The Price is Right

Agents and Distributed Artificial Intelligence
Faculty of Engineering of the University of Porto

2020/2021

João Campos - up201704982@fe.up.pt

Description

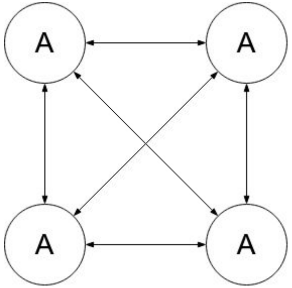
This work consists in the simulation of *The Price is Right*, using agents.

It all start by choosing a random item. The audience will first make their own guesses, adjusting them using help from other audience members. During this competitors will ask each agent from the audience for their guess. It is up to the audience to decide if they help a competitor or not. If they do so, they will send their guess, which the competitor will use to form his own guess.

At the end of the round, the competitor with the closests bet to the real price of the item. Depending on how good each agent performed, the other agents will change their confidence in that agent.

Fernando Mendes, host of *O Preço Certo*

Agents and Interaction

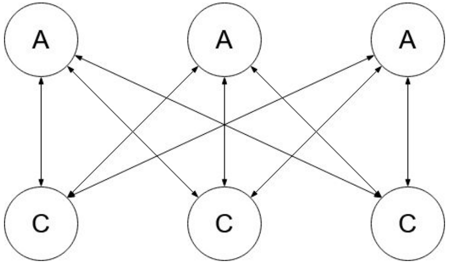


There are two types of agents: Audience and Competitor.

In every round of the game, the agents go through these steps:

1. Firstly, each element of the audience decides an initial bet
2. Then, they trade guesses with each other
3. And calculate a final guess
4. Each competitor sends a query to each audience agent
5. The agents decide if they respond with their guess or not
6. Each competitor calculates their own final guess

To communicate with each other agents use these mechanisms

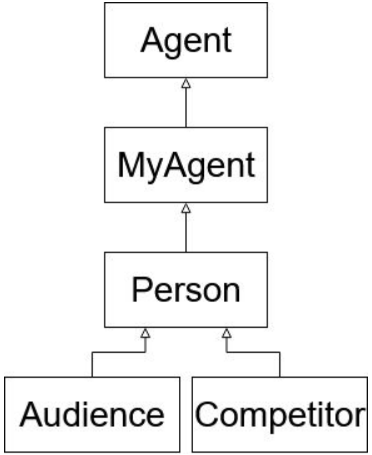


Behaviours - at the beginning of each round, the agents receive a set of behaviours that guide them through what they have to do. These behaviours are used to send and receive messages.

ACLMessage - used to create messages exchanged between agents.
These have a performative, a sender, a receiver and content.

Yellow Pages - to allow agents to search for other agents and send them messages.

Agents and Strategies



Audience

Item Knowledge: each agent knows the price of a random number of items, helping them in making guesses.
Initial guess: if the agent knows the selected item, they will guess a price closer to that of the real one, while if they don't, then they will guess at random.
Final guess: after sharing their guesses, each agent will adjust their guess. They change it based on their initial guess, the guesses of other agents and their confidence in each of those.

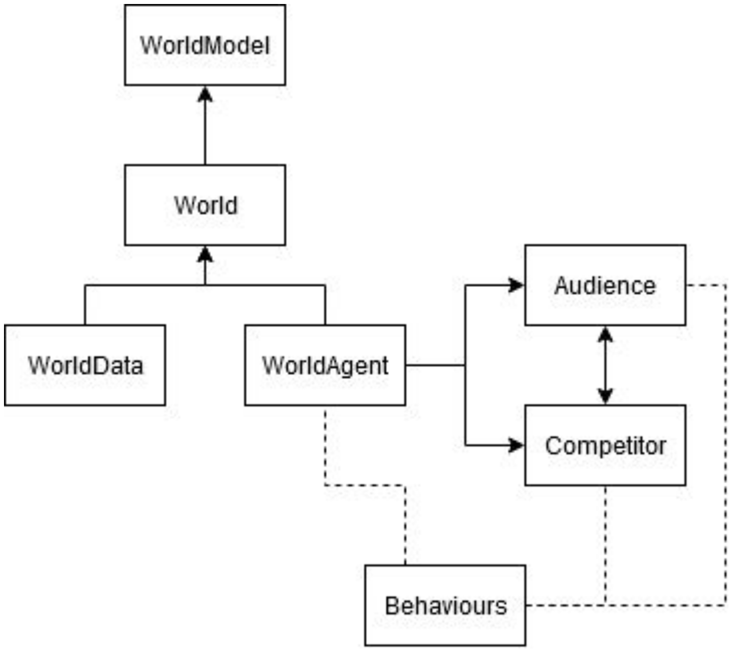
Competitor

Final guess: after receiving the guesses from the audience, each competitor will make an informed guess based on the received guesses and the confidence of the competitor.

