Behav Res (2016) 48:1454–1475
DOI 10.3758/s13428-015-0660-6

Bonn eXperimental System (BoXS): An open-source platform for interactive experiments in psychology and economics

Mirko Seithe[1] · **Jeronim Morina**[2] · **Andreas Glöckner**[2,3]

Published online: 11 November 2015

Abstract The increased interest in complex-interactive behavior on the one hand and the cognitive and affective processes underlying behavior on the other are a challenge for researchers in psychology and behavioral economics. Research often necessitates that participants strategically interact with each other in dyads or groups. At the same time, to investigate the underlying cognitive and affective processes in a fine-grained manner, not only choices but also other variables such as decision time, information search, and pupil dilation should be recorded. The Bonn eXperimental System (BoXS) introduced in this article is an open-source platform that allows interactive as well as non-interactive experiments to be conducted while recording process measures very efficiently and completely browser-based. In the current version, BoXS has particularly been extended to enable conducting interactive eye-tracking and mouse-tracking experiments. One core advantage of BoXS is its simplicity. Using BoXS does not require prior installation for both experimenters and participants, which allows for running studies outside the laboratory and over the internet. Learning to program for BoXS is easy even for researchers without previous programming experience.

✉ Andreas Glöckner
andreas.gloeckner@fernuni-hagen.de

[1] Fraunhofer Institute for Applied Information Technology, Sankt Augustin, Germany

[2] Max Planck Institute for Research on Collective Goods, Bonn, Germany

[3] University of Hagen (FernUniversität), Universitätsstrasse 27, D-58097, Hagen, Germany

Keywords Methodology · Internet experiments · Interactive decision making · Eye-tracking · Social dilemmas · Process tracing · Mechanical turk

Introduction

One important paradigm of investigation in social psychology and behavioral economics is the *interactive study* in which the behavior of two or more persons jointly determines the consequences for the participants and the collective. Interactive studies can be contrasted with non-interactive studies mainly used in fields such as cognitive psychology, in which consequences of behavior are not influenced by the behavior of other persons and are only dependent on the person's own behavior and some predefined algorithm (e.g., determining which answer is right or wrong) or chance. The high and still increasing interest in interactive studies (for reviews see Chaudhuri, 2011; Dawes, 1980; Komorita & Parks, 1995; Sally, 1995; Van Lange, Joireman, Parks, & Van Dijk, 2013) is due to several reasons. Some of the most severe problems society faces today – such as environmental protection and depletion of natural resources – depend on coordinated interactive behavior between persons. Such real-world situations are often structured as a social dilemma in which incentives for the collective and the individual agent diverge (Hardin, 1968). The prototypical examples for an interactive problem structure are prisoners' dilemmas and public good games. In both dilemmas, persons have to simultaneously make strategic decisions about whether to choose the option that maximizes their self-interest (i.e., to defect) or to choose the option that maximizes the outcome for the collective (i.e., to cooperate), if the others in the collective cooperate as well (Rapoport & Chammah, 1965). A good understanding of the mechanisms underlying cognitive and affective processes in

strategic decision making is consequentially of crucial importance for society.

Running interactive studies comes with the pragmatic complication that many individuals participate simultaneously and that their responses have to be anonymously matched in groups of varying size. While in early work on strategic decision making researchers had to rely on effortful and error-prone manual matching or had to use deception (i.e., misinforming participants that they interact with each other although they do not), nowadays research is mainly conducted using software that is specifically developed for this purpose.

One of the most prominent tools for user-generated experiments in this domain is the Zurich Toolbox for ready-made economic experiments (z-Tree). z-Tree, which is developed and maintained by Urs Fischbacher (2007), is probably the most mature and powerful experiment system in this domain yet and continues to be very popular, especially among experimental economists, and this immense popularity is documented in an impressive citation count (Thomson Reuter Web of Knowledge: 1,125 citations for the document as of October, 2014). It offers a graphic approach to programming which appeals to many experimental scientists who either lack sophisticated programming skills or prefer the convenience of an intuitive programming interface. As a core advantage over all-purpose tools (e.g., Java), z-Tree provides easy routines for matching agents and transferring information between them, which is essential for any kind of interactive studies. Technically, the system consists of the programs z-Tree, which contains both the server and the experimenter interface, and z-Leaf, which is the client interface for multiple participants that can interact with each other.

Despite its popularity, z-Tree also has its limitations. Since z-Tree has been developed to be used in economic experiments, it has limited capabilities to record process measures[1] due to the fact that economists are typically less interested in cognitive processes than psychologists. Specifically, it lacks support for precise response-time measurement as well as mouse- and eye-tracking, which are of great interest for many behavioral researchers.[2] Furthermore, the architecture, which requires the manual installation of client software, makes non-laboratory set-ups such as large-scale interactive internet experiments and studies conducted using workers from Amazon's Mechanical Turk (Mason & Suri, 2012) very difficult or in many cases even impossible to conduct. Finally, z-Tree is provided only for the Windows operating system and is based on a closed-source code which makes the system hard to modify or expand upon without explicit support.

Hence, aside from many advantages, z-Tree's limited capacity for investigating cognitive processes in a fine-grained manner (e.g., by recoding eye-fixations that precede a choice) reduces the program's usefulness for psychologists. Researchers that are, for instance, interested in running interactive z-Tree studies with eye-tracking need to rely on effortful and error-prone workarounds based on manual coding (e.g., Fiedler, Glöckner, & Nicklisch, 2012; Fiedler, Glöckner, Nicklisch, & Dickert, 2013).

One alternative is to rely on more general programs for developing and running non-interactive studies. These more general programs have usually been created for local experiments with individual participants that do not interact with each other (Table 1, first section). Programs such as NBS Presentation, E-Prime, or their non-commercial counterparts such as OpenSesame (Mathot, Schreij, & Theeuwes, 2012), PsychoPy (Peirce, 2007), PsyToolkit (Stoet, 2010) or others (e.g., Geller, Schleifer, Sederberg, Jacobs, & Kahana, 2007; von Bastian, Locher, & Ruflin, 2013) already have excellent capabilities for recording process measure. These programs could also possibly be expanded by including libraries that enhance the capabilities for matching, interconnecting, and simultaneously supervising participants in interactive studies, but standard extensions are not available yet and implementing them might be relatively effortful.

A second alternative are programs that have been developed to run browser-based online experiments relying on Javascript (Table 1, second section). Tools such as jsPsych (de Leeuw, 2014) and WEXTOR (Reips & Neuhaus, 2002) have been designed for enabling researchers to develop browser-based studies that run on any platform and without any software installation. jsPsych, for example, is a JavaScript library for creating and running experiments in a browser, which allows building a wide range of experiments that can be run online. jsPsych provides the structure for a study and several plugins, that are ready-made templates for simple experimental tasks such as displaying instructions and collecting responses. The researcher can create a study by adapting a partially pre-specified javascript code that calls these plugins in the order they are needed for implementing the research design. Still, although being based on a generally very promising approach, to date neither jsPsych nor WEXTOR provide standard functionality to manage and supervise the interaction between many participants.

The Bonn eXperiment System (BoXS) presented in this paper provides an alternative that has been developed from

[1] The technical term "process measures" refers to measures of observable variables that are recorded while an individual makes a decision. This is usually done in order to infer how a decider came to his or her choice. Process measures therefore allow testing isomorphic models of decision making, which describe not only the outcome of a choice process but also more details concerning information processing. Prominent process measures are response time, mouse clicks, or mouse-movements (i.e., mouse-tracking), shifts in attention measured by eye-tracking, and measurement of physiologic arousal (e.g., skin conductance or pupil dilation) (for an overview see Schulte-Mecklenbeck, Kühberger & Ranyard, 2011).

[2] One recently suggested workaround involves the use of external photosensors that detect changes in light intensity on the participants' screens to achieve exact stimuli timing and to synchronize event presentation with external physiologic recording devices (Perakakis et al., 2013).

