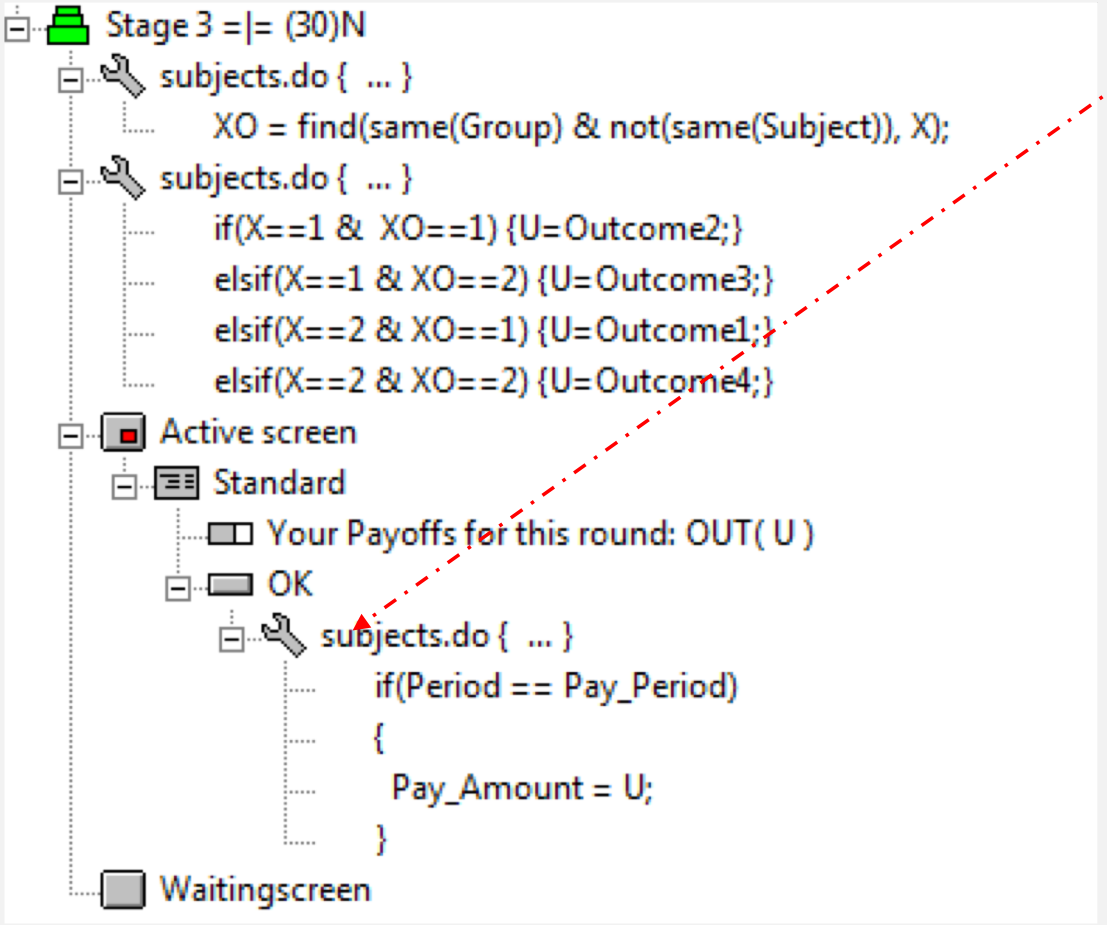
STAGE 2



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

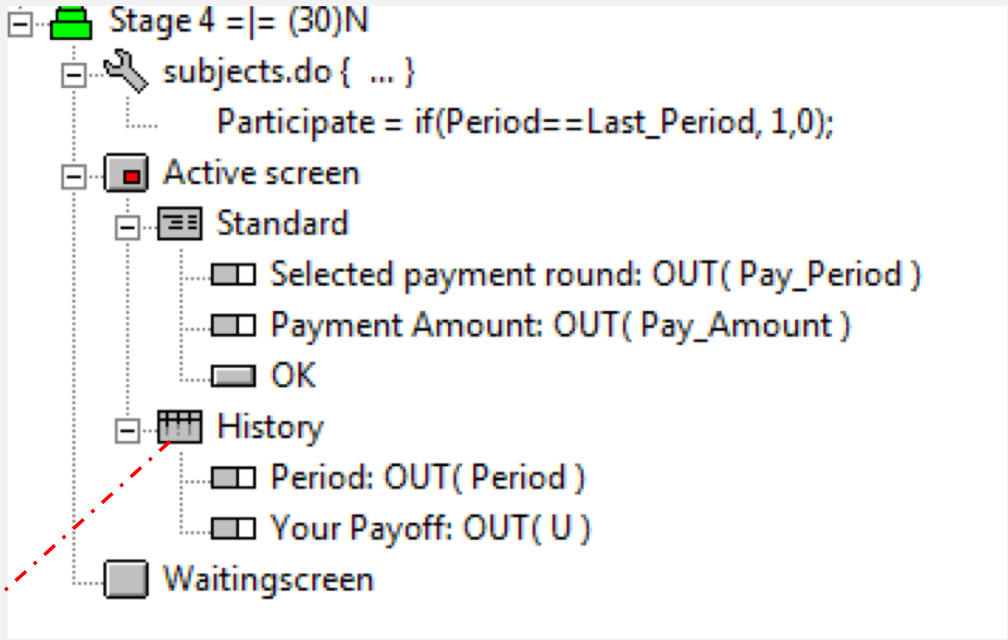


STAGE 3



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, …., 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;}
elsif(Effort == 5) {Cost = 2;}
elsif(Effort == 10) {Cost = 4;}
…
elsif(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],….,C[20]

GENERAL
(generating loops)

Basic Loop

```
if ( condition ) { statements if condition is true;}
elsif ( condition) {statments 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

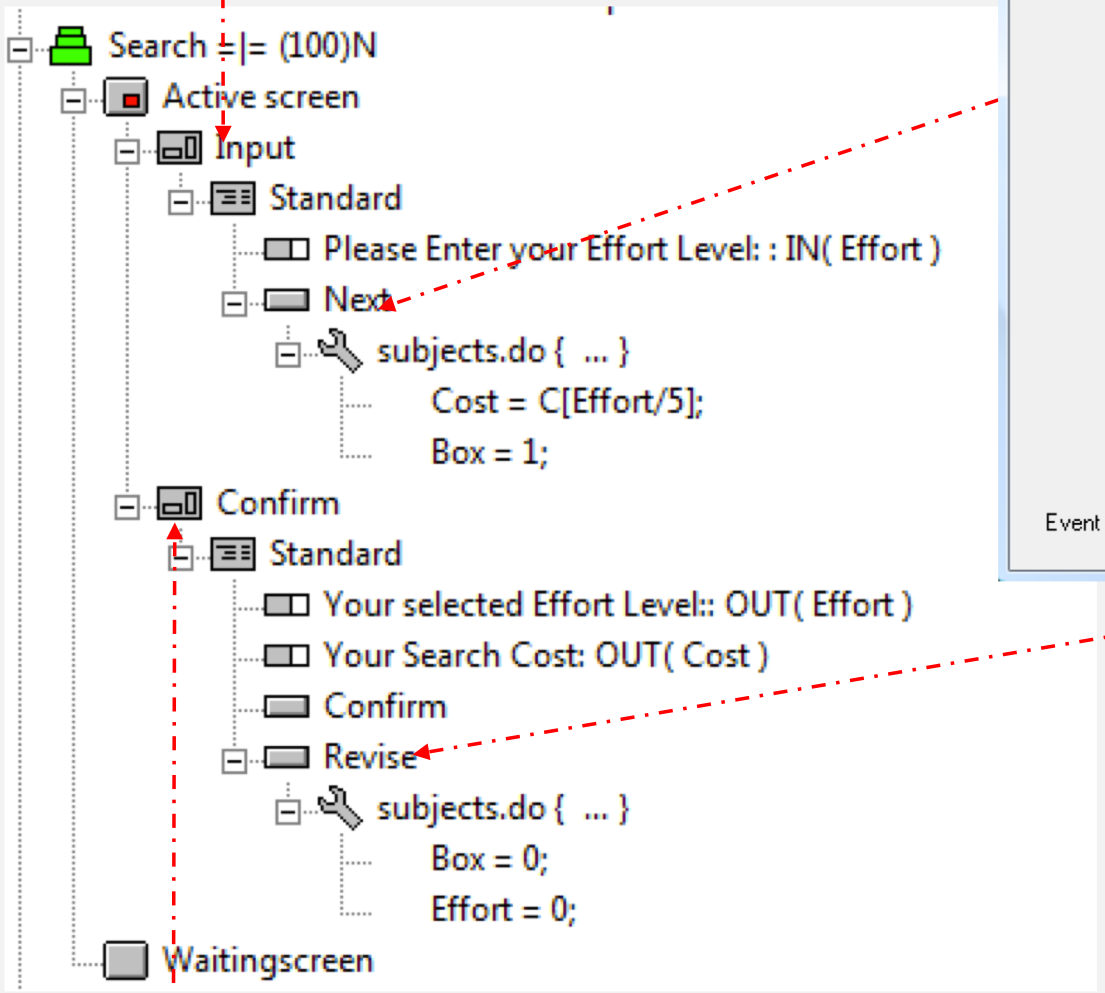
- Choose Effort (and see Cost)

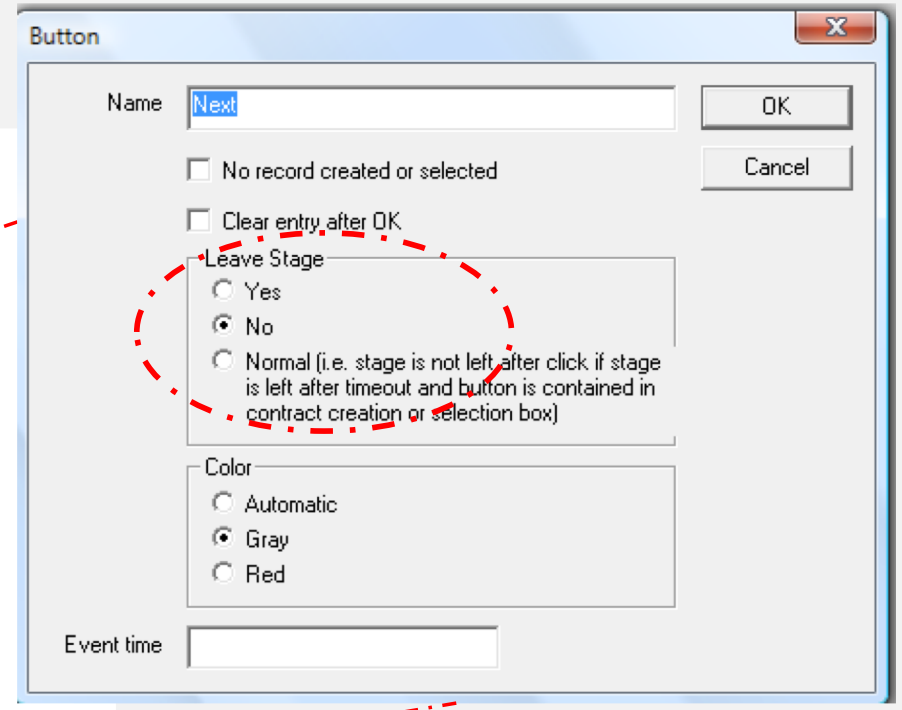
STAGE 2

- See Search Outcome

STAGE 1

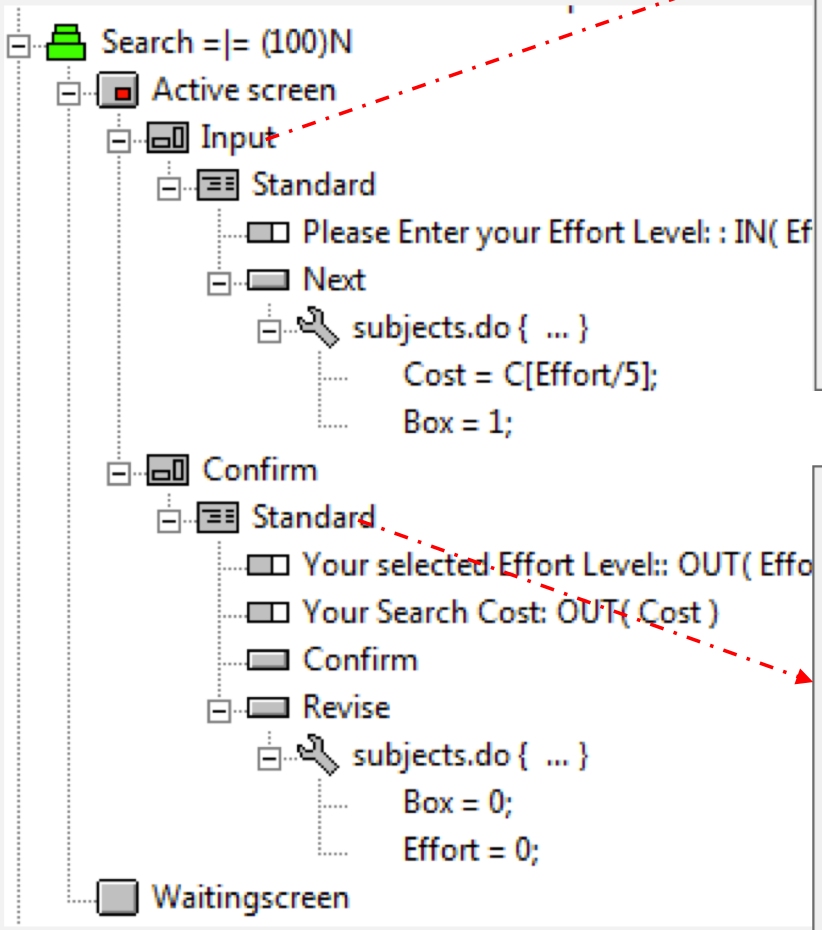
If Box == 0



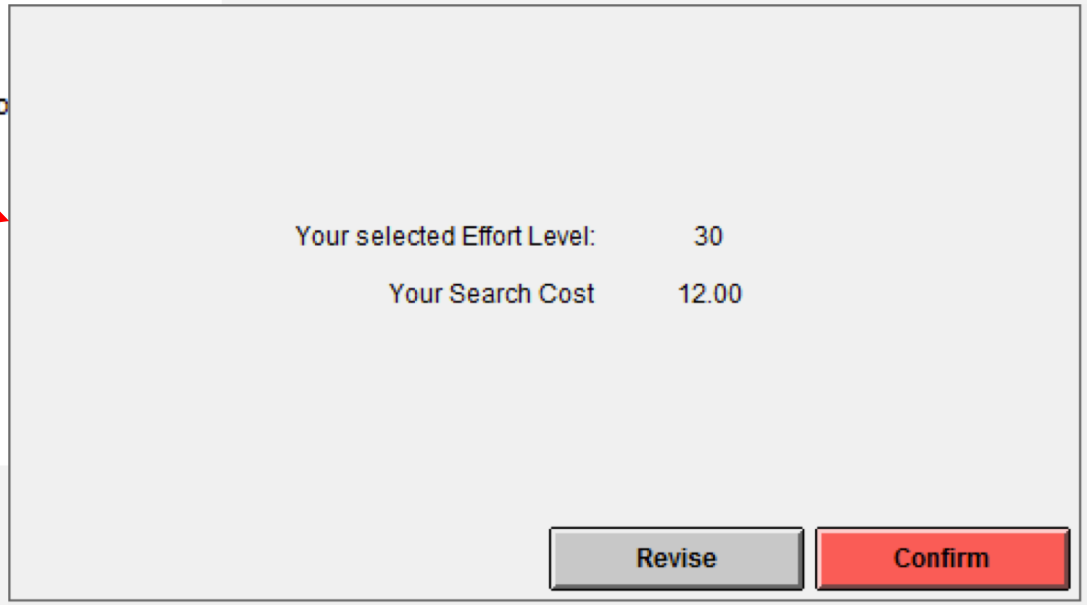


If Box == 1

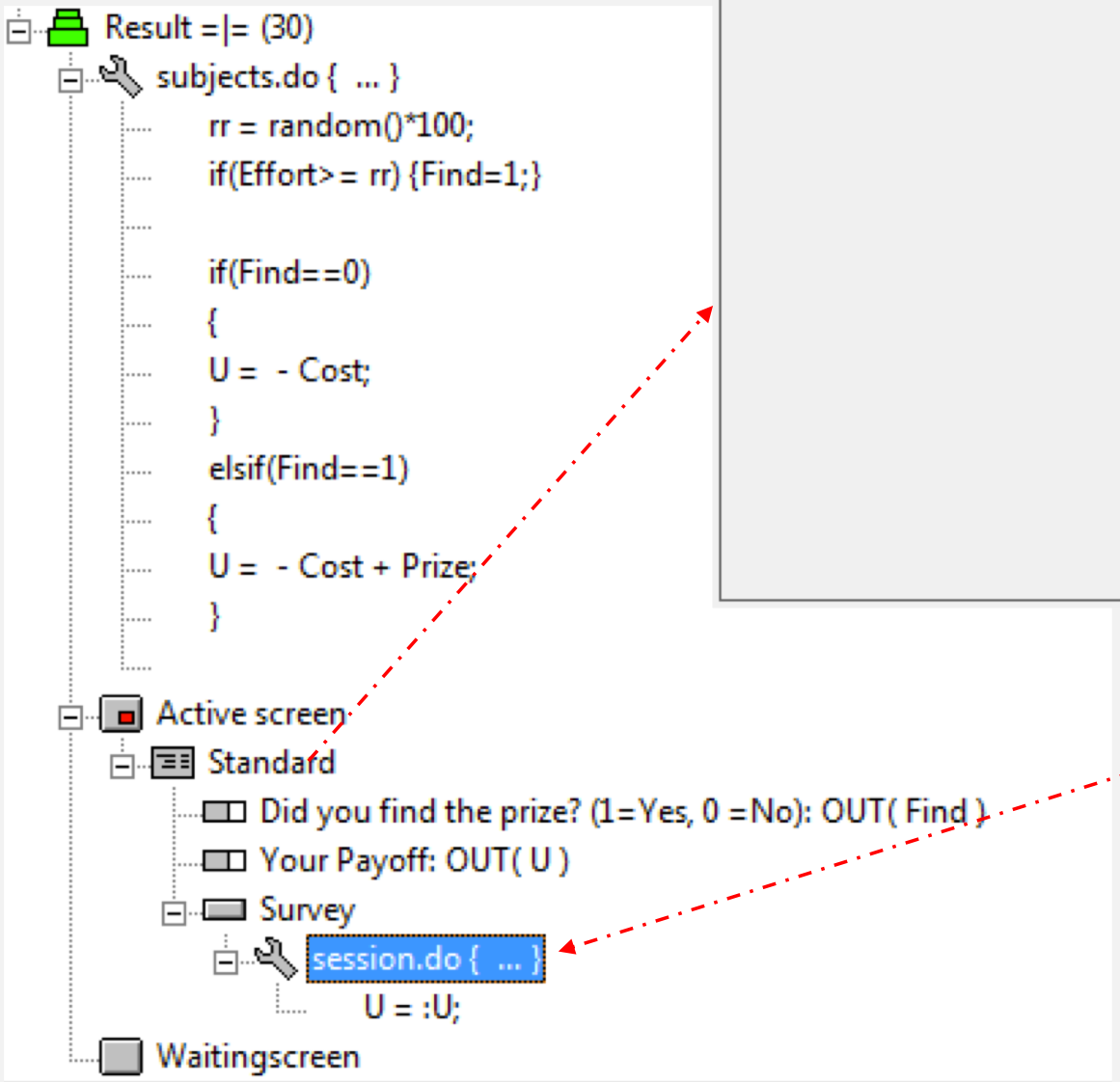
STAGE I

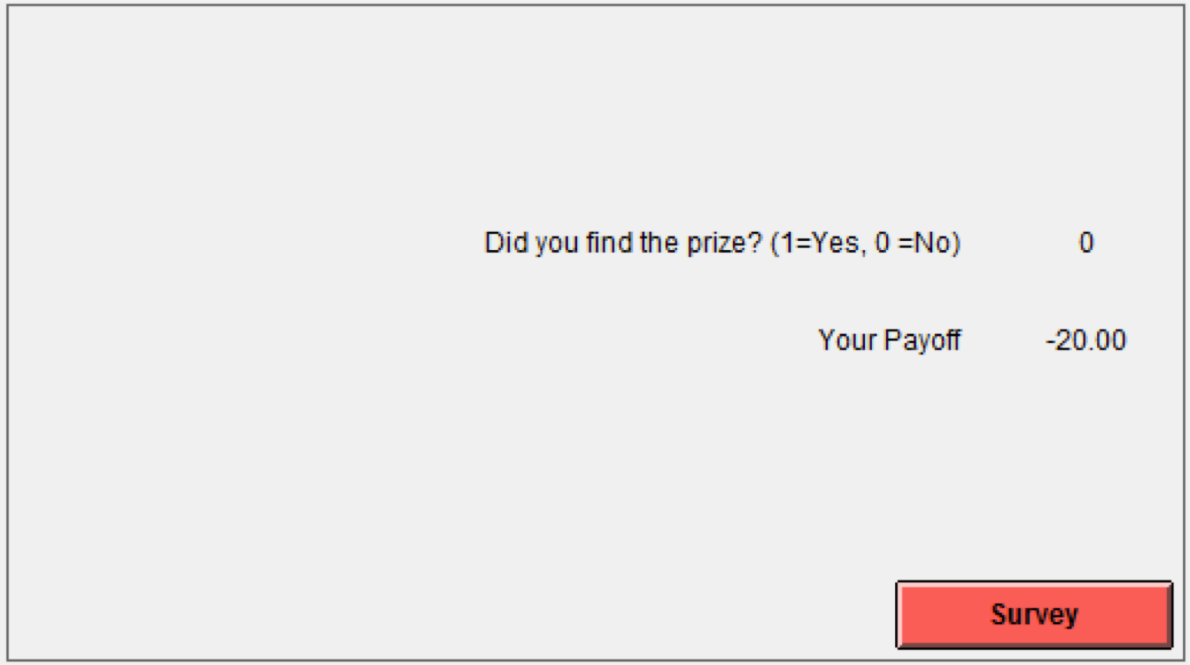

Search =|= (100)N
Active screen
Input
Standard
Please Enter your Effort Level: : IN(Ef
Next
subjects.do { ... }
Cost = C[Effort/5];
Box = 1;
Confirm
Standard
Your selected Effort Level:: OUT(Effo
Your Search Cost: OUT(Cost)
Confirm
Revise
subjects.do { ... }
Box = 0;
Effort = 0;
Waitingscreen


Please Enter your Effort Level:
Next


Your selected Effort Level: 30
Your Search Cost 12.00
Revise
Confirm

STAGE 2





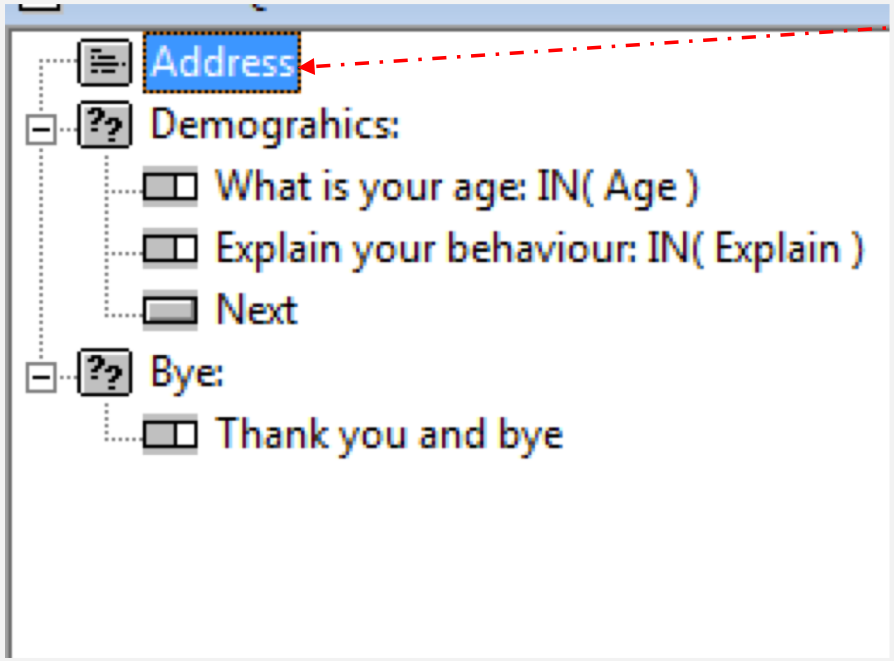
Write into the "session table"

GENERAL
(session table)

- One row per subject

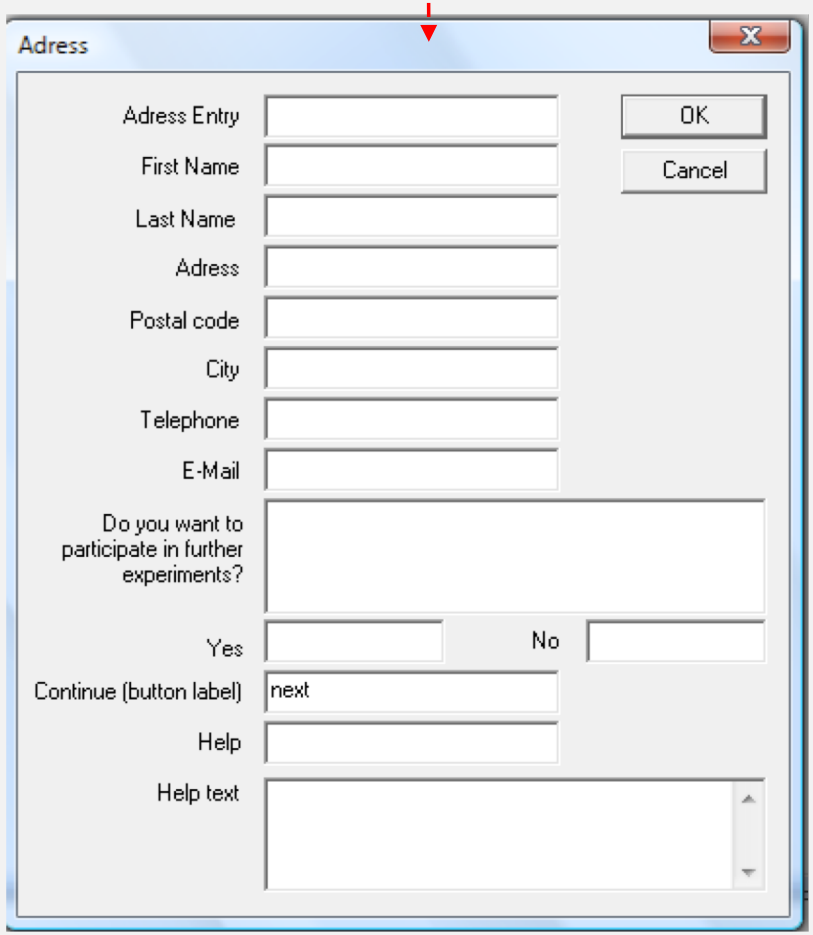
Subject	FinalProfit	ShowUpFee	ShowUpFeeInvested	MoneyAdded	MoneyToPay	MoneyEarned	X
