

**SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN**

Filip Gatarić

IGRA "ČOVJEČE NE LJUTI SE" UZ POMOĆ PROAKTIVNIH AGENATA

PROJEKT

Varaždin, 2020.

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
V A R A Ž D I N

Filip Gatarić

Matični broj: 42676/13–R

Studij: Informacijsko i programsko inženjerstvo

IGRA "ČOVJEČE NE LJUTI SE" UZ POMOĆ PROAKTIVNIH AGENATA

PROJEKT

Mentor/Mentorica:

Izv. prof. dr. sc. Markus Schatten

Varaždin, kolovoz 2020.

Izjava o izvornosti

Izjavljujem da je moj projekt izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Glavni cilj ovog rada je prikazati način na koji se igranje igre "Čovječe ne ljuti se", poznatija pod nazivom "Ludo", može ostvariti putem proaktivnih agenata. Igru "Čovječe ne ljuti se" ćemo u radu dalje oslovljavati kao "Ludo" zbog jednostavnosti i kratkoće naziva. Igrači igre "Ludo" su proaktivni agenti koji samostalno pokušavaju sve svoje pijune (4 pijuna) prebaciti u kućni stupac bacanjem kocke, koja određuje broj pomaka pijuna. Kako je cilj ove igre prebaciti svih četiri pijuna iz početnog kruga u završni kućni stupac, potrebno im je dati neku strategiju kojom igraju, tj. pomiču svoje pijune kako bi što prije došli do cilja ili "pojeli" pijune drugih igrača i time ih vratili na početne pozicije. Svi igrači agenti komuniciraju sa "game master" agentom koji predstavlja igraču ploču po kojoj igrači agenti igraju. Projekt je kreiran pomoću Python programskog jezika u kojem je sva komunikacija i rad agenata implementiran pomoću Spade biblioteke. Svi agenti (igrači i igra) su registrirani i međusobno komuniciraju putem Jabbim servisa.

Ključne riječi: višeagentni sustavi, agent, model, python, spade, ludo, društvene igre

Sadržaj

1. Uvod	1
2. Metode i tehnike rada	2
3. Višeagentni sustavi	3
3.1. Agent	3
3.2. Čvorovi	3
3.3. Komunikacija	4
3.4. Stanje	4
3.5. Višeagentni sustavi i društvene igre	4
4. SPADE	5
4.1. Komunikacija	5
4.1.1. Predlošci	5
4.1.2. Slanje i primanje poruka	5
4.2. Ponašanje	5
5. Čovječe ne ljuti se - Ludo	7
5.1. Pravila	7
5.2. Ploča	7
5.3. Implementacija	8
5.3.1. Arhitektura projekta	9
5.3.2. Inicijalizacija igre	11
5.3.2.1. Dodaj igrača	11
5.3.3. Započni igru	12
5.3.3.1. Igraj potez	15
5.3.3.2. Informiraj igrača	15
5.3.3.3. Čekaj svoj red	17
5.3.3.4. Pomakni odabrani pijun	18
5.3.4. Crtanje igrača ploče	19
6. Zaključak	22
Popis literature	23
Popis slika	24

1. Uvod

"Čovječe ne ljuti se" je igra s mnogo naziva i varijacija u kulturama diljem svijeta. Iako je u svakom kutu svijeta naziv drugačiji, svi mi znamo kako odigrati partiju "Čovječe ne ljuti se" bez da se naljutimo na suigrača koji nam je pojeo pijuna pred kućicom (ovaj zadnji dio možda i ne). Pošto je ova igra globalno popularna, odlučio sam se igru "Čovječe ne ljuti se", poznatiju pod nazivom "Ludo", implementirati putem proaktivnih agenata. Implementacija se nije činila dosta zahtjevno pošto je i sama igra vrlo jednostavna za igranje i bez puno kompliciranih pravila o kojima će malo kasnije biti više riječi.

Interesantan dio planiranja ovakvog projekta je podijeliti poslove između agenata igrača i igre pošto u stvarnom svijetu igrači sve rade sami, svako za sebe. Svako baca kocku za sebe, svako odabire koji će pijun pomaknuti i kreira vlastitu strategiju kojom će što prije doći do pobjede. Ploča služi samo za prikaz trenutnog napretka svakog od igrača. Bilo je jako interesantno raspodijeliti odgovornosti i zaduženja ploče i igrača u ovom višeagentnom sustavu.

Za tehnologije kojima će se ostvariti ovaj sustav sam odabrao već poznate tehnologije koje smo koristili na vježbama. Korištene tehnologije su Spade (eng. Smart Python multi-Agent Development Environment) biblioteka koja se koristi za izradu i komunikaciju između agenata i Python programski jezik koji podržava navedenu biblioteku i u kojem je implementiran cijeli projekt. Pošto sam imao problema s postavljanjem lokalnog XAMPP servera preko kojeg bi agenti komunicirali, odlučio sam se za Jabbim servise. Svi agenti su registrirani i komuniciraju preko XAMPP/Jabber servera, što se pokazalo kao odlična supstitucija za lokalni XAMPP server.

2. Metode i tehnike rada

Za izradu višeagentnog sustava za igranje igre "Ludo" ili "Čovječe ne ljuti se" koristi će se SPADE platforma u Python-u. Pomoću SPADE-a možemo razvijati agente koji mogu komunicirati s drugim agentima ali i s ljudima. Kreirati će se dvije vrste agenta(igrača ploča i igrač). Igrača ploča će komunicirati s četiri agenta(crveni, plavi, zeleni, žuti) koji predstavljau četiri boje na ploči. Agenti su se registrirali na kao jabb.im klijenti zbog određenih poteškoća sa lokalnim poslužiteljem.

Pošto je cijeli projekt rađen u Windows 10 operacijskom sustavu, korištena je aplikacija Anaconda Navigator i Visual Studio Code za rad sa Python-om. Anaconda omogućuje pokretanje aplikacija i jednostavno upravljanje conda paketima, okruženjima i kanalima bez korištenja naredbi iz terminala, a je za Windows, macOS i Linux.

Za slanje podataka putem poruka korištena je jsonpickle biblioteka. Jsonpickle uvelike olakšava pretvaranje podataka u JSON format te isto tako šifriranje i dešifriranje istih prije i poslije slanja/primanja poruka.

Iako je vizualizacija igre "Čovječe ne ljuti se" inicijalno bila planirana putem Tkinter-a, morao sam napustiti tu ideju pošto sam naišao na par problema s Tkinter-om, a vizualizaciju sam na kraju ostvario putem konzole.

