oTree: The "Bomb" Risk Elicitation Task[☆]

Felix Holzmeister[a,*], Armin Pfurtscheller

[a] *Leopold-Franzens Universität, Department of Banking and Finance, Universitätsstraße 15, Innsbruck, Austria*

Abstract

Among recently suggested elicitation procedures for risk preferences, the "bomb" risk elicitation task (BRET) by Crosetto and Filippin (2013) gained noticeable attention. This article presents a ready-to-use software module for use with *oTree* (Chen et al., 2015) which allows for easily conducting the BRET in numerous different variants, tapping the potential and advantages of state-of-the-art web programming technologies *oTree* is based on. In a user-friendly and straightforward way, predefined and thoroughly documented variables are specified in a separate file, providing full control over different aspects of game play. Furthermore, the application is prepared for multilingual use, providing a convenient way for translation into different languages. Additionally offered *Stata* script files facilitate data processing and the evaluation of individual level risk preferences.

Keywords: oTree, Bomb Risk Elicitation Task, Risk Preferences

1. Introduction

Risk is prevalent in numerous domains of economic decision-making. Investigating the implications of individual attitudes towards risk and uncertainty on different aspects of economic behavior has been on the agenda of experimental scholars for several decades. A still growing body of literature proposes an extensive set of tools and methods for properly measuring risk attitudes, commonly

[☆] The software application described here is free of charge and licensed under the MIT open source license with a citation requirement. Any use of the software – whether as a whole or in parts – implies the acceptance of the license agreement available in the folder for download. Demo-versions of several task variants are available at bret-demo.herokuapp.com. The application is available for download at the website of Felix Holzmeister (www.holzmeister.biz).

[*] Corresponding author

Email address: `felix.holzmeister@uibk.ac.at` (Felix Holzmeister)

URL: `www.holzmeister.biz` (Felix Holzmeister) used in many fields of economic research. Among recently suggested elicitation procedures for risk preferences, the "bomb" risk elicitation task (BRET) put forward by Crosetto and Filippin (2013) gained noticeable attention. As an intuitively comprehensible elicitation procedure providing a fine-grained measure of individual level risk attitudes, the BRET is an up-and-coming alternative to well-established methods commonly applied in experimental research.

This article presents a ready-to-use software module for use with *oTree* (Chen et al., 2015) which allows for easily conducting the BRET in different variants proposed by the original authors as well as some natural extensions. As an open-source, object-oriented environment, *oTree* provides a platform-independent software solution deployable on any device such as smartphones or tablet computers. By this means, *oTree* enables the implementation of experiments in the field, the laboratory, or online, as well as any combination thereof. As a ready-made *oTree* application, our software module for conducting the BRET utilizes all the advantages of *oTree*, implying straightforward setup and usage, state-of-the-art graphical display, seamless integration into existing programs, and enhanced flexibility.

2. The Bomb Risk Elicitation Task

The bomb risk elicitation task (BRET) is a visual (real-time) elicitation method for risk preferences requiring subjects to decide on how many out of a matrix containing $m \times n$ boxes to collect, one of which hides a bomb. Each box collected pays off the same amount of money (γ) such that potential earnings increase linearly. However, earnings are zero if the bomb is contained in one of the collected boxes. By this means, for some given m, n, and γ, the BRET elicits consistent decisions in $(mn + 1)$ lotteries – fully described in terms of outcomes and probabilities – by a single parameter, the number of boxes collected (Crosetto and Filippin, 2013).

Subjects' decisions can be formalized as the choice of Lottery L_k, where k denotes the number of boxes collected and m and n refer to the number of rows and columns, respectively, such that

$$L_k = \begin{cases} 0 & \text{with prob. } p = {}^{k}/_{mn} \\ \gamma k & \text{with prob. } p = {}^{(mn-k)}/_{mn}. \end{cases} \tag{1}$$

The expected value of any lottery L_k is $E(L_k) = \gamma(k - {}^{k^2}/_{mn})$ which is trivially equal to zero for $k = 0$ and $k = mn$ and attains a maximum at $k = {}^{mn}/_2$. Assuming a power utility function of the form $u(x) = x^r$, a subject maximizes his or her utility by stopping the collecting process after $k^* = mn \cdot {}^{r}/_{(1+r)}$ boxes.