Table 1 Overview of programs to develop experiments

	General				Experiments			Technical			Measures*					
Name	GUI	Free	OSS	Scripting	MultiPl	Inet	Brow	FIO	SV	DC	IH	RT	MT	ET	Platform	Reference
Programs for local experiments with individual participants																
DirectRT	Yes	No	No	Custom	No	No	No	Yes	-	-	-	-	-	-	Win	Reviewed in Stahl (2006)
DMDX	No	Yes	No	Custom	No	No	No	Yes	-	-	-	-	-	-	Win	Forster & Forster (2003)
E-Prime	Yes	No	No	E-Basic	No	No	No	Yes	-	-	-	-	-	-	Win	Reviewed in Stahl (2006)
Experiment Builder	Yes	No	No	Python	No	No	No	Yes	-	-	-	-	-	-	Win	SR Research, Missisauga, ON, Canada
MATLAB Psychophysics Toolbox	No	Yes	Yes	MATLAB	No	No	No	Yes	-	-	-	-	-	-	Win/Lin/Mac	Brainard (1997)
PEBL	No	Yes	Yes	Custom	No	No	No	Yes	-	-	-	-	-	-	Win/Lin/Mac	Mueller (2010)
Presentation	Yes	No	No	Custom	No	No	No	Yes	-	-	-	-	-	-	Win	Neurobehavioral Systems, Albany, CA
PsyScope	Yes	Yes	Yes	Custom	No	No	No	Yes	-	-	-	-	-	-	Mac	Cohen et al. (1993)
PsyToolkit	No	Yes	Yes	Custom	No	No	No	Yes	-	-	-	-	-	-	Lin	Stoet (2010)
PyEPL	No	Yes	Yes	Python	No	No	No	Yes	-	-	-	-	-	-	Lin/Mac	Geller et al. (2007)
SuperLab	Yes	No	No	Custom	No	No	No	Yes	-	-	-	-	-	-	Win	Reviewed in Stahl (2006)
Tscope	No	Yes	Yes	C/C++	No	No	No	Yes	-	-	-	-	-	-	Win	Stevens et al. (2006)
Vision Egg	No	Yes	Yes	Python	No	No	No	Yes	-	-	-	-	-	-	Win/Lin/Mac	Straw (2008 (Straw))
OpenSesame	Yes	Yes	Yes	Python	No	No	No	Yes	-	-	-	-	-	-	Win/Lin/Mac	Mathôt et al. (2012)
PsychoPy	Yes	Yes	Yes	Python	No	No	No	Yes	-	-	-	-	-	-	Win/Lin/Mac	Peirce (2007)
Inquisit	Yes	No	No	Custom	No	Yes	Yes	Yes	-	-	-	-	-	-	Win	Reviewed in Stahl (2006)
Programs for browser-based online experiments																
WEXTOR	Yes	Yes	No	Javascript	No	Yes	Yes	No	-	-	Yes	Yes	No	No	Win/Lin/Mac	Reips & Neuhaus (2002)
jsPsych	No	Yes	Yes	Javascript	No	Yes	Yes	No	-	-	Yes	Yes	No	No	Win/Lin/Mac	de Leeuw (2014)
Programs for interactive studies with multiple players																
BoXS	Yes	Yes	Yes	Custom	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes**	Win/Lin/Mac	Seithe (2012)
zTree	Yes	Yes	No	Custom	Yes	Yes	No	Yes	Yes	Yes	No	limited	No	No	Win	Fischbacher (2007)

Note Information in this table is in parts based on Mathôt, Schreij, and Theeuwes (2012)

GUI graphical user interface, *OSS* open-source software, *MuliPl* interactive multiplayer experiments, *Inet* internet-based experiments, *Brow* browser-based experiments that require no software installation by participants, *FIO* local file access/manipulation, *SV* supervision during multiplayer experiments, *DC* automatic data collection in multiplayer experiments, *IH* input history (choice revision detection), *RT* response time measurement, *MT* mouse tracking, *ET* eye tracking

* Not coded for programs for individual participants, for which these features are usually all available

** Currently implemented for SMI eye-tracker only

scratch. It aims to combine core advantages of z-Tree and the established non-commercial software in that it contains full capabilities for programming interactive studies and possibilities to record process data in an easy to use, platform-

independent open-source programing tool. At the same time, BoXS also allows for implementing standard questionnaires and non-strategic decision-making studies with persons taking part at the laboratory or over the internet according to standard requirements for web research (e.g., Birnbaum, 2004; Reips, 2002; Reips & Birnbaum, 2011).

Key features of the Bonn Experiment System

BoXS is a platform for conducting experiments that is both powerful and easy to use. The architecture of BoXS is based on open software and supports both laboratory and internet experiments. BoXS allows non-interactive experiments and questionnaires as well as interactive experiments including the recording of process measures such as mouse-tracking and eye-tracking. A comprehensive and up-to-date documentation as well as tutorials and examples are provided to guide and assist experimenters new to BoXS.

Platform independence and flexibility

The BoXS client, which is the part of the system the participants interact with, is implemented as a Java applet. Unlike software compiled for a specific system, Java applets run on Windows, Linux, and MacOS, and do not need to be installed prior to being used, which vastly expands the scope of how experiments using the BoXS can be conducted. Without the need to distribute and install client software, laboratory experiments can be set up easier. Furthermore, BoXS also allows participants to attend experiments from home without the need for a local infrastructure for the experimenter and the "autorun" feature allows them to start experiments at any time. BoXS makes it easier to investigate interactions of participants that are otherwise difficult to reach simultaneously (e.g., intercultural research, forensic research, research on minorities, etc.) and it allows for conducting interactive web-based experiments with participants from different locations, as is for example required in studies involving workers from Amazon's Mechanical Turk. Finally, BoXS can also be used as a powerful teaching tool as students can use it instantly to create and conduct improvised classroom experiments.[3]

Open-source issues/availability

BoXS is released as open-source software (OSS) in accordance with the terms of the GNU Public License (GPL). This license grants every user the right to download, examine, and modify the software in any way as long as the resulting derivative work is attributed to the original author and is itself released as OSS under the GPL.

Releasing the BoXS as OSS offers two major advantages for the users. Many experiments include the recording of sensitive information such as individual preferences, attitudes, and personal information (i.e., income, health). This can raise questions about the system's reliability in handling and protecting this data. In BoXS the experimenter can examine and retrace the software routines herself in order to verify their correctness and appropriateness.

The second major advantage of releasing BoXS as OSS is that it enables experimenters to expand upon and adapt BoXS. If the need for specific extensions arises, for example an element in the user interface that supports a specific type of hardware device, the experimenter can implement such a change herself without relying on the support of the system's main developers. Furthermore, upon creating such an extension, the experimenter can submit and publish this extension to make it available for other researchers in need of a similar functionality.

Feature set

The basic feature set of BoXS allows experimenters to implement most popular economic and many psychological experiments.

The display method can present basic text, static and animated images, as well as tables and enumerations (Fig. 1). This functionality can be enhanced by using web services like Google Image Chart,[4] which allow the experimenter to include complex graphs and diagrams based on experiment data.

The available user interface components include text fields, buttons, radio buttons, check boxes, and slider bars (cf. Reips & Funke, 2008). It is possible to either use two-step input procedures in which data for each page are entered and confirmed with a button press or one-step input procedures in which the program directly reacts on hitting pre-defined keys. The experimenter can define arbitrary restrictions on the participant's input (e.g., ensuring that the person's age is above 10 years), and provide specific error messages to assist participants in correcting mistakes. The elements of the user interface can be laid out both automatically and manually, in which case the experimenter can specify the exact position of each element.

BoXS provides an "input history" and also supports mouse-tracking and eye-tracking. The "input history" records every input submitted by each participant with a time-stamp, which allows the experimenter to detect instances in which choices were revised and to analyze the time participants spent pondering over each item contained in one display. An integrated mouse-tracking routine records every movement of each participant's mouse. Finally, eye-tracker support including a basic

[3] The only requirement for BoXS to run is the free Java Runtime Environment (JRE). Since many other programs require JRE as well, it is already installed on most computers. If it is not installed, most internet browsers automatically notify the user and guide her through the necessary installation process, which requires about 2–3 min.

[4] See https://developers.google.com/chart/image/

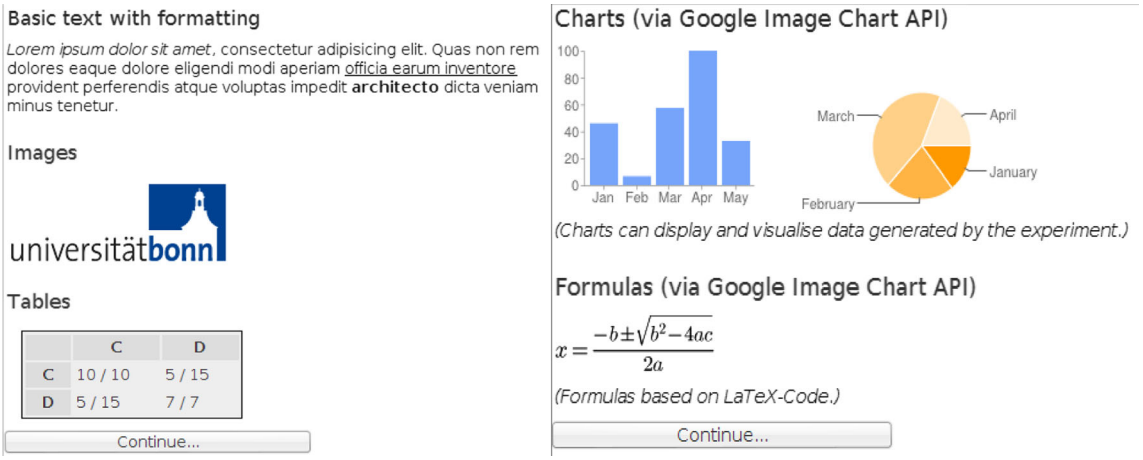

Fig. 1 Examples of a display method

calibration procedure and the recording of the tracking data during the experiment is available.[5] All three data sources are automatically synchronized with the respective participants' system clocks and can therefore be jointly analyzed after the experiment.

