Exp Econ (2007) 10:171–178
DOI 10.1007/s10683-006-9159-4

z-Tree: Zurich toolbox for ready-made economic experiments

Urs Fischbacher

Received: 10 May 2006 / Revised: 30 November 2006 /
Accepted: 11 December 2006 Published online: 7 February 2007

Abstract z-Tree (Zurich Toolbox for Ready-made Economic Experiments) is a software for developing and conducting economic experiments. The software is stable and allows programming almost any kind of experiments in a short time. In this article, I present the guiding principles behind the software design, its features, and its limitations.

Keywords Experiment · Experimental software

JEL Classification C92

1 Introduction

Computerized experiments play a crucial role in experimental economics, and the programming of experiments has therefore become an important issue in experimental economics. However, programming economic experiments used to be time-consuming and difficult. Software packages for a particular family of experiments (such as public goods experiments, oligopoly experiments, or double auctions; see Plott (1991), for example) were developed in a first generation. Professional programmers built these packages—based on libraries that implemented the necessary core functionality. The most widely used library of this type was Ratimage developed in Bonn by Abbink and Sadrieh (1995). Some experimentalists now base their experiments on general programming environments such as Visual Basic or Internet technology (for details see, e.g., Kirchkamp, 2004). However, all of these approaches require considerable effort and significant programming experience for developing a new experiment. z-Tree

U. Fischbacher (✉)
Institute for Empirical Research in Economics, Blümlisalpstrasse 10,
CH-8006 Zurich
e-mail: fiba@iew.unizh.ch

(Zurich Toolbox for Readymade Economic Experiments, see Fischbacher, 1999) is a software package for developing and carrying out economic experiments. The language used to define the experiments is simple and compact, meaning that experiments can be developed quickly, and programming experience is not necessary, though useful.

z-Tree is flexible both with respect to the logic of interaction and the visual representation, allowing the simple programming of normal form games, extensive form games, double auctions, or clock auctions, for example. We began the development of the software about 10 years ago, and have continually added new features. A new version of the software will be released soon, offering major improvements.[1] The most important addition concerns graphics. First, the new version allows the display of pictures. Second, graphical representation of data is now feasible. Possible applications are game trees, pie charts for lotteries, price paths in an auction, links in a network, or the visualization of a market structure. These graphical elements are particularly interesting because they are not restricted to passive use, i.e., as display elements. They can also be related to input. It is therefore easy to implement selection: for example, the subject can input a decision by clicking a graphical element. It is also possible to drag elements, for instance to define relations between elements. The new version includes also a chatting function: Previously, subject input in z-Tree had to be translated into a number, which implied that only predefined messages could be exchanged. In this new version, a chat function can be integrated into subjects' screens. Finally, some improvements concern the use of z-Tree for neuroeconomic applications. For instance, the timing of screen changes and subjects' input can now be recorded precisely. Furthermore, input from external hardware is now supported. The latter function can be used, for example, to record triggers from fMRI scanners or to allow input from fMRI compatible response boxes.

Some specific software components have to be provided in any experimental software: in particular, communication between computers, screen definition, payoff calculation, and interaction between the subjects. I outline the problems in the design of each component and present the solution used in z-Tree. This paper should also give the reader a flavor of how experiments are defined; I will sketch the programming process for some experiments to do this. However, this article is not intended to give a step-by-step explanation of any particular application. I conclude by presenting strength and limitations of the software and giving an outlook to the future development of z-Tree.

2 Components of an experimental software

Since economic experiments are interactive, a core component of any experimental software is the communication between the subjects' computers. There are two principal communication architectures: the "peer-to-peer" and the "client-server"

[1] The development of z-Tree started in 1995. The first version that was used outside of the lab of Zurich was released in 1998 (version 1.0.1). This article is focused on the new features in version 3. Currently (November, 2006), version 3.0.18 is being tested in the lab of Zurich. As soon as we have ascertained that the version is stable, we will release it. Note that experiments programmed in an earlier version of z-Tree can be used with later versions. Except for small changes it the screen layout, experiments programmed with older versions of z-Tree run identically on newer versions of z-Tree.

architecture. The interacting subjects' computers communicate directly with each other in a peer-to-peer architecture, while subjects in a client-server network interact indirectly via a central computer program, the server. The peer-to-peer architecture is useful when most interaction is bilateral and if the same partners interact with each other. The client-server architecture has clear advantages for economic experiments because even when only two players interact, the interaction partner changes often within an experiment and, in particular, there must be a mechanism to match the partner. Furthermore, fewer network connections are necessary in a client-server architecture, since each subject has only one connection to the server instead of a connection to every other subject. Finally, some global coordination is always necessary in an experiment. For instance, the subjects have to know what kind of game is played. The server is a natural place to set up the experiment, to start it, to control it, and, last but not least, to store the data.