3. Višeagentni sustavi

Višeagentni sustav (eng. Multy-Agent System aka MAS) je računalni sustav sastavljen od više inteligentnih agenata koji su u međusobnoj interakciji. Višeagentne sustave ćemo u nastavku zvati MAS. Računanje temeljeno na agentima inovativno je potpolje umjetne inteligencije i informatike koje pruža teorije za razumijevanje ponašanja distribuiranih sustava s više agenata te pruža metode i algoritme za kontrolu pojedinih komponenti takvog sustava i protokola koji upravljaju njihovim interakcijama. Iako je primjena višeagentnih sustava iznimno uspješno u istraživačkim krugovima, još uvijek se bori za širu industrijsku primjenu. Najveće ograničenje masovnog iskorištavanja jest što bi testiranja na složenim sustavima visokih razmjera bili iznimno skupi i mnogim slučajevima rizični. Sofisticirano višeagentno modeliranje i simulacije mogu pružiti visoke točnosti, visoke performanse i skalabilne računske modele ciljnih aplikacija. Modeli se mogu koristiti za empirijsko provjeravanje algoritama, metoda i teorija agenata, a isto tako se mogu koristiti i kao mehanizam za prijenos tehnologije.[1]

3.1. Agent

Agent je računalni sustav koji je u stanju ponašati se autonomno u nekom okružju kako bi postigao ciljeve svog oblikovanja.[2] Agenti su ključni elementi u višeagentnim sustavima, kao što i samo ime govori. U MAS sustavima agenti su entiteti u kojima se nalazi sve funkcionalnosti specifične za aplikaciju. Agenti pružaju životni ciklus za implementaciju usluge te su u stanju opažati i komunicirati. Kako sve ovo dosad navedeno definira svojevršno ljudsko ponašanje, dobra metafora bi bila apstrakcija ljudskog lika. Jednostavna i česta metafora, koja je u osnovi apstrakcija ljudskog oblika tijela, je figurica iz društvene igre "Ludo" koja je kod nas prevedena i poznata kao "Čovječe ne ljuti se".[1]

Sljedeće značajke omogućuju agentima široku primjenjivost i rješavanje složenih zadataka:[3]

- Socijalizacija: Agenti mogu i dijeliti svoje znanje i tražiti informacije od drugih agenata za poboljšanje performanse u postizanju svojih ciljeva.
- Autonomija: Svaki agent može samostalno izvršiti postupak donošenja odluka i poduzimati odgovarajuće mjere
- Proaktivnost: Svaki agent koristi svoju povijest, parametre i informacije drugih agenata za predviđanje moguće buduće radnje. Ta predviđanja omogućavaju agentima poduzimati učinkovite akcije koje pomažu njihovim ciljevima. Ova sposobnost podrazumijeva da isti agent može uzeti različite radnje kada se postavi u drva različita okruženja.

3.2. Čvorovi

Agentni čvorovi u MAS su okruženja za agente u jednom sustavu. Oni pružaju infrastrukturu potrebnu za izvršavanje funkcionalnosti u agentima tako da agenti mogu surađivati

kako bi realizirali složene aplikacije.[1]

3.3. Komunikacija

Komunikacija je jedno od ključnih obilježja interakcije u MAS. Česta metafora za komunikaciju porukama je omotnica s pismom. Sama poruka vizualizira se kao pismo koje se kreće između komunikacijskih partnera. Kako bi se osiguralo da je događaj komunikacije vidljiv dovoljno dugo da ga korisnik može uhvatiti, djelomično prozirna linija koristi se kao metafora za prikaz komunikacijskog kanala između izvora i cilja. Ta će se linija polako otapati do nevidljivosti ako se ne prenesu nove poruke, što stvara učinak da će se često korišteni komunikacijski kanali pojaviti daleko jači od rijetko korištenih.[1]

3.4. Stanje

Stanja entiteta se mogu zamisliti kao način bojanja dijelova površine reprezentacije entiteta. Kako je stanje entiteta detaljna informacija o određenom entitetu, vizualiziranje stanja kao boje je poželjnije i razumljivije u odnosu na promjenu oblika. Jedan od koncepata vizualizacije agenata i čvorova u MAS je ASGARD(eng. A Graphical Monitoring Tool for Distributed Agent Infrastructures) koncept. ASGARD koristi boje za vizualizaciju stanja u agentovim "glavama", koje su obojane prema stanju životnog ciklusa agenta, tako da crvena glava označava zaustavljenog agenta, a zelena glava agenta u radu.[1]

3.5. Višeagentni sustavi i društvene igre

Računalni programi koji mogu igrati različite vrste igara postaju rastuće područje interesa za računalnu industriju igara s povećavanjem potražnje za bolje kvalificiranim računalnim protivnicima. Jedno je istraživanje s Blekinge Instituta za Tehnologiju pokazalo kako MAS može biti prikladan kandidat za stvaranje konkurentnog robota koji može igrati igre s visokim brojem faktora. Društvene igre na kojima se MAS testirao su igre Diplomacija i Rizik. Općenita ideja je imati višeagentni sustav gdje svaka važna jedinica igrača na ploči odgovara agentu u sustavu. Ti agenti mogu odgovarati bilo komadima ili teritorijama igrača. Agenti će se dogovarati o tome koje akcije treba poduzeti u sustavu, možda uz pomoć posrednika koji se brine o stvarnim pregovorima putem pregovaračkog mehanizma. Unatoč uspjehu u tim izoliranim domenama, istraživači su prepoznali potrebu za više istraživanja unutar područja kako bi izvukli daljnje zaključke.[4]

4. SPADE

SPADE je akronim od Smart Python Agent Development Environment. To je platforma za MAS napisana u Python-u i zasnovana na trenutnoj razmjeni poruka(XAMPP). Pomoću SPADE-a možemo razvijati agente koji mogu razgovarati s drugim agentima i s ljudima.

Agent se u osnovi sastoji od mehanizma za vezu s platformom, dispečera poruka i niza različitih ponašanja kojima dispečer daje poruke. Svakom agentu potreban je identifikator koji se zove Jabber ID(JID) i važeća lozinka za uspostavljanje veze s XAMPP poslužiteljem. JID, koji je sastavljen od korisničkog imena, @ i domene poslužitelja, biti će ime koje identificira agenta u platformi, npr. myagent@myprovider.com

Kako bi instalirali SPADE, potrebno je pokrenuti sljedeću naredbu u terminalu:

```
$ pip install spade
```

Ovo je preferirani način instaliranja SPADE-a jer će uvijek instalirati najnoviju stabilnu verziju. SPADE se izvršava na Pythonu i baziran je na istom, a za potrebe SPADE-a potrebno je koristiti minimalno 3.6 verziju Python-a. [5]

4.1. Komunikacija

4.1.1. Predlošci

Predlošci se u SPADE koriste za slanje primljenih poruka ponašanju koje čeka tu poruku. Kod dodavanja ponašanja, može se postaviti predložak za to ponašanje, koji agentu može poslati poruku koju je agent primio u tom registriranom ponašanju. Predložak ima iste atribute kao i poruka i svi atributi moraju biti jednaki u poruci da bi se mogli podudarati.[5]

4.1.2. Slanje i primanje poruka

Kao što je već i prije navedeno, poruke su osnova svakog MAS-a stoga je vrlo važno razumjeti koji su sadržaji pristupni u SPADE za rad s porukama agenata. Klasa Message se može koristiti za kreiranje novih poruka s kojima se može raditi i pruža metodu za uvođenje metapodataka u poruke. Kad je poruka spremna za slanje, može se prenijeti na metodu send() koja će pokrenuti postupak unutarnje komunikacije da poruku zaista pošalje na njezino odredište. Funkcija slanja je asinhronski program pa je treba zvati sa await naredbom.[5]

4.2. Ponašanje

SPADE Agent klase implementiraju nekoliko podklasa koje predstavljaju ponašanja koje agent može imati. Ponašanja koje agent može koristiti su:

- CyclicBehaviour - Ovo se ponašanje izvršava ciklično dok se ne zaustavi.