Compared to other commonly used risk elicitation methods such as multiple choice lists between pairs of lotteries with varying probabilities (Hey and Orme, 1994; Holt and Laury, 2002), multiple decisions between a fixed lottery and varying certain payments (Dohmen et al., 2010; Abdellaoui et al., 2011), or single choice problems of how much to allocate between a safe and a risky asset (Gneezy and Potters, 1997; Charness and Gneezy, 2010), the BRET captivates by its simple and intuitive design, requiring minimal numeracy skills, and by allowing for a precise elicitation of individual-level risk preferences. By reconciling simplicity and fine-grained measurements of risk preferences, the BRET is well-placed in the trade-off between comprehensibleness and precision discussed by Dave et al. (2010) and Charness et al. (2013). Moreover, the BRET neither suffers from the impact of loss aversion – as it does not provide any endogenous reference point outcomes could be compared to – nor from inconsistent decisions as it entails a unique choice (Crosetto and Filippin, 2016, forthcoming).

Software modules for conducting the BRET are available for download on Paolo Crosetto's webpage, either as a *z-Tree* version (Fischbacher, 2007) or as a stand-alone *Python* application. However, while the former can only be run under *Windows* operating systems and might be subject to network problems for some treatment variations, the latter is a local client application requiring that *Python* – together with a graphical user interface library – is installed on any computer the task is intended to run on. Furthermore, both applications are restricted to specific pre-determined treatments only.

Our application alleviates any of these drawbacks by utilizing state-of-the-art web programming technologies and adds further advantages, especially increased flexibility, straightforward setup of the task and unrestricted combinations of any variations described below. As a ready-to-use *oTree* app (Chen et al., 2015), our module taps the full potential of *oTree* and, thereby, is qualified to be run on any operating system and any device – be that a smartphone, a tablet, or a desktop computer – in a state-of-the-art and graphically appealing manner. The only requirement for running the task is an arbitrary web browser being installed and *Javascript* being enabled.[1]

3. Setup and Usage

The BRET software module is programmed in *Javascript* using the *AngularJS* framework by *Google* and is seamlessly embedded within a stand-alone *oTree* application. Therefore, the app can be used in any experiment conducted with the *oTree* framework by copying the app folder into *oTree*'s *Django* project directory and altering the session configurations in *oTree*'s `settings.py` file.

The application allows experimenters to implement the BRET in several different variants. In a user-friendly and straightforward way, predefined and thoroughly documented variables are specified in a separate file (`setup.py`) at the root of the application directory. The file `setup.py` consists of the `Constants` class of the *oTree* app and contains several variables to be specified in order to determine game play. Since the app is programmed as a standard *oTree* application, the range and scope of existing variables can easily be altered or extended by custom-designed functions.

The entire application is prepared for multilingual use utilizing *Django*'s `i18n` internationalization routines. Instructions on how to apply the translation features are included in the folder for download.

3.1. General Specifications and Appearance

Several configurations in the file `setup.py` allow for defining general task properties.. The decimal field `box_value` determines the (potential) payoff for each box collected. The location of the bomb is determined randomly as soon as the task is loaded. While the original experiment by Crosetto and Filippin (2013) consists of a 10×10 grid of boxes – underpinned by the argument that probabilities are most easily recognized given a total of 100 boxes – our application allows for conducting the task with any arbitrary none-prime number of boxes by simply specifying the number of rows and columns (`num_rows` and `num_cols` in `setup.py`, respectively).[2]

[1]Note that ad blockers or other browser-integrated filter mechanisms restricting the functionality of *JavaScript* may prevent the app to load correctly.

[2]In addition, the size of the boxes to be displayed can be determined by setting the width and height of the boxes in pixels. This allows the researcher to ensure that the task can be displayed correctly on any device such as tablets or smartphones.

As for any standard *oTree* application, the field num_rounds can be assigned any positive integer to determine the number of rounds to be played. For the case of num_rounds larger than one, the variable random_payoff may be set to either True or False in order to define whether one round is randomly chosen to determine the task's payoff or whether the final payoff amounts to the sum of the payoffs of all rounds played.