The z-Tree software is implemented as a client-server application with a server application for the experimenter, called z-Tree, and a client application for the subjects, called z-Leaf. An experiment begins by starting the server software. The subjects' computers are then started, and on them, the client application z-Leaf. z-Leaf connects with z-Tree and remains so during the whole session. The server program z-Tree is used for starting "treatments", which are the experiment components programmed in z-Tree. For instance, such a "treatment" can be an ultimatum game, a third party punishment game, or a double auction. When a treatment begins, the program transfers the necessary information about the parameters and the state of the game from z-Tree to the clients (z-Leaf). In the other direction, subjects' input is transferred from the clients to the server. The input is processed at the server and the clients receive messages about how the experiment continues.

The way experimental software handles problems is particularly crucial. Since dozens of interconnected computers are in simultaneous use in an experiment, the crash of a computer or the temporary interruption of a network connection will eventually occur, and it is important that an event like this does not disrupt the whole experiment. Thus, experimental software has to be stable, and it must support recovering the experiment after an incident. Experiments conducted with z-Tree are stable because the delicate parts of the software (in particular networking) are only implemented once in the z-Tree software itself; the programmers of a specific experiment do not have to reinvent it. Furthermore, since they are used in every experiment, they are tested intensively. In addition, if a client crashes—or a subject turns the computer off—it can be restarted; it reconnects with z-Tree and again receives all messages, meaning that it is in the exact state it should be at this time. Thus, the software is also stable with respect to breakdowns of network connections, which occasionally occur when a network cable is slack.

Networking is based on TCP/IP. This means that experiments are not limited to a local area network in a computer lab, but can also be conducted via Internet. The prerequisites for Internet experiments are that the client software z-Leaf is installed and started on the subjects' computers and that z-Tree runs on a computer with an IP address that remains constant during the experiment.

Stability is the main issue for an experimental software. A second priority is that experiments can be developed quickly and easily. How experiments are defined, determines whether this is possible. In the following, I discuss how experiments can be

described in general and how the description is done in z-Tree. I will first discuss some aspects of the screen representation and then the description of the course of action.

Any software for conducting economic experiments has to provide tools both for presenting information and for reading in subjects' input. There are two principle ways for defining how data presentation will appear on subjects' screens: either specifying the exact position of every element on the screen or describing the structure of the screen and letting the computer do the exact positioning. The latter has the advantage that the representation does not depend on screen resolution, and removal or insertion of screen elements is easier because repositioning all the other screen elements is not necessary. We use a mixed approach in z-Tree: the windows[2] are placed by entering numerical values as a percentage of the screen size. The elements within windows are listed without referring to a position and are then placed by the software.

Subjects' input can be made in different formats: numbers and predefined text, for instance, can be entered with the keyboard, and predefined answers can be selected by clicking a radio button (horizontally and vertically arranged). Furthermore, input can be made with sliders and scrollbars. When input with the keyboard is made, range checks are applied in order to keep subjects from making illegal entries.

When thinking about how to represent the course of action in an experiment, one has to take into account that interaction has many different forms: subjects play simultaneous and sequential games; they participate in markets, implemented as posted offer markets, or one-sided or double auctions; they bargain; and they participate in clock auctions. Furthermore, experiments frequently combine institutions freely (e.g. double auction with subsequent effort decision). The provision of software modules for institutions, e.g. a double auction, which can be configured and combined to the actual experiment, might thus seem promising. However, it turns out that the details of an institution change to a large extent from experiment to experiment, implying that software modules implementing institutions would need many parameters. Programming an experiment from scratch can be easier than entering all the necessary parameters for a software module implementing a particular institution.[3] I use a pragmatic approach in z-Tree for solving this problem. Many experiments are very simple and can easily be described as a sequence of stages; this is also reflected in the backbone structure of z-Tree. Subjects go sequentially through so-called "stages". Simultaneous moves are the default, i.e., all subjects enter a stage and make the same decision in the normal case. Sequential decisions are implemented by omitting stages for some of the subjects. These two simple elements permit the definition of any kind of game. Interaction between the subjects is implemented in programs that use data from other subjects. For instance, a program in a public goods game calculates the sum of all contributions in a player's own group.[4]

[2] They are called boxes in z-Tree because they cannot be moved around.

[3] As an anecdote, I wrote an software module for an experimental software implementing a mechanism in public good experiments (Falkinger et al., 2000). Since the researchers wanted to have a lot of flexibility in the information conditions, I had to program a dialog with more than 80 checkboxes. Later, I reprogrammed the experiment in z-Tree. It was quicker to reprogram the experiment from scratch in z-Tree than to program this huge dialog.

