

Stohastička okružja

Projektna dokumentacija

Lovro Predovan
br. indeksa: 43539/14-R

Mentor:
Doc. dr. sc. Markus Schatten

Varaždin, 22. lipnja 2016.

Sadržaj

1	Uvod	1
2	Opis problema	2
3	Višeagentni sustav stohastičkom okruženju	3
3.1	Agent klijent	4
3.2	Agent evaluator	5
4	Programsko rješenje	6
4.1	Agent klijent	7
4.2	Agent evaluator	11
5	Kritički osvrt	12
6	Zaključak	13
	Bibliografija	13
7	Programski kod	15
7.1	Agent klijent	15
7.1.1	client.py	15
7.1.2	evaluator.py	17
7.1.3	print formatter.py	20
7.2	Agent evaluator	20
7.2.1	server.py	20
7.2.2	bookie.py	22
7.2.3	football db.py	24

Poglavlje 1

Uvod

Prema (Wikipedia, 2016) riječ stohastično je grčkih korijena i originalno označava da neki ishod ili događaj zavise o sreći. Na taj način se koristi za opisivanje predmeta, događaja ili fenomena koji sadrže neki element slučajnih ponašanja. Da bi neki sustav bio stohastičan, barem neki dio tog sustava mora biti izložen elementu nesigurnosti. Za razliku od determinističkih sustava, stohastični sustav ne mora uvijek davati određeni izlaz te se kod stohastičkih sustava može izdvojiti nekoliko elemenata poput: ulaza, kašnjenja, ometanja, pa čak i elemenata koji su stohastično dinamični procesi. Zajedničke karakteristike svih stohastičkih okruženja su nepostojanje striktno određenih veza među čimbenicima te nepostojanje čvrsto definirane strukture, međusobnih odnosa elemenata te podložnost utjecaju kombinacija niza slučajnih faktora. Pristup problemu u ovom radu će se temeljiti na tim pretpostavkama nesigurnosti pri definiranju budućih sljedova događaja koji mogu biti kvantificirani na nekoliko načina. (NS2, 2010) Jedan od mogućih pristupa problemu je diferencijalna jednačba koja uključuje stanje, ulazne i izlazne varijable. Ovaj pristup omogućava opis sustava kao implementaciju Markovljevih lanaca. Drugi pristup problemu se temelji samo na kombinaciji ulaznih i izlaznih varijabli te ćemo taj pristup i koristiti u ovom radu. U praksi analitičari se susreću s dva problema. Prvi se odnosi na primjenu informacija i podataka koji su prikupljeni kroz promatranje izlaza iz sustava kako bi smanjili nesigurnost ponašanja i koliko je moguće predvidjeli buduće izlaze iz sustava. Drugi izazov s kojim se analitičari susreću je odabir, odnosno efekt određenih ponašanja i obrazaca ponašanja te odabir onih najpoželjnijih. Prvi izazov vodi do problema procjene i problema identifikacije glavnih odrednica sustava. Drugi izazov pak formulira problem kao problem stohastičke kontrole.

U ovom radu će se navedeni problemi i pogledi uklopiti i simulirati kroz višeagentnu okolinu koja na temelju određenih inputa, odnosno povijesnih podataka pokušava predvidjeti najizgledniji izlaz iz sustava.

Poglavlje 2

Opis problema

Nogomet je globalno najpopularniji sport na svijetu, te kao takav zbog ogromnog interesa brojnih obožavatelja je iznimno pogodan za prikupljanje raznih statističkih podataka koji su uglavnom javno dostupni. U ovom radu će se na temelju skupa ulaznih podataka nastojati predvidjeti izgledi za pobjedu u nogometnoj utakmici između dviju momčadi. Za domenu promatranja su uzete momčadi koje su sudjele na svjetskom prvenstvu u Brazilu 2014. godine a statistički podaci o utakmicama i igračima kao i povijesni rezultati su preuzeti iz javno dostupnog skupa podataka na <https://github.com/openfootball> (dostupno 20.6.2016) . Postoje mnogi radovi koji su se bavili ovom ili sličnom tematikom poput <http://www.davemease.com/papers/football.pdf> (dostupno 20.6.2016) , no kako je riječ o stohastičkim događajima zapravo niti jedan od njih nije u stanju u potpunosti predvidjeti ishod igre. Na ovoj činjenici upravo i počiva cijela industrija kladjenja na nogometne događaje, tako da i brojne kladionice neprestano razvijaju nove metode proučavanja sportskih događaja kako bi uvećale svoje profite. U zadnje vrijeme je najpopularnija metoda rudarenja podacima koja može dati zanimljive informacije i drukčiji pogled na pojedine statističke kategorije no ne može u potpunosti pomoći u predviđanju rezultata utakmica no svakako može biti jedna od metoda koje mogu pripomoći konačnom dojmu i indikaciji kako bi utakmica mogla završiti.

U ovom radu će se uzeti višeagentni pristup problemu pri kojem će jedan ili više agenta imitirati klijente koji odabiru utakmice prema korisnikovom unosu, slati svoje odabire glavnom agentu koji će potom vršiti analizu i obradu nad podacima. Na temelju obrađenih parametara će razlučiti koja nogometna momčad ima veće šanse za pobjedu. Algoritam se temelji na obradi nad podacima koji su javno dostupni i analizira povijesne rezultate pojedinog tima na velikim natjecanjima, broj naslova svjetskih prvaka, broj prethodnih pobjeda, poraza i neriješenih ishoda, iskustvo igrača, snagu momčadi u kojoj igraju, gol razlici i brojnim drugim parametrima koji mogu biti indikativni.