1	12.65	0	0	0	0	0	3
2	11	0	0	0	0	0	4
3	12	0	0	0	0	0	56
4	9	0	0	0	0	0	8
5	16	0	0	0	0	0	9

SURVEYS



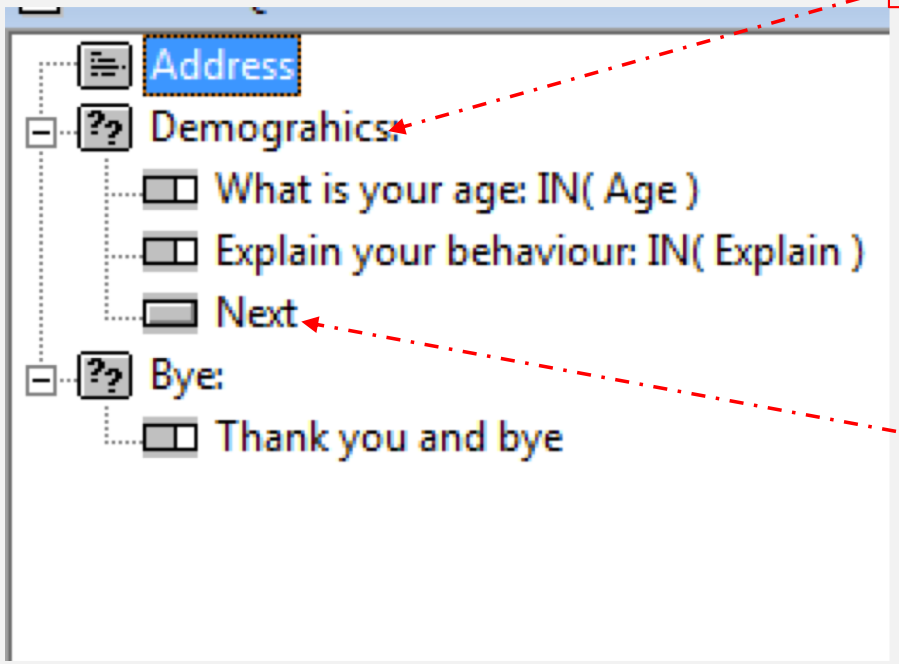
The survey design always starts with an "Address form"

> Questionnaire > New Address Form



Leave the details "empty" if you want to skip the address form

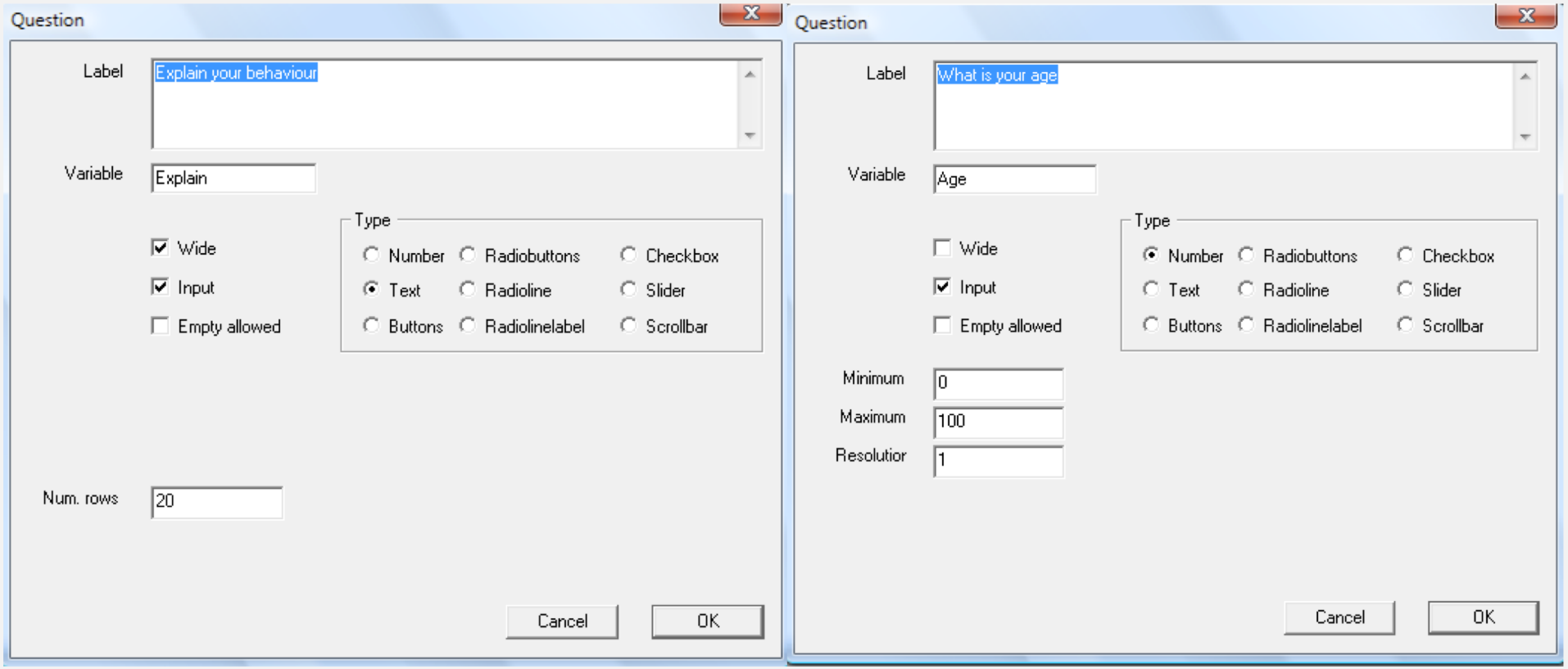
SURVEYS



> Questionnaire > New Question Form

> Questionnaire > New Button

SURVEYS


Question
Label
Explain your behaviour
Variable
Explain
Wide
Input
Empty allowed
Type
Number
Radiobuttons
Checkbox
Text
Radioline
Slider
Buttons
Radiolinelabel
Scrollbar
Num. rows
20
Cancel
OK
Question
Label
What is your age
Variable
Age
Wide
Input
Empty allowed
Type
Number
Radiobuttons
Checkbox
Text
Radioline
Slider
Buttons
Radiolinelabel
Scrollbar
Minimum
0
Maximum
100
Resolutior
1
Cancel
OK

SURVEYS

What is your age

Explain your behaviour

Next

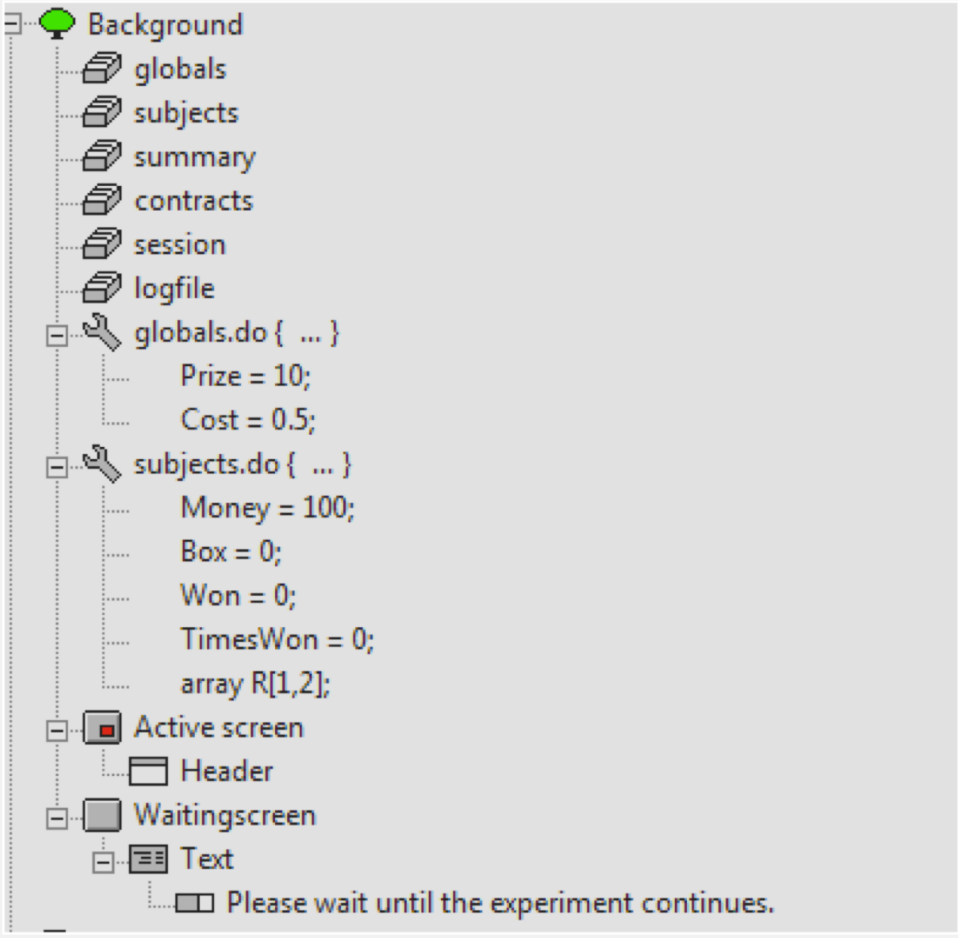
CLASS EXERCISE II

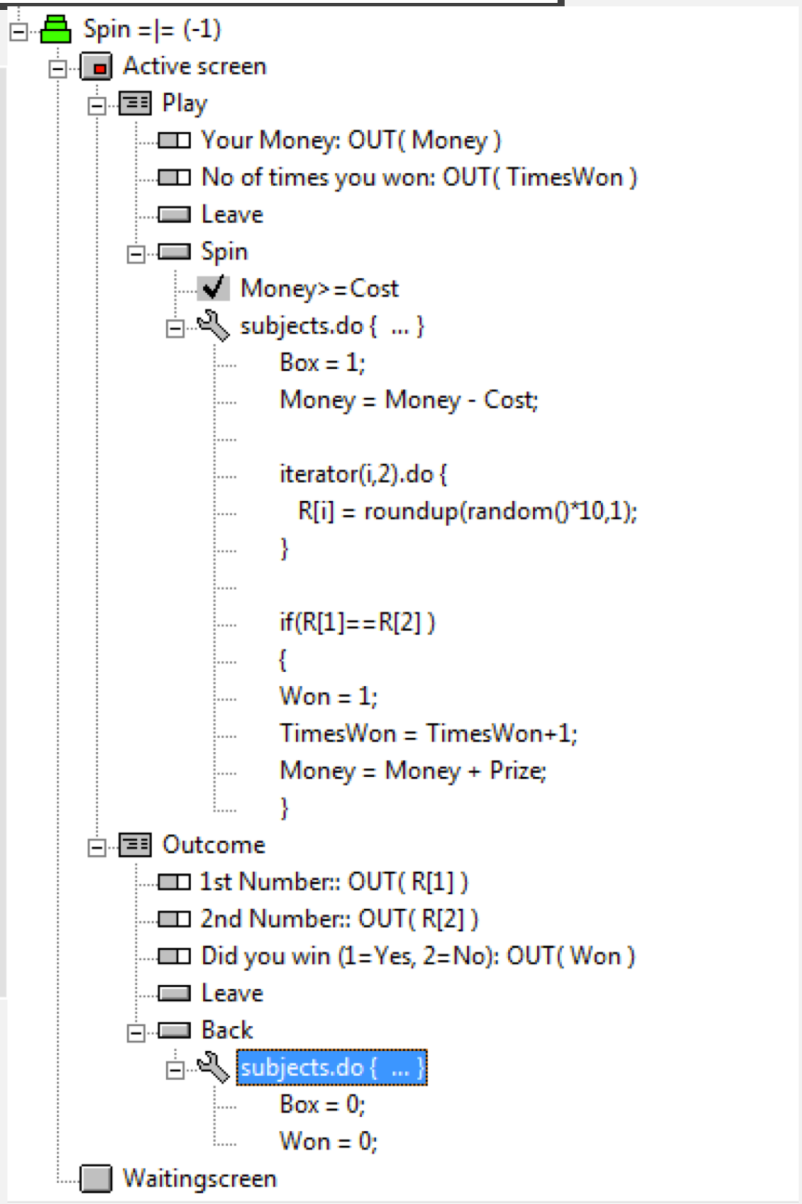
JACKPOT MACHINE

TASK

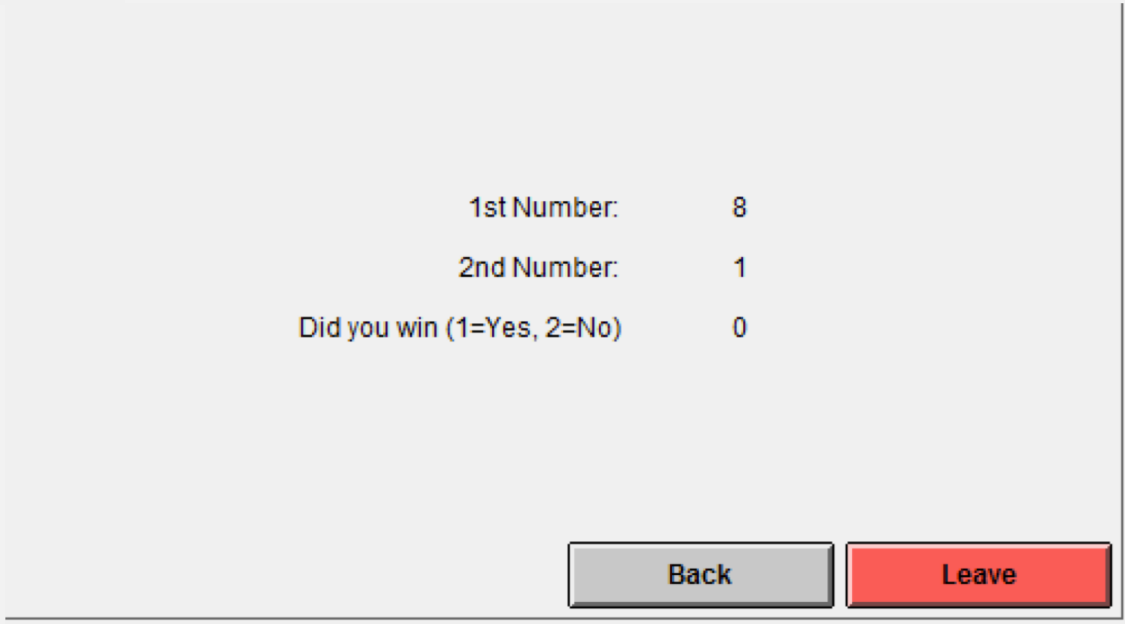
- Do a simple jackpot machine consisting of two numbers (1,2,…,10).
- Subject wins a prize if the two number are identical.
- Subject gets to ”spin” the jackpot as many times as he wants – subjected to budget constraint.
- For each spin:
  - Some money gets deducted (Tokens cost)
  - New random numbers (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



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* }

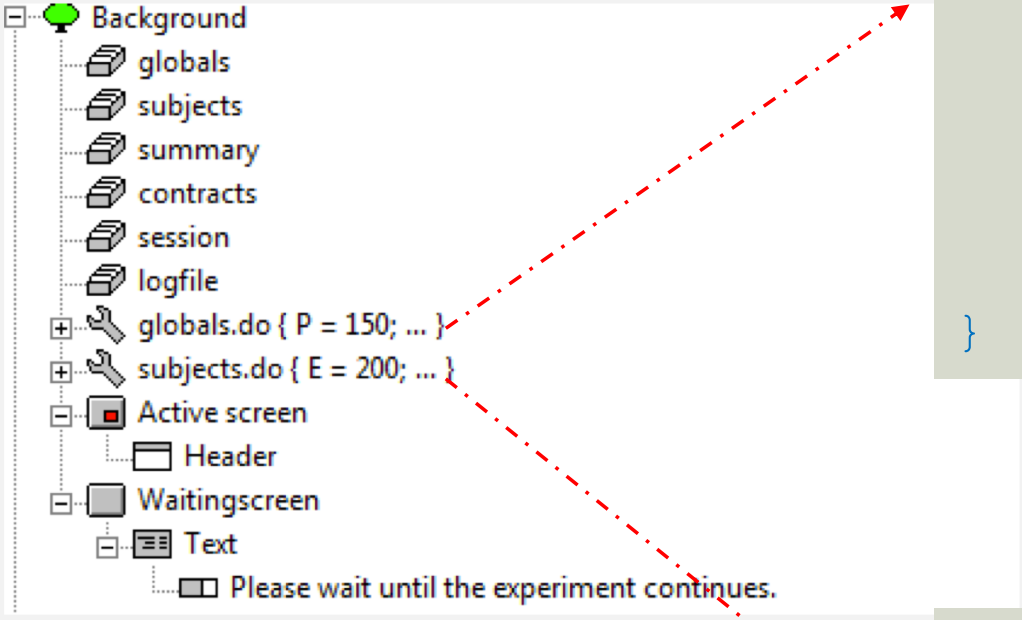
Note that the function does not have a build-in "while" condition.

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!

