**KAUNO TECHNOLOGIJOS UNIVERSITETAS**

**INFORMATIKOS FAKULTETAS**

INTELEKTIKOS PAGRINDAI (P176B101)

**Projektas. Fuzzy logika pagrįstas sprendimas panaudojant intelektinius agentus.**

Atliko:

IFF-4/1 gr. studentai

Mangirdas Kazlauskas

Simonas Baltulionis

IFF-4/3 gr.studentas

Lukas Skamarakas

Priėmė:

Doc. Agnė Paulauskaitė-Tarasevičienė

**KAUNAS 2017**

Turinys

[1. Darbo užduotis 3](#_Toc483381331)

[2. Užduoties sprendimas 4](#_Toc483381332)

[2.1. Fuzzy logikos dalis 4](#_Toc483381333)

[2.1.1. Fuzzy logika 4](#_Toc483381334)

[2.1.2. Priklausomybės funkcijų sudarymas 4](#_Toc483381335)

[2.1.3. Taisyklių sudarymas 6](#_Toc483381336)

[2.1.4. Taisyklių skaičiavimai 8](#_Toc483381337)

[2.1.5. Agregacija 8](#_Toc483381338)

[2.1.6. Defuzifikacija 8](#_Toc483381339)

[2.2. Agentų dalis 8](#_Toc483381340)

[2.2.1. Sistemoje naudojami agentai ir jų paskirtis 8](#_Toc483381341)

[2.2.2. Agentų bendravimas 9](#_Toc483381342)

[3. Užduoties realizacija 9](#_Toc483381343)

[3.1. Konstantos 9](#_Toc483381344)

[3.2. Agentas A 11](#_Toc483381345)

[3.3. Agentas V 15](#_Toc483381346)

[3.4. Agentas SP 17](#_Toc483381347)

[4. Sistemos veikimas bei rezultatai 23](#_Toc483381348)

[4.1. Agentų bendravimas 23](#_Toc483381349)

[4.2. Sistemos veikimo rezultatai 25](#_Toc483381350)

[5. Išvados 25](#_Toc483381351)

# Darbo užduotis

Tarkime, keli draugai sugalvojo eiti į miestą pavakarieniauti. Prieš išsirenkant restoraną reikia nuspręsti ar tai brangus, prieinamos kainos ar pigus restoranas. Tam tikslui reikia sukurti sistemą, kuri padės atsakyti į išsikeltą klausimą ir kuri susideda iš trijų agentų: **Agentas A, Agentas V ir Agentas SP.**

Sistemoje įtrauktas **Agentas A** atlieka apklausą, siekdamas išsiaiškinti kas yra brangu ir nebrangu. **Agentas A** turi trijų patiekalų sąrašą, kurių kainų įvertis jam leidžia klasifikuoti restoranus. Tas sąrašas susideda iš tokių patiekalų/produktų: (1) juodos kavos puodelio kaina, (2) dienos sriubos kaina ir (3) jautienos kepsnio kaina. **Agentas A** komunikuoja su **Agentu V**, kuris pateikia atsakymus į užduotus jam klausimus. **Agentas A** klausia kokia kaina kiekvienam iš patiekalų yra brangu, normalu ir pigu. Pvz., vartotojas gali atsakyti, kad kava brangi >3EUR, normali kaina =2 EUR, pigi >1.5 EUR. Tokius klausimus jis užduoda **Agentui V**, kuris kiekvieną kartą imituoja skirtingus vartotojus su tam tikrais atsakymas. **Agentas A** yra nuolatos veikiantis agentas, kuris cikliškai užduoda tokius klausimus ir kas penktą atsakymą suformuoja/atnaujina Fuzzy priklausomybės funkcijas kiekvienam iš įėjimų (kava, sriuba, kepsnys). Kiekvienam įėjimui priskiriamos trys kategorijos, kurios aprašomos priklausomybės funkcijomis, tačiau tų funkcijų kintamųjų reikšmės kinta laike, nes su lauku didėja duomenų kiekis. Pvz.:

1 lentelė. Kavos kainos vidurkio reikšmės kitimo laike pavyzdys

|  |  |  |  |
| --- | --- | --- | --- |
| Klientas | Kavos kaina | | |
|  | Brangi? | Normali? | Pigi? |
| 1 | >3 | <2 | <1.5 |
| 2 | >4 | <3 | <2 |
| 3 | >2.5 | <1.5 | <1.1 |
| 4 | >2.2 | <1.5 | <1.3 |
| 5 | >3.5 | 2 | <1.8 |
| **Vidurkis** | **3.5** | **2.33** | **2.0** |
| 6 | >3.1 | <2 | <1.3 |
| 7 | **…** | **…** | **….** |

Pagal pateiktą lentelę galima nubrėžti priklausomybės funkcijas (trikampes, trapecines ir pan.), kadangi žinomos vidutinės, minimalios ir maksimalios funkcijų reikšmės.

**Agentas V** gavęs klausimą iš **Agento A,** imituoja skirtingus vartotojus, tačiau kas tam tikrą laiką, **Agentas V** imituoja ir turistą, kuriam pagal pateiktas trijų patiekalų kainas svarbu sužinoti koks tai restoranas, brangus, normalus ar pigus. Šį klausimą su input‘ų reikšmėmis jis užduoda **Agentui A**.

**Agentų funkcionalumai:**

**Agento A** funkcija yra sudaryti priklausomybių funkcijas visiems trims įėjimas ir jas kas tam tikrą laiką atnaujinti. Šiuos duomenis jis turi perduoti **Agentui SP**, kuris remiantis priklausomybės funkcijomis ir sudarytomis taisyklėmis (jas sudaro Agentas SP) paskaičiuoja atsakymą (output) ir perduoda atgal **Agentui A**. **Agentas V** yra ir vartotojas ir turistas. Jeigu Agentas A užduoda klausimus kas Δt, tai turistas inicijuojamas kas Δt\*s, kur

**Agentas A** nuolatos komunikuoja su **Agentu Vartotojas**.

**Agentas SP** komunikuoja tik su **Agentu A**.

# Užduoties sprendimas

Šiame skyriuje pateikiamas užduoties sprendimas, pasitelkiant Fuzzy logiką bei jos metodus, ir Fuzzy logikos panaudojimas realizuojant sistemą, pasitelkiant agentų paradigmą.

## Fuzzy logikos dalis

Užduoties sprendimui naudojama Fuzzy logika, todėl šiame skyriuje pateikiama platesnė informacija apie pačią Fuzzy logiką bei kaip ji buvo pritaikyta mūsų sprendžiamos užduoties kontekste.

### Fuzzy logika

Didžioji dalis tiksliai žinomų faktų gali būti aprašomi tam tikra kiekybine forma, pvz., akmuo sveria 10 kg, PVM tarifas Lietuvoje yra 21% ir pan. Tačiau dalis faktų yra apibrėžiami žodine forma, pvz., žmogus yra aukštas/žemas, jaunas/senas ir t.t. Kai kurių situacijų negalima vertinti griežtai (tiesa-melas), būtina naudoti aritmetines, žmogaus mąstymui artimesnes logikos rūšis. Vienas iš tokių logikos rūšių yra Fuzzy (miglotoji) logika, kai kažkokios tai situacijos „tiesos“ reikšmė gali svyruoti tarp 0 ir 1. Ši logika naudojama tada, kai savybės, kurių įverčiai pateikti miglotai, informacija nėra tiksli, negali būti vertinamos kategoriškai.

