MASSim Scenario Documentation

Agents in the City (2017)

Background Story

In the year 2044, following the great water conflicts of 2033, humankind has finally colonized and terraformed Mars; so much in fact that it greatly (structurally) resembles parts of old Earth. Unfortunately, scientists are still working on making the atmosphere breathable, so humans still need (costly) special equipment to leave their homes. Luckily, some entrepreneurial individuals found remnants of the old 'All Terrain Planetary Vehicles' which they retrofitted to do some serious business. Wanting to improve life for everyone on Mars (and to make a living off of that), they are sending their ATPVs out to gather the planet's natural resources and supply everyone in need with useful items.

Introduction

The scenario consists of two **teams** of a number of agents each moving through the streets of a realistic city. The goal for each team is to earn as much money as possible, which is rewarded for completing certain **jobs**. An agent is characterized by its **battery** (i.e. how long it can move without recharging), its **capacity** (i.e. how much volume it can carry) and its **speed**. The scenario features 4 distinct **roles**: drones, motorcycles, cars and trucks, sorted by increasing capacity and energy, and decreasing speed. The city map is taken from **OpenStreetMap** data and routing is provided by the Contest server. As the simulation is divided into discrete steps, each agent can move a fixed distance each turn (unless its destination is closer than this distance).

Each simulation features a set of random **items**. Items differ by their volume and how they can be acquired.

The agents are positioned randomly on the map, as are a number of **facilities**, among them **shops**, **charging stations**, **resource nodes**, **storage** facilities, and **dumps**.

Jobs comprise the acquisition, and transportation of goods. These jobs can be created by either the system (environment) or one of the agent teams. There is one type of job: **regular jobs**.

Regular jobs have their rewards defined upfront, which is given to the first team to complete that job, while the other team goes away empty-handed. The teams have to decide which jobs to complete and how to do that.

Tournament points are distributed according to the amount of money a team owns at the end of the simulation. To get the most points, a team has to beat the other, as well as surpass the seed capital given to the team at the beginning of the simulation, i.e. make an overall profit.

Locations

Locations in the scenario are given as pairs of latitude and longitude double values. For the sake of simplicity, we assume them to form a uniform grid (which will be okay, if we don't go near the poles).

To simplify reaching a location, these values are rounded and approximated with the help of two parameters:

Proximity: The proximity value specifies which precise locations are still considered the same, which is important when an agent needs to be at the same location as e.g. another agent or a facility. So, it determines how many decimal places of the latitude and longitude values are compared when checking equality of Locations.

Example: 51.1111 and 51.1112 would still be considered equal with a proximity of 3 and different with a proximity of 4.

CellSize: The cell size specifies the length in meters an agent with speed 1 could travel in a single step.

Items

Each simulation features a (random) set of item types. Items have a unique **name** and a **volume**, which comes into play when items must be carried or stored.

Facilities

Facilities are placed randomly on the map. Each facility has a unique name and location.

It is possible for blackouts to occur in the city causing some facilities to be out of order for a small number of steps. Agents are not able to recognize facilities that are affected by a blackout and will only be able to perceive that a facility is currently not working when they get the corresponding failure code after executing an action in this facility.

Currently only charging stations are affected by blackouts.

Shops

In **shops**, items can be bought at prices specific to the particular shop. Each shop only offers limited quantities of a subset of all items. However, items are restocked after a number of steps.

E.g. if a shop has **restock** 5, one piece of each missing item is added to the shop's stock each 5 steps.

Charging stations

Charging stations have to be frequently visited by agents in order to recharge their battery. They have a **rate** which expresses the amount of charge that can be restored in one step (i.e. a charging station with a rate of 40 would restore 40 charge units of an agent's battery).

Dumps

Items can be destroyed at **dumps** (to free capacity). Those items cannot be retrieved.

Storage

Storage facilities allow to store items up to a specific volume and also are the target for completing jobs.

The storage has a limited **capacity** which counts for all teams, i.e. one team can fill a storage while not leaving any space for other teams.

Each team has a separate (unlimited) compartment in each storage. This compartment cannot be filled directly but only as a consequence of other actions:

- If a team delivers items towards job completion, but a) another team is faster or b) the job ends (due to its time limit).

Resouce nodes

Resource nodes represent natural sources for certain items. Agents can go to the nodes to gather those items, but it always takes a certain number of gather actions until a new resource is found.

When there are multiple agents gathering at the same resource node only one of them will get the resource depending on the order in which they executed their actions.

