INTRODUCTION TO Z-TREE

Antonio Alonso Arechar
Yale University

SCHEDULE

- Interactive games
- **Sequential games**
- Advanced Topics
- Frontiers

SEQUENTIAL GAMES

1. Wrap up interactive games
2. Sequential moves
3. Example: Ultimatum game
4. Rich text format
5. (Indefinite) number of periods
6. Multiple players
7. Matching

WRAP UP INTERACTIVE GAMES

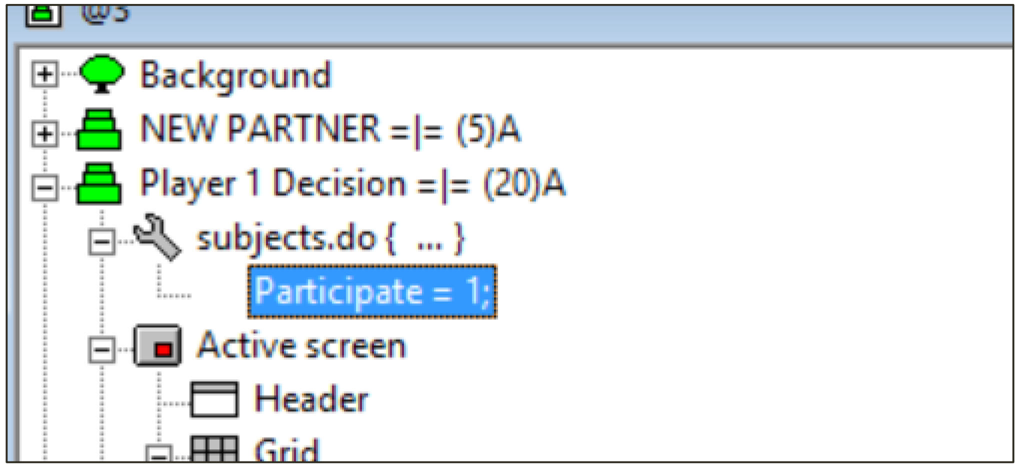
- A public goods game is an N-person version of the PD we just saw

WRAP UP INTERACTIVE GAMES

- **Three** players interacting for **3 periods**
- Their task is to choose how much of **$10** they want to contribute to a PGG
- Whatever amount contributed to the public good gets **doubled** and spilt evenly
- **Hint:** `SumC = sum( same( Group), Contribution);`

SEQUENTIAL MOVES

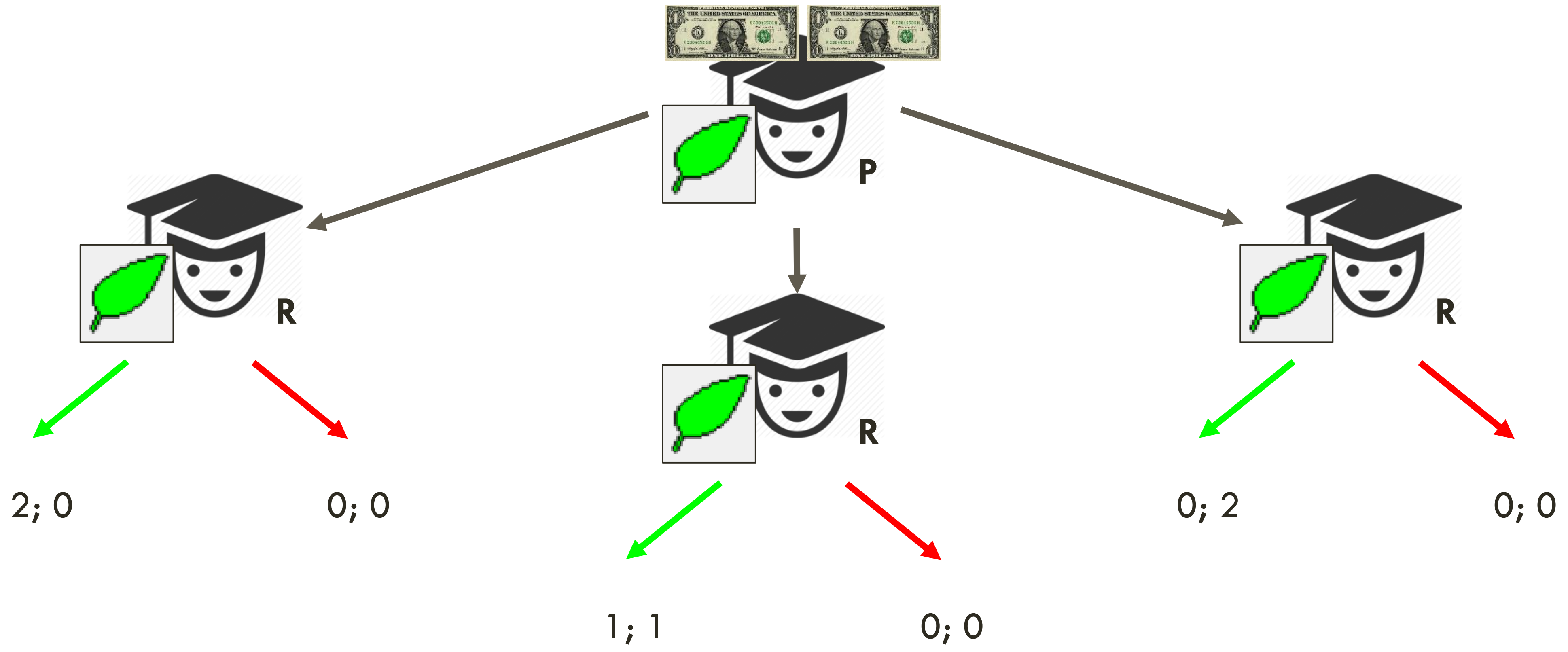
- In z-Tree, every treatment is defined as a linear sequence of stages
- It is possible that not all subjects are always in the same stage
- The variable `Participate` in the **subjects table** defines the access to a stage
- It's default is set to 1, so if you don't to anything everyone participates



```
Background
NEW PARTNER =|= (5)A
Player 1 Decision =|= (20)A
    subjects.do { ... }
        Participate = 1;
    Active screen
        Header
        Grid
```

