

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

Elena Kržina

SIMULACIJA RPG BORBE PUTEM AGENATA

PROJEKT

VIŠEAGENTNI SUSTAVI

Varaždin, 2024.

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

Elena Kržina

Matični broj: 0016135585

Studij: Baze podataka i baze znanja

SIMULACIJA RPG BORBE PUTEM AGENATA

PROJEKT

Mentor:

dr. sc. Bogdan Okreša Đurić

Varaždin, siječanj 2024.

Elena Kržina

Izjava o izvornosti

Izjavljujem da je ovaj 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.

Autorica potvrdila prihvaćanjem odredbi u sustavu FOI Radovi

Sažetak

Projekt se bavi izradom višeagentnog sustava koji simulira RPG borbu junaka i protivnika uz uporabu Python SPADE agenata koji međusobno komuniciraju porukama te putem baze znanja implementirane u SWI Prologu uče slabosti protivika. Program je u cjelosti implementiran u Pythonu uz dodatak Prologa te dobiva podatke iz JSON datoteke. Služi za razumijevanje načina na koji su implementirani agenti u igrama te za pregled moguće implementacije njihove interakcije.

Ključne riječi: Python SPADE; agenti; višeagentni sustavi; RPG borba; simulacije borbe; RPG igra;

Sadržaj

1. Uvod	1
2. Metode i tehnike rada	2
3. Višeagentni sustavi u video igrama	3
3.1. Koncept agenta i Python SPADE	3
3.2. Razvoj agenata u video igrama	3
4. Implementacija višeagentnog sustava	5
4.1. Agenti u igri	5
4.1.1. JSON podaci mogućnosti agenata	5
4.1.2. Klase Enemy i Hero	7
4.1.3. Baze znanja agenata	11
4.1.4. Implementacija agenata	11
4.2. Izgradnja okruženja agenata	17
5. Prikaz rada aplikacije	18
6. Kritički osvrt	23
7. Zaključak	24
Popis literature	25
Popis slika	26
Popis isječaka koda	27

1. Uvod

Višeagentni je sustav koncept umjetne inteligencije koji se sastoji od autonomnih entiteta smještenih u neko okruženje unutar kojeg primjećuju parametre okoline na temelju kojih mogu donositi odluke za ostvarivanje njima postavljenih ciljeva [1]. Domena koju projekt obrađuje je višeagentni sustav koji simulira RPG borbu dvaju grupa igrača, protivnika i heroja. Uz agente, protivnici i heroji biti će u konstantnoj interakciji te će uz bazu znanja učiti o slabostima druge grupe, optimizirajući tako proces napadanja.

Tema je odabrana zbog vlastitog interesa u video igre, načina na koji umjetna inteligencija može utjecati na video igre i zbog želje testiranja što agenti mogu i kako ih učinkovito postaviti da mogu sami upravljati cijelim izvođenjem programa uz potpunu autonomiju radnje, učenja i slučajnih događaja.

Rad je podijeljen na 7 dijela, od kojih prvi pruža uvod u temu višeagentnog sustava koji simulira RPG borbu. Sljedeće poglavlje bavi se opisom metoda i tehnika rada te se objašnjava koncept višeagentnih sustava u video igrama uz teorijski uvod u agente i SPADE te iznošenje važnosti i mogućnosti rada agenata na igračim sustavima. Slijedi implementacija višeagentnog sustava koji se bavi opisom agenata u igri te izgradnjom okruženja unutar kojih će biti agenti. Dalje se prikazuje aplikacija i iznosi se kritički osvrt na projekt. Na kraju se nalazi poglavlje "Zaključak" u kojem se rezimiraju najbitniji dijelovi teme te je popisana literatura, popisane su slike i isječki kodova.

2. Metode i tehnike rada

Za izradu rada, proučava se stručna literatura koja uključuje knjige, znanstvene članke te pouzdane web stranice uz navođenje primjera te uz vlastitu argumentaciju. Za potrebe implementacije programa, korištene su službene dokumentacije programskih jezika i alata.

Projekt je u cjelosti napravljen u Python programskim jeziku, pri čemu su korišteni dodatni moduli "prosody" kao XMPP poslužitelj, "Pillow" za rad sa slikama, "pygame" za uredan prikaz slika i statistika, "spade" za implementaciju i rad s agentima i "pyswip" za rad s povezanom bazom znanja. Baza znanja je pritom implementirana u SWI Prologu.

Za vizualni prikaz likova igre vlastoručno su nacrtani spriteovi koji se koriste u programu. Kako bi se postigao pikselasti oblik samih likova korišten je online alat Pixel It koji se može pronaći na sljedećoj stranici: <https://giventofly.github.io/pixelit/>.

3. Višeagentni sustavi u video igrama

U ovom se poglavlju obrađuju temeljni pojmovi višeagentnih sustava, agenata i SPADE tehnologije te se ističe važnost i načini implementacije višeagentnih sustava unutar današnjih video igara.

3.1. Koncept agenta i Python SPADE

Koncept višeagentnih sustava je koncept koji je nastao iz umjetne inteligencije. Višeagentni sustav sastoji se od agenata koji se definiraju kao autonomni entiteti koji pokazuju inteligentno ponašanje [2].

Pojam agent širok je pojam koji se može shvatiti na brojne načine, zbog čega je teško odrediti što jest, a što nije agent. Generalna je definicija agenta entitet koji je smješten u neko okruženje gdje primjećuje različite parametre koji se koriste za donošenje odluka koje potpomažu ispunjavanju agentova zadanog cilja [1].

Višeagentni su sustavi jedna od 3 kategorija distribuirane umjetne inteligencije [1].

Posebnost je višeagentnih sustava da kolaborativno rješavaju zadatke uz fleksibilnost učenja i autonomnog donošenja odluke [1]. Uobičajeno ima barem sljedeće značajke, mada brojni autori spominju i druge [3]:

- autonomija, odnosno operiranje bez potrebe za intervencijom
- socijalna mogućnost, odnosno sposobnost interakcije s drugim agentima
- reaktivnost, odnosno sposobnost interakcije s okolinom i reakcija na njezino stanje
- proaktivnost, odnosno sposobnost samodjelovanja i težnja za postizanjem ciljeva

Agenti koji se kreiraju u ovom projektu implementirani su u Python programskom jeziku putem SPADE (Smart Python multi-Agent Development Environment) platforme, koja je Python platforma za višeagentne sustave bazirana na trenutnom slanju poruka koje omogućuje XMPP, otvoreni standarda za razmjenu poruka [4].