### Priklausomybės funkcijų sudarymas

Užduoties sąlygoje yra pateikta tam tikra informacija, reikalinga nustatyti restorano kokybę:

1. Juodos kavos puodelio kaina (€);
2. Dienos sriubos kaina (€);
3. Jautienos kepsnio kaina (€);

Šie kriterijai yra suskirstomi į tam tikrus miglotosios logikos lygmenis, kurie yra pateikti 2-4 lentelėse:

2 lentelė. Juodos kavos puodelio kainos Fuzzy lygmenys

|  |  |
| --- | --- |
| Kaina, (€) | Kavos kainos Fuzzy lygmuo |
| <2.50 | Cheap |
| 2.00-3.00 | Normal |
| >2.50 | Expensive |

3 lentelė. Dienos sriubos kainos Fuzzy lygmenys

|  |  |
| --- | --- |
| Kaina, (€) | Sriubos kainos Fuzzy lygmuo |
| <3.50 | Cheap |
| 3.00-4.50 | Normal |
| >4.00 | Expensive |

4 lentelė. Jautienos kepsnio kainos Fuzzy lygmenys

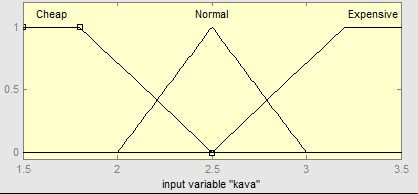
|  |  |
| --- | --- |
| Kaina, (€) | Kepsnio kainos Fuzzy lygmuo |
| <12.00 | Cheap |
| 8.00-15.00 | Normal |
| >13.00 | Expensive |

Pagal šias tris rodiklių grupes yra nusprendžiama, kokia bus restorano kokybė (5 lentelė):

5 lentelė. Restorano kokybės Fuzzy lygmenys

|  |  |
| --- | --- |
| Balas dešimtbalėje sistemoje | Kokybės Fuzzy lygmuo |
| <4 | Cheap |
| 2-8 | Affordable |
| >6 | Luxurious |

Remiantis 2-4 lentelių duomenimis buvo sudarytos įėjimo reikšmių priklausomybės funkcijos:



X3

X2

X1

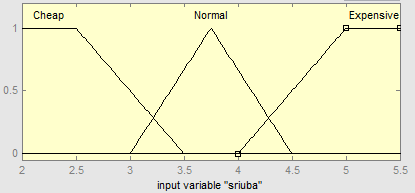
1 pav. Kavos kainos priklausomybės funkcijos

Kavos kainos priklausomybės funkcijų parametrai:

**Cheap** – [0 1.5 X1 2.5], kur X1 – laike kintanti pigios kainos kavos kintamojo reikšmė;

**Normal** – [2 X2 3], kur X2 – laike kintanti normalios kainos kavos kintamojo reikšmė;

**Expensive** – [2.5 X3 3.5 4], kur X3 – laike kintanti brangios kainos kavos kintamojo reikšmė.



X5

X4

X6

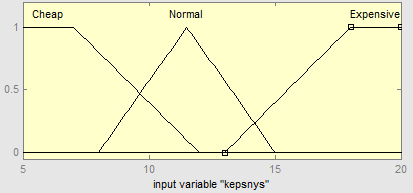
2 pav. Sriubos kainos priklausomybės funkcijos

Sriubos kainos priklausomybės funkcijų parametrai:

**Cheap** – [0 2 X4 3.5], kur X4 – laike kintanti pigios kainos sriubos kintamojo reikšmė;

**Normal** – [3 X5 4.5], kur X5 – laike kintanti normalios kainos sriubos kintamojo reikšmė;

**Expensive** – [4 X6 5.5 6], kur X6 – laike kintanti brangios kainos sriubos kintamojo reikšmė.



X9

X8

X7

3 pav. Kepsnio kainos priklausomybės funkcijos

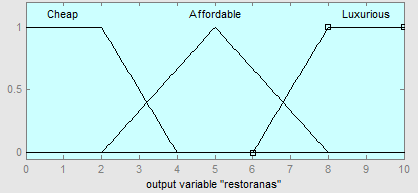
Kepsnio kainos priklausomybės funkcijų parametrai:

**Cheap** – [3 5 X7 12], kur X7 – laike kintanti pigios kainos kepsnio kintamojo reikšmė;

**Normal** – [8 X8 15], kur X8 – laike kintanti normalios kainos kepsnio kintamojo reikšmė;

**Expensive** – [13 X9 20 21], kur X9 – laike kintanti brangios kainos kepsnio kintamojo reikšmė.

Remiantis 5 lentelės duomenimis buvo sudarytos išėjimo reikšmių priklausomybės funkcijos:



4 pav. Restorano kokybės priklausomybės funkcijos

Restorano kokybės priklausomybės funkcijų parametrai:

**Cheap** – [-1 0 2 4];

**Affordable** – [2 5 8];

**Luxurious** – [6 8 10 11].

### Taisyklių sudarymas

Sudarius įvesties ir išvesties kintamųjų funkcijas, sudaromos taisyklės, kurios susieja įėjimo kintamuosius su išėjimo kintamuoju. Visi galimi taisyklių variantai pateikti 6 lentelėje:

6 lentelė. Visų galimų taisyklių rinkinys

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Nr. | Kavos kaina |  | Sriubos kaina |  | Kepsnio kaina | Restoranas |
| 1. | Cheap | *and* | Cheap | *and* | Cheap | Cheap | |
| 2. | Cheap | *and* | Cheap | *and* | Normal | Cheap | |
| 3. | Cheap | *and* | Cheap | *and* | Expensive | Affordable | |
| 4. | Cheap | *and* | Normal | *and* | Cheap | Cheap | |
| 5. | Cheap | *and* | Normal | *and* | Normal | Affordable | |
| 6. | Cheap | *and* | Normal | *and* | Expensive | Affordable | |
| 7. | Cheap | *and* | Expensive | *and* | Cheap | Cheap | |
| 8. | Cheap | *and* | Expensive | *and* | Normal | Luxurious | |
| 9. | Cheap | *and* | Expensive | *and* | Expensive | Luxurious | |
| 10. | Normal | *and* | Cheap | *and* | Cheap | Cheap | |
| 11. | Normal | *and* | Cheap | *and* | Normal | Cheap | |
| 12. | Normal | *and* | Cheap | *and* | Expensive | Affordable | |
| 13. | Normal | *and* | Normal | *and* | Cheap | Cheap | |
| 14. | Normal | *and* | Normal | *and* | Normal | Affordable | |
| 15. | Normal | *and* | Normal | *and* | Expensive | Luxurious | |
| 16. | Normal | *and* | Expensive | *and* | Cheap | Affordable | |
| 17. | Normal | *and* | Expensive | *and* | Normal | Affordable | |
| 18. | Normal | *and* | Expensive | *and* | Expensive | Luxurious | |
| 19. | Expensive | *and* | Cheap | *and* | Cheap | Cheap | |
| 20. | Expensive | *and* | Cheap | *and* | Normal | Affordable | |
| 21. | Expensive | *and* | Cheap | *and* | Expensive | Affordable | |
| 22. | Expensive | *and* | Normal | *and* | Cheap | Affordable | |
| 23. | Expensive | *and* | Normal | *and* | Normal | Luxurious | |
| 24. | Expensive | *and* | Normal | *and* | Expensive | Luxurious | |
| 25. | Expensive | *and* | Expensive | *and* | Cheap | Affordable | |
| 26. | Expensive | *and* | Expensive | *and* | Normal | Luxurious | |
| 27. | Expensive | *and* | Expensive | *and* | Expensive | Luxurious | |