EXAMPLE: ULTIMATUM GAME

- The Ultimatum game can be represented as follows:



EXAMPLE: ULTIMATUM GAME

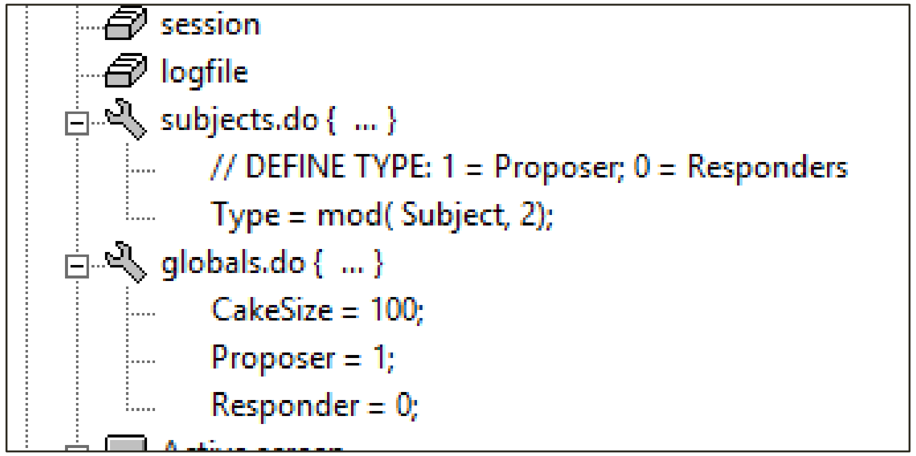
- Let’s program!
- Proposer moves first, making an offer decision
- After this, responder decides whether or not to accept the offer
- How can we determine who is a responder or a proposer?

`mod( x, y )`	Remainder after x is divided by y.
`Random( )`	Uniformly distributed rnd number between 0 and 1

EXAMPLE: ULTIMATUM GAME

- Create a program for the subjects table in Background:

```
// DEFINE TYPE: 1 = Proposer; 0 = Responders
Type = mod( Subject, 2);
```



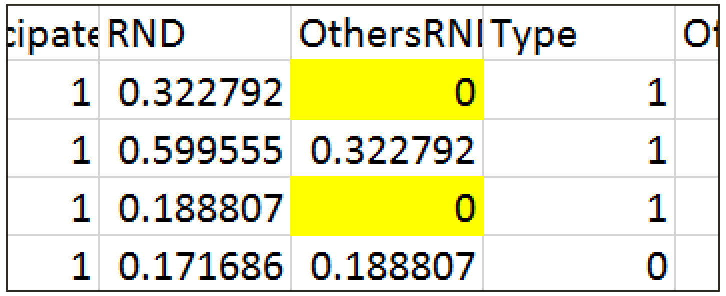
```
session
logfile
subjects.do { ... }
    // DEFINE TYPE: 1 = Proposer; 0 = Responders
    Type = mod( Subject, 2);
globals.do { ... }
    CakeSize = 100;
    Proposer = 1;
    Responder = 0;
```

EXAMPLE: ULTIMATUM GAME

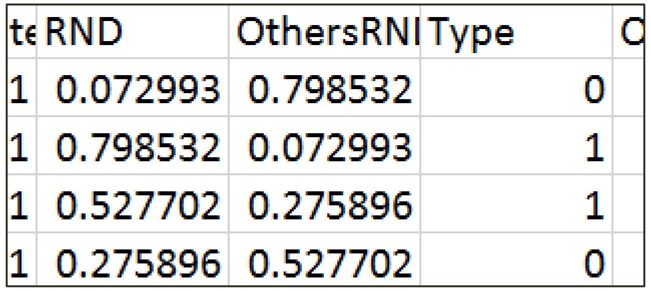
- `mod` is not random though
- If you'd like to randomly allocate subjects to types you'll need **two** programs:

```
RND = random();
```

```
OthersRND = find ( same(Group) & not (same( Subject)),RND);
Type = if ( RND > OthersRND, 1, 0);
```



cipate	RND	OthersRNI	Type	Of
1	0.322792	0	1	
1	0.599555	0.322792	1	
1	0.188807	0	1	
1	0.171686	0.188807	0	

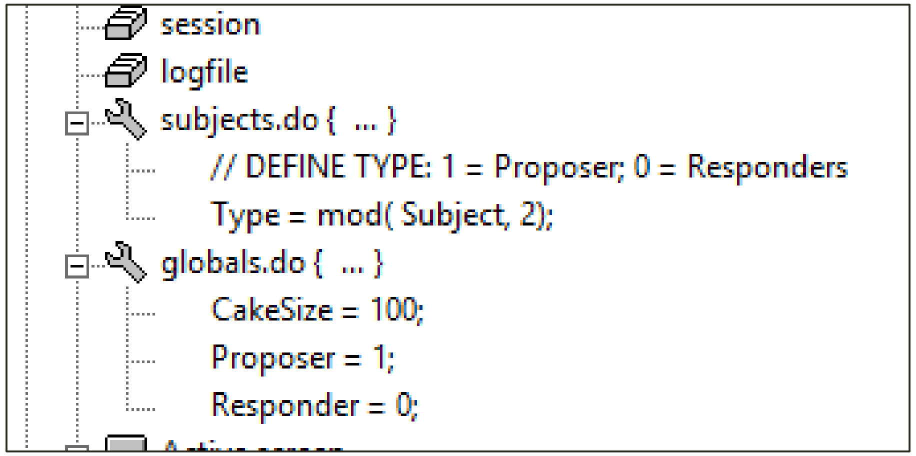


te	RND	OthersRNI	Type	C
1	0.072993	0.798532	0	
1	0.798532	0.072993	1	
1	0.527702	0.275896	1	
1	0.275896	0.527702	0	