The location of resource nodes is not common knowledge and agents are only able to perceive the nodes when they are close to them.

Teams

The scenario can be played with any number of teams simultaneously. Each team contains the same number of agents per **role**.

Roles

The roles in the scenario can be configured under the top-level `roles` key in the simulation JSON object (which is one element of the `match` array).

```
"roles" : {
  "car" : {
    "speed" : 3,
    "load" : 550,
    "battery" : 500,
    "roads" : ["road"]
  },
  "drone" : {
    "speed" : 5,
    "load" : 100,
    "battery" : 250,
    "roads" : ["air"]
  },
  "motorcycle" : {
    "speed" : 4,
    "load" : 300,
    "battery" : 350,
    "roads" : ["road"]
  },
  "truck" : {
    "speed" : 2,
    "load" : 3000,
    "battery" : 1000,
    "roads" : ["road"]
  }
}
```

Each role has its name as key and the following parameters:

- **speed**: how many 'units' the agent can move in one step
- **load**: how much volume the agent may carry
- **battery**: the agent's battery size
- **roads**: which roads the agent can navigate (currently 'road' for all roads and 'air' for travelling linear distances between two points)

These 4 roles will also be used in the contest, howeer, the parameters are still subject to change.

Job

A job is the general way to earn money in this scenario.

- **begin**: the job begins in this step, i.e. this is the first step where the job is perceived by all agents
- **end**: the last step in which the job can be completed. At the end of this step, the job cannot be perceived anymore.
- **reward**: the amount of money that is earned by completing the job
- **storage**: to complete the job, items have to be delivered to this storage

Mission

A mission is a special type of job that is given out randomly. All teams will receive an instance of the same mission to complete. This mission has to be completed (or the fine paid).

Actions

In each step, an agent may execute *exactly one* action. The actions are gathered and executed in random order.

All actions have the same probability to just fail randomly.

Some actions may lead to **conflicts**. For example, two agents might want to buy the same item from a shop (and only one of these items is left). In that case, the agent whose action is executed first gets the item, while the action of the other agent is treated as though no item is available (which is actually true).

Each action has a number of `parameters`. The exact number depends on the type of action. Also, the position of each parameter determines its meaning. Parameters are always string values.

goto

Moves the agent towards a destination. Consumes **10** charge units if successful. Can be used with 0, 1 or 2 parameters. If 0 parameters are used, the agent needs to have an existing route which can be followed.

No	Parameter	Meaning
0	Facility	The name of a facility the agent wants to move to.

Note: Names of *ResourceNodes* are not allowed, as they are not common knowledge.

No	Parameter	Meaning
0	latitude	The latitude of the agent's desired destination.
1	longitude	The longitude of the agent's desired destination.

Failure Code	Reason
failed_wrong_param	The agent has no route to follow (0 parameters), more than 2 parameters were given or the given coordinates were not valid double values (2 parameters).
failed_unknown_facility	No facility by the given name exists (1 parameter).
failed_no_route	No route to the destination exists or the charge is insufficient to reach the next waypoint.

give

Gives a number of items to another agent in the same location.

No	Parameter	Meaning
0	Agent	Name of the agent to receive the items.

No	Parameter	Meaning
1	Item	Name of the item to give.
2	Amount	How many items to give.

Failure Code	Reason
failed_wrong_param	More or less than 3 parameters have been given, no agent by the name is known or an amount < 0 was specified.
failed_unknown_item	No item by the given name is known.
failed_counterpart	The receiving agent did not use the receive action.
failed_location	The agents are not in the same location.
failed_item_amount	The giving agent does not carry enough items to give.
failed_capacity	The receiving agent could not carry all given items.

receive

Receives items from other agents. Can receive items from multiple agents in the same step.

No parameters.

Failure Code	Reason
failed_counterpart	No agent gave items to this agent.

store

Stores a number of items in a storage facility.

Failure Code	Reason
failed_wrong_param	More or less than 2 parameters were given.
failed_location	The agent is not located in a facility.
failed_wrong_facility	The agent is not in a storage facility.
failed_unknown_item	No item by the given name is known.
failed_item_amount	The given amount is not an integer, less than 1 or greater than what the agent is carrying.
failed_capacity	The storage does not have enough free space.
failed	An unforeseen error has occurred.

No	Parameter	Meaning
0	Item	Name of the item to store.
1	Amount	How many items to store.

retrieve & retrieve_delivered

Retrieves a number of items from a storage. The first can be used to retrieve items that have been stored before, while the second is used to retrieve items from the team's 'special' compartment (see Storage).