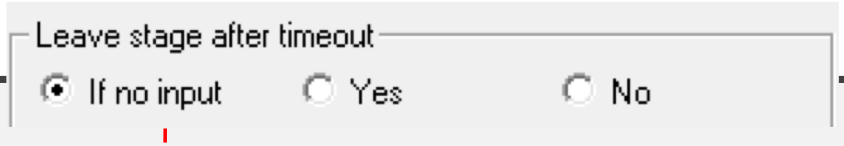
BACKGROUND (INITIAL VALUES)

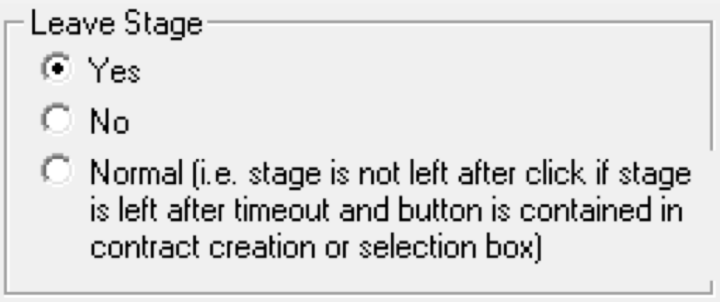


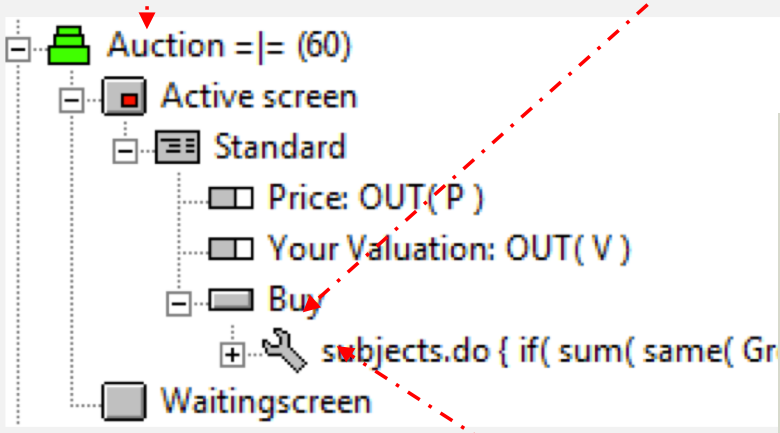

```
P = 150;
R = 80;
later ( 3 ) repeat
{
    P = P - 10;
    if(P<R)
    {
        P = R;
    }
}
```

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

AUCTION STAGE

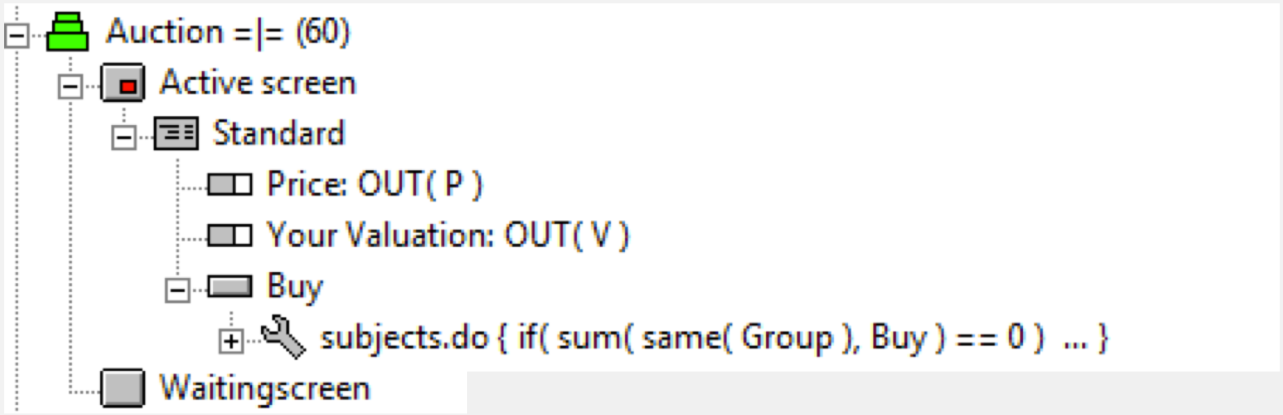


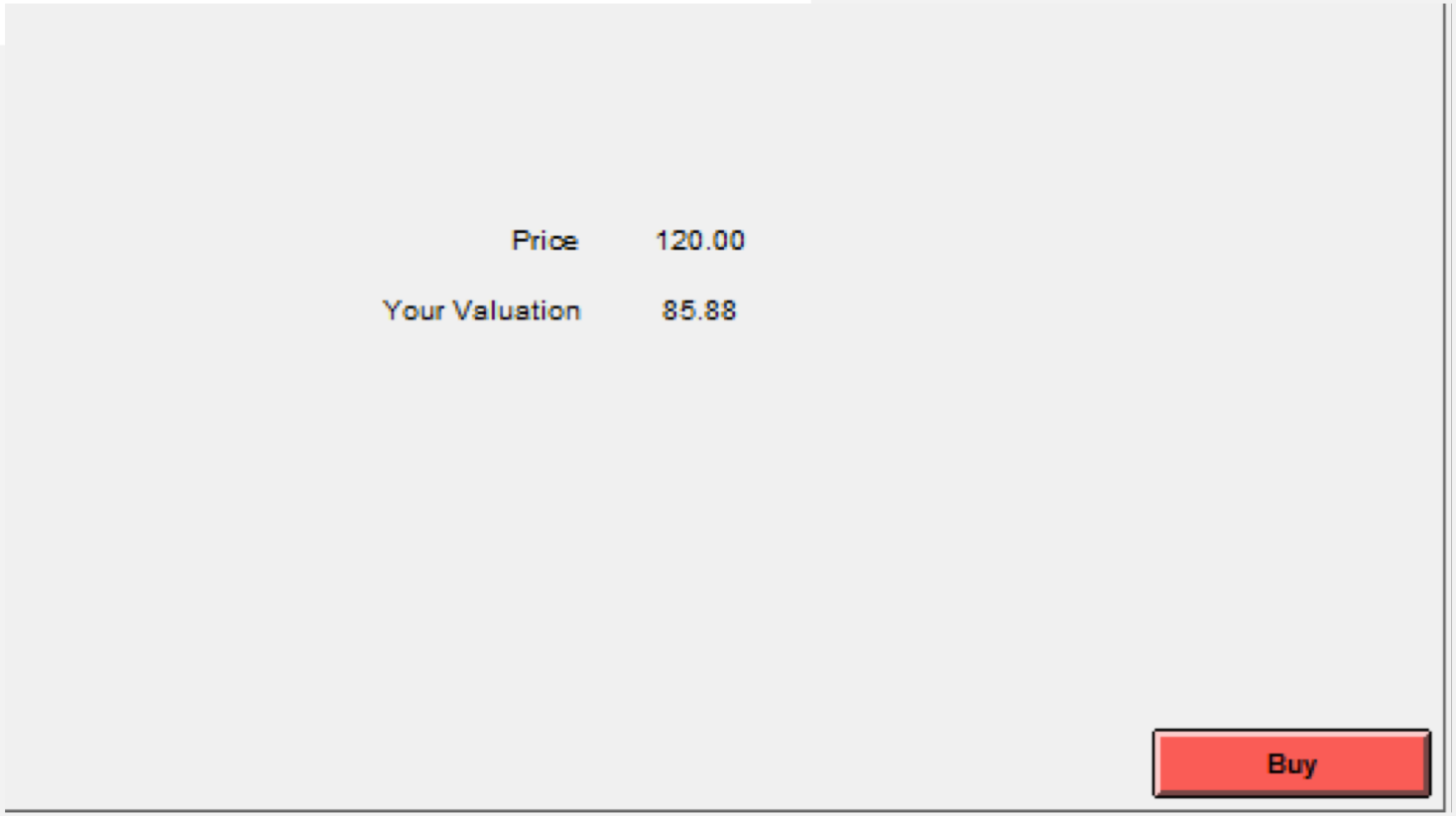




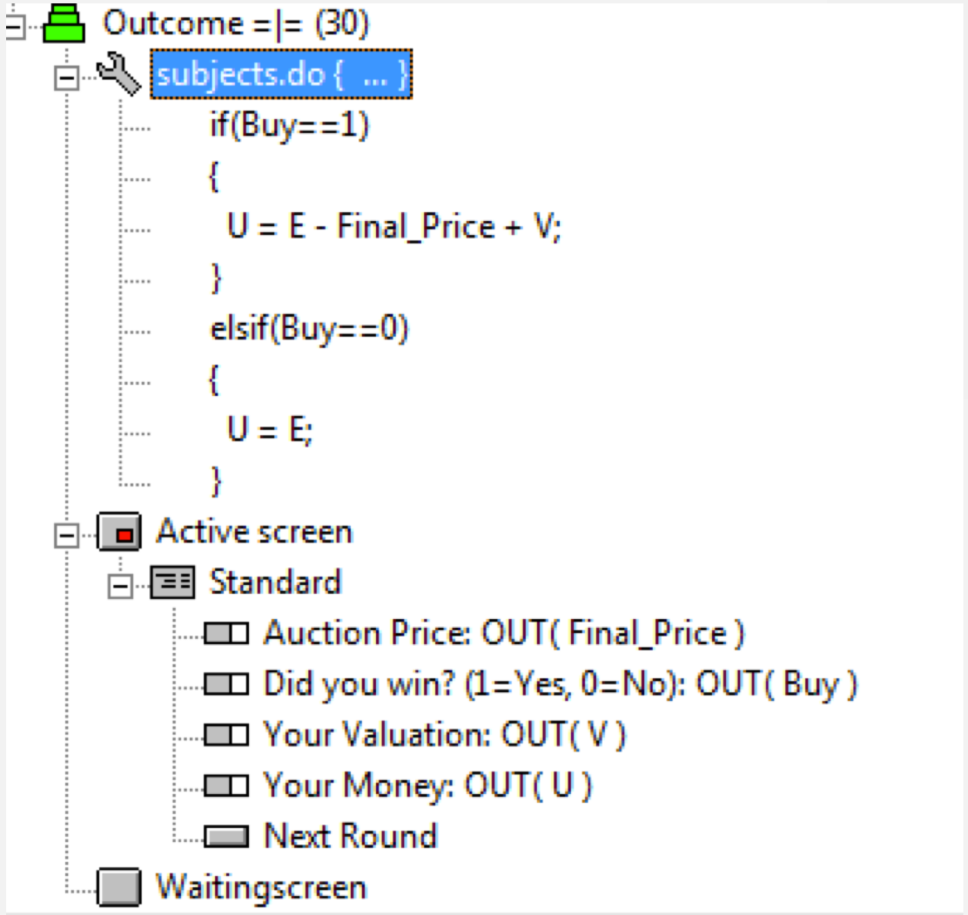

```
if(sum(same(Group,Buy)==0)
{
    Buy = 1;
    subjects.do{
                    if( same( Group ) )
                        {
                         LeaveStage = 1;
                         Final_Price = P;
                        }
                    }
}
```

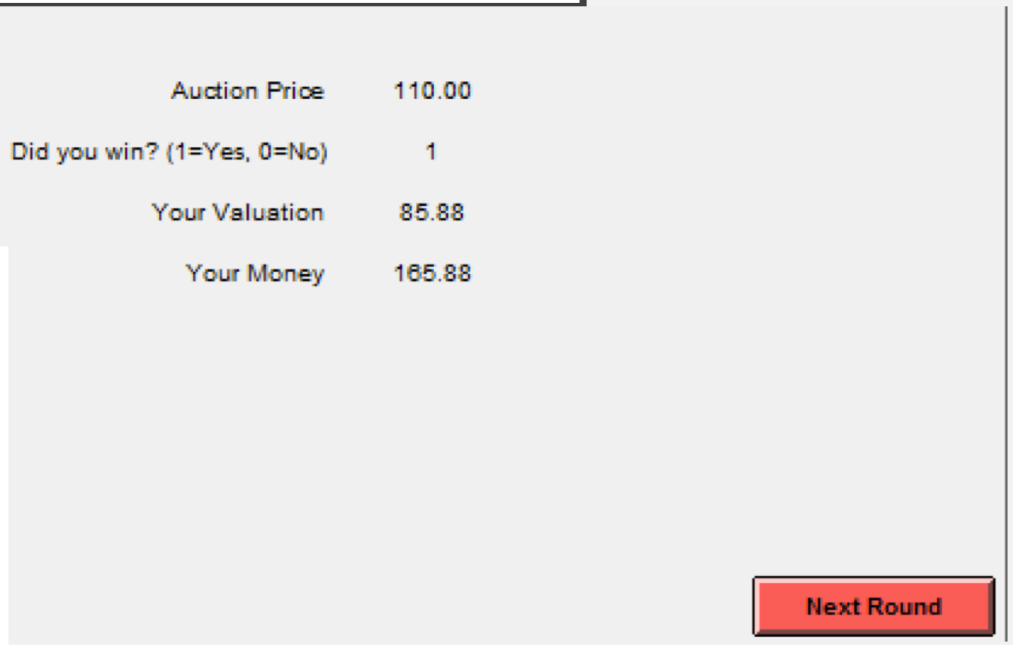
AUCTION STAGE





RESULT STAGE





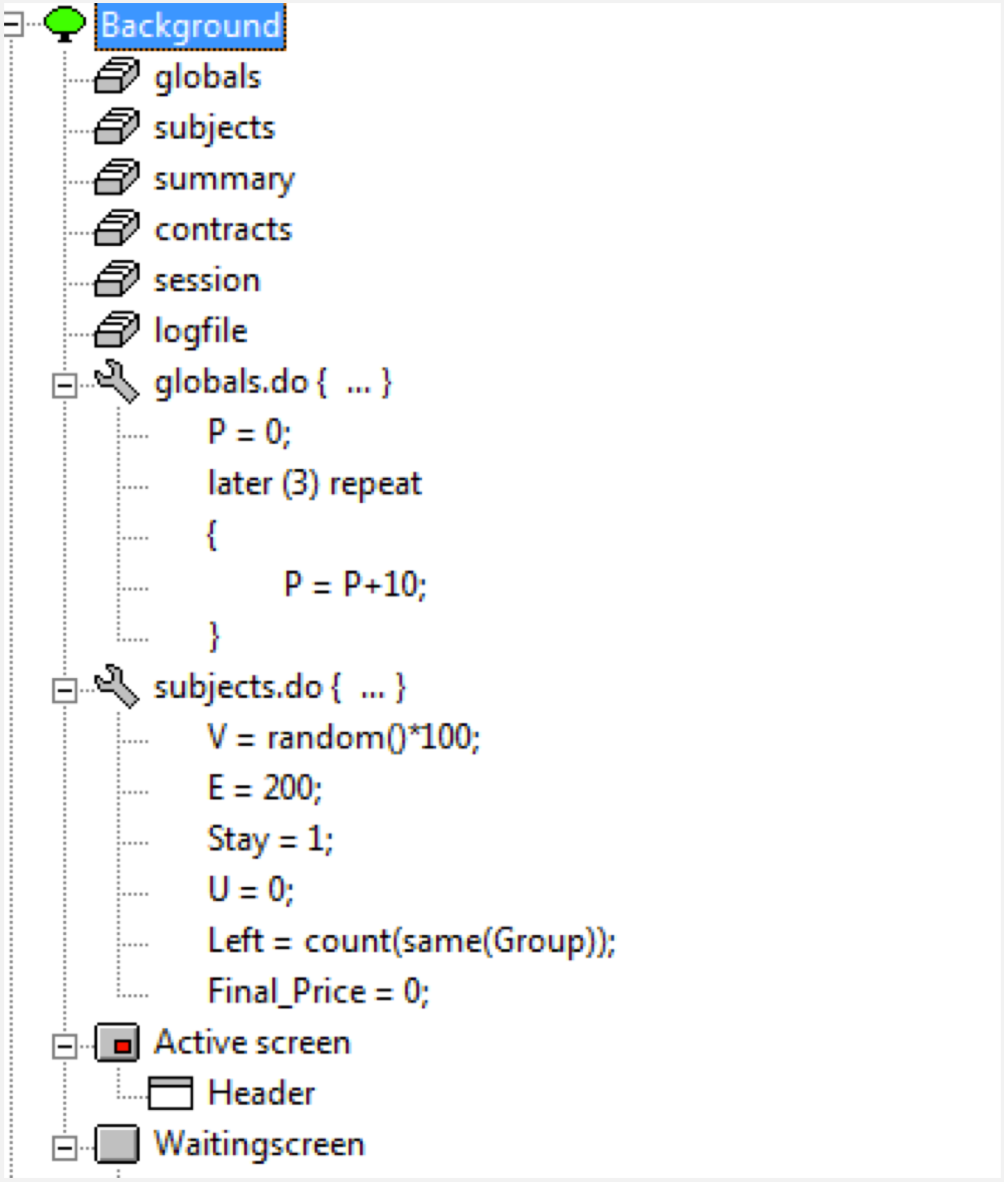
CLASS EXERCISE III

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)

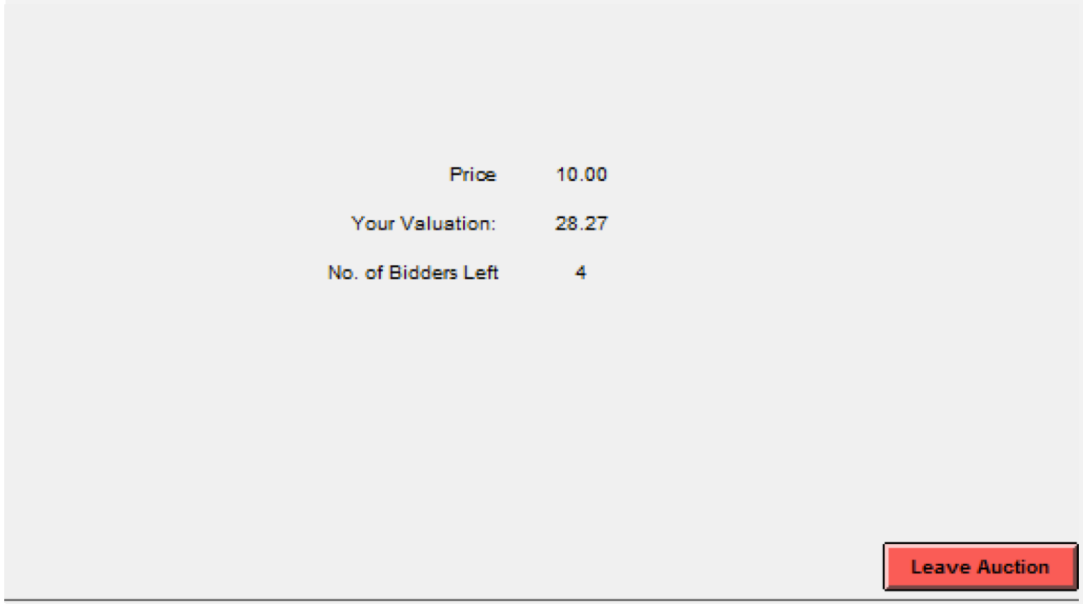


POSSIBLE SOLUTION (AUCTION STAGE)

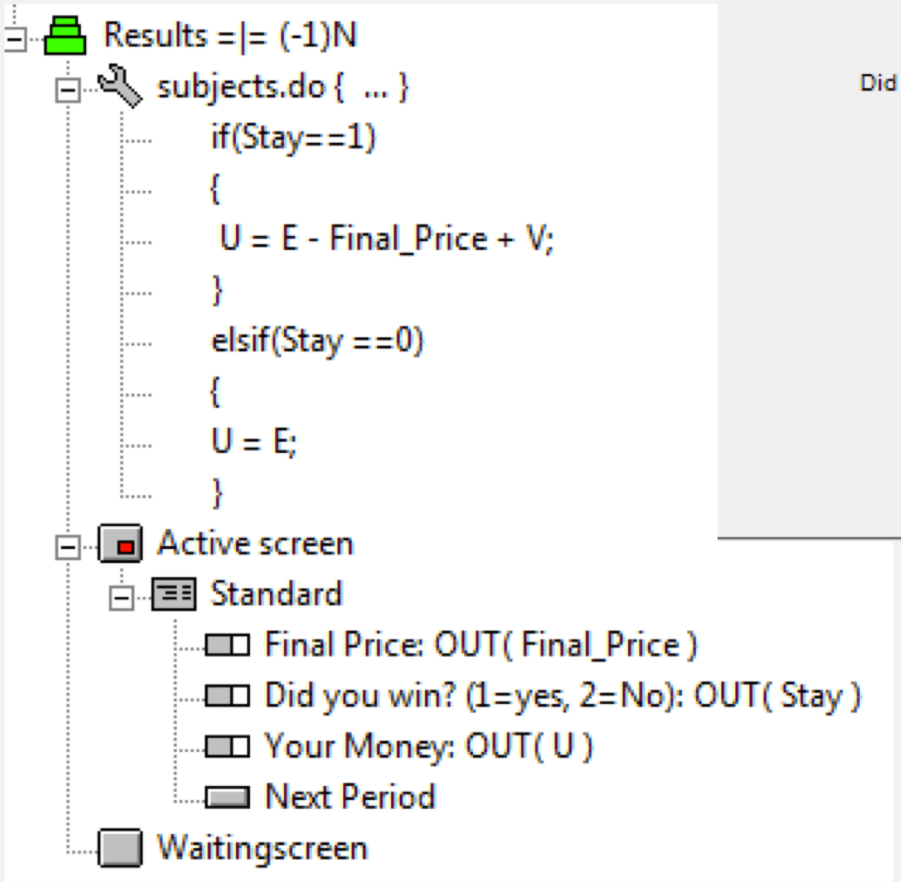
- Auction Stage =|= (30)
  - Active screen
    - Standard
      - Price: OUT(P)
      - Your Valuation:: OUT(V)
      - No. of Bidders Left: OUT(Left)
      - Leave Auction
        - subjects.do { ... }

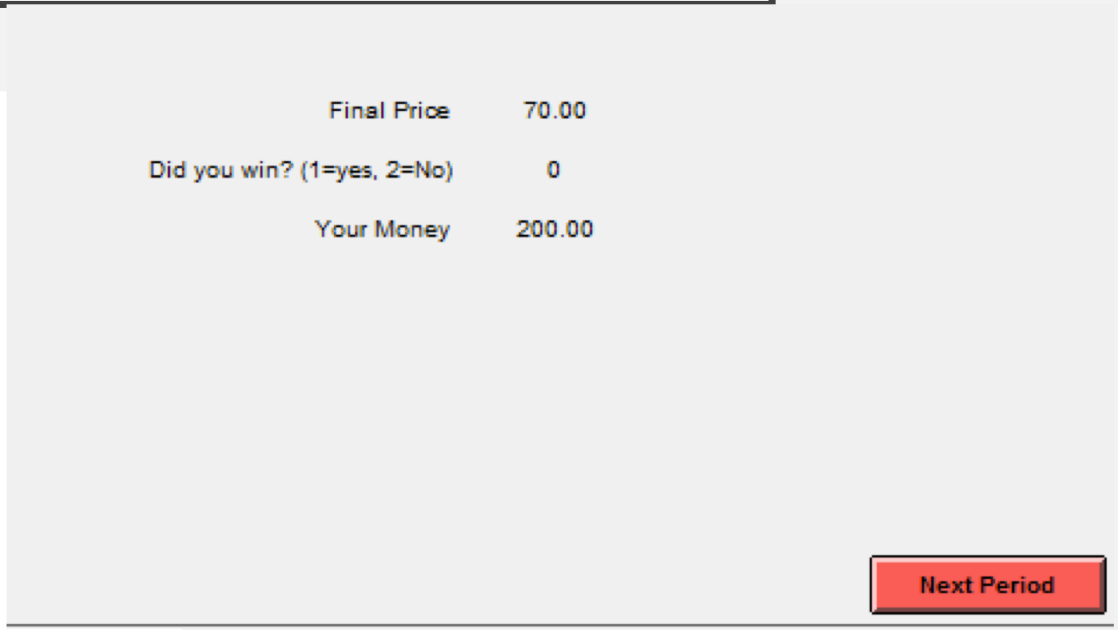
```
if(count(same(Group) & Stay==1) > 1)
{
 Stay = 0;
 subjects.do{
  if(same(Group))
  {
   Left = Left -1;
  }
 }
}
subjects.do{
if(count(same(Group) & Stay==1) == 1)
{
 subjects.do{
  if(same(Group))
 {
  LeaveStage = 1;
  Final_Price = P;
 }
 }
}
}
```

  - Waitingscreen



POSSIBLE SOLUTION (RESULT STAGE)





LESSON PLAN

Day III

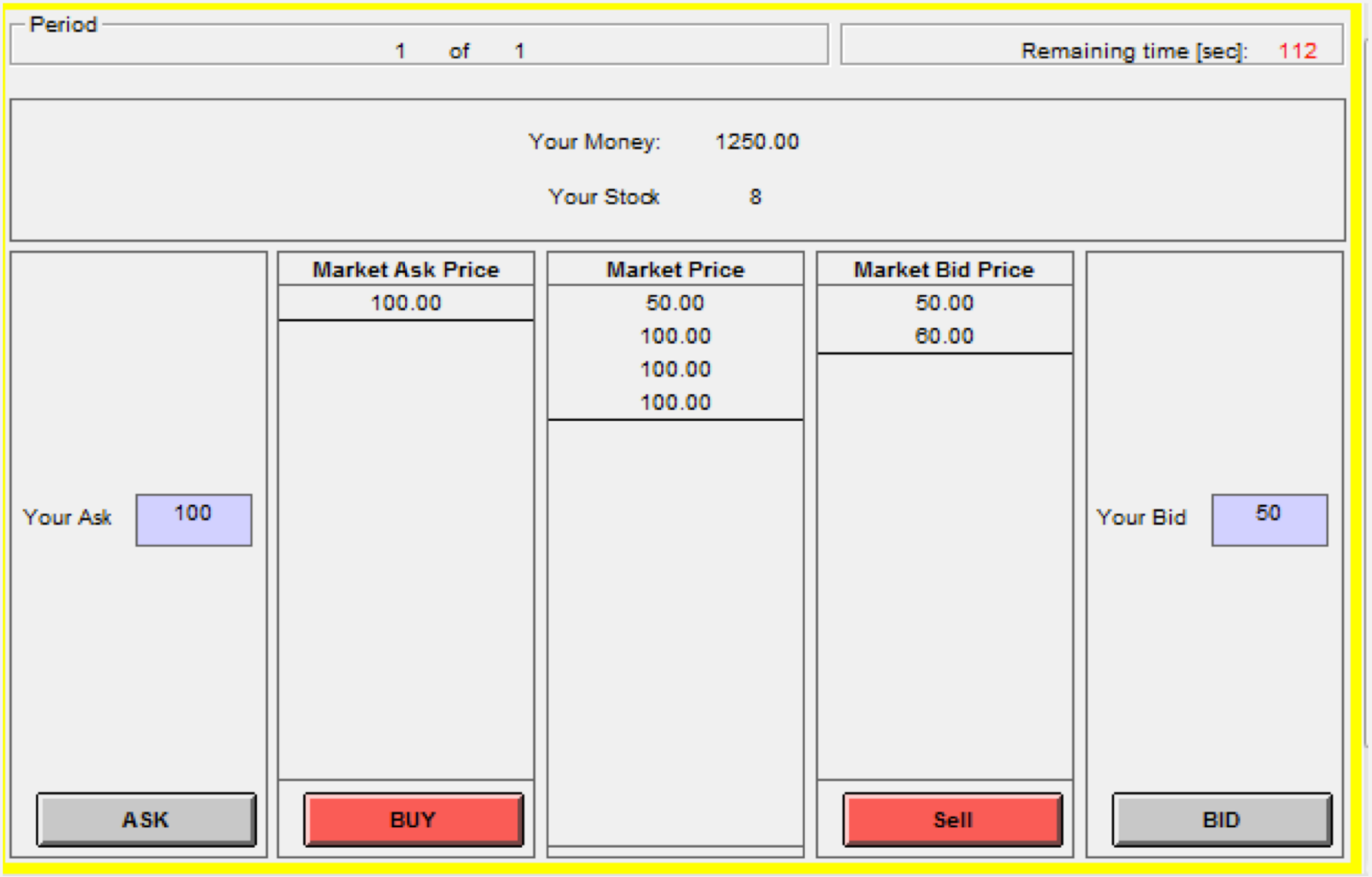
- **Example VI: Continuous Double Auction**
  - Introduction to the Contract table
- **Example VII: Random Stopping Public Goods Game**
  - Creating *infinite* length games
- **Example VIII: Complex Move games**
  - Inserting Figures / Videos
  - Designing complex sequential move formats
- **Example 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.