EXAMPLE: ULTIMATUM GAME

- Once types are defined we set up global variables:

```
CakeSize = 100;
Proposer = 1;
Responder = 0;
```

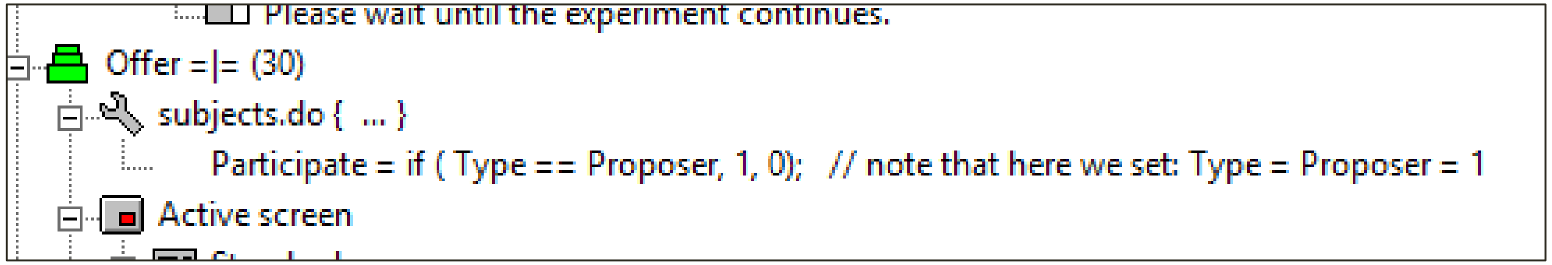


```
session
logfile
subjects.do { ... }
    // DEFINE TYPE: 1 = Proposer; 0 = Responders
    Type = mod( Subject, 2);
globals.do { ... }
    CakeSize = 100;
    Proposer = 1;
    Responder = 0;
```

EXAMPLE: ULTIMATUM GAME

- ***Sequential move:*** We need a stage for **P** and another one for **R**
- How?

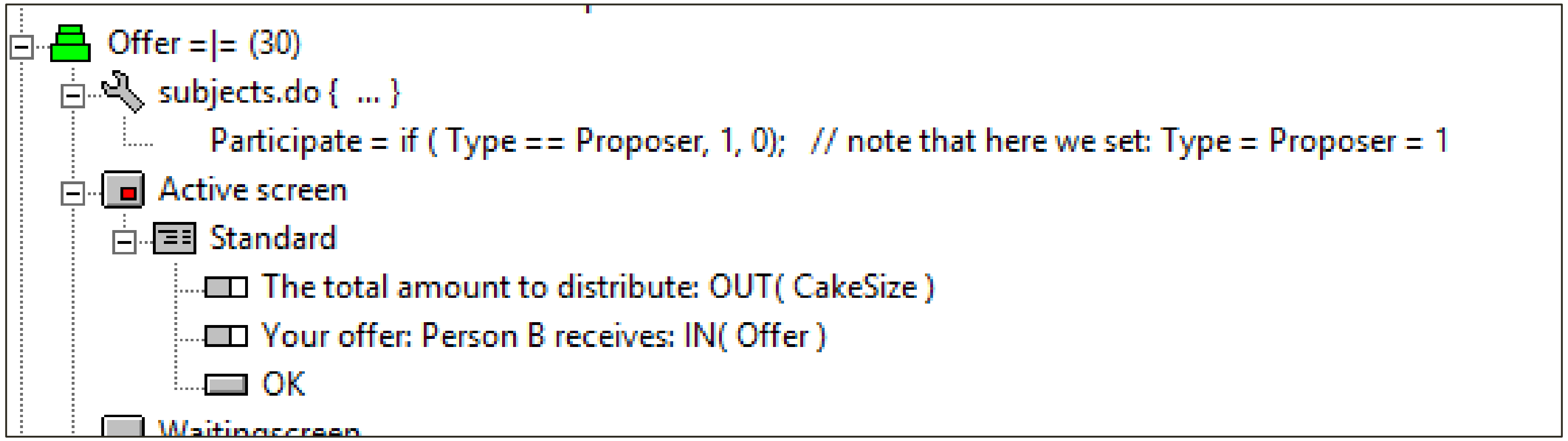
```
Participate = if ( Type == 1, 1, 0);  // Proposer
```



```
Please wait until the experiment continues.
Offer =|= (30)
  subjects.do { ... }
    Participate = if ( Type == Proposer, 1, 0);   // note that here we set: Type = Proposer = 1
  Active screen
```

