Documentation

This application implements the bubble game as proposed by Moinas and Pouget (2013) as well as the speculation elicitation task as put forward by Janssen et al. (2018), as an *oTree* application (Chen et al., 2016) in different variants and parameterizations by simply altering the documented variables in `config.py`.

Installation

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

```
SESSION_CONFIG = [
    ...
    {
        'name': 'bg',
        'display_name': 'Bubble Game (Moinas and Pouget, 2013)',
        'num_demo_participants': 3,
        'app_sequence': ['bg'],
    },
    ...
 ]
```

Please note that global settings as `REAL_WORLD_CURRENCY_CODE`, `USE_POINTS`, as well as `SESSION_CONFIG_DEFAULTS` (including `participation_fee` and `real_world_currency_per_point`) are – as for all *oTree* apps – specified in *oTree*'s `settings.py` file rather than in the application itself. Nevertheless, they affect the display of currency figures as well as the calculations of payoffs and amounts to pay.

Setup and Variables

To set up the game or task, respectively, the only thing one has to do is to alter the pre-defined variables in the file `config.py` at the root of the app's directory. In this way, a number of different variations of the game can easily be implemented.

The following variables can be specified in the file `config.py`:

`num_players` (integer field):

Number (`m`) of players in a group with `i = 1, 2, …, m`.

`num_prices` (integer field):

Number (`k`) of prices offered to players with `i = 1, 2, …, n` and `i ≤ k`.

`num_repetitions` (integer field):

Number of repetitions of the game. If `num_repetitions > 1`, then the game will be played the number of times specified in this variable with new prices randomly drawn for each new repetition.

`random_grouping` (boolean field):

If `random_grouping = False`, then players and groups will be ordered according to their time of arrival in the game – that is, the first player entering the game is Player 1, etc. If `random_grouping = True`, then players are randomly assigned to groups and players' position will be randomly determined within a group. For the case of repeated games, `random_grouping` applies stranger matching.

`multiple` (float field):

The price of the asset is exogenously determined and is always a power of `multiple`. With `multiple = 10`, as in the original bubble game (Moinas and Pouget, 2013), prices are of the form 10^n, 10^{n+1}, 10^{n+2}, etc.

`dist_initprices` (string field):

The first price in each sequence, the initial price, is randomly determined by `multiple` to the power of `x` with `x` being randomly drawn from a certain distribution. Setting `dist_initprices` to one of the values `'geometric'`, `'Poisson'`, `'binomial'`, or `'uniform'` lets the distribution from which `x` is drawn to be either a geometric, Poisson, binomial, or discrete uniform distribution, respectively.

`p` (float field):

If `dist_initprices` is either `'geometric'` or `'binomial'`, `p` acts as the corresponding distribution parameter `p`.

`N` (float field):

If `dist_initprices` is either `'Poisson'`, `'binomial'`, or `'uniform'`, `N` acts as the corresponding distribution parameter `n`.

`cap` (float field):

`cap` determines the cap K on the initial price – that is, the initial price P1 can at most be the value determined by cap. If `cap = 0`, then no cap is enforced and P1 can (theoretically) be any power of `multiple`.

`earnings_noaction` (float field):

`earnings_noaction` is a player's payoff if she decides not to buy the asset at the proposed price

`earnings_success` (float field):

`earnings_success` is a player's payoff if she decides to buy the asset and successfully resells it

`strategy_method` (boolean field):

With `strategy_method` (`True` or `False`), one can use either a direct-response method, in which each player is only offered one price, or a strategy method, in which each player is offered each price before his actual position is revealed.

`order` (string field):

`orde` determines the order of the offered prices if several prices are offered to a player. The variable can take the values `'ascending'`, `'descending'`, or `'random'`. Note that random order is only possible for `strategy_method = True`; if you want to have players at random positions, set `random_grouping = True`.

`one_choice_per_page` (Boolean field):

When using the strategy method (see above), `one_choice_per_page` (either `True` or `False`) determines whether all offered prices are shown on separate pages (`True`) or on one single page (`False`).

`buttons` (Boolean field):

If `one_choice_per_page = True`, then one can choose whether players make decisions via radio buttons (`False`) or via buttons (`True`) by setting the variable buttons to either `False` or `True`, respectively.