Glavna motivacija za izradu ovog seminarskog rada je stvoriti dobru podlogu za buduću implementaciju raznih algoritama kao i usporedbu njihove učinkovitosti naspram rezultata iz realnog svijeta.

Ukratko, u daljnjim poglavljima će biti predstavljena teorijska pozadina problema uz usporedbu s osnovnim postavkama višeagentnih sustava, bit će razrađena domena problema i model rješenja te implementacija u programskim jezicima Python uz pomoć programskog modula SPADE2 te openFootballDb kao podatkovne potpore.

Poglavlje 3

Višeagentni sustav stohastičkom okruŒenju

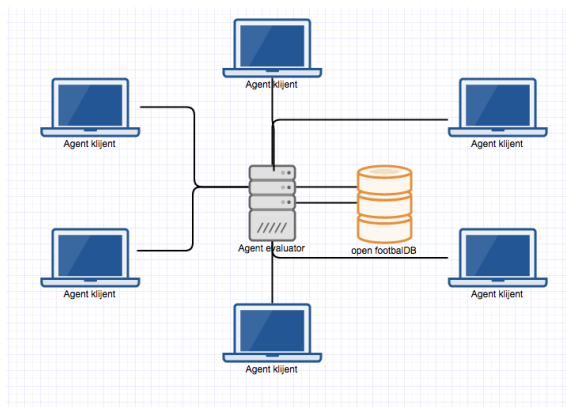
Višeagentni pristup problemima nije jednostavan zbog same činjenice da su agenti u računalnim znanošćima rubni i relativno apstraktan pojam te se na njihovo postojanje i djelovanje može gledati s različitih pogleda i područja znanosti poput umjetne inteligencije, ekonomije, teorije sustava, teorije igara, sociologije, psihologije i mnogih drugih. (Schatten, 2008) Sama definicija agenta kao osnovne jedinice svakog višeagentnog sustava prema (M, 2002) glasi : Agent je računalni sustav koji je smješćen u neku okolinu te je sposoban autonomno obavljati zadatke u okolini kako bi ispunio svoju svrhu, agenti funkcioniraju u ne-determinističkoj stohastičkoj okolini te trebaju pokrivati široki spektar mogućnosti, imati sposobnost odlučivanja i rješavanja problema. Također agenti predstavljaju razinu apstrakcije više od dosad poznatih koncepata u programiranju poput klasa i mogu im se pripisati neka svojstva koja nisu svojstvena računalima. Primjerice pojedinog agenta možemo opisati i kroz skup njegovih želja i težnja što ih čini još kompleksnijim za implementirati.

Višeagentni sustavi (engl. Multi-agent system - MAS) su sustavi koji se sastoje od niza agenata u međusobnoj interakciji koji se ponašaju u skladu s ciljevima i motivima korisnika koje zastupaju što je u ovom radu reprezentirano kako je već spomenuto. (Schatten 2013.)

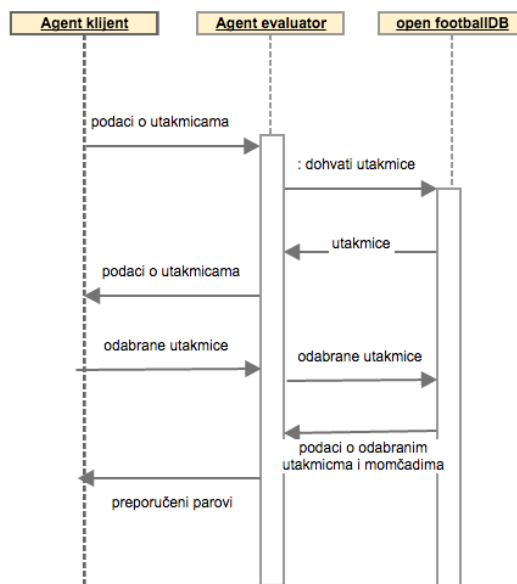
Višeagentni sustavi se još i spominju kao samoorganizirajući sustavi koji imaju težnju pronalaska najboljeg mogućeg rješenja, odnosno udruživanja kako bi se mogli lakše riješiti problemi koji nisu trivijalni te ih je samostalno gotovo nemoguće riješiti no to nije problem koji će se razmatrati u ovom radu. (Predovan, 2009) prema (F., 2013) i (Schatten, 2008)

U ovom radu želje i težnje su predstavljene kao osnovne postavke koje se nasumično pridijele pri pokretanju svakog zasebnog agenta, odnosno njihove preferencije prema kladenju. Tako agent se agent može ponašati rizično, nepredvidivo, može poslušati preporuke glavnog agenta ili može igrati samo na si-gurno.

Arhitektura sustava se može prikazati sljedećim grafom:



Slika 3.1: Dijagram arhitekture sustava (vlastita izrada)



Slika 3.2: Dijagram komunikacije agenata (vlastita izrada)

3.1 Agent klijent

Uloga agenta klijenta je simulacija ponašanja korisnika, odnosno osobe koja želi doznati predviđanje rezultata utakmice. Klijentima se pri pokretanju nasumično dodijeli obrazac ponašanja, odnosno način na koji će polagati oklade ovisno o raspoloženju. U sustavu su predviđena 3 načina ponašanja klijenta, odnosno postavljanje rizičnih oklada na temelju procjene rezultata, postavljanje oklade u skladu s preporukama glavnog agenta te postavljanje oklade u potpunosti nasumično. Sam rad agenta počinje tako da glavnog agenta pita koje su mu utakmice dostupne kako je i prikazano na 3.2, nakon toga agent može primiti korisnikov odabir u smislu broja utakmica i kombinacija utakmica ili može postaviti željeni broj utakmica i tada glavni agent sam odlučuje o kombinacijama momčadi koje će uzeti u obzir. Nakon što dobije preporuke za oklade od glavnog agenta, agent klijent pregledava rješenja, odlučuje hoće li rješenja uzeti u obzir i odlučuje se za neku od momčadi u skladu sa svoj definiranim obrascem

ponašanja. Nakon toga ispisuje rješenja ako takva kombinacija je zapravo postojala u realnom svijetu, odnosno ako su željene momčadi zapravo igrale međusobno u grupnoj fazi svjetskog prvenstva u Brazilu 2014. Ovakvih agenata može biti neodređeno mnogo i zamišljeno je da svi oni komuniciraju s jednim centralnim agentom evaluatorom.