Person

Team affinity: each agent has a set of 4 numbers that identify their team. Audience will only respond to a Competitor they are compatible with.
Confidence: each person has a random amount of confidence in each of the other agents, this will alter their decision making.

Project Structure



World package: used to control global variables to the whole run
> WorldModel, World, WorldData

Agents package: agents that play the game. WorldAgent serves as intermediary between Agents and World.
> Audience, Competitor, WorldAgent

Behaviours package: used to control agent workflow.

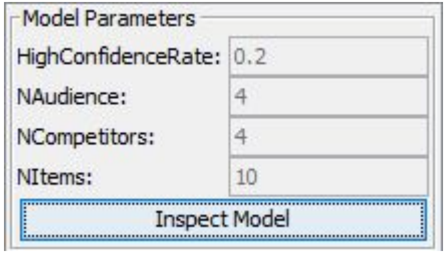
Variables

Independant

NAudience: number of agents of the audience.
NCompetitors: number of agents competitors.
NItems: number of items.
HighConfidenceRate: percentage of audience members with high self confidence.

Dependant

Accuracy over time: how good competitors are at guessing prices.
Wins with number of helpers: how important the number of compatible members is to winning



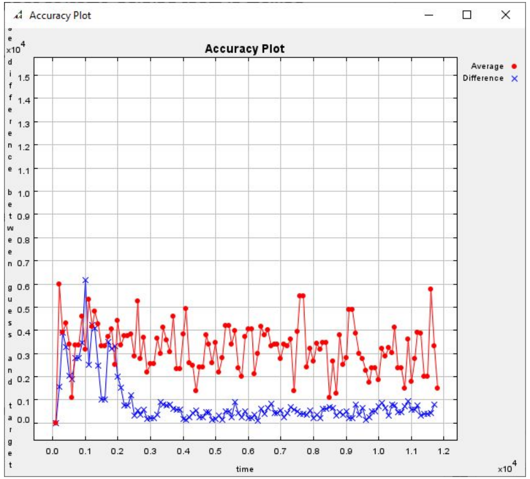
Model Parameters

HighConfidenceRate: 0.2

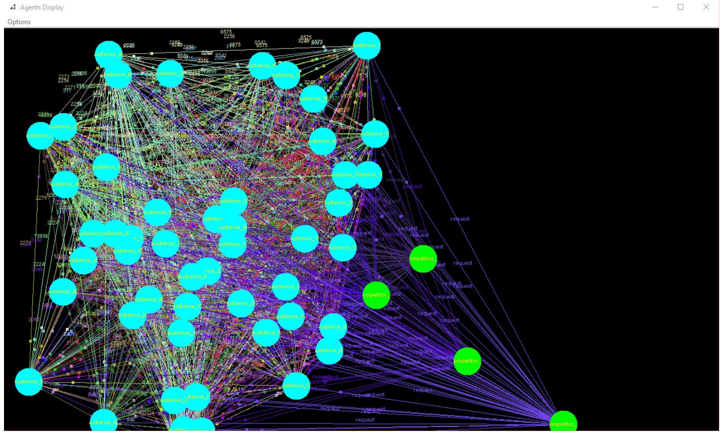
NAudience: 4

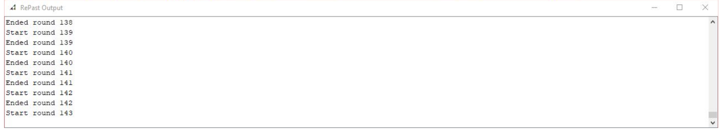
NCompetitors: 4

NItems: 10

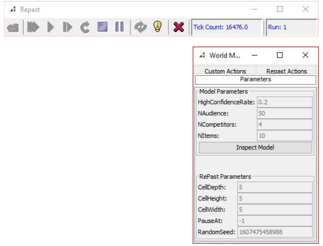


Example of Execution





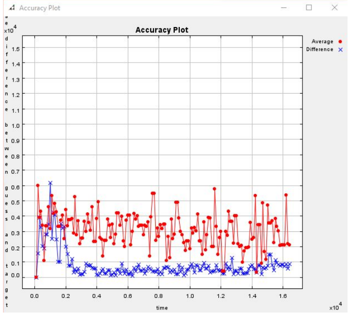
```
Ended round 138
Start round 139
Ended round 139
Start round 140
Ended round 140
Start round 141
Ended round 141
Start round 142
Ended round 142
Start round 143
```