On the basic programming level BoXS provides both basic algebra (+, -, /, *, % (modulo)), logical operators (<, >, <=, >=, ==, !=), common mathematical functions (exp, log, sin, cos, tan, round), uniform and gaussian random number generators, as well as methods for conditional (if) and repeated execution (for, while). The data within a BoXS experiment is stored in local, group-specific, or global variables that can be accessed and modified at any point in the experiment.

Finally, four different kinds of matching of individuals for interactive studies are provided, including alphabetical matching, stranger matching, perfect stranger matching, and manual matching, which can be used to implement arbitrary matching patterns. For more detailed information and example programs please refer to the online documentation.[6]

Documentation and getting started

Experiments for BoXS are written in a custom programming language (BoXSPL), which is designed to be easy to learn and master. Novice users of BoXS can get started by watching a tutorial video and examining examples that demonstrate how basic experiments like questionnaires or public good games can be implemented. An up-to-date online documentation that also provides detailed information on each feature is maintained on the official homepage. The "frequently asked questions"-section (FAQ) of the website and a mailing list provide further support for novice and advanced users.

[5] As of now, the iView X by SensoMotoric Instruments (SMI) is supported and a solution for eye-trackers by Interactive Minds GmbH is available on request. Other eye-trackers can be supported in the future, given that they provide a network based API, which is most often the case.

[6] See http://boxs.uni-bonn.de/documentation/index.html

What is inside BoXS?

From a technical perspective, BoXS consists of a server process, which manages participants and experiments, and a client applet, which provides the graphic user interface (GUI). The server is a central process to which the experimenter and all participants connect. The server parses and executes programs, handles the necessary communication between participants, and collects and relays all data generated by the experiment to the experimenter. The strict separation of experimenter client and server offers the advantage that a crash on an experimenter's computer does not affect running experiments. Equally, neither the crash of a participant's computer nor participants accidentally closing the browser affects the running program. Participants can usually reconnect to and resume the experiment by restarting their web browser (or computer) and reopening the given experiment address. They then can regularly continue the study at the point they left it. The default BoXS server is located in Bonn (Germany) and can be used without further set-up. Experimenters are, however, free to set up their own custom servers, which may be advisable for environments in which no sufficiently reliable internet connection is available.

The client applet connects to the server and provides a GUI for participants and experimenters (Fig. 2). Based on the respective login information, the server decides whether the client applet takes on the role of an experimenter client or a participant client. The experimenter client provides an interface that consists of a code editor, in which the experimenter can program and manage the experiment code. The experimenter client also allows the experimenter to track the status of all connected participants as well as all variables generated in the running experiment. The appearance of the subject client is based on the running experiment and can include text, graphics, and input elements (i.e., radio buttons, slider bars).

Multiple instances of applets can be executed alongside (for example, one experimenter client and two subject clients) which is useful for implementing and testing new experiments. If the connection between any client and the server is disrupted, the

running experiment can usually be resumed without further issues by simply restarting the respective applet. A detailed explanation of the inner workings of the BoXS would exceed the scope of this paper. For more information, please refer to Seithe (2012). The latest version of the source code and files for custom server installations of BoXS can be downloaded from the official webpage (see footnote 6) and are additionally provided at https://github.com/plattenschieber/BoXSServer

Applications and examples

Previous versions of BoXS have already been successfully used for local experiments investigating strategic decision making in social dilemmas within one laboratory (e.g., Glöckner & Hilbig, 2012) but also for experiments including interactions between laboratories in different cities (Dorrough, Glöckner, Hellmann, & Ebert, 2015). In the currently released version 1.0, some further features have been implemented that allow BoXS to be applied to a wider range of research questions in psychology and to make handling easier.

How easy it is to realize various experiments with a few lines of code in BoXS will be demonstrated in the following three examples. In the first two examples we present a basic version of the program in the main text and an advanced version in the appendix. The basic versions already include all functional parts; the advanced versions demonstrate how additional useful features can be added to the programs. Due to formatting restrictions, the printed versions of the provided example programs contain unintended line breaks. Executable versions are available at https://github.com/plattenschieber/BoXSServer/tree/master/examples

Example 1: risky-choice study

The first example involves a classic paradigm from individual decision making. Participants are repeatedly asked to choose between risky prospects. The provided risky choice program makes use of the Google Charts API to present gambles (Fig. 3). In the basic version provided below, ten randomly generated choices between two-outcome gambles are presented. The generated charts are displayed until one gamble is selected by pressing a key. The decision time for each task is measured. The first part (lines 2–8) generates the instruction screen, the main part (lines 9–53) presents ten randomly generated gambles in a loop, and the last part (lines 54–55) finishes the experiment.[7] In the main part, probabilities and outcomes of gambles are generated and stored in variables (lines 11–19), graphic gamble presentation is prepared (lines 23–25), and gambles are presented to the participants until one of two keys is pressed (lines 27–34). Finally, variables and gamble information are stored (lines 36–45) and the program waits for a key response before continuing to the last screen (lines 47–52).

```
// **** BEGIN Code Example 1a ****
while(_lastkey != 32)
{
  display("<CENTER><H2>Choice Study</H2></CENTER><br><br>In the following you are asked to select the gamble you prefer.<br>Each gamble consists of two outcomes which realize with some probability that is visually displayed")
  display("<br><br><H4>Please press space to continue</H4>")
  recordKeys()
  waitTime(50)
}
for (i = 0; i < 11; i = i + 1)
{
  // randomly generate gambles and store in variables to shorten the google chart api call
  Lp1 = round(randomUniform()*100)/100
  Lo1 = randomUniformInteger(-10,10)
  Lp2 = 1-Lp1
  Lo2 = randomUniformInteger(-10,10)
  Rp1 = round(randomUniform()*100)/100
  Ro1 = randomUniformInteger(-10,10)
  Rp2 = 1-Rp1
  Ro2 = randomUniformInteger(-10,10)
  // measure the time when gamble is shown
  beginTime = time
```

[7] Numbering of lines can best be followed by in the online versions of the example programs that are provided at https://github.com/plattenschieber/BoXSServer/tree/master/examples Example programs can be tested by copying these code files in an experimenter client at the standard BoXS server: http://boxs.uni-bonn.de/start.html

```
  // collect images from google api
  tdPic1 = "< img src = 'http://chart.apis.google.com/chart? cht = p&chd = t:" + Lp1 + "," + Lp2 +
"&chs = 260x150&chl = " + Lo1 + "%20EUR|" + Lo2 + "%20EUR' > "
  tdPic2 = "< img src = 'http://chart.apis.google.com/chart? cht = p&chd = t:" + Rp1 + "," + Rp2 +
"&chs = 260x150&chl = " + Ro1 + "%20EUR|" + Ro2 + "%20EUR' > "
  // gather players decision
  while (_lastkey ! = 67 && _lastkey ! = 77)
  {
  // present pies to player
  display("Gamble A: " + tdPic1 + "Gamble B: " + tdPic2 )
  display("< br > <H4 > Please press 'C' to select Gamble A and 'M' to select Gamble B </H4 > ")
  recordKeys()
  waitTime(50)
  }
  // save gambles, choice and time
  results[ i][ 0] = _lastkey
  results[ i][ 1] = time - beginTime
  results[ i][ 2] = Lp1
  results[ i][ 3] = Lo1
  results[ i][ 4] = Lp2
  results[ i][ 5] = Lo2
  results[ i][ 6] = Rp1
  results[ i][ 7] = Ro1
  results[ i][ 8] = Rp2
  results[ i][ 9] = Ro2
  while(_lastkey ! = 32)
  {
  display("< br > <br > Please press Space to continue")
  recordKeys()
  waitTime(50)
  }
}
display("< br > <br > Thank you very much for taking part in the study!")
waitForExperimenter("Finish")
// **** END Code Example 1a ****
```

The full-fletched version of Example 1b in the appendix uses predefined gambles that are hard coded into an array structure, which is then accessed randomly via an index array "shuffle." It furthermore uses a table display and adds the selection of one gamble at the end which is played out and incentivized (see Fig. 3, lower part).

