U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

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

The agents in this project play a version of *The Price Is Right.* There are two types of agents: audience and competitors.

In the beginning, each element of the audience is told the price of a few items. At the start of a round, an item is chosen and the audience guesses the first time. Then, the audience shares their guess with each other and each agent may change their initial guess. After everyone in the audience has a guess, the competitors ask the audience their guess. If an element of the audience likes a competitor, they will send their guess, otherwise, they refuse the request. The competitors then have their chance to take a guess. The competitor with the closest guess wins.

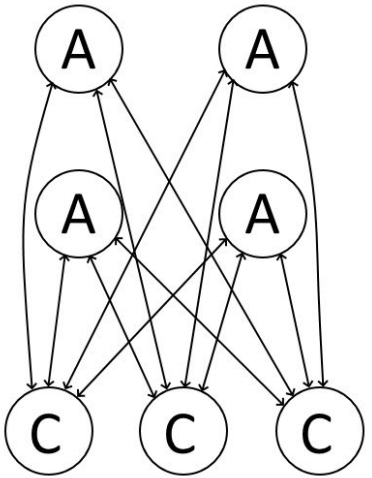
After each round, each agent (audience and competitors) will change their confidence in other agents based on their accuracy in the previous round. They can play with any number of rounds, audience members, competitors and items.

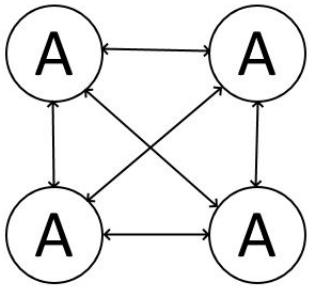
Global Scheme

There are two types of agents: Audience and Competitor.

In every round of the game, the agents go through two types of interactions:
- Firstly, each element of the audience interacts with all others.
- Then, the audience communicates with the competitors.

The audience decides an initial guess, share with others.
Then, they are asked their guess by the competitors,
and send them, if they choose so.





Interactions and Protocols

For the agents to communicate with each other, I decided to use these protocols:

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

Types of interaction:

Audience - Audience:
- A1 sends A2 his initial guess, A2 sends A1 his initial guess.

Audience - Competitor:
- C sends A a request for a guess, this request contains its team affinity;
- If A has similar team affinity to C, then A sends C its guess, else it refuses the request.

Mechanisms

For the development of this project we used different methods to put it all together. One of those was the use of Yellow Pages to allow agents to search for other agents and send them messages.

```java
protected void setupAgent(String type) {
    try {
        ServiceDescription sd = new ServiceDescription();
        sd.setType(type);
        sd.setName(getLocalName());
        dfd.setName(getAID());
        dfd.addServices(sd);
        DFService.register(this, dfd);
    } catch (FIPAException e) {
        logger.severe("Exception thrown while setting up " + id + ".");
        e.printStackTrace();
        System.exit(3);
    }
}
```

Method that registers an agent

```java
DFAgentDescription[] getService(String type) {
    DFAgentDescription[] res = null;
    try {
        DFAgentDescription dfd = new DFAgentDescription();
        ServiceDescription sd = new ServiceDescription();
        sd.setType(type);
        dfd.addServices(sd);
        res = DFService.search(this, dfd);
    } catch (FIPAException e) {
        logger.severe("Exception thrown while getting service " + type + ".");
        e.printStackTrace();
        System.exit(3);
    }
    return res;
}
```

Method that allows any agent to search for all agents with a service of *type*

Architecture and Strategies

Audience

Item Knowledge: When they are created, each audience agent is given the price of a random number of items, this will help them in making a better initial guess.

Initial guess: At the start of the round, the audience is told the chosen item. Then each agent will try and guess its price. If they know that 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: During the round, 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.

Architecture and Strategies

Competitor

Final guess: At the end of the round, after receiving the guesses from the audience, each competitor will make a guess. This guess is based on the guesses of the agents in the audience and the confidence the competitor has in each of those agents.

Person

Team affinity: To each agent, it is given a random number, from 0 to 100, for each category of teams, this is their team affinity. Audience will refuse requests from Competitors where two or more categories have a difference above 20.

