z-Tree

Design: Urs Fischbacher
Programming: Urs Fischbacher and Stefan Schmid

This slides by Ernesto Reuben

Web Resources

zTree hompage

http://www.iew.unizh.ch/ztree/index.php

zTree Wiki

https://www.uzh.ch/iew/ztree/ssl-dir/wiki/

zTree mailing list

send email to majordomo@id.uzh.ch with “subscribe ztree_l” in the message body

Exp. 1: Measuring risk aversion

- We are interested in measuring risk aversion.
- Elicit certainty equivalent of a lottery using the Becker-Degroot-Marschak mechanism:

- Lottery: \$0 with probability p and \$X with probability $(1 – p)$.

- Subjects are asked for their CE:
  - “State the amount of money that makes you indifferent between receiving that amount or playing the lottery”
- A number z is randomly drawn between 0 and X.
  - if z ≥ CE, the subject receives \$z
  - if z < CE the subject plays the lottery

The Stage Tree

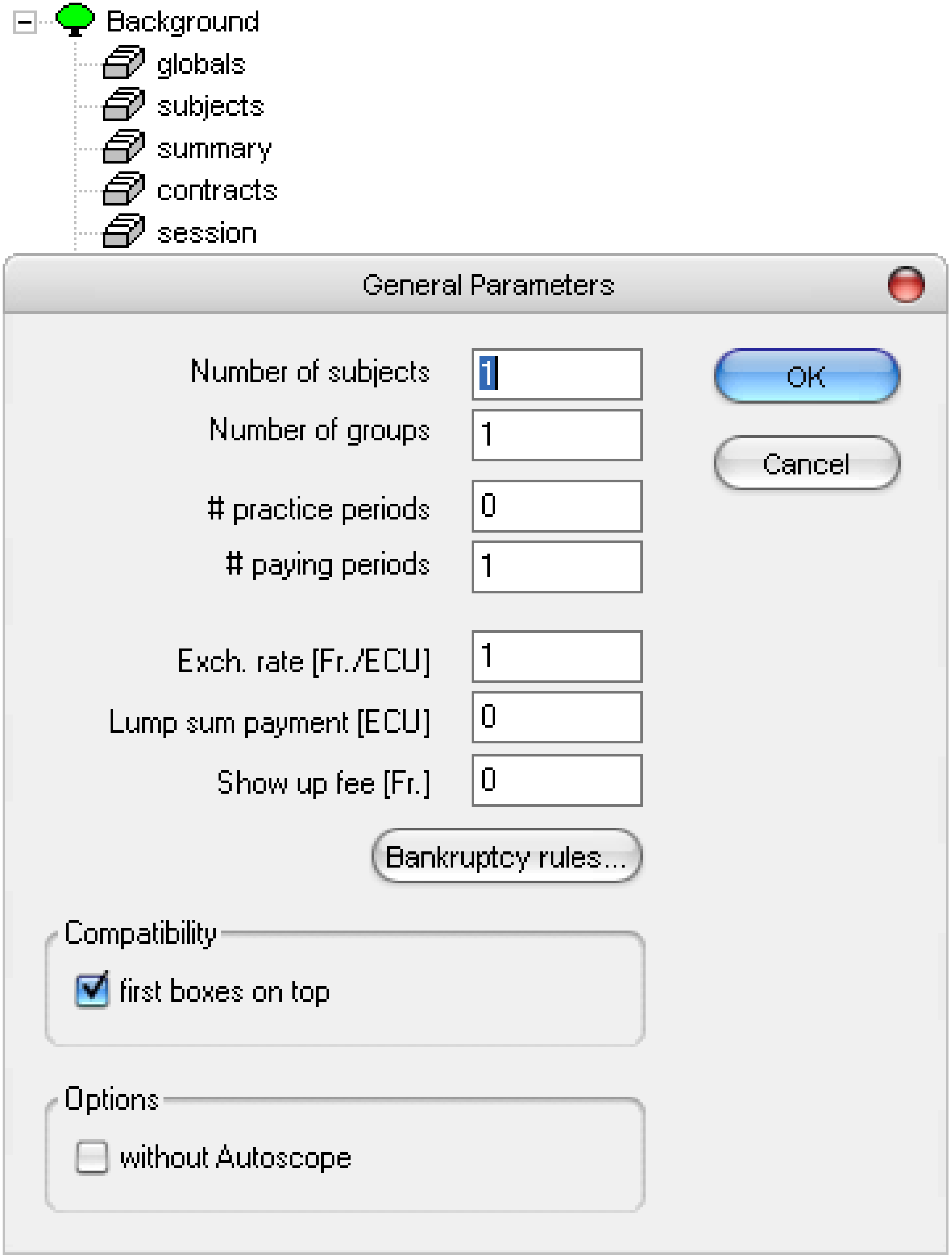
The description of a treatment is arranged in a tree structure:

- The **stage tree** shows the sequence of stages:
  - Stages contain **programs** and the two **screens**.
    - Screens (active and waiting).
      - Used to input and display data (and messages).
      - Screens contain boxes.
        - Boxes contain items and buttons.
    - Programs.
      - Used to manipulate data.
      - Set treatment variables.

z-Tree

Background

- Set number of subjects.
- Set number of rounds.
- Set exchange rate.
- Default screens.
- Treatment variables.



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

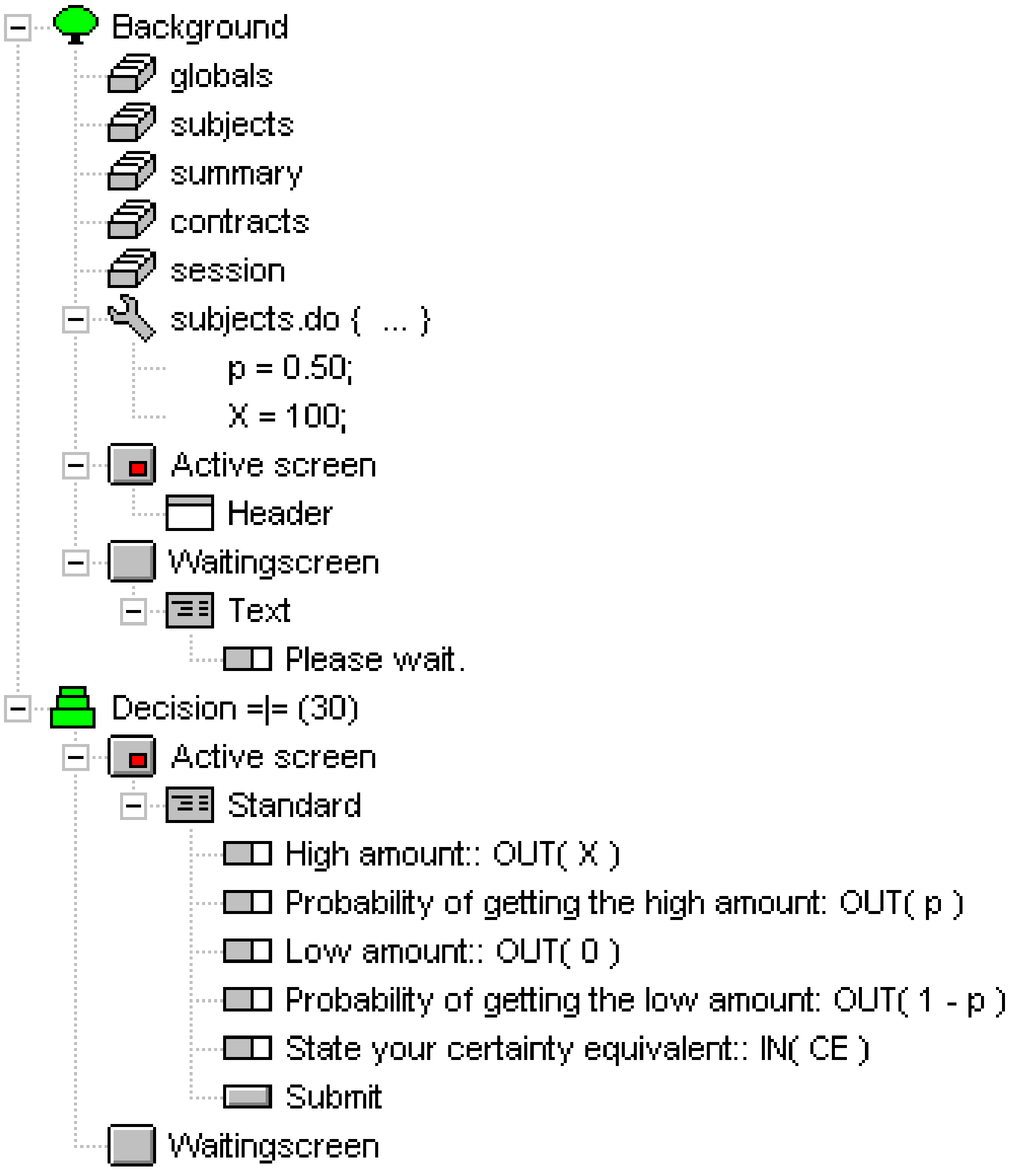
Compatibility
☑ first boxes on top

Options
☐ without Autoscope

z-Tree

Add Stages

- Each stage corresponds (roughly) to one screen.
- In this case we need 2 stages:
  - Decision stage.
  - Results stage.



```
Background
    globals
    subjects
    summary
    contracts
    session
    subjects.do {  ... }
        p = 0.50;
        X = 100;
    Active screen
        Header
    Waitingscreen
        Text
            Please wait.
Decision =|= (30)
    Active screen
        Standard
            High amount:: OUT( X )
            Probability of getting the high amount: OUT( p )
            Low amount:: OUT( 0 )
            Probability of getting the low amount: OUT( 1 - p )
            State your certainty equivalent:: IN( CE )
            Submit
    Waitingscreen
```