Tick Count: 16476.0
Run: 1

Model Parameters
HighConfidenceRate: 0.2
NAudience: 50
NCompetitors: 4
NItems: 10

RePast Parameters
CellDepth: 5
CellHeight: 5
CellWidth: 5
PauseAt: -1
RandomSeed: 1607475458988



When running in graphic mode, Repast will give access to a few features. These include:

- settings that can be changed for each run
- display of the network of agents and messages sent
- plot which shows how good agents are each round
- output used to detect errors and warnings

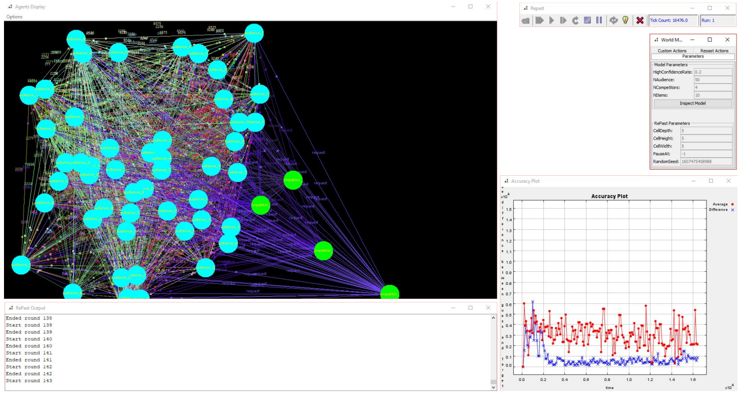
Audience members are identified by cyan colour and are positioned to the left, while Competitors are green and to the right. All messages sent by the same agent have the same colour and will have attached their content (numeric values or “null” for guesses sent by Audience and “request” for queries sent by Competitors)

Experiments and Analysis

Accuracy of agents

After each round, the agents reduce their confidence in worse audience members and increase it on better members. Because of this their guesses get better with time.

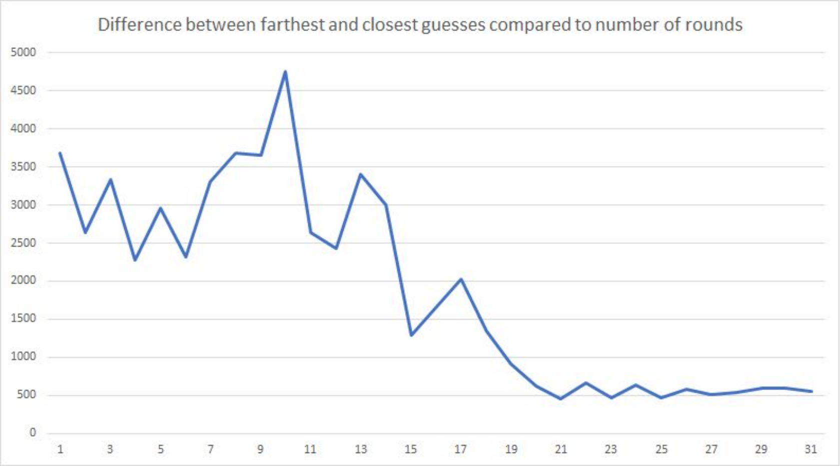
After 15 rounds we can notice a significant decrease in the difference between the farthest guess and the closest one.



Model Parameters
HighConfidenceRate: 0.2
NAudience: 50
NCompetitors: 4
NItems: 10

RePast Parameters
CellDepth: 5
CellHeight: 5
CellWidth: 5
PauseAt: -1
RandomSeed: 1607475458988

```
Ended round 138
Start round 139
Ended round 139
Start round 140
Ended round 140
Start round 141
Ended round 141
Start round 142
Ended round 142
Start round 143
```