Confidence: Initially each agent has a random confidence level in each agent of the audience. Some audience agents will have extraordinary self-confidence (based on highConfidenceRate argument). After each round, depending on how close the agent was to the real price, their confidence level will rise or fall to the other agents.

Experiments and Analysis

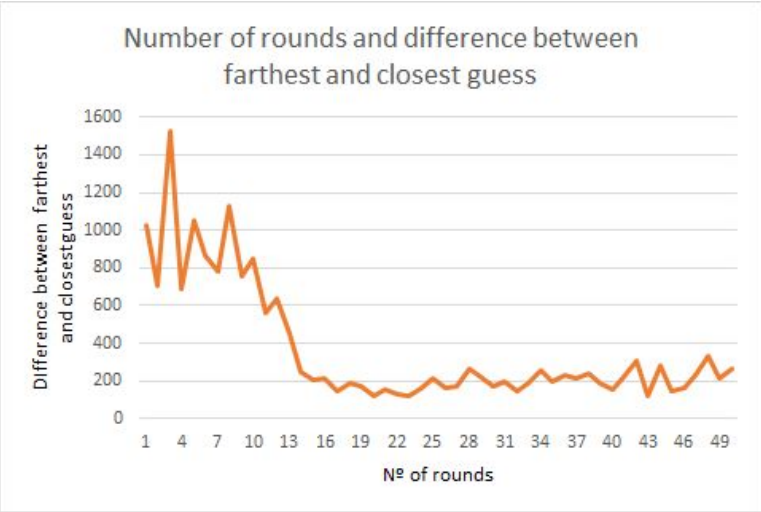
To test the limits of the project and to satisfy a lingering curiosity, a few tests were run.

Firstly, as expected, the bigger the audience, the slower the program progresses. This is because each agent of the audience has to send and receive messages to and from all other agents of the audience, and then each competitor has to send a request and receive a response from each audience member.

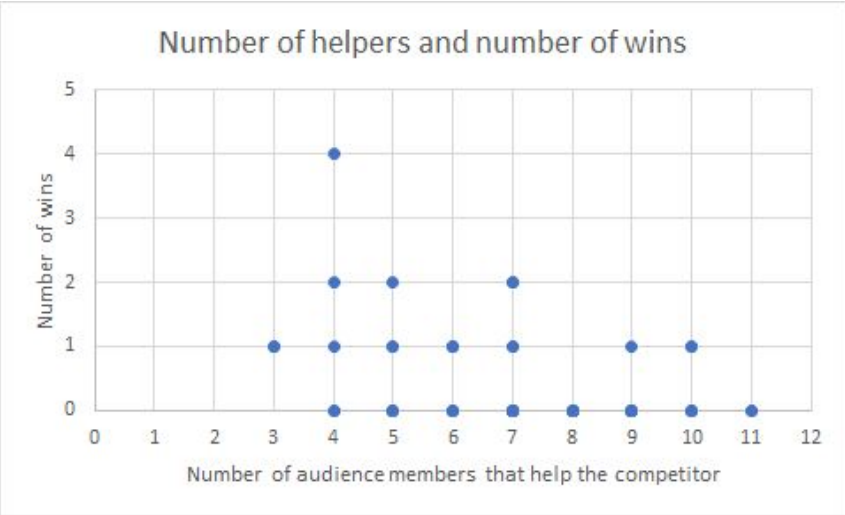
We can also say the more rounds we add, the more the agents improve. Even though they still face difficulties finding the price with accuracy, the farthest and closest guesses from the real price are practically the same after 15 rounds.

Based on tests, we can see that, the more agents from the audience help a competitor, the worse it gets at guessing an item’s price.

Experiments and Analysis



Line chart comparing the difference between the closest and farthest guesses (y-axis) to the round number (x-axis)



Scatter chart comparing wins (y-axis) to the number of audience members that send a guess to the competitor

Conclusions

Even though the project has most of the mechanics that were planned, there are some that could have been added:

- Agents being able to gain more or less confidence based on their team or a new variable;
- Agents being added, removed or changing roles.

All in all, this project allowed me to learn more about systems using multiple agents and how these agents interact with each other. It also further improved my experience with java and its libraries.

