# Programming and Conducting Experiments with z-Tree

Slides by **Silvio Ravaioli**

Columbia Experimental Laboratory for Social Science https://celss.iserp.columbia.edu/

Last update: **October 2019**. Based on the slides created by Han Huynh (Columbia University), Ernesto Reuben (New York University) and Mark Pigors (University of Cologne)

All errors are my own. If you find any, or you have suggestions to improve the slides, please contact me: sr3300@columbia.edu

# (Before) Running an Experiment

- Before diving into coding
- Think HARD about your experiment
- Coding and thinking can help each other but …
- … thinking hard about your experiment BEFORE coding will save you time (and money)
- Because you are actually creating data, think through your experiment from start to finish
  - Coding/Testing
  - Production/Data storage
  - Payment

# (Before) Running an Experiment

- What is the nature of my experiment?
  - Individual decision making or strategic?
- What is the scale of my experiment?
  - Do I foresee my experiment to go beyond the lab?
- What are the treatments?
- What are the variables I want to collect?
- What kind of (technical) support do I need?

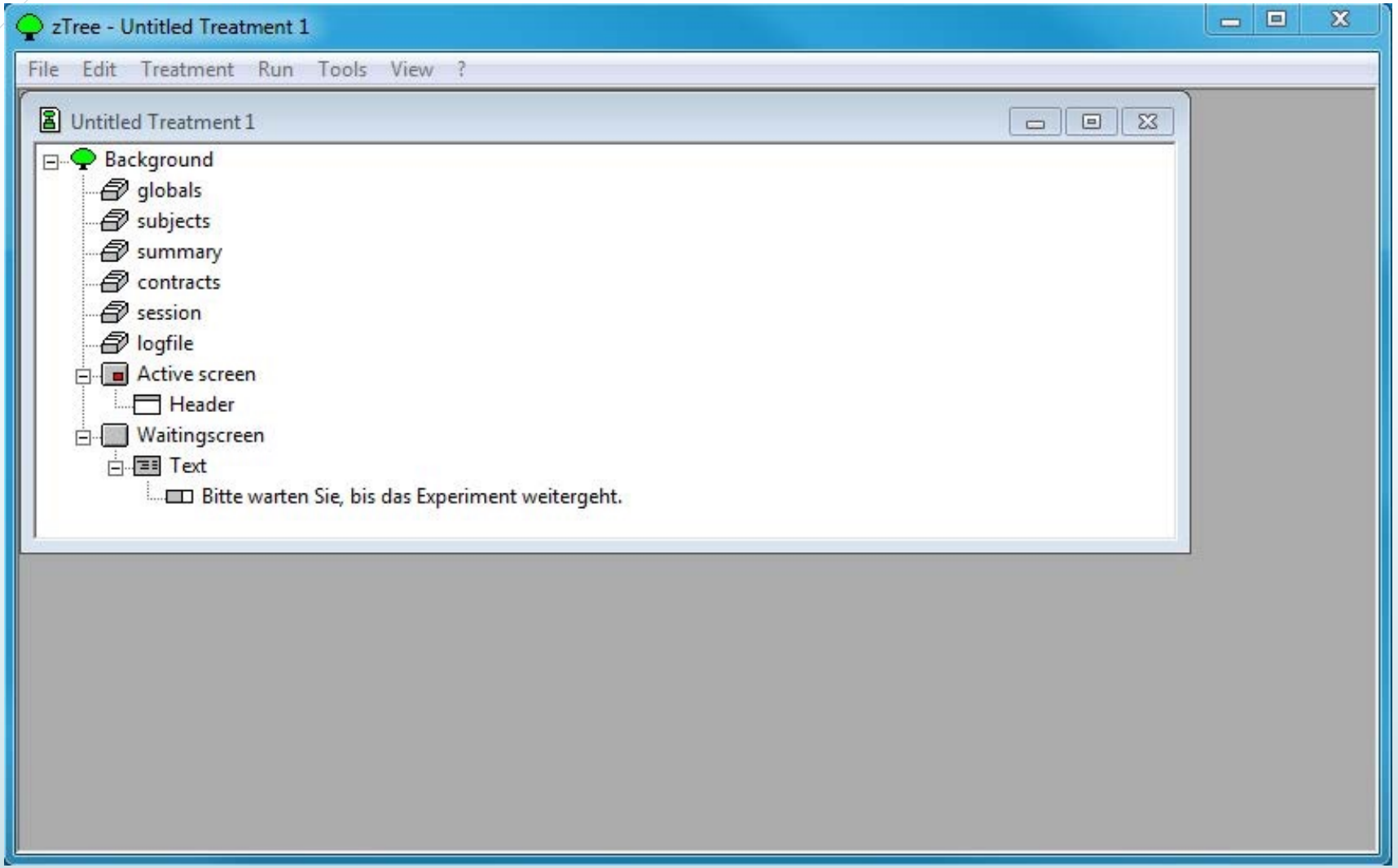
# (Before) Running an Experiment

- **zTree**
  - **Restrictions in interface. No scope beyond the lab**
  - **Testing and production are easy for "standard" experiments**
- **oTree**
  - Require quite a bit of programming knowledge: Python, HTML, Javascript. Great flexibility
  - It's online so scaling is easy. Testing is ok but production is challenging
- **Qualtrics**
  - Annoying coding experience since the server is slow. You need someone to pay for the account
  - Just a little knowledge of HTML and Javascript can give you a lot of flexibility
  - Significantly upgraded survey monkey. Testing and production are extremely easy
- **PsychToolBox (Matlab PTB)**
  - Based on Matlab: it is like learning "HTML for Matlab". Amazing flexibility (even more than oTree!)
  - People in psychology/neuroscience use it a lot. Less common in economics (fewer example codes to use)
  - No scope beyond the lab. Coding strategic experiments is difficult (but possible)

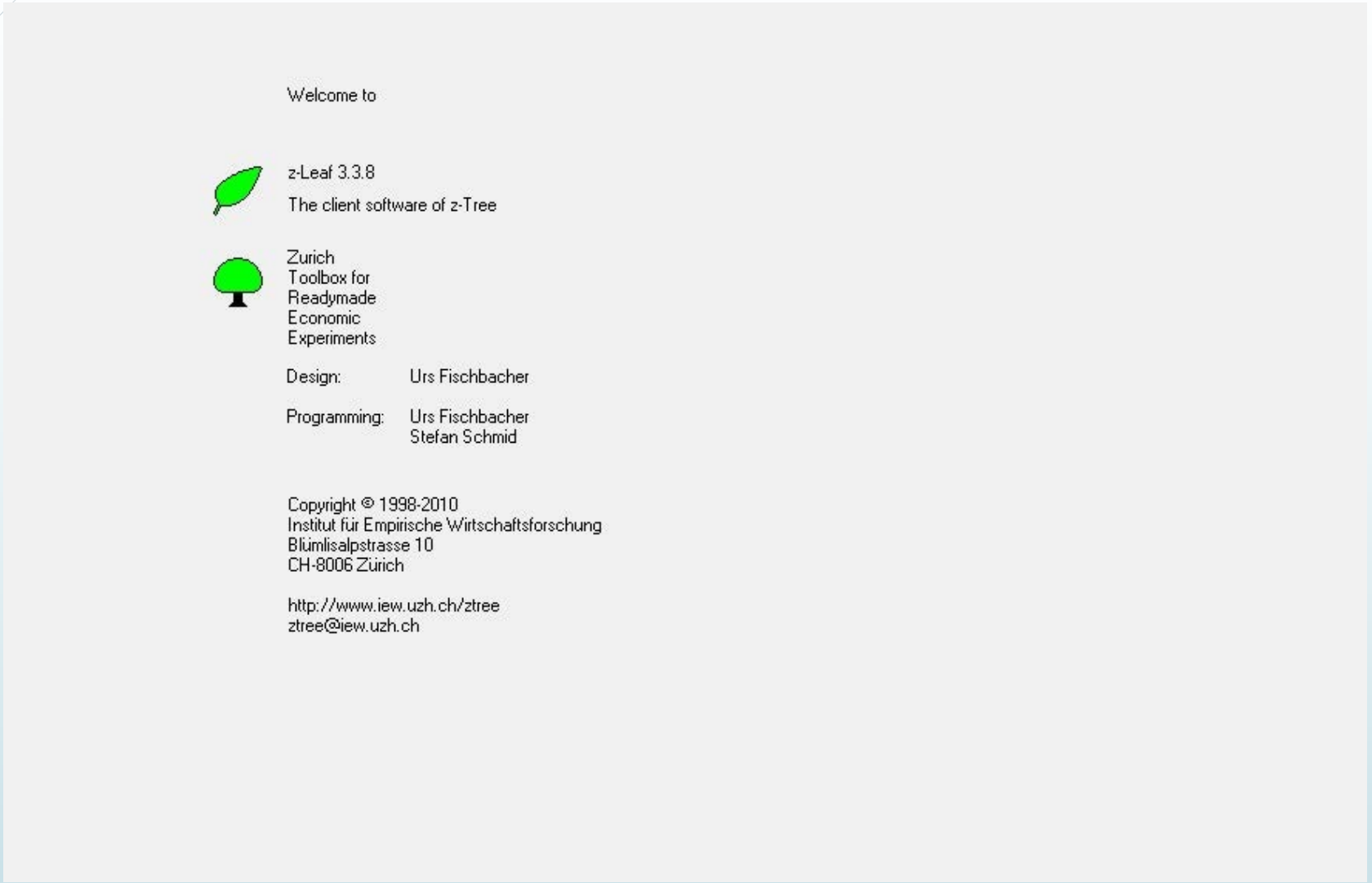
# zTree – Getting Started

- Zurich Toolbox for Ready-made Economic Experiments
- Designed to enable the conduction of economic experiments without much prior experience
- Two parts:
  - **Z-Tree**: to define and conduct experiments (server program)
  - **Z-Leaf**: program used by the subjects (client program)

# zTree Screen



# zLeaf Screen



# zTree – Getting Started

- Free license: https://www.uzh.ch/ztree/ssl-dir/index.php
- Reference manual: https://www.ztree.uzh.ch/static/doc/manual_v4.pdf
- Support and Mailing list: https://www.uzh.ch/cmsssl/ztree/en/support.html
- Ready-made experiments: https://www.ztree.uzh.ch/en/examples.html
- Other resources:
  - Ernesto Reuben: http://www.ereuben.net/teach/
  - Maria Bigoni: https://sites.google.com/site/ztreenotes/home

# zTree – Getting Started

- zTree is Windows only, but you can use this trick to run it on a Mac
- http://economistry.com/2013/05/run-z-tree-on-a-mac/

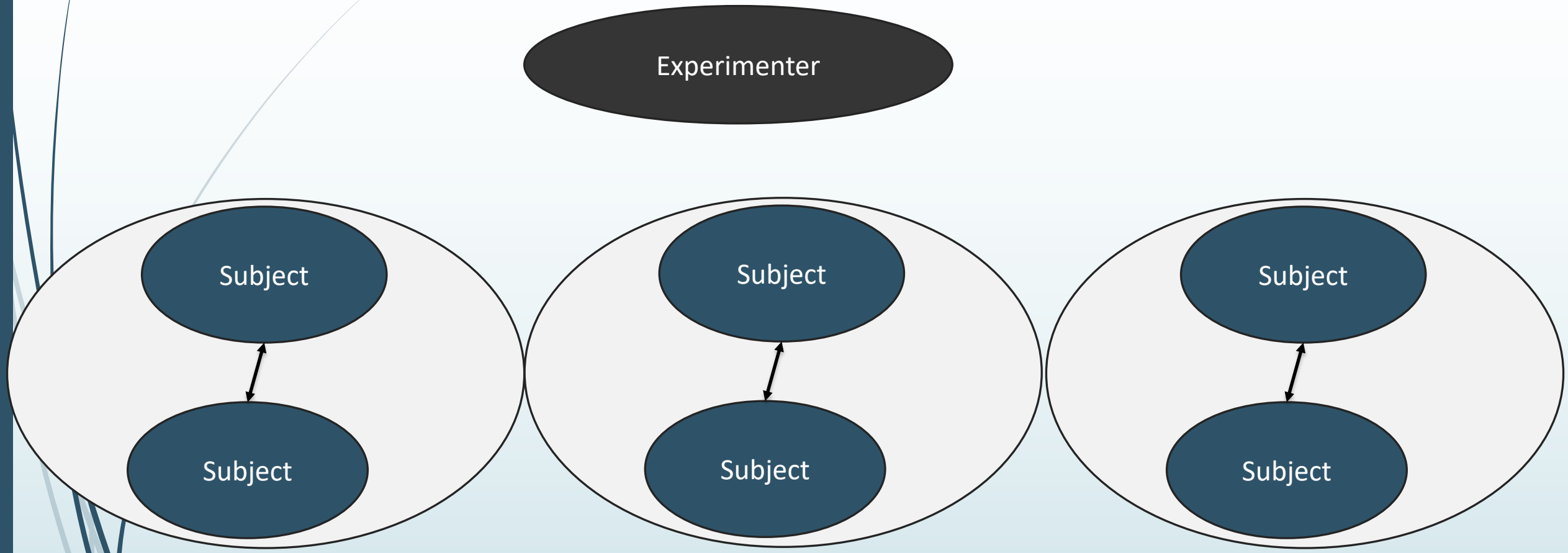
- Setting up a test environment: no need for several computers!
- Start zTree and more than one zLeaf on one computer
- You have to give the zLeaves different names