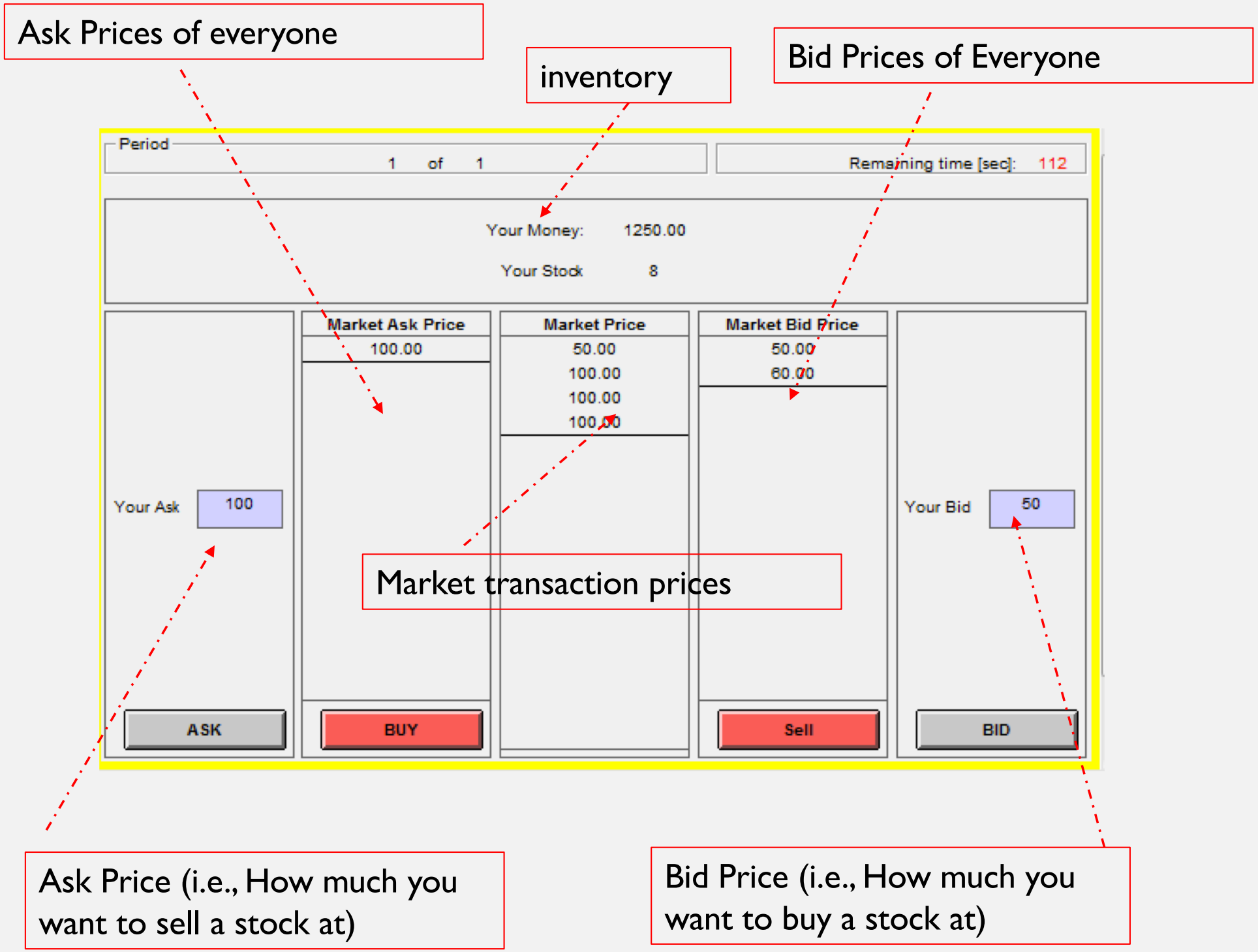
EXAMPLE VI

CONTINUOUS DOUBLE AUCTION MARKET

DESIGN

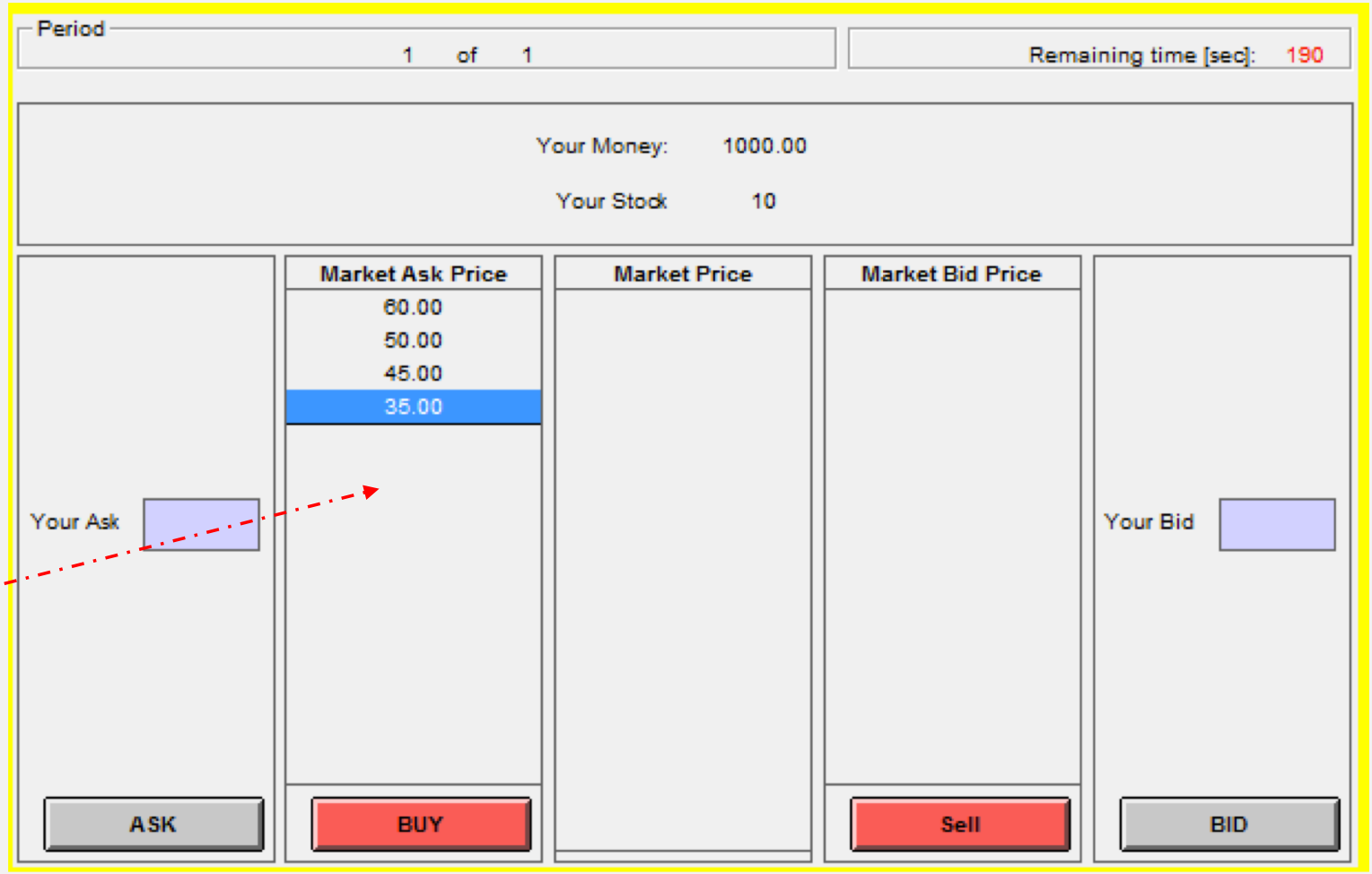
- One-Period market involving N=4 traders
- Each trader endowed with $1000 and 10 assets
- Trade facilitated through continuous double auction




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
100.00
Market Price
50.00
100.00
100.00
100.00
Market Bid Price
50.00
60.00
Your Ask
100
Your Bid
50
Market transaction prices
ASK
BUY
Sell
BID
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	contractI D	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

Lowest Ask price below

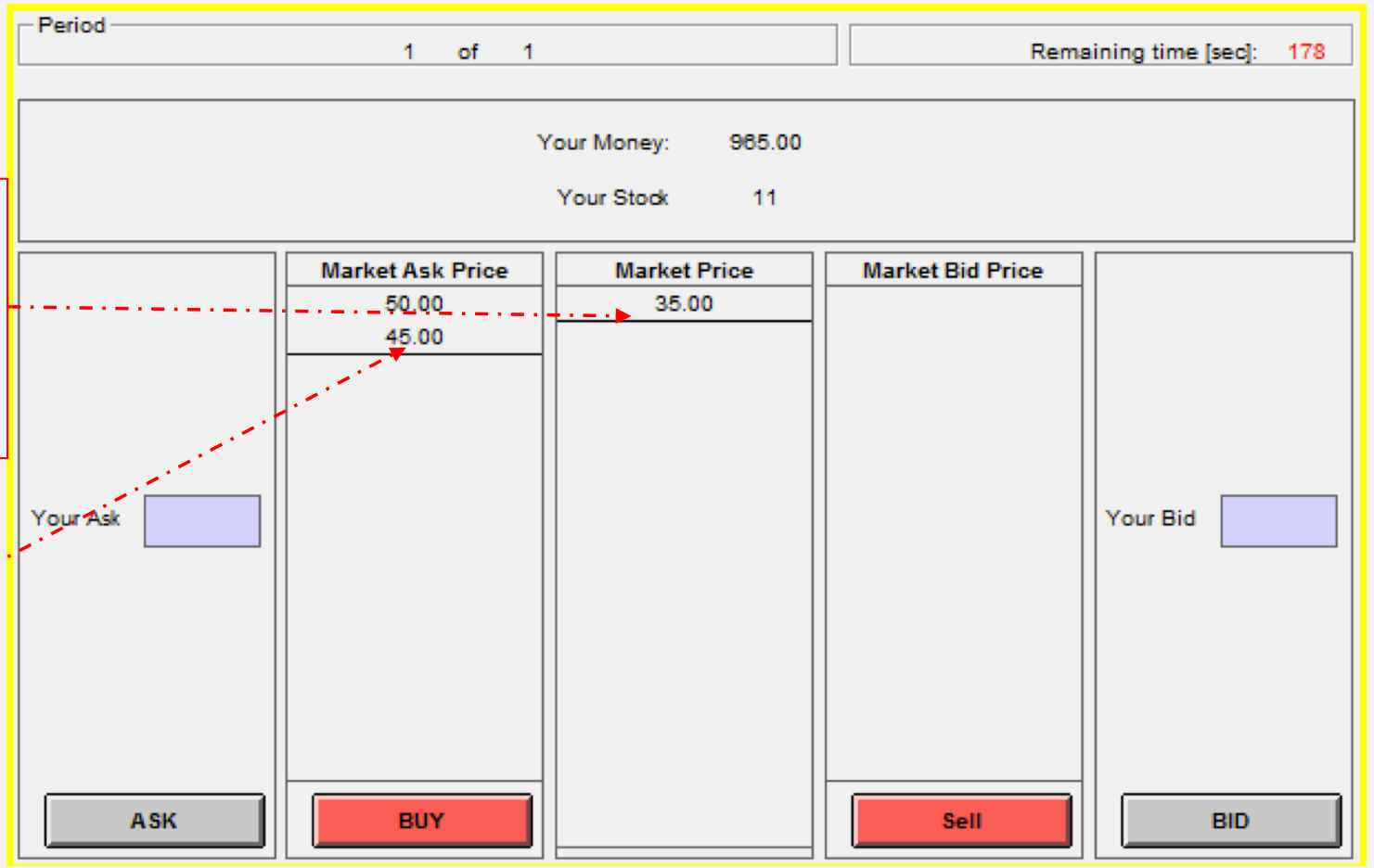


Period	Seller	Buyer	Maker	P	Traded	contractID	tradeID
1	3	-1	3	50	0	1	0
1	3	-1	3	45	0	2	0
1	4	-2	4	60	0	3	0
1	4	1	4	35	1	4	1

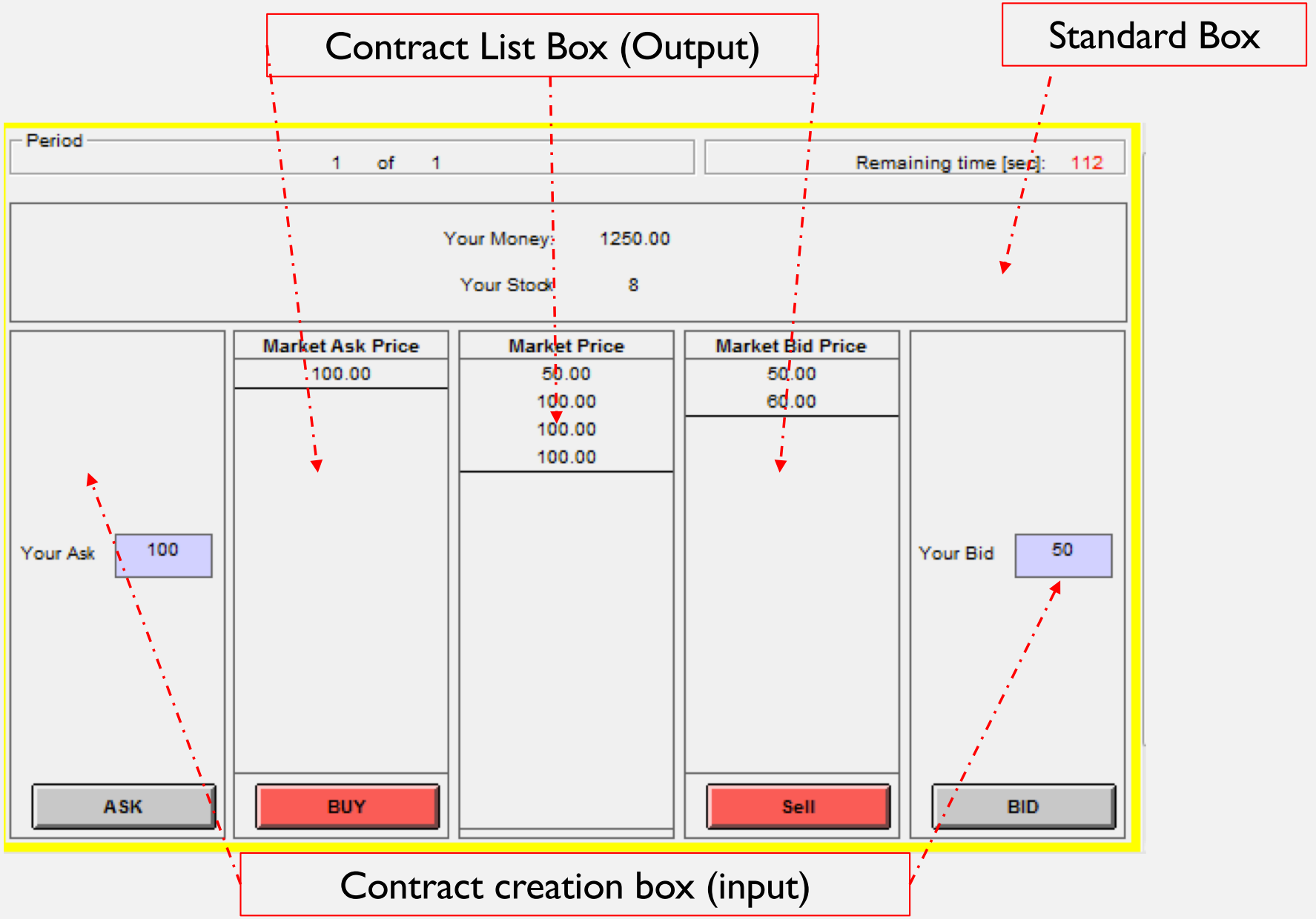
Transaction Price Updated

Subjects' inventory updated

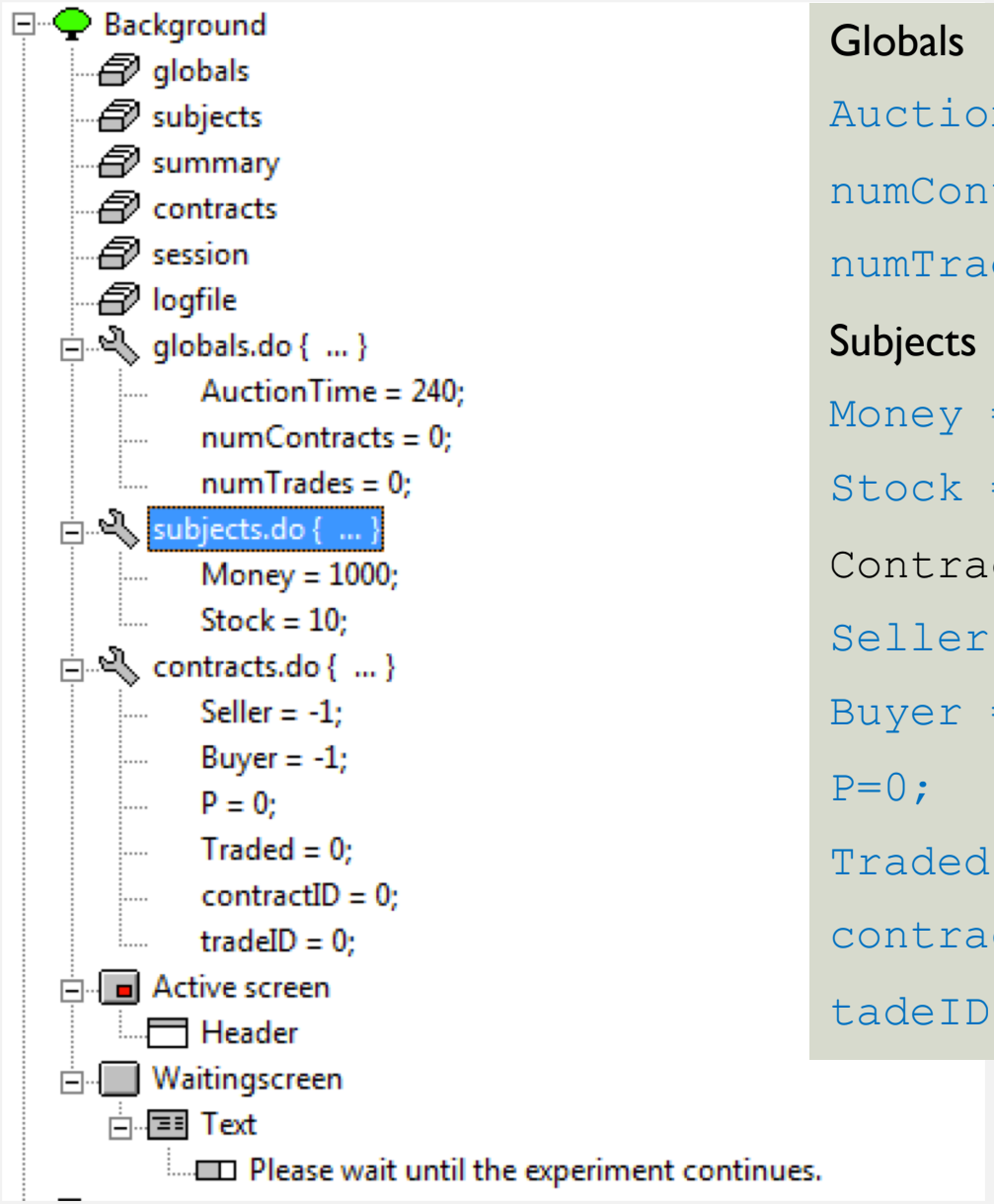
Ask price from Seller 4 removed



TYPES OF CONTRACT BOXES



BACKGROUND STAGE (initial values)



Globals

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

Subjects

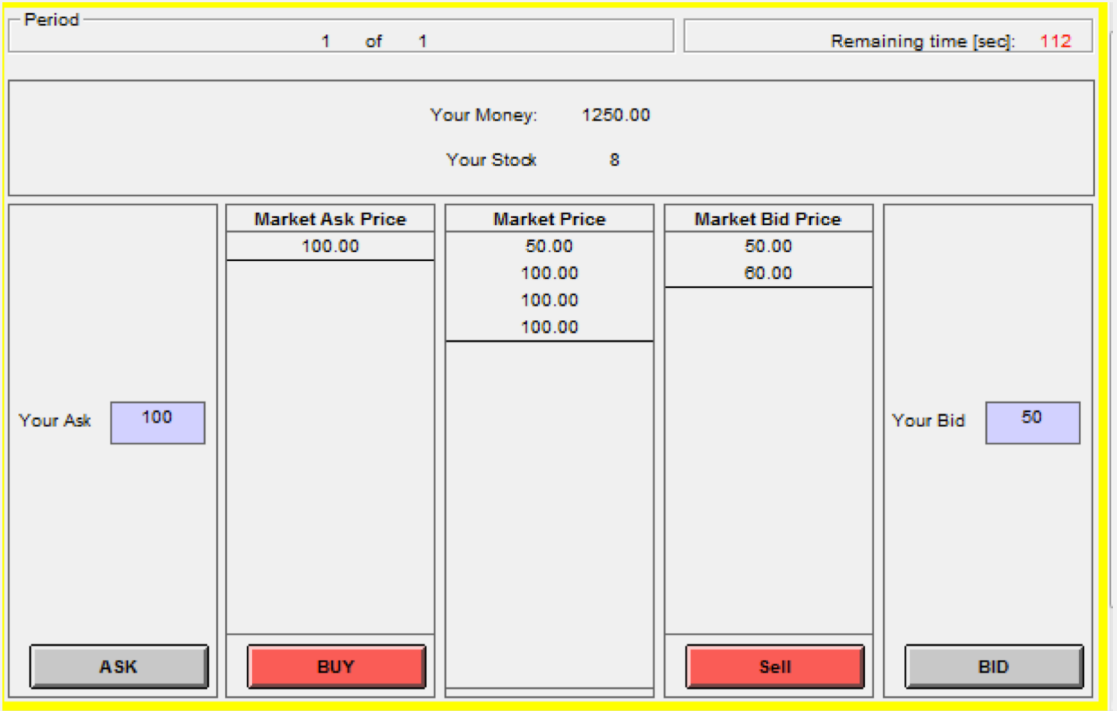
```
Money = 1000;
Stock = 10;
```

Contracts

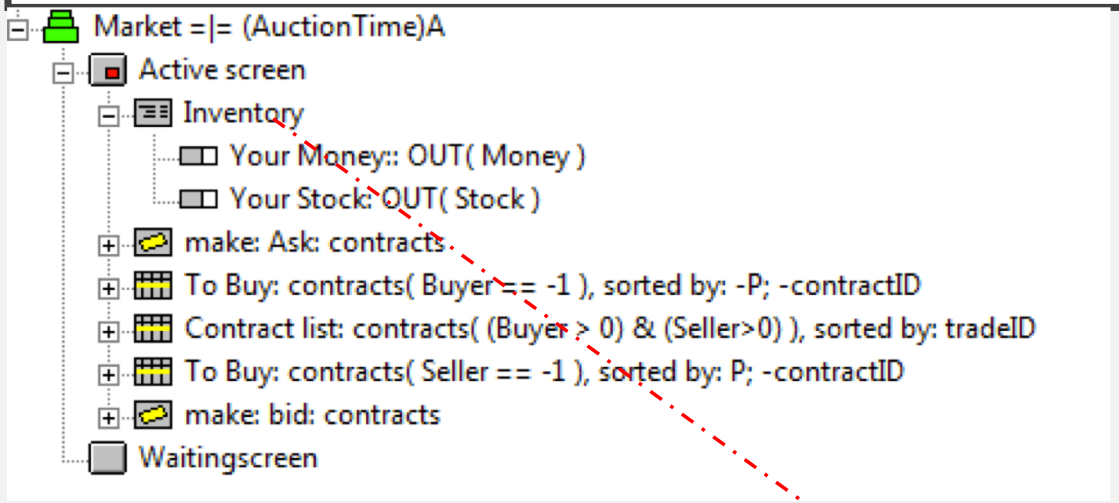
```
Seller = -1;
Buyer = -1;
P=0;
Traded = 0;
contractID = 0
tadeID =0
```

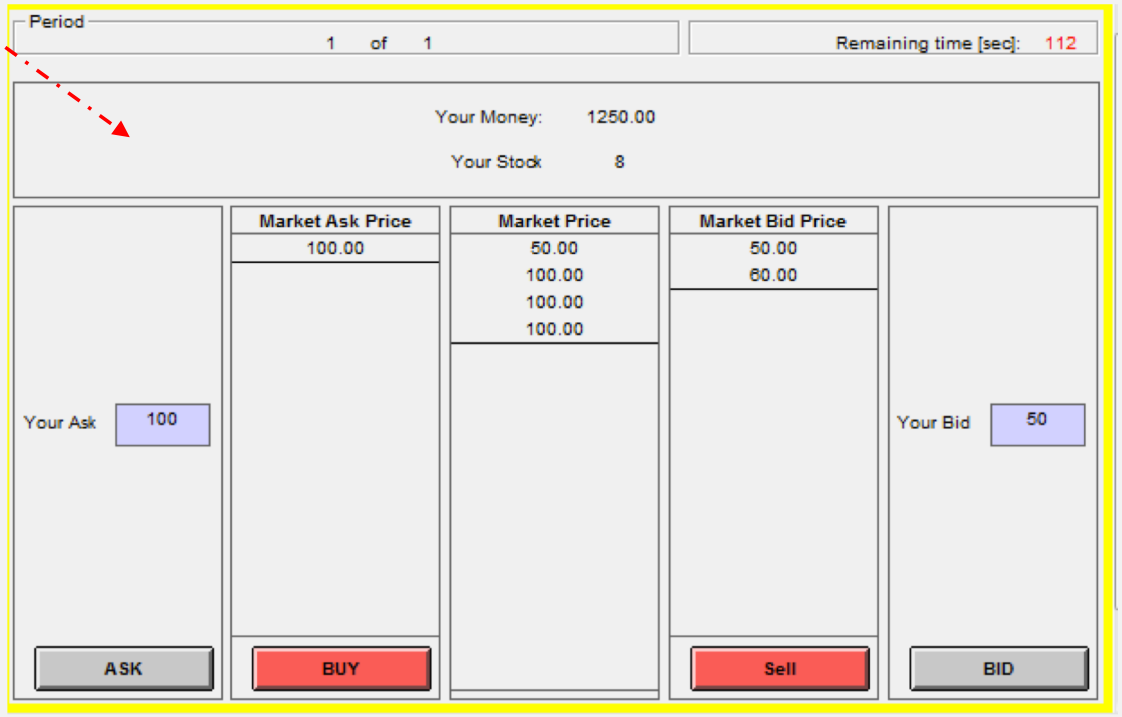
AUCTION STAGE
(initial values)

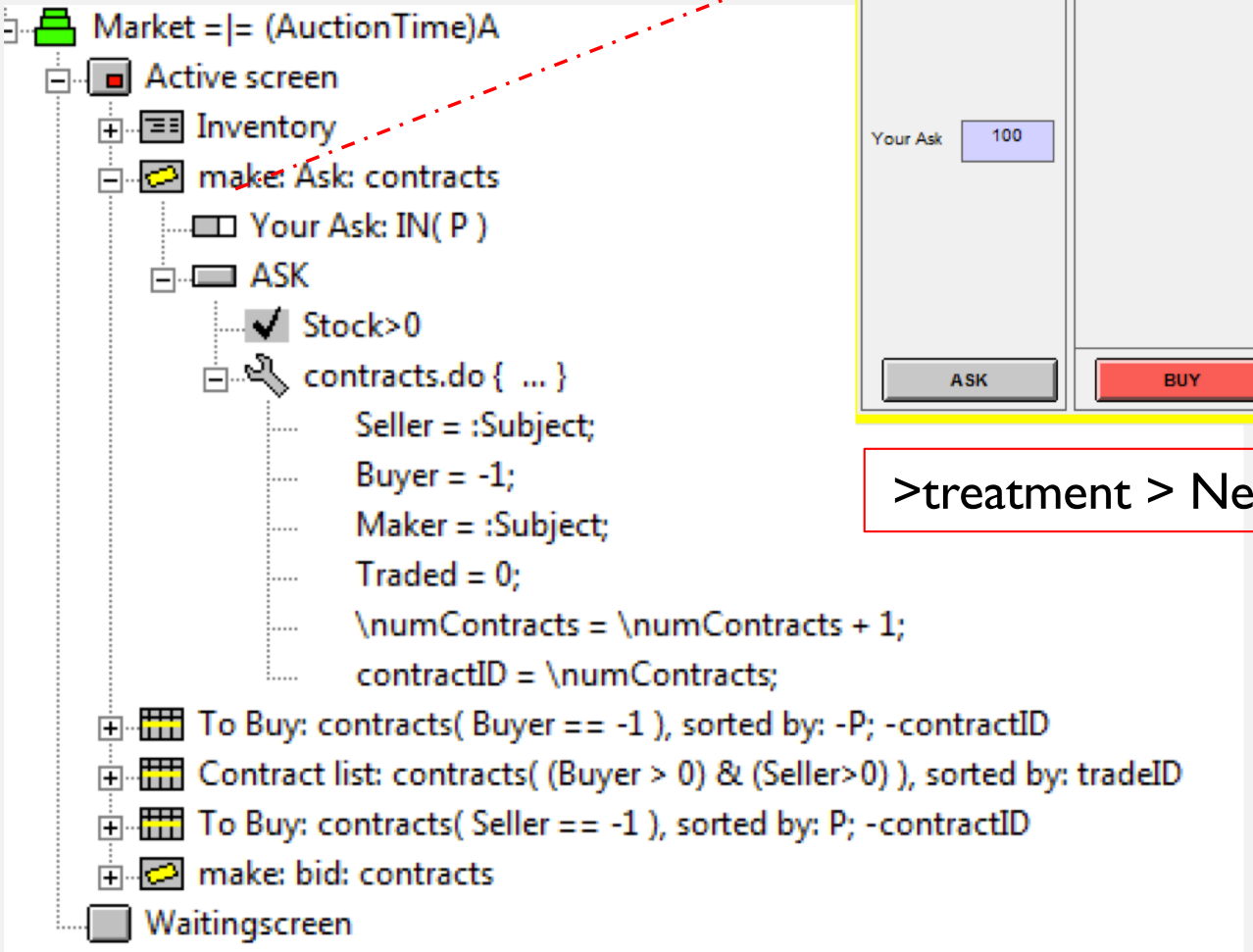