[4] A program doing this would look like: SumContribution = sum(same (Group), Contribution);

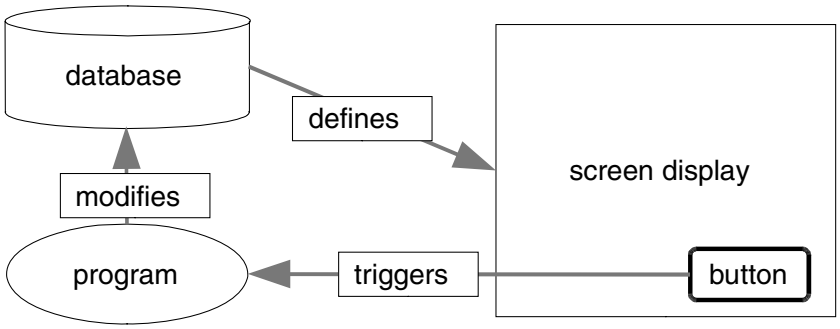

Fig. 1 The data driven action cycle in z-Tree

Institutions like single sided auctions or double auctions cannot be represented sequentially. Thus, they take place within a stage. The data held in z-Tree mediates the interaction in auction stages. Figure 1 shows the general principle: Data in a database describes the current state of an auction; in particular, the screen description refers to this data. During the auction, buttons in the screen trigger programs, which then modify the data in the database, which in turn automatically updates the screen.

In the following paragraph, I will demonstrate this procedure in a concrete auction (see Fig. 2). In this auction, offers are represented as rows in a table. The price "p" of the offer as well as the seller's ("Seller") and buyer's ("Buyer") IDs are specified in each record. These IDs are set to –1 if the offer is open. Therefore, if a seller makes an offer, the price is set to the offer; the seller ID to that of the seller who made the offer, and the buyer ID to –1, reflecting that the offer is open for a buyer to accept. Thus, the box on the screen that displays the "asks" is characterized by the condition Buyer== −1. When a buyer clicks on the "buy" button at the bottom of this box, a program is triggered. In this program, the –1 in the buyer ID is replaced by the ID of the buyer who accepts the offer. This information is transmitted to the subject, and the corresponding offer automatically disappears from the screen because the variable Buyer is no longer equal to –1.

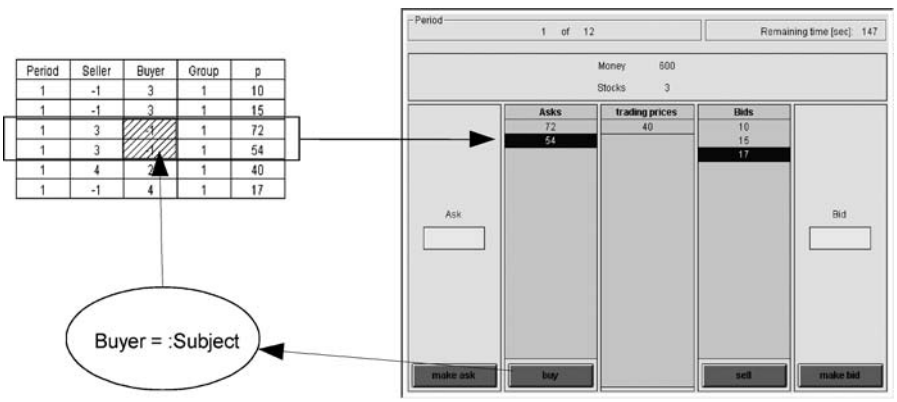

Fig. 2 The action cycle in a double auction

To show graphics, the experiment developer adds a "plot box", a rectangular area on the screen in which a coordinate system can be set up. Graphical elements can then be placed into this area. Points, lines, arrows, circles (ellipses), rectangles, pies, and text are the available predefined graphical elements. The position of these elements can depend on data in the database, i.e. graphics can be made contingent on parameters of the experiment as well as on the subjects' decisions. Furthermore, data in tables, such as price paths, can be displayed in "plot graphs", permitting the display of line plots, box plots, pie charts, scatter plots, etc. It is also possible to make graphical input in plot boxes: the experimenter can define a response to events like clicking in the plot box, clicking a particular object (selecting it), or dragging a particular object. Programming graphical input follows the same scheme as that sketched in Fig. 1: The data in the database determines how graphics are displayed and input elements describe how the data changes. Finally, the changed data is reflected in an updated screen. The graphics features are rather flexible. For example, creating a small graphical editor where polygons can be drawn, moved, and deleted is thus not difficult.

Free form communication is implemented by defining a so-called "chat box" where the experiment developer defines (1) the name of the variable that contains the message entered, (2) which texts are displayed (e.g., a subject should only see the texts within his own group), (3) what other particulars are displayed (e.g., an identification of the sender can precede the message), (4) what additional information is stored together with the message.[5]

Since decisions in the experiment usually forms the basis for remuneration for participation in economic experiments, we provide tools that simplify payment to the subjects. Experiments often also contain non-interactive parts like explanations, control questions, debrief questions, or surveys. They can also be programmed and conducted with z-Tree. However, the presentation options are limited compared to professional questionnaire software, and features like the randomization of the sequence of questions and answers are not implemented.