# Conducting Experiments in zTree

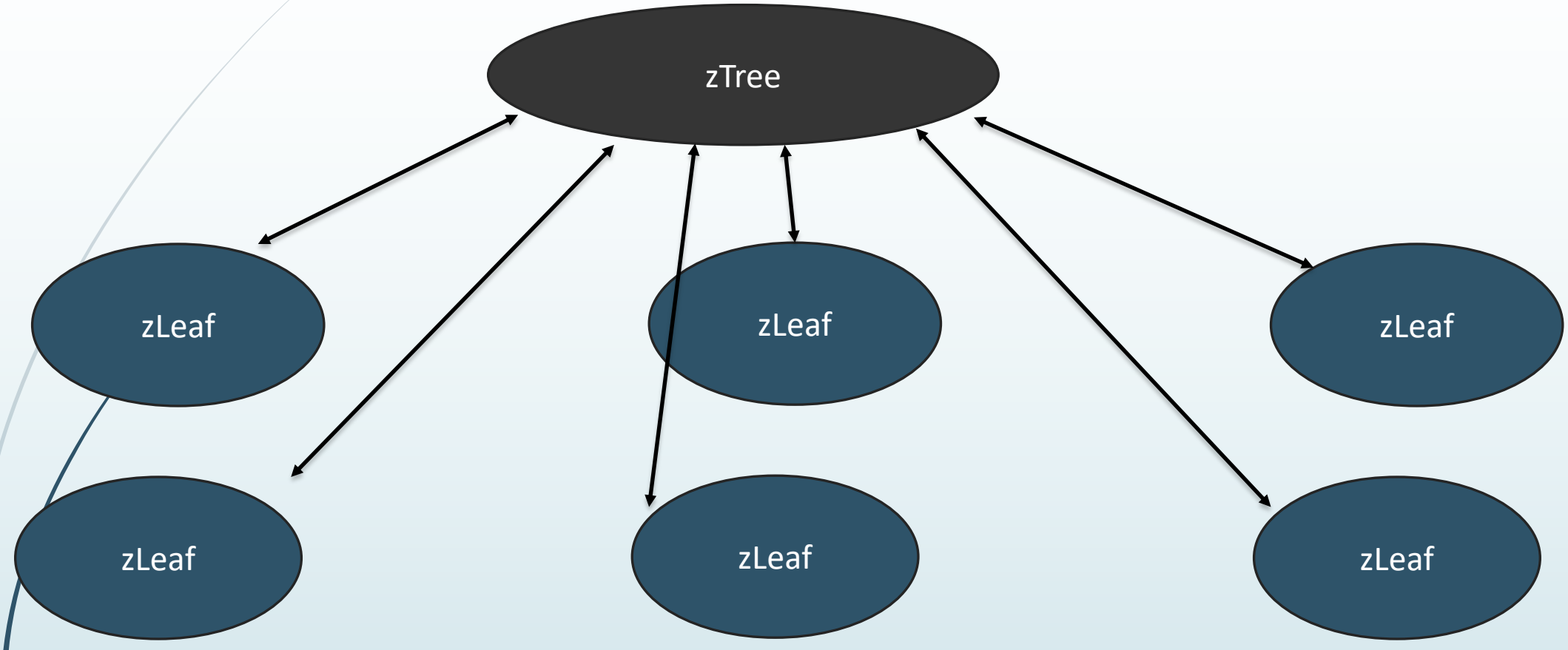
# Conducting Experiments in zTree - Roadmap

- **Terminology**: Session, Treatment, Period,…
- **Tables**: How you observe the experiment in real time
- **Examples**: 3 simple demos
- **Files saved**: Data, Payment, Questionnaire information,…
- **Easy steps** to run a session (without knowing anything else)

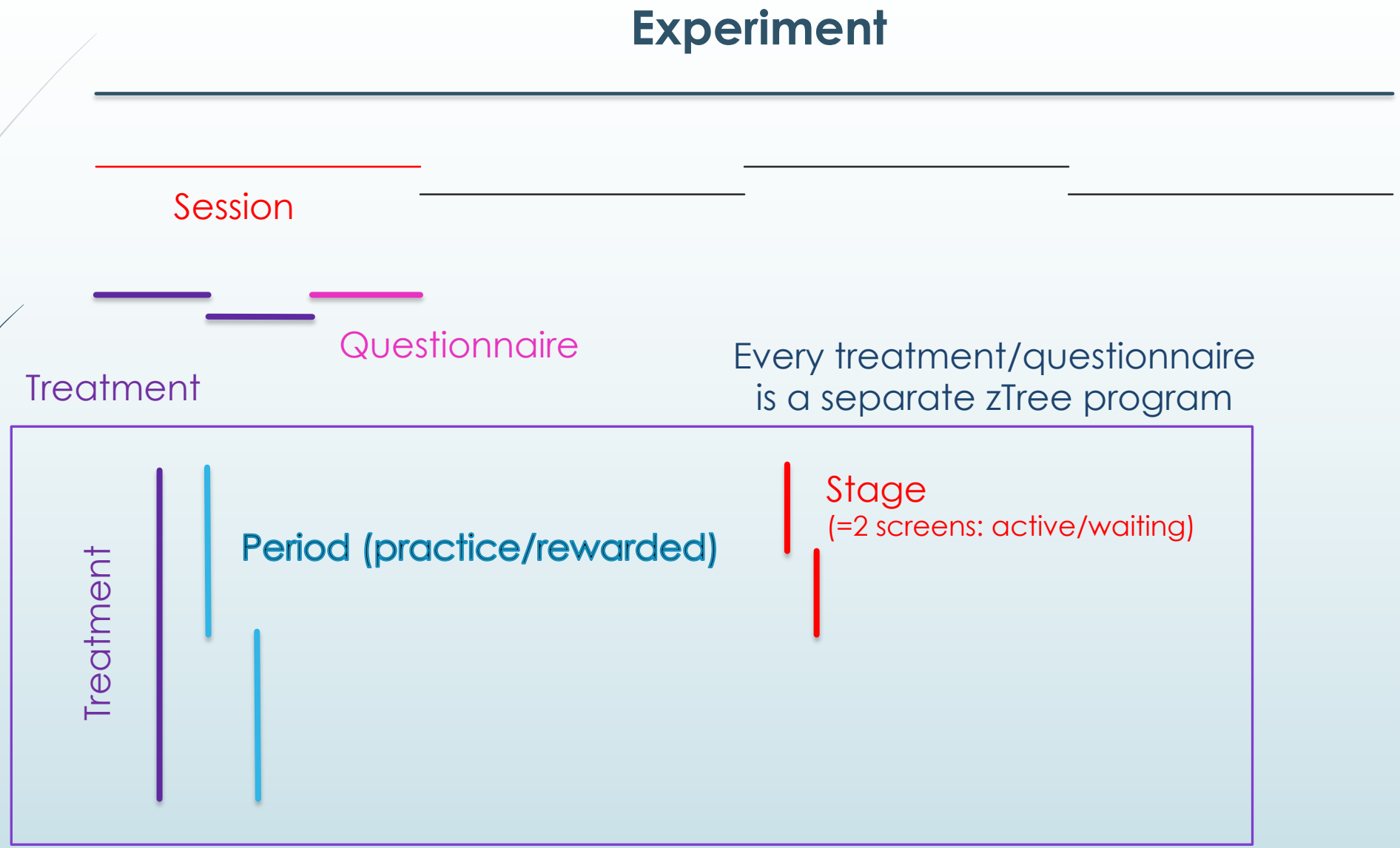
# Structure of Experiments



# Structure of Experiments



# Terms


Experiment
Session
Questionnaire
Treatment
Every treatment/questionnaire
is a separate zTree program
Treatment
Period (practice/rewarded)
Stage
(=2 screens: active/waiting)

# Tables

**Clients' table**

- Shows the state of every Leaf

Clients' Table

| 2 clients | state | time |
|---|---|---|
| first | - Contribution Entry - | 17 |
| second | *** Contribution Entry *** | 17 |

**Global table** (one record)

- Same for every subject
- Default variables: Period, NumPeriods, RepeatTreatment

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 |

**Subjects table** (one record for every subject)

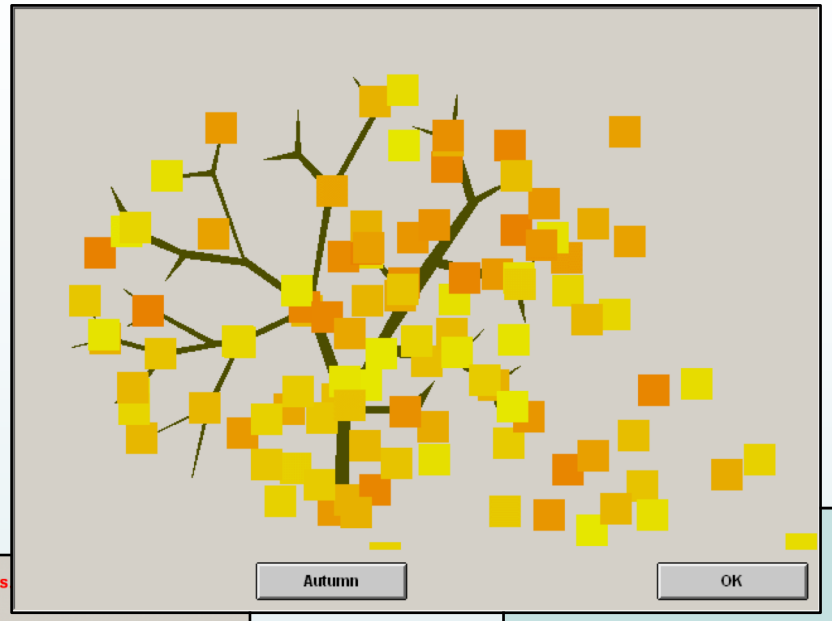
- Default variables: Period, Subject, Group, Profit, TotalProfit, Participate

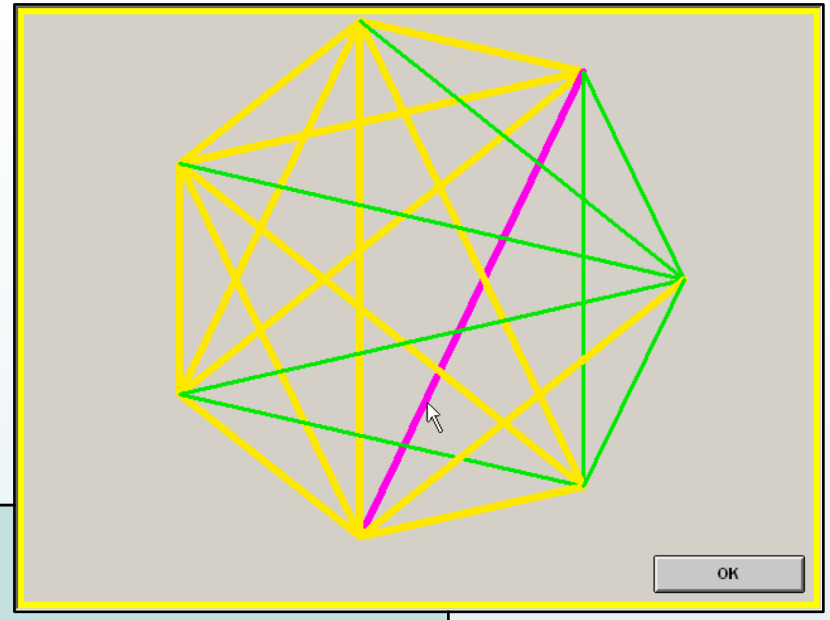
# Three Simple Demos

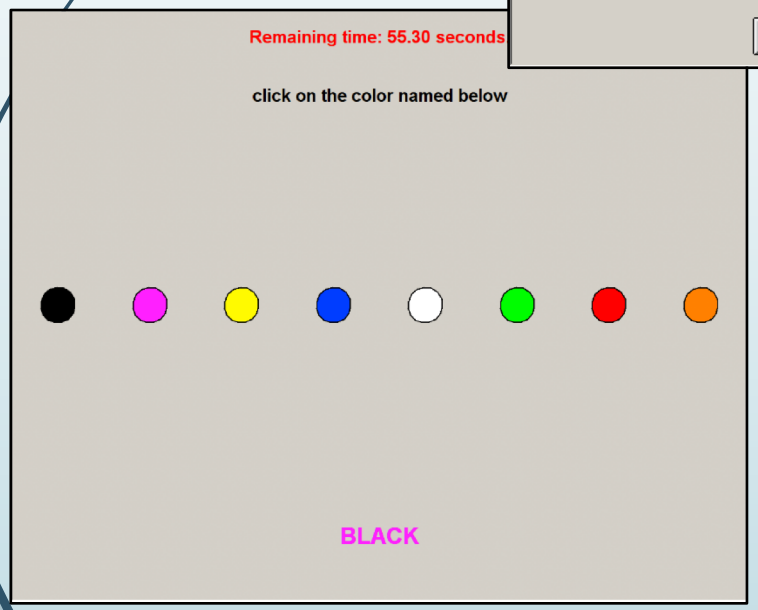
3 examples from zTree website (demos, not full experiments)