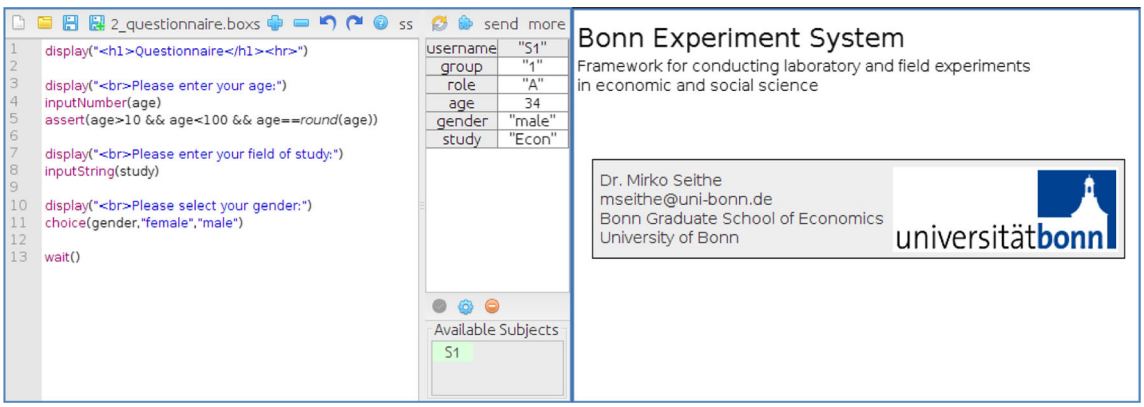

Fig. 2 Experimenter client (left) and participant client (right)

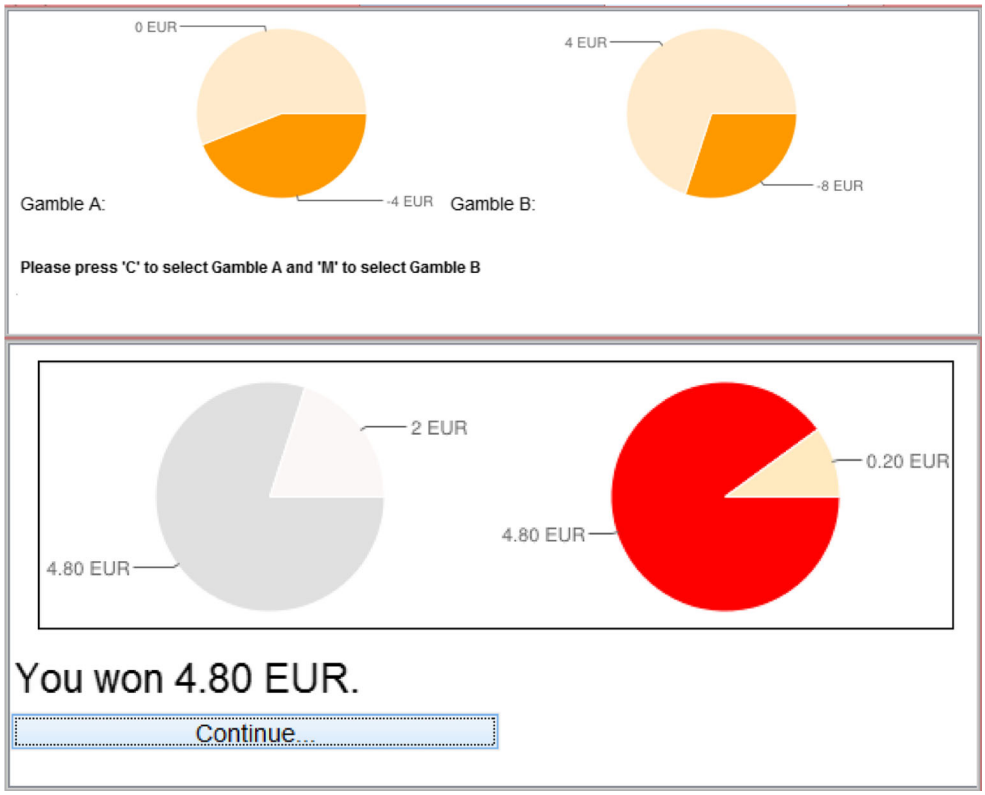

Fig. 3 Display in risky-choice experiment in a basic (top) and an extended (bottom) version

Example 2: Repeated prisoners' dilemma with eye-tracking

The second example shows a basic version of a prisoners' dilemma in which participants play ten rounds and dyads are reshuffled every round (Fig. 4). Using the eye-tracking feature, which currently supports the eye-trackers provided by SensoMotoric Instruments (SMI), consists of five simple commands: (1) initializing the connection: eyetrackerInitialise ("localhost", 4444,5555), (2) performing the calibration: eyetrackerCalibrate(), (3) starting the recording: eyetrackerStart(120, filename), (4) sending triggers to indicate events in BoXS: eyetrackerTrigger("Trigger Name") (see extended Example 2b), and (5) stopping the eyetracker: eyetrackerStop(). Eye-tracking data can be recorded for all participants by removing the comment markers (//) in front of the eye-tracker commands.

```
// **** BEGIN Code Example 2a ****
for (gameCounter = 1; gameCounter < =10; gameCounter = gameCounter + 1)
{
  matchStranger(A,B)
  i = i + 1
  if (i==1)
  {
  display("< H2 > Prisoners Dilemma Game </H2 > <br > In this study in 10 subsequent rounds you and a randomly chosen other person will simultaneously decide whether to cooperate or to defect. < br > ")
  display("If you both decide to cooperate, you will both earn 200 point. If you both decide to defect you will both earn 100 points. <br > ")
  display("If one of you decides to defect and the other to cooperate, the defecting person receives 300 points and the cooperating person receives 50 points. <br > ")
  display("In each round you are randomly matched with a new person.")
  display("The total points your earned are paid to you according to the following exchange rate: 500 point = 1 Euro.")
  wait()
```

```
  //eyetrackerInitialise("localhost",4444,5555)
  display("The calibration begins after pushing the wait button < br > Please hold still and
follow the points.")
  wait()
  //eyetrackerCalibrate()
  display("Thank you - the calibration was successful")
  //eyetrackerStart(60,"PrisonersDilemma_" + username + "_" + gameCounter + ".dat")
  wait()
 }
 clear()
 choice(action,"Cooperate", "Defect")
 waitForPlayers("Continue…")
 // calculate Payoffs
 if (A.action == "Cooperate" && B.action == "Cooperate")
 {
  A.win = 200
  B.win = 200
 }
  if (A.action == "Cooperate" && B.action == "Defect")
 {
  A.win = 50
  B.win = 300
 }
  if (A.action == "Defect" && B.action == "Cooperate")
 {
  A.win = 300
  B.win = 50
 }
  if (A.action == "Defect" && B.action == "Defect")
 {
  A.win = 100
  B.win = 100
 }
 display("You choose " + action + " < br > Your opponent choose " + opponent.action + " < br >
Your roundwin equals " + win)
 // save win and actions into the global array 'vp'
 totalwin = totalwin + win
 *.*.vp[ username][ 1][ i] = action
 *.*.vp[ username][ 2][ i] = win
 wait()
 if (i==10)
 {
 display("You won " + totalwin + " Points. This equals " + totalwin/200 + "EUR.")
 // eyetrackerStop()
 waitForExperimenter("FINISH")
 }
 matchDone()
}
// **** END Code Example 2a ****
```

An extended example with an optimized display that also allows eye-tracking for a single player (e.g., in case one has only eye-trackers in the laboratory) and involves the sending of detailed time triggers is provided in the appendix as Example 2b.

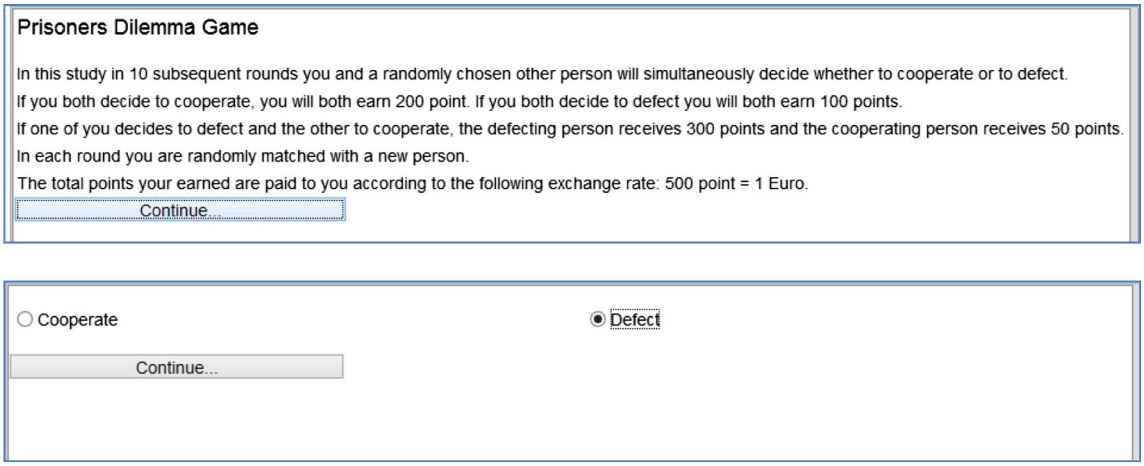

Fig. 4 Display in the Prisoners' Dilemma (Example 2a)

Example 3: Repeated public good with mood induction and eye-tracking or mouse-tracking

Example 3 shows a public good game in which four persons can contribute to maximize their shared profit (Fig. 5). The example includes a mood induction by video (converted to gif), full eye-tracker usage (if comment markers in front of eyetracker commands are removed), usage of mouse-tracking (if comment markers in front of 'enableMouseTracking()' are removed), a graphic visualization of the contributions and reshuffling of groups after every round (stranger matching) in only 43 lines of code (without comments and newlines).