- OneShotBehaviour - Ovo se ponašanje izvršava samo jednom
- PeriodicBehaviour - Ovo se ponašanje izvršava periodično s intervalom
- TimeoutBehaviour - Ovo ponašanje se izvršava jednom nakon navedenog datumskog vremena
- State - Stanje FSMBehaviour je OneShotBehaviour
- FSMBehaviour - Ponašanje sastavljeno od stanja (ponašanja) koja mogu prelaziti iz jednog stanja u drugo.

5. Čovječe ne ljuti se - Ludo

Ludo (od latinskog ludo, što znači igram) je strateška igra na ploči za dva do četiri igrača, u kojoj igrači utrkuju svoja četiri tokena od početka do cilja pomičući ga za broj mjesta određen bacanjem kocke. Ludo je derivacija indijske igre Pachisi, ali pojednostavljena. Igra ima mnogo varijacija i popularna je u mnogim zemljama i poznata pod raznim imenima. U hrvatskoj je poznata pod nazivom "Čovječe ne ljuti se".

5.1. Pravila

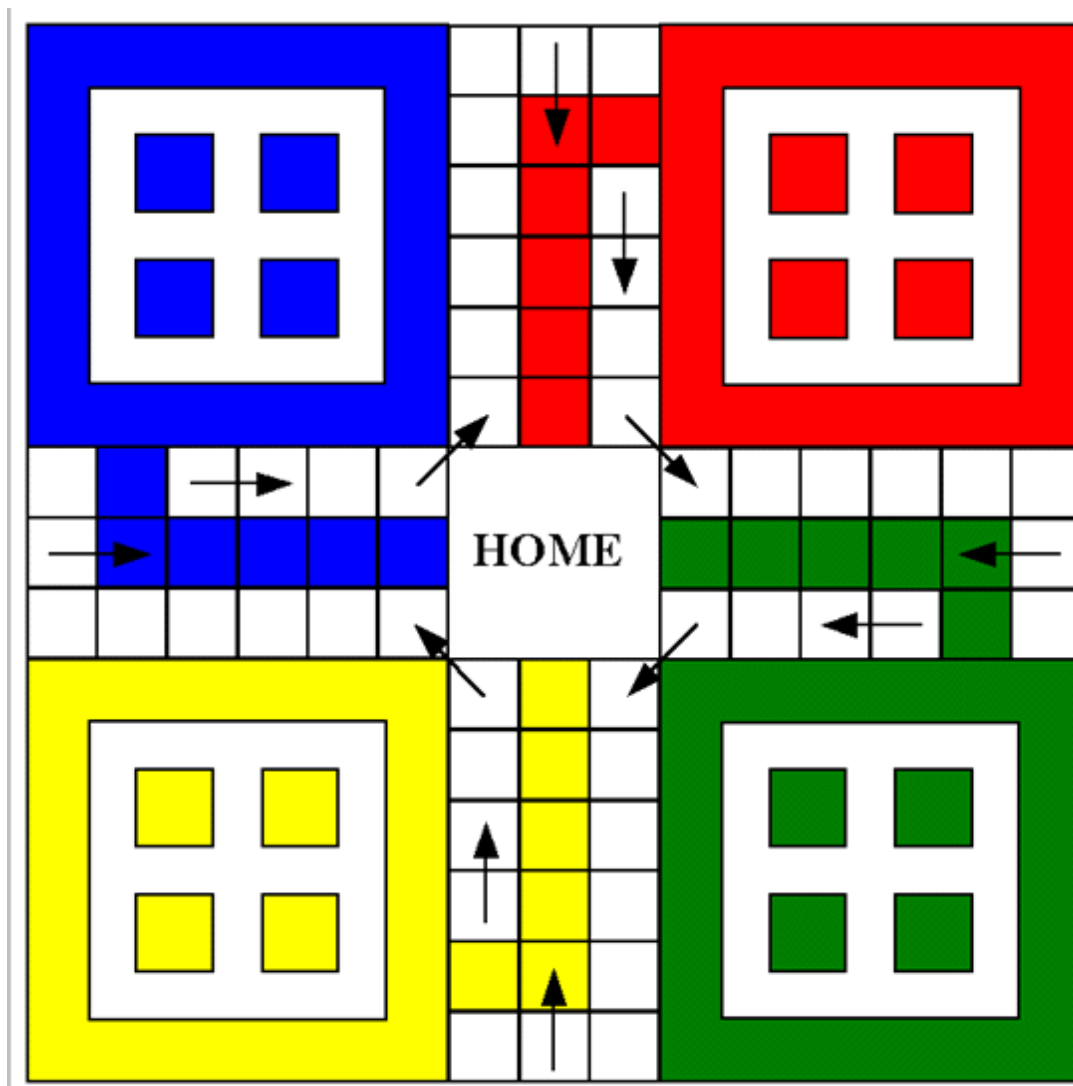
- Igrači se izmjenjuju u smjeru kazaljke na satu. Igru započinje igrač s najvećim bacanjem kocke.
- Svako bacanje, igrač odlučuje koji će token pomaknuti. Token se jednostavno kreće u smjeru kazaljke na satu oko staze dobivene brojem bačene kocke. Ako se niti jedan komad ne može legalno pomaknuti za broj mjesta dobivenih bacanjem kocke, igra prelazi na sljedećeg igrača.
- Bacanje 6 daje još jedan potez/bacanje.
- Igrač mora baciti 6 da bi pomaknuo token iz početnog kruga na prvi kvadrat na stazi. Komad se kreće 6 kvadrata oko kruga, počevši s odgovarajuće obojenim startnim kvadratom (a igrač tada ima još jedan potez).
- Ako token sleti na polje s tokenom druge boje(token drugog igrača), token koji se nalazio na tom polju vraća se u svoj početni krug.
- Ako token sleti na token iste boje(vlastiti token), to kreira blok. Ovaj blok ne može proći ili na njega sletjeti nijedan protivnički token.