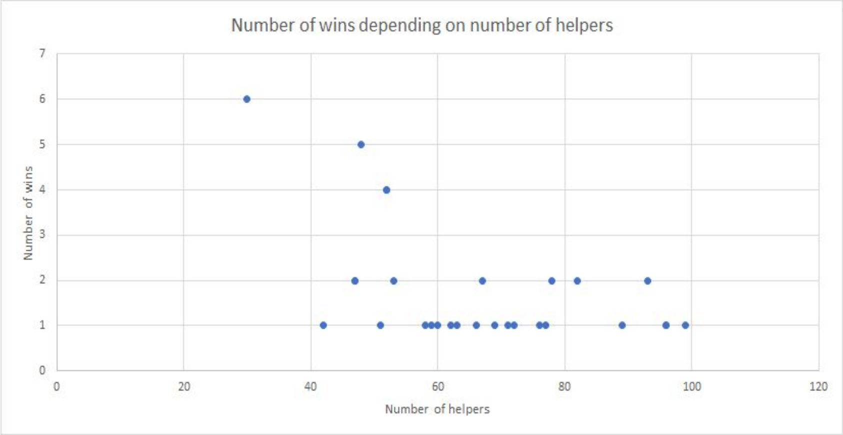


Experiments and Analysis

Helpers

Each competitor receives guesses from a certain number of audience members, those are their “helpers”. According to data, the higher this number, the less likely you are of winning.

This could be because the more agents participate, the more the final guess of the competitor will be deviated from the correct guess.



Conclusions

During the development of this project I believe to have followed all the objectives proposed, ending up with a simulation of a simple game. The primary note to take from the work is how the interaction between agents with different knowledge and views can help them improve and move towards the truth.

The experience I have gained in multi-agent systems will allow me to build better and more efficient programs. In addition to this, I also learned about Java and its graphical frameworks.

Annexes

The following slides include the implementation of classes, examples of execution and other observations.

The main implemented classes were:

- agents: Person, Audience, Competitor, WorldAgent;
- behaviours: ReceiveMsgBehaviour, SendMsgBehaviour;
- world: World, WorldData, WorldModel.

To run the project first compile with:

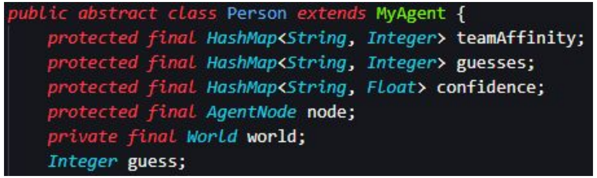
```
javac -d "./out/" -sourcepath "./src/" -cp "./lib/jade/lib/jade.jar;./lib/repast/repast.jar;./lib/sajas/lib/sajas.jar" ./src/world/WorldModel.java
```

Then run using:

```
java -cp "./out;./lib/jade/lib/jade.jar;./lib/repast/repast.jar;./lib/sajas/lib/sajas.jar" world.WorldModel [batch[nAudience[nCompetitors[nItems[highConfidenceRate]]]]]
```

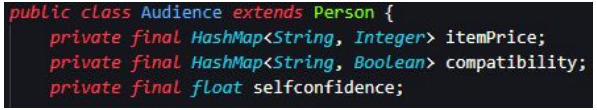
Agents

Person represents an agent that plays in the game, being separated in Audience and Competitor. Each agent is able to use yellow pages to find other agents and send them messages using a SendMsgBehaviour. For receiving messages, a ReceiveMsgBehaviour, which extends CyclicBehaviour, is used.



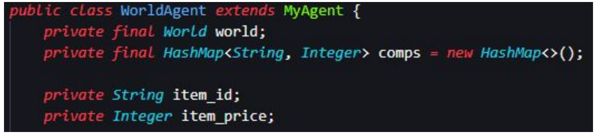
```
public abstract class Person extends MyAgent {
    protected final HashMap<String, Integer> teamAffinity;
    protected final HashMap<String, Integer> guesses;
    protected final HashMap<String, Float> confidence;
    protected final AgentNode node;
    private final World world;
    Integer guess;
```

Persons have a confidence map, which stores their confidence towards other agents, and methods for calculating guesses, and for starting and ending rounds. Audience members have the additional method to calculate compatibility with another Person.



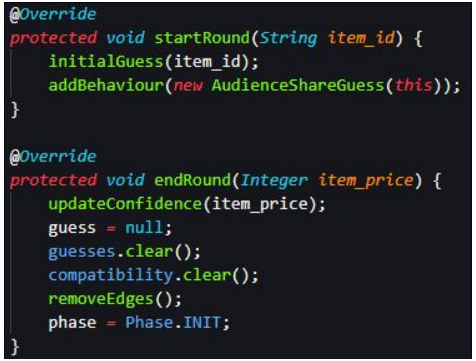
```
public class Audience extends Person {
    private final HashMap<String, Integer> itemPrice;
    private final HashMap<String, Boolean> compatibility;
    private final float selfconfidence;
```

One special agent is WorldAgent, responsible for sending a message to all agents at the start and end of a round. This is done to ensure the flow of the game.