3.2 Agent evaluator

Agent evaluator je centralni agent u sustavu i zamišljeno je da se ponaša kao centralni čvor u višeagentnoj arhitekturi kao i na slici 3.1. Agent evaluator može komunicirati s N agenata te obavlja dvije glavne zadaće. Prva zadaća mu je dohvaćanje skupa utakmica na koje se moguće kladiti, nakon što primi parametre, odnosno timove ili samo broj utakmica koje treba obraditi, tada analizira podatke kojima raspolaže. Agent evaluator analizira povijesne podatke svake nogometne reprezentacije te njihov uspjeh, kao i igrački kadar te njihovo iskustvo kao i snagu klubova u kojima igraju. Na temelju brojnih parametara računa koeficijent kojeg uspoređuje sa suparničkom ekipom te ih stavlja u omjer tako dajući prognozu koja bi momčad trebala odnijeti pobjedu u njihovom susretu. Pseudoalgoritam funkcionira na temelju sljedeće formule po kojoj se računaju konačni bodovi svake momčadi je :

$$\text{brojBodovaMomčadi} = (\sum \text{nasloviPrvaka}) * 200 + (\text{rankReprezentacije}) + (\sum \text{zajednickeUtakmiceIgraca}) + (\sum \text{pobjede}) * 3 + (\sum \text{nerjesene}) - (\sum \text{izgubljene} * 3) + (\text{golRazlika}) + (\text{osvojeniBodovi}) + (\text{sudjelovanja}) + (\sum \text{topKlubovi} * 10)$$

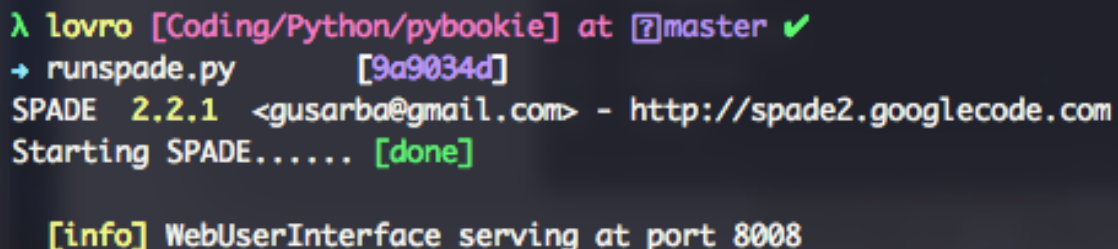
Poglavlje 4

Programsko rješenje

Programsko rješenje je izrađeno u programskom jeziku Python 2.7 uz pomoć razvojnog okruženja SPADE. Pri samoj izradi namjera je stvoriti simuliranu okolinu od minimalno dva agenata koji će međusobno moći komunicirati te na temelju algoritama donositi odluku o procjeni i polaganju oklade na neku od ponuđenih nogometnih utakmica. Postojat će jedan agent kojeg se može okarakterizirati kao glavni agent koji sadrži glavni algoritam usporedbe, obrade podataka i davanja preporuke za okladu, to jest (tj.) agent organizator te ostali agenti koji su sudionici događaja odnosno oklade tj. klijenti. Osnovne zamisli i postavke sustava su da sustav bude dinamičan odnosno da svaki agent bude autonoman, samostalan i da donosi odluku za sebe. Potpuno programsko rješenje je otvorenog koda i samom kodu s uputama se može pristupiti na sljedećoj poveznici:

pybookie github (dostupno 20.6.2016)

Važno je napomenuti kako bi se rad pokrenuo teba imati pokrenut i konfiguriran SPADE server:



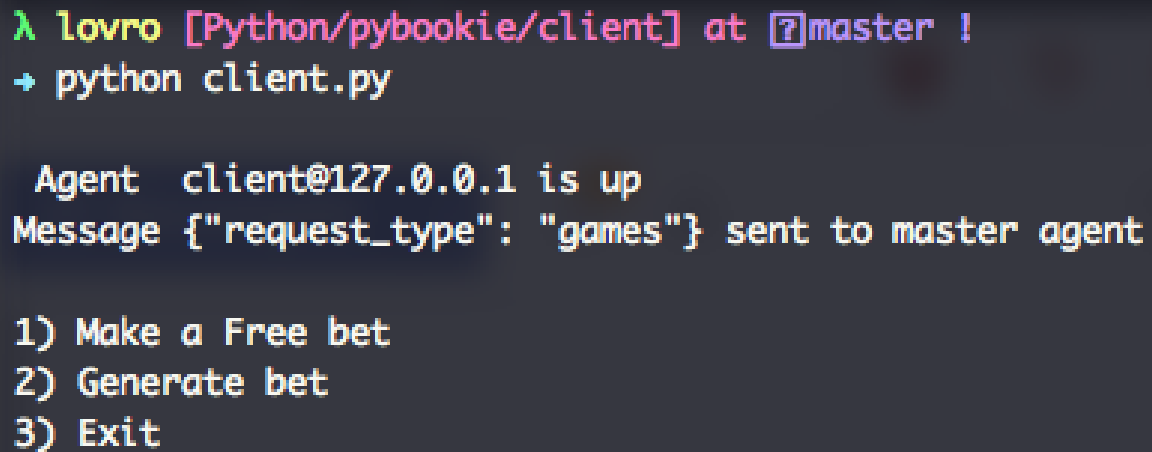
```
λ lovro [Coding/Python/pybookie] at [?]master ✓
→ runspade.py [9a9034d]
SPADE 2.2.1 <gusarba@gmail.com> - http://spade2.googlecode.com
Starting SPADE..... [done]

[info] WebUserInterface serving at port 8008
```

