Market Simulator Overview

Overview

This file is meant to serve as an overview of the simulator to help get started modifying an executing it. Everything in Java is located under the package prefix `edu.umich.srg` so that will be omitted when referring to files.

Running

There are generally three different ways to execute the simulator: Java, command line, and EGTA online.

Java

The main class for executing a simulation is `marketsim.MarketSimulator`. Its constructor takes three arguments. 1. A fundamental, which determines how the asset in the simulator is valued over time, and represents the main source of adverse selection in the simulator (see the fundamental.{md,pdf}). 2. A log, this determines where logging information is set. A good default might be `Log.nullLogger()`. 3. A random to use as the seed of the simulation. Lacking better alternatives `new Random()` will work just fine. If you want to run the same random repeatedly, use the same number for the seed, e.g. `new Random(0)`.

After the MarketSimulator is created, the next step is to add Markets and Agents. Markets must implement the `marketsim.market.Market` interface, and Agents must implements the `marketsim.agent.Agent` interface. It will probably be easier to create Markets first, as Agents will likely need to know about them to be constructed.

After adding all of the important entities, calling `MarketSimulator.initialize()` will set the simulator up to run. Calling `MarketSimulator.executeUntil(<final time>)` will then run the simulator for `<final time>` amount of time.

After executing the simulator, data can be analyzed from any object you still have access to. In addition, `MarketSimulator.getAgentPayoffs()` and

`MarketSimulator.computeFeatures()` will returns some additional information that may be desired.

Command Line

To execute the simulator from the command line, an executable jar of the simulator must first be made. This is created by typing `make jar`. After compiled, the script `market-sim.sh` will now expose the command line interface of the simulator. `market-sim.sh --help` give a general overview of how to execute the simulator, but the general usage will be:

```
< compressed-simulation-spec.json ./market-sim.sh number-of-simulations > output.json
```

or

```
< compressed-simulation-spec.json ./market-sim.sh number-of-simulations | jq ...
```

For more information on jq see jq.{md,pdf}. `compressed-simulation-spec.json` is a json file that indicates the parameters of the simulation. The general structure is:

```
{
    "assignment": {
        "role1": {
            "agent_type:key1_value1_...": num-agent,
            ...
        },
        "role2": ...
    },
    "configuration": {
        "parameter1": value1,
        "parameter2": ...
    }
}
```

`assignment` and `configuration` are hard coded strings and must remain the same. `role1` etc. can be any string. They define groups of agents are only really relevant to execution on EGTA online. `agent_type` is a string that must be a key in `marketsim.EntityBuilder.agentNameMap`. This defines how to construct agents from their command line arguments. The string for `key1` and `parameter1` etc. must be defined in `marketsim.Keys`. They keys are class names that also specify the type of their corresponding value. This allows parameter maps to be type safe. When agents are created they get access to all of the parameter values in `configuration` and all of the values specified after their `agent_type`. If a key is repeated, the one specified after `agent_type` takes precedence. This might be useful for defining a general agent `arrivalRate`, but have one type of agent overwrite that `arrivalRate` to be faster or slower.

EGTA Online

FIXME: Complete