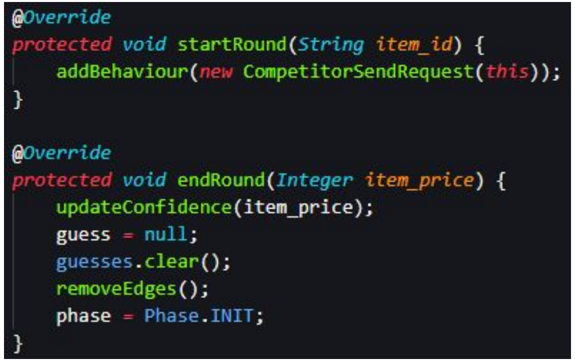
```
public class WorldAgent extends MyAgent {
    private final World world;
    private final HashMap<String, Integer> comps = new HashMap<>();

    private String item_id;
    private Integer item_price;
```

Agents



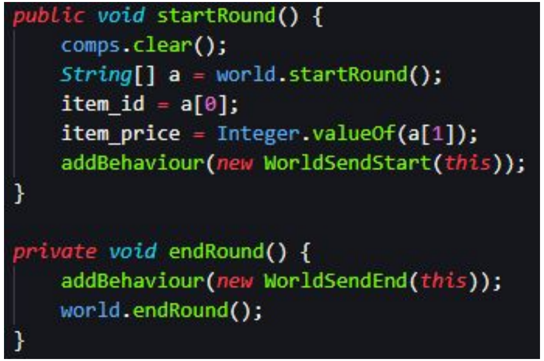
```
@Override
protected void startRound(String item_id) {
    initialGuess(item_id);
    addBehaviour(new AudienceShareGuess(this));
}

@Override
protected void endRound(Integer item_price) {
    updateConfidence(item_price);
    guess = null;
    guesses.clear();
    compatibility.clear();
    removeEdges();
    phase = Phase.INIT;
}
```

Audience methods



```
@Override
protected void startRound(String item_id) {
    addBehaviour(new CompetitorSendRequest(this));
}

@Override
protected void endRound(Integer item_price) {
    updateConfidence(item_price);
    guess = null;
    guesses.clear();
    removeEdges();
    phase = Phase.INIT;
}
```

Competitor methods

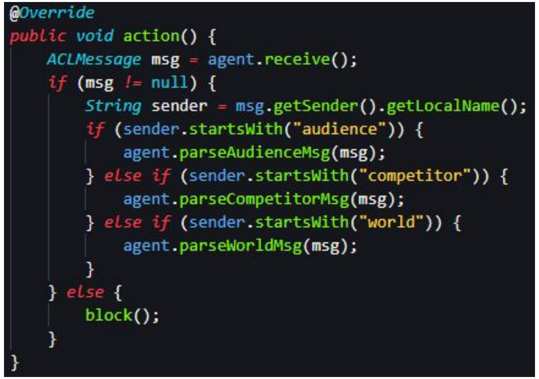


```
public void startRound() {
    comps.clear();
    String[] a = world.startRound();
    item_id = a[0];
    item_price = Integer.valueOf(a[1]);
    addBehaviour(new WorldSendStart(this));
}

private void endRound() {
    addBehaviour(new WorldSendEnd(this));
    world.endRound();
}
```

WorldAgent methods

Behaviours

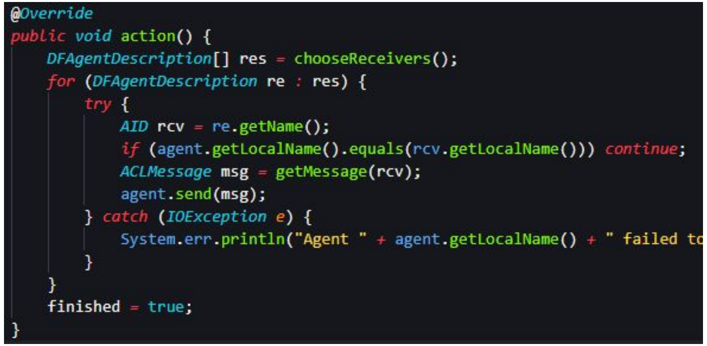
In terms of behaviours, 2 main classes were implemented and are extended by other to fulfill their purpose.



```
@Override
public void action() {
    ACLMessage msg = agent.receive();
    if (msg != null) {
        String sender = msg.getSender().getLocalName();
        if (sender.startsWith("audience")) {
            agent.parseAudienceMsg(msg);
        } else if (sender.startsWith("competitor")) {
            agent.parseCompetitorMsg(msg);
        } else if (sender.startsWith("world")) {
            agent.parseWorldMsg(msg);
        }
    } else {
        block();
    }
}
```