Slika 4.1: Pokretanje spade servera (vlastita izrada)

4.1 Agent klijent

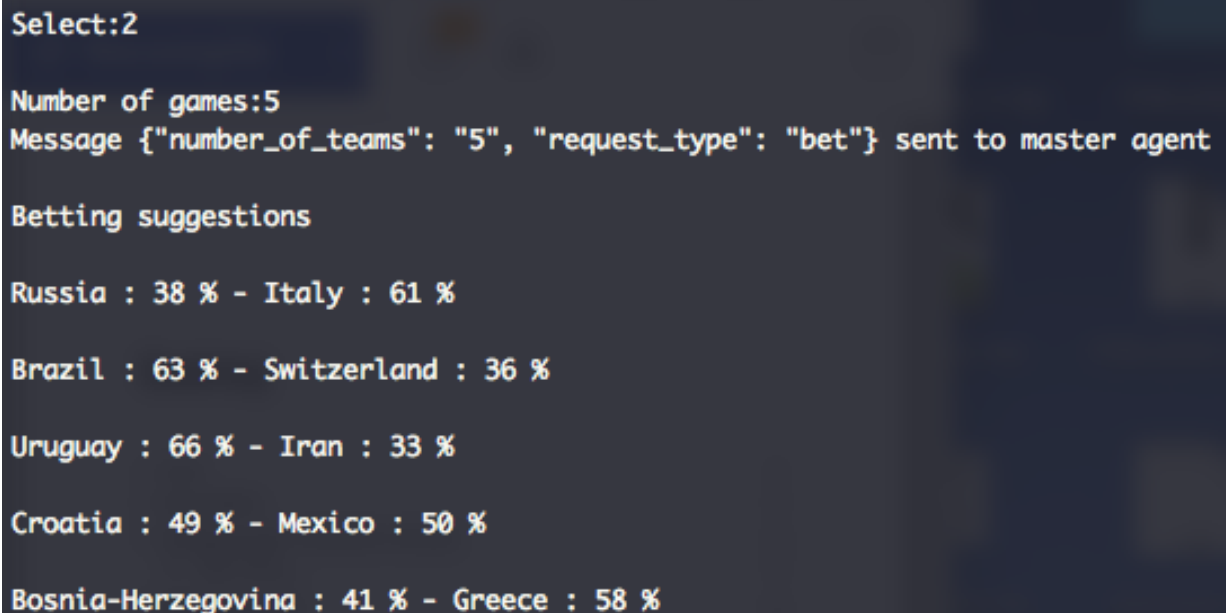
Slijed rada agenta klijenta je opisan sljedećim koracima:



```
λ lovro [Python/pybookie/client] at [?]master !  
→ python client.py  
  
Agent client@127.0.0.1 is up  
Message {"request_type": "games"} sent to master agent  
  
1) Make a Free bet  
2) Generate bet  
3) Exit
```

Slika 4.2: Ponašanje agenta klijenta (vlastita izrada)

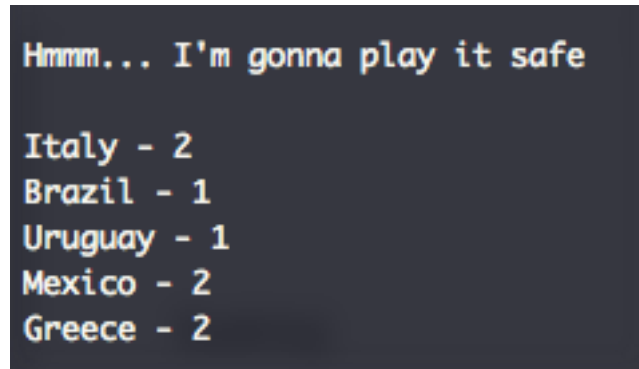
Kako je prikazano na slici 4.2 pri samom pokretanju agenta klijenta obavlja se upit na agenta evaluatora kako bi se dohvatio popis utakmica na koje se klijent može kladiti, te korisnik može odabrati dva načina rada agenta, odnosno može sam odabrati neki od kombinacija parova utakmica ili može sam kombinirati svoje parove te može prepustiti sustavu da sam izgenerira opcije za klađenje.



```
Select:2  
  
Number of games:5  
Message {"number_of_teams": "5", "request_type": "bet"} sent to master agent  
  
Betting suggestions  
  
Russia : 38 % - Italy : 61 %  
  
Brazil : 63 % - Switzerland : 36 %  
  
Uruguay : 66 % - Iran : 33 %  
  
Croatia : 49 % - Mexico : 50 %  
  
Bosnia-Herzegovina : 41 % - Greece : 58 %
```

Slika 4.3: Ponašanje agenta klijenta (vlastita izrada)

Nakon što je napravljen odabir neke od opcija ili su postavljeni parovi za odabir, šalje se zahtjev na agenta evaluatora koji obrađuje podatke te daje svoj prijedlog za klađenje, te agent pregledava ponuđene opcije.