The sequence of views to be displayed can be scheduled by three specifications. The variable instructions determines whether a separate page containing the instructions of the task shall be displayed before the task is rendered.[3] To allow for conducting the BRET as a side experiment before other tasks in the experimental session without distorting monetary incentives, the application allows running the task both without the possibility to disclose the collected boxes and without a separate view summarizing the results. If the variable feedback in setup.py is set to False the 'Solve' button will not be displayed such that subjects will not be informed about whether the bomb was collected or not. Setting the variable results to False will configure the software not to include the template Results.html, summarizing the results after the task has been finished. In order to enable access to variables of interest at the end of the experiment to inform subjects about their earnings, all variables are stored as 'globals' (session.vars['label']) with the same variable labeling. That is, any information from the task is accessible at any time within a session.

3.2. Settings Determining Gameplay

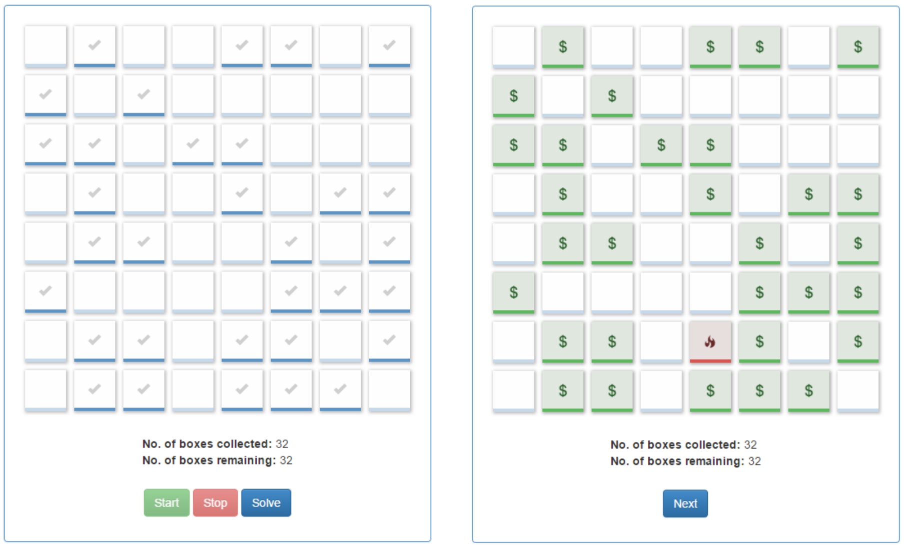
As suggested by the authors in the original article, the elicitation task may be implemented either dynamically or statically. By setting the variable dynamic in setup.py to True or False, boxes are either dynamically collected with some time interval to be specified or statically collected by subjects entering the number of boxes (or clicking on the boxes), respectively.

If dynamic is set to True, the application allows to specify two more properties of game play. First, the time interval between the collections of individual boxes by the computer (in seconds) can be defined by assigning the desired value to the variable time_interval. Second, the variable random can be set

[3]Please note that the instructions included in the template Instructions.html in the download folder only serve exemplary purposes and need to be adapted appropriately for use with different settings of the software module.

to `True` or `False` in order to specify whether boxes are collected in a random order or row-by-row starting with the box in the top left corner. Figure 1, by way of example, shows screenshots of the dynamic random version of the task after stopping the collection process (left panel) and after boxes were disclosed (right panel).

Figure 1: Bomb risk elicitation task (`num_rows = 8`, `num_cols = 8`, `dynamic = True`, `random = True`, and `feedback = True`) after hitting the 'Stop' button (left panel) and after hitting the 'Solve' button (right panel).



No. of boxes collected: 32
No. of boxes remaining: 32

No. of boxes collected: 32
No. of boxes remaining: 32

If `dynamic` is set to `False`, two additional variants of game play are available by setting `devils_game` to `True` or `False`. If `devils_game` is set to `False`, the task is akin to the static treatment as suggested by Crosetto and Filippin (2013). That is, subjects are asked to enter the number of boxes they want to collect into an input field right below the matrix of boxes. In contrast to the static version proposed by the authors, boxes *are* collected in our application. As soon as a subject enters a number into the designated input field (or changes the entry), the respective number of boxes is marked by a tick symbol, providing a graphical representation of probabilities similar to the dynamic version of the task. If `devils_game` is set to `True`, game play is similar to the "devil's game" (Slovic, 1966) requiring subjects to collect each of the desired boxes manually by clicking on them. Additionally, for the devil's game variant, the variable `undoable` can be assigned `True` or `False` to determine whether boxes can be selected only once or whether undoing one's decision should be allowed, i.e. that boxes can be selected and deselected without any limit.