EXAMPLE: ULTIMATUM GAME

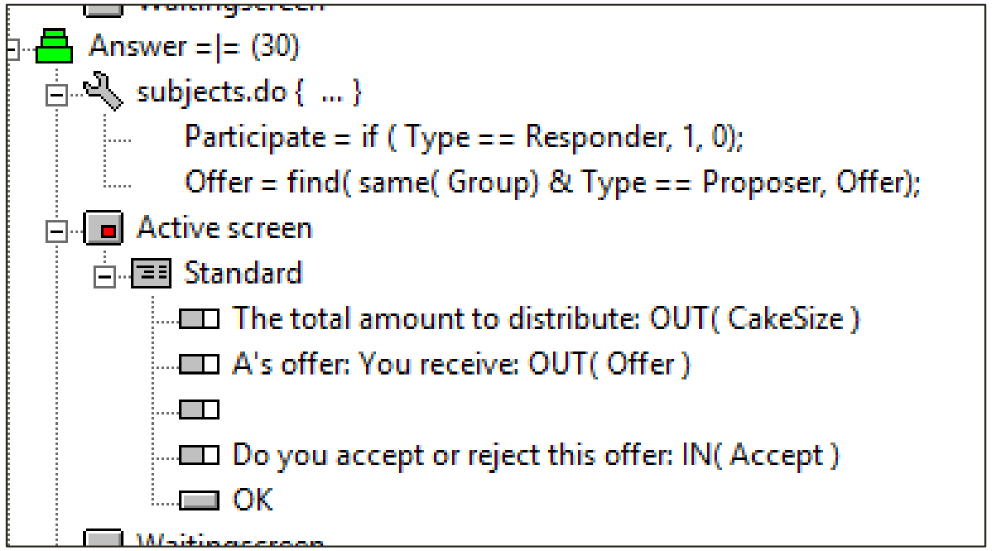
- Now that only Proposers can participate in this stage we can focus in its items
- We elicit Proposer's offer
- We can also remind them about the amount of money they have to divide



```
Offer =|= (30)
  subjects.do { ... }
      Participate = if ( Type == Proposer, 1, 0);   // note that here we set: Type = Proposer = 1
  Active screen
    Standard
      The total amount to distribute: OUT( CakeSize )
      Your offer: Person B receives: IN( Offer )
      OK
  Waitingscreen
```

EXAMPLE: ULTIMATUM GAME

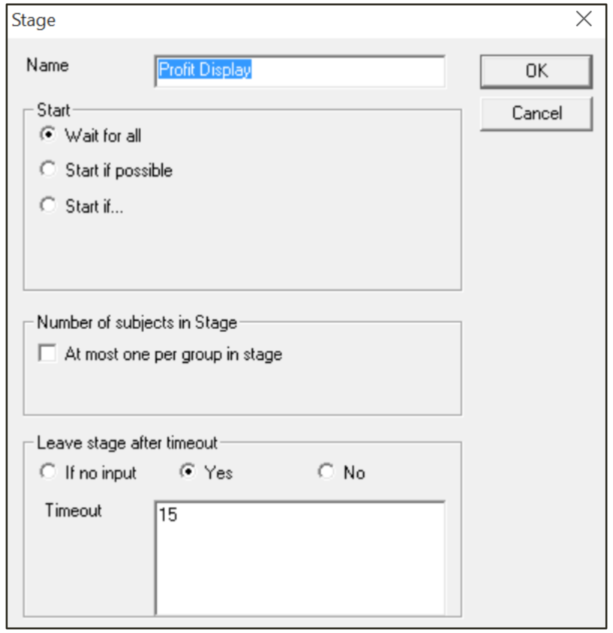
- For **R** we need to set up a new stage and a program to exclude **P**
- And find the offer made by **P**
- And finally elicit a decision on **R**'s offer



```
Answer =|= (30)
  subjects.do {  ... }
      Participate = if ( Type == Responder, 1, 0);
      Offer = find( same( Group) & Type == Proposer, Offer);
  Active screen
    Standard
      The total amount to distribute: OUT( CakeSize )
      A's offer: You receive: OUT( Offer )

      Do you accept or reject this offer: IN( Accept )
      OK
```

EXAMPLE: ULTIMATUM GAME

- Let’s create a *Profit Display* stage
- Note that stages that only provide information can be left after timeout
- Session’s pace can be improved if timeout window is set to 15 seconds or less


Stage

Name: Profit Display

Start
- [✓ Wait for all]
- Start if possible
- Start if...

Number of subjects in Stage
- [ ] At most one per group in stage

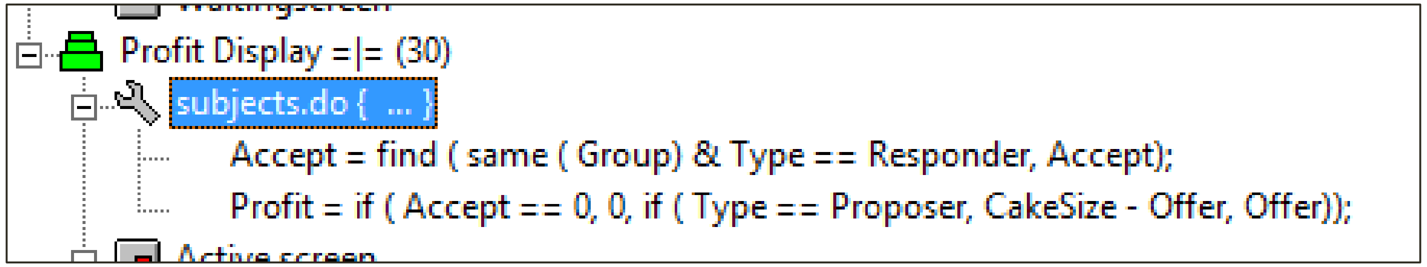
Leave stage after timeout
- If no input [✓ Yes] No

Timeout: 15

EXAMPLE: ULTIMATUM GAME

- Now we need to calculate profits in a new stage:

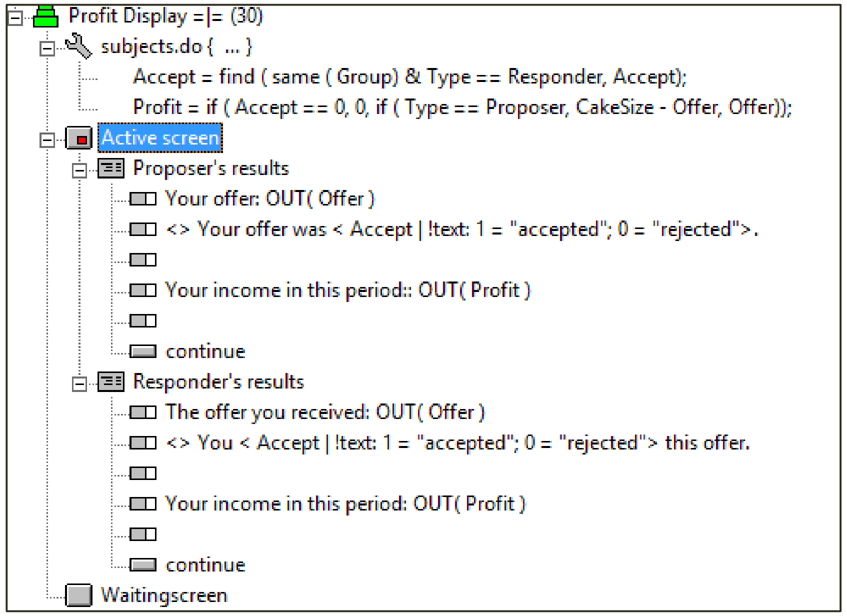
```
Accept = find ( same (Group) & Type==Responder, Accept);
Profit=if(Accept==0,0,if(Type==1,CakeSize-Offer,Offer));
```



```
Profit Display =|= (30)
  subjects.do { ... }
    Accept = find ( same ( Group) & Type == Responder, Accept);
    Profit = if ( Accept == 0, 0, if ( Type == Proposer, CakeSize - Offer, Offer));
Active screen
```

EXAMPLE: ULTIMATUM GAME

- **P** and **R** should get different feedback, so maybe we need two stages?
- Not necessarily, we could edit the box's Display condition
- Note also that if you want to add a variable in the text you can do so with <>



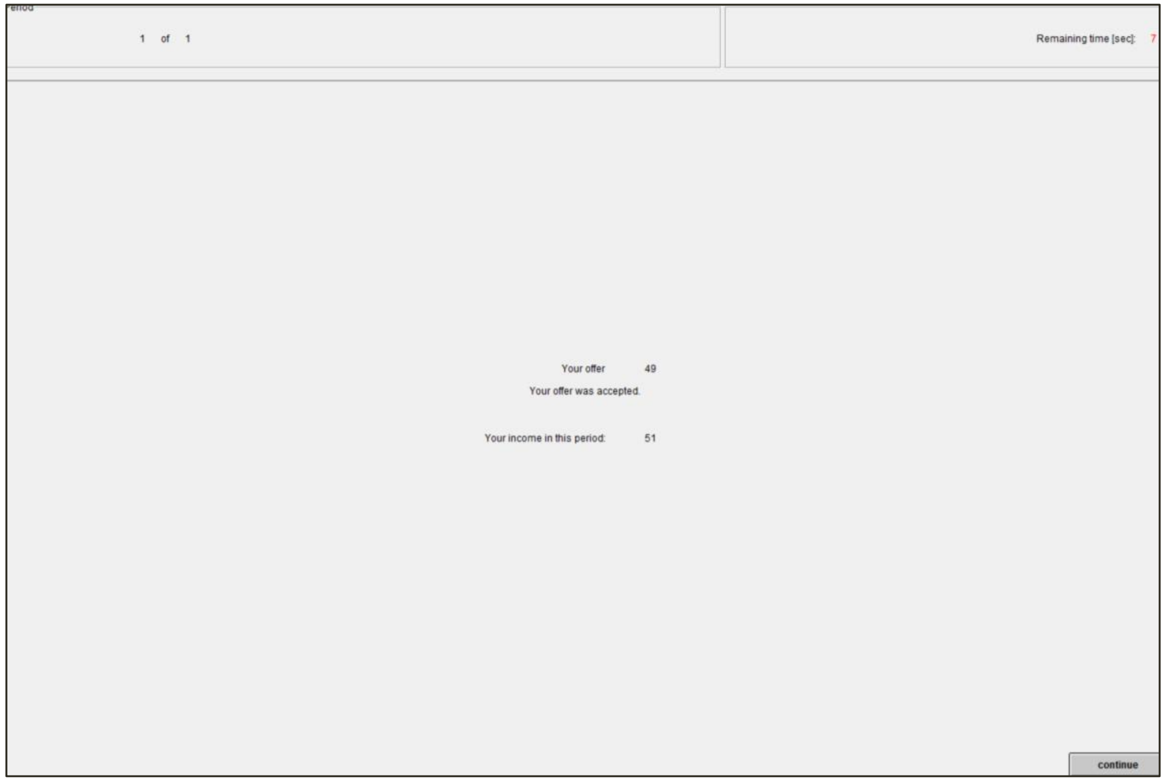
```
Profit Display =|= (30)
  subjects.do { ... }
    Accept = find ( same ( Group) & Type == Responder, Accept);
    Profit = if ( Accept == 0, 0, if ( Type == Proposer, CakeSize - Offer, Offer));
  Active screen
    Proposer's results
      Your offer: OUT( Offer )
      <> Your offer was < Accept | !text: 1 = "accepted"; 0 = "rejected">.

      Your income in this period:: OUT( Profit )

      continue
    Responder's results
      The offer you received: OUT( Offer )
      <> You < Accept | !text: 1 = "accepted"; 0 = "rejected"> this offer.

      Your income in this period: OUT( Profit )

      continue
  Waitingscreen
```

EXAMPLE: ULTIMATUM GAME

- Looks like our treatment is ready, is it?
- We could borrow yesterday’s questionnaire
- Let’s test it!

