Program your own Agent

Digital Urban Visualization. People as Flows.

09.11.2015

iA

zuend@arch.ethz.ch

treyer@arch.ethz.ch

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

DARCH

iA Chair of Information Architecture

Exercise

program your own agent

Image: www.torobar.com


Mouse Position - x: 12 y: 174
D-Map
Density
Counter
Panic
Paths
Deaths by
hunger: 0
thirst: 0
no toilet: 0
no music: 0
674 ClosestNeedPerson
666 RandomPerson
660 DaniZuend
Rec
Net

Exercise

location choice

The people in the simulation first choose their location and then walk to it, choosing the shortest path.

They only re-evaluate their decision with a very low probability or if they have a need that enters a critical value.

Exercise

current agents

In the basic framework, two naive agents are already implemented. *RandomPerson* and *ClosestNeed-Person*.

RandomPerson randomly chooses one of the need providers and goes to it.

ClosestNeedPerson checks what his most urgent need is and then chooses the closest location which provides it.

Exercise

what to do?

You have to implement the function with which the agents decide where to go next.

CalienteFestival.java | Agent.java | Person.java | Rando

```java
package calientefestival;

import java.util.ArrayList;

public class DaniZuend extends Person {

    public DaniZuend(Agent curr) {
        super(curr);
        this.setColor();
    }


    public DaniZuend(double x, double y) {
        super(x,y);
        this.setColor();
    }

    @Override
    protected Needs locationChoice(ArrayList<Agent> others) {
        Needs n;
        ArrayList<Needs> needs = this.getNeeds(others);
        n = needs.get(rGen.nextInt(needs.size()));
        return n;
    }


    @Override
    protected void setColor() {
        this.r = 200;
        this.g = 200;
        this.b = 100;
    }

}
```

Exercise

what to do?

You have to implement the function with which the agents decide where to go next.
Delete the content of the function *locationChoice* and implement your own.

```java
32    @Override
33    protected Needs locationChoice(ArrayList<Agent> others) {
34        Needs n;
35        ArrayList<Needs> needs = this.getNeeds(others);
36        n = needs.get(rGen.nextInt(needs.size()));
37        return n;
38    }
```

→

```java
32    @Override
33    protected Needs locationChoice(ArrayList<Agent> others) {
34    }
35
```

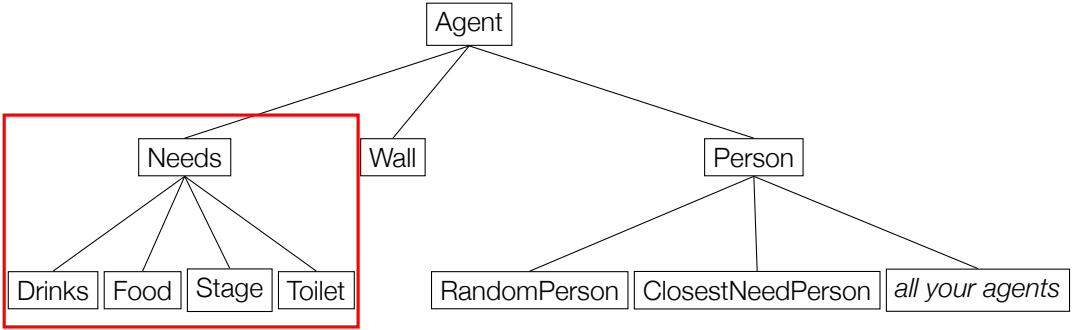
This will result in an error, because the function must return a sub-type of type *Needs*. For example:

```java
40    @Override
41    protected Needs locationChoice(ArrayList<Agent> others) {
42        ArrayList<Needs> needs = this.getNeeds(others);
43        Needs first = needs.get(0);
44        return first;
45    }
46
```

Exercise

what to do?

This will result in an error, because the function must return a sub-type of type *Needs*.



Exercise

helpers

The function *locationChoice* has one input, and that is all the agents which are currently in the system; wall elements, active people, and all need providers.

The input is called *others*.