z-Tree

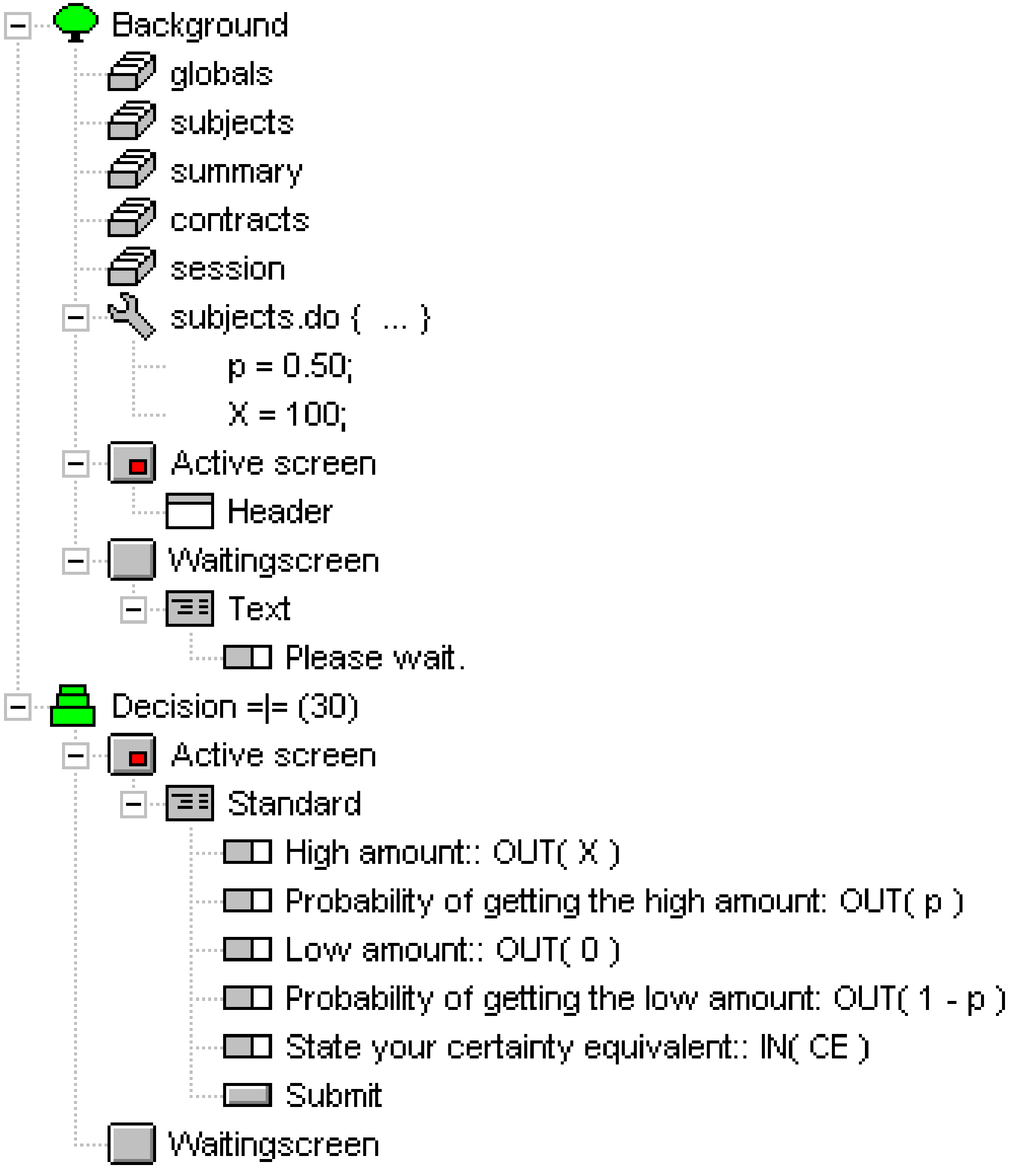
Add Stages

Can subject enter **stage**?

Programs are executed.

Active screen is displayed.

Waiting screen is displayed
(if the next stage cannot be entered)



```
Background
    globals
    subjects
    summary
    contracts
    session
    subjects.do {  ... }
        p = 0.50;
        X = 100;
    Active screen
        Header
    Waitingscreen
        Text
            Please wait.
Decision =|= (30)
    Active screen
        Standard
            High amount:: OUT( X )
            Probability of getting the high amount: OUT( p )
            Low amount:: OUT( 0 )
            Probability of getting the low amount: OUT( 1 - p )
            State your certainty equivalent:: IN( CE )
            Submit
    Waitingscreen
```

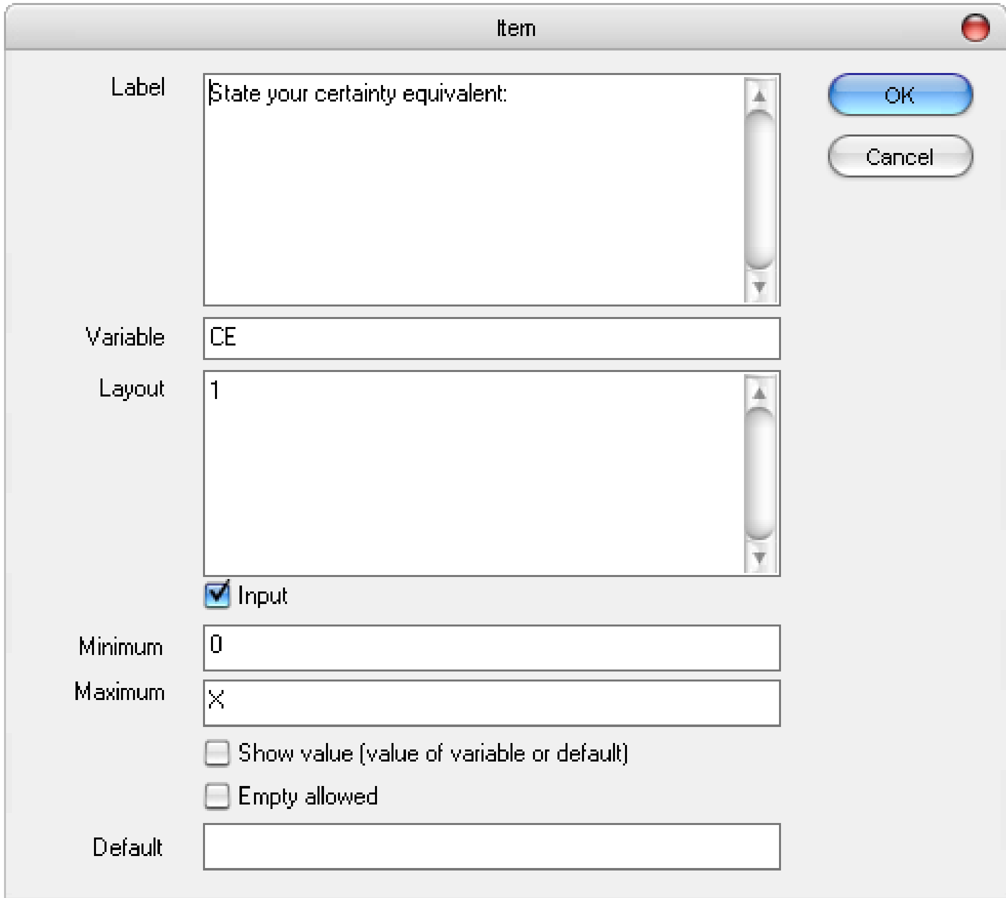
Input and Output

Items are used for the input and output of variables.

- Label (text displayed)
- Variable (for input or output)
- Layout:
  - numbers
  - check boxes
  - scrollbars
  - radio buttons
  - sliders

Note:

- If the item is used for input we also need a **button**.



Item

Label: State your certainty equivalent:

Variable: CE

Layout: 1

☑ Input

Minimum: 0

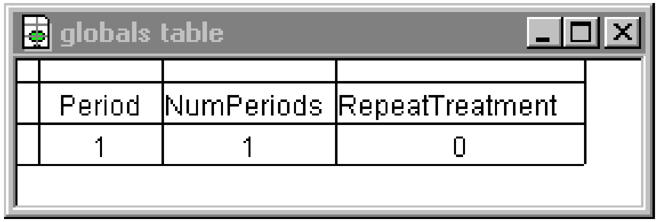
Maximum: X

□ Show value (value of variable or default)

□ Empty allowed

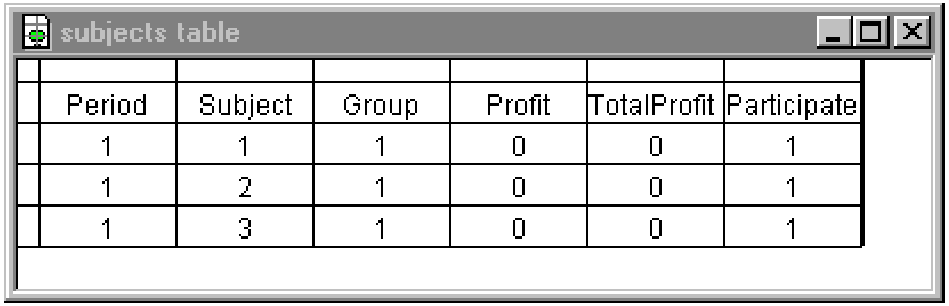
Default:

Input and Tables



globals table

Period	NumPeriods	RepeatTreatment
1	1	0



subjects table

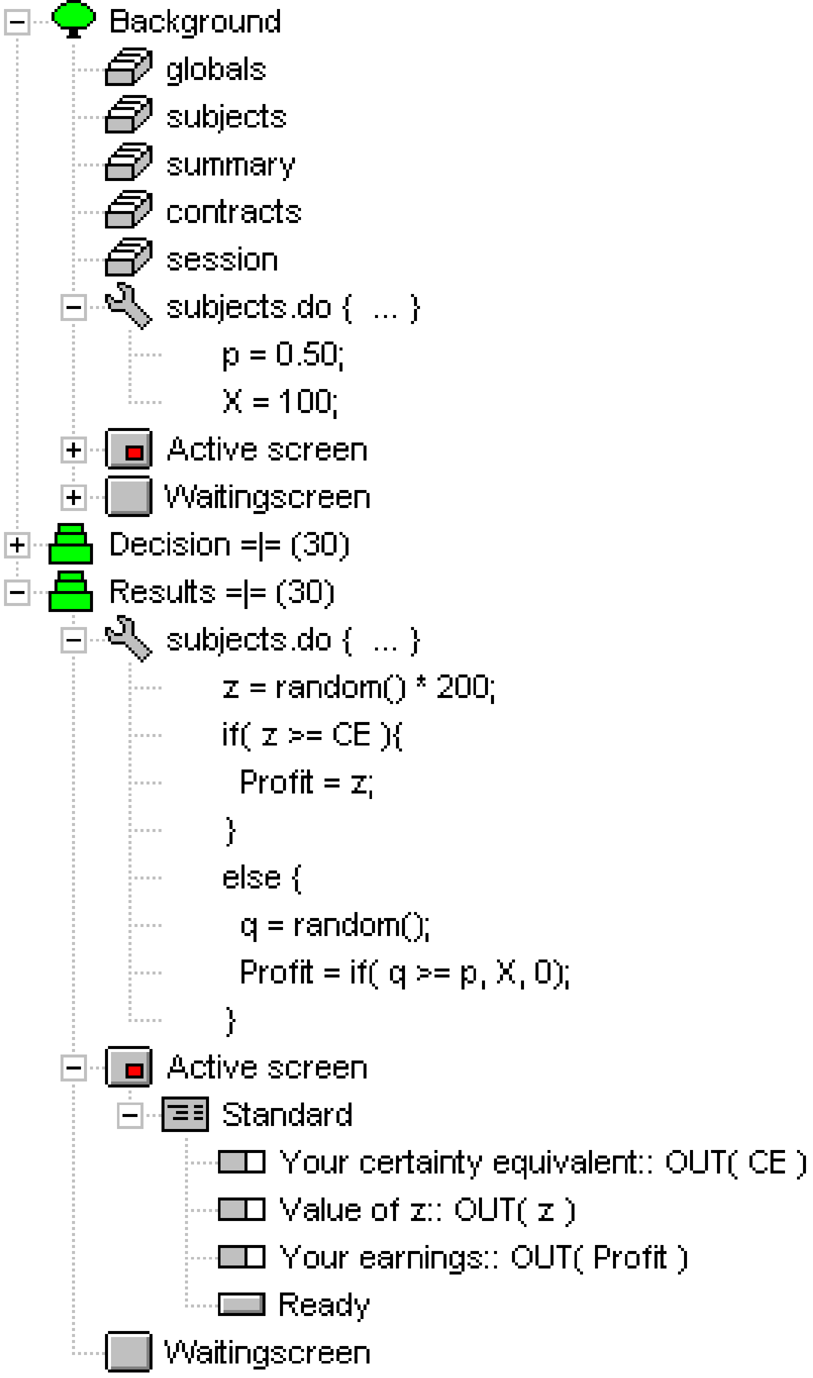
Period	Subject	Group	Profit	TotalProfit	Participate
1	1	1	0	0	1
1	2	1	0	0	1
1	3	1	0	0	1