Uz SPADE, agentima je pridruženo određeno ponašanje, moguće je asinkrono čekati na radnje agenata te je agentima omogućena okolina unutar koje mogu međusobno slati poruke kojima će se mijenjati stanje okoline te koje će mijenjati stanje drugih agenata.

3.2. Razvoj agenata u video igrama

Postoje brojna područja koja imaju benefit od razvijanja višeagentnih sustava. Neki od istaknutih su tehnologije u oblaku, sigurnost, routing, robotika, gradovi i gradska okruženja i drugi [1]. Ipak, ova se tema pojavljuje i u video igrama, koje već dugi niz godina koriste

umjetnu inteligenciju u njihovoj izvedbi. Dobar primjer izvedbe je primjerice CPU borac u borbenim igrama, ali i napadači u igrama na poteze koji odlučuju koji potez napraviti kako bi dobili optimalni rezultat.

Igrača se industrija rapidno razvija. Prošla je velik broj različitih stajališta i tehnologija te se u zadnjih nekoliko godina fokus igara odvaja od renderiranja slika i realističnosti i češće se postavlja na ponašanje likova i je li ono autentično, odnosno prirodno [5].

Domena unutar koje se nalazi implementirani projekt su video igre, RPG igre te njihova simulacija, odnosno projekt se odnosi na razonodu i zabavu. Istražuju se pitanja poboljšanja i problematike video igara i na što i kako bi agenti mogli utjecati prilikom njezinog razvoja.

Problemi u video igrama koji se mogu rješavati višeagentnim sustavom se primjerice pojavljuju u efektivnom sustavu odlučivanja u umjetnoj inteligenciji. Odluke agenata odnose se na diskretno razumijevanje okoline unutar kojih agenti djeluju te autonomno izvršavanje akcija na temelju prikupljenih parametara [6]. Još jedna primjena višeagentnih sustava je u koordinaciji ponašanja neigračkih likova i kreiranja stabla ponašanja za njih kako bi se reprezentiralo ponašanje likova odnosno AI agenata unutar svijeta igre [7]. Nadalje, ističe se potreba za implementiranjem kognitivno inteligentnih neigračkih likova koje ističe njihovu autonomiju, interakciju s drugim likovima, utjecanje na ciljeve i slično [5].

Neki od benefita razvijanja višeagentnog sustava za igračku industriju su tako autentičnost likova, odlučivanje temeljeno na okruženju unutar kojeg se agenti nalaze, biranje optimalnih strategija, praćenje agentova ponašanja i slično. Agenti također mogu pridonijeti poboljšanju video igara tako da se poveća stabilnost sustava tako da se agent koji doživi pogrešku makne iz sustava i zamijeni drugim agentom [6].

Agenti unutar projekta za cilj imaju čitati okolinu unutar koje se nalaze uz preispitivanje drugih agenata tako da biraju bolje strategije prilikom RPG borbe koja se temelji na potezima. Također se ispituje mogućnost suradnje agenata i skalabilnost sustava u kojem se agenti moraju kretati redom, birajući napade na temelju pronalaska slabosti, jačine i imunosti protivničke partije. Budući da se ni jedna partija ne igra i agenti preuzimaju kompletnu autonomiju nad sustavom, projekt simulira RPG video igru.

4. Implementacija višeagentnog sustava

U ovom se poglavlju opisuje način implementacije samog višeagentnog sustava simulacije RPG borbe. Isprva se objašnjava način izrade i provođenja agenata uz prikaz potrebnih klasa i datoteka. Nakon toga opisuje se izgranja samog okruženja unutar kojeg se nalaze agenti, te se na kraju prikazuje implementacije logike igre, odnosno povezivanje agenata međusobno i s okruženjem, iznošenje njihovih mogućnosti i opisivanje uvjeta igre.

Program je implementiran tako da se može koristiti na Windows i Linux operacijskim sustavima.

4.1. Agenti u igri

U ovom dijelu opisuje se implementacija agenata u aplikaciji. Započinje s opisom JSON podataka koji se unose kao mogućnosti agenata. Zatim se opisuju klase protivnika i igrača, agenti koji se stvaraju na temelju tih klasa te baza znanja u koju se unose podaci koje agenti nauče.

4.1.1. JSON podaci mogućnosti agenata

Podaci putem kojih se kreiraju aktori i kasnije agenti su definirani u JSON datoteci zvanoj 'stats.json'. Unutar nje, definirani su mogući oblici pojavljivanja agenata te koje statistike sadrže. Na primjeru prikazanom u nastavku vidi se agent 'GHOST'.

Svakom je agentu definiran ime ("name"), u ovom slučaju "ghostie", slika koja mu je pridružena te oblici napada koji se mogu pojaviti kod agenta.

U ovom slučaju, to su "touch", "look sad" i "drown". Za svaki napad, definirano je ime napada, jačina napada odnosno oštećenje drugog agenta, meta, element napada i tip napada. Bitno je za napomenuti da postoje i napadi liječenja koji ne napadaju protivničke agente, već daju zdravlje savezničkim agentu.

Dalje, agentu se postavljaju slabosti, jačine i imunost na određeneoblike napada. Tako je "ghostie" primjerice imun na bilo koji napad koji je tipa "melee".

Na kraju, pridodaju mu se i atributi. Svaki agent igrač odnosno protivnik ima "HP", koji se odnosi na količinu oštećenja koje agent može primiti prije njegova poraza, "ATK" koja je jačina agentovih fizičkih napada vezana uz tip ("type") napada, "DEF" koji je vezan uz fizičku obranu od napada, "SPATK" za specijalne napade i "SPDEF" za specijalnu obranu.

Dok protivnici imaju fiksno postavljene vrijednosti za ove attribute, junaci imaju raspon vrijednosti iz koje se u programu odabire slučajna vrijednost atributa. Na slici dolje prikazan je isječak koda iscjelitelja te način na koji su kod njega podaci distribuirani. Također je bitno napomenuti da junaci imaju samo slabost, bez imunosti i jačine.

```

1  "GHOST": {
2      "name": "ghostie",
3      "pct": "../img/ghost.png",
4      "attacks": [
5          {
6              "name": "touch",
7              "dmg": 2,
8              "target": "single",
9              "element": "melee",
10             "type": "atk"
11          },
12          {
13              "name": "look sad",
14              "dmg": 0,
15              "target": "single",
16              "element": "poison",
17              "type": "status"
18          },
19          {
20              "name": "drown",
21              "dmg": 8,
22              "target": "everyone",
23              "element": "ice",
24              "type": "spatk"
25          }
26      ],
27      "weakness": [ "none" ],
28      "strength": [ "none" ],
29      "immune": [ "melee" ],
30      "attributes": {
31          "HP": 10,
32          "ATK": 1,
33          "DEF": 100,
34          "SPATK": 1,
35          "SPDEF": 0
36      }
37  }