- Market =|= (AuctionTime)A
  - Active screen
    - Inventory
      - Your Money:: OUT(Money)
      - Your Stock: OUT(Stock)
    - make: Ask: contracts
    - To Buy: contracts(Buyer == -1), sorted by: -P; -contractID
    - Contract list: contracts((Buyer > 0) & (Seller>0)), sorted by: tradeID
    - To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

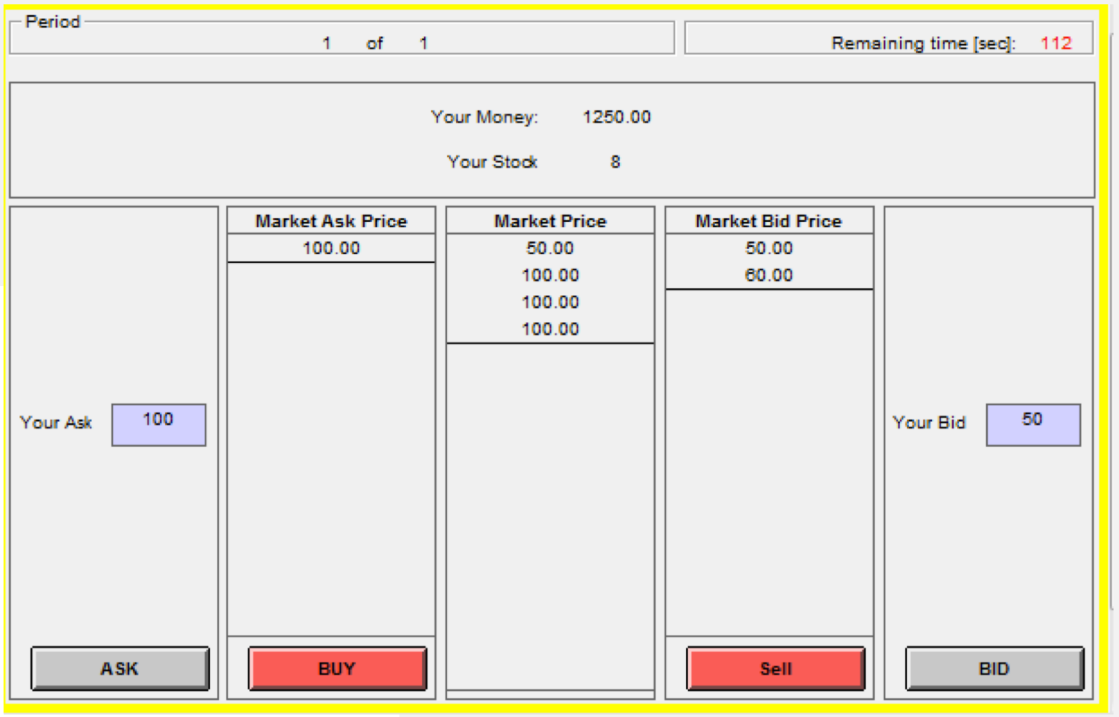


AUCTION STAGE
(initial values)



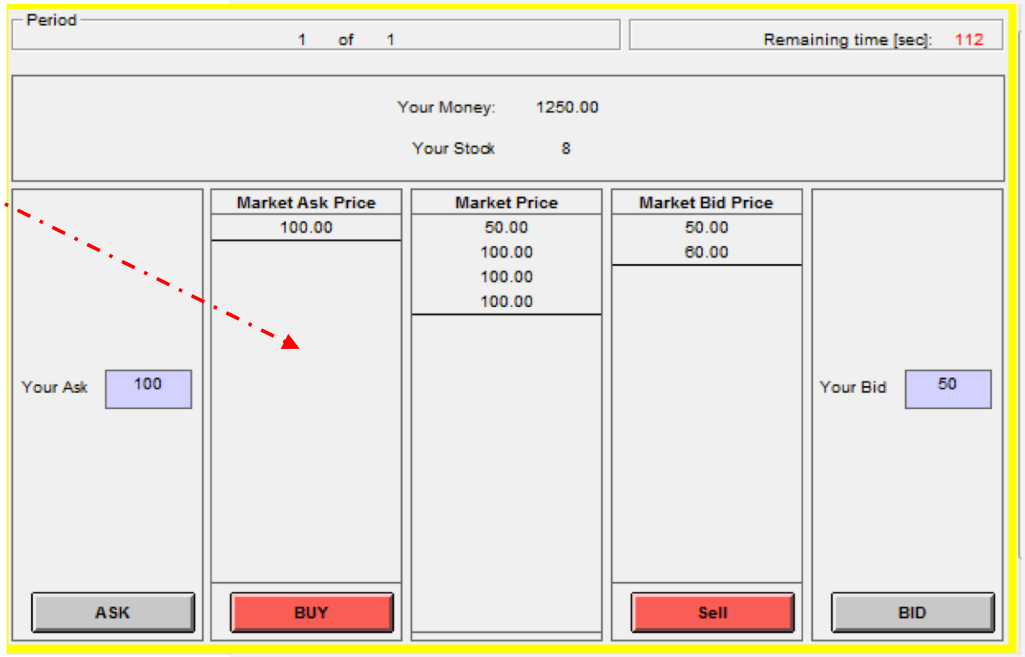






>treatment > New Box > Contract creation box

```
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;
              }
            }
```

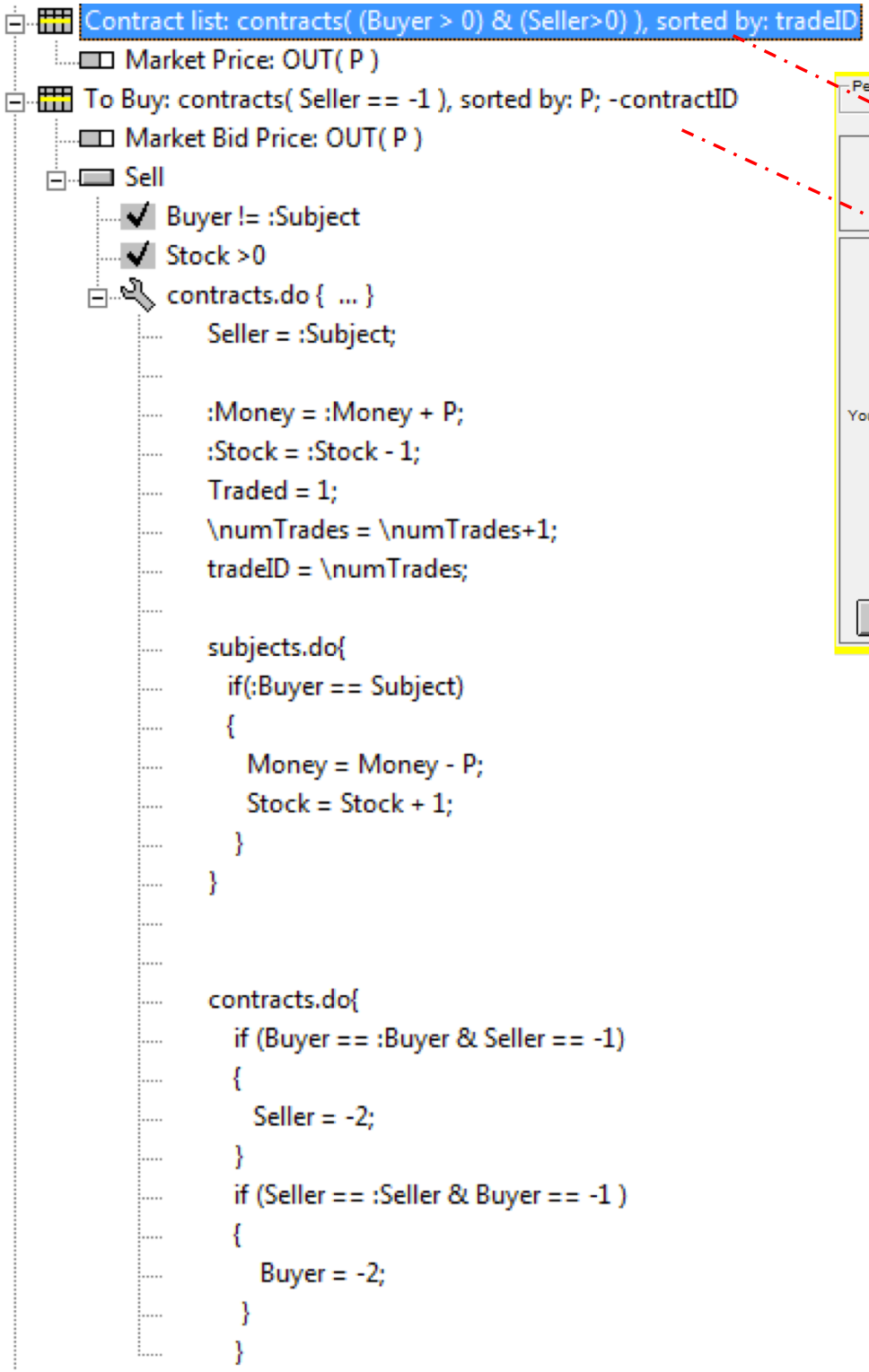


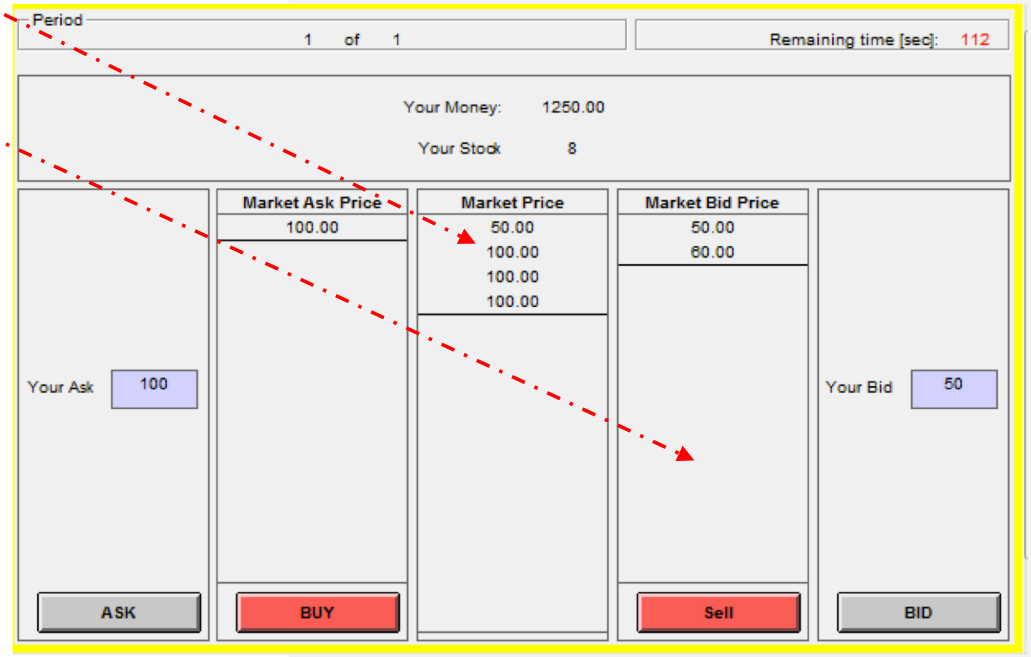
>treatment > New Box > Contract List box

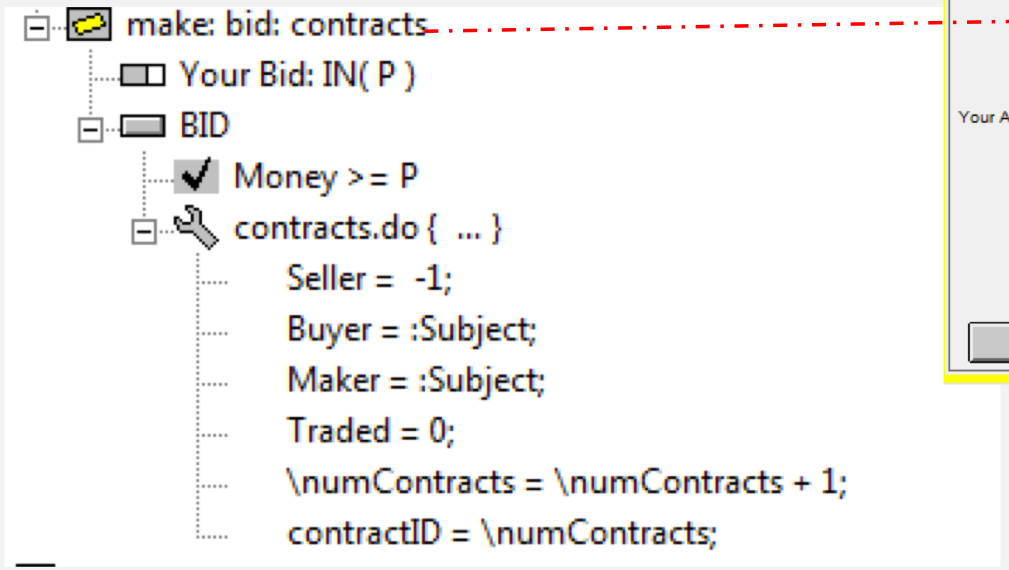
- If condition (Buyer==1)
- Sort (-P; -contractID;)

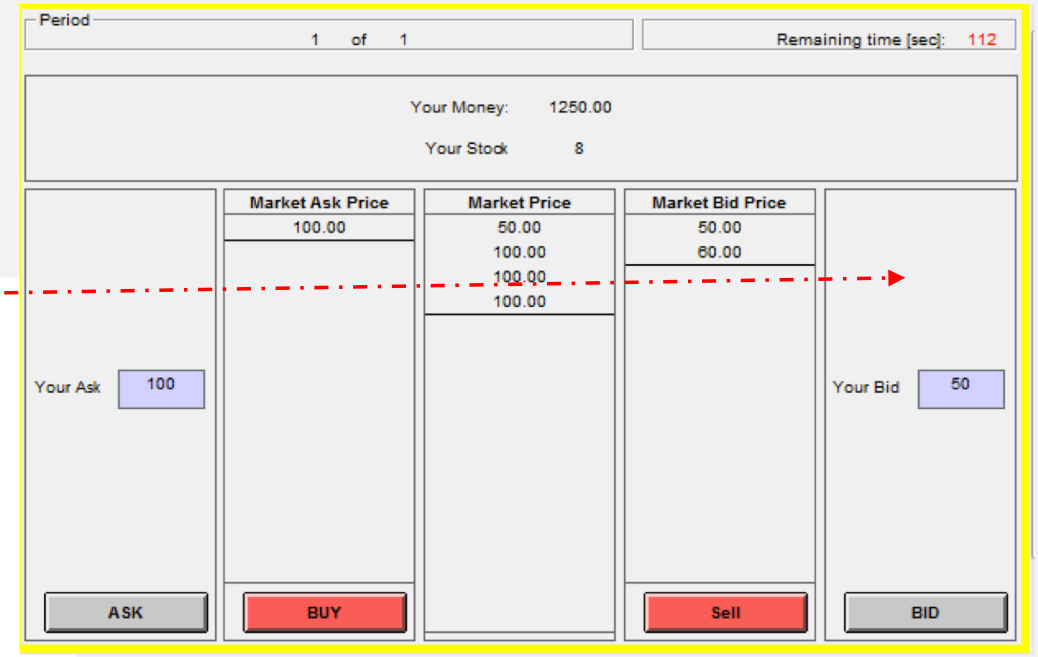
```
            contracts.do{
              if (Buyer == :Buyer & Seller == -1)
              {
                Seller = -2;
              }
              if (Seller == :Seller & Buyer == -1 )
              {
                 Buyer = -2;
              }
```

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
100.00
Market Price
50.00
100.00
100.00
100.00
Market Bid Price
50.00
60.00
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
100.00
100.00
100.00
50.00
60.00
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 ½.

