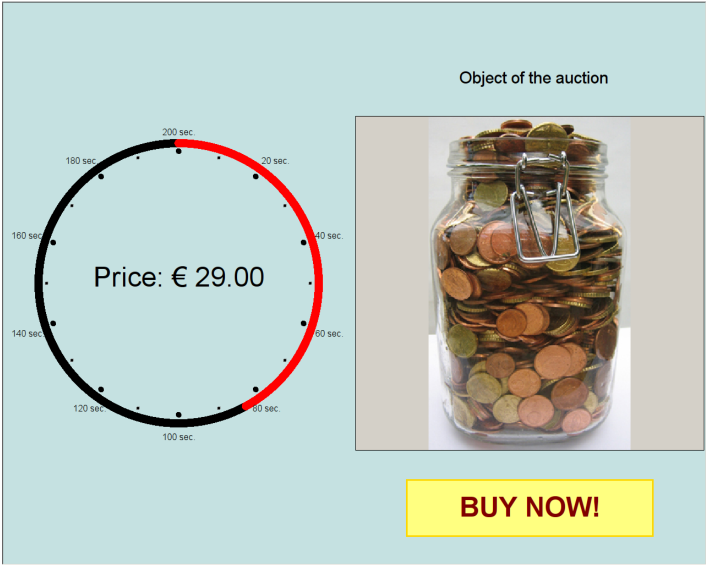
Part VIII

Advanced programming examples

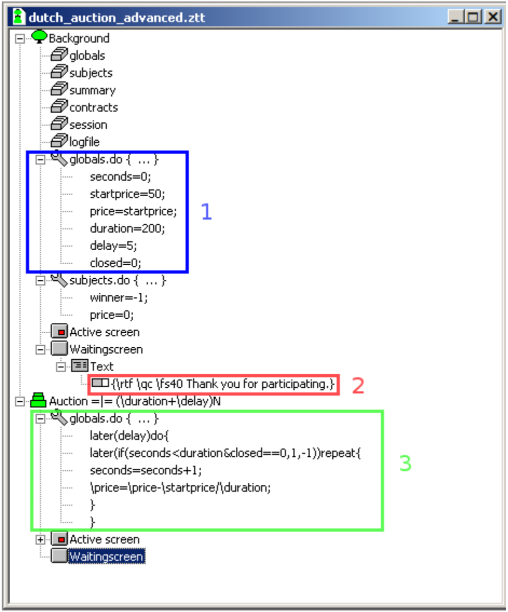
Advanced programming examples

Dutch auction - screenshot

Example: dutch_auction_advanced.ztt



Dutch auction - the background

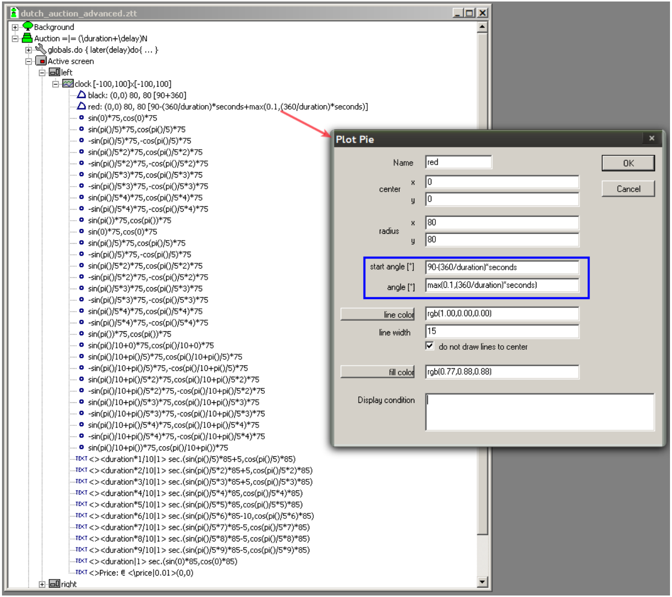


```
dutch_auction_advanced.ztt
Background
    globals
    subjects
    summary
    contracts
    session
    logfile
    globals.do { ... }
        seconds=0;
        startprice=50;
        price=startprice;
        duration=200;
        delay=5;
        closed=0;
    subjects.do { ... }
        winner=-1;
        price=0;
    Active screen
    Waitingscreen
        Text
            {\rtf \qc \fs40 Thank you for participating.}
Auction =|= (\duration+\delay)N
    globals.do { ... }
        later(delay)do{
        later(if(seconds<duration&closed==0,1,-1))repeat{
        seconds=seconds+1;
        \price=\price-\startprice/\duration;
        }
        }
    Active screen
    Waitingscreen
```

Dutch auction - the background

1. define the **global variables**
2. set the message for the **waiting screen**
3. let the **clock** run, using
   - the `later()do` statement
   - and the `later()repeat` statement