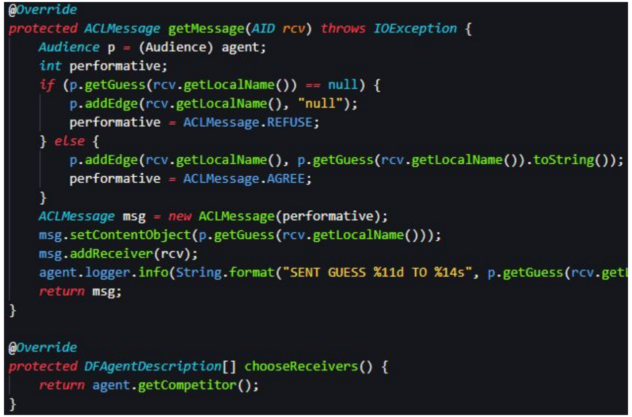
ReceiveMsgBehaviour: its objective is to receive all messages and parse them according to their sender. Each agent has different needs and will implement the parse methods in different ways.

SendMsgBehaviour: this abstract behaviour sends a message created in *getMessage()* to all agents returned by *chooseReceivers()*. After sending all messages it finishes.



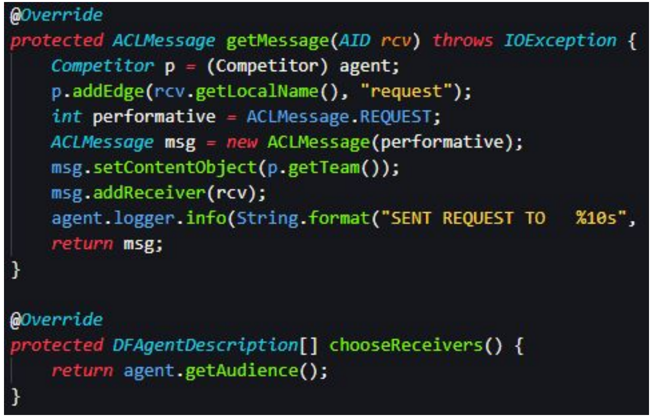
```
@Override
public void action() {
    DFAgentDescription[] res = chooseReceivers();
    for (DFAgentDescription re : res) {
        try {
            AID rcv = re.getName();
            if (agent.getLocalName().equals(rcv.getLocalName())) continue;
            ACLMessage msg = getMessage(rcv);
            agent.send(msg);
        } catch (IOException e) {
            System.err.println("Agent " + agent.getLocalName() + " failed to
        }
    }
    finished = true;
}
```

Behaviours



```java
@Override
protected ACLMessage getMessage(AID rcv) throws IOException {
    Audience p = (Audience) agent;
    int performative;
    if (p.getGuess(rcv.getLocalName()) == null) {
        p.addEdge(rcv.getLocalName(), "null");
        performative = ACLMessage.REFUSE;
    } else {
        p.addEdge(rcv.getLocalName(), p.getGuess(rcv.getLocalName()).toString());
        performative = ACLMessage.AGREE;
    }
    ACLMessage msg = new ACLMessage(performative);
    msg.setContentObject(p.getGuess(rcv.getLocalName()));
    msg.addReceiver(rcv);
    agent.logger.info(String.format("SENT GUESS %11d TO %14s", p.getGuess(rcv.getL
    return msg;
}

@Override
protected DFAgentDescription[] chooseReceivers() {
    return agent.getCompetitor();
}
```

Audience sends guess to Competitor



```java
@Override
protected ACLMessage getMessage(AID rcv) throws IOException {
    Competitor p = (Competitor) agent;
    p.addEdge(rcv.getLocalName(), "request");
    int performative = ACLMessage.REQUEST;
    ACLMessage msg = new ACLMessage(performative);
    msg.setContentObject(p.getTeam());
    msg.addReceiver(rcv);
    agent.logger.info(String.format("SENT REQUEST TO   %10s",
    return msg;
}

@Override
protected DFAgentDescription[] chooseReceivers() {
    return agent.getAudience();
}
```

Competitor sends query to Audience

World

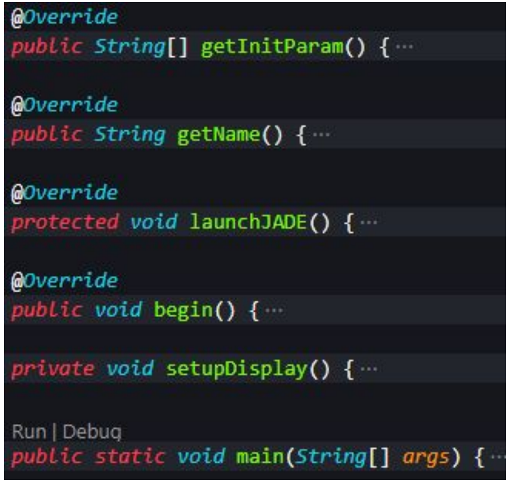
Inside the world package we have the classes that orchestrate the whole system.