- When subjects make an input, the data is transferred to z-Tree.
- The data is stored in tables.
- The tables can be viewed in a window in z-Tree (menu Treatment)
- Most data is stored in the **subjects table**.
  - One row per subject.
  - For every period, there is a new ‘subjects table’.
- Other tables: (contracts, session, globals, summary, and OLDsubjects)
- Other tables can be accessed by table.tablefunction.

z-Tree

Programs

- Programs can be executed at the beginning of a stage and when buttons are clicked.
- Calculations are performed by z-Tree and then sent to the z-Leafs.
- Programs are executed **row by row** (i.e. subject by subject).



```
Background
    globals
    subjects
    summary
    contracts
    session
    subjects.do {  ... }
            p = 0.50;
            X = 100;
    Active screen
    Waitingscreen
Decision =|= (30)
Results =|= (30)
    subjects.do {  ... }
            z = random() * 200;
            if( z >= CE ){
              Profit = z;
            }
            else {
              q = random();
              Profit = if( q >= p, X, 0);
            }
    Active screen
        Standard
            Your certainty equivalent:: OUT( CE )
            Value of z:: OUT( z )
            Your earnings:: OUT( Profit )
            Ready
    Waitingscreen
```

Functions and statements

- There is a good number of functions that can be used for programming:

```
subjects.do{
        z = random() * 200;
        if( z >= CE ){
                Profit = z;
        }
        else {
                q = random();
                Profit = if( q >= p, X, 0);
        }
}
```

Exp. 2: A public goods game

- In each period each subject gets y points.
  - Points can be kept or invested in a public good
- The profit of each subject is:

$$\pi_i = y - c_i + (\alpha/N)\sum_j c_j$$

- The game is played for t periods.

- Note:
  - if no one contributes: $\pi_i = y$
  - if everyone contributes y: $\pi_i = \alpha y$
  - If $1 > \alpha/n$ you are better off if you do not contribute

Table functions

- Syntax 1: table function(expression)
  - Example: Profits in the public goods game:

```
subjects.do{
      SumContribute = sum(Contribute);
      N = count( );
      GroupProfit = EfficiencyFactor * SumContribute / N;
      Profit = Endowment - Contribute + GroupProfit;
}
```

  - Example: Maximum contribution

```
subjects.do{
      MaxContribute = maximum(Contribute);
}
```

Table functions

- Syntax 2: table function(condition, expression)
  - Example: Sum of all the contributions that exceed 10

```
subjects.do{
	SumHighContribute = sum(Contribute > 10, Contribute );
}
```

  - Example: Subject who contributed the least

```
subjects.do{
	CheapSubject = find(Contribute == minimum(Contribute), Subject );
}
```

Exp. 3: A public goods game in groups

- In each period subjects are assigned to groups of *n*
- Each subject gets *y* points.
  - Points can be kept or invested in a public good.
- The profit of each subject is:

$$\pi_i = y - c_i + (\alpha/n)\sum_j c_j$$

- The game is played for *t* periods.

Groups

- The variable **Group** determines the group matching.
- The number of groups can be set in the background stage.
- There are menu commands for different types of matchings (treatment menu):
  - Partner
  - Stranger
  - absolute Stranger
  - typed absolute Stranger

- **Important**:
  - Before running an experiment, check the **Parameter** table (treatment menu).

Groups

- The Group variable can also be changed:
  - Manually in the Parameter table
    - Double-click on each cell and set group
  - Through a program in the background stage

```
subjects.do{
    if( Subject <= 5 ){
        Group = 1;
    }
    elsif( Subject <= 9) {
        Group = 2;
    }
    else {
        Group = 3;
    }
}
```

Same

- same( ) can be used to make group calculations
  - Example: Profits in the public goods game:

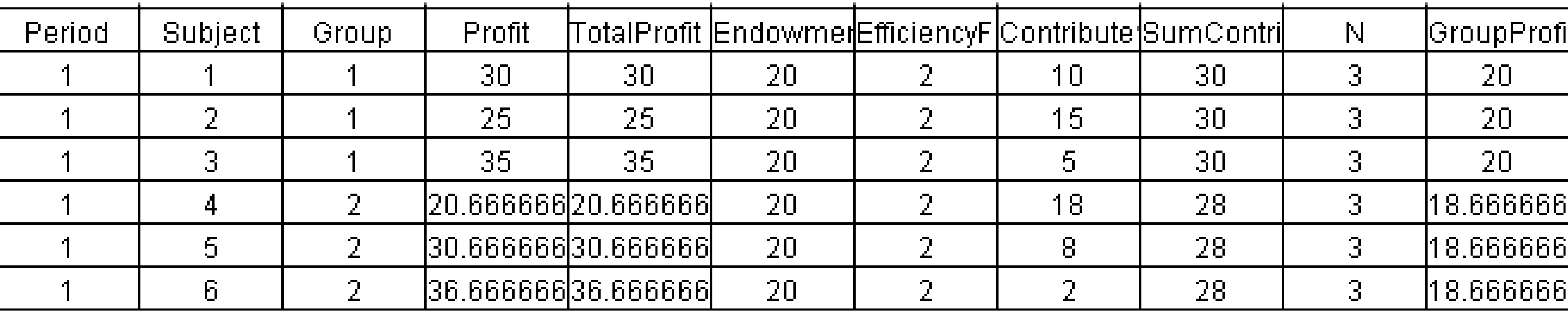
```
subjects.do{
        SumContribute = sum( same(Group), Contribute );
        N = count( same(Group) );
        GroupProfit = EfficiencyFactor * SumContribute / N;
        Profit = Endowment - Contribute + GroupProfit;
}
```

z-Tree

Scope Operator

- Alternatively, one can use the **scope** operator.
  - Sum contributions of all group members.

```
subjects.do{
        SumContribute = sum(Group == :Group, Contribute );
}
```



Period	Subject	Group	Profit	TotalProfit	Endowmer	EfficiencyF	Contribute	SumContri	N	GroupProfi
1	1	1	30	30	20	2	10	30	3	20
1	2	1	25	25	20	2	15	30	3	20
1	3	1	35	35	20	2	5	30	3	20
1	4	2	20.666666	20.666666	20	2	18	28	3	18.666666
1	5	2	30.666666	30.666666	20	2	8	28	3	18.666666
1	6	2	36.666666	36.666666	20	2	2	28	3	18.666666

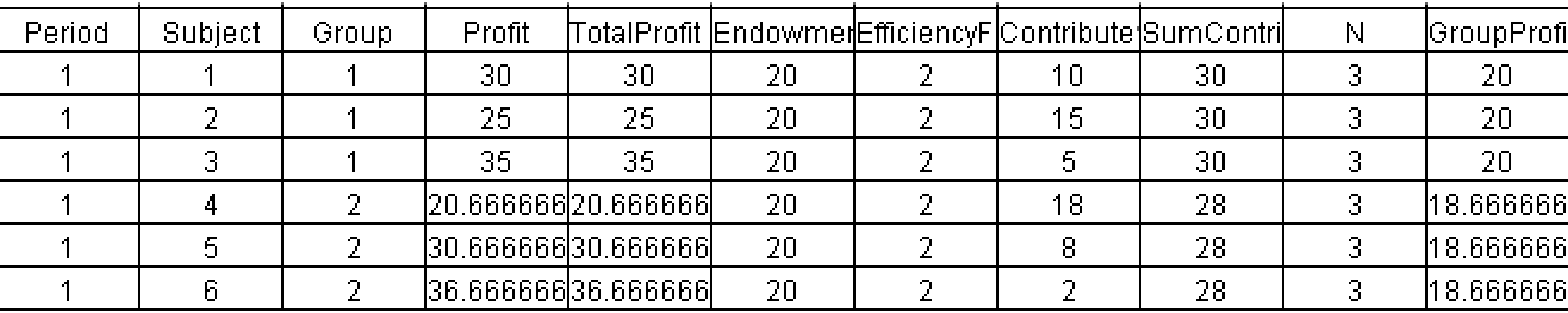
Scope Operator

- Building a ranking: incorrect

```
subjects.do{
        RankContribute = count(Contribute <= Contribute);
}
```

- Building a ranking: correct

```
subjects.do{
        RankContribute = count(Contribute <= :Contribute);
}
```



Period	Subject	Group	Profit	TotalProfit	Endowmer	EfficiencyF	Contribute	SumContri	N	GroupProfi
1	1	1	30	30	20	2	10	30	3	20
1	2	1	25	25	20	2	15	30	3	20
1	3	1	35	35	20	2	5	30	3	20
1	4	2	20.666666	20.666666	20	2	18	28	3	18.666666
1	5	2	30.666666	30.666666	20	2	8	28	3	18.666666
1	6	2	36.666666	36.666666	20	2	2	28	3	18.666666

Some useful matching programs

- Groups of n, partners:

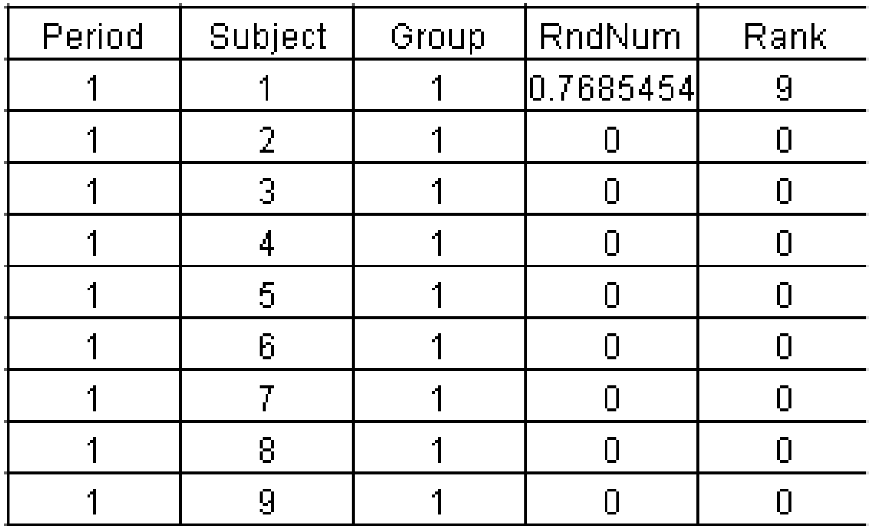
```
subjects.do{
        Group = mod(Subject, n) + 1;
}
```

z-Tree

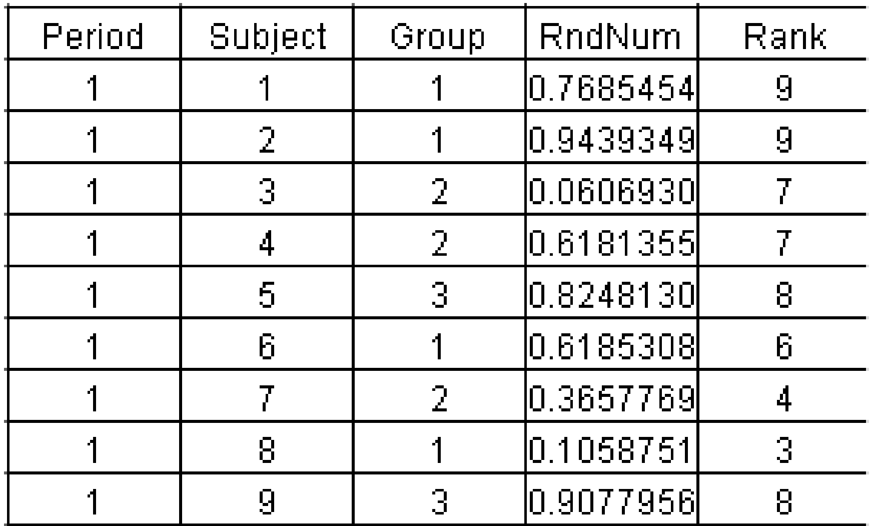
Some useful matching programs

- Groups of n, strangers: incorrect

```
subjects.do{
        RndNum = random();
        Rank = count(RndNum <= :RndNum);
        Group = mod(Rank, n) + 1;
}
```



Period	Subject	Group	RndNum	Rank
1	1	1	0.7685454	9
1	2	1	0	0
1	3	1	0	0
1	4	1	0	0
1	5	1	0	0
1	6	1	0	0
1	7	1	0	0
1	8	1	0	0
1	9	1	0	0

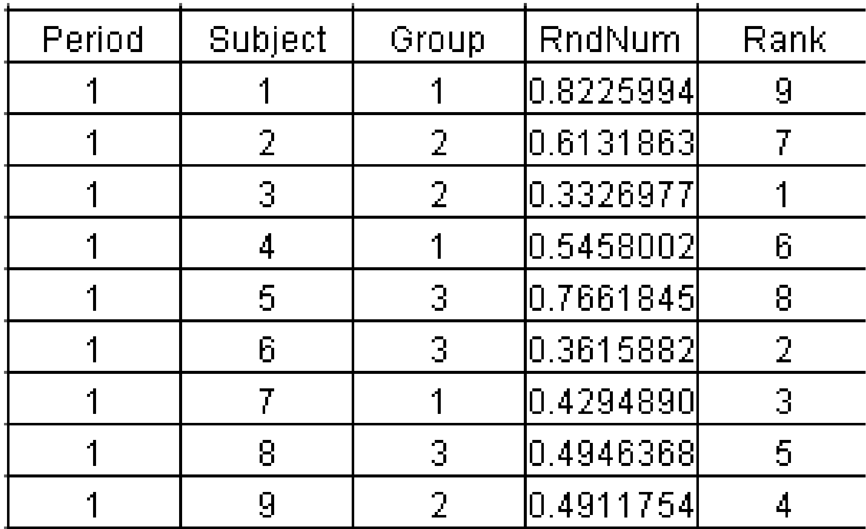


Period	Subject	Group	RndNum	Rank
1	1	1	0.7685454	9
1	2	1	0.9439349	9
1	3	2	0.0606930	7
1	4	2	0.6181355	7
1	5	3	0.8248130	8
1	6	1	0.6185308	6
1	7	2	0.3657769	4
1	8	1	0.1058751	3
1	9	3	0.9077956	8

Some useful matching programs

- Groups of n, strangers: correct

```
subjects.do{
        RndNum = random();
}
subjects.do{
        Rank = count(RndNum <= :RndNum);
        Group = mod(Rank, n) + 1;
}
```

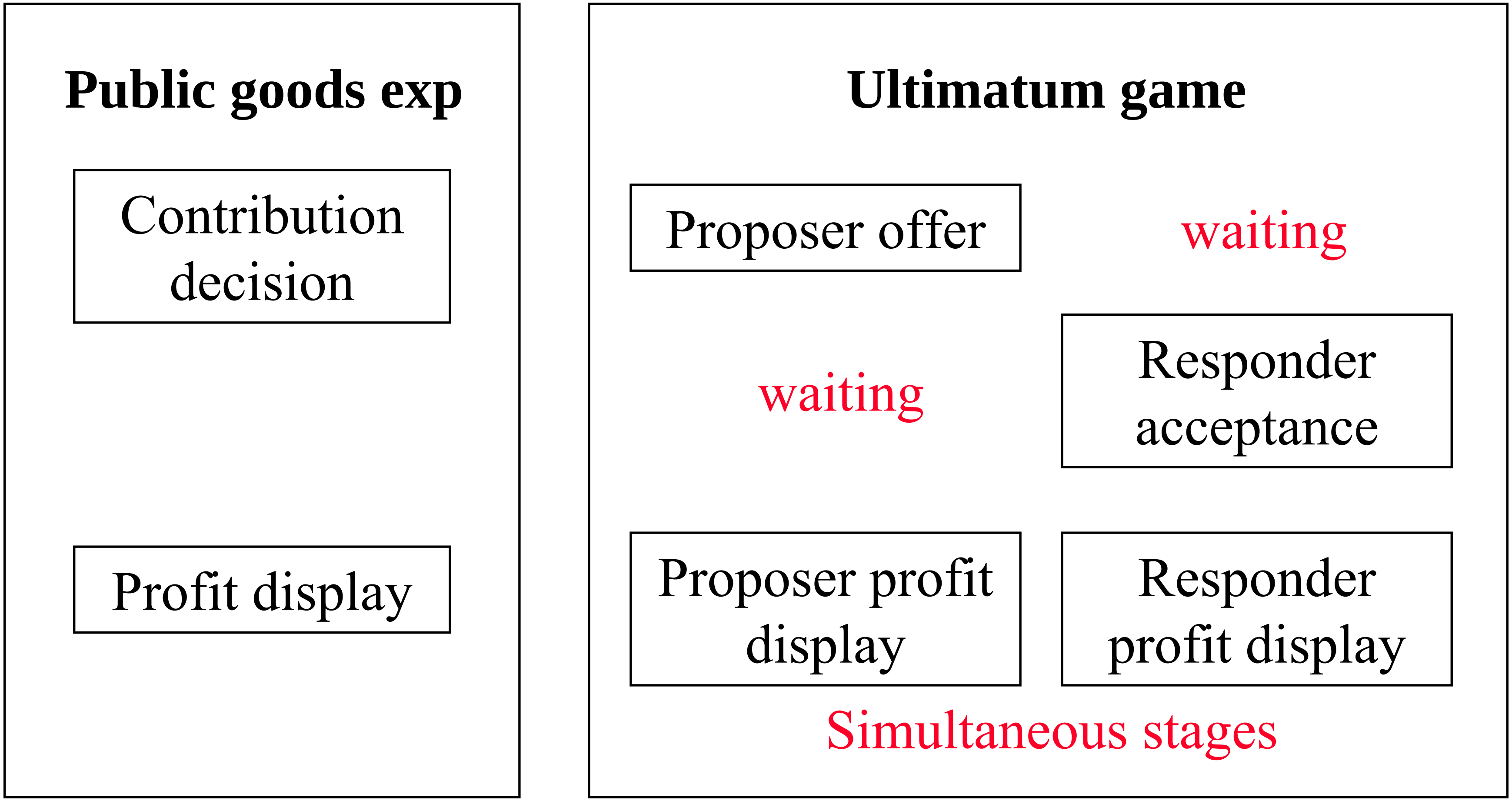


Period	Subject	Group	RndNum	Rank
1	1	1	0.8225994	9
1	2	2	0.6131863	7
1	3	2	0.3326977	1
1	4	1	0.5458002	6
1	5	3	0.7661845	8
1	6	3	0.3615882	2
1	7	1	0.4294890	3
1	8	3	0.4946368	5
1	9	2	0.4911754	4

Exp. 4: An ultimatum game

- Subjects are matched in pairs
  - Each pair has 1 proposer and 1 responder.
  - Each pair receives *y* points.
- Proposers offer responders *x* points from the *y* available points.
- Responders can accept or reject the offer.
  - If the responder accepts:
    - Proposers earn: $\pi_P = y - x$
    - Responders earn: $\pi_R = x$
  - If the responder rejects:
    - Both get 0 points.
- Play for *t* periods.
  - Random matching and random assignment of roles.

Examples



Public goods exp

Contribution decision

Profit display

Ultimatum game

Proposer offer

waiting

waiting

Responder acceptance

Proposer profit display

Responder profit display

Simultaneous stages

Types

- We need to assign types to players.
  - One proposer and one responder per group (randomly allocated)

```
subjects.do{
      RndNum = random();
}
subjects.do{
      RndOther = find(same(Group) & not( same(Subject) ) , RndNum);
      Proposer = if( RndOther > RndNum, 1, 0);
}
```

- Or easier ... You can also do this in the **parameter** table (less flexible)
  - period parameters, subject parameters, period × subject parameters

z-Tree

Participate

- The variable *Participate* can be used to select who enters a stage.
  - Enter stage: Participate = 1.
  - Skip stage: Participate = 0.