Dutch auction - the graphical clock I



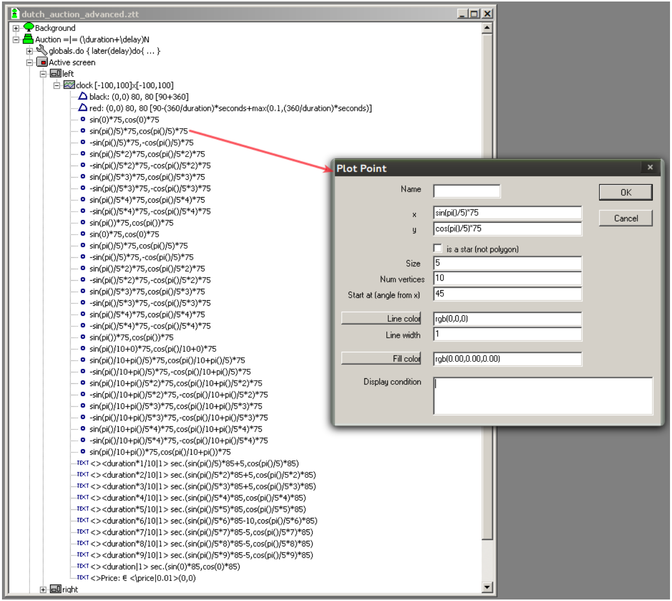
dutch_auction_advanced.ztt

```
Background
Auction =|= (\duration+\delay)N
    globals.do { later(delay)do{ ... }
    Active screen
        left
            clock [-100,100]x[-100,100]
                black: (0,0) 80, 80 [90+360]
                red: (0,0) 80, 80 [90-(360/duration)*seconds+max(0.1,(360/duration)*seconds)]
                sin(0)*75,cos(0)*75
                sin(pi()/5)*75,cos(pi()/5)*75
                -sin(pi()/5)*75,-cos(pi()/5)*75
                sin(pi()/5*2)*75,cos(pi()/5*2)*75
                -sin(pi()/5*2)*75,-cos(pi()/5*2)*75
                sin(pi()/5*3)*75,cos(pi()/5*3)*75
                -sin(pi()/5*3)*75,-cos(pi()/5*3)*75
                sin(pi()/5*4)*75,cos(pi()/5*4)*75
                -sin(pi()/5*4)*75,-cos(pi()/5*4)*75
                sin(pi())*75,cos(pi())*75
                sin(0)*75,cos(0)*75
                sin(pi()/5)*75,cos(pi()/5)*75
                -sin(pi()/5)*75,-cos(pi()/5)*75
                sin(pi()/5*2)*75,cos(pi()/5*2)*75
                -sin(pi()/5*2)*75,-cos(pi()/5*2)*75
                sin(pi()/5*3)*75,cos(pi()/5*3)*75
                -sin(pi()/5*3)*75,-cos(pi()/5*3)*75
                sin(pi()/5*4)*75,cos(pi()/5*4)*75
                -sin(pi()/5*4)*75,-cos(pi()/5*4)*75
                sin(pi())*75,cos(pi())*75
                sin(pi()/10+0)*75,cos(pi()/10+0)*75
                sin(pi()/10+pi()/5)*75,cos(pi()/10+pi()/5)*75
                -sin(pi()/10+pi()/5)*75,-cos(pi()/10+pi()/5)*75
                sin(pi()/10+pi()/5*2)*75,cos(pi()/10+pi()/5*2)*75
                -sin(pi()/10+pi()/5*2)*75,-cos(pi()/10+pi()/5*2)*75
                sin(pi()/10+pi()/5*3)*75,cos(pi()/10+pi()/5*3)*75
                -sin(pi()/10+pi()/5*3)*75,-cos(pi()/10+pi()/5*3)*75
                sin(pi()/10+pi()/5*4)*75,cos(pi()/10+pi()/5*4)*75
                -sin(pi()/10+pi()/5*4)*75,-cos(pi()/10+pi()/5*4)*75
                sin(pi()/10+pi())*75,cos(pi()/10+pi())*75
                TEXT <><duration*1/10|1> sec.(sin(pi()/5)*85+5,cos(pi()/5)*85)
                TEXT <><duration*2/10|1> sec.(sin(pi()/5*2)*85+5,cos(pi()/5*2)*85)
                TEXT <><duration*3/10|1> sec.(sin(pi()/5*3)*85+5,cos(pi()/5*3)*85)
                TEXT <><duration*4/10|1> sec.(sin(pi()/5*4)*85,cos(pi()/5*4)*85)
                TEXT <><duration*5/10|1> sec.(sin(pi()/5*5)*85,cos(pi()/5*5)*85)
                TEXT <><duration*6/10|1> sec.(sin(pi()/5*6)*85-10,cos(pi()/5*6)*85)
                TEXT <><duration*7/10|1> sec.(sin(pi()/5*7)*85-5,cos(pi()/5*7)*85)
                TEXT <><duration*8/10|1> sec.(sin(pi()/5*8)*85-5,cos(pi()/5*8)*85)
                TEXT <><duration*9/10|1> sec.(sin(pi()/5*9)*85-5,cos(pi()/5*9)*85)
                TEXT <><duration|1> sec.(sin(0)*85,cos(0)*85)
                TEXT <>Price: € <\price|0.01>(0,0)
        right
```

Plot Pie

Name		red
center	x	0
	y	0
radius	x	80
	y	80
start angle [°]		90-(360/duration)*seconds
angle [°]		max(0.1,(360/duration)*seconds)
line color		rgb(1.00,0.00,0.00)
line width		15
		☑ do not draw lines to center
fill color		rgb(0.77,0.88,0.88)
Display condition		

Dutch auction - the graphical clock II



dutch_auction_advanced.ztt

```
Background
Auction =|= (\duration+\delay)N
  globals.do { later(delay)do{ ... }
  Active screen
    left
      clock [-100,100]x[-100,100]
        black: (0,0) 80, 80 [90+360]
        red: (0,0) 80, 80 [90-(360/duration)*seconds+max(0.1,(360/duration)*seconds)]
        sin(0)*75,cos(0)*75
        sin(pi()/5)*75,cos(pi()/5)*75
        -sin(pi()/5)*75,-cos(pi()/5)*75
        sin(pi()/5*2)*75,cos(pi()/5*2)*75
        -sin(pi()/5*2)*75,-cos(pi()/5*2)*75
        sin(pi()/5*3)*75,cos(pi()/5*3)*75
        -sin(pi()/5*3)*75,-cos(pi()/5*3)*75
        sin(pi()/5*4)*75,cos(pi()/5*4)*75
        -sin(pi()/5*4)*75,-cos(pi()/5*4)*75
        sin(pi())*75,cos(pi())*75
        sin(0)*75,cos(0)*75
        sin(pi()/5)*75,cos(pi()/5)*75
        -sin(pi()/5)*75,-cos(pi()/5)*75
        sin(pi()/5*2)*75,cos(pi()/5*2)*75
        -sin(pi()/5*2)*75,-cos(pi()/5*2)*75
        sin(pi()/5*3)*75,cos(pi()/5*3)*75
        -sin(pi()/5*3)*75,-cos(pi()/5*3)*75
        sin(pi()/5*4)*75,cos(pi()/5*4)*75
        -sin(pi()/5*4)*75,-cos(pi()/5*4)*75
        sin(pi())*75,cos(pi())*75
        sin(pi()/10+0)*75,cos(pi()/10+0)*75
        sin(pi()/10+pi()/5)*75,cos(pi()/10+pi()/5)*75
        -sin(pi()/10+pi()/5)*75,-cos(pi()/10+pi()/5)*75
        sin(pi()/10+pi()/5*2)*75,cos(pi()/10+pi()/5*2)*75
        -sin(pi()/10+pi()/5*2)*75,-cos(pi()/10+pi()/5*2)*75
        sin(pi()/10+pi()/5*3)*75,cos(pi()/10+pi()/5*3)*75
        -sin(pi()/10+pi()/5*3)*75,-cos(pi()/10+pi()/5*3)*75
        sin(pi()/10+pi()/5*4)*75,cos(pi()/10+pi()/5*4)*75
        -sin(pi()/10+pi()/5*4)*75,-cos(pi()/10+pi()/5*4)*75
        sin(pi()/10+pi())*75,cos(pi()/10+pi())*75
        TEXT <><duration*1/10|1> sec.(sin(pi()/5)*85+5,cos(pi()/5)*85)
        TEXT <><duration*2/10|1> sec.(sin(pi()/5*2)*85+5,cos(pi()/5*2)*85)
        TEXT <><duration*3/10|1> sec.(sin(pi()/5*3)*85+5,cos(pi()/5*3)*85)
        TEXT <><duration*4/10|1> sec.(sin(pi()/5*4)*85,cos(pi()/5*4)*85)
        TEXT <><duration*5/10|1> sec.(sin(pi()/5*5)*85,cos(pi()/5*5)*85)
        TEXT <><duration*6/10|1> sec.(sin(pi()/5*6)*85-10,cos(pi()/5*6)*85)
        TEXT <><duration*7/10|1> sec.(sin(pi()/5*7)*85-5,cos(pi()/5*7)*85)
        TEXT <><duration*8/10|1> sec.(sin(pi()/5*8)*85-5,cos(pi()/5*8)*85)
        TEXT <><duration*9/10|1> sec.(sin(pi()/5*9)*85-5,cos(pi()/5*9)*85)
        TEXT <><duration|1> sec.(sin(0)*85,cos(0)*85)
        TEXT <>Price: € <\price|0.01>(0,0)
    right
```