RICH TEXT FORMAT

- This is how our results screen looks like:



1 of 1

Remaining time [sec]: 7

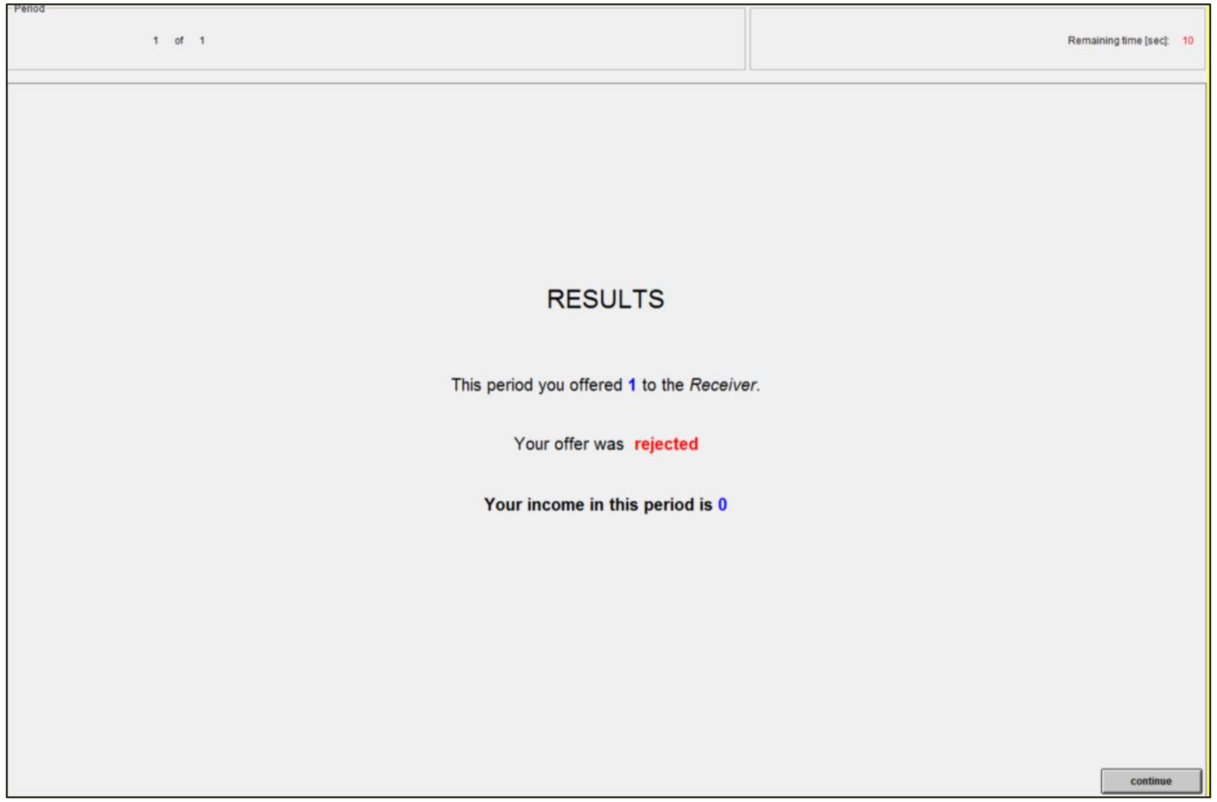
Your offer 49

Your offer was accepted.

Your income in this period: 51

RICH TEXT FORMAT

- If we use rich text format (rtf) this is how it could look like:



RESULTS

This period you offered **1** to the *Receiver*.

Your offer was **rejected**

Your income in this period is 0

RICH TEXT FORMAT

- RTF format begins with **“{\rtf”** (with a blank space at the end) and ends with **“}”**
- In between is the text you wish to be formatted
- In options of `!text` layouts, the relevant **rtf** has to appear in <u>every</u> option

```
Your offer was \b <  Accept | !text: 1 = "{ \cf3 accepted}"; 0 = "{ \cf2 rejected}">
```

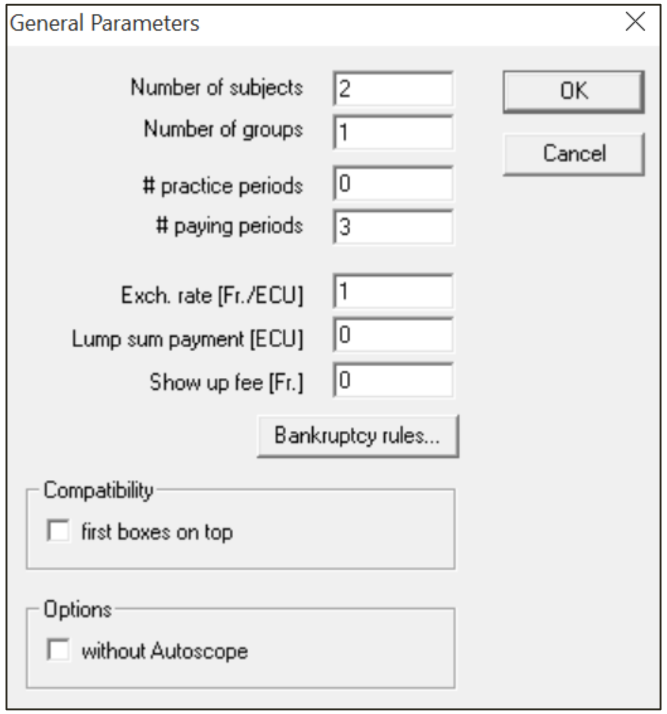
RICH TEXT FORMAT

- Frequently used formatting instructions

```
\tab          \par          \line         \bullet       \ql  (qr)  (qc)

\b  (\b0)     \i  (\i)      \ul  (ul0)    \sub          \super


\colorbl      eg: {\colortbl; \red0\green0\blue255; \red255\green0\blue0;}

\cfn          eg: \cf1  \cf2

 \fsn         eg: \fs30 Hola! \fs50 Hola!
```

(INDEFINITE) NUMBER OF PERIODS

- So far we have played one-shot games, but it’s also interesting to have repetitions
- We can set the number of **periods** in the **General Parameters** window


General Parameters

Number of subjects	2
Number of groups	1
# practice periods	0
# paying periods	3
Exch. rate [Fr./ECU]	1
Lump sum payment [ECU]	0
Show up fee [Fr.]	0