Kad je token obišao ploču, nastavlja se kretati prema kućnom stupcu. Token se može premjestiti na kućni trokut samo preciznim bacanjem. Pobjeđuje prva osoba koja je preselila sva 4 tokena u kućni trokut.[6]

5.2. Ploča

Ploča Ludo je četvrtasta s uzorkom na obliku križa, a svaka ruka podijeljena je u tri susjedna stupa od osam kvadrata. Srednji kvadrati čine kućni stup za svaku boju i u njih ne mogu ući tokeni druge boje. Sredina križa tvori veliki kvadrat koji je područje 'doma' i koji je podijeljen na 4 kućna trokuta, po jedan od svake boje. Na svakom uglu, odvojeno od glavnog kruga, nalaze se krugovi u boji (ili kvadratići), gdje se tokeni stavljaju za početak. Početni kvadrat, početni krug, kućni trokut i svi kvadrati kućnog stupa obojeni su tako da odgovaraju odgovarajućim tokenima. Svaki igrač odabere jednu od 4 boje (zelena, žuta, crvena ili plava)

i smjesti 4 tokena te boje u odgovarajući početni krug. Za određivanje kretanja baca se jedna kocka. [6] Izgled igrače ploče društvene igre Ludo može se vidjeti na Slika 1.



Slika 1: Ploča igre Ludo (Izvor: http://pachisi.vegard2.net/ludo_big.gif, 2020)

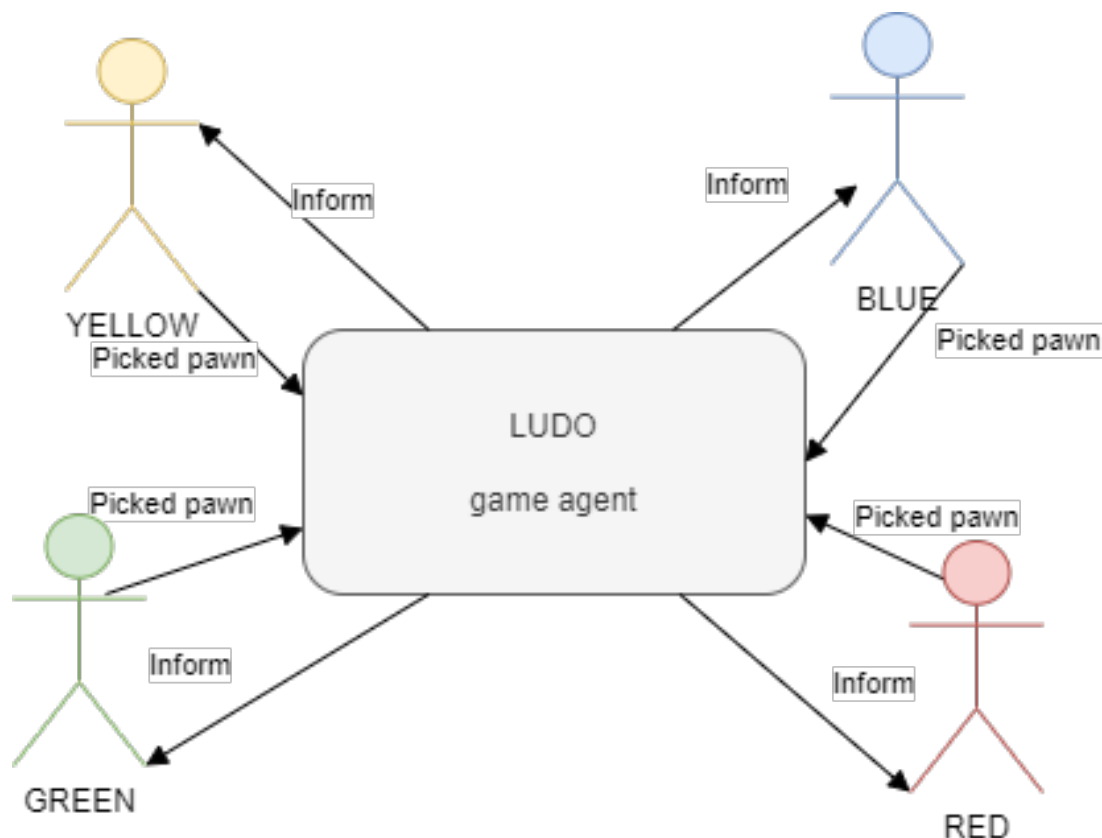
5.3. Implementacija

U ovom će se poglavlju prikazati i objasniti određene funkcionalnosti koje su potrebne za uspješno izvršavanje ovog sustava. Nakon proučavanja rada i modeliranja agenata i pravila igre, odlučio sam se za implementaciju sa proizvoljnim brojem agenata (minimalno 3, maksimalno 5), u kojem je jedan agent uvijek igrača ploča, a ostali agenti su igrači. Za pokretanje igre je potrebno uključiti minimalno dva agenta igrača u igru. Pošto je cilj projekta kreirati sustav koji će samostalno moći igrati igru "Ludo" implementirani agenti automatski pripadaju u klasifikaciju proaktivnih agenata pošto u potpunosti zamjenjuju ljudskog igrača i njegove inpute.

Sustav sa maksimalnim brojem agenata (5 agenata) prikazan je na Slika 2. Na modelu se može vidjeti kako se sva komunikacija odvija preko game agenta tj. igrača ploče. Svi igrači čekaju dok ne dobiju informaciju od game agenta da je njihov red za bacanje i micanje pijuna.

Kada agent igrač primi poruku da je njegov red, odabire pijuna kojeg će micati i šalje ga natrag agentu ploče. U nastavku će se opisati detaljni proces pozivanja funkcija i metoda kako bi sustav ispravno igrao igru i nesmetano međusobno komunicirao.

U ovom radu će biti prikazane i objašnjene samo glavne metode, dok se sve pomoćne metode mogu pogledati u kodu priloženom uz rad.



Slika 2: Reprezentacija modela sustava (Izvor: izrada autora)

5.3.1. Arhitektura projekta

Sveukupna logika i programski kod projekta je raspoređen na ukupno 5 datoteka:

- run.py - služi jedino za pokretanje cijele aplikacije
- cli.py - upravlja komunikacijom između klijenta i aplikacije
- DATA.py - rječnik korisničkih imena i lozinki svih agenata
- game.py - glavna logika aplikacije. Sadrži klase agenata i njihovih ponašanja
- painter.py - služi za crtanje Ludo igrača ploče u konzolu

Kao što smo već i prije naveli, dodavanje broja agenata je proizvoljno, od 3 agenta (1 agent igre i 2 agenta igrača) do 5 agenata (1 agent igre i 4 agenta igrača). Pošto stolna igra Ludo nema puno pravila i pravila nisu komplicirana, ni agenti ni njihova ponašanja nisu

pretjerano kompleksna, štoviša, vrlo su jednostavna. U narednom bloku koda prikazane su klase agenata igrača `Player()` i igre `Game()` sa svim svojim funkcijama i ponašanjima. Struktura agenata SPADE klasa definira podklase koje služe kao njihova ponašanja koje SPADE također podržava. Od svih ponašanja koja SPADE podržava, u ovom projektu koristili smo samo cikličko ponašanje `CyclicBehaviour` (ponašanje koje se izvršava ciklično dok se ne zaustavi). Na kraju agenata se definira asinkrona metoda `setup` u kojoj se povezuju definirani predlošci poruka i ponašanja agenata. Predlošci (eng. `Template`) služe za filtriranje XAMPP poruka koje dolaze agentu.

