A short Course for

zTree

Slides and Concept by Dennis Kubitza

briq Institute on Behavior & Inequality

3. Juli 2018

Contents

1. Installation
2. Introduction
3. Simple Single Player Experiments
4. Implementing Questionaires
5. Changing the Design
6. Programming
7. Programming II

Content Overview

1 Installation

2 Introduction

3 Simple Single Player Experiments

4 Implementing Questionaires

5 Changing the Design

6 Programming

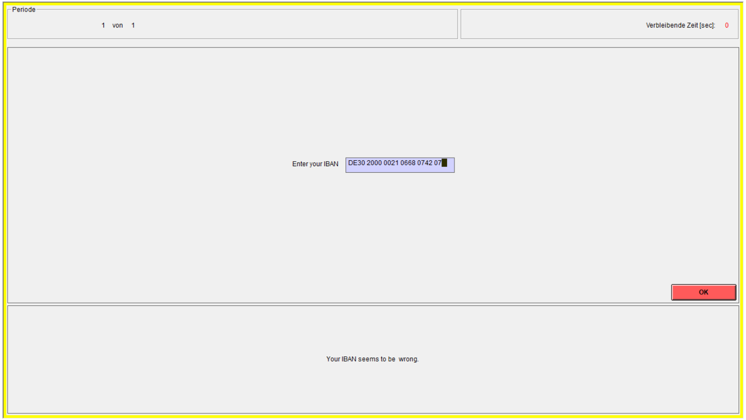
7 Programming II

Content Overview

7 Programming II

- Solution to Iban - Task
- Programming for Mulitplayer Games: Round-Based Games
- Programming for Multiplayer Games: Live Interaction

I uploaded a Solution to the Iban - Task on the Homepage.



Periode

1 von 1

Verbleibende Zeit [sec]: 0

Enter your IBAN DE30 2000 0021 0668 0742 07

Your IBAN seems to be wrong.

Content Overview

7 Programming II

- Solution to Iban - Task
- Programming for Mulitplayer Games: Round-Based Games
  - Preparations
  - Implementing Player Interaction
  - Heterogenous Games
- Programming for Multiplayer Games: Live Interaction

Target

- Todays first target is to implement a global goods game for 2 Groups with 2 Players.
- Secondly we want to implement simple real time interaction for 4 Players

Running multiple Clients

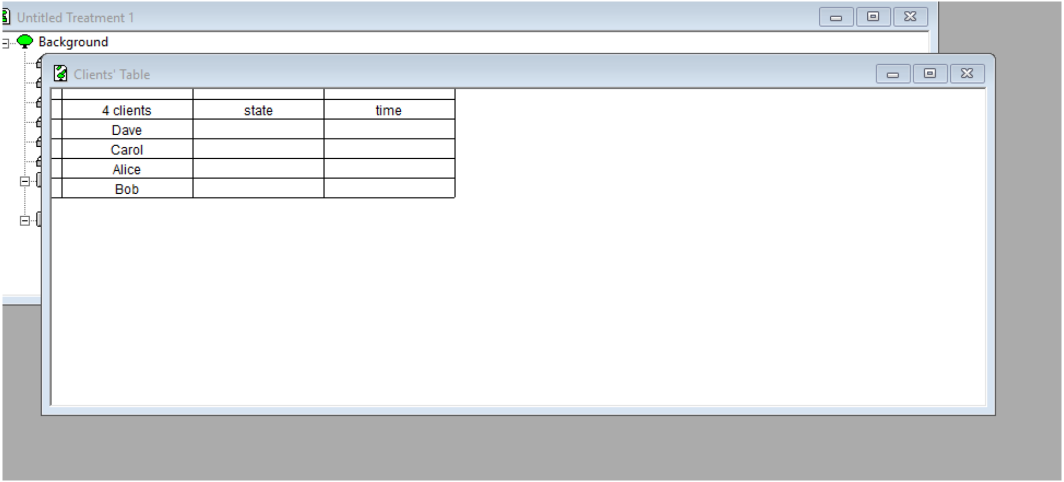
The first step for implementing multiplayer games is to actually start multiple zLeaves. Unfortunetly just opening zLeaf multiple times does not work.