Plot Point

Name:

x: sin(pi()/5)*75

y: cos(pi()/5)*75

☐ is a star (not polygon)

Size: 5

Num vertices: 10

Start at (angle from x): 45

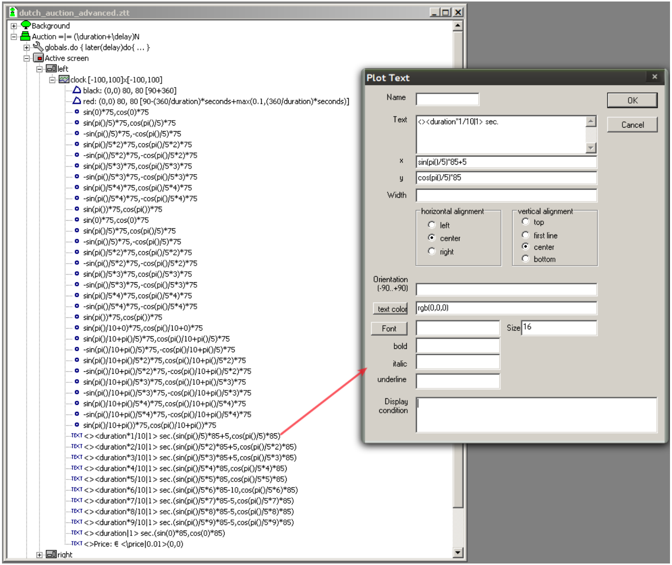
Line color: rgb(0,0,0)

Line width: 1

Fill color: rgb(0.00,0.00,0.00)

Display condition:

Dutch auction - the graphical clock III



dutch_auction_advanced.ztt

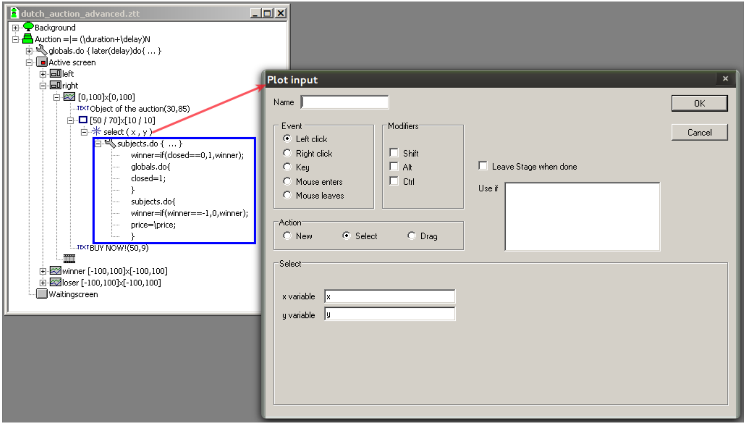
```
Background
Auction =|= (\duration+\delay)N
  globals.do { later(delay)do{ ... }
  Active screen
    left
      clock [-100,100]x[-100,100]
        black: (0,0) 80, 80 [90+360]
        red: (0,0) 80, 80 [90-(360/duration)*seconds+max(0.1,(360/duration)*seconds)]
        sin(0)*75,cos(0)*75
        sin(pi()/5)*75,cos(pi()/5)*75
        -sin(pi()/5)*75,-cos(pi()/5)*75
        sin(pi()/5*2)*75,cos(pi()/5*2)*75
        -sin(pi()/5*2)*75,-cos(pi()/5*2)*75
        sin(pi()/5*3)*75,cos(pi()/5*3)*75
        -sin(pi()/5*3)*75,-cos(pi()/5*3)*75
        sin(pi()/5*4)*75,cos(pi()/5*4)*75
        -sin(pi()/5*4)*75,-cos(pi()/5*4)*75
        sin(pi())*75,cos(pi())*75
        sin(0)*75,cos(0)*75
        sin(pi()/5)*75,cos(pi()/5)*75
        -sin(pi()/5)*75,-cos(pi()/5)*75
        sin(pi()/5*2)*75,cos(pi()/5*2)*75
        -sin(pi()/5*2)*75,-cos(pi()/5*2)*75
        sin(pi()/5*3)*75,cos(pi()/5*3)*75
        -sin(pi()/5*3)*75,-cos(pi()/5*3)*75
        sin(pi()/5*4)*75,cos(pi()/5*4)*75
        -sin(pi()/5*4)*75,-cos(pi()/5*4)*75
        sin(pi())*75,cos(pi())*75
        sin(pi()/10+0)*75,cos(pi()/10+0)*75
        sin(pi()/10+pi()/5)*75,cos(pi()/10+pi()/5)*75
        -sin(pi()/10+pi()/5)*75,-cos(pi()/10+pi()/5)*75
        sin(pi()/10+pi()/5*2)*75,cos(pi()/10+pi()/5*2)*75
        -sin(pi()/10+pi()/5*2)*75,-cos(pi()/10+pi()/5*2)*75
        sin(pi()/10+pi()/5*3)*75,cos(pi()/10+pi()/5*3)*75
        -sin(pi()/10+pi()/5*3)*75,-cos(pi()/10+pi()/5*3)*75
        sin(pi()/10+pi()/5*4)*75,cos(pi()/10+pi()/5*4)*75
        -sin(pi()/10+pi()/5*4)*75,-cos(pi()/10+pi()/5*4)*75
        sin(pi()/10+pi())*75,cos(pi()/10+pi())*75
        TEXT <><duration*1/10|1> sec.(sin(pi()/5)*85+5,cos(pi()/5)*85)
        TEXT <><duration*2/10|1> sec.(sin(pi()/5*2)*85+5,cos(pi()/5*2)*85)
        TEXT <><duration*3/10|1> sec.(sin(pi()/5*3)*85+5,cos(pi()/5*3)*85)
        TEXT <><duration*4/10|1> sec.(sin(pi()/5*4)*85,cos(pi()/5*4)*85)
        TEXT <><duration*5/10|1> sec.(sin(pi()/5*5)*85,cos(pi()/5*5)*85)
        TEXT <><duration*6/10|1> sec.(sin(pi()/5*6)*85-10,cos(pi()/5*6)*85)
        TEXT <><duration*7/10|1> sec.(sin(pi()/5*7)*85-5,cos(pi()/5*7)*85)
        TEXT <><duration*8/10|1> sec.(sin(pi()/5*8)*85-5,cos(pi()/5*8)*85)
        TEXT <><duration*9/10|1> sec.(sin(pi()/5*9)*85-5,cos(pi()/5*9)*85)
        TEXT <><duration|1> sec.(sin(0)*85,cos(0)*85)
        TEXT <>Price: € <\price|0.01>(0,0)
    right
```