- **Beauty contest**
  - Every subject submits a number between 0 and 100
  - The subject who is closest to ½ of the mean is the winner
- **Descending auction (simple Dutch auction)**
  - An item (e.g. stock with private value) is sold with an auction system
  - The price starts from 100 and decreases by 5 points every few seconds
  - The first subject who accepts the offer wins the item and pays the price
- **Trading market (double auction)**
  - Every subject starts with 1000 points and 100 stocks
  - The value of the stock is unknown (e.g. it varies between subjects)
  - Participants can buy or sell stocks using a double auction
  - The treatment ends when everyone leaves the market (press the OK button)

# zTree **can** be flexible (but it takes time)




# Files saved by zTree

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

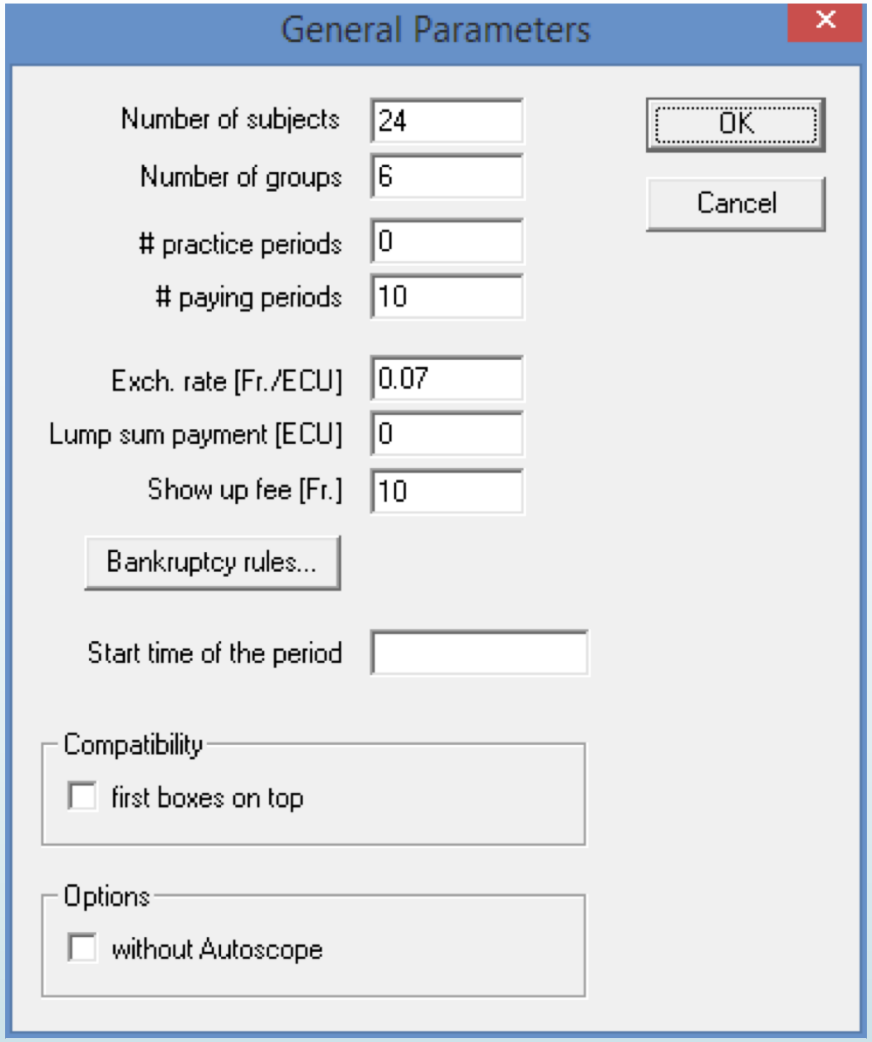
- **.pay**: the payment file, which lists the subjects' final profits including the show-up fee (you can print it for easy payment)
- **.xls**: contains all tables used in a session (subjects, globals, etc.)
- **.gsf**: backup file, in case a crash occurs
- **.adr**: subjects' addresses (from the Questionnaire)
- **.sbj**: answers to questionnaire's questions, without subjects' names

# Running a session is VERY easy

- You can run a session without knowing much!

  Just follow the steps (next slide)

- You may need to edit some parameters of the experiment (click the `Background` icon)
  - Number of subjects
  - Number of groups
  - # practice and paying rounds
  - Payment structure (dollars/experimental currency)



# Running a session is VERY easy

1. Startup of the experimenter PC, open zTree
2. Startup of the subject PCs, open zLeaf (make sure you use the same version, e.g. 3.6.7)
3. Arrival of subjects, close the extra zLeaves (if any)
4. Update the general parameters in the background (# subjects, # groups)
5. Start of the session and first treatment, observe the course of the session (Clients' table)
6. Start further treatments (if any)
7. Conclusion of the session with a questionnaire
8. Payment
9. Switch off the subject PCs
10. Download the files from the experimenter PC

# Programming in zTree

# Programming in zTree - Roadmap

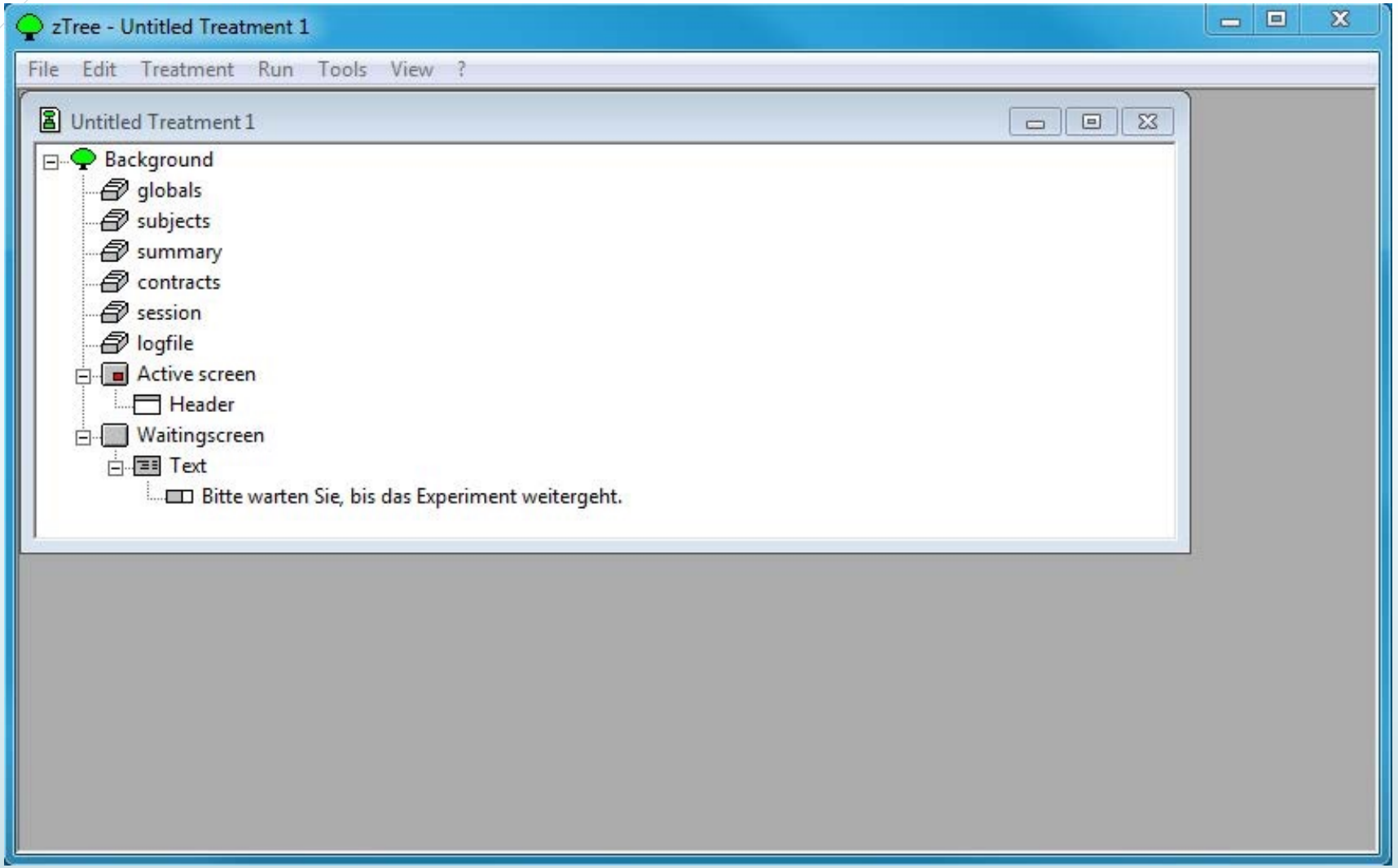
- **Experiment 1: Measuring Risk Aversion**
  Goal: Understand zTree interface and structure
- **Questionnaire and stored files**
- **Experiment 2: Public Good Games**
  Goal: understand basic features of tables, create groups
- **Experiment 3: Ultimatum Game**
  Goal: Create asymmetric roles for the subjects
- **Experiment 4: Simple Auction**
  Goal: Use the Contract table

# Experiment 1: Measuring Risk Aversion

**Goal: understand zTree interface and structure**

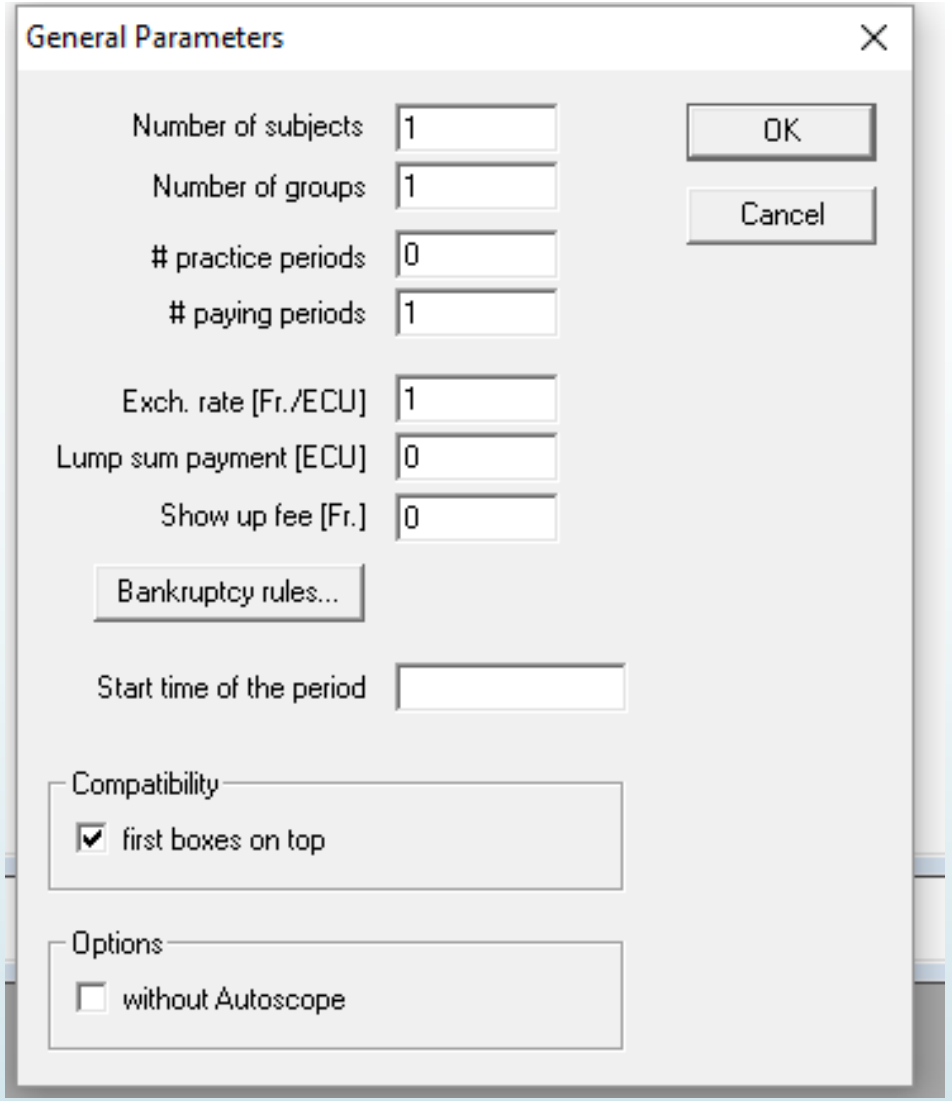
- Measuring Risk Aversion using BDM method (Becker-DeGroot-Marschak)
- Lottery: \$*0* with probability $q$ and \$$x$ with probability ($1-q$)
- Question: "State the amount of money that makes you indifferent between receiving this amount and playing the lottery" (Certainty Equivalent = CE)
- Draw $z$ randomly between *0* and $x$
  - If z>= CE, subject receives \$z
  - If z<CE, subject plays the lottery