Nemažai taisyklių galima apjungti tarpusavyje, todėl sistemos realizavime sudaromas bei naudojamas optimizuotas 13-kos taisyklių variantas (7 lentelė):

7 lentelė. Optimizuotas taisyklių rinkinys

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Nr. | Kavos kaina |  | Sriubos kaina |  | Kepsnio kaina | Restoranas |
| 1. | Cheap | *and* | NOT(Cheap) | *and* | Cheap | Cheap | |
| 2. | Normal | *and* | NOT(Expensive) | *and* | Cheap | Cheap | |
| 3. | NOT(Expensive) | *and* | Cheap | *and* | Normal | Cheap | |
| 4. | NOT(Normal) | *and* | Cheap | *and* | Cheap | Cheap | |
| 5. | Cheap | *and* | NOT(Expensive) | *and* | Expensive | Affordable | |
| 6. | Expensive | *and* | Cheap | *and* | NOT(Cheap) | Affordable | |
| 7. | Expensive | *and* | NOT(Cheap) | *and* | Cheap | Affordable | |
| 8. | Normal | *and* | Cheap | *and* | Expensive | Affordable | |
| 9. | Normal | *and* | Expensive | *and* | NOT(Expensive) | Affordable | |
| 10. | NOT(Expensive) | *and* | Normal | *and* | Normal | Affordable | |
| 11. | Cheap | *and* | Expensive | *and* | NOT(Cheap) | Luxurious | |
| 12. | Expensive | *and* | NOT(Cheap) | *and* | NOT(Cheap) | Luxurious | |
| 13. | Normal | *and* | NOT(Cheap) | *and* | Expensive | Luxurious | |

### Taisyklių skaičiavimai

Visos optimizuotos taisyklės tarpusavyje jungiamos logine operacija *and*, todėl taisyklės išėjimams skaičiuoti naudojamas *min* metodas, kurio išėjimo reikšmė yra lygi visų taisyklę sudarančių teiginių minimaliam teisingumui:

### Agregacija

Agregacijos metu Fuzzy aibės, kurios atvaizduoja kiekvienos taisyklės išėjimus, yra sujungiamos į vieną Fuzzy aibę. Šiam veiksmui atlikti pasirinktas *max* metodas, kai agreguotos funkcijos vertė taške y lygi didžiausiai iš visų aktyvuotų priklausomumo funkcijų verčių tame pačiame taške y.

### Defuzifikacija

Defuzifikacija – iš agreguotosios išvadų priklausomumo funkcijos m(y) (gauta po agregacijos) y skaitinės vertės radimas. Defuzifikacija gali būti atliekama keliais būdais, tačiau pasirinktas *centroid* metodas, kurio esmė – paskaičiuoti svorio centrą plotui, esančiam žemiau kreivės. *Centroid* metodo formulė:

## Agentų dalis

Pagal užduoties sąlyga sistema turi būti agentinė, todėl Fuzzy logika apibrėžtus veiksmus pasirinkta realizuoti Java programavimo kalba, naudojant Java JADE biblioteką – agentų kūrimo karkasą.

### Sistemoje naudojami agentai ir jų paskirtis

Iš užduoties sąlygos išplaukia, kad sistemoje turi būti realizuoti bei naudojami 3 skirtingi agentai: Agentas A, Agentas V bei Agentas SP.

**Agentas A**: agento paskirtis yra „klausinėti“ (rinkti informaciją, pagal kurią vėliau pats sudaro Fuzzy priklausomybės funkcijas) Agentą V apie trijų skirtingų patiekalų kainas. Priklausomybės funkcijas Agentas A turi sudaryti kas penktą atsakymą, gautą iš agento V. Taip pat Agentas A bendrauja su Agentu SP, kuriam jis turi perduoti informaciją apie pasikeitusias priklausomybės funkcijas, bei persiųsti Agento V, kaip turisto, užduotą klausimą apie restorano kokybę, o atsakymą perduoti atgal Agentui V.

**Agentas V**: agento paskirtis – imituoti skirtingus vartotojus bei, Agentui A paprašius informacijos apie patiekalų kainas, sugeneruoti bei Agentui A persiųsti paprašytos informacijos. Taip pat Agentas V kas kažkiek laiko imituoja turistą, kurio tikslas – suformuoti klausimą apie restorano kokybę, kurį turi perduoti Agentui A, o vėliau iš to paties agento ir sulaukti atsakymo, koks yra paklaustasis restoranas: pigus, prieinamos kainos, prabangus.

**Agentas SP**: agento paskirtis – pritaikyti Fuzzy logiką restorano kokybės nustatymui. Agentas SP saugo priklausomybės funkcijų reikšmes, išėjimų priklausomybės funkcijų reikšmes, aprašytas Fuzzy taisykles, turi aprašytus metodus Fuzzy logikai realizuoti, išėjimo reikšmei (restorano kokybei) apskaičiuoti.

### Agentų bendravimas

Agentai tarpusavyje bendrauja siųsdami vienas kitam žinutes su skirtingais žinučių tipais. Žinutės agentų sistemoje yra aprašomos FIPA specifikacija, konkrečiai – FIPA ACL message struktūra. Struktūra apibrėžia, kad žinutė savyje gali talpinti siuntėją, gavėją, žinutės tekstą, protokolą, kalbą, žinutės tipą bei kitus parametrus. Vienas svarbesnių parametrų – *performative*, kuris apibrėžia žinutės paskirtį/žinutės veiksmą. Sistemoje naudojami 3 tipų *performative* parametrai (8 lentelė):

8 lentelė. Sistemoje naudojami skirtingi performative tipų parametrai ir jų paaiškinimai

|  |  |
| --- | --- |
| *Performative* tipas | Paaiškinimas |
| Request | Žinutės paskirtis – žinutės siuntėjas prašo gavėjo atlikti kažkokį tai veiksmą. Žinutės tekstas gali talpinti veiksmo informaciją. pvz., žodine forma, kurį turi atlikti gavėjas. |
| Inform | Žinutės paskirtis – siuntėjas perduoda kažkokią tai informaciją gavėjui, pvz., persiunčia patiekalų kainų sąrašą. |
| Confirm | Žinutės paskirtis – patvirtinti siuntėjui apie gautą pranešimą ir/ar persiųsti informaciją, kurios prašė siuntėjas. |

Iš sąlygos taip pat aišku, kad Agentas A nuolatos bendrauja su Agentu V ir taip pat bendrauja su Agentu SP, kai reikia atnaujinti priklausomybės funkcijas arba reikia paprašyti, kad pastarasis agentas nustatytų restorano kokybę. Agentas V nuolatos bendrauja tik su Agentu A, o Agentas SP – taip pat tik su Agentu A.

# Užduoties realizacija

Kaip jau minėta anksčiau, sistema realizuota JAVA programavimo kalba, naudojant JAVA JADE agentų kūrimo karkasą. Šiame skyriuje pateikiamas programinis kodas, kuriuo realizuota pateikta užduotis.

## Konstantos

Norint sistemoje išlaikyti vientisumą, buvo sukurtas atskiras konstantų failas, kuriame aprašomi sistemos veikimo metu nekintantys dydžiai, pvz., Agento A patiekalų kainų klausinėjimo intervalas, minimalios bei maksimalios priklausomybės funkcijų reikšmės, Fuzzy reikšmių indeksai, išėjimo reikšmių žodinės išraiškos ir pan. Taip pat konstantų faile aprašytas metodas constantsToArray(), padedantis visų priklausomybės funkcijų minimalias ir maksimalias reikšmes patalpinti į sąrašą (tai reikalinga generuojant patiekalų kainas).