Slika 4.4: Ponašanje agenta klijenta (vlastita izrada)

Nakon što je klijent obradio i pregledao predložene podatke o utakmicama, na temelju vlastitih preferencija, odnosno na temelju nasumičnih postavki stanja mu postavljamo ponašanje koje određuje na koji način će donositi odluke. Dostupna stanja su rizična oklada, koja u donekle prihvaća sugestije agenta evaluatora i uzima razmak od 10 posto na strani manje favorizirane momčadi te može predložiti okladu koja nije prema procjeni agenta evaluatora. Drugi način donošenja odluka se može okarakterizirati kao igranje na sigurno odnosno slijepo praćenje uputa koje se dobije od agenta klijenta. Treći način klađenja je potpuno nasumično, odnosno na slijepo, bez ikakvog utjecaja procjene rezultata.



Slika 4.5: Ponašanje agenta klijenta (vlastita izrada)

Slika 4.5 prikazuje inicijalni prikaz ponude utakmica koju agent klijent prima od agenta evaluatora pri samom pokretanju kako bi se mogle odabrati kombinacije parova koja se odabire upisivanjem rednog broja momčadi u sučelje ako što je i prikazano na slici 4.6.

```

Number of games:2
GAME 1.

Team A id: 2
Team B id: 3
GAME 2.

Team A id: 2
Team B id: 4
Message {"request_type": "team_selection", "teams": [{"teamA": "2", "teamB": "3"}, {"teamA": "2", "teamB": "4"}]} sent to master agent
    
```

Slika 4.6: Ponašanje agenta klijenta (vlastita izrada)

Na slici 4.7 možemo vidjeti konačni rezultat obrade sa strane klijenta, odnosno možemo vidjeti prijedloge koje je klijent primio od evaluatora u postocima. Zatim slijedi klijentova odluka na koji će način prema svojim preferencijama zaigrati. U ovom slučaju se ispisuju nazivi pobjedničke momčadi te ovisno o prijedlogu oklade 1,2 ili X. Oklada s brojem 1 označava pobjedu domaćina, broj 2 označava pobjedu gostiju dok X označava neriješen rezultat.

```

Betting suggestions

Croatia : 49 % - Mexico : 50 %

Croatia : 57 % - Cameroon : 42 %

Hmmm... I'm gonna play it safe

Mexico - 2
Croatia - 1

*****Results*****

Croatia 1-3 Mexico
    
```

Slika 4.7: Ponašanje agenta klijenta (vlastita izrada)

U posljednjem koraku na slici 4.7 se uz sve već navedeno i ispisuje stvarni rezultat utakmice ako je utakmica ikad zapravo i bila odigrana.

4.2 Agent evaluator

Usprkos bitnoj ulozi agenta evaluatora, njegovo ponašanje nije prikazano detaljnim ispisom koraka te agent evaluator ispisuje poruke samo u slučajevima kada pošalje poruku nekom od agenata klijenata kao što se i može vidjeti na slici 4.8. Agent evaluator je nositelj većine logike aplikacije stoga njegova funkcija i ne treba biti grafička nego obrada i predlaganje rješenja.

```
λ lovro [Python/pybookie/server] at [?]master !  
→ python server.py
```

```
Agent bookie@127.0.0.1 is up
```

```
Message sent to: client@127.0.0.1 !
```

```
Message sent to: client@127.0.0.1 !
```

```
Message sent to: client@127.0.0.1 !
```

```
Message sent to: client@127.0.0.1 !
```

Slika 4.8: Ponašanje agenta evaluatora (vlastita izrada)

Poglavlje 5

Kritički osvrt

Predviđanje ishoda stohastičkih događaja je kompleksan problem koji zahtjeva kombiniranje različitih tehnika prikupljanja i obrade podataka kako bi se njihovom kombinacijom moglo doći do koliko toliko relevantnih i pouzdanih podataka, iako nikad neće postojati garancija da će ti podaci biti u potpunosti točni. U ovom radu je preuzet relativno jednostavan pristup koji se temelji na povijesnim podacima te zasigurno ima puno još mnogo područja na kojima valja poboljšati postojeći algoritam ali čvrsta osnova na kojoj se dalje može graditi i poboljšavati algoritam je stvorena i objašnjena u ovom radu. Smatram da bi se uključivanjem još nekolicine algoritama u obradu podataka, kao i popunjavanjem još nekolicine statističkih kategorija moglo doći do još relevantnijih podataka koji bi povećali točnost predviđanja koja je i ovako naivnim pristupom relativno dobra.

Poglavlje 6

Zaključak

Smatram da je moj pristup problemu zapravo dosta naivan ali se pokazao relativno učinkovit iz razloga što je pogađao točan rezultat u oko 60 posto slučajeva prema procjeni autora. Osim toga primarni cilj je bio izraditi dobru podlogu za daljnji razvoj sustava i nadogradnje kroz kombinacije različitih algoritama i pristupa analize podataka koji bi se mogli implementirati u agenta evaluatora. Također ovaj primjer može poslužiti kao buduća osnova za izradu sustava koji simuliraju ponašanje igrača kladenja na sreću.

Sam algoritam koji je korišten je u potpunosti heuristički i temelji se na povijesnim podacima i trenutnom okruženju igrača, te se ne oslanja na niti jednu od poznatih metoda već i to jedno područje napretka cijelog sustava. Isto tako postoji niz parametara koji nisu pokriveni poput vremena odigravanja susreta, ozljeda, vremenske prognoze, golgeterske forme, tržišne vrijednosti igrača i slično. Usprkos svim nedostacima i činjenici da se radi o stohastičkim okruženjima s velikom dozom sigurnosti možemo utvrditi da ne postoji najbolji algoritam, stoga je važno poznavati različite algoritme i domenu koju želimo simulirati kako bi naš konačan rezultat bio uspješan a to je prognozirati rezultate nogometnih utakmica s sigurnošću većom od 80 posto.