Exercise

class functions

Since your person inherits from *Person* it already has some functionalities to work with the input *others*. With several *get* functions, you can get the state of the current agent:

bladderHappiness(), *hungerHappiness()*, *musicHappiness()*, and *thirstHappiness()* give you the current happiness of the respective need.

getFoodStalls(others), *getPersons(others)*, *getDrinkStalls(others)*, *getStages(others)*, *getToilets(others)*, *getWalls(others)*, *getNeeds(others)* extract the respective types from the input list *others* and return a list of all existing ones.

Exercise

class functions

Since your person inherits from *Person* it already has some functionalities to work with the input *others*. With several *get* functions, you can get the state of the current agent:

minNeed() returns an integer with the smallest happiness of all the needs.

To draw a random number, you can use ***rGen***. For example to draw a random integer between 0 and 3 (inclusive) you can call *rGen.nextInt(4)*.

Exercise

get locations

The root class of all agents in the system provides some core functionalities, the most interesting for you is the *getPosition()* function, which returns the location of the corresponding agent.

To get the position of a food stall *aFoodStall*, for example, you can store its x-location into *currX* by calling:

double currX = aFoodStall.getPosition().x();

Exercise

get number of customers

All need providers count their number of served customers.

To get the number of served customer of a need provider, do the following. Assume we have again a food stall called *aFoodStall*.

The number of served customers can then be stored in variable *nrCustomers* by calling:

int nrCustomers = aFoodStall.getVisits()

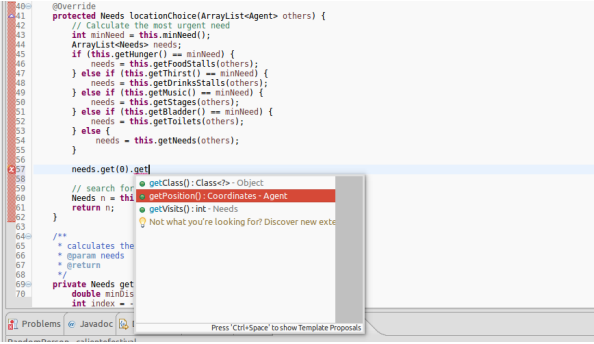
Exercise

using Eclipse autocompletion

Eclipse provides a huge help in finding out which functionalities are provided by the different classes.

Start typing and then hit *Control* (*Command* with Macs) and *Space*. Eclipse provides you then with all the possible options you have.

This is especially useful when you write down the variable name plus a point and then hit the short-cut.



Exercise

additional information

You can earn **5** points for having a running client and **5** for creativity of the implementation. We will run the agents on a few of your suggested layouts and give points according to the number of agents of your type after some time.
Latest hand-in date is **Tuesday, November 17th, 5 o'clock** in the morning.

The winners will be announced in two weeks and will win a price!

Image: wikipedia.org

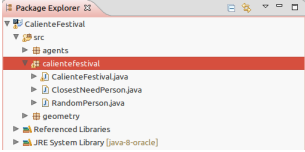
Exercise

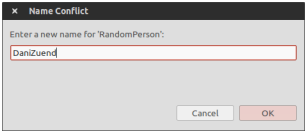
setting up your agent

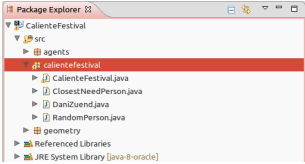
The easiest way to start with your own agent is to copy and paste for example the *RandomPerson.java* file into the same package and then rename it.

Right click on *RandomPerson.java* → click *Copy*
Eight click on the package *calientefestival* → click *Paste*
When the dialogue pops up, put in your name, without whitespaces.

After clicking *OK* there should be a class with your name in the *calientefestival* package.







Exercise

setting up your agent

To make your agent visually distinguishable from other agents, set up a unique color. This is done by opening your agent and set the *RGB* values for your agent in the *setColor* function.

```
@Override
protected void setColor() {
    this.r = 100;
    this.g = 250;
    this.b = 100;
}
```

To add the agent to the whole simulation, one last step has to be taken. You have to register the agent in the *CalienteFestival.java* class. Put the name of your class, e.g. *"DaniZuend"*, to the *classNames* list.

```
/*
 * List of all different person classes.
 */
private final List<String> classNames = Arrays.asList("ClosestNeedPerson", "RandomPerson");
```

→

```
/*
 * List of all different person classes.
 */
private final List<String> classNames = Arrays.asList("ClosestNeedPerson", "RandomPerson", "DaniZuend");
```