Annexes

The next few slides will be dedicated to show the implementation of the classes, execution examples and other observations.

Implemented Classes

Person: abstract class for the agents

```
final String id;
final HashMap<String, Integer> teamAffinity;
final HashMap<String, Integer> guesses;
final HashMap<String, Float> confidence;
final DFAgentDescription dfd;
boolean ready;
Integer guess;
```

Person class fields

```
public void startConfidence() {
    Random rnd = new Random();
    DFAgentDescription[] aud = getAudience();

    for (DFAgentDescription a : aud) {
        String id = a.getName().getLocalName();
        if (id.equals(getLocalName())) continue;
        confidence.putIfAbsent(id, rnd.nextFloat() + 0.2f);
    }
}
```

Method used to initialize the confidence of one agent towards all audience agents

```
void updateConfidence(int price) {
    guesses.values().removeIf(Objects::isNull);
    if (guesses.isEmpty()) return;

    Integer min = Collections.min(guesses.values());
    Integer max = Collections.max(guesses.values());
    int maxDiff = Math.max(Math.abs(price - min), Math.abs(price - max));

    for (Map.Entry<String, Float> entry : confidence.entrySet()) {
        if (guesses.get(entry.getKey()) == null) continue;
        int diff = Math.abs(guesses.get(entry.getKey()) - price);
        confidence.replace(entry.getKey(), entry.getValue() * (1.5f - diff / maxDiff));
    }
}
```

Method used to change the confidence of one agent towards audience members

Implemented Classes

Person

```
float[] finalGuessCalc() {
    guesses.values().removeIf(Objects::isNull);

    float maxConfidence = 0.0f;
    float currentGuess = 0.0f;
    Random rnd = new Random();

    for (Map.Entry<String, Integer> entry : guesses.entrySet()) {
        confidence.putIfAbsent(entry.getKey(), rnd.nextFloat() + 0.2f);
        currentGuess += entry.getValue() * confidence.get(entry.getKey());
        maxConfidence += confidence.get(entry.getKey());
    }
    float[] a = new float[2];
    a[0] = currentGuess;
    a[1] = maxConfidence;

    return a;
}
```

Method that calculates a guess based on all guesses received from the audience and their confidence

Competitor

```
@Override
void behaviours() {
    SequentialBehaviour sb = new SequentialBehaviour();
    sb.addSubBehaviour(new CompetitorSendRequest(this));
    sb.addSubBehaviour(new CompetitorReceiveGuess(this));
    addBehaviour(sb);
}
```

Method that sets the competitor's behaviours

Implemented Classes

Audience

```
public void checkCompetitor(String id, HashMap<String, Integer> map) {
    int comp = 0;
    for (Map.Entry<String, Integer> entry : teamAffinity.entrySet()) {
        comp += Math.abs(entry.getValue() - map.get(entry.getKey()));
    }
    compatibility.put(id, comp);
}
```

Method which calculates the compatibility of two agents

```
@Override
void behaviours() {
    SequentialBehaviour sb = new SequentialBehaviour();
    sb.addSubBehaviour(new AudienceShareGuess(this));
    sb.addSubBehaviour(new AudienceReceiveGuess(this));
    sb.addSubBehaviour(new AudienceReceiveRequest(this));
    sb.addSubBehaviour(new AudienceSendGuess(this));
    addBehaviour(sb);
}
```

Method that sets the audience's behaviours

```
void initialGuess(String item) {
    Random rnd = new Random();
    if (itemPrice.get(item) != null) {
        int p = itemPrice.get(item);
        guess = p + rnd.nextInt((int) (p * 0.2f)) - (int) (p * 0.1f);
    } else {
        guess = rnd.nextInt(15001);
    }
}
```

Method that calculates the initial guess of the audience

Implemented Classes

SendMsgBehaviour: abstract class for behaviours that send messages

```java
@Override
public void action() {
    DFAgentDescription[] res = chooseReceivers();

    for (DFAgentDescription re : res) {
        try {
            AID rcv = re.getName();
            if (person.getLocalName().equals(rcv.getLocalName())) continue;
            ACLMessage msg = getMessage(rcv);
            person.send(msg);
        } catch (IOException e) {
            person.logger.warning("Agent " + person.getLocalName() + " failed
        }
    }

    finished = true;
}
```