```

Isječak koda 1: Struktura agentovih podataka (Izvor: vlastiti uradak)

```

1  "hp_pool": [ 6, 7, 8, 9, 10, 11, 12 ],
2  "atk_pool": [ 1, 2 ],
3  "def_pool": [ 1, 2 ],
4  "dmg_pool": [ 3, 4, 5, 6 ],
5  "spatk_pool": [ 5, 6, 7, 8, 9, 10, 11, 12, 13, 14 ],
6  "spdef_pool": [ 6, 7, 8, 9, 10, 11, 12, 13, 14 ],
7  "weakness": "poison"

```

Isječak koda 2: Struktura podataka junaka (Izvor: vlastiti uradak)

```
1 def __init__(self, name, hp, atk, pdef, spatk, spdef, attack_list, weakness, pct):
2     self.name = name
3     self.hp = hp
4     self.atk = atk
5     self.pdef = pdef
6     self.spatk = spatk
7     self.spdef = spdef
8     self.attack_list = attack_list
9     self.weakness = weakness
10    self.pct = pct
```

Isječak koda 3: Konstruktor apstraktne klase "AbstractActor" (Izvor: vlastiti uradak)

4.1.2. Klase Enemy i Hero

Klasa protivnika ("Enemy") i klasa igrača ("Hero") u programu se koriste kao nadklase, odnosno klase iz kojih će agenti primiti potrebne podatke, implementirane kako bi se smanjila složenost implementacije agenta.

Objekti klase naslijeđuju iz klase "AbstractActor" koja definira zajedničke podatke, dok "Enemy" i "Hero" klase preuzimaju predefinirane elemente te dodaju razlikovne prilikom izvođenja programa. Sama klasa naslijeđuje klasu ABC za naslijeđivanje u Pythonu.

Na isječku koda prikazana je struktura konstruktora i svi podaci koji ulaze u kreiranje agenta.

Ova klasa dalje sadrži sljedeće metode:

- `add_initiative` - dodaje inicijativu agenta birajući nasumični broj od 1 do 20
- `show_picture` - prikazuje sliku agenta
- `print_stats` - ispisuje podatke o agentu na ekran; korištena za provjere i testiranje
- `get_stats_string` - dobiva podatke o agentu koji se koriste za ispis sadržaja o agentu
- `show_description` - prikazuje ispis podataka o agentu putem modula `pygame`

Ove funkcije služe ponajviše za prikaz samih agenata, dok inicijativa služi za postavljanje reda u kojim će se agenti kretati, odnosno kad će doći na red s napadom. Prilikom postavljanja igre, poziva se prikaz slike i popis atributa agenta koji će sudjelovati u igri.

Prikaz slike omogućuje se kodom kako je prikazano u nastavku. Na sličan način prikazuje se i tekst za statistiku agenta.

Kako bi se ovo postiglo, odabiru se nasumični igrači i protivnici, pri čemu je program trenutno testiran s maksimalno 4 igrača i 4 protivnika, poštujući česti standard partije RPG-jeva od 4 junaka.

Otvora se JSON datoteka, odabiru se nasumični entiteti prema kategoriji "ENEMY" ili "HERO" te se vraćaju nasumični atributi za agenta. U nastavku je prikazan kod za hvatanje i postavljanje nasumičnog junaka.

```

1  import pygame

2  @abstractmethod
3  def show_picture(self):
4      if self.pct:
5          try:
6              img_path = os.path.join(os.path.dirname(__file__), self.pct)
7              img = Image.open(img_path)
8              img = img.convert("RGBA")

9              target_size = (450, 420)
10             img = img.resize(target_size)

11             img_data = img.tobytes()

12             pygame.init()
13             screen = pygame.display.set_mode(target_size, pygame.HWSURFACE |
14             ↪ pygame.DOUBLEBUF)
15             pygame.display.set_caption("Agent Picture")

16             img_surface = pygame.image.fromstring(img_data, target_size, "RGBA")

17             font = pygame.font.Font(None, 24)
18             text_name = font.render(f"Name: {self.name}", True, (0, 0, 0))
19             text_name_rect = text_name.get_rect(center=(target_size[0] // 2,
20             ↪ target_size[1] - 20)) # Adjust Y-coordinate

21             clock = pygame.time.Clock()

22             running = True
23             while running:
24                 for event in pygame.event.get():
25                     if event.type == pygame.QUIT:
26                         running = False

27                 screen.fill((255, 255, 255))
28                 screen.blit(img_surface, (0, 0))
29                 screen.blit(text_name, text_name_rect)

30                 pygame.display.flip()
31                 clock.tick(0)

32             pygame.quit()


33     except Exception as e:
34         print(f"Error displaying picture: {e}")

```

Isječak koda 4: Kod za prikazivanje slike (Izvor: vlastiti uradak)



Slika 1: Prikaz slike igrača (Izvor: vlastiti uradak)



```
HP: 9
ATK: 3
DEF: 2
SPATK: 6
SPDEF: 3
Weakness: melee
Attacks:
  ember
  icy wind
  thunderpunch
  snowball
```

Slika 2: Prikaz statistike igrača (Izvor: vlastiti uradak)

```
1 def fetch_random_hero():
2     with open('desc/stats.json') as json_file:
3         data = json.load(json_file)
4
5     hero_names = list(data['heroes'].keys())
6     random_hero_name = random.choice(hero_names)
7
8     random_hero_data = data['heroes'][random_hero_name]
9
10    hero_hp = random.choice(random_hero_data.get('hp_pool', []))
11    hero_atk = random.choice(random_hero_data.get('atk_pool', []))
12    hero_def = random.choice(random_hero_data.get('def_pool', []))
13    hero_spatk = random.choice(random_hero_data.get('spatk_pool', []))
14    hero_spdef = random.choice(random_hero_data.get('spdef_pool', []))
15    hero_weakness = random_hero_data.get('weakness')
16
17    hero_attacks = random.sample(random_hero_data.get('attacks', []), 4)
18
19    random_hero_data['attributes'] = {
20        'HP': hero_hp,
21        'ATK': hero_atk,
22        'DEF': hero_def,
23        'SPATK': hero_spatk,
24        'SPDEF': hero_spdef
25    }
26    random_hero_data['weakness'] = hero_weakness
27    random_hero_data['attacks'] = hero_attacks
28
29    return fetch_hero_entity(random_hero_data)
```

Isječak koda 5: Kod postavljanja junaka (Izvor: vlastiti uradak)

```
1  % ally_knowledge_base.pl
2  :- dynamic has_weakness/2.
3  :- discontinuous has_weakness/2.
4
5  :- dynamic has_strength/2.
6  :- discontinuous has_strength/2.
7
8  :- dynamic is_immune/2.
9  :- discontinuous is_immune/2.
10
11 has_strength(rock-crab, melee).
12 has_strength(ice-crab, melee).
13 is_immune(doom-shroom, poison).
14 has_strength(lava-crab, melee).
15 has_weakness(ice-crab, fire).
```

Isječak koda 6: Baza znanja igrača (Izvor: vlastiti uradak)

Ovaj kod otvara JSON datoteku, dohvaća listu ključeva aktora koji su u "heroes" kategoriji, odabire se nasumični te se odabiru svi njegovi podaci.