Bibliografija

2010. Stohastički procesi u ekonomiji.

F., J., 2013. Samoorganizacija i izranjajuće ponašanje u višeagentnim sustavima. Master's thesis. Available at <https://bib.irb.hr/datoteka/637882.1-JakeliFraneDiplomskiRad2013.pdf>.

M, W., 2002. An Introdution to MultiAgent Systems. John Wiley and Sons. Available at <http://coltech.vnu.edu.vn/http/media/courses/AI++/Tai20lieu/TLTK.pdf>.

Predovan, L., 2009. *The Bees Algorithm, and Its Applications*. Available at <https://www.google.hr/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=0ahUKEwj16XplaDKAhXChA8KHVhMD8kQFggZMAA&url=http%3A%2F%2Fwww.fi.muni.cz%2Fusr%2Fpopelinsky%2Flectures%2Fkdd%2FBees%2520algorithm.ppt&usg=AFQjCNEBdG0wen--Xz6hOLV3M81YWvnRkQ&sig2=mrvkliEWLIJ6fzd4KwHbkQ>.

Schatten, M., 2008. Prezentacije s predavanja. M.sc. diss., Faculty of Organization and Informatics, Varaždin.

Wikipedia, 2016. *Stochastic*. [Online; accessed 20.06.2016].

Poglavlje 7

Programski kod

7.1 Agent klijent

7.1.1 client.py

```
# coding=utf-8
# !/usr/bin/env python
import json
import random
import sys

import spade
from spade.ACLMessage import ACLMessage
from spade.Agent import Agent
from spade.Behaviour import ACLTemplate, MessageTemplate, Behaviour

from evaluator import Evaluator
from print_formatter import PrintFormatter

class ClientAgent(Agent):
    class BookingSettings(Behaviour):

        dialog_selection = None
        msg = None
        games = None

        # 1 – risky , 2 – sure thing , 3 – I don't know what I'm doing
        mood = random.choice([1, 2, 3])

        def _process(self):
            self.msg = self._receive(True)
            if self.msg:
```

```
        request = json.loads(self.msg.content)
        if request['request_type'] == 'games':
            self.games = request['data']
            self.show_dialog()
        if request['request_type'] == 'game_evaluation':
            PrintFormatter.results(request['data'])
            Evaluator.make_bet(self.mood, request['data'])
            print "\n*****Results*****\n"
            Evaluator.find_result(request['data'])
            self.show_dialog()

    def show_dialog(self):
        self.dialog_selection = raw_input("\n1) Make_a_Free_bet\n2) Generate_bet\n")
        if self.dialog_selection == '1':
            self.set_free_bet_preferences()

        if self.dialog_selection == '2':
            self.set_bet_preferences()

        if self.dialog_selection == '3':
            self.stop_agent()

    def stop_agent(self):
        print "Agent_is_dying..."
        self.kill()
        sys.exit()

    def set_bet_preferences(self):
        preferences = None
        number_of_teams = 0

        while number_of_teams == 0 and number_of_teams < 16:
            number_of_teams = raw_input('\nNumber_of_games:')
            preferences = {'request_type': 'bet', 'number_of_teams': number_of_teams}

        self.send_message(json.dumps(preferences))

    def set_free_bet_preferences(self):
        if self.games:
            PrintFormatter.games(self.games)

        teams = []
        number_of_teams = input("\nNumber_of_games:")

        for i in range(0, int(number_of_teams)):
```

```
        print "GAME_%d_\n" % (i + 1)
        team_a = raw_input("\nTeam_A_id:")
        team_b = raw_input("\nTeam_B_id:")

        teams.append({'teamA': team_a, 'teamB': team_b})

    result = json.dumps({'request_type': 'team_selection', 'teams': teams})
    self.send_message(result)

    def send_message(self, content):
        master_agent = spade.AID.aid(name="bookie@127.0.0.1", addresses=["xmpp://"])
        self.msg = ACLMessage()
        self.msg.setPerformative("inform")
        self.msg.setOntology("booking")
        self.msg.setLanguage("eng")
        self.msg.addReceiver(master_agent)
        self.msg.setContent(content)
        self.myAgent.send(self.msg)
        print 'Message_%s_sent_to_master_agent' % content

    def _setup(self):
        print "\nAgent\t" + self.getAID().getName() + " is up"

        feedback_template = ACLTemplate()
        feedback_template.setOntology('booking')

        mt = MessageTemplate(feedback_template)
        settings = self.BookingSettings()
        self.addBehaviour(settings, mt)

        settings.send_message(json.dumps({'request_type': 'games'}))

if __name__ == '__main__':
    p = ClientAgent('client@127.0.0.1', 'booking')
    p.start()
```