Method called when behaviour is active, sends message returned by *getMessage()* to all agents returned by *chooseReceivers()*

```java
@Override
protected ACLMessage getMessage(AID rcv) throws IOException {
    Audience p = (Audience) person;
    int performative;
    if (p.getGuess(rcv.getLocalName()) == null) performative = ACLMessage.REFUSE;
    else performative = ACLMessage.AGREE;
    ACLMessage msg = new ACLMessage(performative);
    msg.setContentObject(p.getGuess(rcv.getLocalName()));
    msg.addReceiver(rcv);
    person.logger.info(String.format("AUDIENCE   %10s SENT GUESS     %7d TO   %10s"
    return msg;
}
```

Example *getMessage()* from AudienceSendGuess

Implemented Classes

ReceiveMsgBehaviour: abstract class for behaviours that receive messages

```java
@Override
public void action() {
    ACLMessage msg = person.blockingReceive();
    if (msg != null) {
        if (msg.getSender().getLocalName().startsWith("audience")) {
            parseAudienceMsg(msg);
        } else if (msg.getSender().getLocalName().startsWith("competitor")) {
            parseCompetitorMsg(msg);
        }
    } else {
        block();
    }

    if (finishCondition()) {
        finish();
        finished = true;
    }
}
```

Method called when behaviour is active, receives a message and treats it differently depending on the sender. The behaviour terminates and runs an exiting method if the finishing condition is achieved

```java
@Override
protected void parseAudienceMsg(ACLMessage msg) {
    String sender = msg.getSender().getLocalName();
    if (person.getGuesses().containsKey(sender)) return;

    try {
        Integer guess = (Integer) msg.getContentObject();
        person.logger.info(String.format("AUDIENCE   %10s R
        person.receiveGuess(sender, guess);
    } catch (UnreadableException e) {
        person.logger.severe("Exception thrown while receiv
        e.printStackTrace();
        System.exit(3);
    }
}
```

Example *parseAudienceMsg()* from AudienceReceiveGuess

Implemented Classes

World: main class, contains the logic of the game

```java
private void playRound(int round) {
    // 1. Select item
    Random rnd = new Random();
    String item_id = Integer.toString(rnd.nextInt(maxItems));

    LOGGER.info("Item selected: " + item_id);

    // 2. Tell item to audience
    for (Audience au : audience) {
        au.startRound(item_id);
    }

    // Wait audience to decide guesses
    HashSet<String> readyAudience = new HashSet<>();
    while (readyAudience.size() < audience.size()) {
        for (Audience au : audience) {
            if (au.ready) {
                readyAudience.add(au.getLocalName());
            }
        }
    }

    LOGGER.info("Audience finished guessing.");

    // 3. Competitors send request
    for (Competitor cm : competitors) {
        cm.startRound();
    }
```

```java
    // 4. Wait for competitors
    HashMap<String, Integer> guesses = new HashMap<>();
    while (guesses.size() < competitors.size()) {
        for (Competitor cm : competitors) {
            if (cm.ready) {
                guesses.putIfAbsent(cm.getLocalName(), cm.getGuess());
            }
        }
    }

    LOGGER.info("Competitors finished guessing.");

    // 5. Declare winner
    String winner = declareWinner(item_id, guesses);
    LOGGER.info("The winner was: " + winner);

    updateRoundPrices(round, itemPrice.get(item_id), guesses);

    // 6. End Round
    for (Audience au : audience) {
        au.endRound(itemPrice.get(item_id));
    }
    for (Competitor cm : competitors) {
        cm.endRound(itemPrice.get(item_id));
    }
}
```