Za igrače specifično je, osim dohvaćanja samih atributa, potrebno i nasumično odabrati statistike.

4.1.3. Baze znanja agenata

Kako bi agenti učili koji napadi utječi ili ne utječu na protivničku stranu, kreirane su baze znanja u koje se unose jednostavni Prolog SWI argumenti kojima agenti pristupaju prilikom napadanja.

Postoje dvije odvojene baze znanja za odvojeni pristup protivnika i igrača. Iako se u teoriji sve može pisati u istu bazu jer su iskazi drukčije oslovljeni, implementirane su dvije posebne za preglednost i za lakše brisanje iz baze za demonstraciju.

U nastavku je prikazan kod baze znanja junaka. Baza znanja protivnika sadrži samo jednu vrstu iskaza zvanu "is_weak_to/2" jer agenti igrači nemaju imunost i jačine.

Svi su iskazi dinamični zbog upisivanja u bazu tijekom provođenja programa i diskontinuirani jer se ne upisuju redom, već kako agenti napadači uče ove činjenice.

4.1.4. Implementacija agenata

Agenti "EnemyNPC" i "AllyNPC" se postavljaju tako da preuzimaju sve attribute aktora koji su kreirani u prvom trenutku. Konstruktori su im stoga jednaki kao kod aktera, no uz dodatne varijable "jid" i "password" kako bi se mogli identificirati u mreži agenata.

U nastavku se prikazuju primjeri na temelju agenata protivnika. Postavljanje samog agenta odvija se na način kao što je to prikazano na isječku koda u nastavku.

Setup se odvija asinkrono. Po kreiranju agenta, ispisuje se poruka da protivnik ulazi na

```

1  async def setup(self):
2      print(f"Enemy {self.nname} enters the battlefield!")
3      await asyncio.sleep(2)

4      # setting up knowledge base
5      self.prolog = Prolog()
6      script_dir = os.path.dirname(os.path.realpath(__file__))
7      file_name = "enemy_knowledge_base.pl"
8      self.file_path = os.path.join(script_dir, file_name).replace('\\', '/')

9      ponasanje = self.EnemyBehaviour()
10     self.add_behaviour(ponasanje)
11     self.combat_hp = self.hp

```

Isječak koda 7: Postavljanje agenta (Izvor: vlastiti uradak)

bojište.

Postavlja se "self.prolog" varijabla koja uključuje Prolog is modula "pyswip" korišten za interoperabilnost na dva operacijska sustava. Pronalazi se baza znanja za protivnike i postavlja se protivničko ponašanje objašnjeno u nastavku, prije no što se postavlja HP agenta.

Funkcija "run(self)" kompleksna je funkcija koja u obzir uzima različite ontologije i parametre. Svi agenti u principu samo čekaju na red i primaju poruke od drugih agenata ovisno i ponašaju se na način koji je ovisan o ontologiji koja im se šalje putem poruka.

Ukoliko je ontologija "initiative", odvijaju se koraci kao što je to prikazano na isječku koda 8.

Agent na redu prvo odabire nasumičnu metu.

Na temelju mete, bira napad koji će koristiti protiv nje.

Agent svaki puta provjerava, odnosno konzultira, pripadajuću bazu znanja zbog mogućnosti da je njegov suradnik našao slabost igrača koji mu je trenutno meta. Ako rezultat postoji, agent će pretražiti sadrži li njegova lista napada napad koji je efektivan na metu. Ako postoji, koristit će efektivan napad koji bi stvorio 2 puta veće oštećenje, no ako ne, ponovno bira nasumični napad.

Ako nema podatke o slabosti igrača, također bira nasumični napad.

Izračun napada temelji se na preuzimaju jačine napada iz JSON dokumenta, biranja između "ATK" ili "SPATK" ovisno o načinu napada te oduzimanje istog s pripadajućim "DEF" ili "SPDEF".

Ako je oštećenje veće od 0, provjerava se i slabost mete. Ukoliko je napad dobro efektivan na metu, u bazu se znanja upisuje nova linija u kojoj se navodi agent i njegova slabost te se oštećenje množi s 2, pri čemu se na kraju šalje samo oštećenje koje se postavlja da je 0 ukoliko ispadne manji broj, kako se agenti ne bi liječili ukoliko ih se napadne.

U nastavku se šalju dvije poruke. Prva je poruka poslana napadnutim agentu i sadrži brojku oštećenja koje će izgubiti. Nakon toga, šalje poruku gospodaru tamnice na "dungeon-master@localhost" (DM) koji je ključ za izmjenu reda napadanja agenata.

```

1  if ontology == "initiative":
2      target = self.pick_target()
3      attack = self.pick_attack(target)

4      # for status moves
5      if attack['type'] == "status":
6          await self.send_status(attack, target)
7      else:
8          dmg = self.calculate_damage(attack, target)

9          # send message to give damage to target
10         damage_message = Message(
11             to=str(target.jid),
12             body=str(dmg), # Serialize the tuple into a string
13             metadata={
14                 "ontology": "damage",
15                 "performative": "inform"
16             }
17         )

18         await self.send(damage_message)
19         print(f"{self.agent.nname} used {attack.get('name')} on
20             ↳ {target.nname}, dealing {dmg} HP!")
21         await asyncio.sleep(3)

22         end_turn_msg = Message(
23             to="dungeonmaster@localhost",
24             body="done",
25             metadata={
26                 "ontology": "initiative",
27                 "performative": "inform"
28             }
29         )
30         await self.send(end_turn_msg)
31         self.combat_hp = self.agent.combat_hp

```

Isječak koda 8: Primanje inicijative (Izvor: vlastiti uradak)

```

1  def pick_target(self):
2      target_candidates = [ag for ag in agent_list if isinstance(ag, AllyNPC)]
3      return random.choice(target_candidates) if target_candidates else None