**Programinis kodas:**

package Libraries;

import java.util.ArrayList;

public class Constants {

// General constants

public static final int ASKING\_TIME = 10000;

public static final int ASKING\_TIME\_TOURIST = 9 \* ASKING\_TIME;

public static final String NAME\_AGENT\_A = "AgentA";

public static final String NAME\_AGENT\_V = "AgentV";

public static final String NAME\_AGENT\_SP = "AgentSP";

public static final String AGENTS\_PACK = "Agents";

// Coffee constants

public static final Double COFFEE\_CHEAP\_MIN = 1.5;

public static final Double COFFEE\_CHEAP\_MAX = 2.5;

public static final Double COFFEE\_NORMAL\_MIN = 2.0;

public static final Double COFFEE\_NORMAL\_MAX = 3.0;

public static final Double COFFEE\_EXPENSIVE\_MIN = 2.5;

public static final Double COFFEE\_EXPENSIVE\_MAX = 3.5;

// Soup constants

public static final Double SOUP\_CHEAP\_MIN = 2.0;

public static final Double SOUP\_CHEAP\_MAX = 3.5;

public static final Double SOUP\_NORMAL\_MIN = 3.0;

public static final Double SOUP\_NORMAL\_MAX = 4.5;

public static final Double SOUP\_EXPENSIVE\_MIN = 4.0;

public static final Double SOUP\_EXPENSIVE\_MAX = 5.5;

// Stake constants

public static final Double STEAK\_CHEAP\_MIN = 5.0;

public static final Double STEAK\_CHEAP\_MAX = 12.0;

public static final Double STEAK\_NORMAL\_MIN = 8.0;

public static final Double STEAK\_NORMAL\_MAX = 15.0;

public static final Double STEAK\_EXPENSIVE\_MIN = 13.0;

public static final Double STEAK\_EXPENSIVE\_MAX = 20.0;

// Output functions constants

public static final Double OUTPUT\_CHEAP\_MIN = 0.0;

public static final Double OUTPUT\_CHEAP\_MAX = 4.0;

public static final Double OUTPUT\_AFFORDABLE\_MIN = 2.0;

public static final Double OUTPUT\_AFFORDABLE\_MAX = 8.0;

public static final Double OUTPUT\_LUXURIOUS\_MIN = 6.0;

public static final Double OUTPUT\_LUXURIOUS\_MAX = 10.0;

// Fuzzy values indexes

public static final int INDEX\_COFFE\_CHEAP = 0;

public static final int INDEX\_COFFE\_NORMAL = 1;

public static final int INDEX\_COFFE\_EXPENSIVE = 2;

public static final int INDEX\_SOUP\_CHEAP = 3;

public static final int INDEX\_SOUP\_NORMAL = 4;

public static final int INDEX\_SOUP\_EXPENSIVE = 5;

public static final int INDEX\_STEAK\_CHEAP = 6;

public static final int INDEX\_STEAK\_NORMAL = 7;

public static final int INDEX\_STEAK\_EXPENSIVE = 8;

// Price types

public static final String PRICE\_CHEAP = "Cheap";

public static final String PRICE\_NORMAL = "Normal";

public static final String PRICE\_EXPENSIVE = "Expensive";

// Restaurant states

public static final String RESTAURANT\_CHEAP = "Cheap";

public static final String RESTAURANT\_AFFORDABLE = "Affordable";

public static final String RESTAURANT\_LUXURIOUS = "Luxurious";

// Function to get dishes price ranges values array

public static ArrayList<Double> constantsToArray(){

ArrayList<Double> constants = new ArrayList<>();

constants.add(COFFEE\_CHEAP\_MIN);

constants.add(COFFEE\_CHEAP\_MAX);

constants.add(COFFEE\_NORMAL\_MIN);

constants.add(COFFEE\_NORMAL\_MAX);

constants.add(COFFEE\_EXPENSIVE\_MIN);

constants.add(COFFEE\_EXPENSIVE\_MAX);

constants.add(SOUP\_CHEAP\_MIN);

constants.add(SOUP\_CHEAP\_MAX);

constants.add(SOUP\_NORMAL\_MIN);

constants.add(SOUP\_NORMAL\_MAX);

constants.add(SOUP\_EXPENSIVE\_MIN);

constants.add(SOUP\_EXPENSIVE\_MAX);

constants.add(STEAK\_CHEAP\_MIN);

constants.add(STEAK\_CHEAP\_MAX);

constants.add(STEAK\_NORMAL\_MIN);

constants.add(STEAK\_NORMAL\_MAX);

constants.add(STEAK\_EXPENSIVE\_MIN);

constants.add(STEAK\_EXPENSIVE\_MAX);

return constants;

}

}

## Agentas A

**Programinis kodas:**

package Agents;

import Libraries.Constants;

import jade.core.AID;

import jade.core.Agent;

import jade.core.behaviours.CyclicBehaviour;

import jade.core.behaviours.OneShotBehaviour;

import jade.core.behaviours.TickerBehaviour;

import jade.lang.acl.ACLMessage;

import jade.lang.acl.UnreadableException;

import jade.wrapper.AgentContainer;

import jade.wrapper.AgentController;

import jade.wrapper.StaleProxyException;

import java.io.IOException;

import java.util.ArrayList;

import java.util.Arrays;

public class AgentA extends Agent{

// List dishesList - answers from users

// List in list (ArrayList<Double>):

// Index 0-2: Coffee (cheap, normal, expensive)

// Index 3-5: Soup (cheap, normal, expensive)

// Index 6-8: Steak (cheap, normal, expensive)

private final ArrayList<ArrayList<Double>> allAnswersList;

// DishesList input functions

private final ArrayList<ArrayList<Double>> inputFunctions;

int answersCounter = 0;

private AgentController AC = null;

// Constructor

public AgentA(){

this.inputFunctions = new ArrayList<>();

this.allAnswersList = new ArrayList<>();

initializeInputFunctions();

}

@Override

protected void setup(){

// Behaviour purpose - create all necessary agents

addBehaviour(new OneShotBehaviour() {

@Override

public void action() {

System.out.println(getAID().getLocalName() + " has been created.");

createAgent(Constants.NAME\_AGENT\_V);

createAgent(Constants.NAME\_AGENT\_SP);

}

});

// Ask AgentV for answers about dishes prices

addBehaviour(new TickerBehaviour(this, Constants.ASKING\_TIME) {

@Override

protected void onTick() {

ACLMessage message = new ACLMessage(ACLMessage.REQUEST);

message.addReceiver(new AID(Constants.NAME\_AGENT\_V, AID.ISLOCALNAME));

myAgent.send(message);

}

});

// Behaviour for incoming messages handling

addBehaviour(new CyclicBehaviour() {

@Override

public void action() {

ACLMessage receivedMessage = myAgent.receive();

if(receivedMessage != null){

String agentName = receivedMessage.getSender().getName();

agentName = agentName.substring(0, agentName.indexOf("@"));

// Check sender's name

if(agentName.equals(Constants.NAME\_AGENT\_V)){

// Check received message's type

switch(receivedMessage.getPerformative()){

// Received meesage about dishes prices from AgentV

case ACLMessage.INFORM:

// Getting prices

ArrayList<Double> answersList = new ArrayList<>();

try {

answersList = (ArrayList<Double>)receivedMessage.getContentObject();