Method *playRound()* which controls the flow of each round

Execution Examples

Sending and receiving guesses

```
[2020-11-15 14:25:21] [INFO   ] AUDIENCE   audience_0 SENT GUESS        6730 TO   audience_1
[2020-11-15 14:25:21] [INFO   ] AUDIENCE   audience_0 SENT GUESS        6730 TO   audience_3
[2020-11-15 14:25:21] [INFO   ] AUDIENCE   audience_0 SENT GUESS        6730 TO   audience_5
[2020-11-15 14:25:21] [INFO   ] AUDIENCE   audience_0 SENT GUESS        6730 TO   audience_8
[2020-11-15 14:25:21] [INFO   ] AUDIENCE   audience_0 SENT GUESS        6730 TO   audience_2
[2020-11-15 14:25:21] [INFO   ] AUDIENCE   audience_0 SENT GUESS        6730 TO   audience_4
[2020-11-15 14:25:21] [INFO   ] AUDIENCE   audience_0 SENT GUESS        6730 TO   audience_6
[2020-11-15 14:25:21] [INFO   ] AUDIENCE   audience_0 SENT GUESS        6730 TO   audience_9
[2020-11-15 14:25:21] [INFO   ] AUDIENCE   audience_0 SENT GUESS        6730 TO   audience_7
[2020-11-15 14:25:21] [INFO   ] AUDIENCE   audience_0 RECEIVED GUESS    4732 FROM audience_3
[2020-11-15 14:25:21] [INFO   ] AUDIENCE   audience_0 RECEIVED GUESS    7560 FROM audience_2
[2020-11-15 14:25:21] [INFO   ] AUDIENCE   audience_0 RECEIVED GUESS    5514 FROM audience_6
[2020-11-15 14:25:21] [INFO   ] AUDIENCE   audience_0 RECEIVED GUESS    1443 FROM audience_5
[2020-11-15 14:25:21] [INFO   ] AUDIENCE   audience_0 RECEIVED GUESS    7011 FROM audience_8
[2020-11-15 14:25:21] [INFO   ] AUDIENCE   audience_0 RECEIVED GUESS    5264 FROM audience_1
[2020-11-15 14:25:21] [INFO   ] AUDIENCE   audience_0 RECEIVED GUESS   14426 FROM audience_7
[2020-11-15 14:25:21] [INFO   ] AUDIENCE   audience_0 RECEIVED GUESS   11324 FROM audience_4
[2020-11-15 14:25:21] [INFO   ] AUDIENCE   audience_0 RECEIVED GUESS    8114 FROM audience_9
```

Audience agent sending a guess to rest of the audience and receiving their guess back

```
[2020-11-15 14:25:23] [INFO   ] AUDIENCE   audience_0 RECEIVED REQUEST       FROM competitor_1
[2020-11-15 14:25:23] [INFO   ] AUDIENCE   audience_0 RECEIVED REQUEST       FROM competitor_0
[2020-11-15 14:25:24] [INFO   ] AUDIENCE   audience_0 RECEIVED REQUEST       FROM competitor_3
[2020-11-15 14:25:24] [INFO   ] AUDIENCE   audience_0 RECEIVED REQUEST       FROM competitor_4
[2020-11-15 14:25:24] [INFO   ] AUDIENCE   audience_0 RECEIVED REQUEST       FROM competitor_2
[2020-11-15 14:25:24] [INFO   ] AUDIENCE   audience_0 SENT GUESS        null TO   competitor_0
[2020-11-15 14:25:24] [INFO   ] AUDIENCE   audience_0 SENT GUESS        9400 TO   competitor_2
[2020-11-15 14:25:24] [INFO   ] AUDIENCE   audience_0 SENT GUESS        9400 TO   competitor_4
[2020-11-15 14:25:24] [INFO   ] AUDIENCE   audience_0 SENT GUESS        9400 TO   competitor_1
[2020-11-15 14:25:24] [INFO   ] AUDIENCE   audience_0 SENT GUESS        9400 TO   competitor_3
```

Audiences receiving requests from competitors and responding with guess (null in the case of competitor_0 because they do not have similar teams)