Plot Text

Name	
Text	<><duration*1/10|1> sec.
x	sin(pi()/5)*85+5
y	cos(pi()/5)*85
Width	
horizontal alignment	○ left ● center ○ right
vertical alignment	○ top ○ first line ● center ○ bottom
Orientation (-90..+90)	
text color	rgb(0,0,0)
Font	
Size	16
bold	
italic	
underline	
Display condition	

Dutch auction - the input

1. Transform a `Rectangle` in a button, by adding a `plot input`.
2. Add a `program` to trigger the consequences of the subject's action.



dutch_auction_advanced.ztt

```
⊞ Background
⊟ Auction =|= (\duration+\delay)N
    ⊞ globals.do { later(delay)do{ ... }
    ⊟ Active screen
        ⊞ left
        ⊟ right
            ⊟ [0,100]x[0,100]
                TEXT Object of the auction(30,85)
                ⊟ [50 / 70]x[10 / 10]
                    ⊟ select ( x , y )
                        ⊟ subjects.do {  ... }
                            winner=if(closed==0,1,winner);
                            globals.do{
                            closed=1;
                            }
                            subjects.do{
                            winner=if(winner==-1,0,winner);
                            price=\price;
                            }
                TEXT BUY NOW!(50,9)
        ⊞ winner [-100,100]x[-100,100]
        ⊞ loser [-100,100]x[-100,100]
    Waitingscreen
```

Plot input

Name

Event: ● Left click ○ Right click ○ Key ○ Mouse enters ○ Mouse leaves

Modifiers: □ Shift □ Alt □ Ctrl

□ Leave Stage when done

Use if

Action: ○ New ● Select ○ Drag

Select

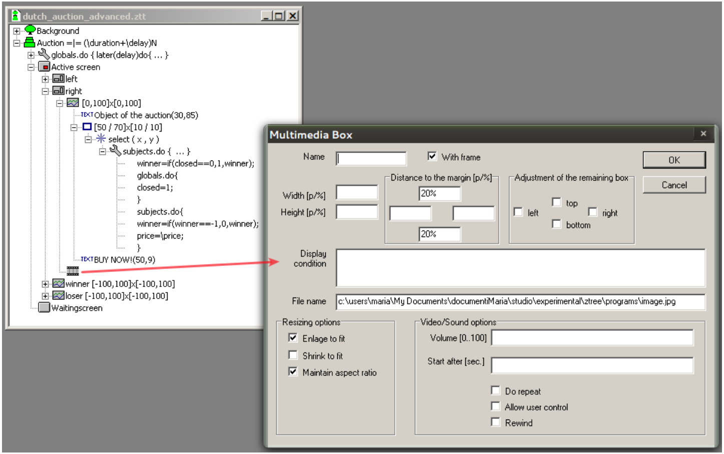
x variable: x

y variable: y

Dutch auction - the figure

Use a `Multimedia box` to insert the picture of the object of the auction.

Remember to write the full file path.



dutch_auction_advanced.ztt

```
Background
Auction =|= (\duration+\delay)N
    globals.do { later(delay)do{ ... }
    Active screen
        left
        right
            [0,100]x[0,100]
                Object of the auction(30,85)
                [50 / 70]x[10 / 10]
                    select ( x , y )
                        subjects.do {  ... }
                            winner=if(closed==0,1,winner);
                            globals.do{
                            closed=1;
                            }
                            subjects.do{
                            winner=if(winner==-1,0,winner);
                            price=\price;
                            }
                BUY NOW!(50,9)

        winner [-100,100]x[-100,100]
        loser [-100,100]x[-100,100]
    Waitingscreen
```

Multimedia Box

Name:

☑ With frame

Width [p/%]:

Height [p/%]:

Distance to the margin [p/%]: top 20%, bottom 20%

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

Display condition:

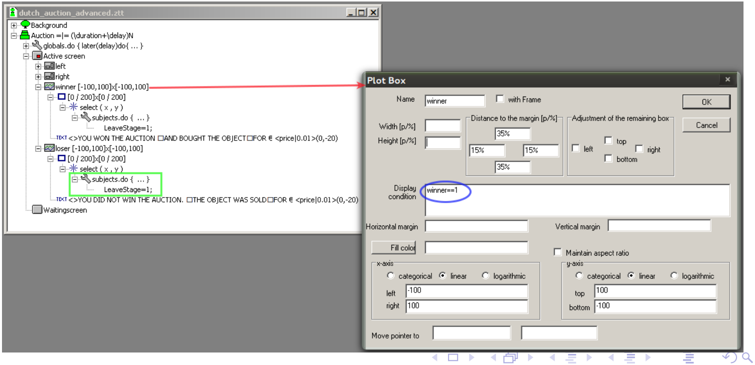
File name: c:\users\maria\My Documents\documentiMaria\studio\experimental\ztree\programs\image.jpg

Resizing options:
☑ Enlage to fit
☐ Shrink to fit
☑ Maintain aspect ratio

Video/Sound options:
Volume [0..100]:
Start after [sec.]:
☐ Do repeat
☐ Allow user control
☐ Rewind

Dutch auction - the final message

1. Use the `Display condition` to show different messages to the winner of the auction and to the other subjects.
2. With a `program` within a `plot input`, you let the subjects leave the stage when they click on the final message.



```
dutch_auction_advanced.ztt
⊞ Background
⊟ Auction =|= (\duration+\delay)N
  ⊞ globals.do { later(delay)do{ ... }
  ⊟ Active screen
    ⊞ left
    ⊞ right
    ⊟ winner [-100,100]x[-100,100]
      ⊟ [0 / 200]x[0 / 200]
        ⊟ select ( x , y )
          ⊟ subjects.do { ... }
                LeaveStage=1;
      TEXT <>YOU WON THE AUCTION □AND BOUGHT THE OBJECT□FOR € <price|0.01>(0,-20)
    ⊟ loser [-100,100]x[-100,100]
      ⊟ [0 / 200]x[0 / 200]
        ⊟ select ( x , y )
          ⊟ subjects.do { ... }
                LeaveStage=1;
      TEXT <>YOU DID NOT WIN THE AUCTION. □THE OBJECT WAS SOLD□FOR € <price|0.01>(0,-20)
  Waitingscreen
```

Plot Box

Name: winner

□ with Frame

Width [p/%]:

Height [p/%]:

Distance to the margin [p/%]: 35% / 15% / 15% / 35%

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

Display condition: winner==1

Horizontal margin:

Vertical margin:

Fill color:

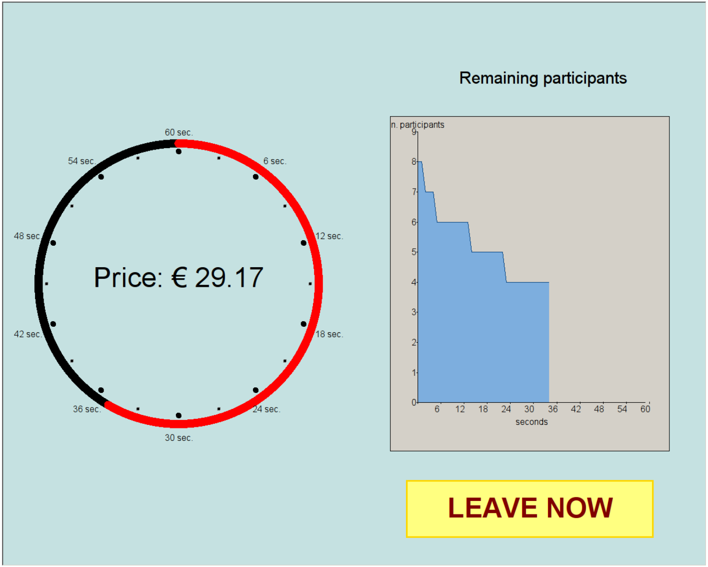
□ Maintain aspect ratio

x-axis: ○ categorical ● linear ○ logarithmic; left -100; right 100

y-axis: ○ categorical ● linear ○ logarithmic; top 100; bottom -100

Move pointer to:

Exercise: English auction



solution: english_auction_advanced.ztt

Exercise: what changes?

1. price **increases** in time
2. subject's action: **leave** the auction
3. **plot** on the right, showing the number of remaining participants, and the time when each of the others left the auction ⇒ how to implement it?

Exercise: what changes?

1. price **increases** in time

2. subject's action: **leave** the auction

3. **plot** on the right, showing the number of remaining participants, and the time when each of the others left the auction ⇒ how to implement it?
   Suggestions:
   - save the number of remaining subjects at each point in time in a `user defined table` or in the `contracts table`
   - plot the content of this table using a `graph`, which is one of the `plot items`

A real effort task: pick the right color

The **task**:

1. correctly answer as many questions as possible
2. in a given time interval (60 seconds)

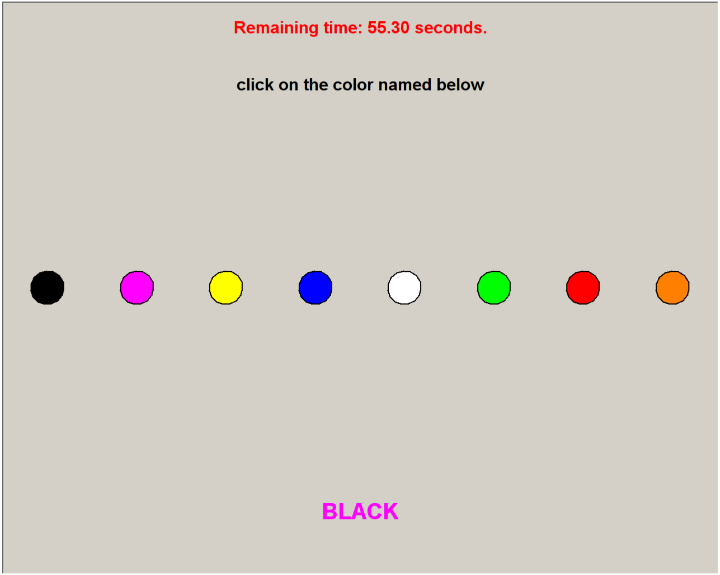
Two different **questions**:

1. click on the color of the word written on the screen
2. click on the color corresponding to the word written on the screen

To **answer**, the subject must click on one of eight alternative colors, presented in random order on the screen.

Example: Colors.ztt

Screenshot - I

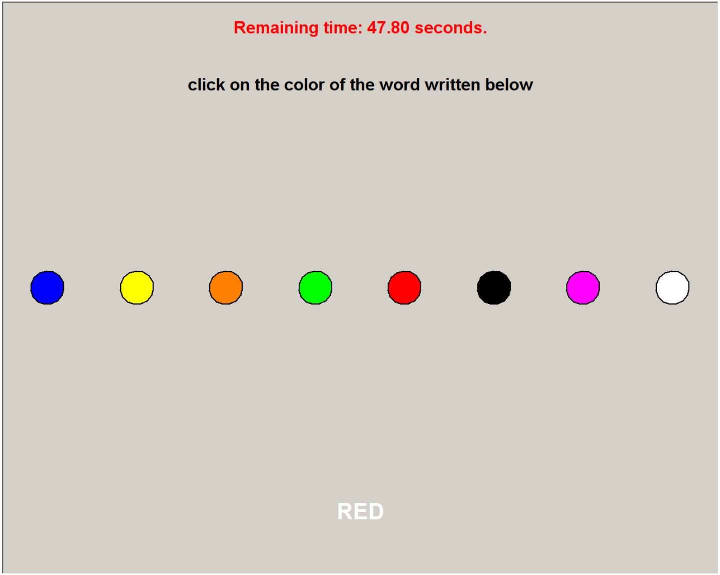


Remaining time: 55.30 seconds.

click on the color named below

BLACK

Screenshot - II

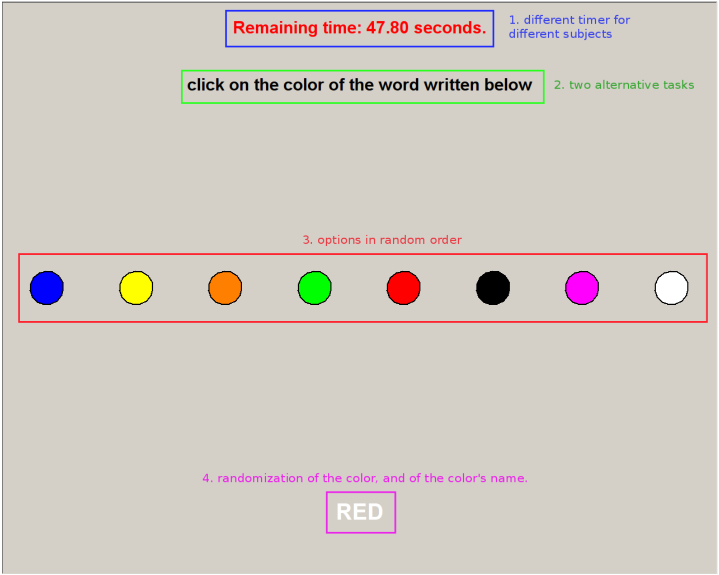


Remaining time: 47.80 seconds.

click on the color of the word written below

RED

Screenshot - III



Remaining time: 47.80 seconds.