} catch (UnreadableException ex) {}

allAnswersList.add(answersList);

// Check if input functions update is needed

if(++answersCounter % 5 == 0){

// Calculate average of last 5 answers about dishes price

int index = allAnswersList.size()-5;

ArrayList<Double> resultsSum = new ArrayList<>(Arrays.asList(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0));

for (int i = index; i < allAnswersList.size(); i++){

answersList = allAnswersList.get(i);

for (int j = 0; j < answersList.size(); j++){

resultsSum.set(j, answersList.get(j) + resultsSum.get(j));

}

}

// Calculate average price of every dish and update input functions

for(int i = 0; i < inputFunctions.size(); i++){

inputFunctions.get(i).set(1, Math.round((Double)((inputFunctions.get(i).get(1)\*(answersCounter-5) + resultsSum.get(i))/answersCounter)\*100.0)/100.0);

}

// Inform AgentSP about input functions changes

ACLMessage message = new ACLMessage(ACLMessage.INFORM);

message.addReceiver(new AID(Constants.NAME\_AGENT\_SP, AID.ISLOCALNAME));

try {

message.setContentObject(inputFunctions);

} catch (IOException ex) {}

myAgent.send(message);

}

break;

// Received message from tourist to identify restaurant quality

case ACLMessage.REQUEST:

ArrayList<Double> touristQuestionPrices = new ArrayList<>();

try {

touristQuestionPrices = (ArrayList<Double>)receivedMessage.getContentObject();

} catch (UnreadableException ex) {}

// Generating message for Agent\_SP in order to identify restaurant's quality

ACLMessage message = new ACLMessage(ACLMessage.REQUEST);

message.addReceiver(new AID(Constants.NAME\_AGENT\_SP, AID.ISLOCALNAME));

try {

message.setContentObject(touristQuestionPrices);

} catch (IOException ex) {}

myAgent.send(message);

break;

}

}

// If the message's sender is Agent\_SP

else if(agentName.equals(Constants.NAME\_AGENT\_SP)){

// Generating message for Agent\_V to send the answer to the question about restaurant's quality

ACLMessage message = new ACLMessage(ACLMessage.INFORM);

message.setContent(receivedMessage.getContent());

message.addReceiver(new AID(Constants.NAME\_AGENT\_V, AID.ISLOCALNAME));

myAgent.send(message);

}

}

else{

block();

}

}

});

}

// Function which creates an agent of name Pav and appends it to main container

private void createAgent(String Pav){

try{

AgentContainer Konteineris = (AgentContainer) getContainerController();

AC = Konteineris.createNewAgent(Pav, Constants.AGENTS\_PACK + "." + Pav, null);

AC.start();

}

catch(StaleProxyException any){}

}

// Function for dishes prices input functions initialization

private void initializeInputFunctions(){

inputFunctions.add(new ArrayList<>(Arrays.asList(Constants.COFFEE\_CHEAP\_MIN, 0.0, Constants.COFFEE\_CHEAP\_MAX)));

inputFunctions.add(new ArrayList<>(Arrays.asList(Constants.COFFEE\_NORMAL\_MIN, 0.0, Constants.COFFEE\_NORMAL\_MAX)));

inputFunctions.add(new ArrayList<>(Arrays.asList(Constants.COFFEE\_EXPENSIVE\_MIN, 0.0, Constants.COFFEE\_EXPENSIVE\_MAX)));

inputFunctions.add(new ArrayList<>(Arrays.asList(Constants.SOUP\_CHEAP\_MIN, 0.0, Constants.SOUP\_CHEAP\_MAX)));

inputFunctions.add(new ArrayList<>(Arrays.asList(Constants.SOUP\_NORMAL\_MIN, 0.0, Constants.SOUP\_NORMAL\_MAX)));

inputFunctions.add(new ArrayList<>(Arrays.asList(Constants.SOUP\_EXPENSIVE\_MIN, 0.0, Constants.SOUP\_EXPENSIVE\_MAX)));

inputFunctions.add(new ArrayList<>(Arrays.asList(Constants.STEAK\_CHEAP\_MIN, 0.0, Constants.STEAK\_CHEAP\_MAX)));

inputFunctions.add(new ArrayList<>(Arrays.asList(Constants.STEAK\_NORMAL\_MIN, 0.0, Constants.STEAK\_NORMAL\_MAX)));

inputFunctions.add(new ArrayList<>(Arrays.asList(Constants.STEAK\_EXPENSIVE\_MIN, 0.0, Constants.STEAK\_EXPENSIVE\_MAX)));

}

}

## Agentas V

**Programinis kodas:**

package Agents;

import Libraries.Constants;

import jade.core.AID;

import jade.core.Agent;

import jade.core.behaviours.CyclicBehaviour;

import jade.core.behaviours.TickerBehaviour;

import jade.lang.acl.ACLMessage;

import java.io.IOException;

import java.util.ArrayList;

import java.util.Random;

public class AgentV extends Agent{

@Override

protected void setup(){

System.out.println(getAID().getLocalName() + " has been created.");

// Behaviour for incoming messages handling

addBehaviour(new CyclicBehaviour() {

@Override

public void action() {

ACLMessage receivedMessage = myAgent.receive();

if(receivedMessage != null){

switch(receivedMessage.getPerformative()){

// Received message from Agent\_A to provide answer about dishes prices

case ACLMessage.REQUEST:

// Generating message with answers about dishes prices

ACLMessage answerMessage = receivedMessage.createReply();

answerMessage.setPerformative(ACLMessage.INFORM);

ArrayList<Double> answersList = generatePrices(1);

try {

answerMessage.setContentObject(answersList);

} catch (IOException ex) {}

myAgent.send(answerMessage);

break;

// Received the answer from Agent\_A about restaurant's quality

case ACLMessage.INFORM:

System.out.println(receivedMessage.getContent());

break;

}

}

else{

block();

}

}

});

// Behaviour for simulation of tourists and their questions about dishes prices

addBehaviour(new TickerBehaviour(this, Constants.ASKING\_TIME\_TOURIST) {

@Override

protected void onTick() {

// Generating message for Agent\_A about dishes prices in restaurant

ACLMessage message = new ACLMessage(ACLMessage.REQUEST);

message.addReceiver(new AID(Constants.NAME\_AGENT\_A, AID.ISLOCALNAME));

ArrayList<Double> pricesList = generatePrices(5);

try {

message.setContentObject(pricesList);

} catch (IOException ex) {}

myAgent.send(message);

}

});

}

// Generating random prices using price ranges from constants

private ArrayList<Double> generatePrices(int rangeOffset){

ArrayList<Double> answersList = new ArrayList<>();

ArrayList<Double> rangesList = Constants.constantsToArray();

Random rnd = new Random();

for (int i = 0; i < rangesList.size(); i+= (rangeOffset+1)){

Double minRangeDouble = rangesList.get(i)\*100;

Double maxRangeDouble = rangesList.get(i+rangeOffset)\*100;

answersList.add((double)(rnd.nextInt(maxRangeDouble.intValue() - minRangeDouble.intValue() + 1) + minRangeDouble.intValue())/100);

}

return answersList;

}

}

## Agentas SP

**Programinis kodas:**

package Agents;

import Libraries.Constants;

import jade.core.Agent;

import jade.core.behaviours.CyclicBehaviour;

import jade.lang.acl.ACLMessage;

import jade.lang.acl.UnreadableException;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

public class AgentSP extends Agent {

// The list of input functinos

private ArrayList<ArrayList<Double>> inputFunctions;