WorldModel: contains the main function and stores the parameters. It is responsible for initializing and terminating the whole system, including the displays for graphic mode.

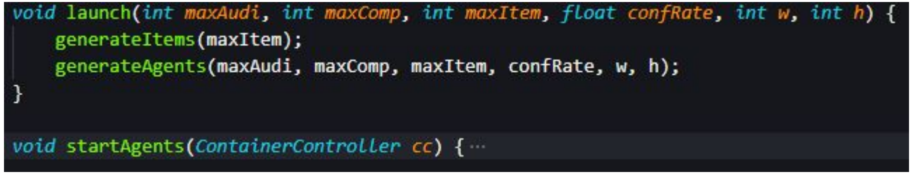
World: this class is responsible for initializing all the agents and starting and ending each round. It stores all the data relevant to the process.

WorldData: manages data that is needed to display on screen or stored in log/csv files.

WorldAgent: not inside the world package but its use is relevant to it since it is responsible for messaging all agents at the start and end of each round.

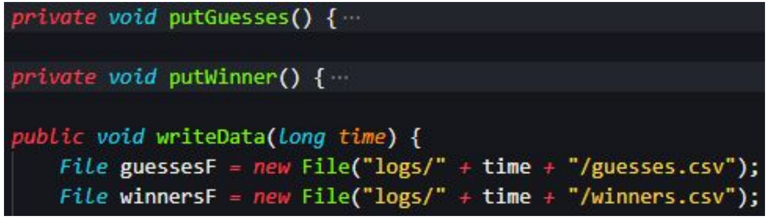


```
@Override
public String[] getInitParam() {⋯

@Override
public String getName() {⋯

@Override
protected void launchJADE() {⋯

@Override
public void begin() {⋯

private void setupDisplay() {⋯

Run | Debug
public static void main(String[] args) {⋯
```



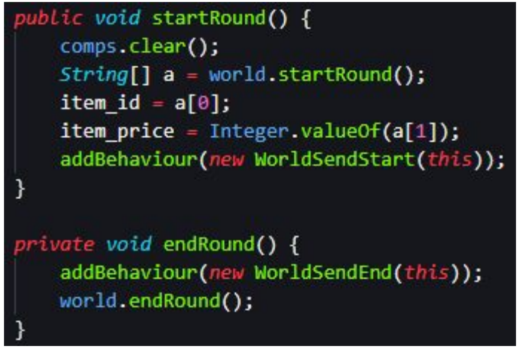
```
void launch(int maxAudi, int maxComp, int maxItem, float confRate, int w, int h) {
    generateItems(maxItem);
    generateAgents(maxAudi, maxComp, maxItem, confRate, w, h);
}

void startAgents(ContainerController cc) {⋯
```

World



```
private void putGuesses() {···

private void putWinner() {···

public void writeData(long time) {
    File guessesF = new File("logs/" + time + "/guesses.csv");
    File winnersF = new File("logs/" + time + "/winners.csv");
```

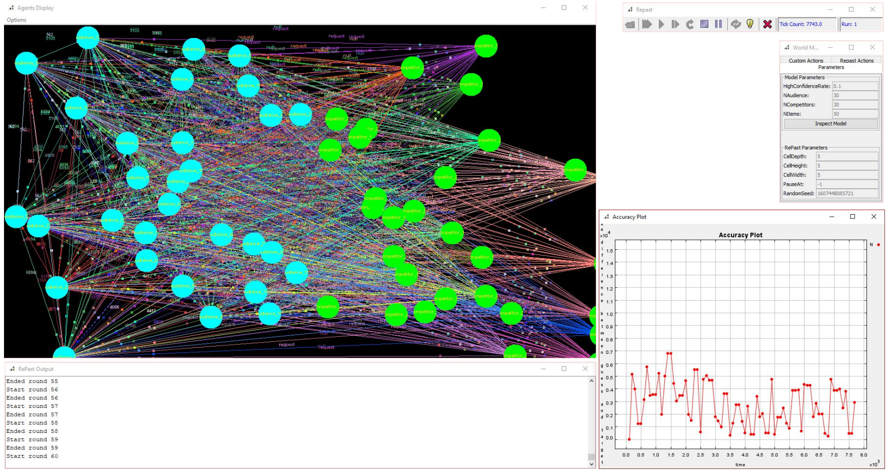
WorldData methods



```
public void startRound() {
    comps.clear();
    String[] a = world.startRound();
    item_id = a[0];
    item_price = Integer.valueOf(a[1]);
    addBehaviour(new WorldSendStart(this));
}

private void endRound() {
    addBehaviour(new WorldSendEnd(this));
    world.endRound();
}
```