```
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;
    }
    elsif(rr>0.5)
    {
    RepeatTreatment = 1;
    }
```

Globals table

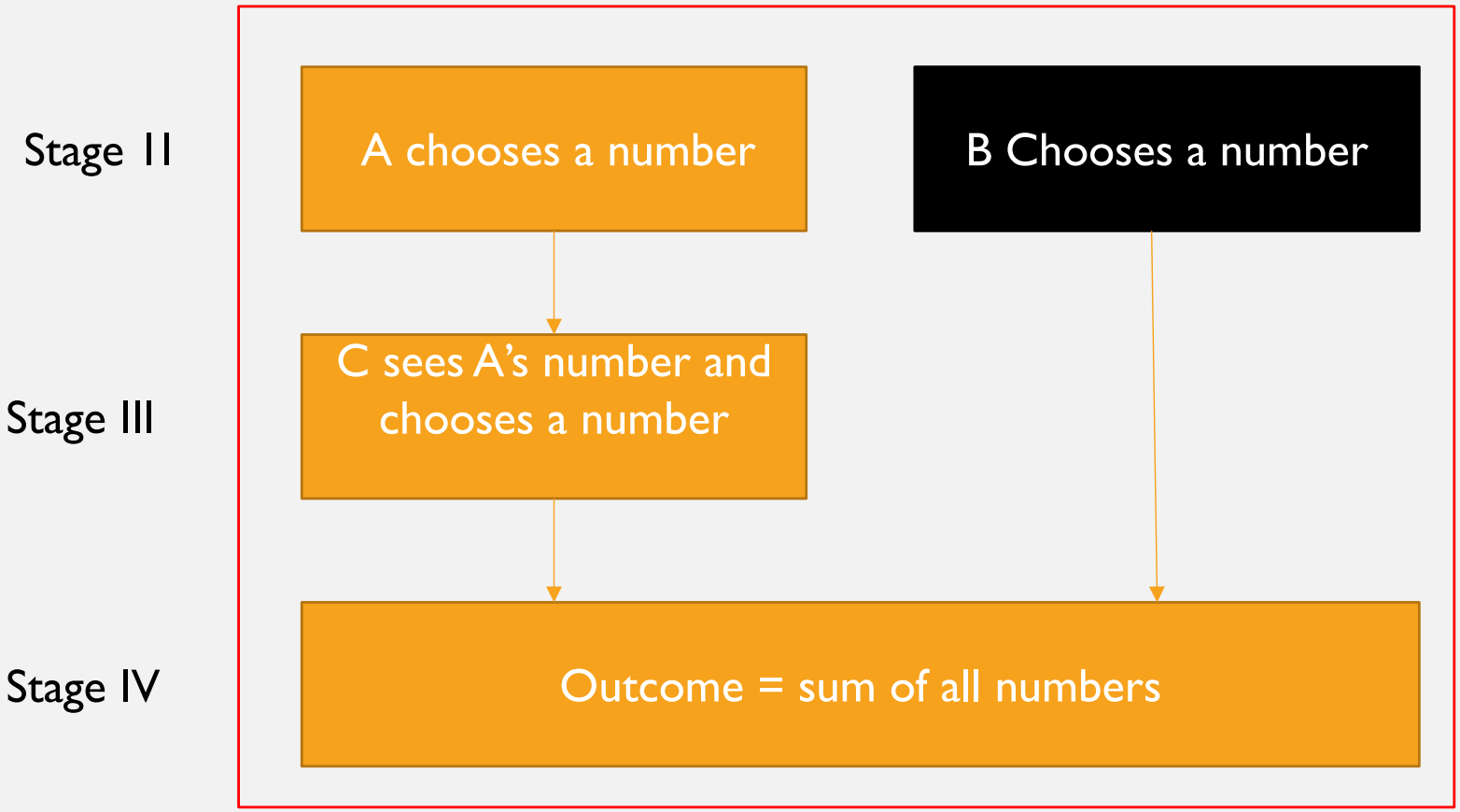
```
RepeatTreatment = 1 or 0;
```

EXAMPLE VIII

COMPLEX MOVES

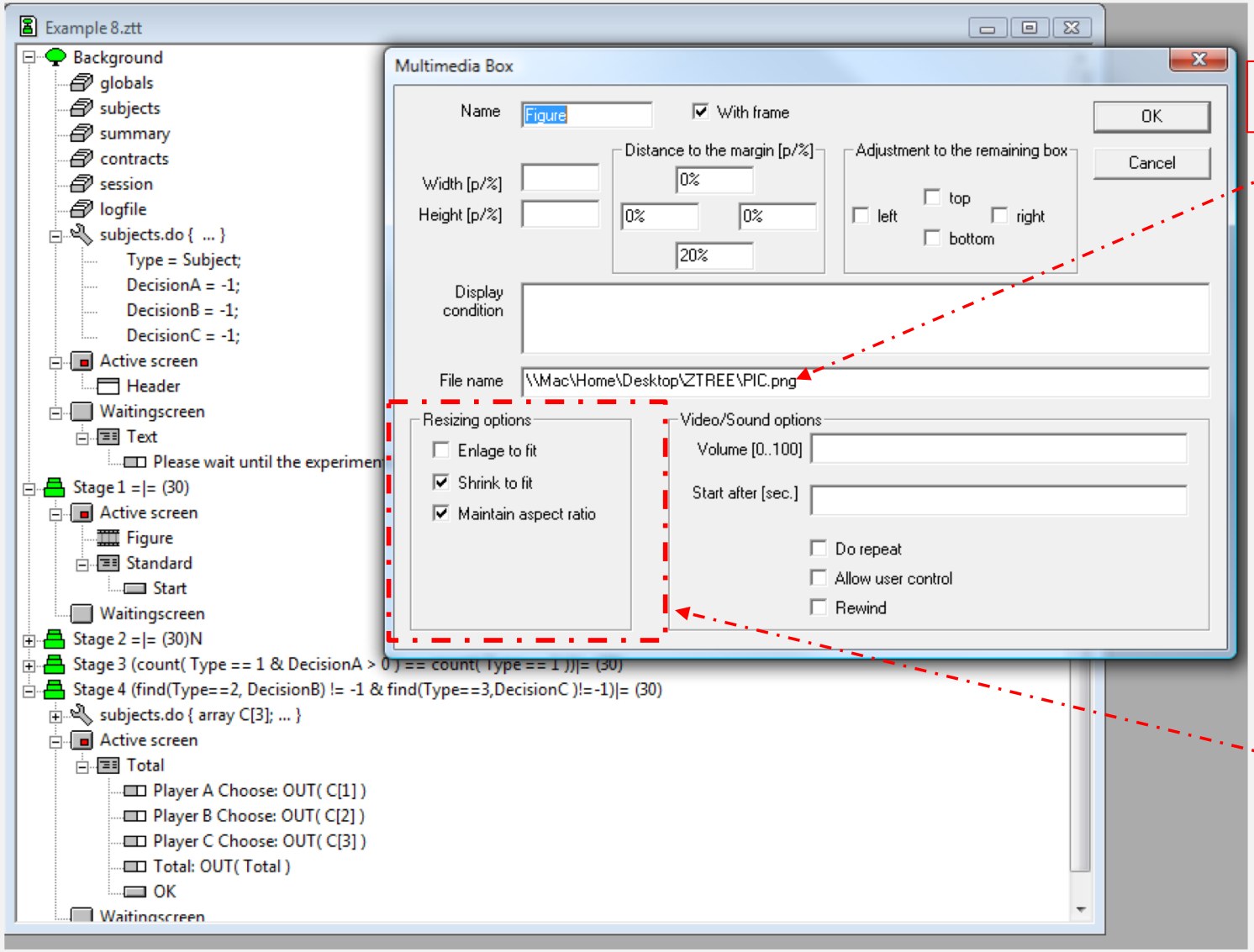
DESIGN

- 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 1.



STAGE 1

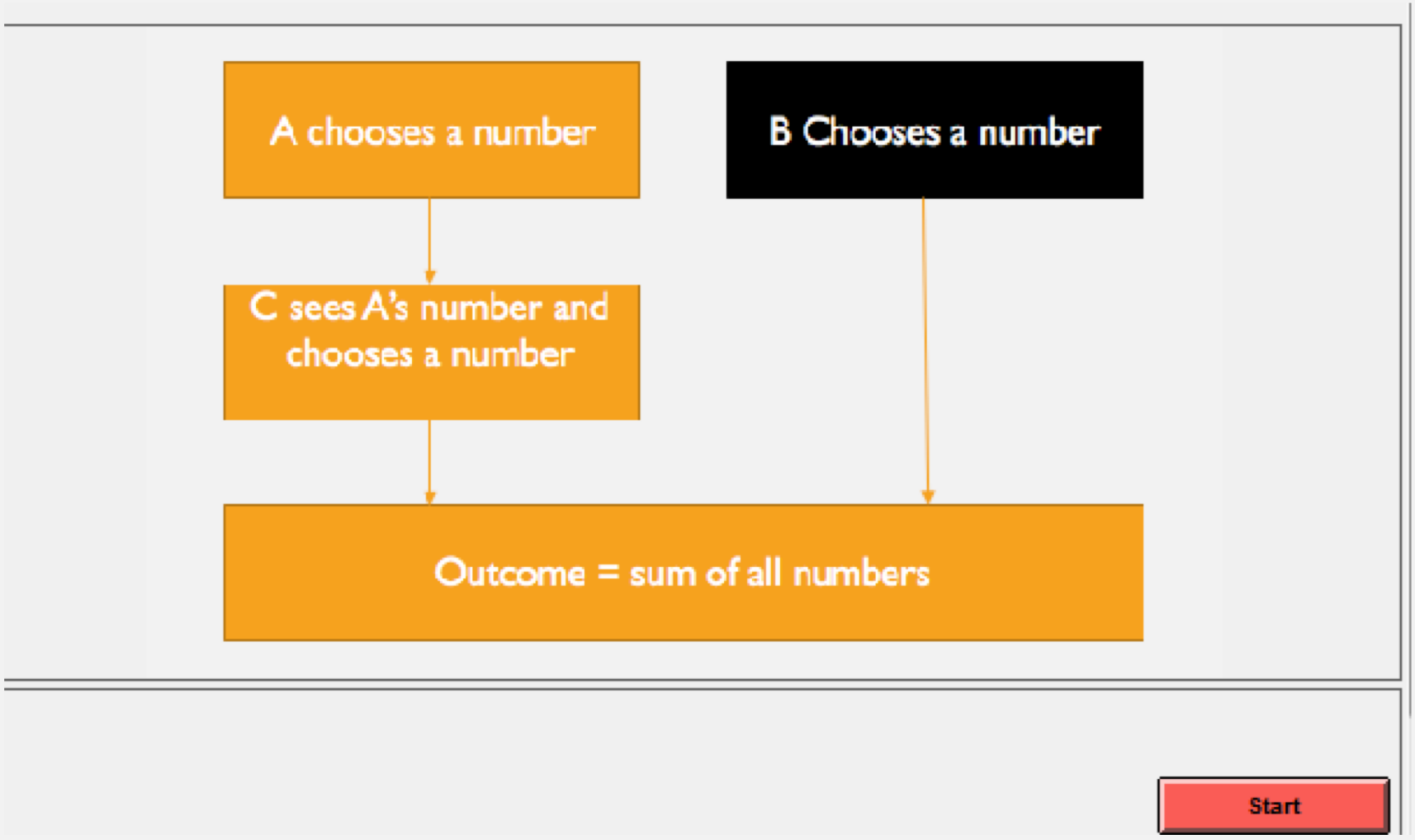
>treatment > New Box > New Multimedia box



Path location of file

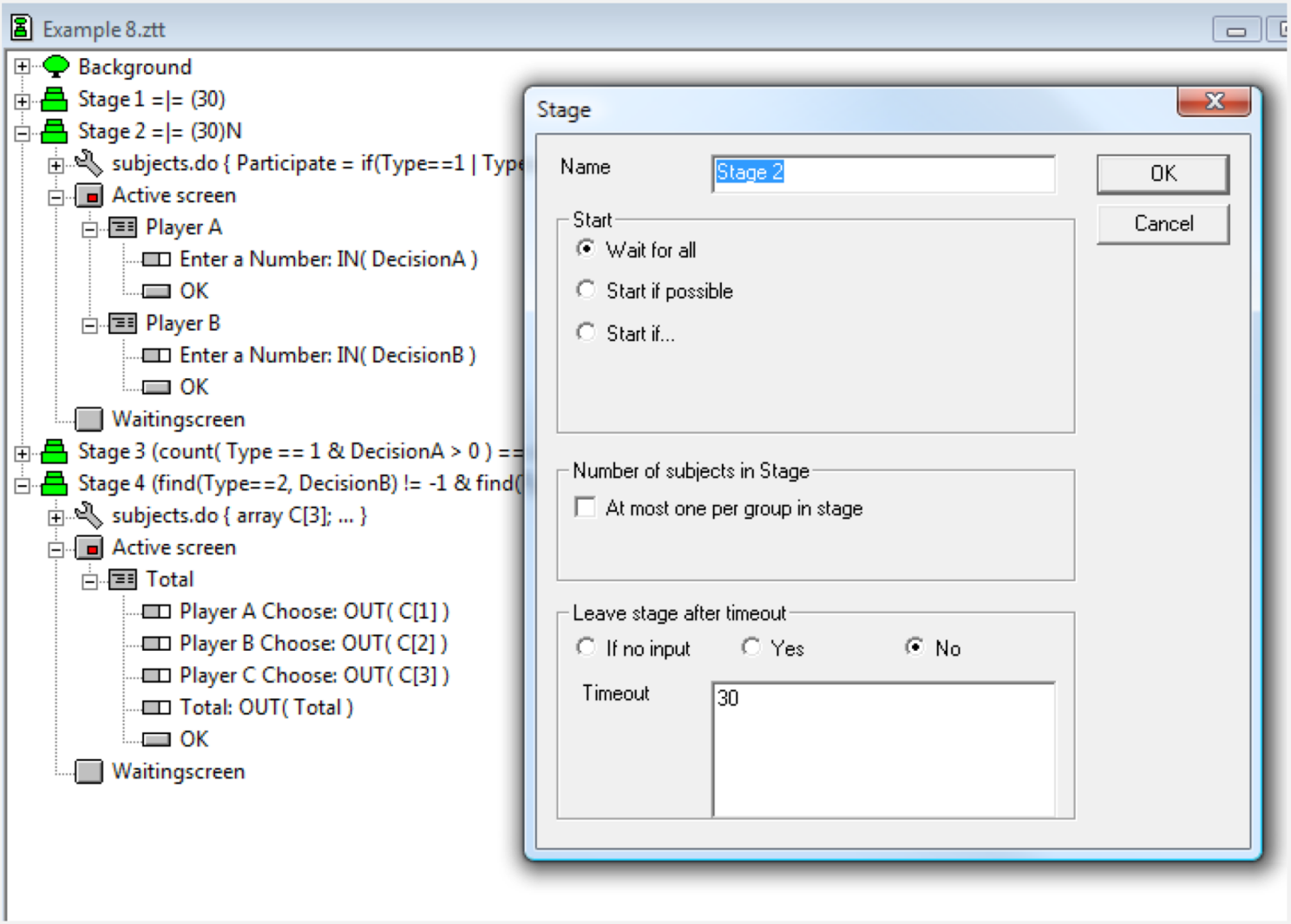
Manage distortions

STAGE 1


A chooses a number
B Chooses a number
C sees A's number and chooses a number
Outcome = sum of all numbers
Start

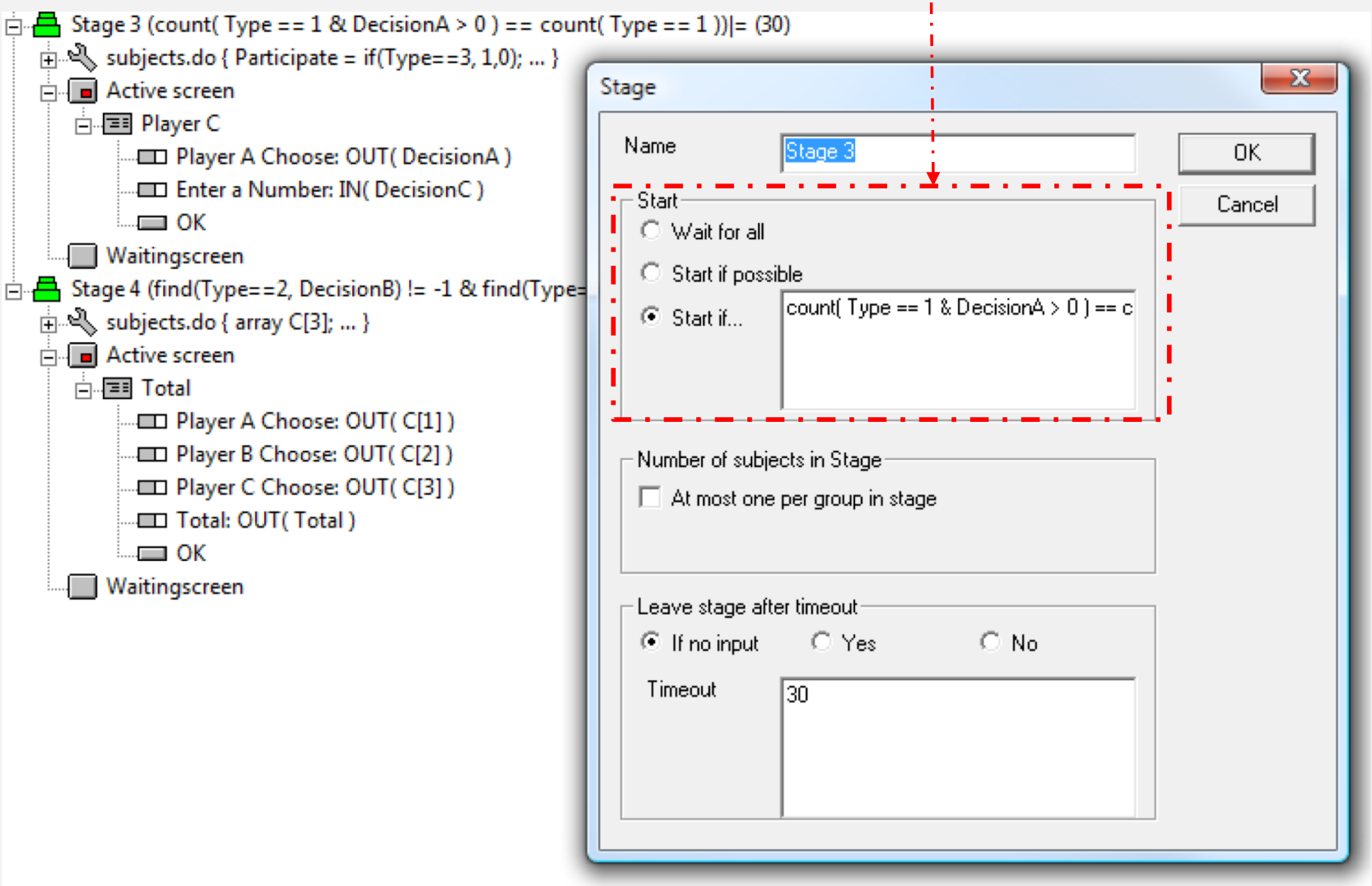
STAGE 11

As per normal



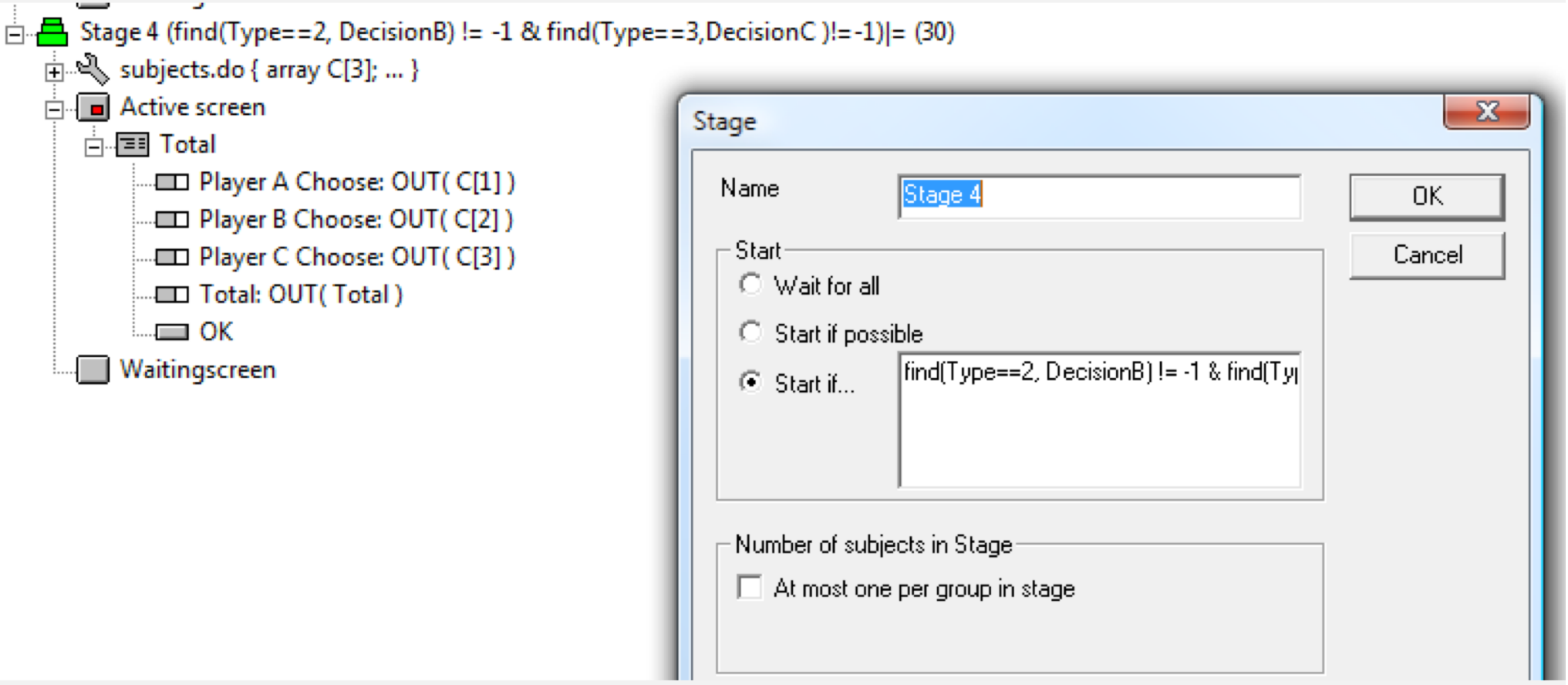
STAGE III

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



STAGE IV

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

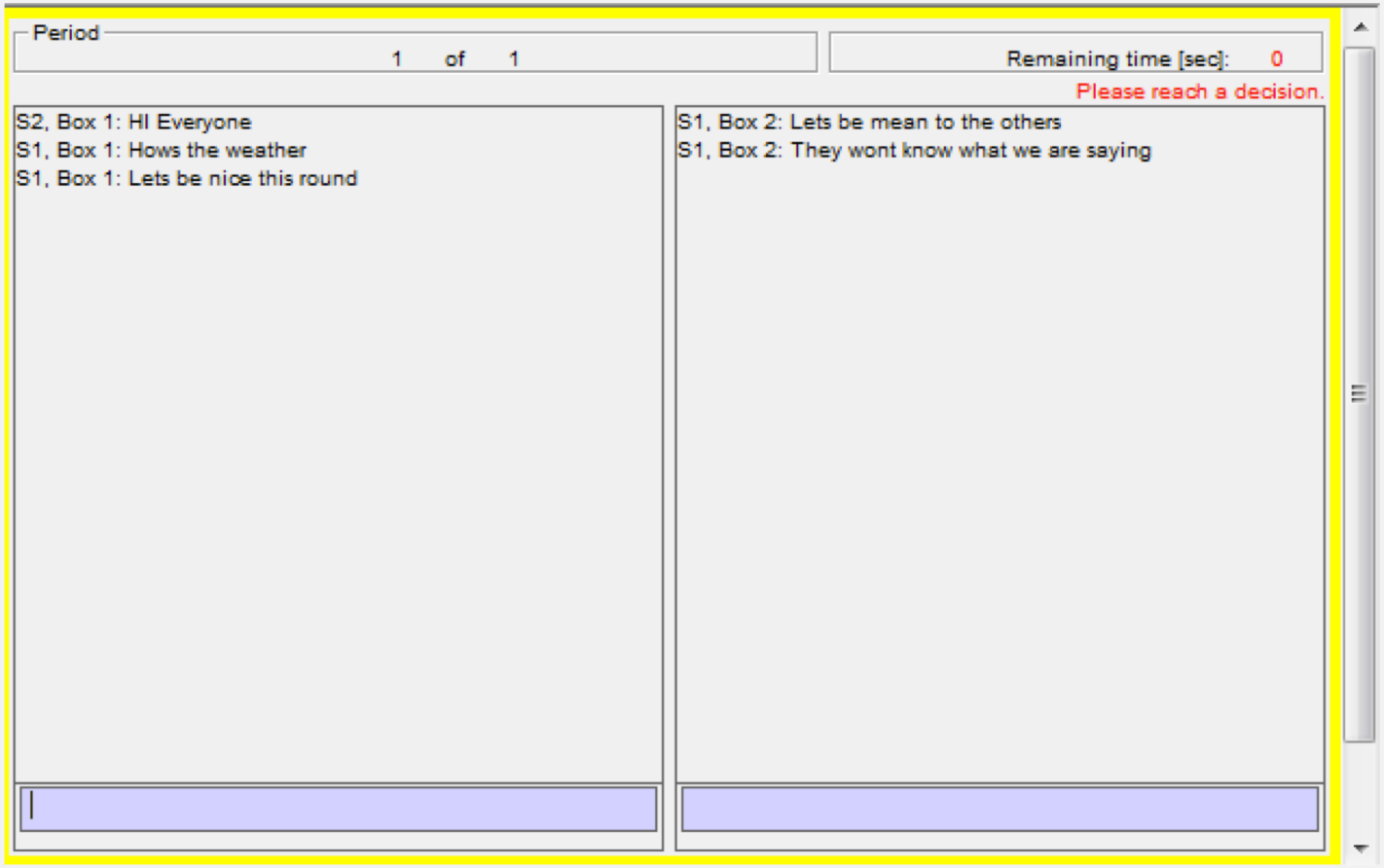


EXAMPLE IX

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



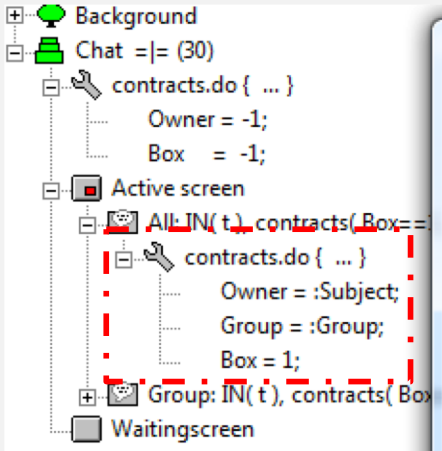
CONSIDERATIONS

- 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

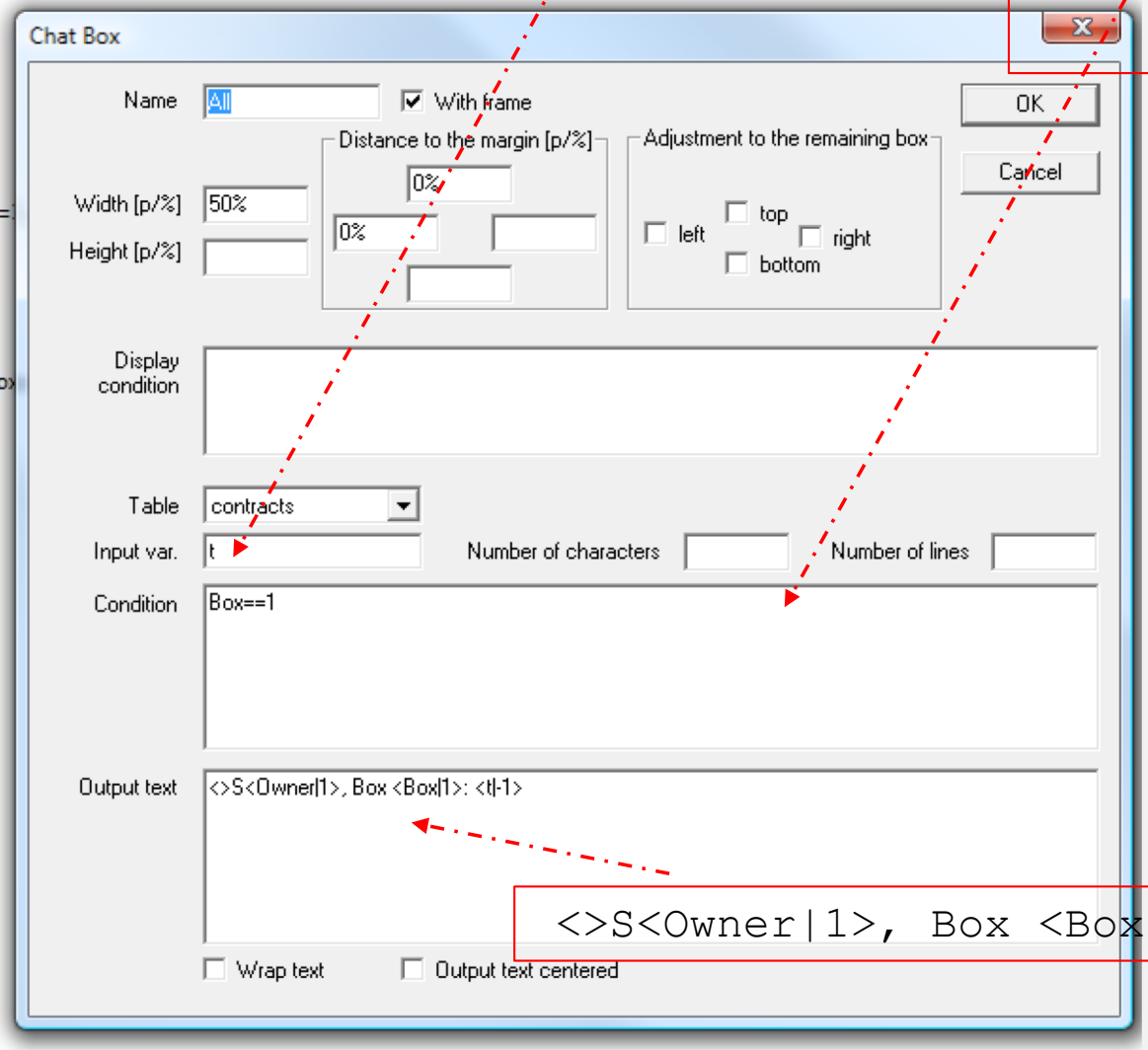
CHAT STAGE

- We first program the Box 1
- >treatment >new box >New Chat