Compatibility
☐ first boxes on top

Options
☐ without Autoscope

(INDEFINITE) NUMBER OF PERIODS

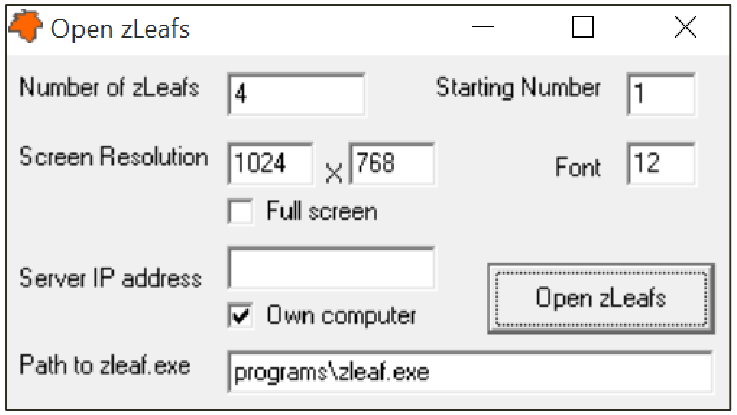
- What if we want participants to play for an indefinite number of periods?
- Recall **globals** table holds default variables:
- `Period, NumPeriods, RepeatTreatment`
- We could create a new program in Background!

```
p_continue = 0.50;
random_continue = random();
RepeatTreatment = if( random_continue <= p_continue, 1, 0);
```

MULTIPLE PLAYERS

- So far we've tested our treatments with 2-3 players and only 1 group
- To launch multiple z-Leaves you can download Ernesto Reuben's zip file:

  http://www.ereuben.net/teach/zTreeTestEnvironment.zip
- Copy **ztree.exe** and **zleaf.exe** in the *programs* subfolder
- Open z-Tree with **openztree.bat**
- And then open **zleafs.bat** ☺

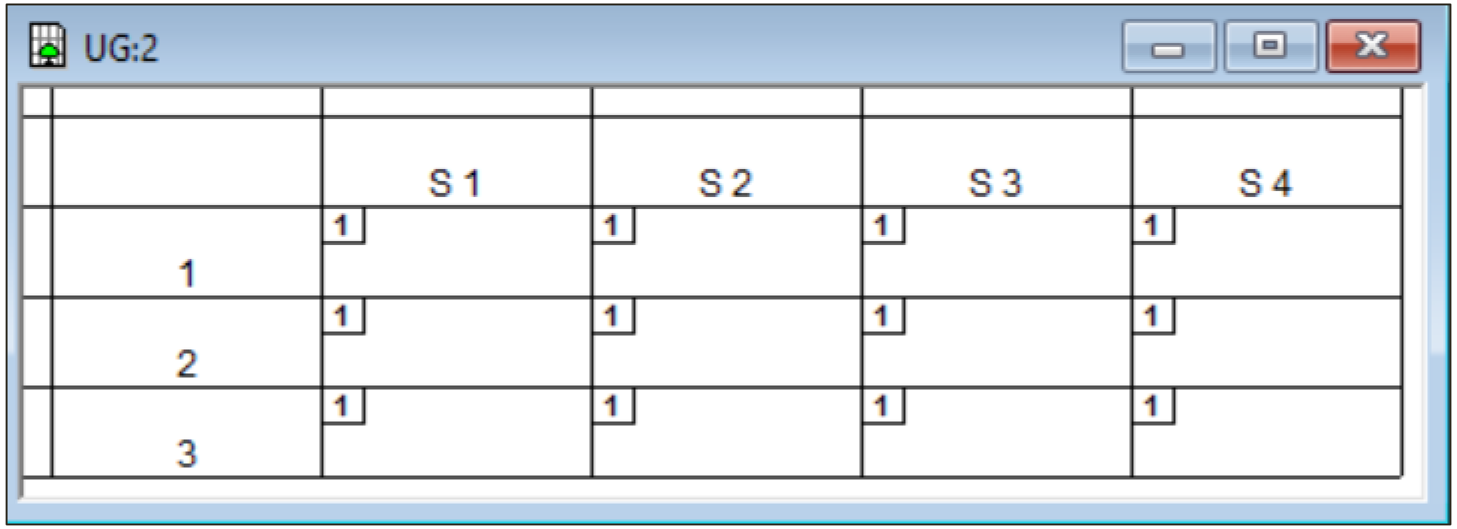


Open zLeafs

Number of zLeafs: 4
Starting Number: 1
Screen Resolution: 1024 x 768
Font: 12
☐ Full screen
Server IP address:
☑ Own computer
Path to zleaf.exe: programs\zleaf.exe

MATCHING

- When more than 1 group interacts z-Tree needs to know how you intend to match them
- Set number of subjects to 4 in the General Parameters window
- Open **Treatment → Parameter Table**
- z-Tree does not know yet how to match groups (even if we set Number of groups=2)