The interaction goes over ten rounds implemented by a loop over gameCounter. Participants are randomly rematched every round in new groups of four (lines 5 and 48). Before the first round starts, several issues are taken care of (lines 7–23): the eye-tracker is turned on (lines 10–11) and calibrated (lines 15–17), the instruction is shown (lines 12–14), and the mood induction is carried out by playing a video (lines 20–22). Participants' contributions to the public good are recorded and checked (lines 27–30) and graphic feedback concerning the contributions of all four players is presented (lines 31–32). The round payoff and the total payoff for all participants are calculated (line 34–35) and displayed (lines 40–41) and all variables are stored (lines 36–39). After the last round, the final payoff is displayed and the eye-tracker is stopped (lines 42–47).

```
// **** BEGIN Code Example 3 ****
for(gameCounter = 1; gameCounter < =10; gameCounter = gameCounter + 1)
{
  // match 4 players each round
  matchStranger(A,B,C,D)
  i = i + 1
  if (i==1)
  {
    // enable mouse tracking
    // enableMouseTracking()
    // eyetrackerInitialise("localhost",4444,5555)
    display(" < H2 > Public Good Example</H2 > <br > Please read the paper instructions in front of you before starting the experiment.")
    display("The calibration begins after pushing the wait button. < br > Please hold still and follow the points.")
    wait()
    //eyetrackerCalibrate()
    display("Thank you - the calibration was successful")
    //eyetrackerStart(60,"PublicGoodGame" + username + "_" + i + ".dat")
    wait()
```

```
    // mood induction video, showed for 10 sec.
    display("Please watch the following movie!")
    display("<img src='http://www.wonderoftech.com/wp-content/uploads/2012/12/fireplace.gif'>")
    waitTime(10000)
  }
  beginTime = time
  clear()
  // read in contribution and check for consistency
  display("<br>Please choose the amount you would like to contribute:")
  inputNumber(z)
  assert(z>=0 && z<=10 && z==round(z))
  waitForPlayers("Continue…")
  display("You are player " + role + ". You contributed <b>" + z + " Talers</b>.<br>The total amount donated is <b>" + sum(z) + " Talers</b>.")
  display("<img src='http://chart.apis.google.com/chart?cht=p&chd=t:" + A.z + "," + B.z + "," + C.z + "," + D.z + "&chs=200x150&chl=A|B|C|D'>")
  // Calculate payoff and save decision, decision time in ms and payoff
  payoff = 10 - z + 0.4*sum(z)
  totalPayoff = totalPayoff + payoff
  results[i][0] = z
  results[i][1] = time - begintime
  results[i][2] = payoff
  results[i][3] = totalPayoff
  display("Your payoff is <b>" + payoff + " Talers</b>")
  wait()
  if(i==10)
  {
      display("Your total payoff is <b>" + totalPayoff + " Talers</b>")
      // eyetrackerStop()
      waitForExperimenter("FINISH")
  }
      matchDone()
}
// **** END Code Example 3
```

Fig. 5 Display of the public good experiment from Code Example 3; left: mood induction by video, middle: contribution screen, right: feedback about contributions of others

Validation of time measurement and presentation timing

To assure exact measurements of response times as well as synchronization with external devices such as eye-trackers, BoXS reads time directly from the system clock of the client PC. Measurement errors and delays in stimulus presentation due to slow connections with the server can therefore be avoided.

We used the time measure of a high precision eye-tracker from Sensomotoric Instruments SMI RED250 to validate time measurement in BoXS. Specifically, we wrote a program that sends time stamps from BoXS to the eye-tracker and to record the timing of these stamps in BoXS as well. The times recorded by the eyetracker and in BoXS on average deviated by 0.2 ms and the maximum difference observed was 4.2 ms. Hence, time measurement in BoXS can be considered to be very precise and valid. At the same time, this validation shows that measurement in BoXS and in external eye-trackers is very well synchronized.

In a second validation study, we investigated the precision of stimulus presentation in BoXS. Stimulus presentation is naturally limited by the refresh rate of the monitor in that stimuli cannot be presented shorter than the time which the monitor needs to overwrite the previous display. Many of the programs for local experiments with individual participants such as PsychoPy or DirectRT have the ability to control stimuli on the level of single display frames. BoXS does not have this capability, so that lower precision in timing can be expected. In the validation study we presented stimuli (numbers vs. pictures from IAPS (Lang, Bradley, & Cuthbert, 2005)) for durations of 20, 50, 100, 200, 500, and 1,000 ms each 200 times using the waitTime function of BoXS. We measured the (validated) time after each presentation and calculated time differences to infer presentation time. The differences were on average 1.6 ms larger than the programmed durations (SD = 1 ms) with a range of deviations between 0 and 16 ms. Note that our measurement includes time for stimulus presentation plus processing some lines of code so that a positive deviation could be expected. The time for presenting IAPS pictures was 0.16 ms longer than the time for presenting numbers, thus complexity of the stimuli did not make a noticeable difference for presentation time. Hence, overall the waitTime function can be considered to work very precisely for timing stimuli.

Although the second validation study shows that stimulus timing in BoXS is very precise this does not necessarily mean that the true presentation duration is precise as well since the stimuli might be timed for presentation but still not be shown for the correct time on the screen. Most importantly, for very short presentation times the interplay between hardware and software could lead to the fact that stimuli are not presented at all. We conducted a third validation study in which we presented stimuli for 50, 80, 100, and 200 ms with an intertrial interval of 1,000 ms on a laptop with a 50-Hz display. Visual inspection showed that only very few stimuli were actually presented at 50 ms, some stimuli were omitted for 80 ms, and essentially all stimuli were shown for 100 ms and beyond. The presentation time was noticeably longer for 200 ms as compared to 100 ms, although the precise actual presentation duration on-screen could not be measured further. Given this result, BoXS cannot be used for priming below 100 ms, such as subliminal priming. Furthermore, taking 50 ms as the threshold for the time necessary to present stimuli on the screen, arguably screen presentation time could be considered to be precise ±50 ms.

Challenges and limitations

Complex applications and interactivity

One further challenge for web-based experimental systems such as BoXS is that they should be able to implement designs that require fairly intensive, real-time interaction between many persons. Could, for example, a fully functional double auction with 16 participants be programmed in this software? And can BoXS handle second-by-second interaction between players in real time? In general, the BoXS server can handle such complex interactions with many subjects at the same time, including quasi real-time cross-subject interactions. While the BoXS was already successfully tested in experiments involving many simultaneous users, it is hard to specify the system's exact limits as they rely on numerous factors including the performance of the involved computers, the latency of the network, and the type and complexity of the experiment. Hence, sufficient pre-testing of such complex experimental programs is required.

Furthermore, it would be useful for some kinds of interactive studies if a web-based experimental system could have the functionality to handle interactions in which participants use input devices (e.g., sliders), which instantly causes real-time changes to a graph. As introduced above, in BoXS graphs and diagrams can be included in an experiment and shown to the subjects by using the Google Charts API. This API can create different types of diagrams based on data provided by the program. Examples for this would be a pie chart visualizing risky prospects (Fig. 3) or the contributions in a public goods game (Fig. 5) or a line graph showing the change in contributions in a repeated trust game. While this approach does provide many useful diagram types, it is limited by the fact that the subjects cannot interact with these diagrams. Compared to the BoXS, z-Tree does provide a greater selection of interactive user interface elements, and if these are necessary in an experiment it might be advisable to implement the study using z-Tree.

Mobile devices

For some types of studies (e.g., tracking of behavioral changes over longer periods) it becomes increasingly important that studies can also be run on mobile devices such as "smartphones" and tablets. As introduced above, the BoXS client requires the Java browser plugin in order to work properly. While this plugin can be easily installed on most computers, it is unfortunately available for neither iOS nor Android, which are the most popular smartphone- and tablet-operating systems. Hence, in its current implementation BoXS does not allow implementation of such studies since it does not run on most smartphones and tablets. For such studies Javascript-based experimental systems such as jsPsych might be an alternative, since Javascript is usually available on smartphones and tablets too (although they do not allow implementation of interactions between persons yet).

In the long run and given sufficient support, the BoXS could be extended to support such mobile devices. Even experiments in which both desktop and smartphone users participate at the same time might be possible. From a technical perspective there are two approaches implementing mobile device support. The first approach would be to program platform-specific clients for mobile-operating systems. Given its large user base and the fact that it is itself based on Java, the Android system seems to be a particularly appealing choice for this approach. Programming such a platform-specific client would allow most of the BoXS features to work on mobile devices. In addition, it would be possible to create device-specific user interface elements which could adapt to smaller screen sizes and other device-specific properties. The alternative approach would be to write a purely HTML-based client for simple experiments which would work on PCs and mobile devices alike. The advantage of this approach would be that it would work on all mobile operating systems. Unfortunately, the set of features which can be implemented using pure HTML is limited. Especially advanced features like time-based experiments or experiments involving real-time interaction may be impossible to implement using this approach. One potential solution would be to also include elements of Javascript, potentially combining positive features of jsPsych and BoXS.