# zTree Screen



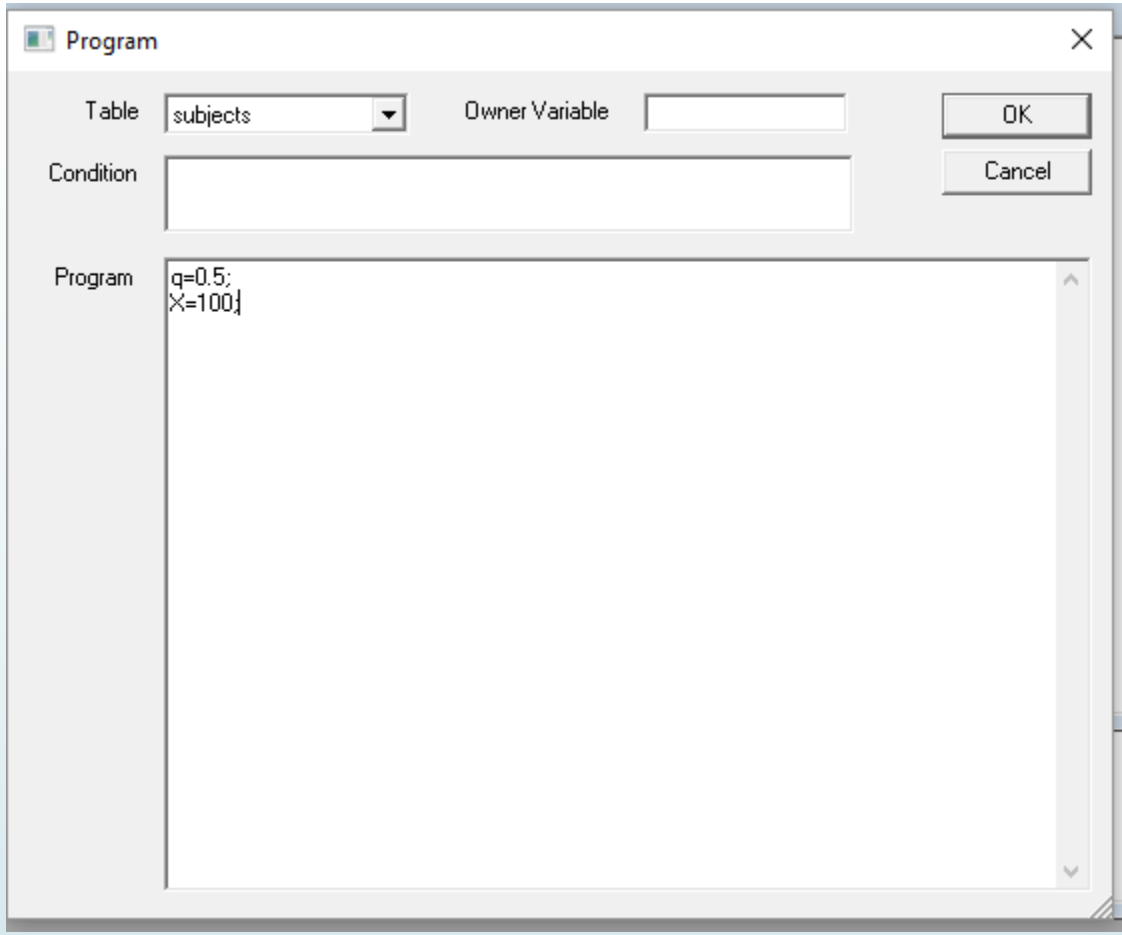
# Background

- Double click on `Background`
- Set number of subjects (1)
- Set number of groups (1)
- Set number of periods (0 + 1)



# Background

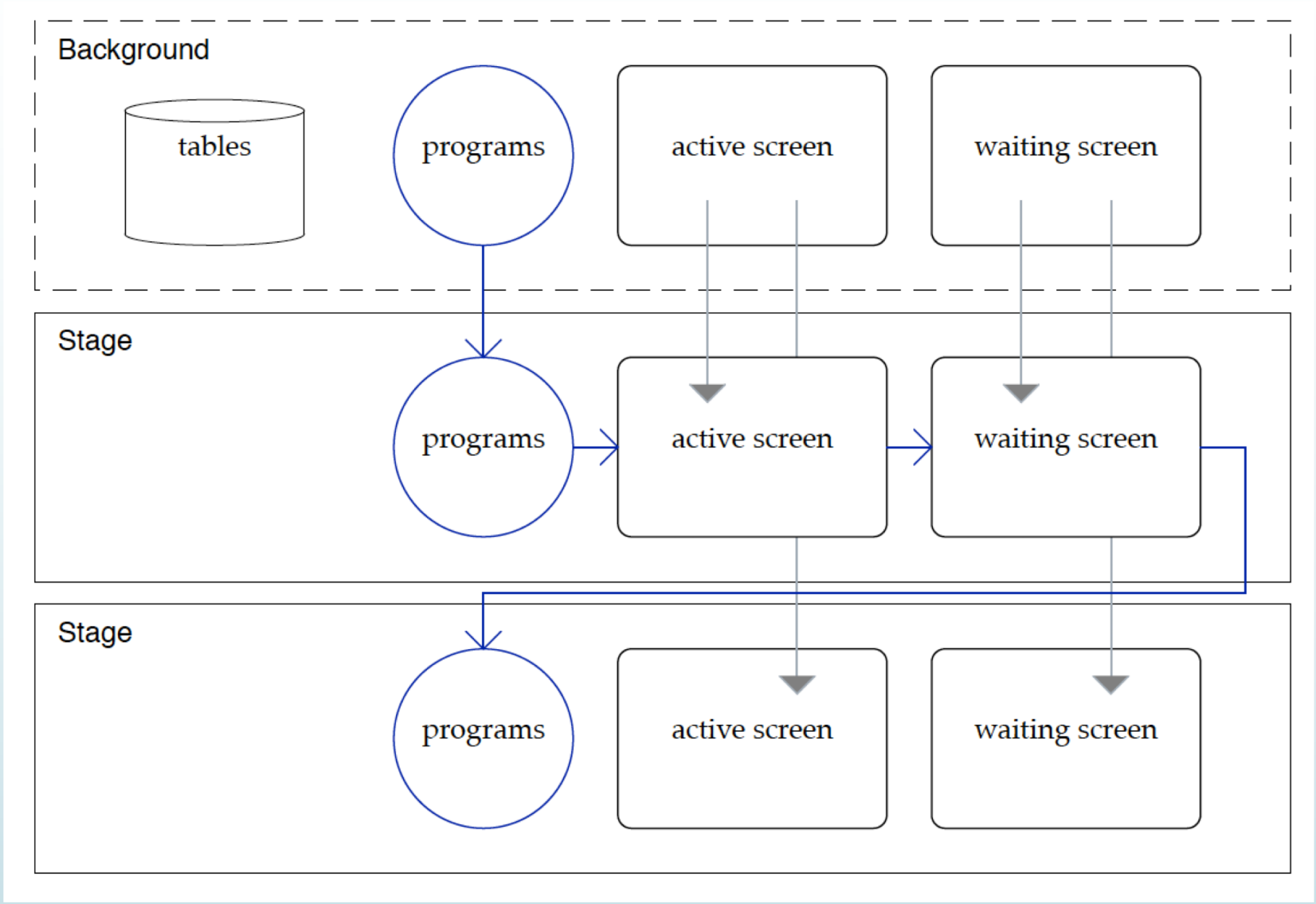
- `program` inside `background`
  - Execute at the very beginning of the treatment
- `Treatment` → `New Program`
- Need to decide at which “level” to add the program (table)
  - `globals` vs `subjects`
- In this screen: define the parameters for the lottery and save them in the `subjects` table



# Programs

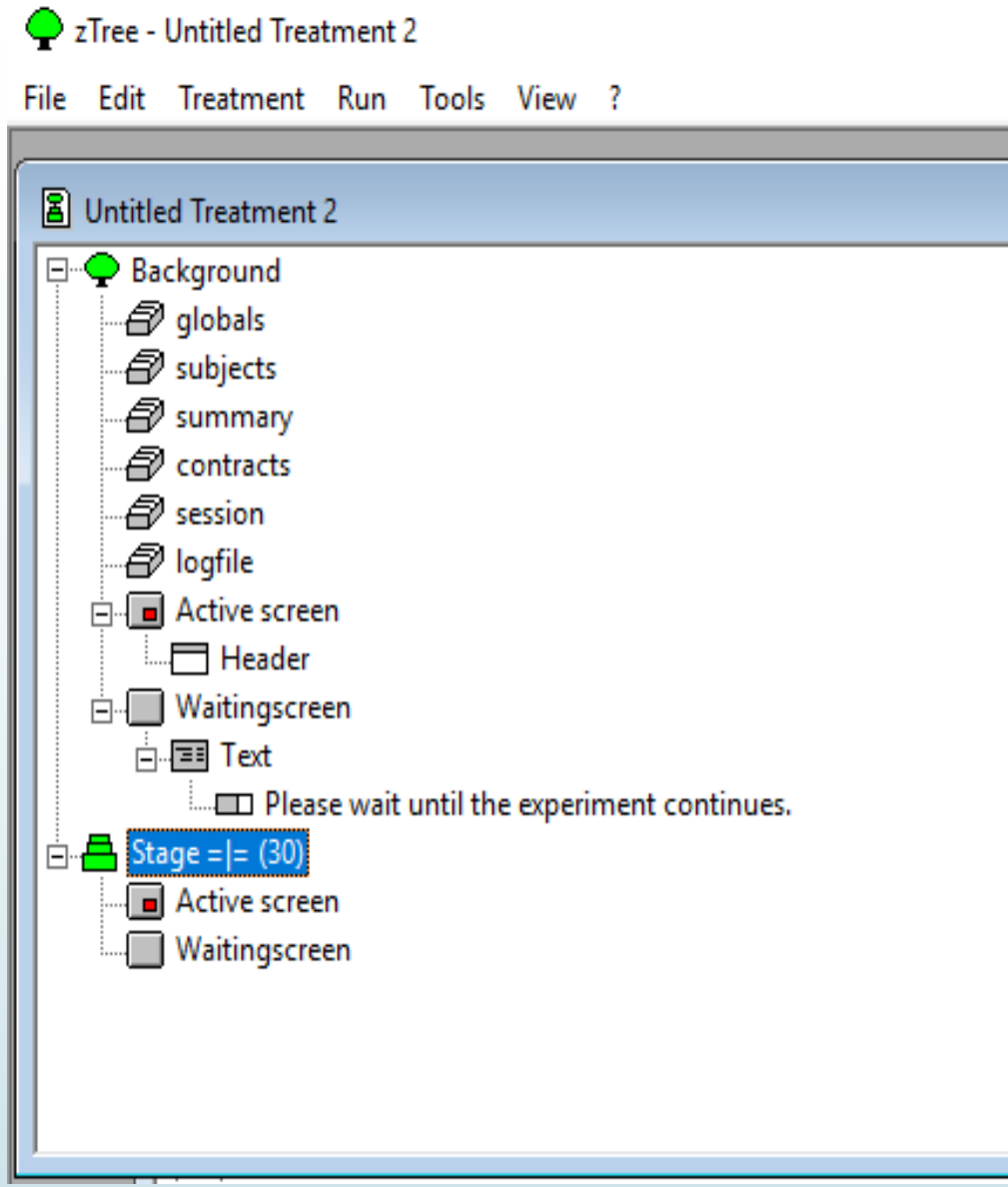
- Not very different from how you program in other platforms
- Variables, assignment `q = 0.5`
- Loops `for` and `while`, conditions `if, and, or`
- Comment on your code (starts with //)
- End statements with ;
- Useful pre-defined table functions (more later)