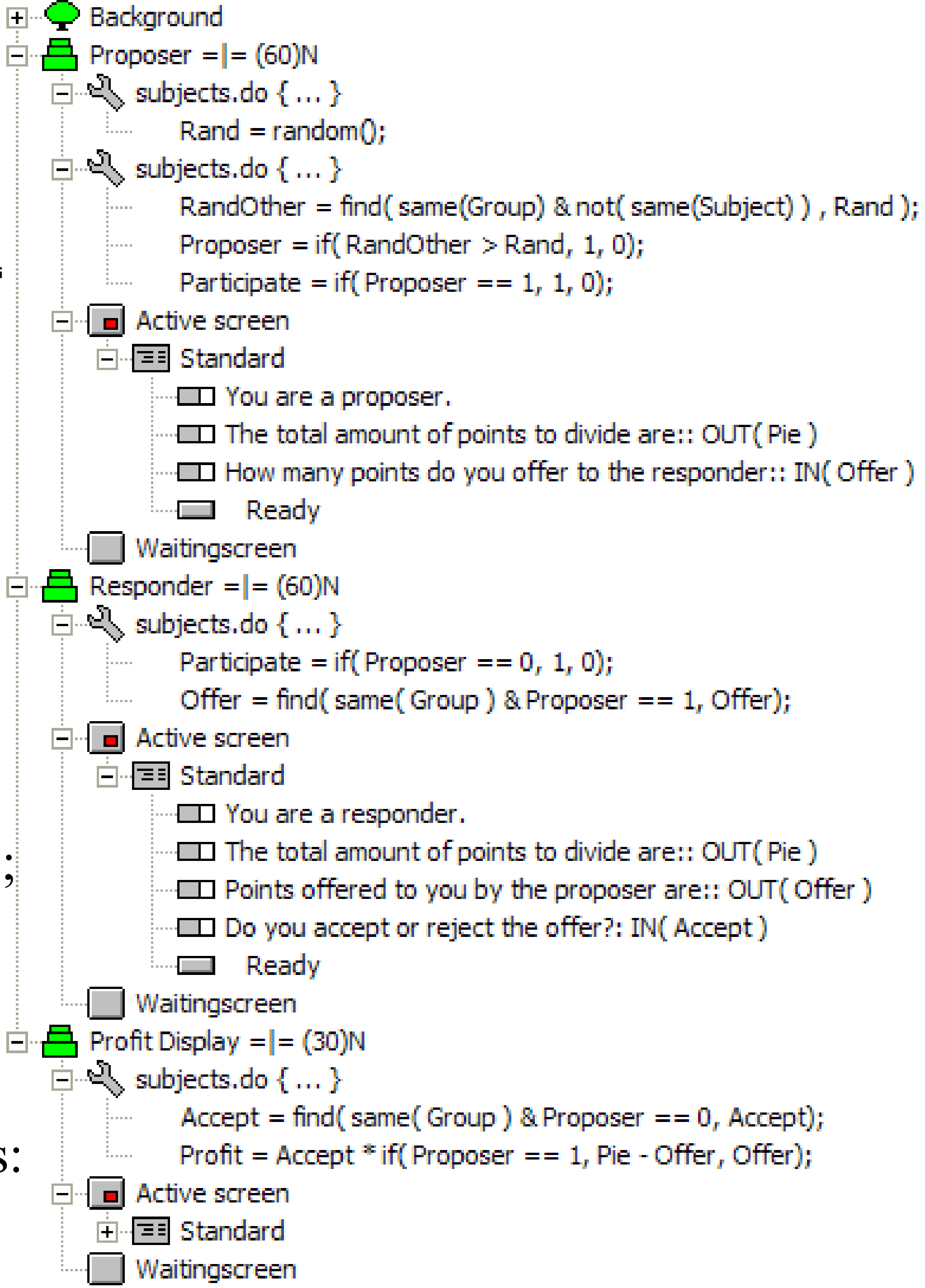
- For the ultimatum game we use:

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

Additionally:

- For the input of the responder’s decision we can use radio buttons:

  !radio: 0=“Reject”; 1=“Accept”;



```
⊞ Background
⊟ Proposer =|= (60)N
    ⊟ subjects.do { ... }
          Rand = random();
    ⊟ subjects.do { ... }
          RandOther = find( same(Group) & not( same(Subject) ) , Rand );
          Proposer = if( RandOther > Rand, 1, 0);
          Participate = if( Proposer == 1, 1, 0);
    ⊟ Active screen
        ⊟ Standard
              You are a proposer.
              The total amount of points to divide are:: OUT( Pie )
              How many points do you offer to the responder:: IN( Offer )
                Ready
      Waitingscreen
⊟ Responder =|= (60)N
    ⊟ subjects.do { ... }
          Participate = if( Proposer == 0, 1, 0);
          Offer = find( same( Group ) & Proposer == 1, Offer);
    ⊟ Active screen
        ⊟ Standard
              You are a responder.
              The total amount of points to divide are:: OUT( Pie )
              Points offered to you by the proposer are:: OUT( Offer )
              Do you accept or reject the offer?: IN( Accept )
                Ready
      Waitingscreen
⊟ Profit Display =|= (30)N
    ⊟ subjects.do { ... }
          Accept = find( same( Group ) & Proposer == 0, Accept);
          Profit = Accept * if( Proposer == 1, Pie - Offer, Offer);
    ⊟ Active screen
        ⊞ Standard
      Waitingscreen
```

Exp. 5: Another ultimatum game

- Proposers offer responders *x* points from the *y* available points.
- Responders state what is the minimum offer they would accept.
  - If the offer ≥ minimum acceptable offer:
    - Proposers earn: $\pi_P = y - x$
    - Responders earn: $\pi_R = x$
  - If the offer < minimum acceptable offer:
    - Both get 0 points.

- This is an example of using the **strategy method**.

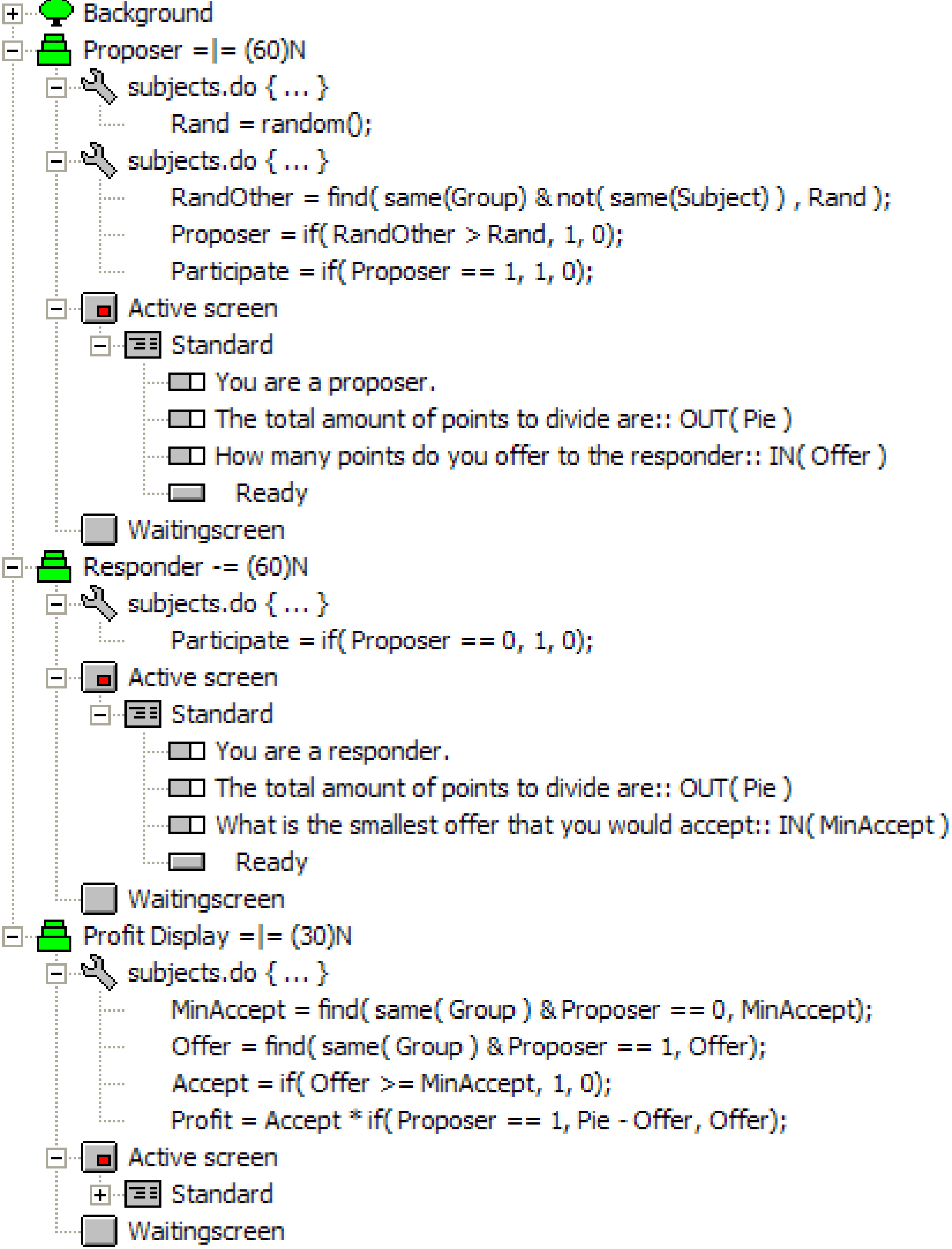
z-Tree

Stage: start options

- Proposers and responders decide simultaneously.

Stage start options:

- Wait for all
  - general case
- As soon as possible
  - simultaneous stages
  - stages that do not depend on other participants



```
Background
Proposer =|= (60)N
    subjects.do { ... }
        Rand = random();
    subjects.do { ... }
        RandOther = find( same(Group) & not( same(Subject) ) , Rand );
        Proposer = if( RandOther > Rand, 1, 0);
        Participate = if( Proposer == 1, 1, 0);
    Active screen
        Standard
            You are a proposer.
            The total amount of points to divide are:: OUT( Pie )
            How many points do you offer to the responder:: IN( Offer )
                Ready
    Waitingscreen
Responder -= (60)N
    subjects.do { ... }
        Participate = if( Proposer == 0, 1, 0);
    Active screen
        Standard
            You are a responder.
            The total amount of points to divide are:: OUT( Pie )
            What is the smallest offer that you would accept:: IN( MinAccept )
                Ready
    Waitingscreen
Profit Display =|= (30)N
    subjects.do { ... }
        MinAccept = find( same( Group ) & Proposer == 0, MinAccept);
        Offer = find( same( Group ) & Proposer == 1, Offer);
        Accept = if( Offer >= MinAccept, 1, 0);
        Profit = Accept * if( Proposer == 1, Pie - Offer, Offer);
    Active screen
        Standard
    Waitingscreen
```

Some useful matching programs

- k types of players, each group has one player of each type, strangers:

```
subjects.do{
        Type = mod( Subject – 1, k) + 1;
        RndNum = random();
}
subjects.do{
        Group = count( same(Type) & RndNum <= :RndNum);
}
```

Exp. 6: A coordination game

- Subjects are matched in pairs
  - Each pair has 1 row player and 1 column player.
- Subjects can choose between a high risk, a low risk and a no risk action. The higher payoffs are achieved when both subjects choose the same action:

	High Risk	Low Risk	No Risk
High Risk	9 , 9	0 , 3	0 , 5
Low Risk	3 , 0	6 , 6	3 , 5
No Risk	5 , 0	5 , 3	5 , 5

z-Tree

Arrays

- To calculate payoffs:

  if(Action == 1) {

      Profit = if(ActionOther == 1,
      Pay11, if(ActionOther == 2,
      Pay12, Pay13));