1. different timer for different subjects

click on the color of the word written below

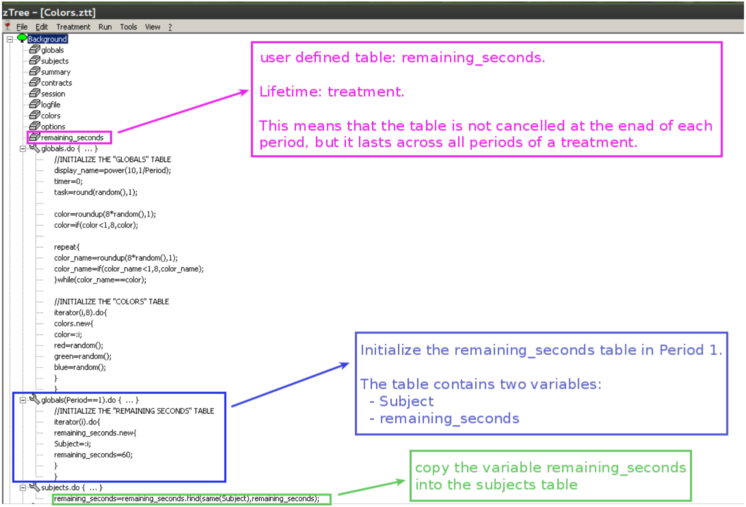
2. two alternative tasks

3. options in random order

4. randomization of the color, and of the color's name.

RED

Remaining seconds



```
Background
  globals
  subjects
  summary
  contracts
  session
  logfile
  colors
  options
  remaining_seconds
  globals.do { ... }
      //INITIALIZE THE "GLOBALS" TABLE
      display_name=power(10,1/Period);
      timer=0;
      task=round(random(),1);

      color=roundup(8*random(),1);
      color=if(color<1,8,color);

      repeat{
      color_name=roundup(8*random(),1);
      color_name=if(color_name<1,8,color_name);
      }while(color_name==color);

      //INITIALIZE THE "COLORS" TABLE
      iterator(i,8).do{
      colors.new{
      color=:i;
      red=random();
      green=random();
      blue=random();
      }
      }
  globals(Period==1).do { ... }
      //INITIALIZE THE "REMAINING SECONDS" TABLE
      iterator(i).do{
      remaining_seconds.new{
      Subject=:i;
      remaining_seconds=60;
      }
      }
  subjects.do { ... }
      remaining_seconds=remaining_seconds.find(same(Subject),remaining_seconds);
```

user defined table: remaining_seconds.

Lifetime: treatment.

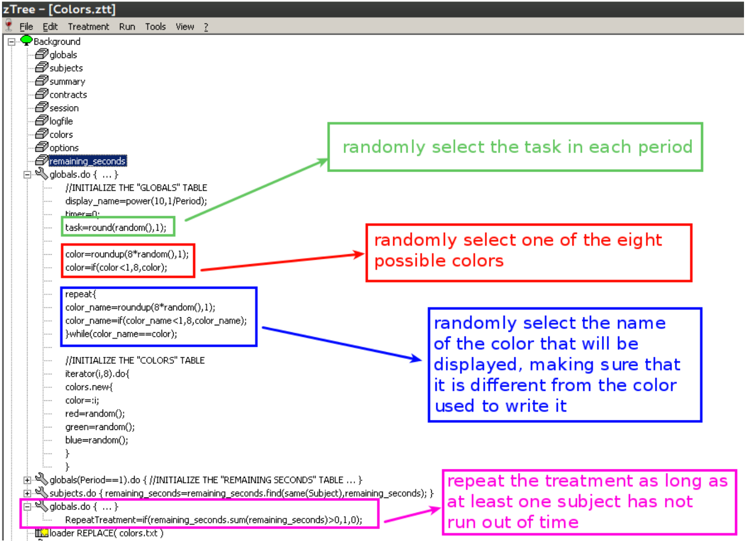
This means that the table is not cancelled at the enad of each period, but it lasts across all periods of a treatment.

Initialize the remaining_seconds table in Period 1.

The table contains two variables:
- Subject
- remaining_seconds

copy the variable remaining_seconds into the subjects table

Randomize the task



```
globals.do { ... }
    //INITIALIZE THE "GLOBALS" TABLE
    display_name=power(10,1/Period);
    timer=0;
    task=round(random(),1);

    color=roundup(8*random(),1);
    color=if(color<1,8,color);

    repeat{
    color_name=roundup(8*random(),1);
    color_name=if(color_name<1,8,color_name);
    }while(color_name==color);

    //INITIALIZE THE "COLORS" TABLE
    iterator(i,8).do{
    colors.new{
    color=:i;
    red=random();
    green=random();
    blue=random();
    }
    }
globals(Period==1).do { //INITIALIZE THE "REMAINING SECONDS" TABLE ... }
subjects.do { remaining_seconds=remaining_seconds.find(same(Subject),remaining_seconds); }
globals.do { ... }
    RepeatTreatment=if(remaining_seconds.sum(remaining_seconds)>0,1,0);
loader REPLACE( colors.txt )
```

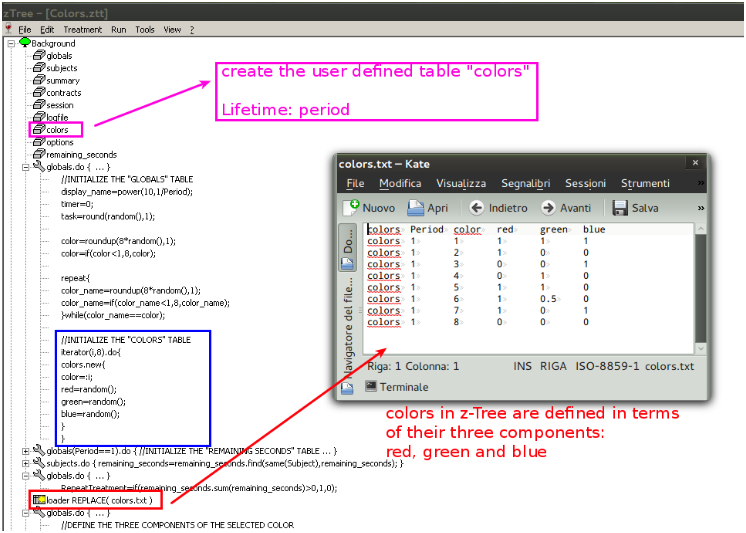
randomly select the task in each period

randomly select one of the eight possible colors

randomly select the name of the color that will be displayed, making sure that it is different from the color used to write it

repeat the treatment as long as at least one subject has not run out of time

Define the colors



zTree - [Colors.ztt]
File Edit Treatment Run Tools View ?
colors.txt – Kate
File Modifica Visualizza Segnalibri Sessioni Strumenti
Nuovo Apri Indietro Avanti Salva
Do...
Navigatore del file...
Riga: 1 Colonna: 1 INS RIGA ISO-8859-1 colors.txt
Terminale

```
Background
    globals
    subjects
    summary
    contracts
    session
    logfile
    colors
    options
    remaining_seconds
    globals.do { ... }
        //INITIALIZE THE "GLOBALS" TABLE
        display_name=power(10,1/Period);
        timer=0;
        task=round(random(),1);

        color=roundup(8*random(),1);
        color=if(color<1,8,color);

        repeat{
        color_name=roundup(8*random(),1);
        color_name=if(color_name<1,8,color_name);
        }while(color_name==color);

        //INITIALIZE THE "COLORS" TABLE
        iterator(i,8).do{
        colors.new{
        color=:i;
        red=random();
        green=random();
        blue=random();
        }
        }
    globals(Period==1).do { //INITIALIZE THE "REMAINING SECONDS" TABLE ... }
    subjects.do { remaining_seconds=remaining_seconds.find(same(Subject),remaining_seconds); }
    globals.do { ... }
        RepeatTreatment=if(remaining_seconds.sum(remaining_seconds)>0,1,0);
    loader REPLACE( colors.txt )
    globals.do { ... }
        //DEFINE THE THREE COMPONENTS OF THE SELECTED COLOR
```