Discussion

Bonn eXperimental System (BoXS) allows experimenters to develop and run interactive and non-interactive studies in and outside the laboratory. It is particularly useful for conducting research with persons at different locations interacting over the internet, which is for example necessary in studies involving participants from Amazon's Mechanical Turk. BoXS combines core features of z-Tree and specialized programs for developing non-interactive studies in that it provides simple routines for connecting, matching, and supervising participants and recording process measures. BoXS is platform-independent and completely browser-based to allow for large-scale interactive experiments around the world. The source-code of BoXS is provided under an open license which allows experimenters to extend the program if necessary and to verify data handling for data protection reasons. Finally, BoXS provides a relatively simple programing language which is easy to learn and to apply.

BoXS has been successfully used in recent research and could provide a viable tool for the growing number of researchers interested in cognitive processes and interactive behavior. BoXS can also be applied for conducting simple non-interactive studies. However, one has to acknowledge that the tailoring of BoXS for allowing browser-based interactive studies comes at a cost. Some functionalities concerning data manipulation and data input (i.e., loading stimuli files and saving data files) are currently limited due to Java's inherently strict security policies. Workarounds, such as hard coding of stimuli (see Example 1b, appendix), can be implemented, but they might not be very elegant. Other programs such as PsychoPy and OpenSesame might be preferable to BoXS for running non-interactive cognitive studies that involve comprehensive data manipulation. Furthermore, we could show in validation studies that time measurement in BoXS is very precise and synchronize with other external devices such as eye-trackers by relying on the same high precision system time at the user client. The internal timing of presentation durations is precise as well, but it was found that the interplay between software and hardware makes priming with stimuli presentation durations below 100 ms basically impossible in BoXS. At 100 ms and above we estimate display time to be precise±50 ms, which should be sufficient for many applications. Also BoXS has the functionality to implement complex real-time interactions of many individuals, although it has to be acknowledged that it has not been intensely tested for such purposes yet. Two further important limitations are that BoXS in the current version cannot handle real-time changes in interactive graphs and that it does not run on most mobile devices since it requires Java.

The currently released version 1.0 of BoXS contains many crucial features that allow investigating cognitive processes in interactive studies. BoXS is aimed to make conducting such research easier. BoXS will allow answering research questions that were hard to address with previously established tools such as hypotheses concerning changes in attention and arousal in the course of making interactive decisions as well as many others. BoXS is also constantly improved and extended to include additional features and support for external devices of other manufacturers to allow broadening the range of experimental applications and making the program attractive for more potential users. One important project for future

developments is also the extension of the BoXS to run on mobile devices. We hope that BoXS can contribute to the cumulative development of knowledge by supporting researchers with implementing their innovative research ideas also concerning complex topics involving cognition and interactive behavior.

Appendix

Example 1b: "RiskyChoiceFull.boxs"

```
// Number of Risky Choices
nGambles = 5
// gamblesL/R[ gamble][ outcome1, outcome2,..][ p/o]
// first gamble
gamblesL[ 0][ 0][ 0] = 0.85
gamblesL[ 0][ 0][ 1] = 5
gamblesL[ 0][ 1][ 0] = 0.15
gamblesL[ 0][ 1][ 1] = 2.5
gamblesR[ 0][ 0][ 0] = 0.95
gamblesR[ 0][ 0][ 1] = 5
gamblesR[ 0][ 1][ 0] = 0.05
gamblesR[ 0][ 1][ 1] = 0.35
// second gamble
gamblesL[ 1][ 0][ 0] = 0.8
gamblesL[ 1][ 0][ 1] = 4.8
gamblesL[ 1][ 1][ 0] = 0.2
gamblesL[ 1][ 1][ 1] = 2
gamblesR[ 1][ 0][ 0] = 0.9
gamblesR[ 1][ 0][ 1] = 4.8
gamblesR[ 1][ 1][ 0] = 0.1
gamblesR[ 1][ 1][ 1] = 0.2
// third gamble
gamblesL[ 2][ 0][ 0] = 0.95
gamblesL[ 2][ 0][ 1] = 4.8
gamblesL[ 2][ 1][ 0] = 0.05
gamblesL[ 2][ 1][ 1] = 0.6
gamblesR[ 2][ 0][ 0] = 0.9
gamblesR[ 2][ 0][ 1] = 4.8
gamblesR[ 2][ 1][ 0] = 0.1
gamblesR[ 2][ 1][ 1] = 2.4
// fourth gamble
gamblesL[ 3][ 0][ 0] = 0.05
gamblesL[ 3][ 0][ 1] = 4.8
gamblesL[ 3][ 1][ 0] = 0.95
gamblesL[ 3][ 1][ 1] = 0.6
gamblesR[ 3][ 0][ 0] = 0.1
gamblesR[ 3][ 0][ 1] = 2.6
gamblesR[ 3][ 1][ 0] = 0.9
gamblesR[ 3][ 1][ 1] = 0.6
// fifth gamble
gamblesL[ 4][ 0][ 0] = 0.5
```

```
gamblesL[ 4][ 0][ 1] = 2.2
gamblesL[ 4][ 1][ 0] = 0.5
gamblesL[ 4][ 1][ 1] = 2
gamblesR[ 4][ 0][ 0] = 0.5
gamblesR[ 4][ 0][ 1] = 4.8
gamblesR[ 4][ 1][ 0] = 0.5
gamblesR[ 4][ 1][ 1] = 2
// first part of the presentation table
tOpen = " < table border = '0' cellspacing = '0' cellpadding = '0' width = '520' > "
trOpen = " < tr > "
trClose = " </tr > "
tClose = "</table > "
td1 = " < td align = 'center' > Gamble A </td > "
td2 = " < td align = 'center' > Gamble B </td > "
td3 = " < td align = 'center' > Press C for this Gamble</td > "
td4 = " < td align = 'center' > Press M for this Gamble</td > "
// write some numbers into an array ;)
for (i = 0; i < nGambles; i = i + 1)
{
        shuffle[ i] = i
}
// and shuffle them a lot of times e.g. 100x
for (i = 0; i < 100; i = i + 1)
{
        for (j = 0; j < nGambles; j = j + 1)
        {
              // shuffle with your neighbour in half of the cases
              if (randomUniform() < =0.5)
              {
                        // save number to be shuffled and …
                        tmp = shuffle[ j]
                        // shuffle upward
                        if (j + 1 ! = nGambles)
                        {
                                shuffle[ j] = shuffle[ j + 1]
                                shuffle[ j + 1] = tmp
                        }
                        // shuffle downward
                        if (j + 1 == nGambles)
                        {
                                shuffle[ j] = shuffle[ j-1]
                                shuffle[ j-1] = tmp
                        }
              }
        }
}
randChoosenGamble = randomUniformInteger(0,nGambles-1)
// instructions
while(_lastkey ! = 32)
{
        display(" < CENTER > <H2 > Choice Study </H2 > </CENTER > ")
```

```
        display("In the following you are asked to select the gamble you prefer.")
        display("Each gamble consists of two outcomes which realize with some probability
that is visually displayed in pie charts.")
        display("Press 'C' to select the left gamble, and press 'M' to select the right gamble.")
        display("One of the decisions will be randomly selected and paid.")
        display("<br> If you have understood the instructions please press space to continue.")
        recordKeys()
        waitTime(500)
}
for (i = 0; i < nGambles; i = i + 1)
{
        // shortening the google chart api call
        Lp1 = gamblesL[ shuffle[ i]][ 0][ 0]
        Lo1 = gamblesL[ shuffle[ i]][ 0][ 1]
        Lp2 = gamblesL[ shuffle[ i]][ 1][ 0]
        Lo2 = gamblesL[ shuffle[ i]][ 1][ 1]
        Rp1 = gamblesR[ shuffle[ i]][ 0][ 0]
        Ro1 = gamblesR[ shuffle[ i]][ 0][ 1]
        Rp2 = gamblesR[ shuffle[ i]][ 1][ 0]
        Ro2 = gamblesR[ shuffle[ i]][ 1][ 1]
        // measure the time when gamble is shown
        beginTime = time
        // collect images from google api
        tdPic1 = " < td > <img src = 'http://chart.apis.google.com/chart?cht = p&chd =
t:" + Lp1 + "," + Lp2 + "&chs = 260x150&chl = " + Lo1 + "%20EUR|" + Lo2 + "%20EUR' > </td > "
        tdPic2 = " < td > <img src = 'http://chart.apis.google.com/chart?cht = p&chd =
t:" + Rp1 + "," + Rp2 + "&chs = 260x150&chl = " + Ro1 + "%20EUR|" + Ro2 + "%20EUR' > </td > "
        // and place them into a nice table
        table = tOpen + trOpen + td1 + td2 + trClose + trOpen + tdPic1 + tdPic2 + trClose +
tClose
        // gather players decision
        while (_lastkey ! = 67 && _lastkey ! = 77)
        {
              // present pies to player
              display(table)
              recordKeys()
              waitTime(50)
        }
        // save players decision (for presented gamble - shuffle[ i] )
        results[ shuffle[ i]][ 0] = _lastkey
        // length of players decision making in ms (for presented gamble)
        results[ shuffle[ i]][ 1] = time - beginTime
        // save the round in which player saw shuffle[ i]
        results[ shuffle[ i]][ 2] = i
        // choose a gamble by chance
        if (shuffle[ i] == randChoosenGamble)
        {
              if(_lastkey == 67)
              {
              decision = "A"
              }
              if(_lastkey == 77)
              {
```

```
                decision = "B"
                }
        }
        // this one is usually not needed, but we have to change _lastkey, before entering the loop again
        while(_lastkey ! = 32)
        {
                display("Please press Space to continue")
                recordKeys()
                waitTime(50)
        }
}
// present choosen gamble to user and let him start the dice rolling
// shortening the google chart api call
Lp1 = gamblesL[ randChoosenGamble][ 0][ 0]
Lo1 = gamblesL[ randChoosenGamble][ 0][ 1]
Lp2 = gamblesL[ randChoosenGamble][ 1][ 0]
Lo2 = gamblesL[ randChoosenGamble][ 1][ 1]
Rp1 = gamblesR[ randChoosenGamble][ 0][ 0]
Ro1 = gamblesR[ randChoosenGamble][ 0][ 1]
Rp2 = gamblesR[ randChoosenGamble][ 1][ 0]
Ro2 = gamblesR[ randChoosenGamble][ 1][ 1]
// measure the time when gamble is shown TODO: Check where to put timeMeasurements
beginTime = time
// collect images from google api
  tdPic1 = " < td > <img src = 'http://chart.apis.google.com/chart? cht = p&chd = t:" + Lp1 + "," + Lp2 + "&chs = 260x150&chl = " + Lo1 + "%20EUR|" + Lo2 + "%20EUR' > </td > "
tdPic2 = " < td > <img src = 'http://chart.apis.google.com/chart? cht = p&chd = t:" + Rp1 + "," + Rp2 + "&chs = 260x150&chl = " + Ro1 + "%20EUR|" + Ro2 + "%20EUR' > </td > "
// and place them into a nice table
table = tOpen + trOpen + td1 + td2 + trClose + trOpen + tdPic1 + tdPic2 + trClose + tClose
while(_lastkey ! = 10)
{
        display(table)
        display(" < h1 align = 'center' > This is the randomly choosen gamble</h1 > ")
        display("Your decision was: Gamble " + decision)
        display("Please press Enter to roll the dice…")
        recordKeys()
        waitTime(500)
}
// roll the dice
rand = randomUniform()
if (decision == "A")
{
        if (rand < = gamblesL[ randChoosenGamble][ 0][ 0] )
        {
                outcome = gamblesL[ randChoosenGamble][ 0][ 1]
                tdPic1 = " < td > <img src = 'http://chart.apis.google.com/chart? cht = p&chd = t:" + Lp1 + "," + Lp2 + "&chco = FF0000,FFEAC0&chs = 260x150&chl = " + Lo1 + "%20EUR|" + Lo2 + "%20EUR' > </td > "
        }
        if (rand > gamblesL[ randChoosenGamble][ 0][ 0] )
        {
```

```
            outcome = gamblesL[ randChoosenGamble][ 1][ 1]
            tdPic1 = "<td><img src='http://chart.apis.google.com/
chart?cht=p&chd=t:" + Lp1 + "," + Lp2 + "&chco=FF9900,FF0000&chs=260x150&chl=" + Lo1 +
"%20EUR|" + Lo2 + "%20EUR'></td>"
      }
  tdPic2 = "<td><img src='http://chart.apis.google.com/chart?cht=p&chd=t:" + Rp1 + ",
" + Rp2 + "&chco=E0E0E0,FCF7F7&chs=260x150&chl=" + Ro1 + "%20EUR|" + Ro2 + "%20EUR'></
td>"
}
if (decision == "B")
{
      tdPic1 = "<td><img src='http://chart.apis.google.com/chart?cht=p&chd=t:" +
Lp1 + "," + Lp2 + "&chco=E0E0E0,FCF7F7&chs=260x150&chl=" + Lo1 + "%20EUR|" + Lo2 +
"%20EUR'></td>"
      if (rand<= gamblesR[ randChoosenGamble][ 0][ 0] )
      {
            outcome = gamblesR[ randChoosenGamble][ 0][ 1]
            tdPic2 = "<td><img src='http://chart.apis.google.com/
chart?cht=p&chd=t:" + Rp1 + "," + Rp2 + "&chco=FF0000,FFEAC0&chs=260x150&chl=" + Ro1 +
"%20EUR|" + Ro2 + "%20EUR'></td>"
      }
      if (rand > gamblesR[ randChoosenGamble][ 0][ 0] )
      {
            outcome = gamblesR[ randChoosenGamble][ 1][ 1]
            tdPic2 = "<td><img src='http://chart.apis.google.com/
chart?cht=p&chd=t:" + Rp1 + "," + Rp2 + "&chco=FF9900,FF0000&chs=260x150&chl=" + Ro1 +
"%20EUR|" + Ro2 + "%20EUR'></td>"
      }
}
// present lottery win again with a chart (color changed)
table = tOpen + trOpen + tdPic1 + tdPic2 + trClose + tClose
display(table)
display("<h1 align='center'>You won " + outcome + " EUR.</h1>")
wait()
```