No	Parameter	Meaning
0	Item	Name of the item to retrieve.
1	Amount	How many items to retrieve.

Failure Code	Reason
failed_wrong_param	More or less than 2 parameters have been given.
failed_location	The agent is not located in a facility.
failed_wrong_facility	The agent is not in a storage facility.
failed_unknown_item	No item by the given name is known.
failed_item_amount	The given amount is not an integer, less than 1 or more than available.
failed_capacity	The agent has not enough free space to carry the items.

buy

Buys a number of items in a shop.

No	Parameter	Meaning
0	Item	Name of the item to buy.

No	Parameter	Meaning
1	Amount	How many items to buy.

Failure Code	Reason
failed_wrong_param	More or less than 2 parameters have been given.
failed_location	The agent is not in a facility.
failed_wrong_facility	The agent is not in a shop.
failed_unknown_item	No item by the given name is known.
failed_item_amount	The given amount is not an integer, less than 1, or greater than the shop's available quantity.
failed_capacity	The agent does not have enough free space to carry the items.

deliver_job

Delivers items towards the completion of a job. The agent is automatically drained of all items matching the job's remaining requirements.

No	Parameter	Meaning
0	Job	The name of the job to deliver items for.

Failure Code	Reason
failed_wrong_param	Not exactly 1 parameter has been given.
failed_unknown_job	No job by the given name is known.

Failure Code	Reason
failed_job_status	The given job is not active or has not been assigned to the agent's team.
failed_location	The agent is not in the storage associated with the job.
successful_partial	Not really a failue. Items have been delivered but the job has not been completed by this action.
useless	The agent does not have any items to contribute to the job.

dump

Destroys a number of items at a dump facility.

No	Parameter	Meaning
0	Item	Name of the item to destroy.
1	Amount	How many items to destroy.

Failure Code	Reason
failed_wrong_param	Not exactly 2 parameters have been given.
failed_location	The agent is not in a facility.
failed_wrong_facility	The agent is not at a dump location.
failed_unknown_item	No item by the given name is known.
failed_item_amount	The given amount is not a positive integer or more than the agent is carrying.

charge

Charges the agent's battery at a charging station.

No parameters.

Failure Code	Reason
failed_wrong_param	Parameters have been given.
failed_location	The agent is not in a facility.
failed_wrong_facility	The agent is not in a charging station.
failed_facility_state	The charging station is currently out of order due to a blackout.

recharge

Uses the agent's solar collectors to recharge its battery (slowly).

No parameters.

Failure Code	Reason
failed_wrong_param	Parameters have been given.

continue & skip

Follows an agent's route or does nothing if the agent has no route.

No parameters.

Failure Code	Reason
failed_wrong_param	Parameters have been given.

Failure Code	Reason
failed_no_route	The agent's route could not be followed any longer (charge may be too low).

abort

Does nothing and clears the agent's route (if it exists).

unknownAction

This action is substituted if an agent submitted an action of unknown type.

randomFail

This action is substituted if the agent's action randomly failed.

noAction

This action is substituted if the agent did not send an action in time.

gather

Gathers a resource from a resource node.

No parameters.

Failure Code	Reason
failed_wrong_param	Parameters have been given.
failed_location	The agent is not in a facility.
failed_wrong_facility	The agent is not in a resource node.
failed_capacity	The agent does not have enough free space to carry the resource.

Failure Code	Reason
partial_success	Not really a failure. The action was executed successfully but there was no resource found.

Percepts

Percepts are sent by the server as XML files and contain information about the current simulation. Initial percepts (sent via `SIM-START` messages) contain static information while other percepts (sent via `REQUEST-ACTION` messages) contain information about the current simulation state.
The complete XML format is discussed in protocol.md.