Execution Examples

Sending requests and receiving guesses.

```
[2020-11-15 14:25:21] [INFO   ] COMPETITOR competitor_0 SENT REQUEST          TO   audience_1
[2020-11-15 14:25:21] [INFO   ] COMPETITOR competitor_0 SENT REQUEST          TO   audience_3
[2020-11-15 14:25:21] [INFO   ] COMPETITOR competitor_0 SENT REQUEST          TO   audience_5
[2020-11-15 14:25:21] [INFO   ] COMPETITOR competitor_0 SENT REQUEST          TO   audience_8
[2020-11-15 14:25:21] [INFO   ] COMPETITOR competitor_0 SENT REQUEST          TO   audience_2
[2020-11-15 14:25:21] [INFO   ] COMPETITOR competitor_0 SENT REQUEST          TO   audience_4
[2020-11-15 14:25:21] [INFO   ] COMPETITOR competitor_0 SENT REQUEST          TO   audience_6
[2020-11-15 14:25:21] [INFO   ] COMPETITOR competitor_0 SENT REQUEST          TO   audience_9
[2020-11-15 14:25:21] [INFO   ] COMPETITOR competitor_0 SENT REQUEST          TO   audience_7
[2020-11-15 14:25:21] [INFO   ] COMPETITOR competitor_0 SENT REQUEST          TO   audience_0
[2020-11-15 14:25:21] [INFO   ] COMPETITOR competitor_0 RECEIVED GUESS    5280 FROM audience_1
[2020-11-15 14:25:21] [INFO   ] COMPETITOR competitor_0 RECEIVED GUESS    null FROM audience_5
[2020-11-15 14:25:21] [INFO   ] COMPETITOR competitor_0 RECEIVED GUESS    7350 FROM audience_8
[2020-11-15 14:25:21] [INFO   ] COMPETITOR competitor_0 RECEIVED GUESS    null FROM audience_9
[2020-11-15 14:25:21] [INFO   ] COMPETITOR competitor_0 RECEIVED GUESS    7033 FROM audience_4
[2020-11-15 14:25:21] [INFO   ] COMPETITOR competitor_0 RECEIVED GUESS    6732 FROM audience_0
[2020-11-15 14:25:21] [INFO   ] COMPETITOR competitor_0 RECEIVED GUESS    7190 FROM audience_2
[2020-11-15 14:25:21] [INFO   ] COMPETITOR competitor_0 RECEIVED GUESS    7759 FROM audience_3
[2020-11-15 14:25:21] [INFO   ] COMPETITOR competitor_0 RECEIVED GUESS    5525 FROM audience_6
[2020-11-15 14:25:21] [INFO   ] COMPETITOR competitor_0 RECEIVED GUESS   14374 FROM audience_7
```

Competitor sending requests and receiving guesses
(null in case request refused)

World logs

```
[2020-11-15 14:25:21] [INFO   ] World created
[2020-11-15 14:25:21] [INFO   ] Start round 0
[2020-11-15 14:25:21] [INFO   ] Item selected: 12
[2020-11-15 14:25:21] [INFO   ] Audience finished guessing.
[2020-11-15 14:25:21] [INFO   ] Competitors finished guessing.
[2020-11-15 14:25:21] [INFO   ] The winner was: competitor_0
[2020-11-15 14:25:21] [INFO   ] End round 0
[2020-11-15 14:25:21] [INFO   ] Start round 1
[2020-11-15 14:25:21] [INFO   ] Item selected: 1
[2020-11-15 14:25:22] [INFO   ] Audience finished guessing.
[2020-11-15 14:25:22] [INFO   ] Competitors finished guessing.
[2020-11-15 14:25:22] [INFO   ] The winner was: competitor_4
[2020-11-15 14:25:22] [INFO   ] End round 1
[2020-11-15 14:25:22] [INFO   ] Start round 2
[2020-11-15 14:25:22] [INFO   ] Item selected: 5
[2020-11-15 14:25:22] [INFO   ] Audience finished guessing.
[2020-11-15 14:25:22] [INFO   ] Competitors finished guessing.
[2020-11-15 14:25:22] [INFO   ] The winner was: competitor_0
[2020-11-15 14:25:22] [INFO   ] End round 2
```

Logs from World, showing the order of actions

Observations

To run this project you need to download the JADE framework.

Then, compile the project with:

javac -d "out/" -sourcepath "src/" -cp "path/to/jade.jar" src/world/World.java

With the class files generated, now run:

java -cp "out/;path/to/jade.jar" world.World <nAudience> <nCompetitors> <nItems>
<highConfidenceRate> <tries> <rounds>

Port 9090 must be unoccupied before running the project, either kill the process occupying it, or change the port in the code.