# Programs


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

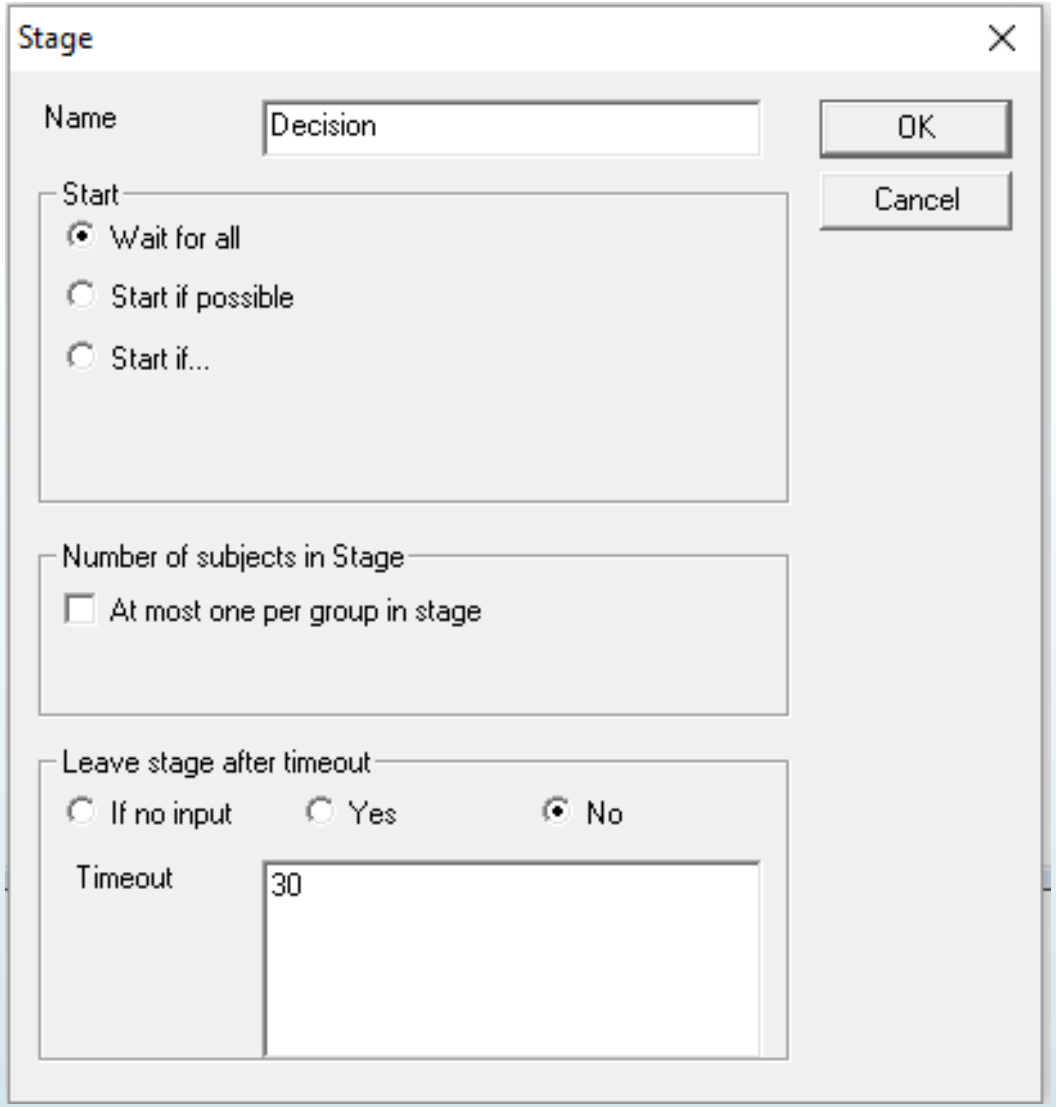
# Add a Stage

- **Parameters**: Background
- **Actions**: Stage
  - Treatment → New Stage
  - Active Screen
  - Waiting Screen



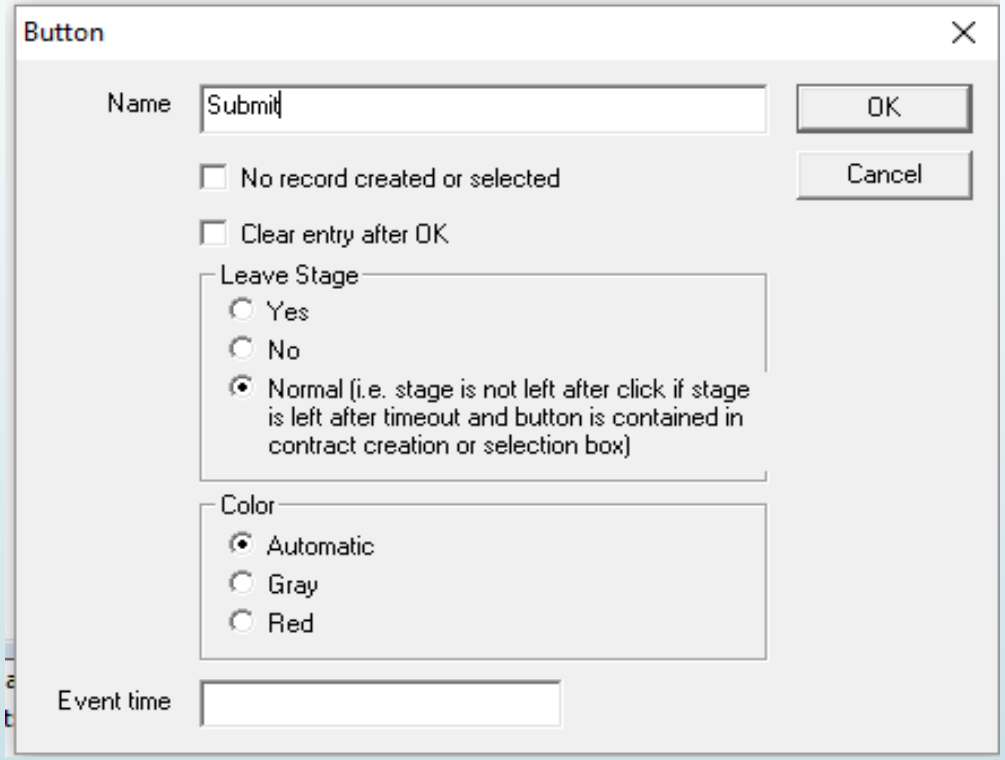
# Add a Stage

- A stage (roughly) corresponds to a screen
- No distinction between input stages (action), output stages (results, feedback), or mixed ones
- Parameters of the stage
  - Name of stage
  - Condition to start
  - Timeout



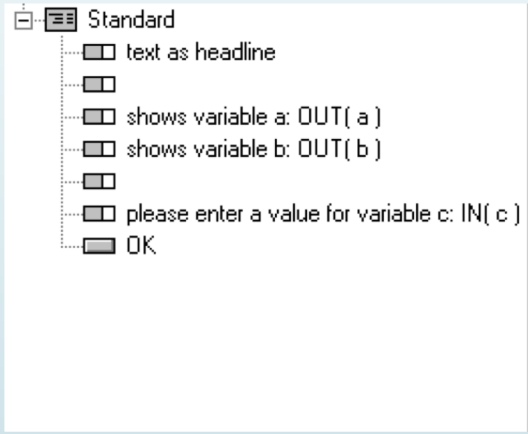
# Design the Active Screen

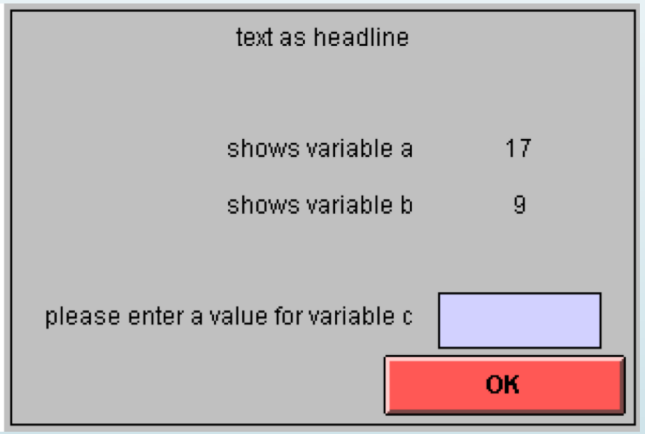
- Display your question/parameters
  - Boxes, items and buttons
- Active Screen -> Treatment -> New Box
- Box -> Treatment -> New Item
  - Used for inputs and outputs
- Box -> Treatment -> New Button
  - Especially when there is an input



# Boxes

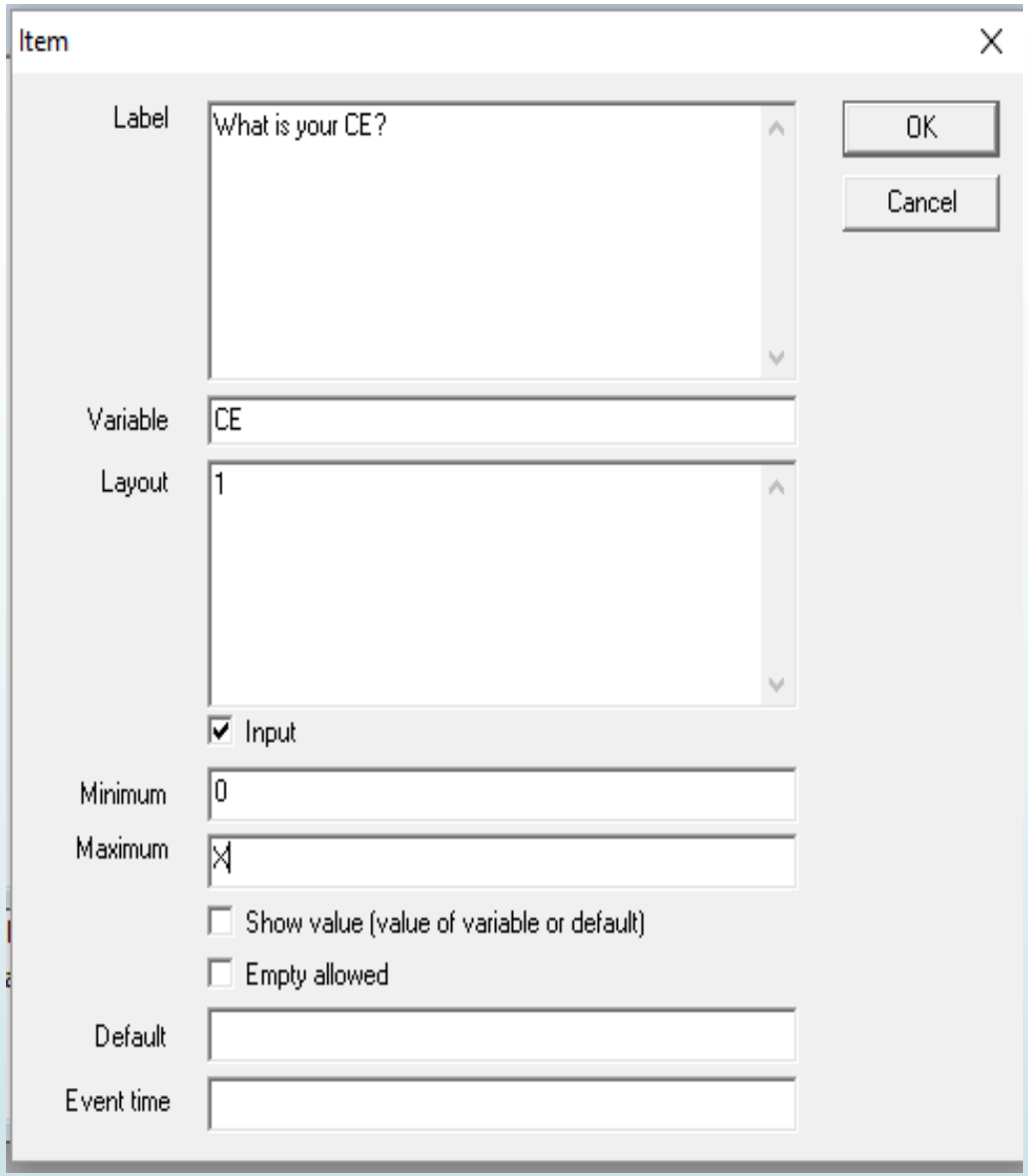
- Container box: rectangular area containing other boxes
  - Useful: keep things in place, move many boxes at the same time
- Distances can be set as % of the screen or in pixels
- Display condition
  - Used to make boxes appear (when true) or disappear (when false)
- Standard box
- Chat box, Plot box, Contract box



# Input and Output

- Label: text displayed
- Variable: use variable names to be used in programs
- Layout: format, e.g. number of digits 1/0.1/0.01
  - More options to layout
- For input: you can specify more parameters

