SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

PRIMJENA REINFORCEMENT LEARNINGA ZA KONTROLU 2D VOZILA.

Seminarski rad

Raspoznavanje uzoraka i strojno učenje

Luka Šimić

Osijek, 2020

Sadržaj

1. UVOD

Pojačano učenje (engl. *reinforcement learning*) je postupak kojim treniramo agent kako bi donosio odluke(engl. *action*) temeljene na dostupnim opažanjima (engl. *observation*). Agent se uči tako što za određenu odluku ili niz odluka dobije nagradu (engl. *reward*). U slučaju dobre odluke, nagrada je pozitivna, dok je u slučaju loše odluke nagrada manja ili negativna. Agent s vremenom teži tome da dobije najveću moguću nagradu, što će ostvariti donošenjem optimalnih odluka.

Projekt se sastoji od agenta koji uči upravljati pojednostavljenim modelom vozila. Cilj agenta je u što kraćem vremenu (u što manjem broju sličica(engl. *frames*) posjetiti unaprijed definirane točke u 2D ravnini. Vozilo je definirano brzinom, maksimalnim kutom skretanja, te svojom pozicijom i orijentacijom u 2D ravnini. Definirane točke su također definirane pomoću X i Y koordinata. Ravnina u kojoj se nalaze vozilo i točke je ograničena u na interval [-50, 50] po obje osi.

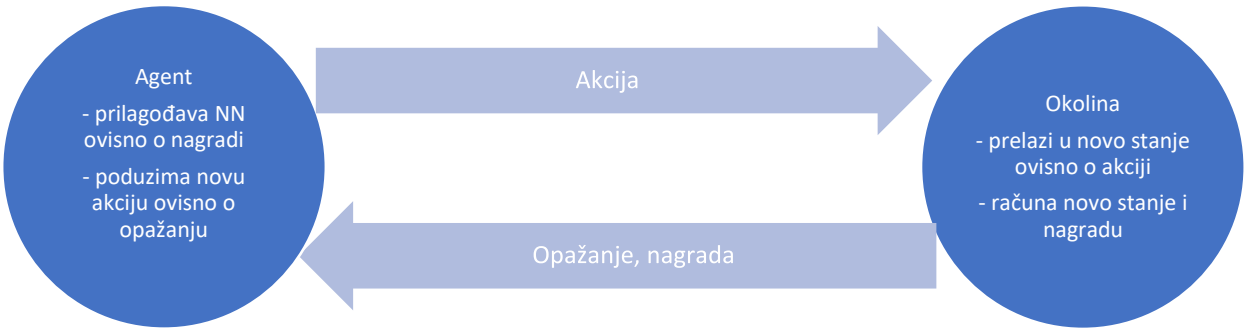
Projekt je implementiran koristeći programski jezik *Python* i biblioteku *tf-Agents* [1] *(tensorflow-agents*).

2. POJAČANO UČENJE

Pojačano učenje (engl. *reinforcement learning)* je područje strojnog učenja pomoću kojega agenti (softverski) na osnovu prethodne interakcije s okolinom donose odluke koje mijenjaju stanje okoline, s ciljem ostvarivanja što veće nagrade.

Osnovni tijek pojačanog učenja je slijedeći. Agent, odnosno NN se inicijalizira s nasumičnim težinama. Agent u ovisnosti o opažanju iz okoline na svom izlazu daje određenu akciju. Okolina mijenja svoje stanje u ovisnosti o poduzetoj akciji, te vraća vrijednosti opažanja i nagrade za prethodni korak. Zatim se težine NN podešavaju ovisno o tome je li nagrada bila pozitivna, odnosno negativna, i većeg, odnosno manjeg iznosa.

Pojednostavljeni prikaz pojačanog učenja prikazan je na slici 2.1.



Slika 2.1 - Pojednostavljeni tijek pojačanog učenja

2.1. Agent

Agent pojačanog učenja predstavlja algoritam kojim se rješava određeni problem. Agent za donošenje odluka najčešće koristi neuronsku mrežu, ali može koristiti i neke druge modele.

Svrha agenta je da procesira opažanja i nagradu koju dobije iz okoline, te generira akciju. Zatim na osnovu povratne informacije (novog opažanja i nagrade) odlučuje je li donesena odluka u prethodnom koraku bila dobra ili loša, te optimizira model/mrežu za donošenje odluke kako bi kod budućih slučajeva povećao vjerojatnost donošenja odluke koja daje najveću nagradu

Neki od algoritama koji se često koriste su *brute force, Value function, Monte Carlo methode, Q learning…*

2.2. Okolina, opažanja i akcije

Okolina (engl. *environment*) opisuje okruženje u kojemu se agent nalazi, te iz kojega dobiva opažanja. Okolina u svakom koraku, ovisno o poduzetoj akciji prelazi u novo stanje. Za okolinu ne mora biti poznato optimalno analitičko rješenje, niti okolina mora biti računalno simulirana, ali je bitno da agent smije donositi loše odluke bez štetnih posljedica.