  }

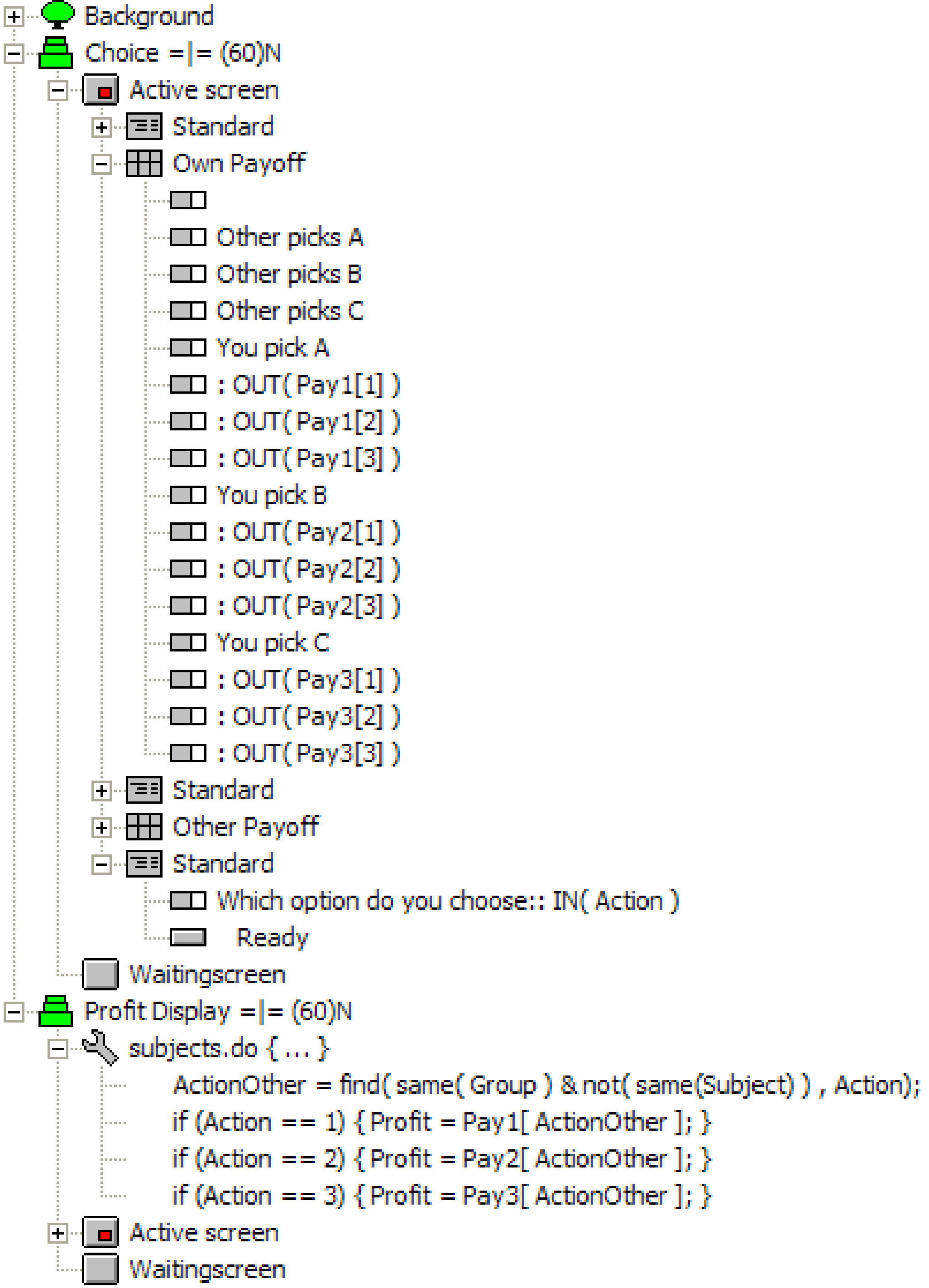
- Easier:

  array Pay1[3];

  if(Action == 1) {

      Profit = Pay1[ActionOther];

  }

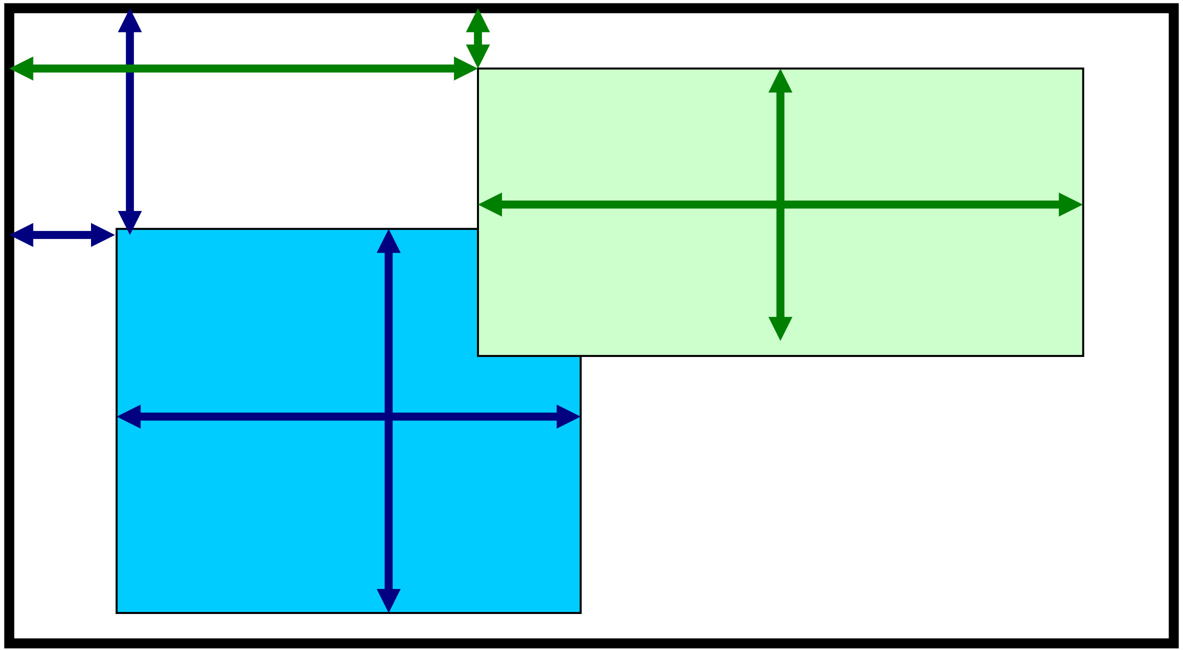


```
Background
Choice =|= (60)N
    Active screen
        Standard
        Own Payoff
            
            Other picks A
            Other picks B
            Other picks C
            You pick A
            : OUT( Pay1[1] )
            : OUT( Pay1[2] )
            : OUT( Pay1[3] )
            You pick B
            : OUT( Pay2[1] )
            : OUT( Pay2[2] )
            : OUT( Pay2[3] )
            You pick C
            : OUT( Pay3[1] )
            : OUT( Pay3[2] )
            : OUT( Pay3[3] )
        Standard
        Other Payoff
        Standard
            Which option do you choose:: IN( Action )
              Ready
    Waitingscreen
Profit Display =|= (60)N
    subjects.do { ... }
        ActionOther = find( same( Group ) & not( same(Subject) ) , Action);
        if (Action == 1) { Profit = Pay1[ ActionOther ]; }
        if (Action == 2) { Profit = Pay2[ ActionOther ]; }
        if (Action == 3) { Profit = Pay3[ ActionOther ]; }
    Active screen
    Waitingscreen
```

Boxes

Box = rectangular area of the screen containing stuff

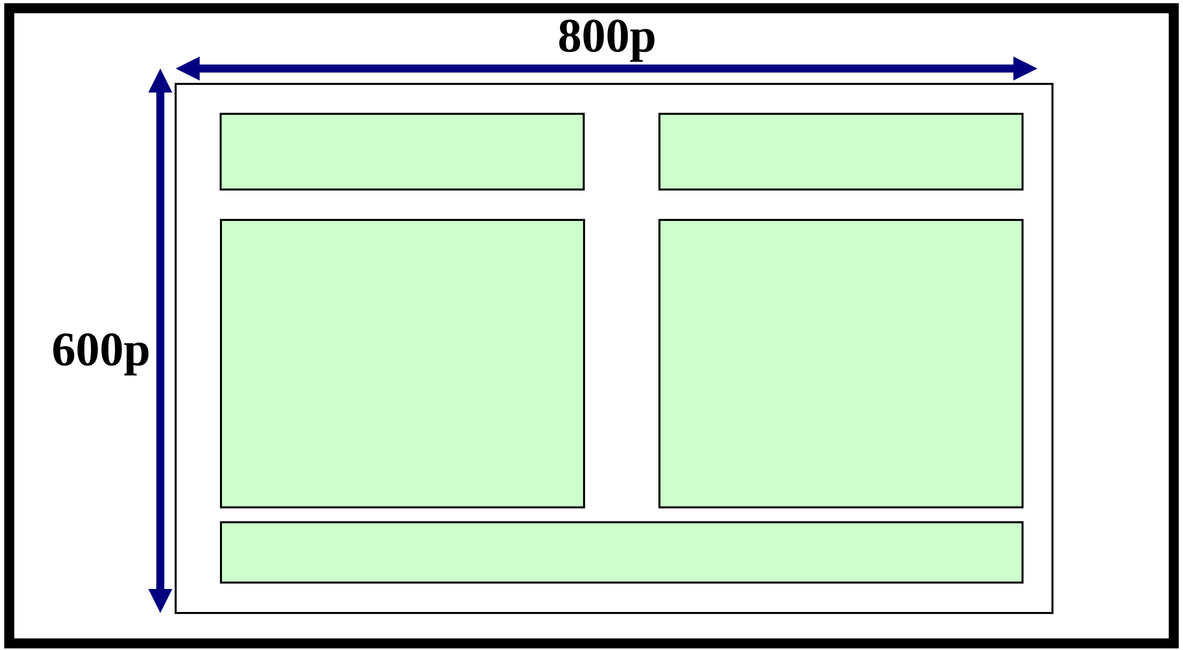
- Boxes are positioned over each other.
  - standard box
  - header box
  - help box
  - grid box
  - history box