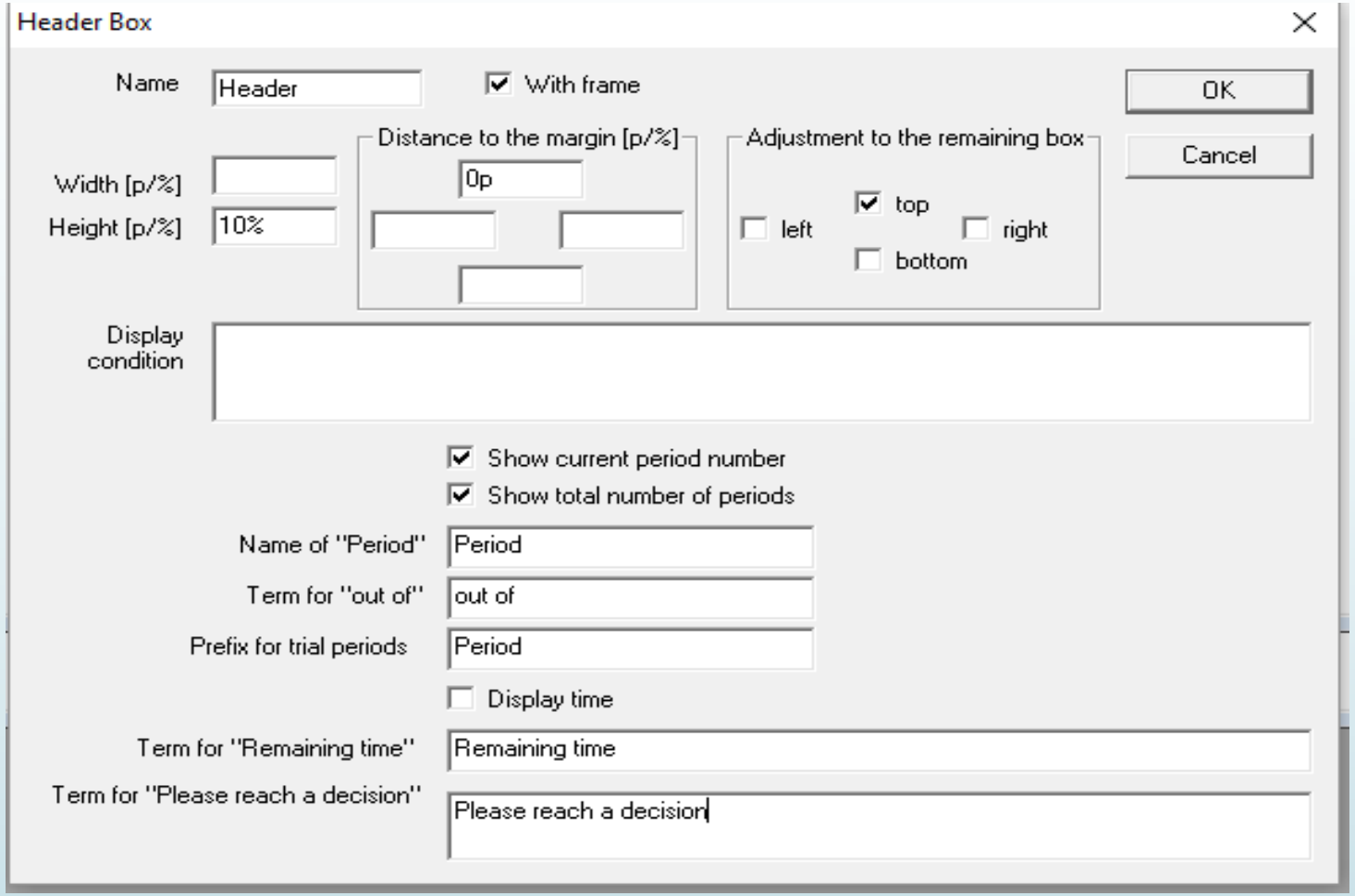


# Input and Output

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

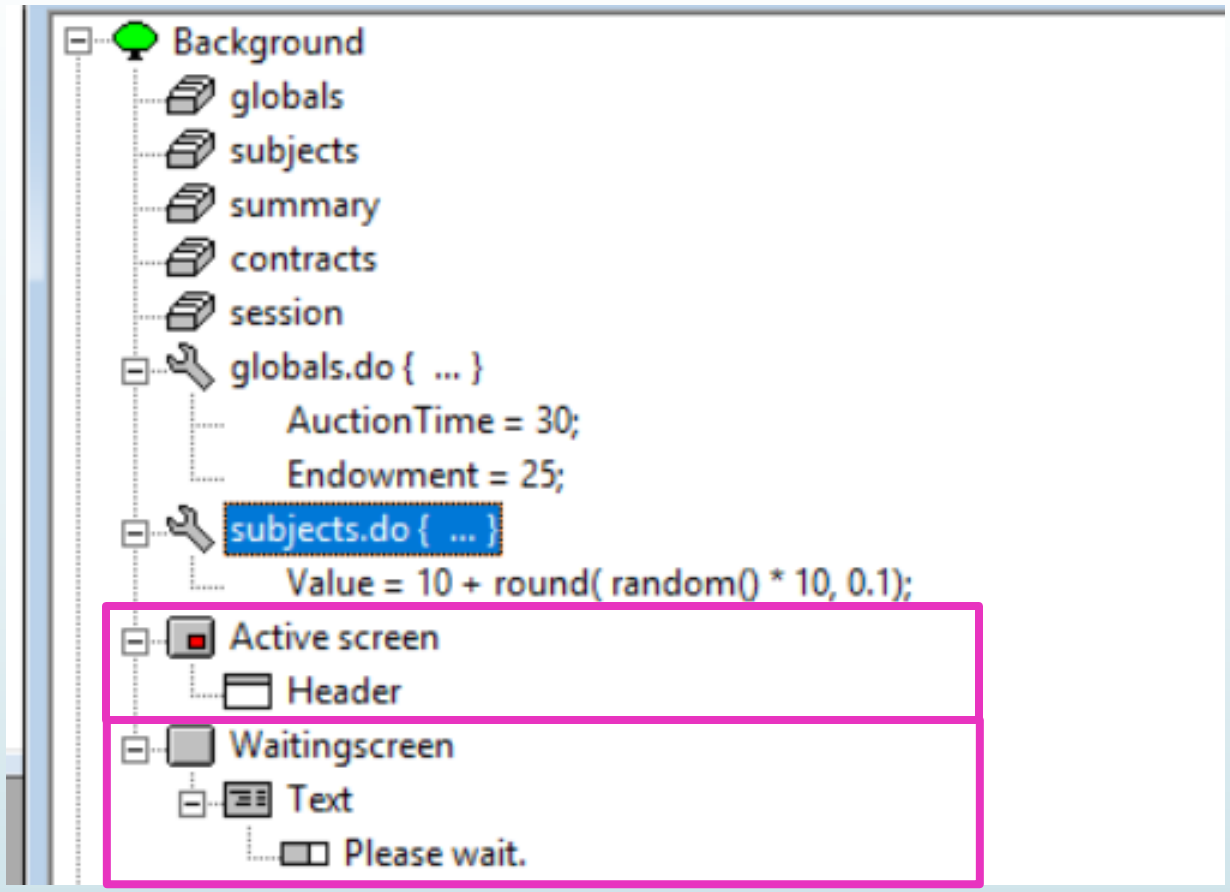
# Default element on Active Screen

- Add to Active Screen in Background



# Default `Waiting Screen`

- Default message on `Waiting screen` in `Background`



# Result stage

- Need to write a simple program to calculate the payment
- Use the predefined functions

```
z = random()*100;
if (z >= CE){
Profit = z;
}
else{
q = random();
Profit = if(q >= p, X, 0);
}
```

# Test your zTree program

- `OpenZleafs.exe` to run several Leaves at the same time
- Specify the number of zLeafs to be the number of subjects
- `Run -> Start Treatment`
- Once you start a treatment, you cannot edit the program

# Questionnaire

- Now you know
  - How to start a treatment: background, stages
  - Some basic elements in designing and programming a treatment: boxes, items, buttons
  - How to test a treatment: zLeaf
  - How to monitor subjects' progress in an experiment: tables
- How to end a treatment: Questionnaire
- Questionnaire ends your treatment with payoff-irrelevant questions
  - The questions are optional but the questionnaire is not
  - The (possibly empty) address form is mandatory

# Questionnaire

- `File -> New Questionnaire`
- `Questionnaire -> New Address Form`
  - If `First Name` and `Last Name` are omitted, the address form will not be displayed
- Add another question after the address form
  - Display Final Profit
  - Thank you/Goodbye screen
- Once the last subject finishes the address form, a payment file will be written
  - Print it and pay the subjects according to the amount

# Organizing your files

- If you run an experiment using zTree in the lab, your files are automatically organized in folders:
  - `paydir`: gathers the payment files
  - `programs`: save your programs
  - `expdata`: gathers the experimental data
  - `temp`: gather the temporary files

- You can do the same in your computer by changing the properties of zTree
  - Create a shortcut of zTree outside of programs
  - Change `Start in` to the parent folder
  - New folders: `paydir`, `priv`, `temp`, `expdata`
  - Add to `Target`: `/tempdir temp /gsfdir temp /privdir priv /paydir paydir /datadir expdata /language en`

# Experiment 2: Public Good Games

**Goal: understand basic features of tables, create groups**

- In each period subjects are assigned to groups of $n$
- Each subject starts with $y$ points.
- Subjects keep their points or invest in a public good
- Let $c\_i$ be the amount invested in public good by subject $i$
- The profit of each subject $i$:

$$\pi\_i = y - c\_i + (\alpha/n)\Sigma c\_j$$

where $j$'s are the members of the same group

- The game is played for $t$ periods

# Groups

- Set the number of groups in `background`
- Matching (in the tab `Treatment`)
  - Partner, stranger
  - Absolute stranger, Absolute typed stranger
- Matching can also be done as a program in `background`

```
if(Subject …){Group = 1;}

else …
```

- Check the `Parameter Table`
  - May change the group manually
- Variable `group` will be added into zTree tables

# zTree program for public good game

- Set $y$ and $\alpha$ in `background` as elements of the Subjects table
- Ask the subjects to contribute in the Contribution Stage
- All contribution is now stored in the subjects table

subjects table

| Period | Subject | Group | Profit | TotalProfit | Participate | Endowmer | Efficiency | Contribute | TimeSubm | SumContri | N | GroupProfi |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 | 1 | 20 | 2 | 0 | 0 | 0 | 1 | 0 |
| 1 | 2 | 1 | 0 | 0 | 1 | 20 | 2 | 0 | -10 | - | - | - |
| 1 | 3 | 1 | 0 | 0 | 1 | 20 | 2 | 3 | -2 | - | - | - |
| 1 | 4 | 2 | 0 | 0 | 1 | 20 | 2 | 8 | 5 | - | - | - |
| 1 | 5 | 2 | 0 | 0 | 1 | 20 | 2 | 6 | 14 | - | - | - |
| 1 | 6 | 2 | 0 | 0 | 1 | 20 | 2 | 5 | 21 | - | - | - |

- Endowment = $y$, Efficiency = $\alpha$, Contribute = $c$
- TimeSubmission: store time to collect input
- SumContribution, N, GroupProfit, and Profit are empty (for now)

# zTree program for public good game

- Use table functions to calculate profit
  - function(variables)
  - function(condition, variables)
  - sum, maximum, minimum, find, count, …

```
SumContribute = sum(same(Group), Contribute);

N = count(same(Group));

GroupProfit = Efficiency*SumContribute/N;

Profit = Endowment - Contribute + GroupProfit;
```

# Scope Operator

- The scope operator (:) allows you to get to the next “higher” level
- Summing the contribution

```
SumContribute = sum(Group == :Group, Contribute);
```

- Ranking the contribution

```
RankContribute = count(Contribute <= :Contribute);
```

# Scope Operator

- Matching using the scope operator
- Create *n* groups randomly

```
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 |

**Incorrect**

# Scope Operator

- Matching using the scope operator
- Remember that program is executed subject by subject