Example 2b: “PrisonersDilemmaFull.boxs”

```
rounds = 10
// play the game
for (i=1; i<=rounds; i=i+1)
{
  matchStranger(A,B)
    // if a variable is never used before, it is by default 0 …
  if (isFirstRound == 0)
  {
    isFirstRound = 1
    // new style for tables
    style("body{ padding: 0px; font-size: 12px; } h1{ font-size: 130 %; margin-top:
0px; margin-bottom: 3px; font-weight: normal;} h2{ font-size: 115 %; margin-top: 0px;
margin-bottom: 3px; font-weight: normal; } table{ border-collapse:collapse;} th{ text-
align:center; background:#fff4c6; color:#333; padding:8px 16px 8px 8px; border-
```

```
right:1px solid #fff; border-bottom:1px solid #fff;} td{ text-align:center; color:#333; padding:8px; border-right:1px solid #f3f0e4; border-bottom:1px solid #f3f0e4;}")
    // Initialize variables
    //C D
    //   _______ _______
    //  |   |   |
    //C |200/200| 50/300|
    //  |_______|_______|
    //  |   |   |
    //D  |300/50 |100/100|
    //  |_______|_______|
    // WinMatrix[ Zeile][ Spalte][ Spieler]
    WinMatrix[ 0][ 0][ 0] = 200
    WinMatrix[ 0][ 0][ 1] = 200
    WinMatrix[ 0][ 1][ 0] = 50
    WinMatrix[ 0][ 1][ 1] = 300
    WinMatrix[ 1][ 0][ 0] = 300
    WinMatrix[ 1][ 0][ 1] = 50
    WinMatrix[ 1][ 1][ 0] = 100
    WinMatrix[ 1][ 1][ 1] = 100
    // overall win
    totalwin = 0
    // number of rounds per player combination
    rounds = 10
    // Welcome screen
    display("<H2>Prisoners Dilemma Game </H2><br>In this study in 10 subsequent rounds you and a randomly chosen other person will simultaneously decide whether to cooperate or to defect.<br>")>  display("If you both decide to cooperate, you will both earn 200 point. If you both decide to defect you will both earn 100 points. <br>")
    display("If one of you decides to defect and the other to cooperate, the defecting person receives 300 points and the cooperating person receives 50 points. <br>")
    display("These outcomes will be shown to you in a table format on screen each time you make a decision. <br>")
    display("In each round you are randomly matched with a new person.")
    display("The total points your earned are paid to you according to the following exchange rate: 500 point = 1 Euro.")
    //Subject Code
    display("<br> <br>Please indicate subject code:")
    inputString(sub_number)
    // EYETRACKER ONLY FOR S1
    if (username == "S1")
    {
       // INIT EYETRACKER
       // eyetrackerInitialise("localhost",4444,5555)
       display("The calibration begins after pushing the continue button. <br>Please hold still and follow the points.")
       wait()
       // CALIBRATE EYETRACKER AND START TRACKING
       // eyetrackerCalibrate()
       display("Thank you - the calibration was successful")
```

```
      filename = "PrisonersDilemma_" + sub_umber + ".dat"
      // eyetrackerStart(60,filename)
      // SEND TRIGGER
      // eyetrackerTrigger("Calibration Done")
  }
  // Wait for everybody being ready
  waitForPlayers("continue")
}
      table = "<table>"
      table = table + " <thead>"
      table = table + " <tr>"
      table = table + " <th> </th>"
      table = table + " <th>Cooperate</th>"
      table = table + " <th>Defect</th>"
      table = table + " </tr>"
      table = table + " </thead>"
      table = table + " <tbody>"
      table = table + " <tr>"
      table = table + " <th>Cooperate</th>"
      table = table + " <td>" + WinMatrix[ 0][ 0][ 0] + " / " + WinMatrix[ 0][ 0][ 1] + "</td>"
      table = table + " <td>" + WinMatrix[ 0][ 1][ 0] + " / " + WinMatrix[ 0][ 1][ 1] + "</td>"
      table = table + " </tr>"
      table = table + " <tr>"
      table = table + " <th>Defect</th>"
      table = table + " <td>" + WinMatrix[ 1][ 0][ 0] + " / " + WinMatrix[ 1][ 0][ 1] + "</td>"
      table = table + " <td>" + WinMatrix[ 1][ 1][ 0] + " / " + WinMatrix[ 1][ 1][ 1] + "</td>"
      table = table + " </tr>"
      table = table + " <tr>"
      table = table + " </tbody>"
      table = table + "</table>"
         display(table)
         clear()
      choice(action,"Cooperate", "Defect")
         if (username == "S1")
         {
         // SEND TRIGGER for start of contribution screen
         // eyetrackerTrigger("contribution start for round: " + i )
         }
      waitForPlayers("Weiter")
  // Calculate Payoff
  if (A.action == "Cooperate")
  {
      if(B.action == "Cooperate")
      {
          A.win = WinMatrix[ 0][ 0][ 0]
          B.win = WinMatrix[ 0][ 0][ 1]
      }
      if(B.action == "Defect")
      {
```

```
            A.win = WinMatrix[ 0][ 1][ 0]
            B.win = WinMatrix[ 0][ 1][ 1]
        }
    }
    if (A.action == "Defect")
    {
        if(B.action == "Cooperate")
        {
            A.win = WinMatrix[ 1][ 0][ 0]
            B.win = WinMatrix[ 1][ 0][ 1]
        }
        if(B.action == "Defect")
        {
            A.win = WinMatrix[ 1][ 1][ 0]
            B.win = WinMatrix[ 1][ 1][ 1]
        }
    }
    // present actions and payoff
    display("You have chosen: '" + action + "'." )
    if (role == "B")
    {
        display("Your opponent has chosen: '" + A.action + "'." )
    }
    if (role == "A")
    {
        display("Your opponent has chosen: '" + B.action + "'." )
    }
    display("Your income in this round is: " + win)
    totalwin = totalwin + win
    display("Your total income so far is: " + totalwin)
    win[ i] = win
    action[ i] = action
    if (username == "S1")
    {
    // SEND TRIGGER for start of feedback screen - note that screen display is build at the wait
command
    // eyetrackerTrigger("feedback start for round: " + i )
    }
    wait()
    matchDone()
}
if (username=="S1")
{
    // eyetrackerTrigger("finish after round: " + i )
    // eyetrackerStop()
}
// End of game reached
display("<h1>The study is finished. Please click continue. </h1>")
wait()
```

```
display("Your total earning is: " + totalwin + "Points")
display("This equal: " + totalwin / 200 + " Euro")
display("<br><br> Please contact the experimenter.")
waitForPlayers("Finish")
```