Boxes

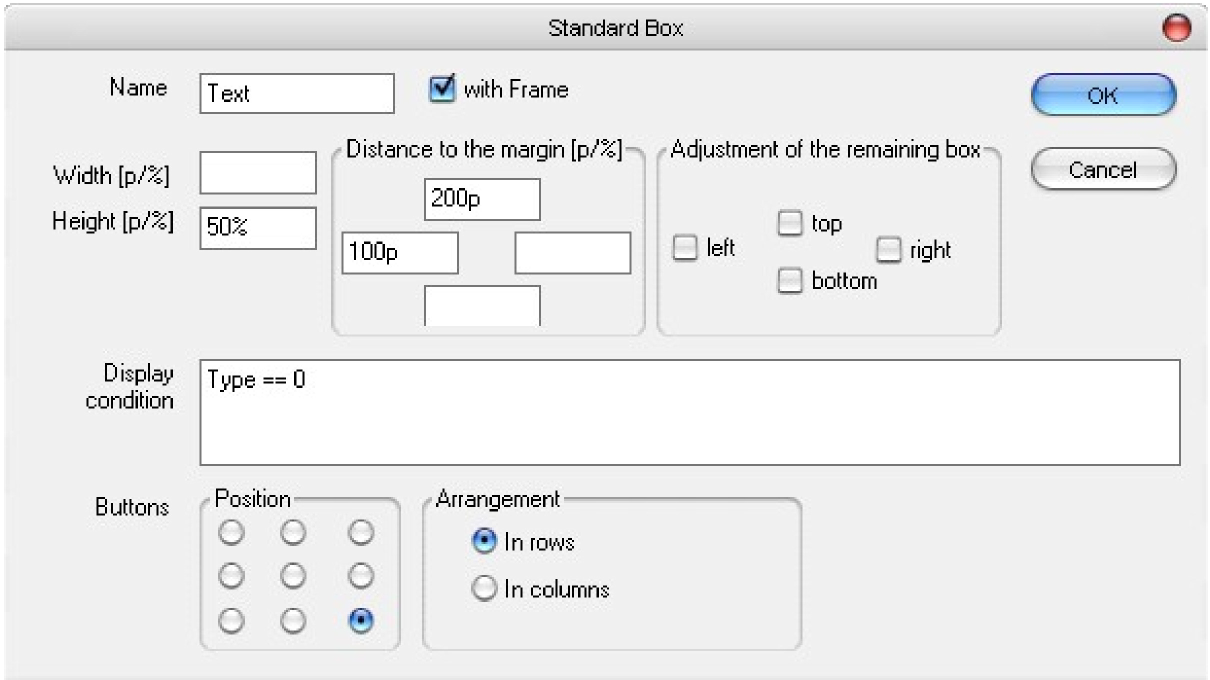
Container Box = rectangular area containing other boxes

- Very useful
  - move many boxes at the same time
  - Keep things in place with different resolutions



Boxes

- Distances can be set as % of the screen or in pixels



Standard Box

Name: Text ☑ with Frame

OK

Cancel

Width [p/%]:

Height [p/%]: 50%

Distance to the margin [p/%]

200p

100p

Adjustment of the remaining box

□ left □ top □ right □ bottom

Display condition: Type == 0

Buttons

Position

○ ○ ○
○ ○ ○
○ ○ ●

Arrangement

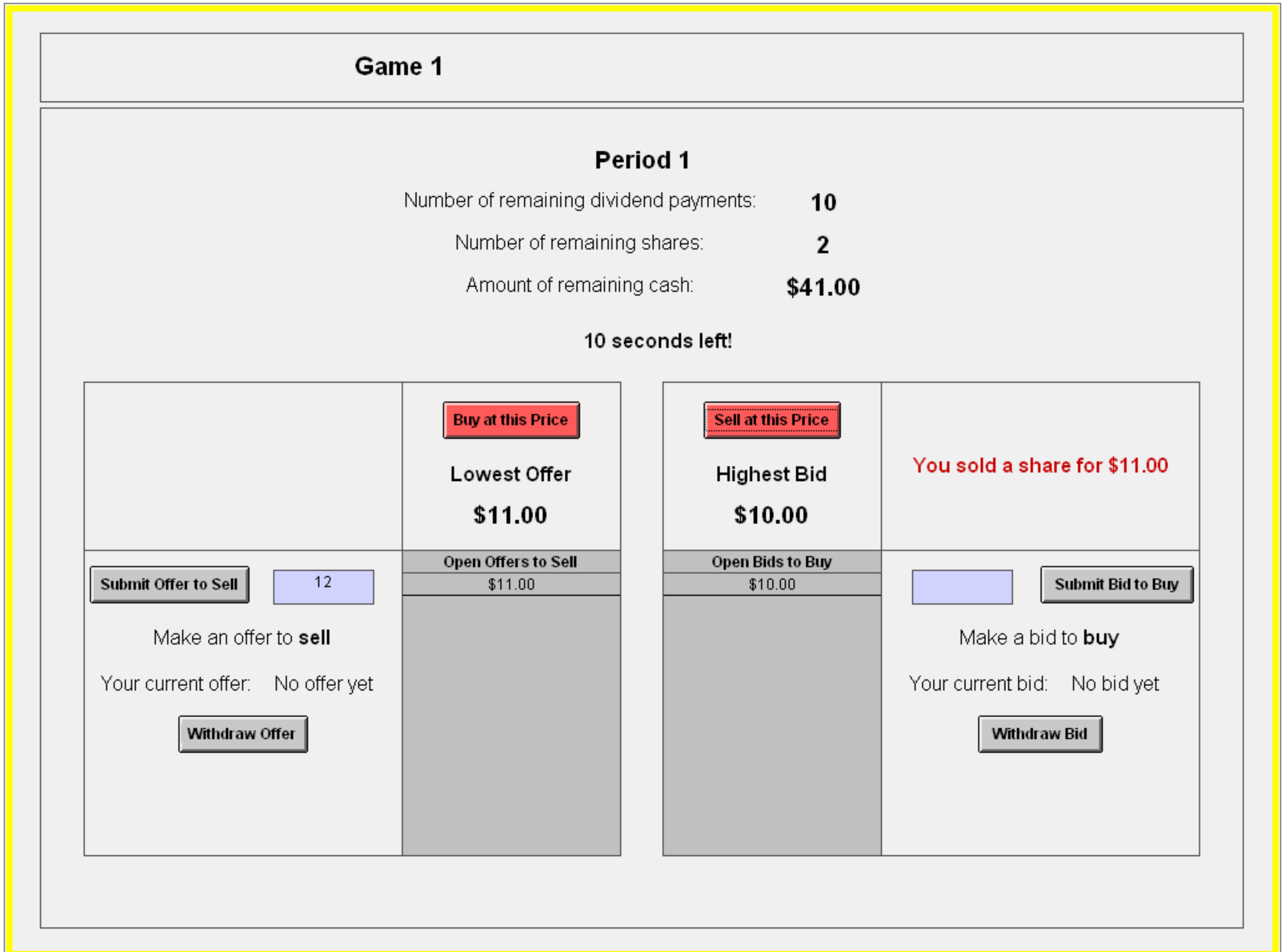
● In rows

○ In columns

- Display condition
  - Used to make boxes appear (when true) or disappear (when false)

Boxes

- Example



Game 1

Period 1

Number of remaining dividend payments: **10**

Number of remaining shares: **2**

Amount of remaining cash: **$41.00**

10 seconds left!

Lowest Offer

$11.00

Highest Bid

$10.00

You sold a share for $11.00

Open Offers to Sell

$11.00

Open Bids to Buy

$10.00

12

Make an offer to **sell**

Your current offer: No offer yet

Make a bid to **buy**

Your current bid: No bid yet

Variables integrated into text

- To display:

You sold a share for $10.00!

or

You bought a share for $10.00!

- Type:

<>You <Buyer |!text: 0=“sold”; 1=“bought”;>
a share for $<Price | 0.01>.

Variables integrated into text

- To display:

  Your *profit* in this period was 25.00 points.

  or

  Your *profit* in this period was **–5.00 points**.

- Type:

```
<>{\rtf Your \i profit \i0 in this period was
<Profit |!text: 1=“”; –1=“\b ”;><Profit |0.01>
points<Profit |!text: 1=“”; –1=“\b0 ”;>.}
```

- Most RTF is supported so you can do a lot of stuff

Exp. 7: A very simple auction

- Subjects are all buyers.
  - Subjects get a (random) private value for the auctioned good
  - Subjects make bids
  - Winner pays the second highest price
  - The auction is terminated after a fixed timeout
  - Winner gets: $\pi_B = y + v_i - b_2$
  - Sellers get: $\pi_S = y$
- For market experiments we need:
  - contracts table
  - new types of boxes:
    - contract creation box, contract list box, and contract grid box

Contracts table

- Table has a *flexible* number of records (records can be added).
  - New records are created in contract creation boxes.
  - or with the new command: contracts.new{ x=1; }

Buyer	Bid	Order	Remark

Contracts table

- Table has a *flexible* number of records (records can be added).
  - New records are created in contract creation boxes.
  - or with the new command: contracts.new{ x=1; }

Buyer	Bid	Order	Remark
2	10	1	Subject 2 makes a bid (highest bid)

Contracts table

- Table has a *flexible* number of records (records can be added).
  - New records are created in contract creation boxes.
  - or with the new command: contracts.new{ x=1; }

Buyer	Bid	Order	Remark
2	10	2	Subject 2 makes a bid (second highest bid)
5	12	1	Subject 5 makes a bid (highest bid)

Contracts table

- Table has a *flexible* number of records (records can be added).
  - New records are created in contract creation boxes.
  - or with the new command: contracts.new{ x=1; }

Buyer	Bid	Order	Remark
2	10	3	Subject 2 makes a bid
5	12	2	Subject 5 makes a bid (second highest bid)
4	15	1	Subject 4 makes a bid (highest bid)