Finally, how is data prepared for data analysis? z-Tree uses one text file containing tab separated tables to store the data. After the experiment, menu commands in z-Tree allow processing of this file and creating tab separated text files with the variables names as defined in z-Tree in the header. These files can directly be imported into statistics or spreadsheet programs.

3 Conclusions

Software for conducting economic experiments has to achieve the following goals: It has to be stable, the experiments must be rapidly implemented and easy to change, and the software must be sufficiently general to allow the realization of a wide range of experiments. I tried to achieve these goals with z-Tree. The new features in z-Tree 3,

[5] For readers familiar with z-Tree: the chat box combines a contract creation box with a contract list box. The additional information that is stored with the message is entered in a program, which is executed in the record containing the new message. Since free texts cannot be entered in items, we had to find another way to implement free form communication. We will ultimately implement the entry of free text in items—and even more flexibility will be possible.

in particular graphics and free form communication, permit the rapid implementation of a wide range of experiments. The new version also contains some improvements for making z-Tree more usable as a stimulus presenter for neuroeconomic experiments; in particular, it can record the precise time when a state is entered and when the subject makes an action (without delay caused by the network). Further, the time can be recorded when triggers, as used by fMRI scanners, are received. Input from the serial port can be used instead of the keyboard.

What are the limitations of the current version of z-Tree? The most notable limitation concerns the timing of screen presentations. Because the server program z-Tree controls the timing, there are delays before a screen presentation z-Tree triggers is actually displayed at the client program z-Leaf. These delays are in the range of tenths of seconds. This is usually irrelevant for purely economic experiments, but can be a problem for more psychologically oriented experiments. A second limitation concerns external hardware. There is no general interface at present for connecting external hardware, such as psychophysiological measurement devices or devices for presenting other types of stimuli, e.g. olfactory stimuli. Third, although the programming language is sufficiently general for implementing any kind of interaction, there are situations in which programming is unnecessarily complicated, since z-Tree neither knows procedures nor complex data types.

These limitations will be the target of future improvements. One of the planned future steps in the development of z-Tree is thus an extension of the programming language to make programming of more complex experiments easier. Additionally, speed will very likely become more important when experimenters implement more graphically oriented experiments; thus, increasing speed might also be an issue in future versions.

The software z-Tree can be ordered by sending in the license contract that can be downloaded at http://www.iew.uzh.ch/ztree/howtoget.php. It is available free of charge; the only requirement is the citation of this article when experiments conducted with z-Tree are published. A reference manual and a tutorial are available from the web page http://www.iew.uzh.ch/ztree

Acknowledgment I would like to thank Oliver Kirchkamp, Shawn Lamaster, Otto Perdeck, and Martin Strobel for valuable advice in the development of the software; the numerous users who reported bugs and gave comments, in particular Armin Falk, Ernst Fehr, Simon Gächter, Daniel Reding, and Christian Zehnder; the users who helped others in the discussion group; the Humboldt University at Berlin, SFB Mannheim, Viadrina University Frankfurt an der Oder, University of Bari, IHS in Vienna, Harvard Business School, Pompeu Fabra University in Barcelona, GATE Ecully/Lyon, University of Amsterdam, New York University, Sangyo University, Kyoto for inviting me to present the software; Björn Bartling, Ernst Fehr, Sally Gschwend, Daria Knoch, Christian Zehnder, Arthur Schram and two anonymous referees for valuable advice concerning this manuscript; and in particular Stefan Schmid for implementing the free form communication and completing version 3. This paper is part of the Research Priority Program of the University of Zurich on "the Foundations of Human Social Behavior: Altruism versus Egoism."

References

Abbink, K., & Abdolkarim, S. (1995). RatImage—research assistance toolbox for computer-aided human behavior experiments. *SFB 303 Discussion Paper* B-325, University of Bonn.

Falkinger, J., Fehr, E., Gächter, S., & Winter-Ebmer, R. (2000). A simple mechanism for the efficient provision of public goods: Experimental evidence. *American Economic Review*, *90*, 247–264.

Fischbacher, U. (1999). z-Tree. Toolbox for readymade economic experiments. IEW Working paper 21, University of Zurich.

Kirchkamp, O. (2004). WWW experiments for economists, a technical introduction. http://www.kirchkamp.de/research/webexp.html, accessed at March 29, 2006.

Plott, C. R. (1991). A computerized laboratory market system for research support systems for the multiple unit double auction. *Social Science Working Paper* 783. Pasadena: California Inst. Tech.