Input variable is "t"

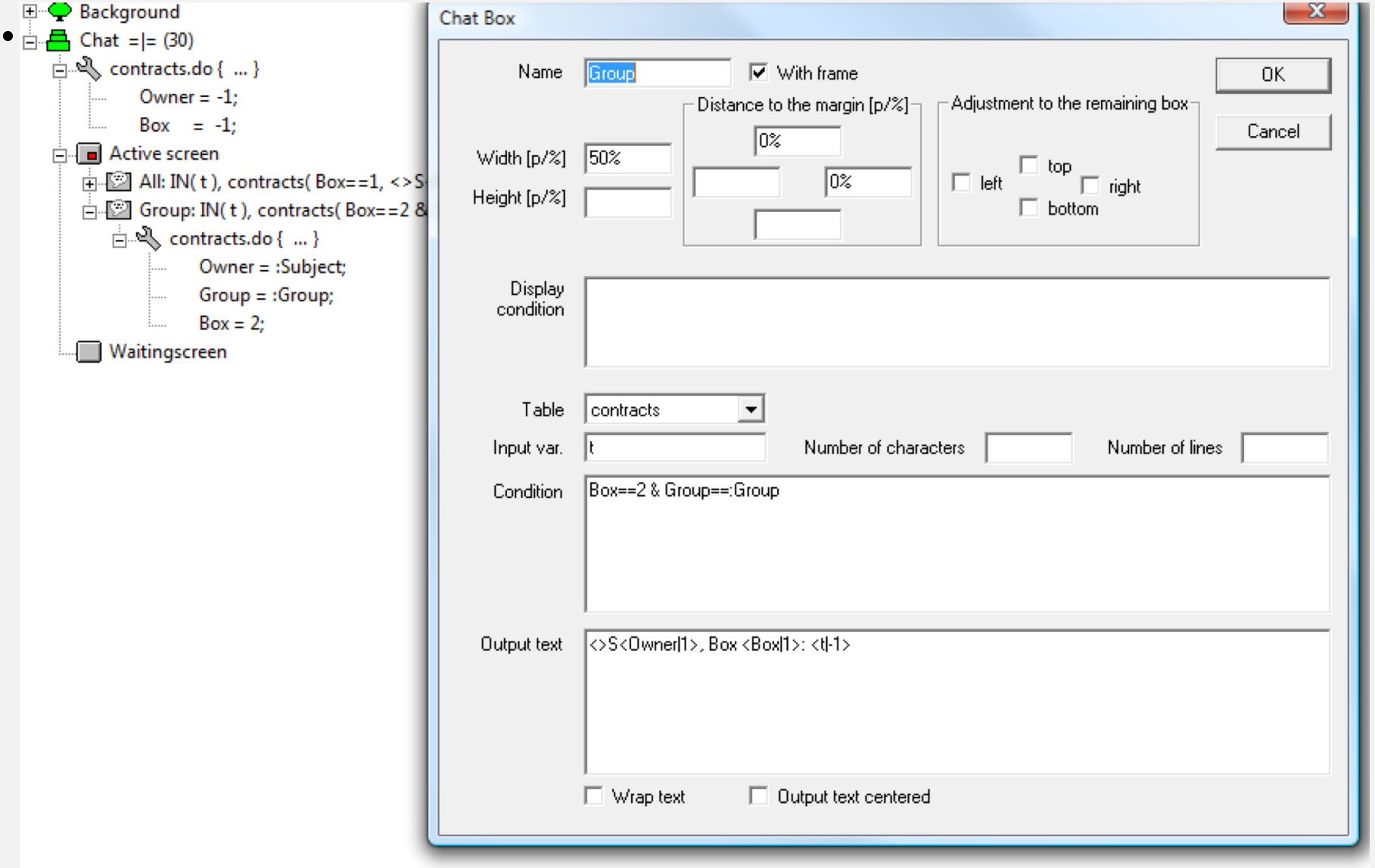
Only "t" associated with Box==1 is listed




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

CHAT STAGE

- Now we program the Box 2

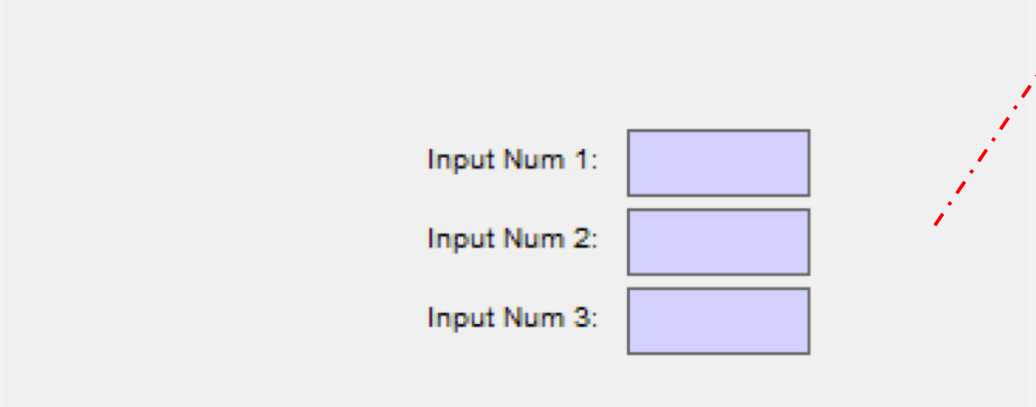


EXAMPLE X

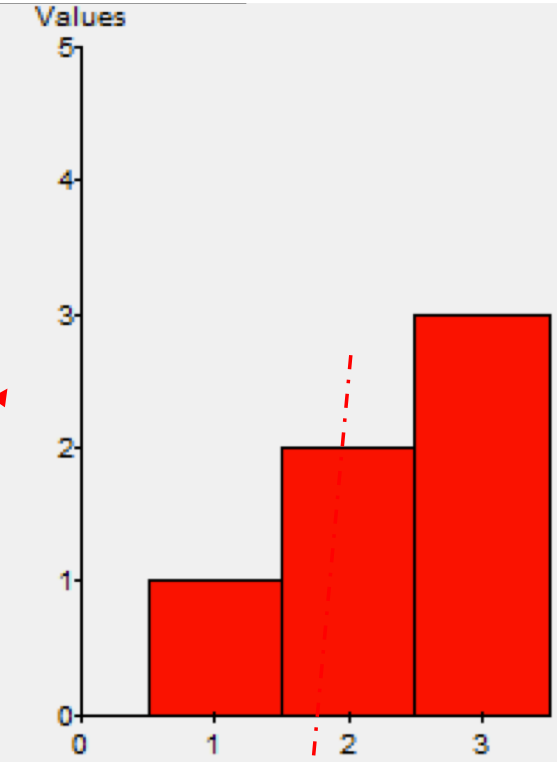
2 DIMENSIONAL GRAPHING

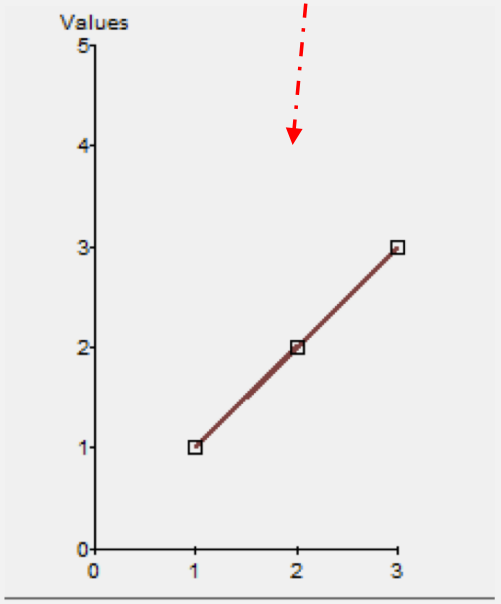
DESIGN

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

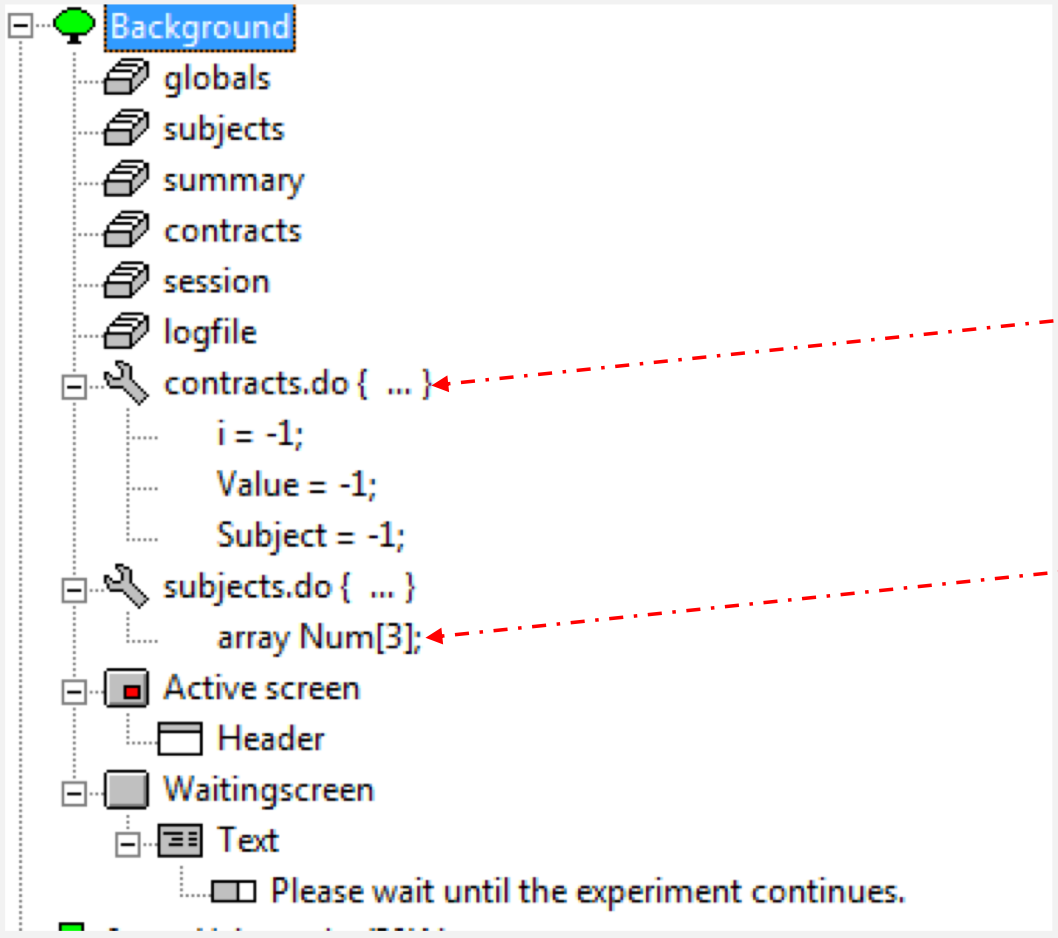


- We are going to utilise the contracts table





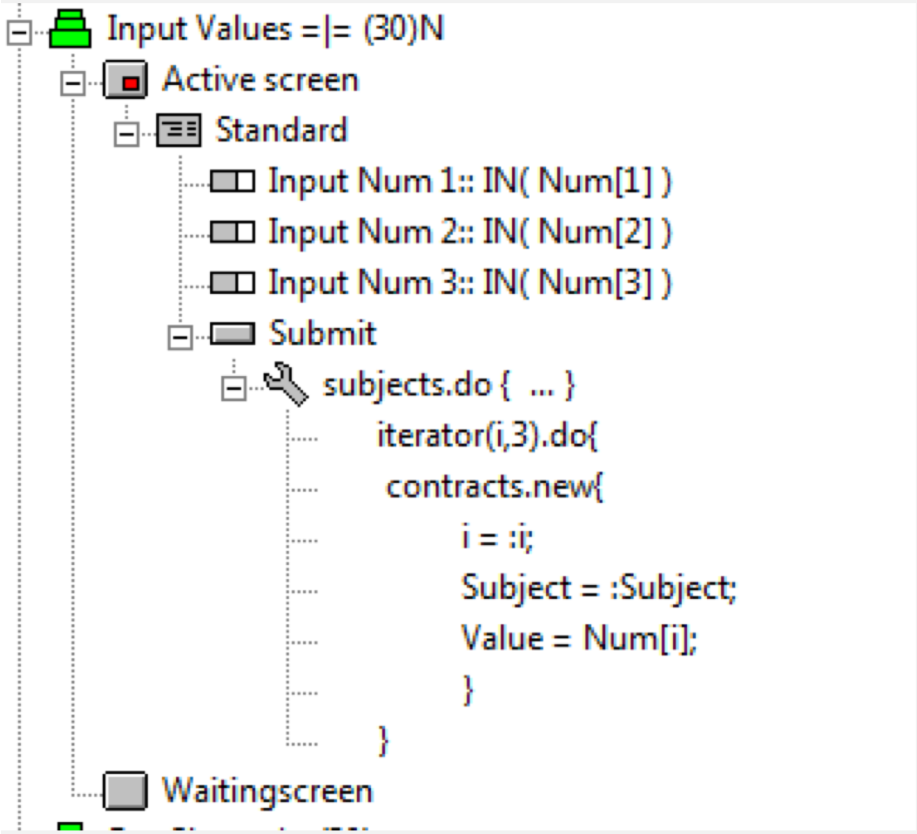
BACKGROUND

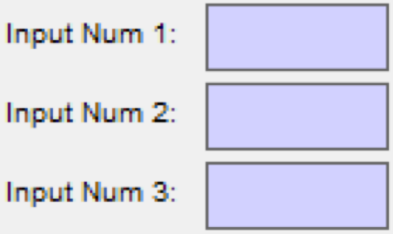


Numbers will be sorted in the contracts table

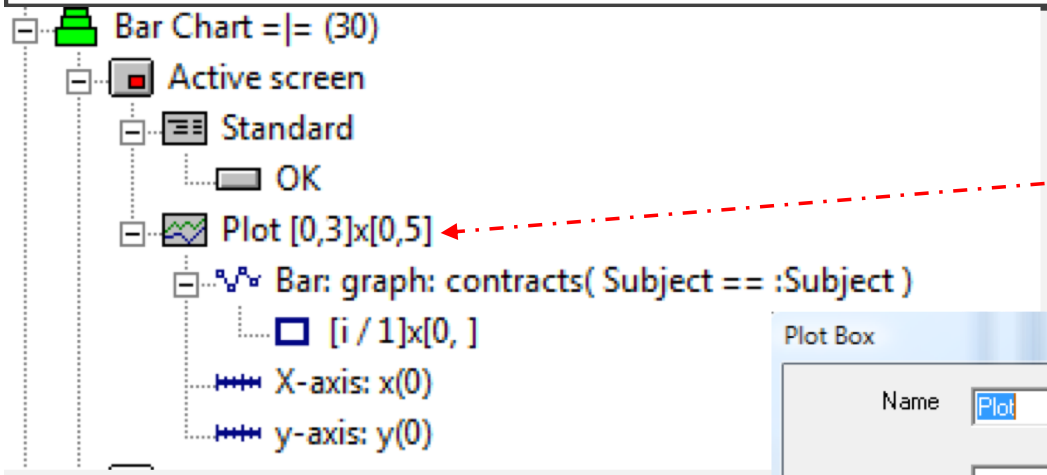
Subjects enter 3 numbers

STAGE 1

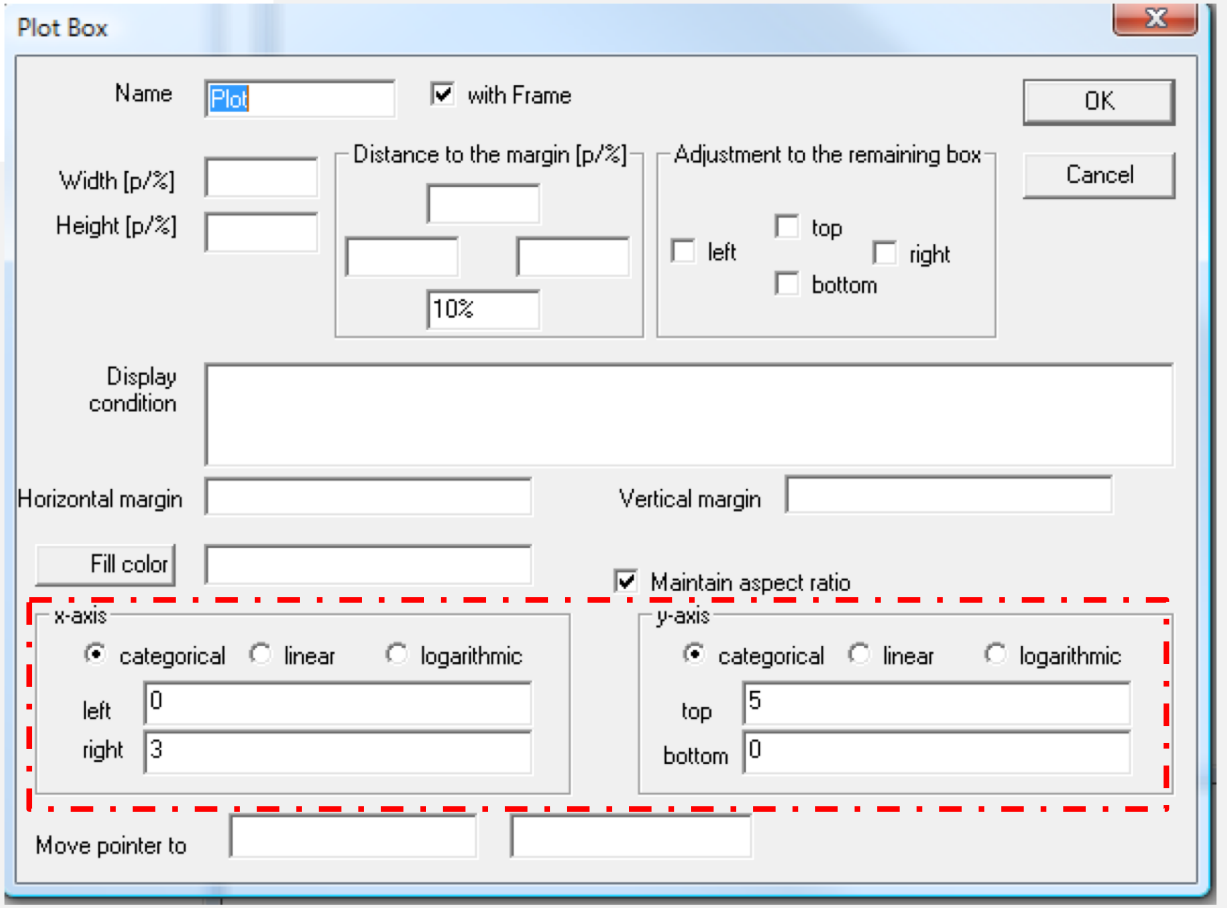

Input Values =|= (30)N
Active screen
Standard
Input Num 1:: IN(Num[1])
Input Num 2:: IN(Num[2])
Input Num 3:: IN(Num[3])
Submit
subjects.do { ... }
iterator(i,3).do{
contracts.new{
i = :i;
Subject = :Subject;
Value = Num[i];
}
}
Waitingscreen


Input Num 1:
Input Num 2:
Input Num 3:

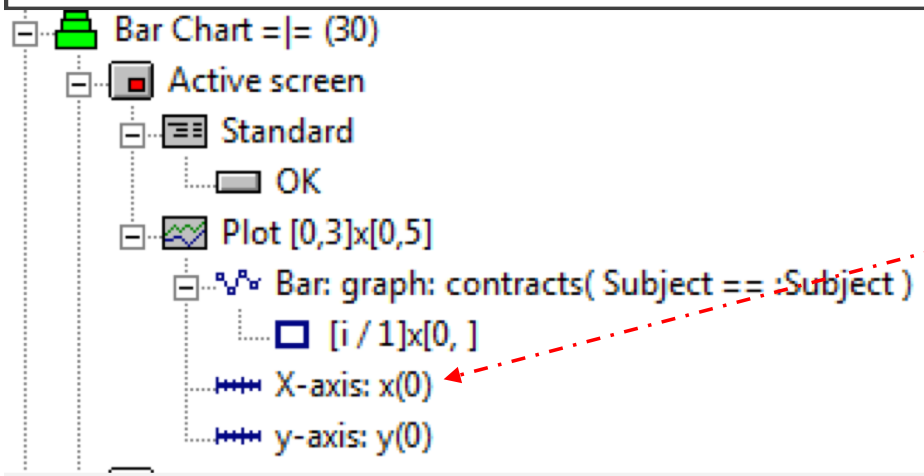
STAGE II (BAR CHART)



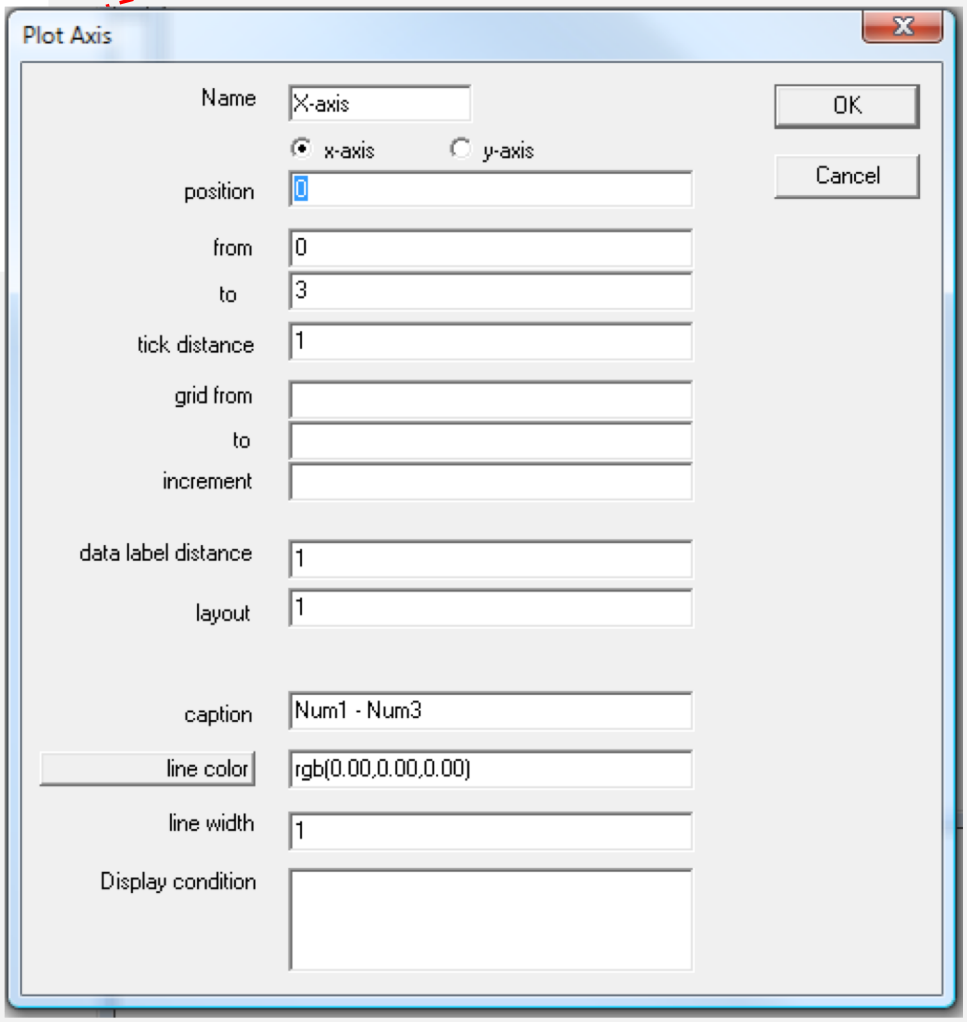
>treatment >New Box >New Plot box



