Documentation

This application allows conducting the Bomb Risk Elicitation Task (BRET) as proposed by Crosetto/Filippin (2013), Journal of Risk and Uncertainty (47): 31-65, as an *oTree* application in numerous different variants by simply altering the documented variables in `setup.py`.

Installation

To install the app to your local *oTree* directory, copy the folder "bret" to your *oTree* Django project and extent the session configurations in your `settings.py` at the root of the *oTree* directory by something like

```
SESSION_CONFIG = [
   ...
   {
      'name': 'bret',
      'display_name': "Bomb Risk Elicitation Task",
      'num_demo_participants': 1,
      'app_sequence': ['bret'],
   },
   ...
]
```

Please note that – as for any other *oTree* application – global settings such as `REAL_WORLD_CURRENCY_CODE`, `USE_POINTS` as well as `SESSION_CONFIG_DEFAULTS` (including `real_world_currency_per_point` and `participation_fee`) have to be specified in *oTree*'s `settings.py` rather than the application itself but do affect the display of currency figures as well as the calculations of payoffs and amounts to pay.

Setup

To set up the task, the only thing to be done is to alter pre-defined variables in the file `setup.py` at the root of the application's directory. Any combination of the variables described below is operable. In this way, all treatment variations proposed by Crosetto/Filippin (2013) as well as several obvious and natural extensions can be easily implemented. The following variables can be specified:

box_value (decimal/currency field):

Specifies the (currency or point) value of each single box collected. If the bomb was among the collected boxes, all earnings are destroyed and the (round's) payoff is equal to zero. If the bomb has not been collected, the (round's) payoff amounts to `box_value` times `boxes_collected`. (Please note that the currency of payoffs displayed to subjects is determined by the global settings of *oTree* in `settings.py`.)

num_rows and **num_cols** (integer fields):

Number of rows and columns, respectively. The total number of boxes in the matrix is determined by `num_rows` times `num_colums`.

box_height and **box_width** (character fields):

Height and width of a single box to be displayed. Measures (in pixel) have to be entered in single quotes including "`px`", for instance `'50px'`. By adjusting `box_height` and `box_width`, the size of the boxes is set in order to fit the matrix to any arbitrary device including smartphones or tablets. (Please note that the matrix layout is responsive, i.e. boxes will break into new rows if they do not fit. In this case, the number or rows and columns displayed may not match the inputs specified.)

num_rounds (Integer field):

Number of rounds to be played.

random_payoff (boolean field):

Defines whether a subject's total payoff is determined by one randomly chosen round (`random_payoff = True`) or whether the total payoff equals the sum of the individual payoffs of all rounds played (`random_payoff = False`).

Instructions (boolean field):

If `instructions = True`, a separate template `Instructions.html` is rendered prior to the task in round one. If `instructions = False`, the task starts immediately (e.g. in case of printed instructions). Please note that the instructions included serve only exemplary purposes and need to be adjusted to your settings in `setup.py`.

`Feedback` (boolean field):

Determines whether subjects get feedback after finishing the task by toggling boxes and showing whether the collected boxes contains the bomb or not. If `feedback = True`, a button “Solve” will be rendered and active after the task was finished by hitting the “Stop” button. By hitting “Solve”, boxes are toggled and each box shows either a dollar sign or a fire symbol (indicating the bomb). If `feedback = False`, no “Solve” button is rendered and no feedback about the task outcome is provided.

`Results` (boolean field):

Determines whether a results page summarizing game outcome is rendered or not. If `results = True`, a separate view `Results.html` containing all relevant information (i.e. number of boxes collected, whether the bomb was among the collected boxes, the location of the bomb, and the payoff) is rendered. If `results = False`, no separate page is displayed. For the case of `num_rounds` larger than one and `results = True`, all rounds’ results are summarized in a table and shown only after all rounds have been finished.

`Dynamic` (boolean field):

Determines whether the task is conducted in its dynamic or static version. If `dynamic = True`, one box per time interval (see below) is collected automatically. The collection process is initiated by subjects hitting the “Start” button and terminated by hitting the “Stop” button at any time. If `dynamic = True`, game play is affected by two more variables, `time_interval` and `random` (as described below).

If `dynamic = False`, boxes are collected by subjects (either by entering a number or clicking on the boxes, see below) rather than in an automated way. Subjects initiate the collection process by entering the respective number or clicking on boxes (no “Start” button) and terminate the task by hitting the “Stop” button at any time. If `dynamic = False`, game play is affected by three more variables, `random`, `devils_game`, and `undoable` (as described below).

`time_interval` (decimal field):

Defines the time interval between single boxes being collected if `dynamic = True`. The base unit is set to seconds, i.e. `time_interval = 0.5` implies an interval of half a second, etc.

`random` (boolean field):

Determines whether boxes are collected randomly or systematically. If `random = True`, boxes are collected randomly based on the Java-Script-implemented Fisher-Yates algorithm. If `random = False`, boxes are collected row-by-row, starting in the top-left corner of the matrix. That is, denoting rows by [A, B, C, …] and columns by [1, 2, …, k], boxes are collected in the sequence {A1, A2, …, Ak, B1, B2, … Bk, ...}. Note that the field `random` affects game play in both cases, `dynamic = True` and `dynamic = False`.

`devils_game` (boolean field):

Determines whether subjects collect boxes by entering the number they want to collect into an input field below the box matrix (`devils_game = False`) or whether boxes are collected by clicking on each individual box (`devils_game = True`).

If `devils_game = False`, any number entered into the input field is immediately collected by marking the respective boxes with a tick symbol. Altering the number in the input field immediately leads to an update of the collection. Note that boxes are collected either randomly or systematically depending on whether `random = True` or `random = False`.

If `devils_game = True`, game play resembles the “devil’s game” (as proposed by Slovic, P. (1966), “Risk-taking in children: Age and sex differences”, Child Development 37(1), pp. 169–176. That is, rather than selecting a subset of a pre-defined collection, subjects choose which boxes to collect by clicking on them.

`undoable` (boolean field):

If `dynamic = False` and `devils_game = True`, the variable undoable allows specifying whether boxes can be selected only once or whether undoing one’s decision is permissible. If `undoable = True`, boxes can be selected and de-selected indefinitely often (by repeatedly clicking on them). If `undoable = False`, boxes can be selected only once, i.e. any decision to choose a particular box cannot be undone.