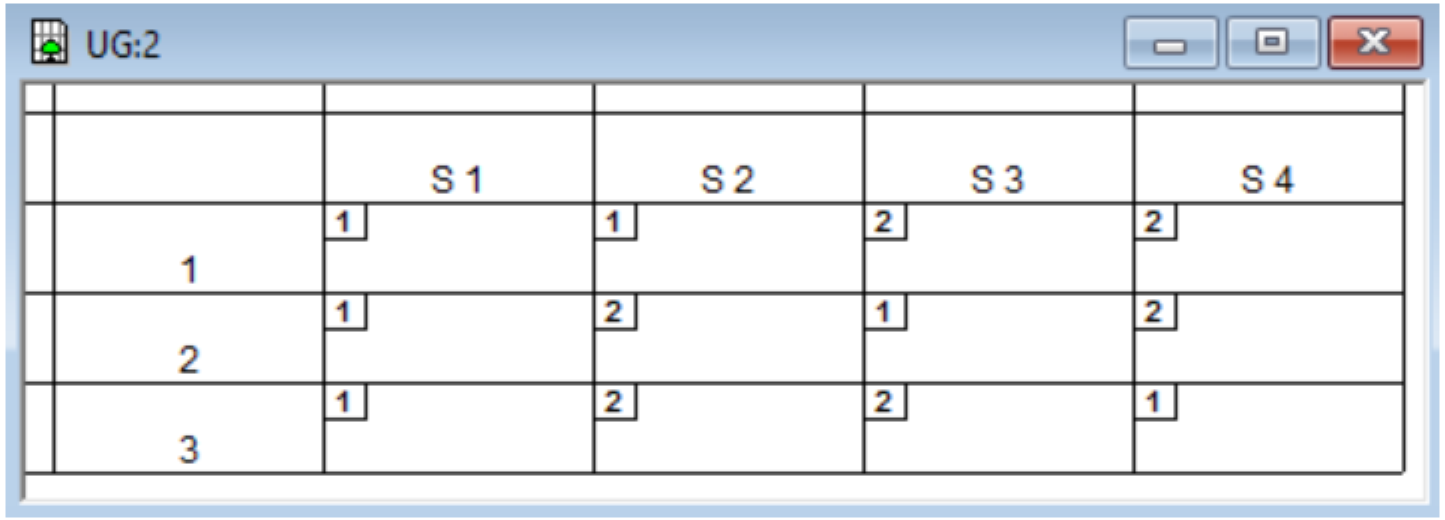


UG:2

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

MATCHING

- We can define *standard* matches in **Treatment → Matching** (reapplied if number of subjects or periods change)
- You *could* also double-click in each cell and edit manually
- You can also import .txt *matching* files through **Treatment → Import Variable Table**

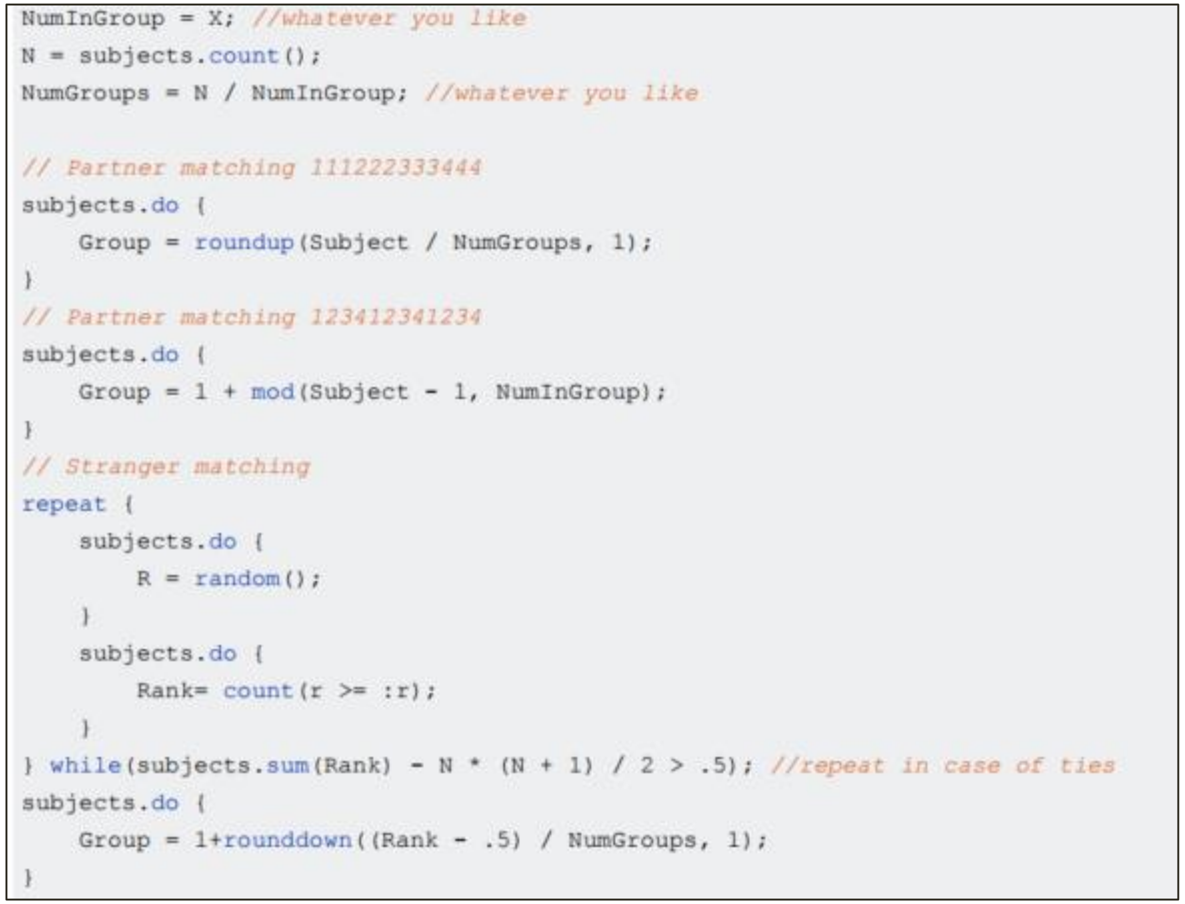

UG:2

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

MATCHING

- You can program group matching
- First set the number of groups in

  General Parameters to 1



```
NumInGroup = X; //whatever you like
N = subjects.count();
NumGroups = N / NumInGroup; //whatever you like

// Partner matching 111222333444
subjects.do {
    Group = roundup(Subject / NumGroups, 1);
}
// Partner matching 123412341234
subjects.do {
    Group = 1 + mod(Subject - 1, NumInGroup);
}
// Stranger matching
repeat {
    subjects.do {
        R = random();
    }
    subjects.do {
        Rank= count(r >= :r);
    }
} while(subjects.sum(Rank) - N * (N + 1) / 2 > .5); //repeat in case of ties
subjects.do {
    Group = 1+rounddown((Rank - .5) / NumGroups, 1);
}
```