Initial percept

This percept contains information that does not change during the whole simulation. As mentioned in the protocol description, everything is contained in a `simulation` element. Example:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<message timestamp="1489763697332" type="sim-start">
  <simulation id="2017-QuickTest-Sim" map="london"
              seedCapital="10" steps="1000" team="A"
              minLat="1.1" maxLat="1.2" minLon="2.1" maxLon="2.3" centerLat="1.15"
centerLon="2.2">
    <role battery="500" load="1000" name="SampleRole" speed="10">
    </role>
    <item name="item0" volume="72"/>
    <item name="item14" volume="3">
    </item>
  </simulation>
</message>
```

Simulation details

The `simulation` tag has attributes for the simulation `id`, the name of the `map` that is used and its bounds, the seed capital, the number of simulation `steps` to be played and the name of the agent's `team`.
Role details
The percept also contains the agent's `role` and its details; speed, maximum load, name and maximum battery charge.

Contest note

The roles and their details will be defined (and made public) in before and not change between simulations.

Step percept

This percept contains information about the simulation state at the beginning of each step. A more or less complete example can be found here.

Example:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<message timestamp="1489763697397" type="request-action">
  <percept deadline="1489763701395" id="0">
    <simulation step="0"/>
    <self charge="101" facility="shop1" lat="51.4741" load="0" lon="-0.06057"
name="agentA1"
          role="SampleRole" team="A">
      <action result="successful" type="skip"/>
      <items amount="4" name="item0"/>
      <route i="0" lat="51.4739" lon="-0.06657"/>
      <route i="1" lat="51.47384" lon="-0.07346"/>
      <route i="2" lat="51.47379" lon="-0.08041"/>
      <route i="3" lat="51.47349" lon="-0.08759"/>
      <route i="4" lat="51.47452" lon="-0.09439"/>
      <route i="5" lat="51.47704" lon="-0.10037"/>
      <route i="6" lat="51.4777" lon="-0.10676"/>
    </self>
    <team money="10"/>

    ...

  </percept>
</message>
```

The information is contained in the `percept` element within the message. This element contains an arbitrary number of child nodes, each representing an element of the simulation.

Self details

The `self` element contains information about the agent itself; its current battery charge and used carrying capacity, position, role and team. If the agent is in the same location as a facility, that facility is listed as an attribute as well. The action the agent executed in the last step together with its result is included in a child node of the `self` element.
Also, a child element for each carried item type is nested. Finally, if the agent currently follows a route, each waypoint of that route is listed in a child node, containing its location and index within the route.

Team details

The `team` element contains information about the agent's team; currently only how much money it owns.

Entity details

For each entity (or agent) in the simulation, one `entity` element is added.
Example:

```
<entity lat="51.4659" lon="-0.1035" name="agentB3" role="SampleRole" team="B"/>
```

Each of these elements contains the entity's name, position, role and team.

Facility details

For each facility, a specific element is included.

Shop details

Example:

```
<shop lat="51.4861" lon="-0.1477" name="shop1" restock="3">
  <item amount="10" name="item0" price="217"/>
</shop>
```

For each shop, its name, position and restock value are included. Each shop contains a child node for each item type that can currently be bought, consisting of the item's name, its price and the available quantity.

Charging station details

Example:

```
<chargingStation lat="51.5182" lon="-0.0361" name="chargingStation6" rate="88"/>
```

Dump details

Example:

```
<dump lat="51.5163" lon="-0.1588" name="dump2"/>
```

Storage details

Example:

```
<storage lat="51.4906" lon="-0.0825" name="storage6" totalCapacity="9277"
         usedCapacity="0">
```

```
        <item delivered="3" name="item0" stored="0"/>
</storage>
```
The storage contains a child node for each item type that is stored or delivered (or both) for an agent's team.

ResourceNode details

Example:

```
<resourceNode lat="51.478" lon="-0.03632" name="resourceNode1" resource="item7"/>
```
The resource node element contains the item that can be mined. Attention: This percept is only visible if the agent is "close" to the node (see `visibilityRange` parameter).

Job details

An element for each job is added (`job` or `mission`).
Regular job example:

```
<job start="59" end="159" id="job0" reward="5018" storage="storage2">
  <required amount="3" name="item0"/>
</job>
```
Mission job example: (all currently active missions for the team)

```
<mission auctionTime="0" start="10" end="100" fine="500" id="job2"
         lowestBid="1000" reward="1001" storage="storage0">
  <required amount="42" name="item1"/>
</mission>
```

Commands

A number of commands are handled by this scenario, which may help with debugging or testing your agents.

`print facilities`: prints all facilities to the console
`print items`: prints all items to the console
`give itemX agentY Z`: gives Z units of item "itemX" to an agent "agentY" (if the agent has enough free space)
`store storageX itemY A Z`: stores Z units of item "itemY" for team "A" in storage "storageX" (if the storage has enough free space)
`addJob X Y Z storage0 item0 A item1 B ...`: adds a new job to the system, starting in step X, ending in step Y, with reward Z, target storage "storage0", and requiring A units of "item0", B units of "item1", etc.