- Right-Click on zLeaf: Create Shortcut
- Right-Click on the Shortcut
- Add to the target line

$$/name \; < Name >$$

When you now start zTree and Click on the Shortcut a Client with your defined names should appear (If you run the Clients table).

Running multiple Clients



4 clients	state	time
Dave		
Carol		
Alice		
Bob		

Preparing zTree

If you try to run a treatment now you get an Error Message. We still need to

1. Increase the number of Players
2. Define Grouping

Preparation I

If you double-click on Background in a Treatment you can alter:

1. The number of subjects
2. The number of groups

When starting the Treatment the given number of subjects will be participating but not automatically assigned to each group. You have to execute

$$Treatment > Matching > Stranger$$

This will assign the predefined Variable **Group** to each subject. Despite the predefined matching methods it is also possibible to implement user defined matchings.

Preparation II

zTree automatically enables Programms running on a certain Table access to variables on other Tables.

+ This makes coding easier
- Results in strange behaviour if variables are not unique.
- For this course we want to disable the so called autoscope.

Double click on Background > Disable auto scope

Table functions

Public goods Initialization

For implementing a Public good game we need:

- A stage where each participant is assigned its staring parameters
- A program calculating his final payments.

We know how to assign a fixed value to each subject if we assign it in the subjects table.

Accessing Variables

Now we want to specify a global endowment and afterwards assign it to the subjects. First of all we need access other tables. There are two ways for that.

1. We can access a table directly by using <Tablename>.<Variablename>.
2. We can access it with scope operators. : /

Scope operators

The standard tables in zTree have a hierarchical order:

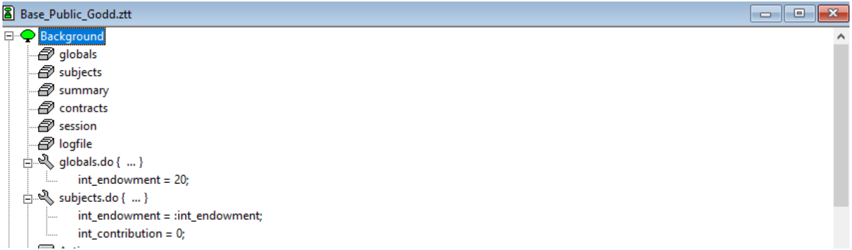
1. globals
2. subjects
3. summary (used for creating histories)
4. contracts (used for creating interactions)
5. etc.

If we write a programm for one of the tables, we can access a variable from the next highest table by adding

:

in front of the variable name. If we use / we always move to the globals table.

Example



Base_Public_Godd.ztt

```
Background
    globals
    subjects
    summary
    contracts
    session
    logfile
    globals.do { ... }
        int_endowment = 20;
    subjects.do { ... }
        int_endowment = :int_endowment;
        int_contribution = 0;
```

With this, we can implement the first requirement.

Table Functions

To implement the payment rule, we need access to all other payments in a group. zTree has predefined functions operating over a table:

sum(<VariableName>)
average(<VariableName>)
product(<VariableName>)
count( )
etc.

If we use this functions in a table like subjects, zTree will use the whole Table for calculating it.

$$number\_of\_participants = count()$$

Table Functions

To reduce the calculation on a subset of the table, we can give an additional condition to these functions

sum(<Condition>,<VariableName>)

average(<Condition>,<VariableName>)

product(<Condition>,<VariableName>)

count(<Condition>)

etc.

For condition we can use any Boolean Statement. For example [<VariableName> > 4].

More to conditions

This results in a problem: A program in the subjects table is run for each subject seperatly

- How do we restrict the computation for a particular Group ?

$$sum(Group == 2, int\_contribution)???$$

$$sum(Group == Group, int\_contribution)???$$

- First solution would require a case destinction for each Subject.
- Second solution does not work, because Group will be evaluated respective to the table row.