```

Isječak koda 9: Odabir mete napada (Izvor: vlastiti uradak)

```

1 def pick_attack(self, target):
2     try:
3         self.agent.prolog.consult(self.agent.file_path)
4         weakness_result =
5             ↪ list(self.agent.prolog.query(f'is_weak_to({target.nname},
6             ↪ Weakness).'))
7
8         if weakness_result:
9             print(f"The enemy knows {target.nname}'s weakness!")
10            target_weakness = weakness_result[0]["Weakness"]
11            matching_weakness_attacks = [attack for attack in
12            ↪ self.agent.attack_list if attack.get("element") ==
13            ↪ target_weakness]
14
15            if matching_weakness_attacks:
16                chosen_attack = random.choice(matching_weakness_attacks)
17                print(f"...And it attacks {target.nname}'s weakness:
18                ↪ {chosen_attack['element']}!")
19                return chosen_attack
20            else:
21                print("But it doesn't seem to act on the weakness.")
22                # return something, always
23            return random.choice(self.agent.attack_list)
24
25 except Exception as e:
26     print(f"Error in pick_attack: {e}")

```

Isječak koda 10: Odabir napada (Izvor: vlastiti uradak)

```

1 def calculate_damage(self, attack, target):
2     dmge = int(attack.get("dmg"))
3
4     if attack.get("type") == "spatk":
5         dmge += self.agent.spatk
6         dmge -= target.spdef
7
8     elif attack.get("type") == "atk":
9         dmge += self.agent.atk
10        dmge -= target.pdef
11
12    if(dmge >= 0):
13        my_type = attack.get("element")
14        weaknesses = target.weakness
15        # consult the knowledge base to see whether it has the same weakness
16        ↪ for the ally
17        # if not, assert into it
18        if weaknesses == my_type:
19            dmge *= 2
20            self.agent.prolog.consult(self.agent.file_path)
21            if not list(self.agent.prolog.query(f"is_weak_to({target.nname},
22            ↪ {my_type}).")):
23                print(f"The enemy found {target.nname}'s weakness:
24                ↪ {my_type}!")
25                self.write_to_file(f"is_weak_to({target.nname},
26                ↪ {my_type}).")
27        else:
28            dmge = 0
29
30    return round(dmge)

```

Isječak koda 11: Račun oštećenja (Izvor: vlastiti uradak)

```

1 elif ontology == "damage":
2     current_hp = self.agent.combat_hp
3     damage_value = msg.body
4     current_hp = current_hp - int(damage_value)
5     if current_hp < 0:
6         current_hp = 0
7     self.agent.combat_hp = current_hp
8     if current_hp == 0:
9         agent_list.remove(self.agent)
10        print(f"{self.agent.nname} was defeated!")
11        await asyncio.sleep(2)
12        self.kill()

```

Isječak koda 12: Primanje oštećenja (Izvor: vlastiti uradak)

```

1 async def start_turn(self):
2
3     if self.agent.npc_turn < len(agent_list):
4         # first print banner
5         await asyncio.sleep(2)
6         print("-----")
7         print(f"-----TURN-{self.agent.battle_duration}-----")
8         print("-----")
9         await asyncio.sleep(2)
10
11        # now send msg
12        to_whom_it_may_concern = agent_list[self.agent.npc_turn].jid
13        starting_turn_msg = Message(
14            to=str(to_whom_it_may_concern),
15            body="go",
16            metadata={
17                "ontology": "initiative",
18                "performative": "inform"
19            }
20        )
21        await self.send(starting_turn_msg)

```

Isječak koda 13: Funkcija slanja poruka agenta "DM" (Izvor: vlastiti uradak)

Ukoliko dobije poruku s ontologijom "damage", agentu se skida HP.

Ukoliko mu HP padne na 0, ponašanje se prekida i poziva se "on_end" metoda unutar koje se čeka na prekid rada agenta.

Isto tako, agent može dobiti poruku ontologije "gameover", što se događa u slučaju kad se svi agenti s protivničke ili igračke strane pobjede. U tom slučaju, nema smisla dalje igrati igru jer je jedna strana pobijedila, i DM šalje poruku kraja igre, koja također gasi ponašanje agenata.

Agent DM gospodar je tamnice ove igre i uveden je kako bi dijelio inicijativu agentima napadačima. On redom čita agente prema generiranom broju inicijative, od najveće inicijative do najmanje, i tako redom pušta agente u borbu.

Pritom "on_start(self)", agent odmah pokreće funkciju "start_turn(self)", dok u "run(self)" prvo provjerava uvjete unutar igre.

Isprva agent provjerava je li broj reda veći od duljine liste agenata, koja sadrži sve agente u igri. Ukoliko je, šalje poruku inicijative prvim agentu u redu i tako pokreće samu igru.

```

1  async def run(self):
2      msg = await self.receive(timeout=60)
3      if msg:
4          both = self.check_participants()
5          if both:
6              self.agent.npc_turn += 1
7              if self.agent.npc_turn > len(agent_list) - 1:
8                  self.agent.npc_turn = 0
9                  self.agent.battle_duration += 1
10             await self.start_turn()
11         else:
12             print(f"The Game has Been Decided! It lasted
13                 ↳ {self.agent.battle_duration} turns. The victors are:")
14             for agent in agent_list:
15                 print(f"{agent.nname} with {agent.combat_hp} HP left!")
16                 show_picture(agent)
17                 self.kill()
18         else:
19             print(f"Player {agent_list[self.agent.npc_turn].name} is not
20                 ↳ responding.")

```

Isječak koda 14: Run metoda agenta "DM" (Izvor: vlastiti uradak)

```

1  async def on_end(self):
2      print(f"Cleaning up Game...\n")
3      await asyncio.sleep(2)
4      for a in agent_list:
5          end_msg = Message(
6              to=str(a.jid),
7              body="bye",
8              metadata={
9                  "ontology": "gameover",
10                 "performative": "inform"
11             }
12          )
13      await self.send(end_msg)
14      await wait_until_finished(a)
15      await self.agent.stop()

```

Isječak koda 15: Metoda on_end(self) agenta "DM" (Izvor: vlastiti uradak)

Nakon što igrač završi svoj red, agent će primiti natrag poruku te proslijediti inicijativu sljedećem agentu u redu. To radi u "run(self)" funkciji koja je sastavljena od čekanja na poruke uz timeout od 60 sekundi.

Provjerava se postoje li u igri još i protivnici i igrači. Ukoliko postoje, inkrementira se "npc_turn" i postavlja se logika za brojanje ukupnih redova odnosno poteza u igri. Ukoliko nema više jedne protivničke strane, igra je završena i DM svim preostalim agentima šalje poruku "gameover" ontologije pri završetku njegova ponašanja.