```
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 |

# Experiment 3: Ultimatum Game

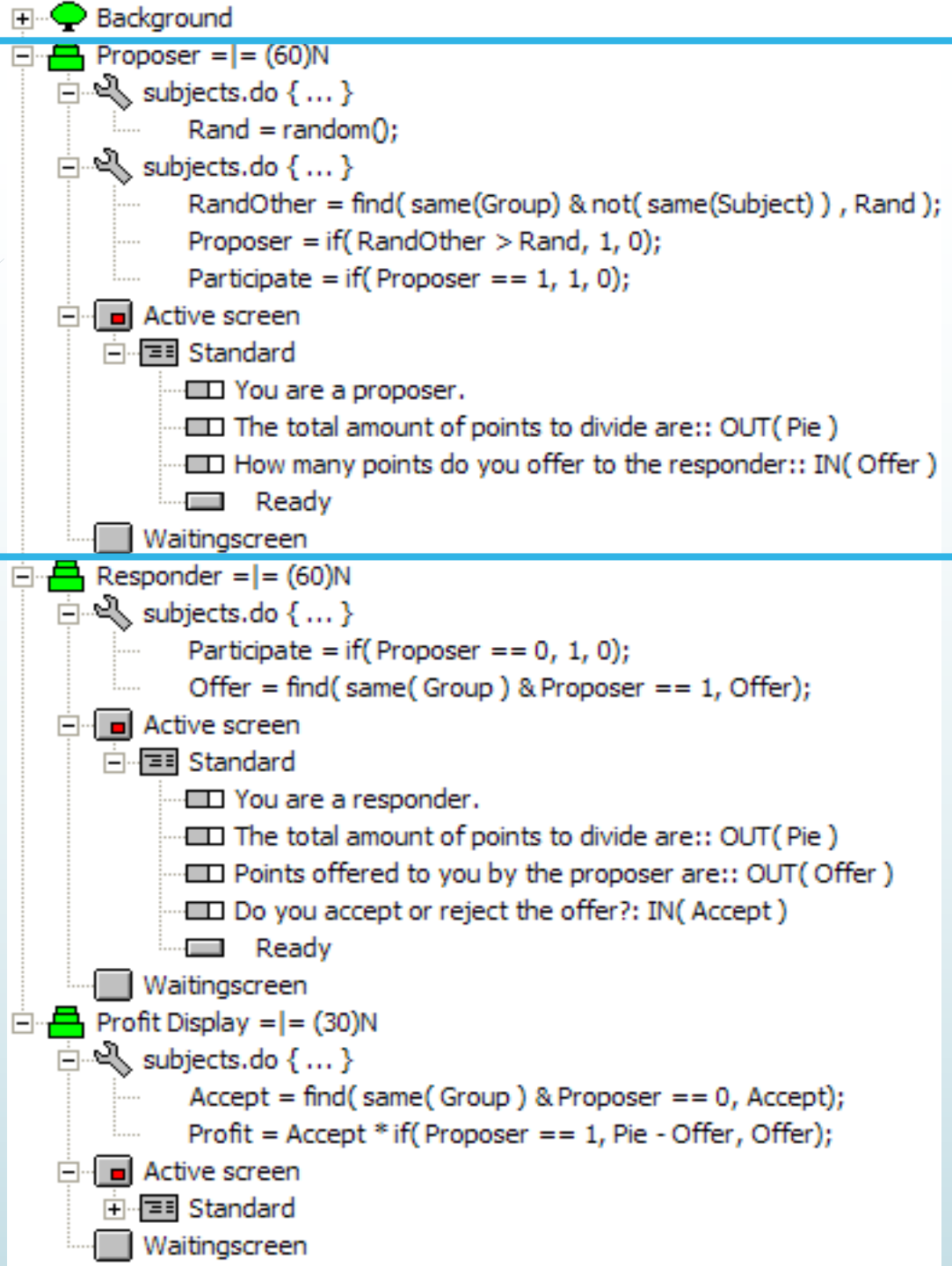
**Goal: Use "participate" to create asymmetric roles for the subjects**

Dynamically display some stages to some roles but not other

- Subjects are matched in pairs
- Each pair receives $y$ points
- Each pair has 1 proposer and 1 responder.
  - Proposers offer responders $x$ points, $x <= y$
  - If the responder rejects: both get 0 points.
  - If responder accepts the offer
    - Proposers earn: $\pi\_P = y-x$
    - Responders earn: $\pi\_R = x$
- Play for $t$ periods. Each period with a new pair
- Random matching and random assignment of roles

# Not everyone participates in a given stage

- Random matching to create pairs: check matching in tab `Treatment`
- Random assignment of roles
  - Use random number generator within a group
- Proposal stage vs. Acceptance stage
- In each stage, one is playing and one is waiting
- Use `Participate` variable in the `subjects table`

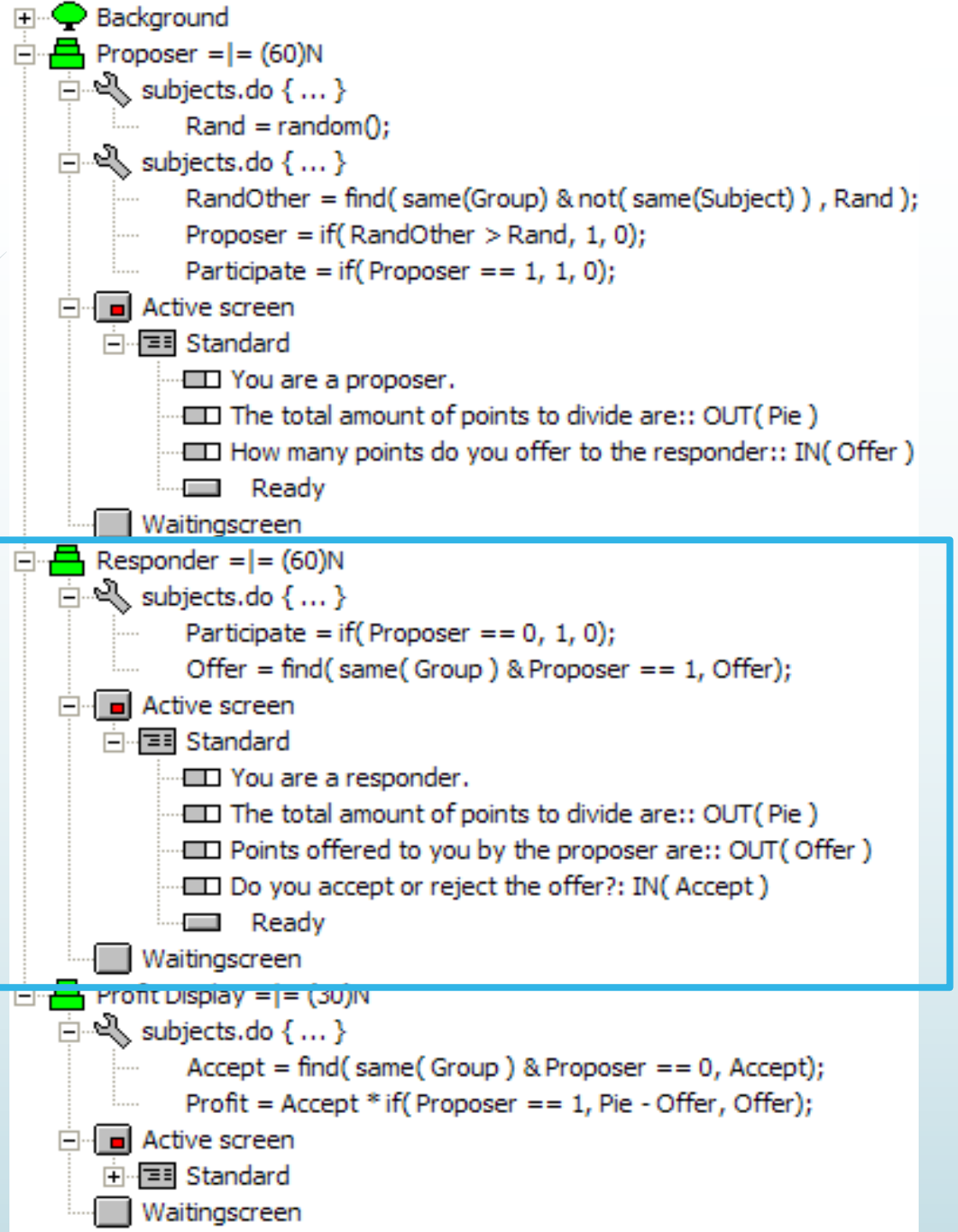



Rand for every player in the pair

Proposer dummy (0/1)

Participate iff Proposer

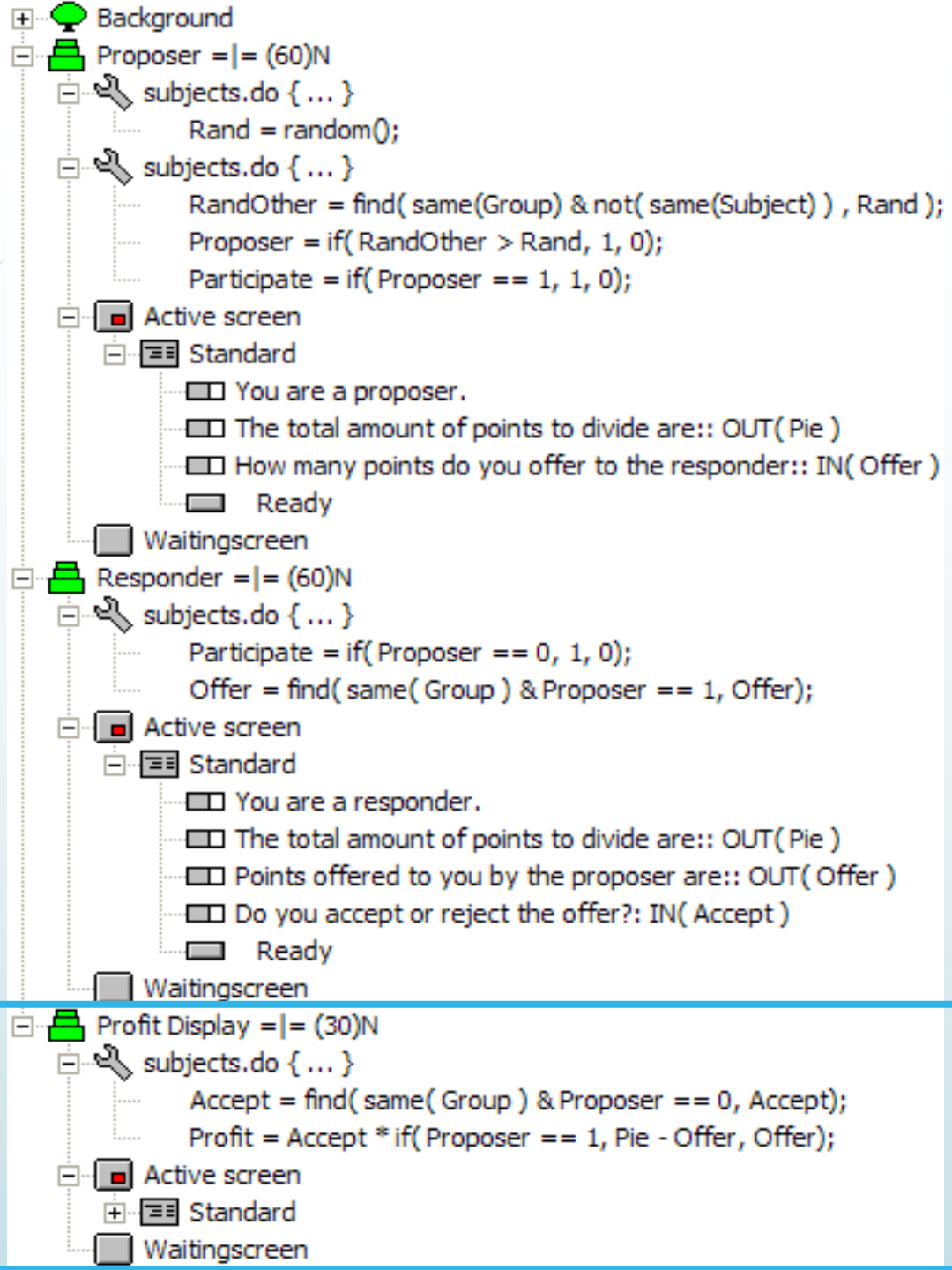
Input (Offer)