Table-Scope and Same

Inside tables we can use the scope operator : to refer to the current executers Variable or easier: same()

Solution

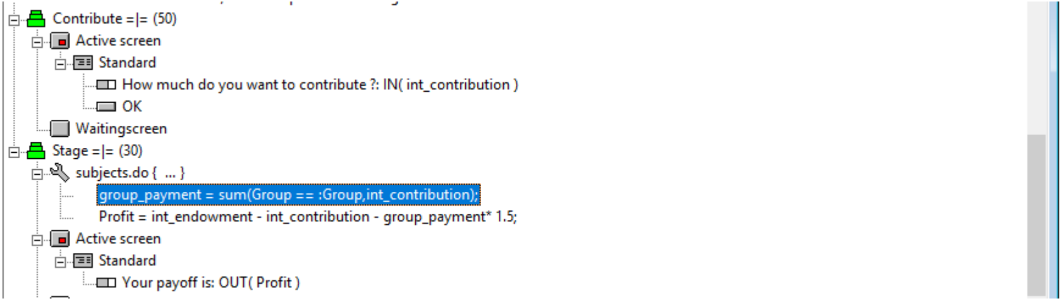
$$sum(Group == : Group, int\_contribution)$$

better Solution

$$sum(same(Group), int\_contribution)$$

With this knowledge we can now implement a Public goods game:

Example



```
Contribute =|= (50)
  Active screen
    Standard
      How much do you want to contribute ?: IN( int_contribution )
      OK
  Waitingscreen
Stage =|= (30)
  subjects.do {  ... }
    group_payment = sum(Group == :Group,int_contribution);
    Profit = int_endowment - int_contribution - group_payment* 1.5;
  Active screen
    Standard
      Your payoff is: OUT( Profit )
```

Important

Outside Table Functions “:” refers to a variable in the **next higher table**

Inside Table Functions “:” refers to a variable **belonging to the Function Executer**

Table Functions on other Tables

For now: Each table function is executed for the table the program is defined on.

- If we want to use table functions on other tables we need to
- $< TableName > . < TableFunction > (...)$

For example in a programm running on the globals table:

$$avg\_contribution = subjects.average(int\_contribution)$$

The usage of the matching algorithm only devides the player in Groups, but doesn't assign them a role.

- The easiest way for implementation is to code roles with numbers
- Then assign each player in each group his role.

But how ?

- The subject ID is the only unique variable that we have.
- IDEA: For each group we assign the roles by counting players with lower subject ID

Role assignment

We define a Program for the subjects table at the beginning:

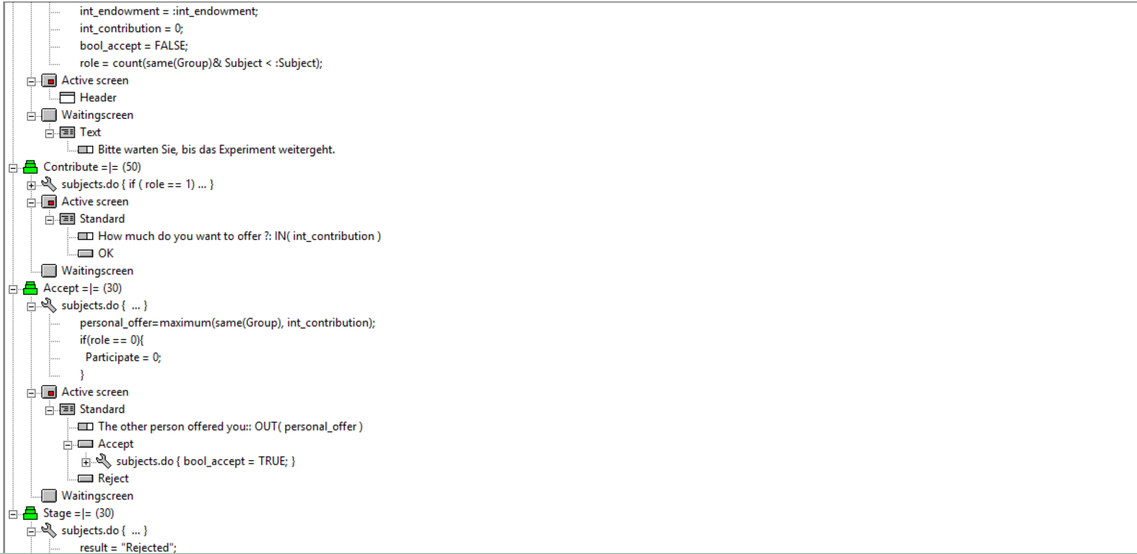
$$role = count(same(Group) \& Subject < : Subject)$$