WorldAgent methods

Example of Execution



Tick Count: 7743.0 | Run: 1

Model Parameters

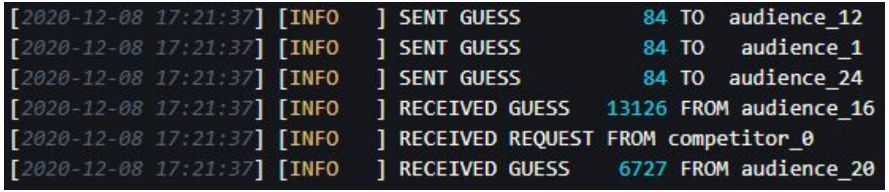
HighConfidenceRate:	0.1
NAudience:	30
NCompetitors:	30
NItems:	50

Inspect Model

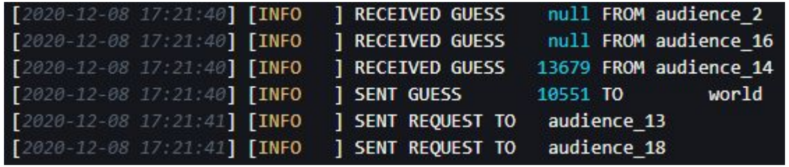
RePast Parameters

CellDepth:	5
CellHeight:	5
CellWidth:	5
PauseAt:	-1
RandomSeed:	1607448085721

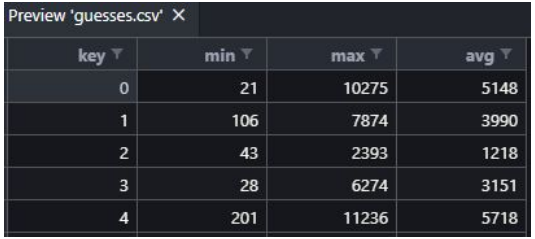
```
Ended round 55
Start round 56
Ended round 56
Start round 57
Ended round 57
Start round 58
Ended round 58
Start round 59
Ended round 59
Start round 60
```



```
[2020-12-08 17:21:37] [INFO   ] SENT GUESS          84 TO  audience_12
[2020-12-08 17:21:37] [INFO   ] SENT GUESS          84 TO   audience_1
[2020-12-08 17:21:37] [INFO   ] SENT GUESS          84 TO  audience_24
[2020-12-08 17:21:37] [INFO   ] RECEIVED GUESS   13126 FROM audience_16
[2020-12-08 17:21:37] [INFO   ] RECEIVED REQUEST FROM competitor_0
[2020-12-08 17:21:37] [INFO   ] RECEIVED GUESS    6727 FROM audience_20
```

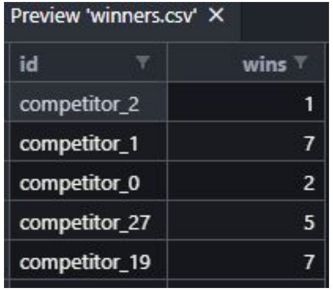


```
[2020-12-08 17:21:40] [INFO   ] RECEIVED GUESS    null FROM audience_2
[2020-12-08 17:21:40] [INFO   ] RECEIVED GUESS    null FROM audience_16
[2020-12-08 17:21:40] [INFO   ] RECEIVED GUESS   13679 FROM audience_14
[2020-12-08 17:21:40] [INFO   ] SENT GUESS       10551 TO        world
[2020-12-08 17:21:41] [INFO   ] SENT REQUEST TO   audience_13
[2020-12-08 17:21:41] [INFO   ] SENT REQUEST TO   audience_18
```



Preview 'guesses.csv' ✕

key	min	max	avg
0	21	10275	5148
1	106	7874	3990
2	43	2393	1218
3	28	6274	3151
4	201	11236	5718



Preview 'winners.csv' ✕

id	wins
competitor_2	1
competitor_1	7
competitor_0	2
competitor_27	5
competitor_19	7