Na taj način, stvara se ciklička igra s izmjenama poteza svih sudjelujućih agenata.

```
1 agent_list = sorted(actors, key=lambda x: x.initiative, reverse=True)
```

Isječak koda 16: Sortiranje aktera prema inicijativi (Izvor: vlastiti uradak)

```
1 async def let_agents_loose(agent_list):
2     players = []
3     i = 1
4
5     # kreiraj broj agenata koji su u listi agenata
6     for attacker in agent_list:
7         # create agent with attacker behavior (for now)
8         new_player = await create_instance(attacker, i)
9         i += 1
10        players.append(new_player)
11
12    # svi su agenti pripremljeni i cekaju poruke
13    # kreiraj prvog agenta koji je DM
14    # DM salje poruke agentu po redu, s inicijativom
15    # ujedno salje listu igraca kako bi znali target
16    dm = DM("dungeonmaster@localhost", "tajna", players)
17    await dm.start()
18
19    # završi igru
20    await wait_until_finished(dm)
```

Isječak koda 17: Kreiranje agenata (Izvor: vlastiti uradak)

4.2. Izgradnja okruženja agenata

Program se izvodi i pokreće preko "main.py" skripte, koja provodi funkcije "setupGame()" koja vraća igrače igre i "startCombat(actors)" koja postavlja okruženje igre te izvodi samu simulaciju.

Metoda "setupGame()" nalazi se u posebnoj "stupGame.py" skripti koja dohvaća nasumičnih X junaka i protivnika postupkom kako je opisano ranije. Nakon njihova dohvaćanja, predaju se metodi "startCombat(actors)", koja se nalazi u "combat.py" datoteci. Isprva se za svakog agenta u listi postavlja inicijativa od 1 do 20 te se akteri sortiraju prema inicijativi.

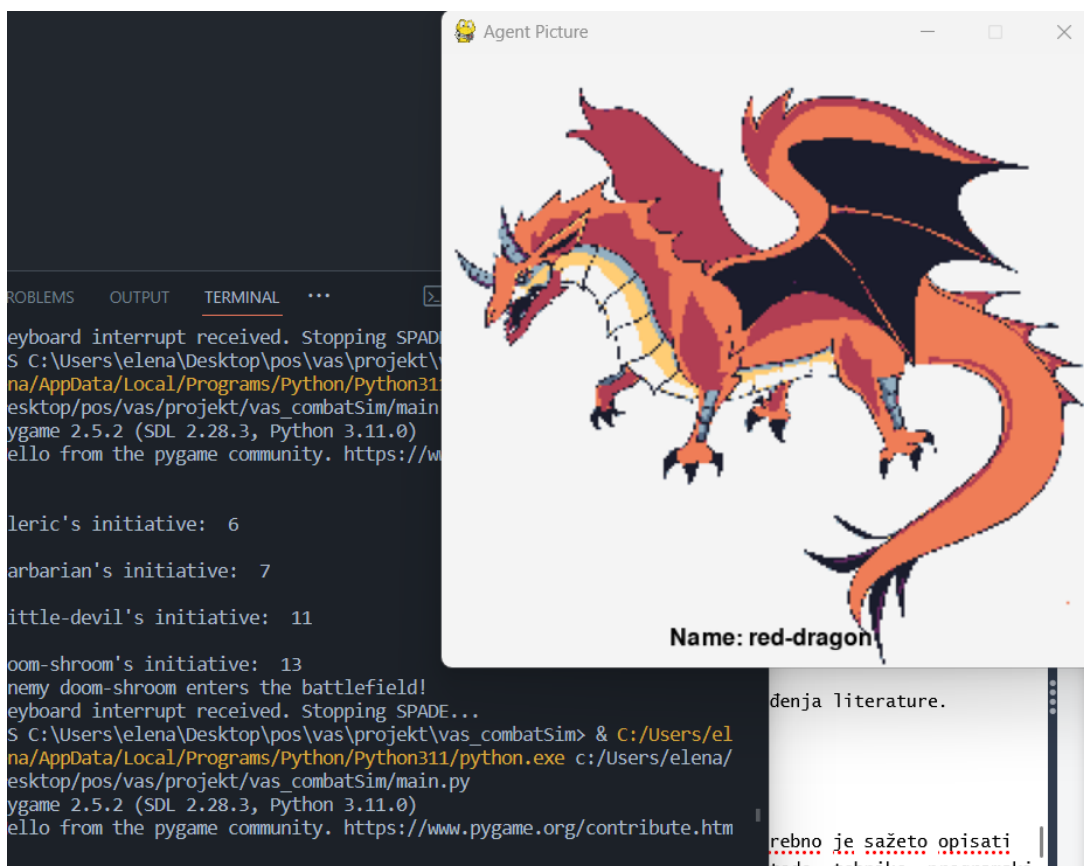
Funkcija "let_agents_loose(agent_list)" pokreće agente igrače koje naziva "player<i>", pri čemu je "i" broj od 1 do duljine liste agenta kako bi se kreirali agenti određenih naziva.

Nakon što su svi igrači pokrenuti, kreira se "dungeonmaster@localhost" kojem se proslijeđuje lista igrača, te se čeka na njegov početak i završetak.