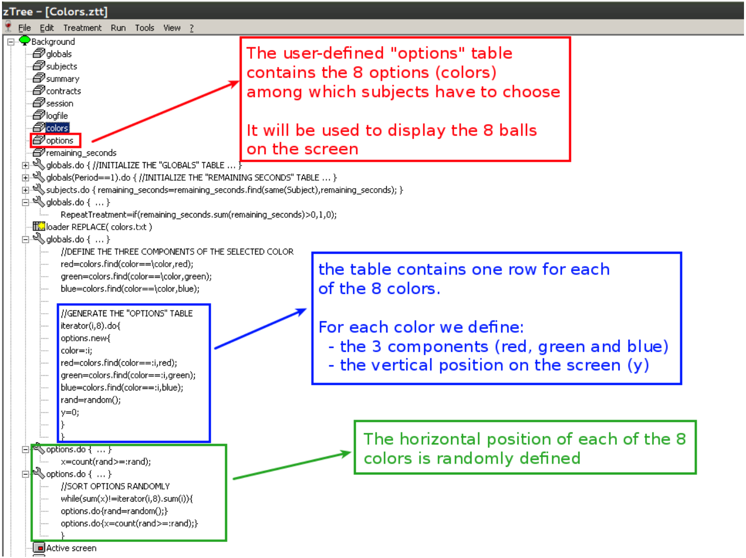
create the user defined table "colors"

Lifetime: period

```
colors	Period	color	red	green	blue
colors	1	1	1	1	1
colors	1	2	1	0	0
colors	1	3	0	0	1
colors	1	4	0	1	0
colors	1	5	1	1	0
colors	1	6	1	0.5	0
colors	1	7	1	0	1
colors	1	8	0	0	0
```

colors in z-Tree are defined in terms of their three components: red, green and blue

Define the options



zTree - [Colors.ztt]

```
Background
    globals
    subjects
    summary
    contracts
    session
    logfile
    colors
    options
    remaining_seconds
    globals.do { //INITIALIZE THE "GLOBALS" TABLE ... }
    globals(Period==1).do { //INITIALIZE THE "REMAINING SECONDS" TABLE ... }
    subjects.do { remaining_seconds=remaining_seconds.find(same(Subject),remaining_seconds); }
    globals.do { ... }
        RepeatTreatment=if(remaining_seconds.sum(remaining_seconds)>0,1,0);
    loader REPLACE( colors.txt )
    globals.do { ... }
        //DEFINE THE THREE COMPONENTS OF THE SELECTED COLOR
        red=colors.find(color==\color,red);
        green=colors.find(color==\color,green);
        blue=colors.find(color==\color,blue);

        //GENERATE THE "OPTIONS" TABLE
        iterator(i,8).do{
        options.new{
        color=:i;
        red=colors.find(color==:i,red);
        green=colors.find(color==:i,green);
        blue=colors.find(color==:i,blue);
        rand=random();
        y=0;
        }
        }
    options.do { ... }
        x=count(rand>=:rand);
    options.do { ... }
        //SORT OPTIONS RANDOMLY
        while(sum(x)!=iterator(i,8).sum(i)){
        options.do{rand=random();}
        options.do{x=count(rand>=:rand);}
        }
    Active screen
```

The user-defined "options" table contains the 8 options (colors) among which subjects have to choose

It will be used to display the 8 balls on the screen

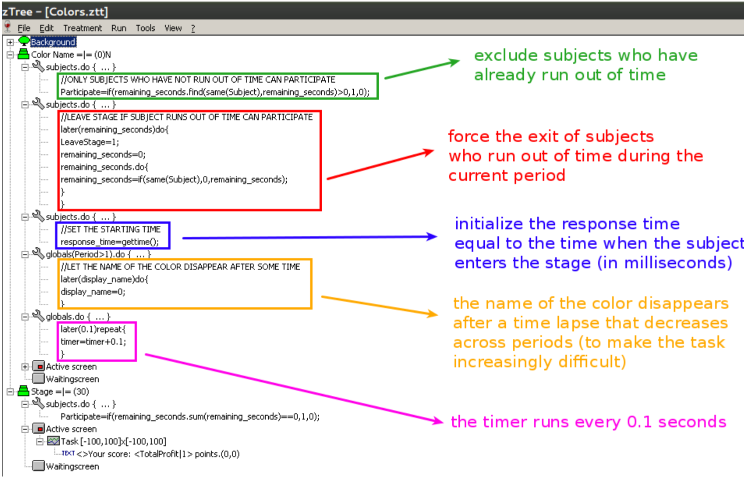
the table contains one row for each of the 8 colors.

For each color we define:

- the 3 components (red, green and blue)
- the vertical position on the screen (y)

The horizontal position of each of the 8 colors is randomly defined

Programs



```
Background
Color Name =|= (0)N
  subjects.do { ... }
    //ONLY SUBJECTS WHO HAVE NOT RUN OUT OF TIME CAN PARTICIPATE
    Participate=if(remaining_seconds.find(same(Subject),remaining_seconds)>0,1,0);
  subjects.do { ... }
    //LEAVE STAGE IF SUBJECT RUNS OUT OF TIME CAN PARTICIPATE
    later(remaining_seconds)do{
    LeaveStage=1;
    remaining_seconds=0;
    remaining_seconds.do{
    remaining_seconds=if(same(Subject),0,remaining_seconds);
    }
    }
  subjects.do { ... }
    //SET THE STARTING TIME
    response_time=gettime();
  globals(Period>1).do { ... }
    //LET THE NAME OF THE COLOR DISAPPEAR AFTER SOME TIME
    later(display_name)do{
    display_name=0;
    }
  globals.do { ... }
    later(0.1)repeat{
    timer=timer+0.1;
    }
  Active screen
  Waitingscreen
Stage =|= (30)
  subjects.do { ... }
    Participate=if(remaining_seconds.sum(remaining_seconds)==0,1,0);
  Active screen
    Task [-100,100]x[-100,100]
      TEXT <>Your score: <TotalProfit|1> points.(0,0)
  Waitingscreen
```