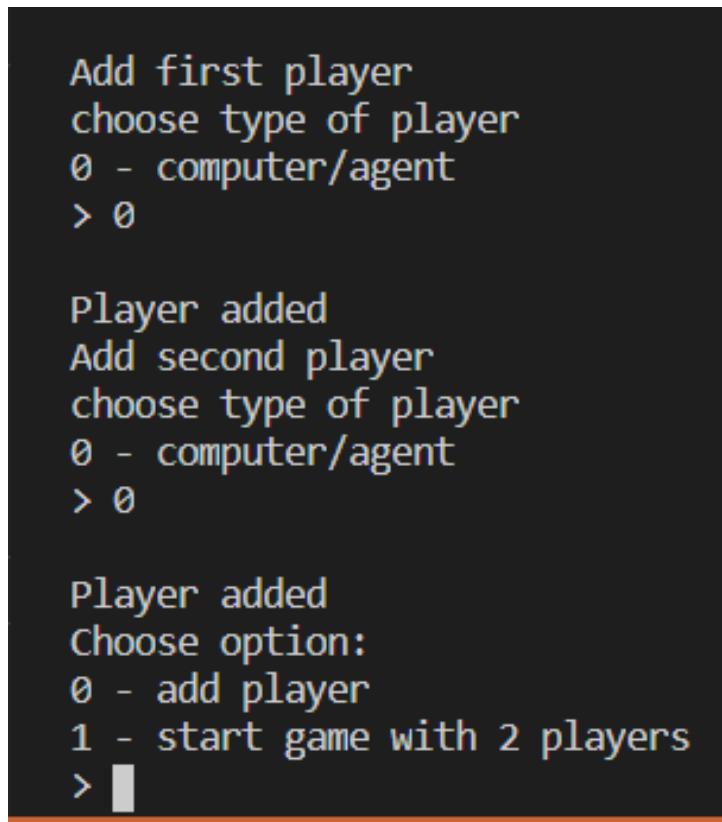
```
class Player(Agent):
    def __getitem__(self):
    def __str__(self):
    def choose_pawn(self, pawns):
    class WaitTurn(CyclicBehaviour)
    async def setup(self):
        print("Player_started")

        b = self.WaitTurn()
        template = Template()
        template.set_metadata("ludoMETA", "getIndex")
        self.add_behaviour(b, template)

class Game(Agent):
    def add_player(self, player):
    def get_available_colours(self):
    def _get_next_turn(self):
    def get_pawn_from_board_pool(self, player):
    def get_allowed_pawns_to_move(self, player, rolled_value):
    def get_board_pic(self):
    def _jog_foreign_pawn(self, pawn):
    def _make_move(self, player, pawn):
    def prompt_choose_pawn(self):
    def print_info_after_turn(self):
    def print_board(self):
    def start_players(self):
    def start_player_agent(self, agent):
    def play_turn(self):
    def contains_index(self, index):
    class InformPlayer(CyclicBehaviour):
    async def setup(self):
        print("Game_started")
        b = self.InformPlayer()
        template = Template()
        template.set_metadata("ludoMETA", "informPlayer")
        self.add_behaviour(b, template)
```

5.3.2. Inicijalizacija igre

Pri pokretanju aplikacije potrebno je dodati igrače u igru. Igrači se dodaju preko jednostavnog izbornika prikazanog na Slika 3. U izborniku je moguće odabrati dodavanje igrača kao računalo/agent, a nakon dva dodana igrača, što je minimum za igranje igre, otvara se i opcija za pokretanje igre s N igrača. Ukoliko želimo dodavati još igrača, to može učiniti odabirom opcije za dodavanje igrača kao računalo/agent. Nakon dodavanja četiri igrača, igra se automatski pokreće.



```
Add first player
choose type of player
0 - computer/agent
> 0

Player added
Add second player
choose type of player
0 - computer/agent
> 0

Player added
Choose option:
0 - add player
1 - start game with 2 players
> 
```

Slika 3: Izbornik za dodavanje novih igrača i/ili pokretanje igre (Izvor: izrada autora)

5.3.2.1. Dodaj igrače

Proces dodavanja igrača u igru se može vidjeti u dolje navedenim funkcijama. Za dodavanje prva dva igrača u igru se prikazuje samo opcija za dodavanje novih igrača. Odabirom opcije odabira novog igrača poziva se funkcija koja iz liste raspoloživih boja uzima jednu i kreira igrača agenta prema odabranoj boji. Pošto sam koristio Jabbim, registrirao sam sve agente, a njihove podatke uzimam iz datoteke DATA.py koja mi služi za pohranjivanje korisničkih imena i šifri svih agenata. Novo kreirani igrač se dodaje u listu igrača u klasi Game i postavlja na ploči.

```
def prompt_for_players(self):
    '''put all players in the game'''
    counts = ("first", "second", "third", "fourth_last")
    text_add = "Add_{}_player"
    for i in range(2):
        print(text_add.format(counts[i]))
```



```

        self.prompt_for_player()
        print("Player_added")

text = linesep.join(["Choose_option:",
                    "0_-_add_player",
                    "1_-_start_game_with_{}_players"])

for i in range(2, 4):
    choice = self.validate_input(text.format(str(i)), int, (0, 1))
    if choice == 1:
        break
    elif choice == 0:
        print(text_add.format(counts[i]))
        self.prompt_for_player()
        print("Player_added")

def prompt_for_player(self):
    ''' get player attributes from input,
    initial player instance and
    add player to the game
    '''
    available_colours = self.game.get_available_colours()
    text = linesep.join(["choose_type_of_player",
                        "0_-_computer/agent"])
    choice = self.validate_input(text, int, (0))

    if choice == 0:
        # automatically assign colours
        colour = available_colours.pop()
        ['yellow', 'blue', 'red', 'green']
        if(colour == 'yellow'):
            player = Player(colour, data.yellowPlayer, data.password)
        if(colour == 'blue'):
            player = Player(colour, data.bluePlayer, data.password)
        if(colour == 'red'):
            player = Player(colour, data.redPlayer, data.password)
        if(colour == 'green'):
            player = Player(colour, data.greenPlayer, data.password)
        self.game.add_player(player)

def add_player(self, player):
    self.players.append(player)
    for pawn in player.pawns:
        self.board.put_pawn_on_board_pool(pawn)