References

Birnbaum, M. H. (2004). Human research and data collection via the internet. *Annual Review of Psychology, 55,* 803–832.

Brainard, D. H. (1997). The psychophysics toolbox. *Spatial Vision, 10,* 433–436.

Chaudhuri, A. (2011). Sustaining cooperation in laboratory public goods experiments: a selective survey of the literature. *Experimental Economics, 14*(1), 47–83.

Cohen, J., MacWhinney, B., Flatt, M., & Provost, J. (1993). PsyScope: An interactive graphic system for designing and controlling experiments in the psychology laboratory using Macintosh computers. *Behavior Research Methods, Instruments, & Computers, 25*(2), 257–271.

Dawes, R. M. (1980). Social dilemmas. *Annual Review of Psychology, 31,* 169–193.

de Leeuw, J. R. (2014). jsPsych: A JavaScript library for creating behavioral experiments in a Web browser. *Behavior Research Methods*, 1–12. doi: 10.3758/s13428-014-0458-y

Dorrough, A. R., Glöckner, A., Hellmann, D. M., & Ebert, I. (2015). The development of ingroup favoritism in repeated social dilemmas. *Frontiers in Psychology, 6*. doi: 10.3389/fpsyg.2015.00476

Fiedler, S., Glöckner, A., Nicklisch, A., & Dickert, S. (2013). Social Value Orientation and information search in social dilemmas: An eye-tracking analysis. *Organizational Behavior and Human Decision Processes, 120*(2), 272–284.

Fiedler, S., Glöckner, A., & Nicklisch, A. (2012). The influence of social value orientation on information processing in repeated voluntary contribution mechanism games: an eye-tracking analysis. In A. Innocenti & A. Sirigu (Eds.), *Neuroscience and the economics of decision making* (Vol. 5, pp. 21–53). New York: Routledge.

Fischbacher, U. (2007). z-Tree: Zurich toolbox for ready-made economic experiments. *Experimental Economics, 10*(2), 171–178.

Forster, K. I., & Forster, J. C. (2003). DMDX: A Windows display program with millisecond accuracy. *Behavior Research Methods, Instruments, & Computers, 35*(1), 116–124.

Geller, A. S., Schleifer, I. K., Sederberg, P. B., Jacobs, J., & Kahana, N. J. (2007). PyEPL: A cross-platform experiment-programming library. *Behavior Research Methods, 39*(4), 950–958.

Glöckner, A., & Hilbig, B. (2012). Risk is relative: Risk aversion yields cooperation rather than defection in cooperation-friendly environments. *Psychonomic Bulletin & Review, 19*(3), 546–553.

Hardin, G. (1968). The tragedy of the commons. *Science, 162,* 1243–1248.

Komorita, S. S., & Parks, C. D. (1995). Interpersonal-relations - mixed-motive interaction. *Annual Review of Psychology, 46,* 183–207.

Lang, P. J., Bradley, M. M., & Cuthbert, B. N. (2005). *International affective picture system (IAPS): Affective ratings of pictures and instruction manual*: NIMH, Center for the Study of Emotion & Attention.

Mason, W., & Suri, S. (2012). Conducting behavioral research on Amazon's Mechanical Turk. *Behavior Research Methods, 44*(1), 1–23.

Mathôt, S., Schreij, D., & Theeuwes, J. (2012). OpenSesame: An open-source, graphical experiment builder for the social sciences. *Behavior Research Methods, 44*(2), 314–324.

Mueller, S. T. (2010). The PEBL manual. Retrieved from http://pebl.sourceforge.net/

Peirce, J. W. (2007). PsychoPy—psychophysics software in Python. *Journal of Neuroscience Methods, 162*(1), 8–13.

Perakakis, P., Guinot, J. V., Conde, A., Jaber-López, T., García-Gallego, A., & Georgantzis, N. (2013). *A technical note on the precise timing of behavioral events in economic experiments*. Working paper of the Economics Department, Universitat Jaume I, Castellón (Spain).

Rapoport, A., & Chammah, A. M. (1965). *Prisoner's dilemma*. Ann Arbor, MI: University of Michigan Press.

Reips, U.-D. (2002). Standards for internet-based experimenting. *Experimental Psychology, 49*(4), 243–256.

Reips, U.-D., & Birnbaum, M. H. (2011). Behavioral research and data collection via the internet. In R. W. Proctor & K.-P. L. Vu (Eds.), *The handbook of human factors in web design* (pp. 563–585). Mahwah: Erlbaum.

Reips, U.-D., & Funke, F. (2008). Interval-level measurement with visual analogue scales in Internet-based research: VAS Generator. *Behavior Research Methods, 40*(3), 699–704.

Reips, U.-D., & Neuhaus, C. (2002). WEXTOR: A Web-based tool for generating and visualizing experimental designs and procedures. *Behavior Research Methods, Instruments, & Computers, 34*(2), 234–240.

Sally, D. (1995). Conversation and cooperation in social dilemmas: A meta-analysis of experiments from 1958 to 1992. *Rationality and Society, 7*(1), 58–92.

Schulte-Mecklenbeck, M., Kuehberger, A., & Ranyard, R. (2011). *A handbook of process tracing methods for decision research: a critical review and user's guide*. New York: Psychology Press.

Seithe, M. (2012). Introducing the Bonn Experiment System (BoXS). *Bonn ECON Papers, No. 01/2012*.

Stahl, C. (2006). Software for generating psychological experiments. *Experimental Psychology, 53*(3), 218.

Stevens, M., Lammertyn, J., Verbruggen, F., & Vandierendonck, A. (2006). Tscope: A C library for programming cognitive experiments on the MS Windows platform. *Behavior Research Methods, 38*(2), 280–286.

Stoet, G. (2010). PsyToolkit: A software package for programming psychological experiments using Linux. *Behavior Research Methods, 42*(4), 1096–1104.

Straw, A. D. (2008). Vision egg: an open-source library for realtime visual stimulus generation. *Frontiers in neuroinformatics, 2*.

Van Lange, P. A. M., Joireman, J., Parks, C. D., & Van Dijk, E. (2013). The psychology of social dilemmas: A review. *Organizational Behavior and Human Decision Processes, 120*(2), 125–141.

von Bastian, C. C., Locher, A., & Ruflin, M. (2013). Tatool: A Java-based open-source programming framework for psychological studies. *Behavior Research Methods, 45*(1), 108–115.