A step-by-step instruction of how to install and set up the task, including detailed descriptions and explanations for every single variable, and how to make use of the internationalization routines is part of the download repository.

4. Data Output (Variables Stored)

Regardless of the specifications in `setup.py`, the app stores four variables (in addition to any variables pre-defined by *oTree*) whenever hitting the 'Next' button after the task has been accomplished: (i) `bomb` is an integer field taking a value of 0 or 1, depending on whether the bomb was among the collected boxes or not, (ii) `boxes_collected` is an integer field containing the number of boxes collected until a subject decides to stop the task by hitting the respective button, (iii) `bomb_location` is a string field determining the row and column of the bomb, and (iv) `boxes_scheme` provides a string consisting of all boxes (indexed by rows and columns) which have been collected. Additionally, all relevant variables are stored as 'globals' in *oTree*'s `session.vars['label']` in order to allow to access any data at any time within a session after the task has been completed.

5. Additional Material

Apart from the *oTree* application itself, the folder for download contains three *Stata* script files. If the task is considered to be run with a higher or lower number of boxes than 100, the first file may be of use as it provides iterative solutions for the minimum, maximum, and average value of the coefficient r in a standard power utility function of the form $u(x) = x^r$ for any number of boxes collected ($k \in [1, 2, \ldots, mn-1]$) given the number of rows (m) and columns (n). A second file loads the standard output file provided by *oTree* and assigns the minimum, maximum, and average values of r to each participant depending on the number of boxes the subject decided to collect. A third file allows analysis of the collection scheme which might be of interest in the case of the devil's game version of the task.

Acknowledgments

We thank Paolo Crosetto, Antonio Filippin, Jürgen Huber, Michael Kirchler, Stefan Palan, Matthias Stefan and an anonymous referee for helpful comments on the software module, the documentation, and the manuscript.

References

Abdellaoui, M., Driouchi, A., L'Haridon, O., 2011. Risk aversion elicitation: Reconciling tractability and bias minimization. Theory and Decision 71, 63–80.

Charness, G., Gneezy, U., 2010. Portfolio choice and risk attitudes: An experiment. Economic Inquiry 48 (1), 133–146.

Charness, G., Gneezy, U., Imas, A., 2013. Experimental methods: Eliciting risk preferences. Journal of Economic Behavior & Organization 87, 43–51.

Chen, D. L., Schonger, M., Wickens, C., 2015. otree – an open-source platform for laboratory, online and field experiments. Journal of Behavioral and Experimental Finance 9, 88–97.

Crosetto, P., Filippin, A., 2013. The bomb risk elicitation task. Journal of Risk and Uncertainty 47 (1), 31–65.

Crosetto, P., Filippin, A., 2016, forthcoming. A theoretical and experimental appraisal of four risk elicitation methods. Experimental Economics.

Dave, C., Eckel, C. C., Johnson, C. A., Rojas, C., 2010. Eliciting risk preferences: when is simple better? Journal of Risk and Uncertainty 41 (3), 219–243.

Dohmen, T., Falk, A., Huffman, D., Sunde, U., 2010. Are risk aversion and impatience related to cognitive ability? American Economic Review 100 (3), 1238–1260.

Fischbacher, U., 2007. z-tree: Zurich toolbox for ready-made economic experiments. Experimental Economics 10, 171–178.

Gneezy, U., Potters, J., 1997. An experiment on risk taking and evaluation periods. Quarterly Journal of Economics 112 (2), 631–645.

Hey, J. D., Orme, C., 1994. Investigating generalizations of expected utility theory using experimental data. Econometrica 62 (6), 1291–1326.

Holt, C. A., Laury, S. K., 2002. Risk aversion and incentive effects. American Economic Review 92 (5), 1644–1655.

Slovic, P., 1966. Risk-taking in children: Age and sex differences. Child Developement 37 (1), 169–176.