```

5.3.3. Započni igru

Kada su svi igrači dodani (Slika 4), pokreće se glavni agent igre. Pošto se u glavnom agentu igre vrti cikličko ponašanje (eng. *CyclicBehaviour*), potrebno je sve agente igrače dodati u ciklus i postaviti ih pod određenim imenom kako bismo im kasnije mogli pristupiti. Cikličko ponašanje se izvršava ciklično dok se ne zaustavi. Nakon dodavanja svih agenata u ciklus

```
Player added
Game start with 4 players:
(yellow)
(red)
(green)
(blue)
```

Slika 4: Igrači dodani u igru (Izvor: izrada autora)

pokrećemo agent igre. Sve dok je agent igre aktivan/živ i dok zastavica za kraj igre nije podignuta, agent igre igra potez. Ako je agent igre zaustavljen ili je igra gotova ispisuju se rezultati tj. poredak igrača, zaustavljaju se agenti igrači i agent igre. Igru je također moguće i prisilno zaustaviti pritiskom kombinacije tipki CTRL + C na tipkovnici.

```
def play_game(self):
    self.add_player_agents()
    self.start_game_agent()

    try:
        while self.game.is_alive() and not self.game.finished:
            time.sleep(1)
            self.game.play_turn()
        print("Game_finished")
        self.print_standing()
        self.stop_player_agents()
        self.game.stop()
    except (KeyboardInterrupt, EOFError):
        self.stop_player_agents()
        self.game.stop()
        print("Agents_finished")

    print(linesep + "Exiting_game._")
    raise

def start_game_agent(self):
    players = []
    for agent in self.playerAgents:
        agent.start()
        self.game.set(agent.name, agent)
        players.append(agent)
    self.game.set("players", cycle(players))

    future = self.game.start(auto_register=False)
    future.result()
```

Prikaz početne igrače ploče s maksimalnim brojem igrača može se vidjeti na Slika 5. Na slici možemo vidjeti da je pokrenut agent igre (Game running) i da je poslao poruku agentu crvenog igrača. Pošto je ovo tek prvi potez, lista pijuna na ploči je prazna, pa agent vraća

$$\begin{array}{|c|c|} \hline \# & \# \\ \hline \# & \# \\ \hline \end{array} \quad (\text{red})$$

```
# # # # #
# #         YELLOW        # |   |   # |   #
# #         #---#---#---# V          BLUE      #
# #         #   #   #   #           #           #
# #         #---#---#---#           #           #
# #         | Y1 | Y2 |             | B1 | B2 |     #
# #         #---#---#---#           #           #
# #         | Y3 | Y4 |             | B3 | B4 |     #
# #         #---#---#---#           #           #
# #         #   #   #   #           #           #
# #         #---#---#---#           #           #
# --> #   #   #   #           #           #
##### #---#---#---# #####
# |   |   |   |   |   |   |   #   #   |   |   |   |   |   |   #
# --- ##### --- ##### --- #####
# |   |   |   |   |   |   | X |   |   |   |   |   |   |   #
# --- ##### --- ##### --- #####
# |   |   |   |   |   |   |   #   #   |   |   |   |   |   |   #
##### #---#---#---# #####
# #   #   #   #                                     <-- #
# #         GREEN        #---#---#---#           RED      #
# #         #   #   #   #           #           #
# #         #---#---#---#           #           #
# #         #---#---#---#           #           #
# #         | G1 | G2 |             | R1 | R2 |     #
# #         #---#---#---#           #           #
# #         | G3 | G4 |             | R3 | R4 |     #
# #         #---#---#---#           #           #
# #         #   #   #   #           #           #
# ^ #---#---#---#           #           #
# | #   |   |   #           #           #
#####
```

Slika 5: Čovječe ne ljuti se igrača ploča ispisana u konzoli (Izvor: izrada autora)

odgovor 0. Agent igrača vraća index pijuna kojeg želi pomaknuti, a indexi se kreću od 1 do 4. Pošto je poslao index 0, to znači da ne miče niti jednog pijuna. Iscrtava se kocka s licem koje korespondira s bačenim brojem, a u zagradi se ispisuje naziv/boja igrača koji trenutno igra. Iscrtava se i cijela igračka ploča sa svim pijunima pozicioniranim na svojim početnim mjestima.

5.3.3.1. Igraj potez

U prošlom smo bloku koda vidjeli da sve dok je agent igre aktivan/živ i dok zastavica za kraj igre nije podignuta, agent igre igra potez. U ovom ćemo poglavlju pobliže opisati igranje poteza. U početku inicijaliziramo listu `jog_pawns` u koju ćemo pohraniti pijune protivnika koje bi mogli gurnuti s ploče (kod nas je popularan izraz za takvu akciju "pojesti" protivničkog pijuna). Kako bi dobili boju igrača koji je trenutno na redu, pozivamo `_get_next_turn()` koja rotira listu igrača i prosljeđuje nam prvog igrača iz liste. Ako se ova funkcija pozove van klase, poredak će se prekinuti i pomiješati. Sada znamo koji je sljedeći igrač na redu i za njega se baca kocka. Bacanje kocke se moglo raditi i u agentu igrača, ali na kraju sam odlučio da će se kocka bacati na agentu igre. Agent igre uzima sve pijune koje igrač ima na ploči i koje može micati s trenutno bačenim brojem kocke koji označava broj pomicanja na ploči. Kada je agent sakupio sve te informacije poziva se ponašanje `InformPlayer()`.

```
Player waiting for turn
Game running
blue player received message with content: [[Pawn(index=1, colour='blue', id='B1')], 2]
Ludo Board received message with content: 0

      -----
      |      |
      |  #  #  |
      |      |
      -----
      (blue)

B1 possible pawns to move. B1 is moved.
```

Slika 6: Igrač igra potez (Izvor: izrada autora)

```
def play_turn(self):

    self.jog_pawns = []
    self.curr_player = self._get_next_turn()
    self.rolled_value = Die.throw()
    self.allowed_pawns = self.get_allowed_pawns_to_move(
        self.curr_player, self.rolled_value)

    self.InformPlayer()

def _get_next_turn(self):

    if not self.rolled_value == Die.MAX:
        self.players.rotate(-1)
    return self.players[0]
```