7.1.2 evaluator.py

```
import os
import random

class Evaluator:
    BASE_DIR = os.path.dirname(os.path.dirname(__file__))
```

```
path = BASE_DIR + '/client/results'

def __init__(self):
    pass

@staticmethod
def find_result(data):
    if os.path.isfile(Evaluator.path):
        f = open(Evaluator.path)

        for game in data:
            team_a = game['result'].split('_')[0]
            team_b = game['result'].split('_')[5]

            for line in f:
                compare_a = line.split('_')[0]
                compare_b = line.split('_')[2]

                if (str(team_a.strip()) == str(compare_b.strip()) or str(team_a.s
                    and (str(team_b.strip()) == str(compare_b.strip()) or str
                        compare_a.strip())):
                    print line

@staticmethod
def make_bet(mood, data):
    bets = []
    # risky
    try:

        if mood == 1:
            print "\nI'm gonna play it risky\n"
            for game in data:
                team_a = game['result'].split('_')[0]
                team_a_rating = int(game['result'].split('_')[2])
                team_b = game['result'].split('_')[5]
                team_b_rating = int(game['result'].split('_')[7])

                if team_a_rating > team_b_rating or (team_b_rating - team_a_rating
                    bet = '%s_-%d' % (team_a, 1)
                    bets.append(bet)

                elif team_a_rating < team_b_rating or (team_a_rating - team_b_rati
                    bet = '%s_-%d' % (team_b, 2)
                    bets.append(bet)
```

```
        elif team_a_rating == team_b_rating:
            bet = '%s--%s--%s' % (team_a, 'X', team_b)
            bets.append(bet)

# sure
if mood == 2:
    print "\nHmmm... I 'm gonna play it safe\n"
    for game in data:
        team_a = game['result'].split('_')[0]
        team_a_rating = int(game['result'].split('_')[2])
        team_b = game['result'].split('_')[5]
        team_b_rating = int(game['result'].split('_')[7])

        if team_a_rating > team_b_rating:
            bet = '%s--%d' % (team_a, 1)
            bets.append(bet)

        elif team_a_rating < team_b_rating:
            bet = '%s--%d' % (team_b, 2)
            bets.append(bet)

        elif team_a_rating == team_b_rating:
            bet = '%s--%s--%s' % (team_a, 'X', team_b)
            bets.append(bet)

# random
if mood == 3:
    print "\nDon't have an idea what I 'm doing\n"
    for game in data:
        team_a = game['result'].split('_')[0]
        team_b = game['result'].split('_')[5]

        bet = '%s--%s' % (random.choice([team_a, team_b]), random.choice(
            bets.append(bet)

except:
    pass

Evaluator.print_bets(bets)

@staticmethod
def print_bets(bets):

    for bet in bets:
        print bet
```

7.1.3 print_formatter.py

```
import json

class PrintFormatter:
    def __init__(self):
        pass

    @staticmethod
    def games(games):
        games = json.loads(games)

        for group in games:
            print '\nGroup:' + group['group']
            for team in group['teams']:
                print "%d.\t%s" % (team['id'], team['team'])
            print '_____'

    @staticmethod
    def results(results):
        print '\nBetting suggestions\n'
        for game in results:
            print '%s\t\n' % game['result']
```

7.2 Agent evaluator

7.2.1 server.py

```
# coding=utf-8
# !/usr/bin/env python
import json
import sys

import spade
from spade.ACLMessage import ACLMessage
from spade.Agent import Agent, os
from spade.Behaviour import ACLTemplate, Behaviour

from bookie import Bookie

os.path.dirname(os.path.realpath(__file__))

class MasterBettingAgent(Agent):
    class Booking(Behaviour):
```

```
msg = None
```

```
def _process(self):
    self.msg = self._receive(True)

    if self.msg:
        request = json.loads(self.msg.content)
        if request['request_type'] == 'games':
            bookie = Bookie()
            self.send_message(json.dumps({'request_type': 'games', 'data': bo

        if request['request_type'] == 'bet':
            bookie = Bookie()
            data = bookie.make_random_evaluation(request['number_of_teams'])
            self.send_message(json.dumps({'request_type': 'game_evaluation',

        if request['request_type'] == 'team_selection':
            bookie = Bookie()
            data = bookie.make_evaluation(request['teams'])
            self.send_message(json.dumps({'request_type': 'game_evaluation',

    else:
        pass

def stop_agent(self):
    print "Agent_is_dying..."
    self.kill()
    sys.exit()

def send_message(self, message):

    client = "client@127.0.0.1"
    address = "xmpp://" + client
    receiver = spade.AID.aid(name=client, addresses=[address])

    self.msg = ACLMessage()
    self.msg.setPerformative("inform")
    self.msg.setOntology("booking")
    self.msg.setLanguage("eng")
    self.msg.addReceiver(receiver)
    self.msg.setContent(message)

    self.myAgent.send(self.msg)
    print "\nMessage_sent_to:_%s_" % client
```

```
def _setup(self):
    print "\nAgent\t" + self.getAID().getName() + "_is_up"

    template = ACLTemplate()
    template.setOntology('booking')

    behaviour = spade.Behaviour.MessageTemplate(template)
    booking = self.Booking()
    self.addBehaviour(booking, behaviour)

if __name__ == "__main__":
    mba = MasterBettingAgent('bookie@127.0.0.1', 'booking')
    mba.start()
```

7.2.2 bookie.py

```
from random import randint
```

```
from sources import football_db
```

```
class Bookie:
    def __init__(self):
        pass

    @staticmethod
    def make_random_evaluation(number_of_teams):
        result = []
        number_of_teams = int(number_of_teams)
        for num in range(0, number_of_teams):
            result.append({'teamA': randint(1, 32), 'teamB': randint(1, 32)})

        return Bookie.make_evaluation(result)

    @staticmethod
    def make_evaluation(data):
        results = []
        for team in data:
            team_a = football_db.FootballIDB.get_team_by_id(team['teamA'])
            team_b = football_db.FootballIDB.get_team_by_id(team['teamB'])

            if team_a and team_b:

                if team_a == team_b:
                    results.append({'result': ('SAME_TEAM_%s_50%%' % team_a)})
```