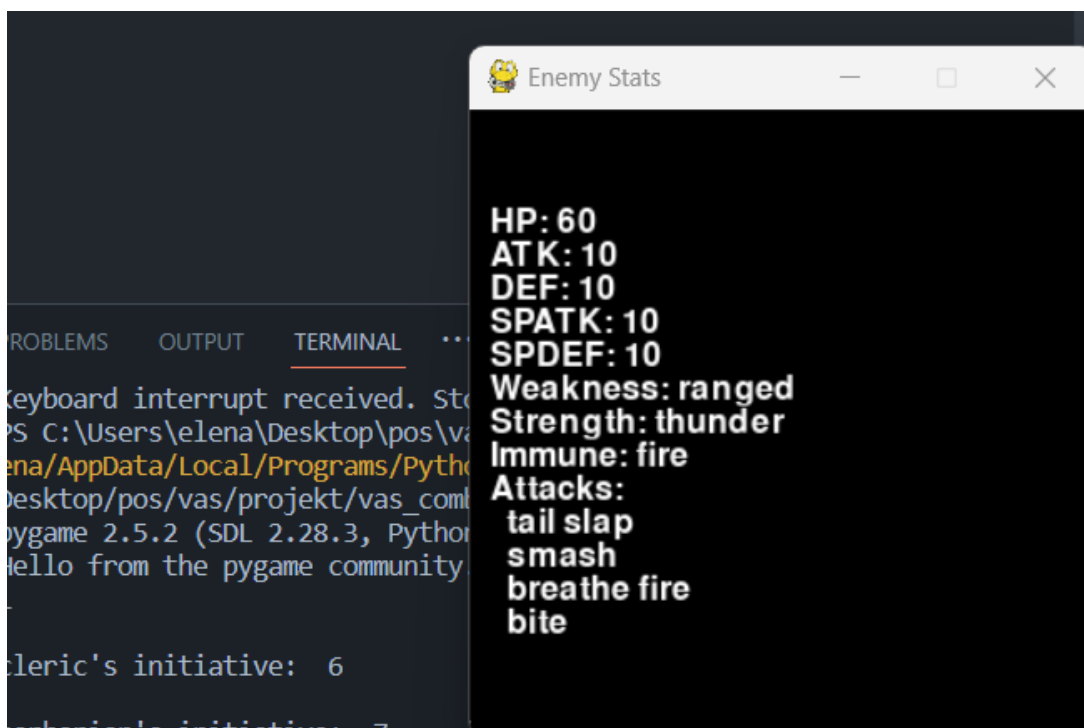
5. Prikaz rada aplikacije

U ovom se dijelu rada prikazuje sam rad aplikacije uz slike ekrana njezina rada.

Aplikacija se pokreće putem "main.py" skripte koja poziva druge dijelove aplikacije. Sav se sadržaj igre ispisuje na terminal, uz posebnost pokazivanja agenata na početku te pobjednika na kraju.



Slika 3: Prikaz slike napadača (Izvor: vlastiti uradak)



Slika 4: Prikaz statistike napadača (Izvor: vlastiti uradak)


```
The DM Enters the Game!
-----
-----TURN-1-----
-----
We know the strength of ice-crab (melee)!
...So we will instead use thoron!
wizard used thoron on ice-crab, dealing 12 HP!
ice-crab was defeated!
-----
-----TURN-1-----
-----
barbarian used slap on doom-shroom, dealing 7 HP!
-----
-----TURN-1-----
-----
doom-shroom used whip on barbarian, dealing 0 HP!
-----
-----TURN-2-----
-----
We found doom-shroom's immunity: poison!
wizard used miasma on doom-shroom, dealing 0 HP!
-----
-----TURN-2-----
-----
We know the immunity of doom-shroom (poison)!
```

Slika 5: Pronalazak slabosti (Izvor: vlastiti uradak)

```
12 is_immune(ice-crab, ice).
13 has_strength(ice-crab, melee).
14 has_weakness(lava-crab, ice).
15 has_weakness(doom-shroom, fire).
16 has_strength(lava-crab, melee).
17 is_immune(little-devil, poison).
18 has_weakness(red-dragon, ranged).
19 has_strength(undine, ice).
20 has_weakness(undine, thunder).
21 is_immune(red-dragon, fire).
22
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python

Enemy lava-crab enters the battlefield!
Enemy red-dragon enters the battlefield!
Ally barbarian enters the battlefield!
The DM Enters the Game!
-----TURN-1-----


We know the weakness of lava-crab!
...And I have an attack of the same element as lava-crab's weakness: {'name': 'snowball', 'dmg': 1, 'target': 'single', 'element': 'ice', 'type': 'spatk'}!
bard used snowball on lava-crab, dealing 14 HP!
lava-crab was defeated!
-----TURN-1-----
red-dragon used smash on barbarian, dealing 13 HP!
-----TURN-1-----
We know the weakness of red-dragon!
...And I have an attack of the same element as red-dragon's weakness: {'name': 'throw', 'dmg': 5, 'target': 'single', 'element': 'ranged', 'type': 'atk'}!
barbarian used throw on red-dragon, dealing 8 HP!
barbarian was defeated!
-----TURN-2-----

Slika 6: Prikaz korištenja baze znanja (Izvor: vlastiti uradak)

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
```

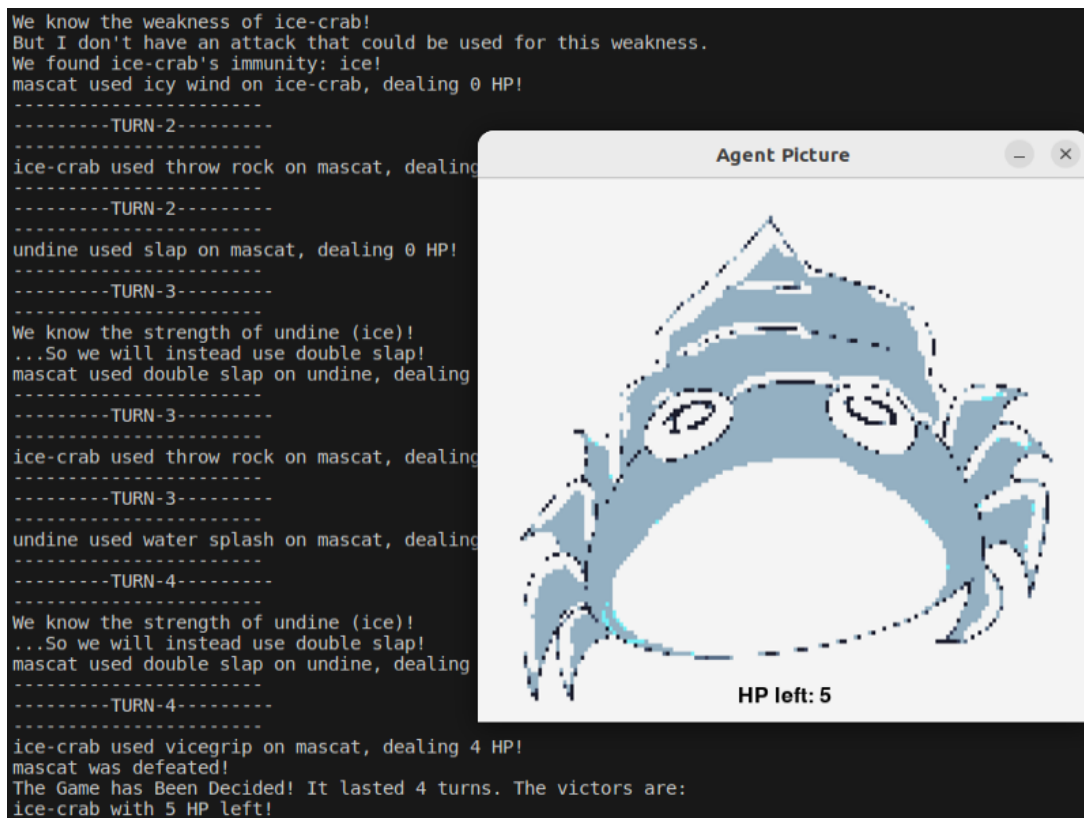
Ranger was defeated!
-----TURN-1-----
Ghostie used touch on Mascat, dealing 0 HP!
-----TURN-2-----
Mascat used ember on Ghostie, dealing 10 HP!
Ghostie was defeated!
-----TURN-2-----
Red Dragon used bite on Mascat, dealing 6 HP!
-----TURN-3-----
Mascat used bow storm on Red Dragon, dealing 6 HP!
-----TURN-3-----
Red Dragon used bite on Mascat, dealing 6 HP!
-----TURN-4-----
Mascat used bow storm on Red Dragon, dealing 6 HP!