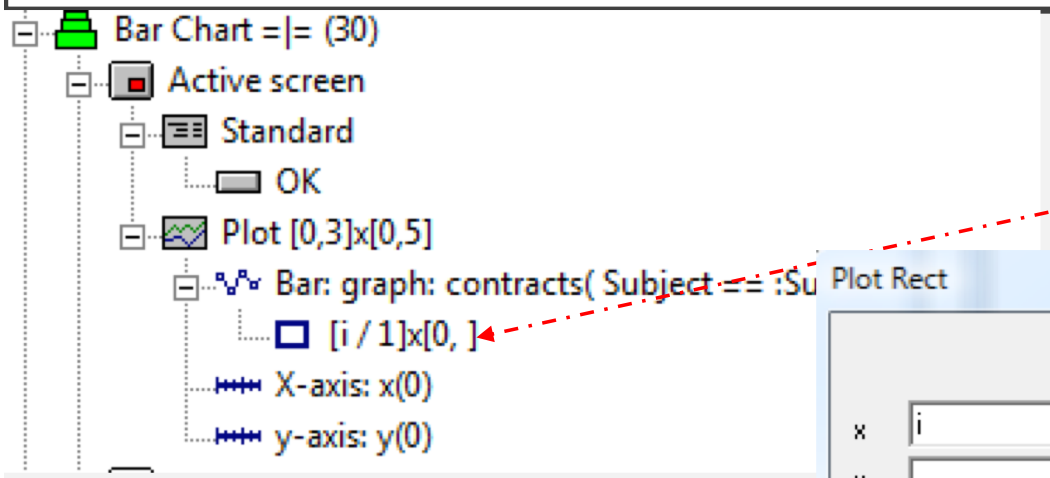
STAGE II (BAR CHART)
DEFINE THE AXIS



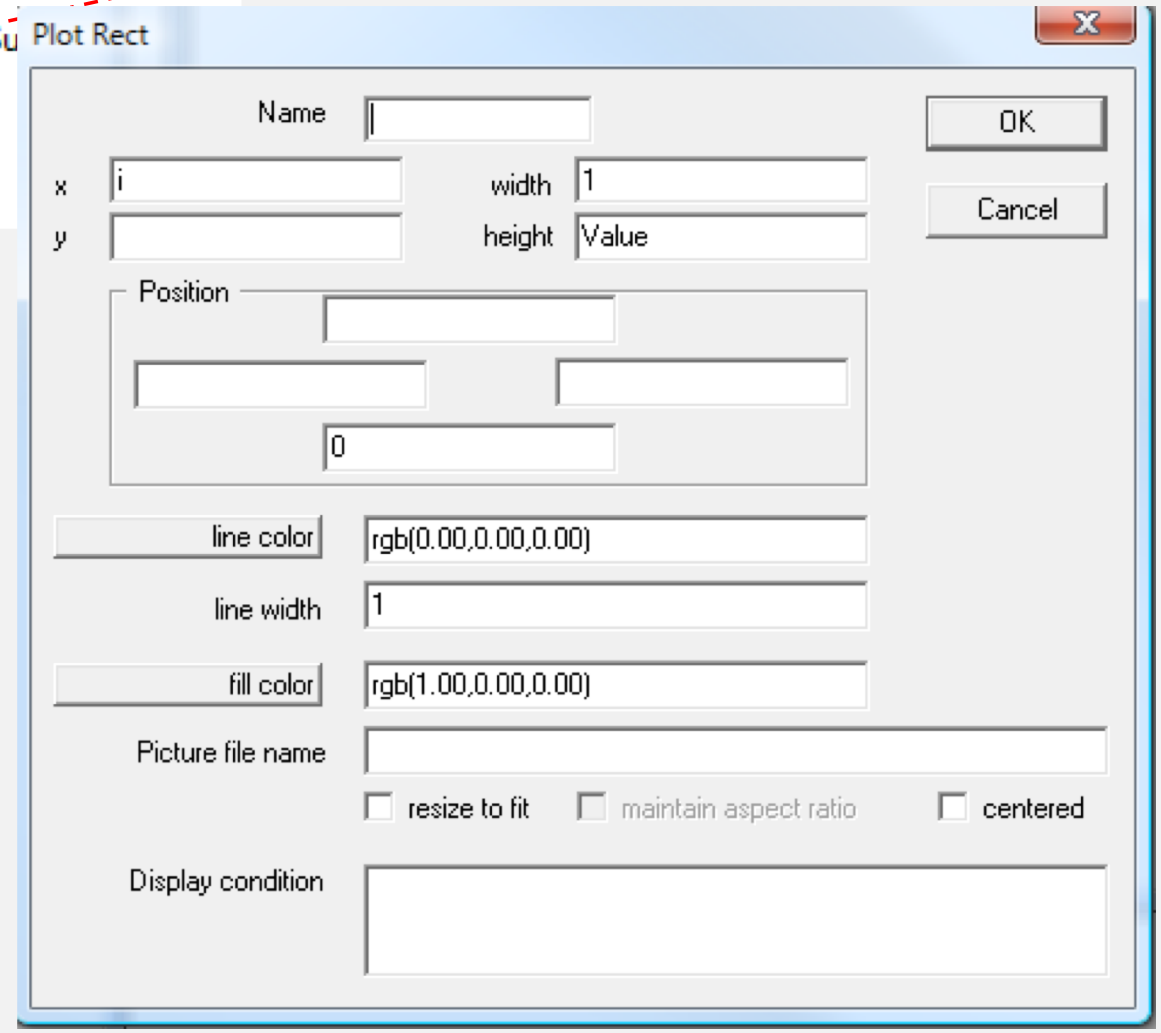
>treatment >Graphics >New Axis



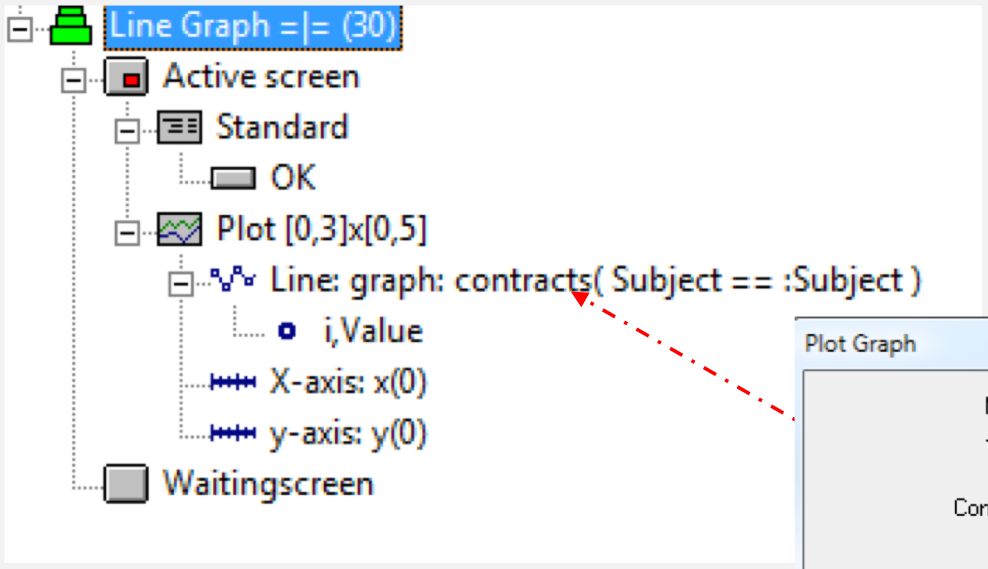
STAGE II (BAR CHART) CONNECT DATA TO PLOT

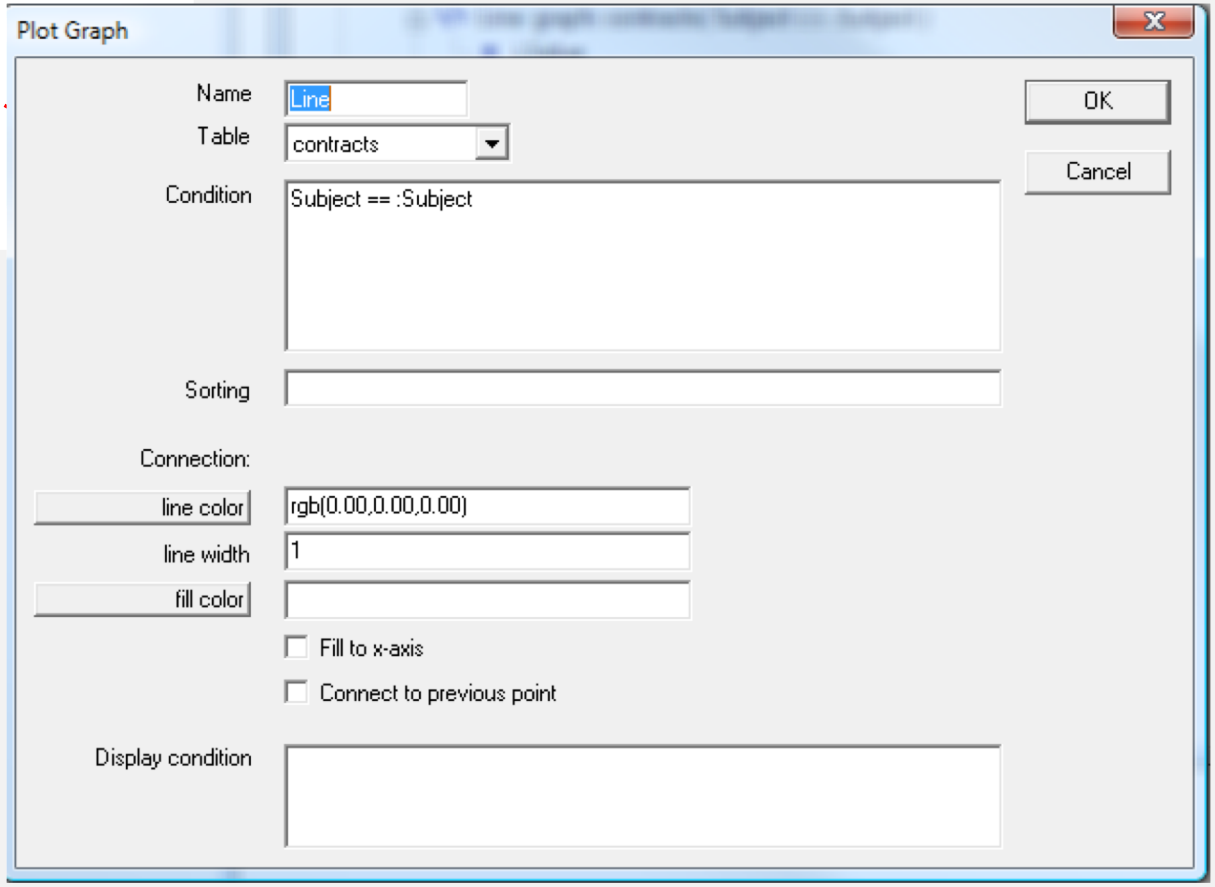


>treatment >Graphics >New Rect

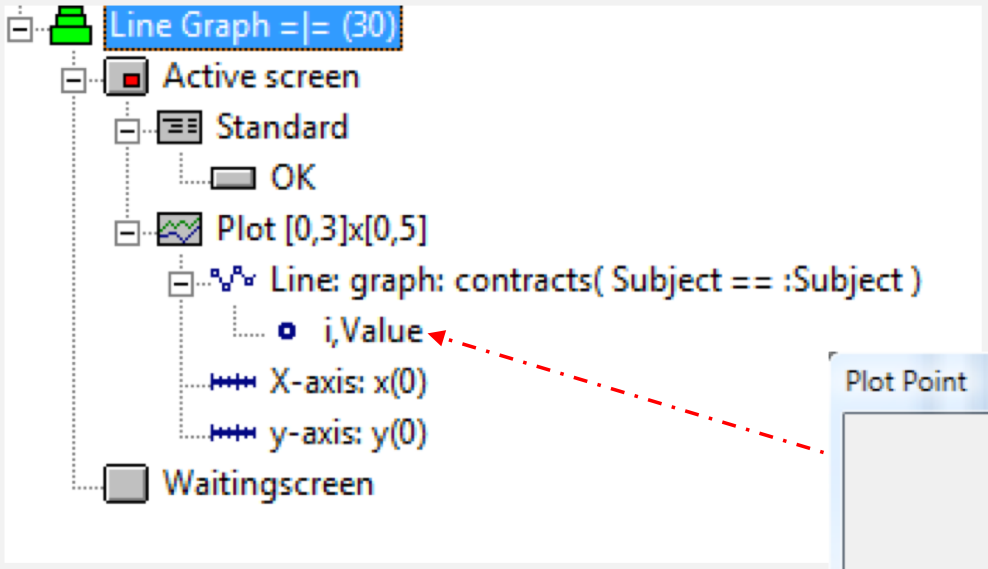


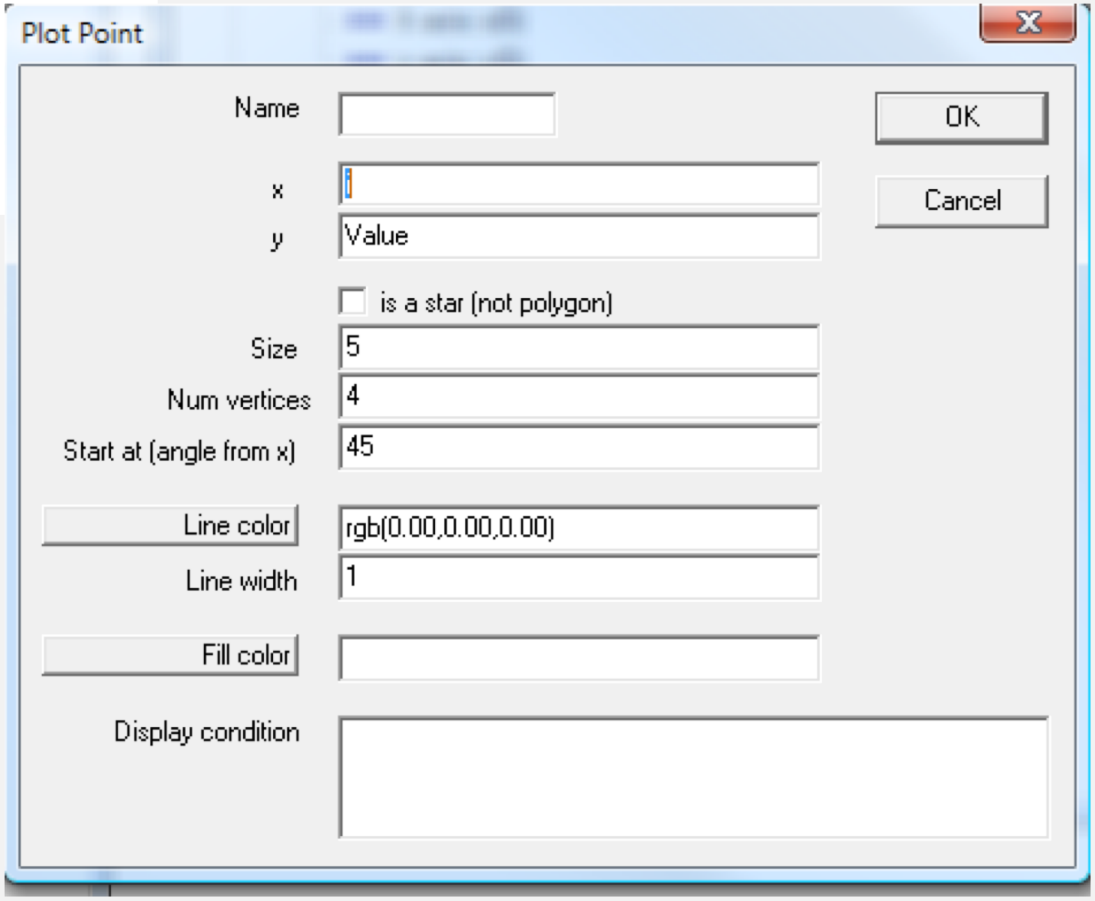
STAGE III (LINE) SETUP





STAGE III (LINE)
CONNECT DATA TO PLOT





EXAMPLE XI

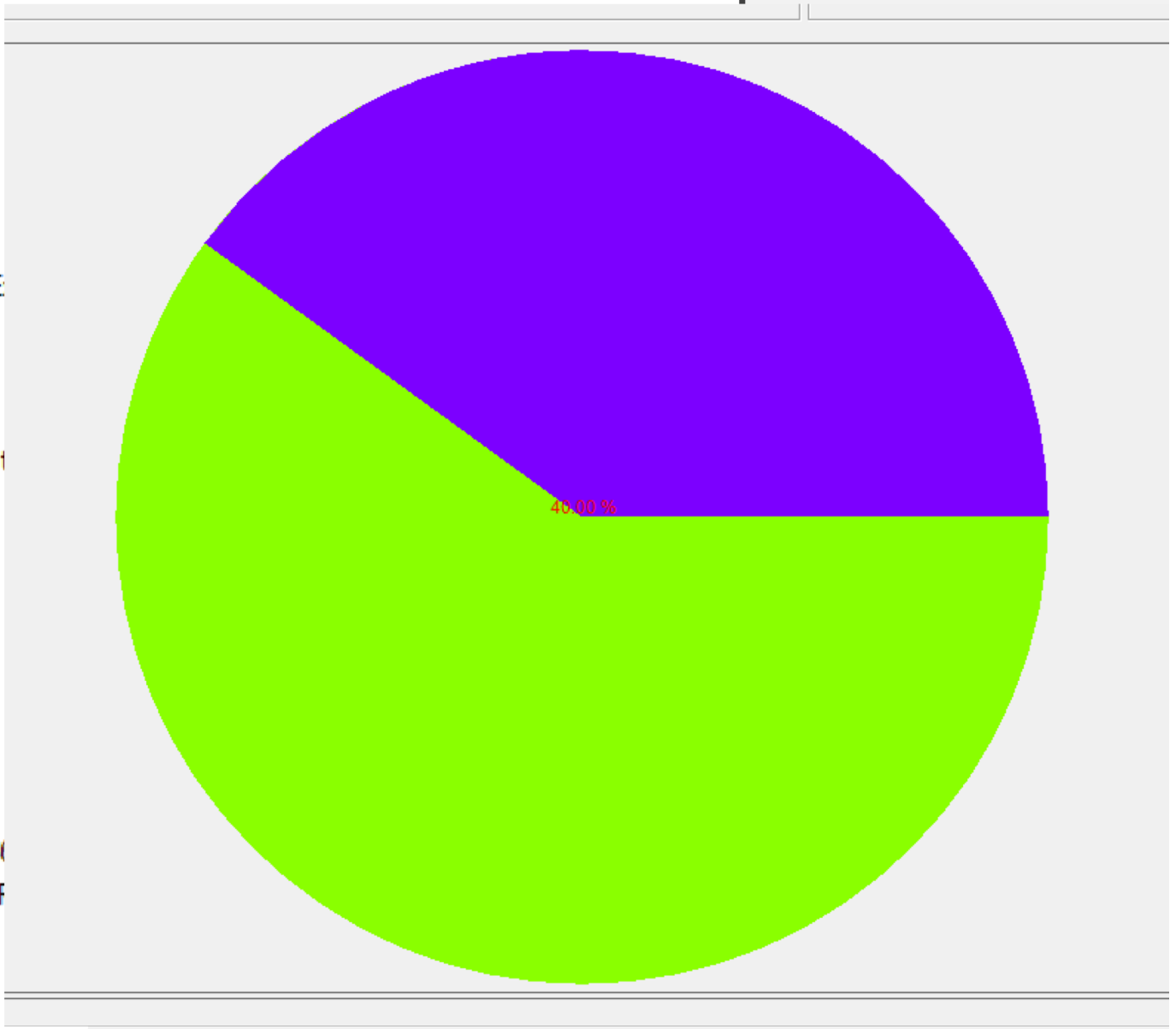
GRAPHING PIE CHARTS

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

- subjects.do { ... }
  - PIE = 100;
  - OFFER = 0;
- Active screen
  - Header
- Waitingscreen
  - Text
    - Bitte warten Sie, bis das E
- Make Decision -= (-1)N
- Active screen
  - INPUT
    - How much do you want t
    - OK
- Waitingscreen
- Stage =|= (30)
- subjects.do { ... }
  - R = OFFER / PIE * 360;
  - P = OFFER / PIE * 100;
- Active screen
  - PIE BOX [0,100]x[0,100]
    - BACK: (50,50) 50, 50 [0+3
    - INPUT: (50,50) 50, 50 [0+P
    - TEXT <><P|.01> %(50,50)
  - BUTTONBOX
    - OK
- Waitingscreen



EXERCISE IV

SSW MARKETS

DESIGN

- N>2 Traders each endowned
  - 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