5.3.3.2. Informiraj igrača

`InformPlayer(CyclicBehaviour)` je cikličko ponašanje agenta igre `Game(Agent)` koje se ciklički izvršava i informira agente igrača da je trenutno njihov red da igraju. Od agenta igrača na kojemu je trenutno red da igra, uzima se jid u kojem je pohranjeno njegovo korisničko ime i server preko kojeg se komunicira tj. šalju poruke. U ovo slučaju to je Jabbim server. Sljedeće što

radimo je instanciramo poruku i definiramo primatelja poruke(trenutnog agenta igrača). Postavljamo metapodatke prema FIPA standardu kako bismo znali kojem se ponašanju ti metapodaci šalju. Nakon toga dodajemo sadržaj poruke. Pošto želimo poslati listu, te podatke je prvo potrebno prebaciti u JSON format. Tu nam jsonpickle biblioteka jako olakšava posao sa šifriranjem i dešifriranjem JSON podataka. Poruka je poslana agentu igrača, čekamo njegov odgovor, a kada ga primimo, postupamo u skladu s primljenim podacima.

```
class InformPlayer(CyclicBehaviour):
    async def run(self):
        print("Game_running")
        if(self.agent.curr_player != None):
            currentPlayer = self.agent.curr_player.jid
            reciever = currentPlayer[0]+"@"+currentPlayer[1]

            msg = Message(to=reciever)
            msg.set_metadata("ludoMETA", "getIndex")
            msg.body = jsonpickle.encode(
                [self.agent.allowed_pawns, self.agent.rolled_value])
            await self.send(msg)

            # wait for a message for 10 seconds
            msg = await self.receive(timeout=10)
            #player = self.get("players")
            if msg:
                print("Ludo_Board_received_message_with_content:{}".format(
                    jsonpickle.decode(msg.body)))

                if self.agent.allowed_pawns != []:
                    self.agent.index = int(msg.body)

                    self.agent.picked_pawn = self.agent.choose_pawn(
                        self.agent.index)
                    self.agent._make_move(
                        self.agent.curr_player, self.agent.picked_pawn)
                else:
                    self.agent.index = -1
                    self.agent.picked_pawn = None

                self.agent.print_info_after_turn()
                self.agent.print_board()
            else:
                print(
                    "Did_not_received_any_message_after_10_seconds_Get_Pawn_Index")

            # stop agent from behaviour
            await asyncio.sleep(1)

    async def on_end(self):
        await self.agent.stop()
```

5.3.3.3. Čekaj svoj red

WaitTurn(CyclicBehaviour) je cikličko ponašanje agenta igrača Player(Agent) koje se ciklički izvršava i čeka poruku od agenta igre da je trenutno njegov red da odigra potez. Kada agentu igraču stigne poruka potrebno ju je dešifrirati. Tu opet u igru uskače jsonpickle, koji jednom naredbom dešifrira poruku. U poruci se prima lista pijuna koji se nalaze na igračkoj ploči i rezultat bacanja kocke tj. broj pomaka koji igrač može napraviti u trenutnom potezu. Ukoliko je lista pijuna u igri prazna, odabiremo jedan od četiri moguća pijuna. Ako je lista puna, odabiremo jedan od tih pijuna za pomak u potezu. Kada smo odabrali kojeg ćemo pijuna pomicati, šaljemo njegov indeks natrag agentu igre. Moramo instancirati poruku i definirati primatelja poruke(agenta igre). Postavljamo metapodatke prema FIPA standardu kako bismo znali kojem se ponašanju ti metapodaci šalju. Nakon toga dodajemo sadržaj poruke kojeg šifriramo pomoću jsonpickle.

```
class WaitTurn(CyclicBehaviour):
    async def run(self):
        print("Player_waiting_for_turn")
        allowedPawns = []
        dice = 1
        # wait for a message for 10 seconds
        msg = await self.receive(timeout=10)
        if msg:
            print(self.agent.colour + "_player_received_message_with_content:_" +
                ".format(
                    jsonpickle.decode(msg.body))
                allowedPawns = jsonpickle.decode(msg.body)[0]
                dice = jsonpickle.decode(msg.body)[1]
            else:
                print("Did_not_received_any_message_after_10_seconds_Waiting_turn...")

        if(allowedPawns != []):
            chosenPawn = self.agent.choose_pawn(allowedPawns, dice)
        else:
            chosenPawn = self.agent.choose_pawn(self.agent.pawns, dice)

        msg = Message(to=data.ludoBoard)
        msg.set_metadata("ludoMETA", "informPlayer")
        msg.body = jsonpickle.encode(chosenPawn)

        await self.send(msg)

        # stop agent from behaviour
        await asyncio.sleep(1)

    async def on_end(self):
        await self.agent.stop()
```

5.3.3.4. Pomakni odabrani pijun

Ovdje opet promatramo ponašanje InformPlayer(CyclicBehaviour) čiji je kod prikazan u sekciji Informiraj igrača. Kada je igrač poslao poruku sa indeksom pijuna kojeg želi micati i kada smo primili njegovu poruku, tu je poruku potrebno dešifrirati. Poruku dešifriramo pomoću json-pickle naredbe decode koja nam vraća čitljivi oblik poruke. Primljeni indeks uspoređujemo s s indeksom pijuna na ploči od trenutnog igrača i odabiremo taj pijun. Podatke o trenutnom agentu i odabranom pijunu za pomak prosliješujemo funkciji self.agent._make_move(self.agent.curr_player, self.agent).

```
#class Game (Agent)
    def _make_move(self, player, pawn):
        if self.rolled_value == Die.MAX and\
            self.board.is_pawn_on_board_pool(pawn):
            self.board.put_pawn_on_starting_square(pawn)
            self._jog_foreign_pawn(pawn)
            return
        self.board.move_pawn(pawn, self.rolled_value)
        if self.board.does_pawn_reach_end(pawn):
            player.pawns.remove(pawn)
            if not player.pawns:
                self.standing.append(player)
                self.players.remove(player)
                if len(self.players) == 1:
                    self.standing.extend(self.players)
                    self.finished = True
            else:
                self._jog_foreign_pawn(pawn)

#class Board()
def move_pawn(self, pawn, rolled_value):
    common_poss, private_poss = self.pawns_possiotion[pawn]
    end = self.COLOUR_END[pawn.colour.lower()]
    if private_poss > 0:
        # pawn is already reached own final squares
        private_poss += rolled_value
    elif common_poss <= end and common_poss + rolled_value > end:
        # pawn is entering in own squares
        private_poss += rolled_value - (end - common_poss)
        common_poss = end
    else:
        # pawn will be still in common square
        common_poss += rolled_value
        if common_poss > self.BOARD_SIZE:
            common_poss = common_poss - self.BOARD_SIZE
    position = common_poss, private_poss
    self.set_pawn(pawn, position)

def set_pawn(self, pawn, position):
    self.pawns_possiotion[pawn] = position
```

Funkcija make_move() javlja ploči(klasa Board()) da pomakne pijun. Ukoliko je bačena

```
#####
Game finished
Standing:
1 - (red)
2 - (yellow)
```

Nakon svakog poteza ispisuju se informacije o potezu (popis mogućih pijuna za pomak, odabrani pijun za pomak, broj micanja pijuna, da li je pijun bacio suparničkog pijuna s igrače ploče) što se može vidjeti na Slika 8, i crta se ploča s novim pozicijama pijuna.