`enforce_consistency` (Boolean field):

If `one_choice_per_page = False`, the experimenter can enforce consistent decisions by setting `enforce_consistency = True` – that is, if a player buys at a certain price, she is forced to buy at all lower prices; if a player declines to buy at a certain price, she is forced to also decline to buy at any higher price. Note that this feature requires JavaScript.

`instructions` (Boolean field):

`instructions` (`True` or `False`) determines whether a separate, prepared template of experimental instructions (Instructions.html) is rendered prior to the game. Note that the prepared instructions only serve as an example and need to be adapted appropriately for use with different settings.

`num_instrprices` (integer field):

`num_instrprices` determines how many possible initial prices and their corresponding probabilities (given dist_initprices, `p`, and `N`, respectively) are shown in the instructions. If a cap on the initial price is enforced, all possible initial prices are shown.

`graph_instructions` (string field):

The instructions come with a stylized decision tree visualizing each player's possible moves and corresponding payoffs. By setting the variable `graph_instructions` to either `'horizontal'`, `'vertical'`, or `'none'`, the experimenter can set the graph's orientiation or remove it from the instructions screen. As the decision tree automatically adapts to different settings determined in `config.py` and thereby expands when increasing the number of players in a group, if the number of players in a group is larger than four, a vertical orientation of the decision tree should be the better option.

`controlquestions` (Boolean field):

`controlquestions` (`True` or `False`) determines whether a separate, prepared template of control questions (`ControlQuestions.html`) is rendered after the experimental instructions but prior to the game.

`controlquestions_set1` (list of integers):

`controlquestions_set1` and `controlquestions_set2` allow to select which of the prepare question one wants to include. Set 1 consists of the nine questions put forward by Moinas and Pouget (2016) and Hong et al. (2018); they can be included by including the corresponding number from 1 to 9 in the list `controlquestions_set1` (e.g. `[1, 2, 3]`, for questions 1 through 3). Set 2 consists of the seven questions used by Janssen et al. (2018) and can be included by including the corresponding number from 10 to 16 in the list `controlquestions_set2` (e.g. `[10, 11, 12]` for questions 10 through 12). All prepared questions as well as their corresponding answers can be found in the file `models.py` in the app's root directory. Note that – as the prepared instructions – the prepared control questions only serve as an example and might have to be adapted according to the settings.

`controlquestions_set2` (list of integers):

see `controlquestions_set1`

`controlquestions_correct` (Boolean field):

`controlquestions_correct` allows the experimenter to require correct answers for the selected control questions. If its value is set to `True`, players are also informed that the game only begins when all players in the group have correctly answered the questions.

`results` (Boolean field):

`results` (`True` or `False`) determines whether the results of the game are realized and whether a template presenting the results (Results.html) is rendered after the game has been completed. If its value is set to `True`, after making all decisions in a stage of the (potentially repeated) game, a player has to wait for the other players in her group after being presented with the results. After all group members completed the game, outcomes and corresponding payoffs are calculated. In case of a repeated game, results and corresponding outcomes are separately calculated for each repetition. If `results = False`, the game ends after a player has entered all decisions without calculating and presenting any outcomes.

`graph_results` (Boolean field):

The stylized decision tree included in the instructions can also be shown on the screen presenting the game's outcomes by setting `graph_results = True`.

`exp_currency` (string field):

Determines the name of the experimental currency, e.g. `'ECU'`, `'Taler'`, etc.

References

Chen, D. L., Schonger, M., & Wickens, C. (2016). oTree—An open-source platform for laboratory, online, and field experiments. *Journal of Behavioral and Experimental Finance*, *9*, 88-97.

Hong, J., Moinas, S., & Pouget, S. (2018). *Learning in Speculative Bubbles: An Experiment* (No. 18-882). Toulouse School of Economics (TSE).

Janssen, D. J., Füllbrunn, S., & Weitzel, U. (2018). Individual speculative behavior and overpricing in experimental asset markets. *Experimental Economics*, 1-23.

Moinas, S., & Pouget, S. (2013). The bubble game: An experimental study of speculation. *Econometrica*, *81*(4), 1507-1539.

Moinas, S., & Pouget, S. (2016). The bubble game: A classroom experiment. *Southern Economic Journal*, *82*(4), 1402-1412.