// The list of output functions

private final ArrayList<ArrayList<Double>> outputFunctions;

// Constructor

public AgentSP(){

this.inputFunctions = new ArrayList<>();

this.outputFunctions = new ArrayList<>();

// Initialising output functions list

this.outputFunctions.add(new ArrayList<>(Arrays.asList(-1.0, Constants.OUTPUT\_CHEAP\_MIN, Constants.OUTPUT\_CHEAP\_MAX)));

this.outputFunctions.add(new ArrayList<>(Arrays.asList(Constants.OUTPUT\_AFFORDABLE\_MIN, 5.0, Constants.OUTPUT\_AFFORDABLE\_MAX)));

this.outputFunctions.add(new ArrayList<>(Arrays.asList(Constants.OUTPUT\_LUXURIOUS\_MIN, Constants.OUTPUT\_LUXURIOUS\_MAX, 11.0)));

}

@Override

protected void setup(){

System.out.println(getAID().getLocalName() + " has been created.");

// Behaviour for incoming messages handling

addBehaviour(new CyclicBehaviour() {

@Override

public void action() {

ACLMessage receivedMessage = myAgent.receive();

if(receivedMessage != null){

// Check received message's type

switch(receivedMessage.getPerformative()){

// Received message about input functions update

case ACLMessage.INFORM:

try {

// Updating input functions

inputFunctions = (ArrayList<ArrayList<Double>>)receivedMessage.getContentObject();

} catch (UnreadableException ex) {}

System.out.print("Input functions values: ");

System.out.println(inputFunctions);

break;

// Received a request to identify restaurant quality

case ACLMessage.REQUEST:

ArrayList<Double> touristQuestionPrices = new ArrayList<>();

try {

touristQuestionPrices = (ArrayList<Double>)receivedMessage.getContentObject();

} catch (UnreadableException ex) {}

System.out.print("Tourist prices: ");

System.out.println(touristQuestionPrices);

// Calculating fuzzy values from input functions using tourist's provided dishes prices

ArrayList<Double> fuzzyValues = calculateFuzzyValuesOfInput(touristQuestionPrices);

System.out.print("Calculated fuzzy values: ");

System.out.println(fuzzyValues);

// Calculating values of every rule using calculated fuzzy values

ArrayList<Double> rulesValues = calculateRulesValues(fuzzyValues);

System.out.print("Calculated rules values: ");

System.out.println(rulesValues);

// Defuzification using centroid method

double outputValue = calculateOutputUsingCentroid(calculateAggregatedFunctionsValuesFromRules(rulesValues));

System.out.print("Calculated output: ");

System.out.println(outputValue);

// Generating message for Agent\_A about calculated restaurant quality

ACLMessage answerMessage = receivedMessage.createReply();

answerMessage.setPerformative(ACLMessage.CONFIRM);

answerMessage.setContent("Restaurant is: " + verbalizeOutputValue(outputValue));

myAgent.send(answerMessage);

break;

}

}

else{

block();

}

}

});

}

// Function for fuzzy values calculation of every input function

private ArrayList<Double> calculateFuzzyValuesOfInput(ArrayList<Double> input){

ArrayList<Double> fuzzyValues = new ArrayList<>();

int inputIndex = 0;

for(int i = 0; i < inputFunctions.size(); i++){

switch((i+1)%3){

// If the function is of type cheap (trapezium function)

case 1:

fuzzyValues.add(calculateFuzzyValueFromFunction(inputFunctions.get(i), input.get(inputIndex), Constants.PRICE\_CHEAP));

break;

// If the function is of type normal (triangle function)

case 2:

fuzzyValues.add(calculateFuzzyValueFromFunction(inputFunctions.get(i), input.get(inputIndex), Constants.PRICE\_NORMAL));

break;

// If the function is of type expensive (trapezium function)

case 0:

fuzzyValues.add(calculateFuzzyValueFromFunction(inputFunctions.get(i), input.get(inputIndex), Constants.PRICE\_EXPENSIVE));

inputIndex++;

break;

}

}

return fuzzyValues;

}

// Function for getting y value from input function, when x is a dish price provided by tourist

private double calculateFuzzyValueFromFunction(ArrayList<Double> inputFunction, double inputValue, String priceType){

double fuzzyValue = 0.0;

switch(priceType){

case Constants.PRICE\_CHEAP:

if(inputValue >= inputFunction.get(0) && inputValue < inputFunction.get(1)){

fuzzyValue = 1.0;

}

else if(inputValue >= inputFunction.get(1) && inputValue <= inputFunction.get(2)){

fuzzyValue = (double)(inputFunction.get(2)-inputValue)/(inputFunction.get(2)-inputFunction.get(1));

}

break;

case Constants.PRICE\_NORMAL:

if(inputValue >= inputFunction.get(0) && inputValue < inputFunction.get(1)){

fuzzyValue = (double)(inputValue - inputFunction.get(0))/(inputFunction.get(1)-inputFunction.get(0));

}

else if(inputValue >= inputFunction.get(1) && inputValue <= inputFunction.get(2)){

fuzzyValue = (double)(inputFunction.get(2)-inputValue)/(inputFunction.get(2)-inputFunction.get(1));

}

break;

case Constants.PRICE\_EXPENSIVE:

if(inputValue >= inputFunction.get(0) && inputValue < inputFunction.get(1)){

fuzzyValue = (double)(inputValue - inputFunction.get(0))/(inputFunction.get(1)-inputFunction.get(0));

}

else if(inputValue >= inputFunction.get(1) && inputValue <= inputFunction.get(2)){

fuzzyValue = 1.0;

}

break;

}

return fuzzyValue;

}

// Function for rules' values calculation using fuzzy values of every dish price

private ArrayList<Double> calculateRulesValues(ArrayList<Double> fuzzyValues){

ArrayList<Double> rulesValues = new ArrayList<>();

// There are 13 different rules, so loop from 1 to 13

for (int i = 1; i <= 13; i++){

// Calculates using minimum of all fuzzy values

switch(i){

// Coffe - Cheap, soup - NOT(Cheap), steak - Cheap

// Result: Restaurant - Cheap

case 1:

rulesValues.add(Math.min(Math.min(fuzzyValues.get(Constants.INDEX\_COFFE\_CHEAP), (1-fuzzyValues.get(Constants.INDEX\_SOUP\_CHEAP))), fuzzyValues.get(Constants.INDEX\_STEAK\_CHEAP)));

break;

// Coffe - Normal, soup - NOT(Expensive), steak - Cheap

// Result: Restaurant - Cheap

case 2:

rulesValues.add(Math.min(Math.min(fuzzyValues.get(Constants.INDEX\_COFFE\_NORMAL), (1-fuzzyValues.get(Constants.INDEX\_SOUP\_EXPENSIVE))), fuzzyValues.get(Constants.INDEX\_STEAK\_CHEAP)));

break;

// Coffe - NOT(Expensive), soup - Cheap, steak - Normal

// Result: Restaurant - Cheap

case 3:

rulesValues.add(Math.min(Math.min((1-fuzzyValues.get(Constants.INDEX\_COFFE\_EXPENSIVE)), fuzzyValues.get(Constants.INDEX\_SOUP\_CHEAP)), fuzzyValues.get(Constants.INDEX\_STEAK\_NORMAL)));

break;