Agent Picture



HP left: 54

Slika 7: Ispis pobjednika bez pronalaska slabosti (Izvor: vlastiti uradak)



Slika 8: Trenirani pobjednik (Izvor: vlastiti uradak)

6. Kritički osvrt

Primjena aplikacije je zabava, igra i razonoda. Igra uzima u obzir složenost borbenog sustava unutar RPG igara temeljenih na potezima uz SPADE agente koji operiraju asinkrono. U igru su uključeni napadi i iscjeljivanje uz uzimanje dviju različitih skupina agenata u obzir.

Igra također koristi bazu znanja kako bi agenti naučili slabosti, jačine i imunosti protivničke strane tako da se mogu birati potezi na temelju njih.

Aplikacija za svoje izvođenje treba Python programski jezik, Prolog SWI, XMPP server poput Prosody ili Ejabberda, te dodatne module spomenute ranije.

Projekt je izvediv i kompleksan, no proširiv i postoji puno prostora za dodavanje novih značajki, metoda i funkcionalnosti. Neke od značajki su primjerice dodavanje statusnih efekata koji bi mogli proširiti postojeće funkcionalnosti, postavljanje različitih uvjeta kako igra ne bi trajala predugo, uključivanje različitosti protivnika i dodavanje novih klasa, detaljizirati bazu znanja i iskorištavanje određenih strategija napadanja te razvoj kompletnog grafičkog sučelja. Igra je napravljena tako da bude proširiva te je dodavanje ovih značajki moguće.

Razvoj aplikacije koja učinkovito troši resurse i koristi nova saznanja i autonomiju kompleksan je, no vidljiva je korist ovakve implementacije igara zbog mogućnosti autonomije i komunikacije agenata uz dostizanje željenih ciljeva.

7. Zaključak

U radu se kreirao višeagentni sustav koji simulira RPG borbu između dviju odvojenih skupina junaka i protivnika, kreiranih putem Python SPADE tehnologije uz konzultiranje baze znanja unutar kojih se popisuju otkrivene slabosti, imunosti i jačine suprotne strane kako bi se omogućio inteligentniji odabir napada za što veće postizanje efekata. Agenti samostalno djeluju u postavljenoj okolini, samoregulirajući su i autonomno vode cijelu sustav do završetka igre, ispisivajući pobjedike i spremajući naučeno znanje unutar dinamičnih SWI Prolog datoteka.

Postavljanje samih agenata u takvo okruženje proces je koji zahtjeva mnogo planiranja, no vidljivo je da se ovim putem igre mogu optimizirati zbog načina na koji interagiraju s drugim agentima i s naučenim znanjem te je tehnologija veoma primjenjiva i na većim, kompleksnijim sustavima unutar kojih su odluke vrlo bitan aspekt igre.

Popis literature

- [1] W. Van der Hoek i M. Woolridge, „Multi-agent systems,” *IEEE Access*, sv. 6, str. 28 573–28 593, 2018. DOI: 10.1109/ACCESS.2018.2831228.
- [2] V. Julian i V. Botti, „Multi-agent systems,” *Applied Sciences*, str. 1–7, 4. 2019.
- [3] H. Zedan i A. Cau, *Object-Oriented Technology and Computing Systems Re-engineering*. Woodhead Publishing, 1999., ISBN: 978-1-898563-56-3.
- [4] J. Palanca. „SPADE Documentation.” (2018.), adresa: <https://buildmedia.readthedocs.org/media/pdf/spade-mas/develop/spade-mas.pdf>.
- [5] J. Westra, W. van Doesburg i M. Harbers, „Games and agents: Designing intelligent gameplay,” *International Journal of Computer Games Technology*, str. 1–19, 3. 2009.
- [6] T. Moncef, R. Ali i M. G., „Game decision making in multi agent systems,” *Journal of Global Research in Computer Science*, sv. 4, str. 71–75, 2013.
- [7] R. A. Agis, S. Gottifredi i A. J. Garcia, „An event-driven behavior trees extension to facilitate non-player multi-agent coordination in video games,” *ScienceDirect*, sv. 155, 2020.

Popis slika

1.	Prikaz slike igrača (Izvor: vlastiti uradak)	9
2.	Prikaz statistike igrača (Izvor: vlastiti uradak)	10
3.	Prikaz slike napadača (Izvor: vlastiti uradak)	19
4.	Prikaz statistike napadača (Izvor: vlastiti uradak)	19
5.	Pronalazak slabosti (Izvor: vlastiti uradak)	20
6.	Prikaz korištenja baze znanja (Izvor: vlastiti uradak)	21
7.	Ispis pobjednika bez pronalaska slabosti (Izvor: vlastiti uradak)	21
8.	Trenirani pobjednik (Izvor: vlastiti uradak)	22

Popis isječaka koda

1.	Struktura agentovih podataka (Izvor: vlastiti uradak)	6
2.	Struktura podataka junaka (Izvor: vlastiti uradak)	6
3.	Konstruktor apstraktne klase "AbstractActor" (Izvor: vlastiti uradak)	7
4.	Kod za prikazivanje slike (Izvor: vlastiti uradak)	8
5.	Kod postavljanja junaka (Izvor: vlastiti uradak)	10
6.	Baza znanja igrača (Izvor: vlastiti uradak)	11
7.	Postavljanje agenta (Izvor: vlastiti uradak)	12
8.	Primanje inicijative (Izvor: vlastiti uradak)	13
9.	Odabir mete napada (Izvor: vlastiti uradak)	13
10.	Odabir napada (Izvor: vlastiti uradak)	14
11.	Račun oštećenja (Izvor: vlastiti uradak)	14
12.	Primanje oštećenja (Izvor: vlastiti uradak)	15
13.	Funkcija slanja poruka agenta "DM" (Izvor: vlastiti uradak)	15
14.	Run metoda agenta "DM" (Izvor: vlastiti uradak)	16
15.	Metoda on_end(self) agenta "DM" (Izvor: vlastiti uradak)	16
16.	Sortiranje aktera prema inicijativi (Izvor: vlastiti uradak)	17
17.	Kreiranje agenata (Izvor: vlastiti uradak)	17