Slika 8: Informacije o potezu (Izvor: izrada autora)

[illegible]


```
[ '#', 'L', 'U', 'V', 'Y', 'E', 'L', 'O', 'W', ' ', '-', '-', '-', '-', '-', '#', '-', '-', '-', '-', '-', '#', 'V', 'U', 'E', ' ', '#'],  
... ]
```

Prilikom svake nove instance ove klase kreira se dubinska kopija (eng. deepcopy) predloška ploče. Dubinsko kopiranje je kopiranje podataka pri čemu se postojeće reference miču sa trenutnih objekata i time osiguravaju da ne postoji neželjena referenca koja bi mogla uzrokovati nepoželjno ponašanje koda. Kao argument prima pozicije pijuna koje postavlja na te definirane pozicije u matrici.

```

        row, column = CODE_COLOUR_SQUARES[colour][private_poss]
    elif common_poss == 0:
        # pool
        row, column = CODE_POOL_PLACES[colour][pawn.index]
        offset = 0 # we do not need from offset in the pool only in squares
    else:
        # common squares
        row, column = CODE_COMMON_SQUARES[common_poss]
    if offset > 0:
        self.board_tmpl_curr[row - 1][column + offset] = pawn.id[1]
    else:
        self.board_tmpl_curr[row - 1][column - 1] = pawn.id[0]
        self.board_tmpl_curr[row - 1][column] = pawn.id[1]

def _place_pawns(self, position_pawns):
    for position, pawns in position_pawns.items():
        for index, pawn in enumerate(pawns):
            self._place_pawn(pawn, position, index)

```

6. Zaključak

Glavna misao vodilja u ovom radu bila je prikaz izrade igre "Čovječe ne ljuti se" uz pomoć proaktivnih agenata. Aplikacije, tj. višeagentni sustav je kreiran u Python programskom jeziku. Agenti su kreirani pomoću SPADE biblioteke koja omogućuje izradu i komunikaciju između pametnih agenata. Komunikacija između agenata se izvršava XAMPP protokolom, a u ovom radu se ta komunikacija odvijala preko Jabber servera koji podržava Jabber i XAMPP servise.

Za ostvarenje autonomnog sustava koji samostalno igra društvenu igru "Čovječe ne ljuti se" kreirane su dvije vrste agenata. Jedan tip agenta je namijenjen za upravljanje cijelog sustava, odnosno igre, dok je drugi tip agenta namijenjen za slušanje i primanje podataka od prvog tipa agenta i reagiranja na iste te slanja povratnih informacija. Zbog jednostavnosti igre "Čovječe ne ljuti se" agenti nisu komplicirani, štoviše dosta su jednostavni i postižu dosta nisku razinu inteligencije koja bi se lako mogla podići uključivanjem određene doze umjetne inteligencije (eng. Artificial Intelligence) u sustav. Najveći problem kod izrade ovog sustava je bilo manjak primjera aplikacija ostvarenih putem SPADE agenata. Jedini izvor informacija bila je SPADE dokumentacija, koju moram pohvaliti na detaljnosti i opširnom opisu određenih elemenata i ponašanja agenata. Pošto sam u izradi ovog sustava koristio samo ponašanja tipa `CyclicBehaviour` ne mogu govoriti u ime svih ponašanja, ali cikličko ponašanje mi se iznimno svidjelo zbog svoje jednostavnosti. Zadaje mu se što se treba inicijalizirati kod pokretanja agenta, koji dio koda mora izvršavati prilikom svakog novog ciklusa i što se događa kada se agent zatvori. Stvarno jednostavno. Iako ovaj sustav funkcionira poprilično dobro i točno, želio bih se u budućnosti vratiti na njega i pokušati implementirati neku od tehnika strojnog učenja kako bi se povećala razina inteligencije agenata. Rad s agentima se u početku činio dosta apstraktan, ali nakon malo istraživanja i dosta pokušaja i pogrešaka, dosta je intuitivan i uvelike olakšava rad s većim brojem objekata koji zahtijevaju isti set ponašanja i logiku.

Radom na ovom projektu poboljšao sam svoje znanje i tehniku pisanja u Python programskom jeziku, i upoznao sam se s novim bibliotekama SPADE i jsonpickle. Biblioteka jsonpickle mi je bila otkriće i siguran sam da ću je koristiti i u budućim projektima zbog svoje jednostavnosti. Iako je ovaj rad bio vrlo jednostavna implementacija višeagentnih sustava, otvorio horizonte o beskonačnim mogućnostima koje pružaju višeagentni sustavi i kako višeagentni sustavi nisu nešto strašno i apstraktno, samo je potrebno malo sjesti i proučiti način na koji funkcioniraju. U ovaj projekt sam uložio poprilično puno vremena i truda i znam da rezultat nije spektakularan, ali ja sam iznimno zadovoljan novo stečenim znanjima koja ću moći prenijeti na buduće projekte.

Popis literature

- [1] D. Yves, D. Frank, R. Juan i B. Javier, „Advances in Practical Applications of Agents and Multiagent Systems”, str. 165–167, adresa: <https://books.google.hr/books?id=5BBjs-Dm5PUC&pg=P..#v=onepage&q&f=false> (pogledano 23. 8. 2020).
- [2] M. Schatten, „Višeagentni sustavi Inteligentni agenti”, 2020.
- [3] A. Dorri, R. Jurdak i S. Kanhere, „Multi-Agent Systems: A survey”, adresa: https://www.researchgate.net/publication/324847369_Multi-Agent_Systems_A_survey (pogledano 23. 8. 2020).
- [4] J. Stefan J., „On using Multi-agent Systems in Playing Board Games”, adresa: <https://www.researchgate.net/publication/221455376> (pogledano 23. 8. 2020).
- [5] „The SPADE agent model — SPADE 3.1.6 documentation”, adresa: <https://spade-mas.readthedocs.io/en/latest/model.html> (pogledano 23. 8. 2020).
- [6] „Rules and instructions for the games of Ludo”, adresa: <https://www.mastersofgames.com/rules/ludo-rules-instructions-guide.htm> (pogledano 23. 8. 2020).

Popis slika

1.	Ploča igre Ludo (Izvor: http://pachisi.vegard2.net/ludo_big.gif , 2020)	8
2.	Reprezentacija modela sustava (Izvor: izrada autora)	9
3.	Izbornik za dodavanje novih igrača i/ili pokretanje igre (Izvor: izrada autora) . . .	11
4.	Igrači dodani u igru (Izvor: izrada autora)	13
5.	Čovječe ne ljuti se igrača ploča ispisana u konzoli (Izvor: izrada autora)	14
6.	Igrač igra potez (Izvor: izrada autora)	15
7.	Kraj igre sa poretком igrača (Izvor: izrada autora)	19
8.	Informacije o potezu (Izvor: izrada autora)	19