To prohibit a participant from entering a stage we can use:

$$Participate = 0$$

in the Beginning of a stage

Example: Ultimatum Game



```
        int_endowment = :int_endowment;
        int_contribution = 0;
        bool_accept = FALSE;
        role = count(same(Group)& Subject < :Subject);
    Active screen
        Header
    Waitingscreen
        Text
            Bitte warten Sie, bis das Experiment weitergeht.
Contribute =|= (50)
    subjects.do { if ( role == 1) ... }
    Active screen
        Standard
            How much do you want to offer ?: IN( int_contribution )
            OK
    Waitingscreen
Accept =|= (30)
    subjects.do {  ... }
        personal_offer=maximum(same(Group), int_contribution);
        if(role == 0){
          Participate = 0;
        }
    Active screen
        Standard
            The other person offered you:: OUT( personal_offer )
            Accept
                subjects.do { bool_accept = TRUE; }
            Reject
    Waitingscreen
Stage =|= (30)
    subjects.do {  ... }
        result = "Rejected";
```

Content Overview

7 Programming II

- Solution to Iban - Task
- Programming for Mulitplayer Games: Round-Based Games
- Programming for Multiplayer Games: Live Interaction

Preparation

For this task, set the group Number to 1.

- We want to implement a simple experiment, where Subjects can tell a number (for example an offer) and update them in real time.
- Therefore: Create a Stage with 2 Boxes: 1 to the left and one 2 the right.
  1. For the Left Box:

     Treatment > NewBox > ContractCreationBox

  2. Choose for the right box:

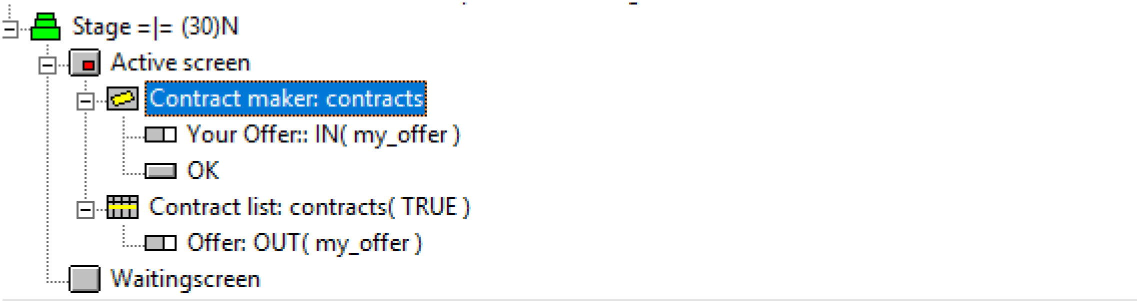
     Treatment > NewBox > ContractListBox

Contracts Table

The basis for Live interactions is the Contracts Table. While the Table is handled only internally, it can be displayed as a contract List box.

- To enable inputs into the contracts creation box, we need to add at least a New Input Item (with Input Variable) and a Button.
- To display this item we need to define an item with this variable in the Contracts list box

Example



- Stage =|= (30)N
  - Active screen
    - Contract maker: contracts
      - Your Offer:: IN(my_offer)
      - OK
    - Contract list: contracts(TRUE)
      - Offer: OUT(my_offer)
  - Waitingscreen

Non-Input Variables

Variables from other tables can be added in a program for the Contracts table after the button.

- As contracts is lower ranked than subjects we need to use the scope operator to access them.

$$offerer = :Subject$$

- To show these variables in the list box, you need to create new items.

Acessing the contracts table

And how can we access these variable from another table?

- We can access the values with table functions:
- In a programm for subjects:

$$best\_offer = contracts.minimum(subject ==: Subject, my\_offer)$$

Any Questions???