Participate iff NOT Proposer

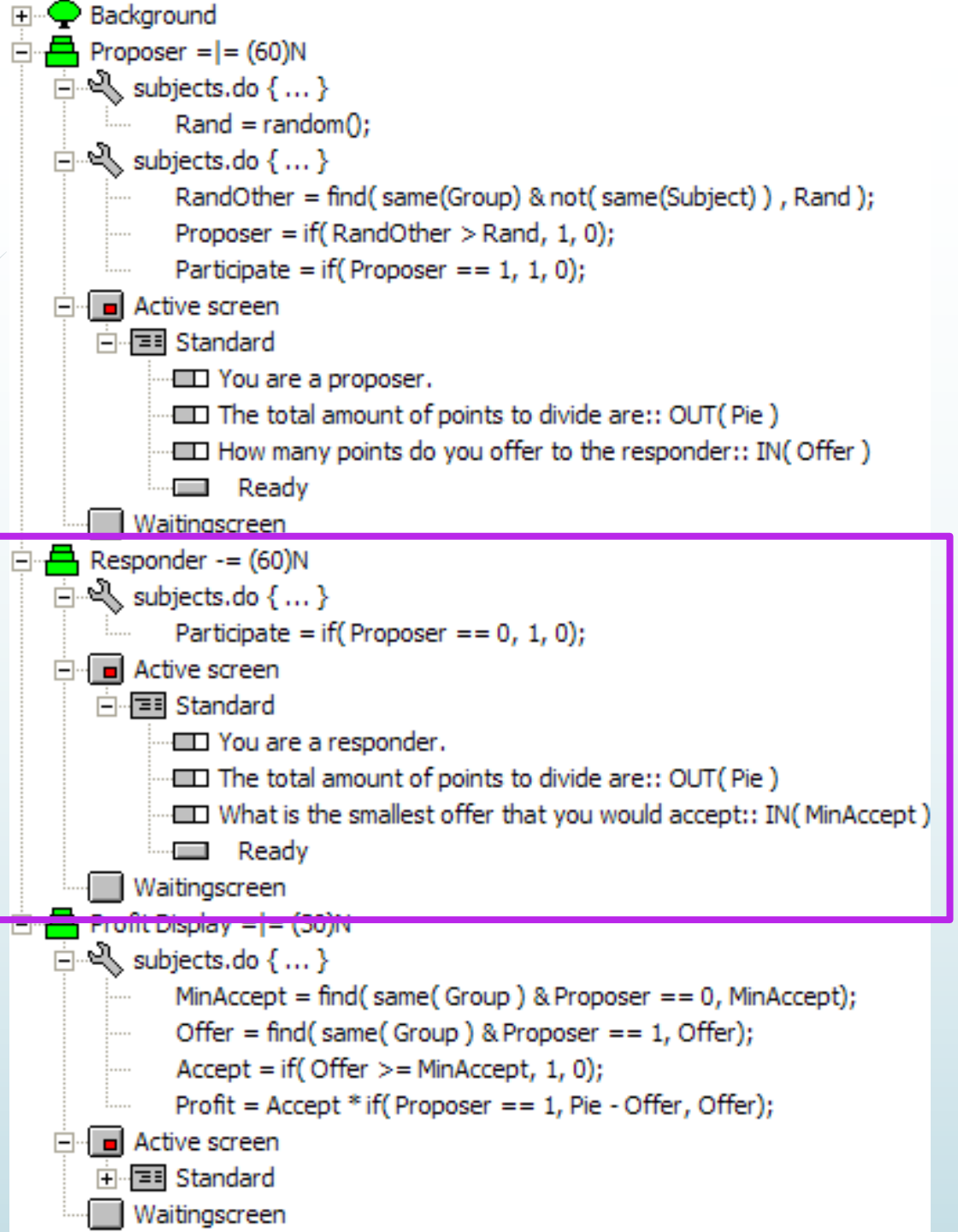
Read the Offer (from above)

Input (Accept/Reject)




Calculate Profit

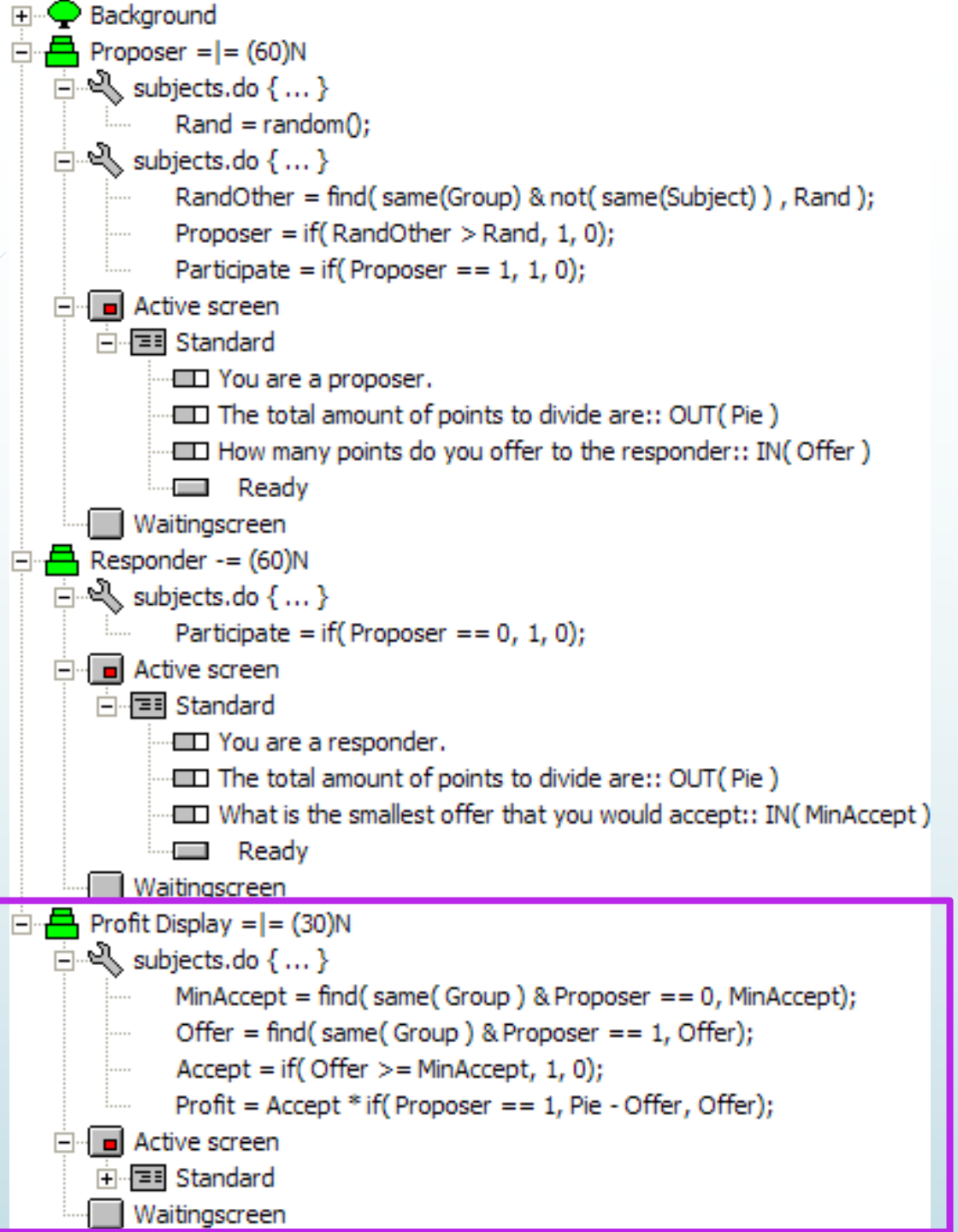
Display Profit




## Strategy Method

Participate iff NOT Proposer

Input (Minimum)




## Strategy Method

Calculate Profit

Display Profit

# Experiment 4: Simple Auction

- **Goal: Use the Contract table**

- Subjects are buyers
- Subjects get a (random) private value for an auctioned good $v_i$
- Subjects make public bids $b_i$
- Winner pays the **second** highest price
- The auction is terminated after a fixed timeout
- The winner gets: $\pi_B = y + v_i - b_2$

# Contract table

- We want to display the current highest bid
  - Subjects can only bid more than the current highest bid
- Example: 5 subjects with random private value

| Buyer | Bid | Order |
|---|---|---|
| 2 | 10 | Highest bid |

# Contract table

- We want to display the current highest bid
  - Subjects can only bid more than the current highest bid
- Example: 5 subjects with random private value

| Buyer | Bid | Order |
|---|---|---|
| 2 | 10 | 2nd-highest bid |
| 5 | 12 | Highest bid |

# Contract table

- We want to display the current highest bid
  - Subjects can only bid more than the current highest bid
- Example: 5 subjects with random private value

| Buyer | Bid | Order |
|---|---|---|
| 2 | 10 | |
| 5 | 12 | 2nd-highest bid |
| 1 | 15 | Highest bid |

# Contract table

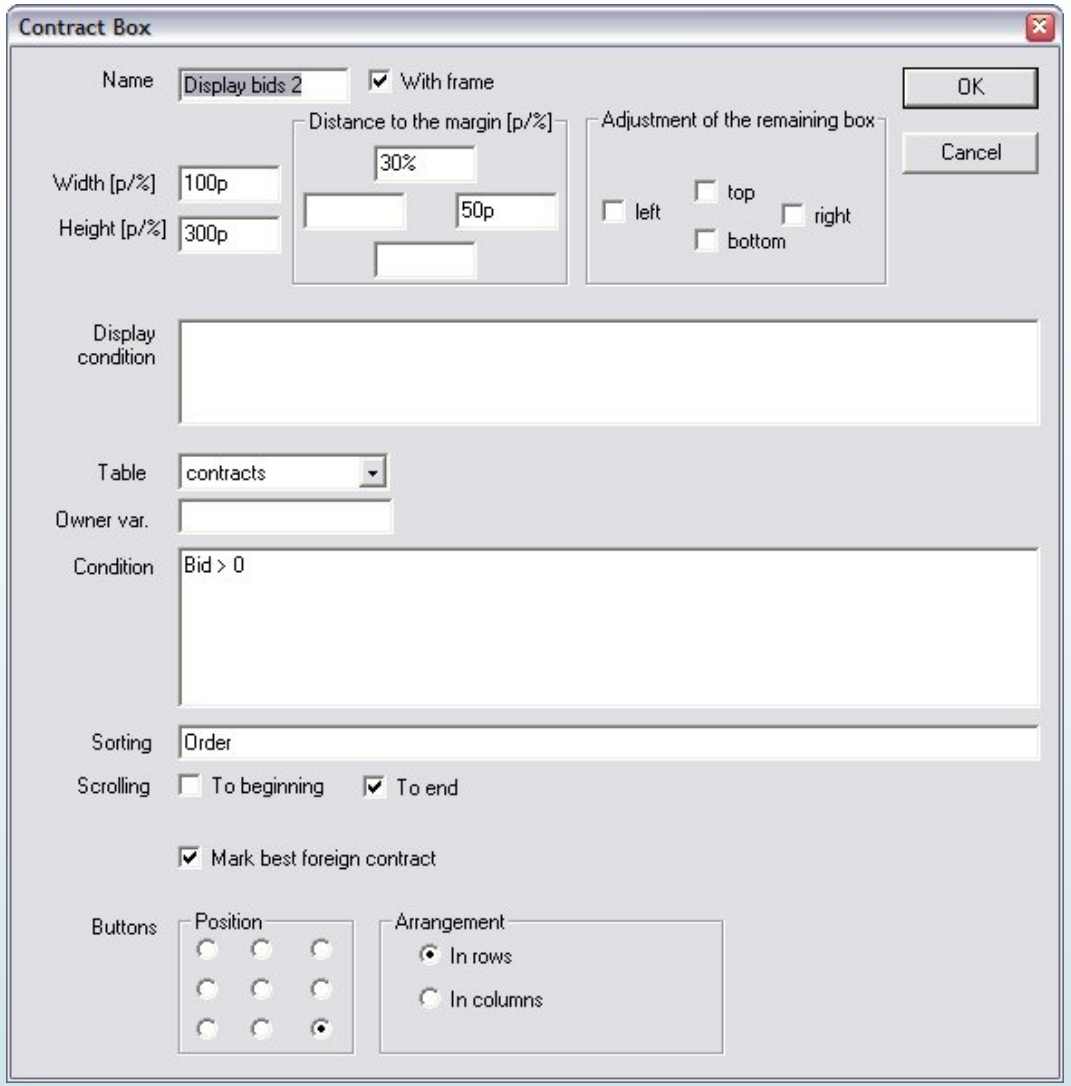
- Contract table: flexible number of records and manipulation
- Initiate contract table

```
globals.do{
contracts.new{
Buyer = 0;
Bid = 0;
Order = 1;
}
}
```

- Subjects enter a contract using `Contract Creation Box`
- Content of the contracts table can be displayed using `Contract Box`

# Contract table

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



# Plan your Experiment in zTree

# Plan an Experiment

- Instructions and Comprehension questions
- Multiple treatments
  - Welcome treatment (example to familiarize with the environment)
  - Same treatment with different baseline parameters (e.g. high vs low endowment)
  - Different treatments (e.g. risk attitude elicitation, then public good game)
  - Or you can have “between-subjects” designs (different subjects participate to different treatments)
- Payment
  - You can overwrite the variable TotalProfit (e.g. implement one random treatment)
- Questionnaire
  - You need it to create the payment file
  - You can use an “empty” questionnaire, but it can be helpful to collect demographic information, feedback, etc.

# Some formatting tips

- Add media to your program (images, animations)
- Communication among subjects (public or direct chat)
- Display text using RTF
  - Modify label

Your **profit** equals:

```
-><>{\rtf Your \b profit \b0 equals:}
```

  - Integrate variables into text

You *bid* for the item for $10

```
-> <>{\rtf You \i bid \i0 for the item for $<Bid|0.01>}
```

You **won**/did not win the auction

```
-> <>{\rtf You <if(Winner==Subject,1,0) | !text: 0="did not win "; 1="\b won \b0";> the auction!}
```

# Final remarks

- Free, easy to start testing and collecting data. Various examples online
- The manual is a great source and the online community is pretty active

- The coding experience can be difficult if you start with a non-standard design
- Laboratory experiments only, the interface changes based on screen resolution

- Draw the interface stage by stage (e.g. in PowerPoint) before starting
- Comment your code generously. Test the task frequently to spot errors
- Testing and troubleshooting are more helpful than learning the manual