continue

```
team_a_sum = footbal_db.FootballDB.get_wc_titles(team_a) * 200
team_b_sum = footbal_db.FootballDB.get_wc_titles(team_b) * 200

team_a_sum += footbal_db.FootballDB.get_ranking(team_b)
team_b_sum += footbal_db.FootballDB.get_ranking(team_a)

team_a_sum += footbal_db.FootballDB.get_wc_games_played(team_a)
team_b_sum += footbal_db.FootballDB.get_wc_games_played(team_b)

team_a_sum += footbal_db.FootballDB.get_won_wc_games_played(team_a) *
team_b_sum += footbal_db.FootballDB.get_won_wc_games_played(team_b) *

team_a_sum += footbal_db.FootballDB.get_draw_wc_games_played(team_a)
team_b_sum += footbal_db.FootballDB.get_draw_wc_games_played(team_b)

team_a_sum -= footbal_db.FootballDB.get_lost_wc_games_played(team_a) *
team_b_sum -= footbal_db.FootballDB.get_lost_wc_games_played(team_b) *

team_a_sum += footbal_db.FootballDB.get_goal_difference_wc_games_played(team_a)
team_b_sum += footbal_db.FootballDB.get_goal_difference_wc_games_played(team_b)

team_a_sum += footbal_db.FootballDB.get_wc_points(team_a)
team_b_sum += footbal_db.FootballDB.get_wc_points(team_b)

team_a_sum += footbal_db.FootballDB.get_wc_participations(team_a)
team_b_sum += footbal_db.FootballDB.get_wc_participations(team_b)

team_a_sum += footbal_db.FootballDB.get_wc_team_player_ratings(team_a)
team_b_sum += footbal_db.FootballDB.get_wc_team_player_ratings(team_b)

total_sum = team_a_sum + team_b_sum

result = '%s: %d%%- %s: %d%%' % (
    team_a, float(team_a_sum) / float(total_sum) * 100, team_b,
    float(team_b_sum) / float(total_sum) * 100)
results.append({'result': result})
else:
    return 'INVALID_TEAM_CODE'

return results
```

```
@staticmethod
def get_games():
```

```
        return fotbal_db.FootballDB.get_games()
```

7.2.3 football db.py

```
# coding=utf-8
```

```
import json
```

```
import os
```

```
class FootballDB:
```

```
    BASE_DIR = os.path.dirname(os.path.dirname(__file__))
```

```
    groups_file = BASE_DIR + '/sources/groups.json'
```

```
    wc_history_file = BASE_DIR + '/sources/wc_history'
```

```
    wc_team_file = BASE_DIR + '/sources/squads/'
```

```
    top_teams = ['RealMadrid(ESP)', 'Barcelona(ESP)', 'Chelsea(ENG)', 'ManchesterCity(ENG)',  
                 'BayernMunich(GER)', 'Internazionale(ITA)', 'Napoli(ITA)', 'ManchesterUnited(ENG)',  
                 'Liverpool(ENG)', 'Juventus(ITA)', 'BorussiaDortmund(GER)', 'AtleticoMadrid(ESP)']
```

```
    def __init__(self):
```

```
        pass
```

```
    @staticmethod
```

```
    def get_team_by_id(team_id):
```

```
        data = json.loads(FootballDB.get_games())
```

```
        result = None
```

```
        for group in data:
```

```
            for team in group['teams']:
```

```
                if int(team['id']) == int(team_id):
```

```
                    result = team['team']
```

```
        return result
```

```
    @staticmethod
```

```
    def get_ranking(team_name):
```

```
        return int(FootballDB.get_wc_history(team_name, 0))
```

```
    @staticmethod
```

```
    def get_wc_games_played(team_name):
```

```
        return int(FootballDB.get_wc_history(team_name, 2))
```

```
    @staticmethod
```

```
    def get_won_wc_games_played(team_name):
```

```
        return int(FootballDB.get_wc_history(team_name, 3))
```

```
@staticmethod
def get_draw_wc_games_played(team_name):
    return int(FootballDB.get_wc_history(team_name, 4))

@staticmethod
def get_lost_wc_games_played(team_name):
    return int(FootballDB.get_wc_history(team_name, 5))

@staticmethod
def get_goal_difference_wc_games_played(team_name):
    gd = FootballDB.get_wc_history(team_name, 6)
    gd = gd.split(':')

    goals_for = int(gd[0])
    goals_against = int(gd[1])

    return goals_for - goals_against

@staticmethod
def get_wc_points(team_name):
    return int(FootballDB.get_wc_history(team_name, 7))

@staticmethod
def get_wc_participations(team_name):
    return int(FootballDB.get_wc_history(team_name, 8))

@staticmethod
def get_wc_titles(team_name):
    titles = FootballDB.get_wc_history(team_name, 9)
    try:
        if titles.isalpha() and int(titles) != 0:
            titles = titles[0]
            return int(titles)
        else:
            return 0
    except Exception:
        return 0

@staticmethod
def get_wc_history(team, result_row_index):
    path = FootballDB.wc_history_file
    if os.path.isfile(path):
        f = open(path)

        for line in f:
```

```
        if line[0].isdigit():
            row = line.replace('\n', '')
            row = row.replace('_', '')
            row = row.split('|')
            if row[1] == team.replace('_', '-'):
                f.close()
                try:
                    return row[result_row_index]
                except BaseException:
                    return 0

    @staticmethod
    def get_wc_team_player_ratings(team):
        path = '%s%s.txt' % (FootballDB.wc_team_file, (team.replace('_', '-')))
        path = path.lower()
        team_rating = 0
        if os.path.isfile(path):
            f = open(path)

            for line in f:
                try:
                    row = line.split('###')
                    row = row[1].replace('_', '').split(',')

                    team_rating += int(row[0])
                    team_name = row[1].replace('\n', '')

                    if team_name in FootballDB.top_teams:
                        team_rating += 10

                except Exception:
                    pass

            return team_rating

    @staticmethod
    def get_games():
        data = None
        path = FootballDB.groups_file
        if os.path.isfile(path):
            with open(path, 'r') as football_teams:
                data = football_teams.read().replace('\n', '')

        return data
```