exclude subjects who have already run out of time

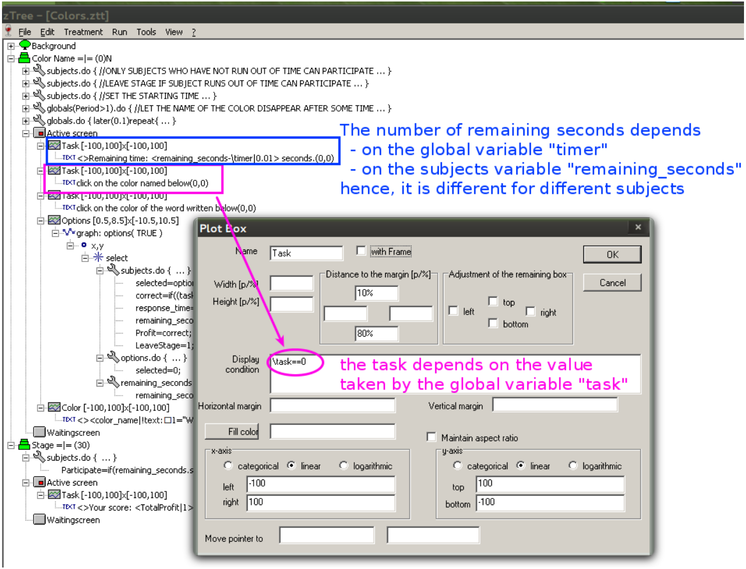
force the exit of subjects who run out of time during the current period

initialize the response time equal to the time when the subject enters the stage (in milliseconds)

the name of the color disappears after a time lapse that decreases across periods (to make the task increasingly difficult)

the timer runs every 0.1 seconds

Information to be displayed



```
Background
Color Name =|= (0)N
  subjects.do { //ONLY SUBJECTS WHO HAVE NOT RUN OUT OF TIME CAN PARTICIPATE ... }
  subjects.do { //LEAVE STAGE IF SUBJECT RUNS OUT OF TIME CAN PARTICIPATE ... }
  subjects.do { //SET THE STARTING TIME ... }
  globals(Period>1).do { //LET THE NAME OF THE COLOR DISAPPEAR AFTER SOME TIME ... }
  globals.do { later(0.1)repeat{ ... }
  Active screen
    Task [-100,100]x[-100,100]
      TEXT <>Remaining time: <remaining_seconds-\timer|0.01> seconds.(0,0)
    Task [-100,100]x[-100,100]
      TEXT click on the color named below(0,0)
    Task [-100,100]x[-100,100]
      TEXT click on the color of the word written below(0,0)
    Options [0.5,8.5]x[-10.5,10.5]
      graph: options( TRUE )
        x,y
          select
            subjects.do { ... }
                selected=optio
                correct=if((tas
                response_time=
                remaining_seco
                Profit=correct;
                LeaveStage=1;
            options.do { ... }
                selected=0;
            remaining_seconds
                remaining_seco
    Color [-100,100]x[-100,100]
      TEXT <><color_name|!text:□1="W
  Waitingscreen
Stage =|= (30)
  subjects.do { ... }
      Participate=if(remaining_seconds.s
  Active screen
    Task [-100,100]x[-100,100]
      TEXT <>Your score: <TotalProfit|1>
  Waitingscreen
```

Plot Box

Name: Task □ with Frame

Width [p/%]:

Height [p/%]:

Distance to the margin [p/%]: 10% / 80%

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

Display condition: \task==0

Horizontal margin:

Vertical margin:

Fill color:

□ Maintain aspect ratio

x-axis: ○ categorical ● linear ○ logarithmic; left -100; right 100

y-axis: ○ categorical ● linear ○ logarithmic; top 100; bottom -100

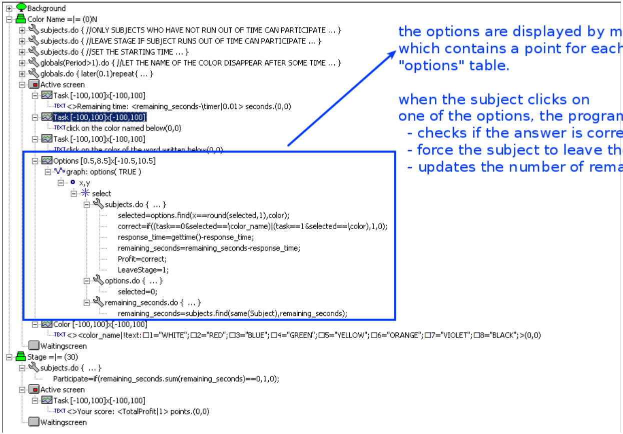
Move pointer to:

The number of remaining seconds depends
- on the global variable "timer"
- on the subjects variable "remaining_seconds"

hence, it is different for different subjects

the task depends on the value taken by the global variable "task"

Subjects' actions



```
⊞ Background
⊟ Color Name =|= (0)N
    ⊞ subjects.do { //ONLY SUBJECTS WHO HAVE NOT RUN OUT OF TIME CAN PARTICIPATE ... }
    ⊞ subjects.do { //LEAVE STAGE IF SUBJECT RUNS OUT OF TIME CAN PARTICIPATE ... }
    ⊞ subjects.do { //SET THE STARTING TIME ... }
    ⊞ globals(Period>1).do { //LET THE NAME OF THE COLOR DISAPPEAR AFTER SOME TIME ... }
    ⊞ globals.do { later(0.1)repeat{ ... }
    ⊟ Active screen
        ⊟ Task [-100,100]x[-100,100]
            TEXT <>Remaining time: <remaining_seconds-\timer|0.01> seconds.(0,0)
        ⊟ Task [-100,100]x[-100,100]
            TEXT click on the color named below(0,0)
        ⊟ Task [-100,100]x[-100,100]
            TEXT click on the color of the word written below(0,0)
        ⊟ Options [0.5,8.5]x[-10.5,10.5]
            ⊟ graph: options( TRUE )
                ⊟ x,y
                    ⊟ select
                        ⊟ subjects.do {  ... }
                              selected=options.find(x==round(selected,1),color);
                              correct=if((task==0&selected==\color_name)|(task==1&selected==\color),1,0);
                              response_time=gettime()-response_time;
                              remaining_seconds=remaining_seconds-response_time;
                              Profit=correct;
                              LeaveStage=1;
                        ⊟ options.do {  ... }
                              selected=0;
                        ⊟ remaining_seconds.do {  ... }
                              remaining_seconds=subjects.find(same(Subject),remaining_seconds);
        ⊟ Color [-100,100]x[-100,100]
            TEXT <><color_name|!text:□1="WHITE";□2="RED";□3="BLUE";□4="GREEN";□5="YELLOW";□6="ORANGE";□7="VIOLET";□8="BLACK";>(0,0)
    Waitingscreen
⊟ Stage =|= (30)
    ⊟ subjects.do {  ... }
          Participate=if(remaining_seconds.sum(remaining_seconds)==0,1,0);
    ⊟ Active screen
        ⊟ Task [-100,100]x[-100,100]
            TEXT <>Your score: <TotalProfit|1> points.(0,0)
    Waitingscreen
```

the options are displayed by means of a graph which contains a point for each element of the "options" table.

when the subject clicks on one of the options, the program

- checks if the answer is correct
- force the subject to leave the stage
- updates the number of remaining seconds