// Coffe - NOT(Normal), soup - Cheap, steak - Cheap

// Result: Restaurant - Cheap

case 4:

rulesValues.add(Math.min(Math.min((1-fuzzyValues.get(Constants.INDEX\_COFFE\_NORMAL)), fuzzyValues.get(Constants.INDEX\_SOUP\_CHEAP)), fuzzyValues.get(Constants.INDEX\_STEAK\_CHEAP)));

break;

// Coffe - Cheap, soup - NOT(Expensive), steak - Expensive

// Result: Restaurant - Affordable

case 5:

rulesValues.add(Math.min(Math.min(fuzzyValues.get(Constants.INDEX\_COFFE\_CHEAP), (1-fuzzyValues.get(Constants.INDEX\_SOUP\_EXPENSIVE))), fuzzyValues.get(Constants.INDEX\_STEAK\_EXPENSIVE)));

break;

// Coffe - Expensive, soup - Cheap, steak - NOT(Cheap)

// Result: Restaurant - Affordable

case 6:

rulesValues.add(Math.min(Math.min(fuzzyValues.get(Constants.INDEX\_COFFE\_EXPENSIVE), fuzzyValues.get(Constants.INDEX\_SOUP\_CHEAP)), (1-fuzzyValues.get(Constants.INDEX\_STEAK\_CHEAP))));

break;

// Coffe - Expensive, soup - NOT(Cheap), steak - Cheap

// Result: Restaurant - Affordable

case 7:

rulesValues.add(Math.min(Math.min(fuzzyValues.get(Constants.INDEX\_COFFE\_EXPENSIVE), (1-fuzzyValues.get(Constants.INDEX\_SOUP\_CHEAP))), fuzzyValues.get(Constants.INDEX\_STEAK\_CHEAP)));

break;

// Coffe - Normal, soup - Cheap, steak - Expensive

// Result: Restaurant - Affordable

case 8:

rulesValues.add(Math.min(Math.min(fuzzyValues.get(Constants.INDEX\_COFFE\_NORMAL), fuzzyValues.get(Constants.INDEX\_SOUP\_CHEAP)), fuzzyValues.get(Constants.INDEX\_STEAK\_EXPENSIVE)));

break;

// Coffe - Normal, soup - Expensive, steak - NOT(Expensive)

// Result: Restaurant - Affordable

case 9:

rulesValues.add(Math.min(Math.min(fuzzyValues.get(Constants.INDEX\_COFFE\_NORMAL), fuzzyValues.get(Constants.INDEX\_SOUP\_EXPENSIVE)), (1-fuzzyValues.get(Constants.INDEX\_STEAK\_EXPENSIVE))));

break;

// Coffe - NOT(Expensive), soup - Normal, steak - Normal

// Result: Restaurant - Affordable

case 10:

rulesValues.add(Math.min(Math.min((1-fuzzyValues.get(Constants.INDEX\_COFFE\_EXPENSIVE)), fuzzyValues.get(Constants.INDEX\_SOUP\_NORMAL)), fuzzyValues.get(Constants.INDEX\_STEAK\_NORMAL)));

break;

// Coffe - Cheap, soup - Expensive, steak - NOT(Cheap)

// Result: Restaurant - Luxurious

case 11:

rulesValues.add(Math.min(Math.min(fuzzyValues.get(Constants.INDEX\_COFFE\_CHEAP), fuzzyValues.get(Constants.INDEX\_SOUP\_EXPENSIVE)), (1-fuzzyValues.get(Constants.INDEX\_STEAK\_CHEAP))));

break;

// Coffe - Expensive, soup - NOT(Cheap), steak - NOT(Cheap)

// Result: Restaurant - Luxurious

case 12:

rulesValues.add(Math.min(Math.min(fuzzyValues.get(Constants.INDEX\_COFFE\_EXPENSIVE), (1-fuzzyValues.get(Constants.INDEX\_SOUP\_CHEAP))), (1-fuzzyValues.get(Constants.INDEX\_STEAK\_CHEAP))));

break;

// Coffe - Normal, soup - NOT(Cheap), steak - Expensive

// Result: Restaurant - Luxurious

case 13:

rulesValues.add(Math.min(Math.min(fuzzyValues.get(Constants.INDEX\_COFFE\_NORMAL), (1-fuzzyValues.get(Constants.INDEX\_SOUP\_CHEAP))), fuzzyValues.get(Constants.INDEX\_STEAK\_EXPENSIVE)));

break;

}

}

return rulesValues;

}

// Calculating function values necessary for output functions aggregation

private ArrayList<Double> calculateAggregatedFunctionsValuesFromRules(ArrayList<Double> rulesValues){

ArrayList<Double> aggregatedValues = new ArrayList<>();

// 72---------------------------

double aggregatedCheapValue = maxOfArrayList(rulesValues.subList(0, 4));

double aggregatedAffordableValue = maxOfArrayList(rulesValues.subList(4, 10));

double aggregatedLuxuriousValue = maxOfArrayList(rulesValues.subList(10, 13));

//-------------------------------

for (int i = 0; i <= 10; i++){

if(i <= 2)

aggregatedValues.add(aggregatedCheapValue);

else if(i == 3)

aggregatedValues.add(aggregatedCheapValue >= aggregatedAffordableValue ? aggregatedCheapValue : aggregatedAffordableValue);

else if(i >= 4 && i <= 6)

aggregatedValues.add(aggregatedAffordableValue);

else if(i == 7)

aggregatedValues.add(aggregatedLuxuriousValue >= aggregatedAffordableValue ? aggregatedLuxuriousValue : aggregatedAffordableValue);

else

aggregatedValues.add(aggregatedLuxuriousValue);

}

return aggregatedValues;

}

// Function for finding the maximum value of list

private double maxOfArrayList(List<Double> list){

double currentMax = 0.0;

for(Double d : list) currentMax = Math.max(currentMax, d);

return currentMax;

}

// Function for calculation of output value using centroid method

private double calculateOutputUsingCentroid(ArrayList<Double> outputValues){

double numerator = 0.0;

for(int i = 0; i <= 10; i++) numerator += i \* outputValues.get(i);

double denominator = 0.0;

for(double d : outputValues) denominator += d;

return (double)numerator/denominator;

}

// Function for output value verbalisation (converting output value to one of String values: "Cheap", "Affordable", "Luxurious" according to the calculated output value)

private String verbalizeOutputValue(double outputValue){

int outputIndex = 0;

double maxValue = 0.0;

for(int i = 0; i < outputFunctions.size(); i++){

double fuzzyValue = 0.0;

ArrayList<Double> function = outputFunctions.get(i);

if(outputValue >= function.get(0) && outputValue < function.get(1)){

fuzzyValue = (double)(outputValue - function.get(0))/(function.get(1)-function.get(0));

}

else if(outputValue >= function.get(1) && outputValue <= function.get(2)){

fuzzyValue = (double)(function.get(2)-outputValue)/(function.get(2)-function.get(1));

}

if (fuzzyValue > maxValue){

maxValue = fuzzyValue;

outputIndex = i;

}

}

return outputIndex == 0 ? Constants.RESTAURANT\_CHEAP : outputIndex == 1 ? Constants.RESTAURANT\_AFFORDABLE : Constants.RESTAURANT\_LUXURIOUS;