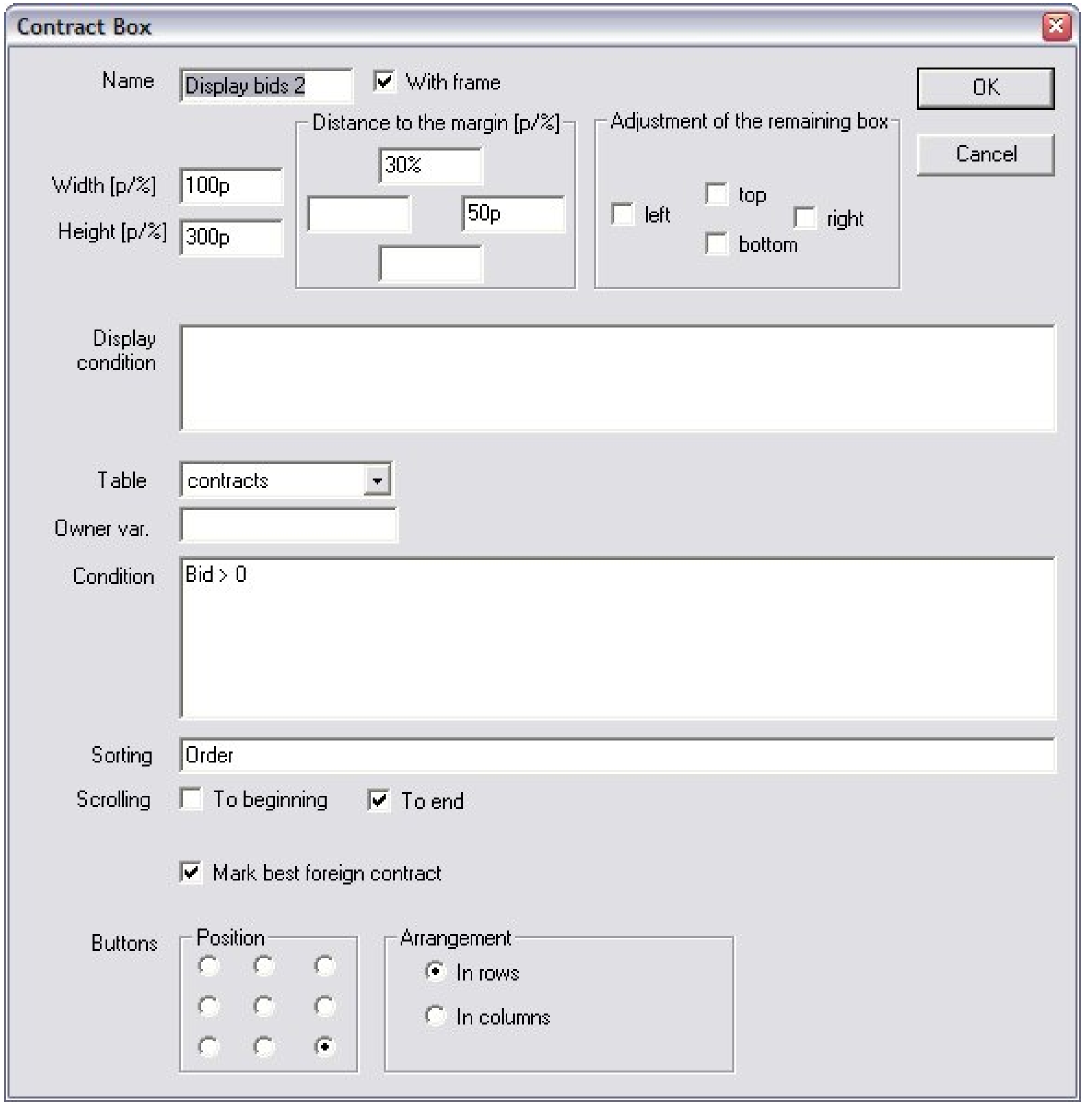
Contracts table

- Table has a *flexible* number of records (records can be added).
  - New records are created in contract creation boxes.
  - or with the new command: contracts.new{ x=1; }

Buyer	Bid	Order	Remark
2	10	4	Subject 2 makes a bid
5	12	3	Subject 5 makes a bid
4	15	2	Subject 5 makes a bid (second highest bid)
2	17	1	Subject 2 makes another bid (highest offer)

Contracts table

- The contents of the contracts table can be displayed with a *contracts list box* or with a *contracts grid box*.



Contract Box

Name: Display bids 2 ☑ With frame

Width [p/%]: 100p

Height [p/%]: 300p

Distance to the margin [p/%]: 30% (top), (left empty), 50p (right), (bottom empty)

Adjustment of the remaining box: □ left □ top □ right □ bottom

Display condition:

Table: contracts

Owner var.:

Condition: Bid > 0

Sorting: Order

Scrolling: □ To beginning ☑ To end

☑ Mark best foreign contract

Buttons

Position:
○ ○ ○
○ ○ ○
○ ○ ●

Arrangement: ● In rows ○ In columns

Exp. 8: A continuous public good game

- In each period each subject gets 20 points.
  - Points can be kept or invested in a public good
  - Each point invested in the public good pays 0.5 to everyone.
- The profit of each subject is:

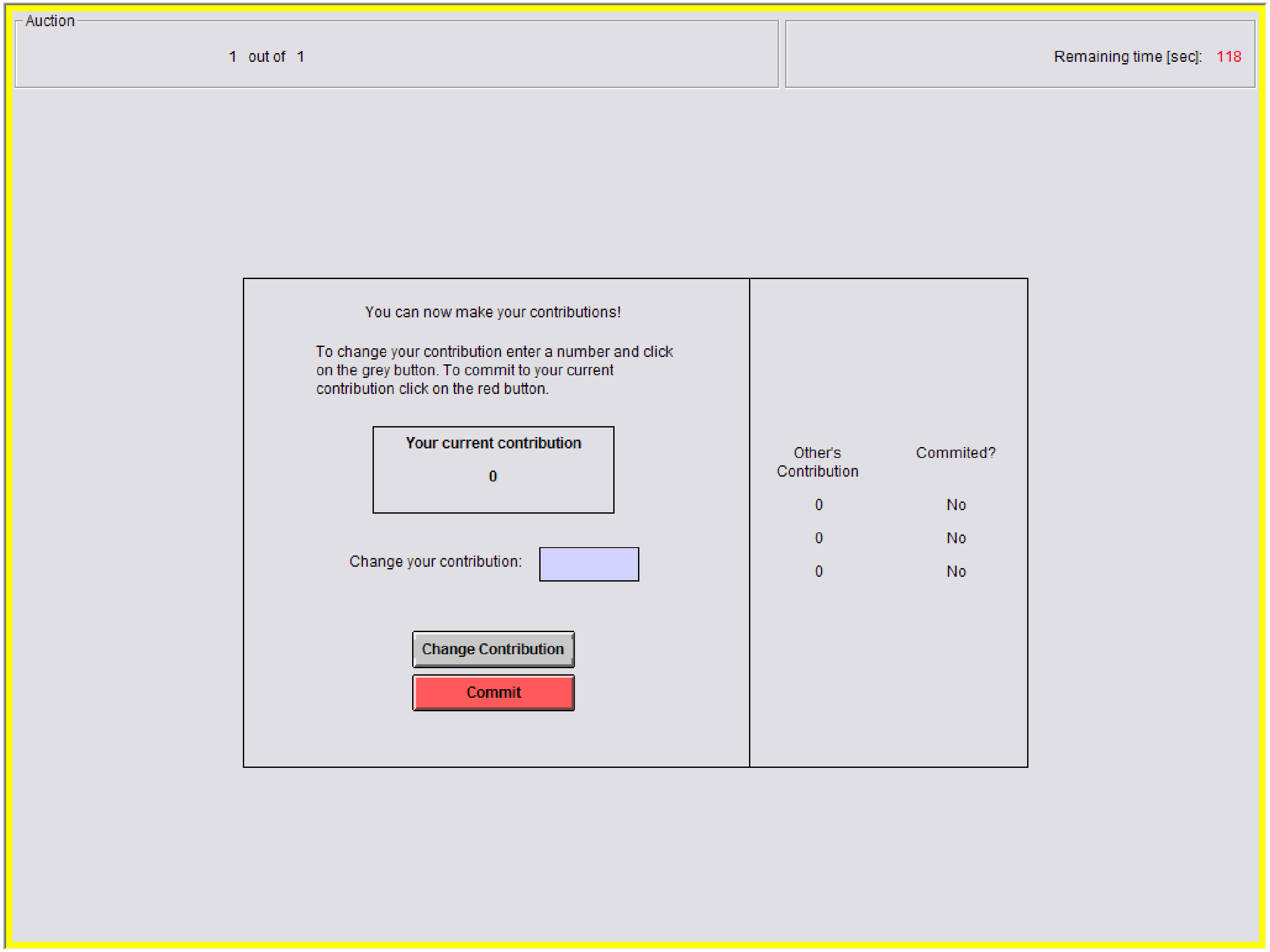
$$\pi_i = 20 - c_i + 0.5 \times \sum_j c_j$$

- The game is played for 2 periods.

- There are 90 sec to make **non-binding** contributions.
- Contributions become binding when time expires or when the subject chooses to commit him/herself.
- Contributions are observer on real-time by everyone.

z-Tree

Exp. 8: A continuous public good game



Auction

1 out of 1

Remaining time [sec]: 118

You can now make your contributions!

To change your contribution enter a number and click on the grey button. To commit to your current contribution click on the red button.

Your current contribution

0

Change your contribution:

Other's Contribution	Commited?
0	No
0	No
0	No

More contracts table

- Note that the contracts table can also be used for interaction within the same screen.
  - Use the new command to create the table
  - Use contract grid boxes
  - Important: Changes to variables during the screen are NOT recorded in the data

Other Features

- Programming
  - Loops: while(condition) { statements; }
- Complex move structures
  - goto next stage if …
- Treatments with indefinite length
  - end with a given probability
  - end when a specific action is taken
- Graphics
  - Charts
  - Display Pictures/Videos
- Communication
  - Chat box

Questionnaires

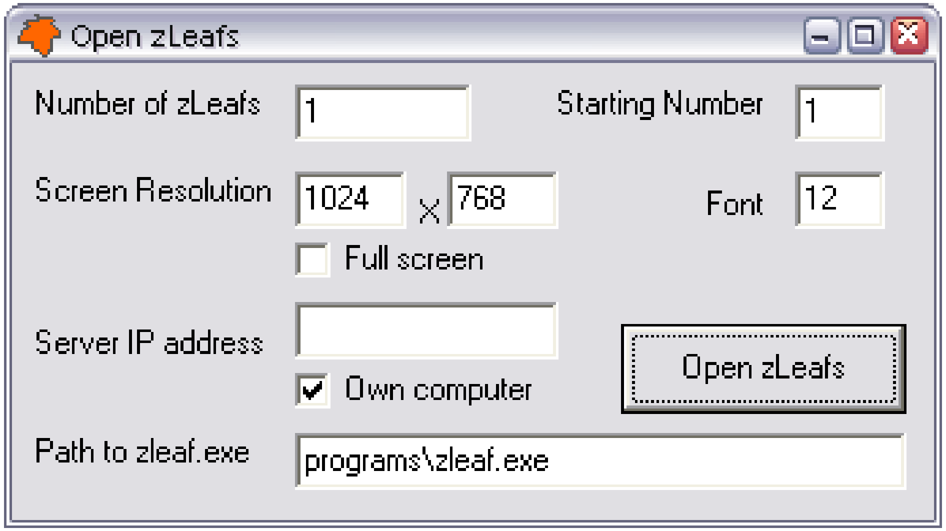
- Must be run so that the **payoff** file is written.
- Questions with no consequence on payoff.
  - Different formats for the questions.
  - Layout is not screen oriented: indefinite end with scrollbar.
  - Text entry possible.
- Typical Questionnaires:
  - Address form (writes the payment file)
  - Questions concerning their strategies
  - Profit display
  - Goodbye screen

Planning a simple session

- Welcome treatment (welcome.ztt)
    - Set the show-up fee
    - Control questions
- Public goods experiment (pg.ztt)
    - The main treatment
- Ultimatum game (ug.ztt)
    - A second treatment
- Questionnaires and payment (end.ztq)
    - payment file

How to build a test environment

- Unzip ztree.zip folder.
  - If they are not there, you need to copy the files ztree.exe and zleaf.exe to the folder “programs”
- Open ztree with the batch file: “openztree.bat”
- Open the file: “Open Zleafs.exe”
  - Set as many zLeafs as necessary
  - If needed, change screen resolution and other options



Open zLeafs

Number of zLeafs: 1

Starting Number: 1

Screen Resolution: 1024 × 768

Font: 12

□ Full screen

Server IP address:

☑ Own computer

Path to zleaf.exe: programs\zleaf.exe