# RAM: A Collection of Mechanisms for Resource Allocation in oTree.

### Installation

The first thing to do is to copy all app directories (BOS, DA, GSPDM, RSD, TTC, and UMBS) provided in the package to your *oTree* root folder (e.g. C:/Users/<User>/oTree/).

Next, in settings.py under the Section SESSION_CONFIG, add the following lines:

```
SESSION_CONFIG = [
...

{
‘name’: ‘rsd’,
‘display_name’: "Random Serial Dictatorship",
‘num_demo_participants’: 3,
‘app_sequence’: [‘rsd’],
},
]
```

Do so for all apps you intend to use.num_demo_participants can also be set to some other value to satisfy your needs for testing. Settings.py also contains LANGUAGE_CODE, where you can specify the working language and make use of the i18n interface of RAM. Also make sure that your *oTree* version is up-to-date. The app was originally programmed in version 2.1.23. You can obtain your current version with the command otree --version from your project directory. For update instructions, please see the *oTree* manual.[1] After setting the variables explained in the following Section, the *oTree* command otree resetdb has to be executed.

### User Settings

All six mechanisms are standalone *oTree* apps which can be seamlessly combined with other *oTree* apps and are highly customizable by the experimenter. For this purpose, a file (user_settings.py) containing the *oTree* class Constants is placed in the application folder and allows the experimenter to customize appearance, mechanism parameters, and general settings for every mechanism. For a detailed description of the functionality of every customizable parameter, user_settings.py also includes explanations for all variables. The following variables are to be specified in order to set up the experiment:

[1] https://otree.readthedocs.io/en/latest/install.html

**Design Variables**

`players_per_group` (int)
Here the user can specify how many players should be active in the market. For BOS, DA, and TTC, this refers to the players on the proposing side of the market.

`valuations_t*` (list)
... with * referring to a specific type and being an integer from 1 to 10. Here the user can specify the valuations for the active players in the market. It is a list with the structure [<valuation_r$_1$>, <valuation_r$_2$>, ... , <valuation_r$_n$>]. If more than one of these lists are specified, the *oTree* built-in function role(self) defines types of players using id_in_group The length of the list(s) also defines the number of resources.

`priorities_r*` (list)
... with * referring to a specific type and being an integer from 1 to 10. Here the user can specify the priority profiles for the passive side of the market. It is a list with the structure [<Player with Priority 1>, <Player with Priority 2>, ... , <Player with Priority m>].
You need to specify a list for each resource.

`capacities` (list)
Defines a vector of capacities for each resource in the game. Each element in the list refers to the capacity of the respective resource.

`s_len` (int)
Refers to the number of resources that can at most be allocated to active players in the market. This variable is only present in GSPDM, RSD, and UMBS.

`endowment` (int)
Specifies the amount of fictional currency units that the players distribute over resources. This variable is only present in GSPDM, RSD, and UMBS.

**Appearance Variables**

`application_framing` (boolean)
The experiment can either choose a neutral framing (Participants/Resources), or framed version of the experiment. The framed version represents the most common applications for the respective type of allocation problem (Participants/Schools or Students/Courses).

`instructions` (boolean)
Defines if the instructions for the mechanism be included. The instructions provided are partly taken from the seminal experimental papers cited in the paper and are only samples.

`instructions_example` (boolean)
Additionally, the user can choose to include a minimum example of the mechanism. The instructions provided are partly taken from the seminal experimental papers cited in the paper.

`show_counter` (boolean)
A live counter implemented in *javascript* is implemented such that subjects can see how many bidding points they have left to bid. This variable is only present in GSPDM and UMBS.

`confirm_button` (boolean)
If set to True, a confirm button is added to the standard *Next* button of *oTree*. This has proven to be a desirable feature, since subjects tend to accidentally hit the Enter-Key while making their decisions.

`results` (boolean)
Defines if the market results should be displayed to subjects. `Results.html` sums up the outcomes of the market and display the *oTree* variable `player.payoff` for each participant.

**Information Variables**

`show_capacities` (boolean)
If set to "True", the quota specified in `capacities` will be shown to players on the decision screen and in the instructions.

`show_types` (boolean)
If set to True, players will have a hint on the decision page and in the instructions that there are different types of players in the market. Only works if multiple type vectors have been specified in `valuations_t*`.

`show_valuations` (boolean)
Should players see the other players' valuation profiles and on the decision page? Only works if `show_types` has been set to True above.

`show_priorities` (boolean)
Specifies whether a player see the resources' priorities for her in the instructions and on the decision page. This variable is only present in BOS, DA, and TTC.

**Miscellaneous Variables**

`enforce_binding` (boolean)
If set to True, players are forced to use up all bidding points specified in `endowment`. This variable is only present in GSPDM and UMBS.