}

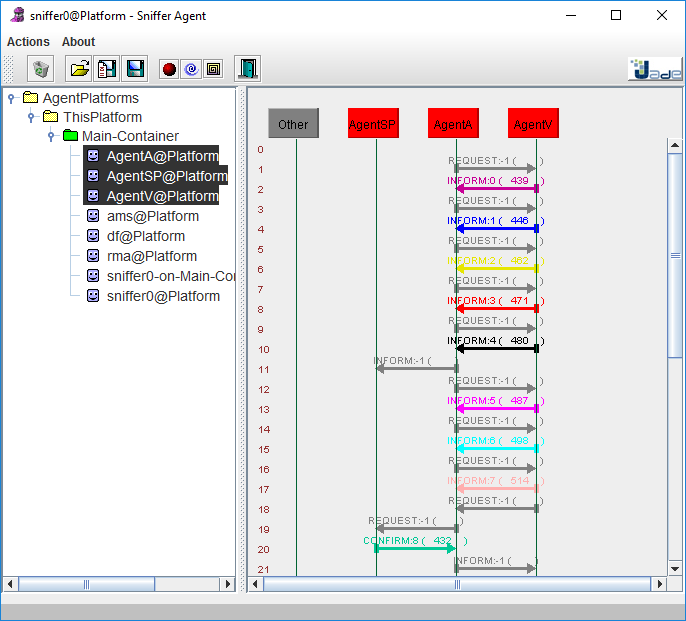
}

# Sistemos veikimas bei rezultatai

Šiame skyriuje pateikiama informacija apie tai, kaip sistema veikia, kokie buvo gauti rezultatai bei parodoma, kaip agentai bendrauja tarpusavyje sistemos veikimo metu.

## Agentų bendravimas

5 paveiklėlyje pateiktas grafinės agentų valdymo sąsajos sniffer langas, kuriame matoma, kaip agentai bendrauja tarpusavyje:



6

5

4

3.3

3.2

3.1

3

1.5

1.4

1.3

1.2

1.1

2

1

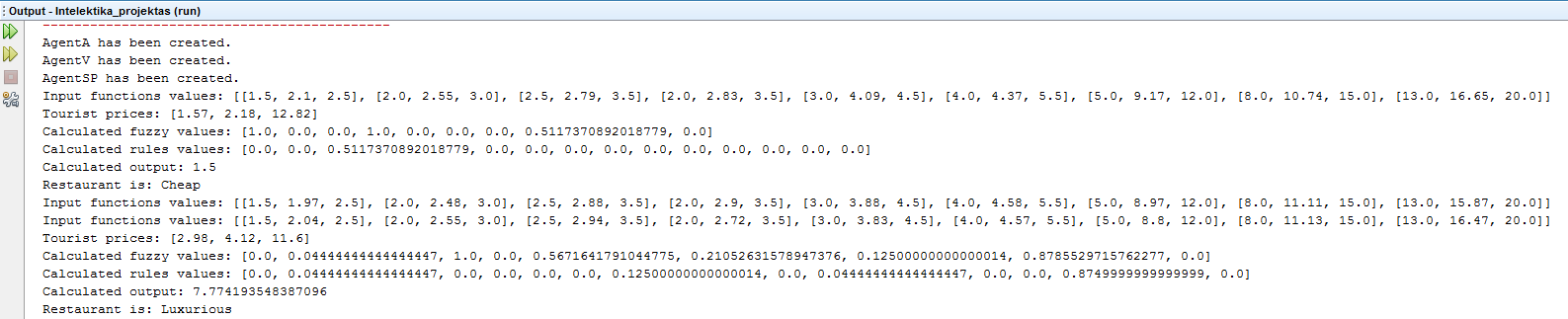
5 pav. Agentų grafinės valdymo sąsajos sniffer langas

Agentų bendravima galima suskirstyti į tokias dalis:

1. Agentas A klausinėja Agento V apie kiekvieno patiekalo kainas (ar patiekalo kaina yra pigi/normali/brangi). Kaip matome, tai vyksta 5 kartus (1.1 – 1.5) punktai;
2. Gavus 5 atsakymus, pakeičiamos priklausomybės funkcijos, todėl po 5 gautų atsakymų Agentas A siunčia žinutę su tipu *inform* apie pasikeitusias funkcijų reikšmes;
3. Toliau vyksta komunikacija tarp Agento A ir Agento V dėl patiekalų kainų reikšmių (3.1 – 3.3 punktai);
4. Po tam tikro laiko Agentas V imituoja turistą ir siunčia žinutę Agentui A su klausimu apie restorano kokybę – žinutės tipas *request* nurodo, kad Agentas V prašo Agento A įvykdyti restorano kokybės nustatymo veiksmą;
5. Iš Agento V gavęs žinutę apie prašymą nustatyti restorano kokybę, Agentas A prašo (žinutės tipas *request*) Agento SP nustatyti restorano kokybę, persiųsdamas jam turisto perduotas patiekalų kainas. Agentas SP patvirtina gavęs restorano kokybės nustatymo prašymą kartu persiųsdamas Agentui A atsakymą apie restorano kokybę;
6. Gavęs rezultatus iš Agento SP, Agentas A informuoja (žinutės tipas *inform*) Agentą V apie tai, jog buvo nustatyta restorano kokybė, kartu persiųsdamas restorano kokybės įvertinimą (ar tai pigus/prieinamos kainos/brangus restoranas).

## Sistemos veikimo rezultatai

Sistemos veikimo metu, svarbiausia informacija apie priklausomybės funkcijų pasikeitimus, apskaičiuotas Fuzzy reikšmes, nustatytą restorano kokybę ir kitus panašius aspektus, išvedama JAVA output konsolės lange (6 paveikslėlis):

Kaip matome iš pateiktos informacijos, pirmiausia pranešama apie agentų sukūrimą. Vėliau sistema kas penktą pateiktą Agento V atsakymą apie patiekalų kainas informuoja apie pasikeitusias priklausomybės funkcijų reikšmes. Taip pat matome, kada Agento V imituotas turistas užduoda klausimą (išvedamos turisto nurodytos patiekalų kainos) restorano kokybei nustatyti. Prieš pateikiant restorano kokybę žodine forma, taip pat išvedamos iš priklausomybės funkcijų nustatytos Fuzzy reikšmės, kiekvienos taisyklės stiprumai taikant loginius operatorius, pateikiamas restorano įvertinimas balais (nuo 0 iki 10), kuris vėliau paverčiamas į restorano įvertinimą žodine forma.

6 pav. JAVA output konsolės lange pateikiama sistemos vykdymo informacija

# Išvados

Sistema buvo realizuota naudojant JAVA programavimo kalbą bei JAVA JADE biblioteką. Kadangi sistemą realizuoti pavyko, galime daryti išvadą, jog sistemos realizacijai pasirinktos teisingos priemonės.

Duomenų apdorojimui, rezultatų skaičiavimui buvo naudojama Fuzzy logika. Kaip matome iš gautų rezultatų, pasirinkti šią logikos rūšį buvo tinkamas variantas, pilnai išpildęs projekto užduotį, o miglotosios logikos metodai pagelbėjo skaičiavimus, jų rezultatus paversti bei pateikti žmogaus mąstymui priimtinesne – žodine forma.

Iš sistemos veikimo rezultatų bei pateiktos informacijos JAVA output konsolės lange galime teigti, kad sistema veikia taip, kaip buvo suplanuota, t.y., pritaikyti Fuzzy logikos metodai veikia tinkamai, priklausomybės funkcijos parinktos tinkamos (turint omeny, kad jos yra koreguojamos sistemos veikimo metu), taisyklės optimizuotos tinkamai, nes remiantis įvedimo duomenimis (input) ir gautais rezultatais (output) matoma, kad sistema pateikia teisingus rezultatus/žodines išvadas apie restorano kokybę.