Svaka okolina načelno sadrži dvije osnovne funkcije:

- *step(action)* – mijenja stanje okoline ovisno o poduzetoj akciji, i kao rezultat vraća nagradu(engl. *reward*) i opažanje(engl. *observation*).
- *reset()* – postavlja stanje okoline na prethodno definirane ili nasumične vrijednosti.

Opažanja mogu biti vektori i matrice, odnosno slike, koji predstavljaju stanje okoline. Opažanja u sebi ne sadrže nužno sve varijable koje sadrži okolina, već samo određene varijable koje su na neki način vidljive agentu.

Akcije mogu biti u obliku vektora ili matrice, te mogu biti diskretne ili kontinuirane. Kod diskretnih akcija agent odabire jednu od mogućih akcija, dok kod kontinuiranih odabire vrijednost iz prethodno definiranog raspona

Postoji nekoliko biblioteka, od kojih je najpoznatija *OpenAI Gym* [2], koje služe kao standardna implementacija određenih problema pojačanog učenja, i koriste se kao *benchmark* za algoritme pojačanog učenja.

2.3. Funkcija nagrade

Funkcija nagrade (engl. *reward function*) je funkcija koja računa nagradu nakon poduzete akcije.

Velik problem pojačanog učenja je upravo oblikovanje ove funkcije. U slučaju gdje okolina dodjeljuje nagradu samo u koraku kada dođe u željeno stanje, agent će morati dobro odabrati sve korake u nizu kako bi došao do nagrade. Često je taj broj koraka velik, pa je vjerojatnost da će agent svojim akcijama dovesti okolinu u traženo stanje minimalna. Zbog toga agent nikada ne može dobiti pozitivnu nagradu, pa zbog toga i samo rješenje nikada ne konvergira. Ta se situacija naziva *sparse reward setting*.

Jedan od načina na koji možemo pristupiti ovom problemu je da svaki korak damo malu nagradu, kako bismo bolje usmjerili agent prema željenom stanju. U ovom slučaju postoji vjerojatnost da agent ne konvergira prema optimalnom rješenju, već se prilagodi navedenim malim nagradama kako bi maksimizirao nagradu. U najgorem slučaju agent se može *overfittati* malim nagradama tako što pronađe način da ih eksploatira, i nikada neće odvesti okolinu u traženo stanje.

Također, jedan od problema je što određene akcije koje agent poduzme često nemaju trenutačni efekt, i njihov će utjecaj tek biti vidljiv nakon nekoliko koraka. Analogno tome, nagrada ne proizlazi nužno iz akcija koje su poduzete u prethodnom koraku.

2.4. DQN

Q-Učenje (engl. *Q learning)* [3] je algoritam za pojačano učenje. Cilj agenta za duboko Q učenje (engl. deep Q learning, DQN), je doći do sustava odluke (engl. *policy*) koji odluke donosi na način kako bi došao do najvećih Q vrijednosti (engl. *Q Values*). Q vrijednost predstavlja očekivani iznos nagrade za sve buduće korake simulacije.

Q učenjem se uz beskonačan broj koraka(beskonačno vrijeme treniranja) može odrediti optimalni sustav odluke, koji će donositi takve odluke da nagrada bude najveća. U slučaju loše oblikovane funkcije nagrade, agent će doći do takvog sustava koji će maksimizirati nagradu (a možda nije optimalan) (engl. *exploatation).*

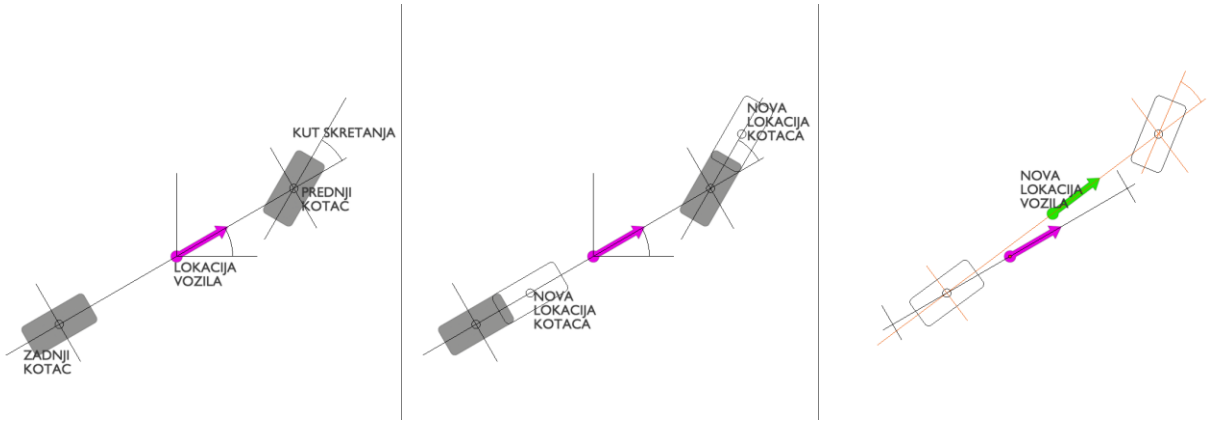
3. PRIJEDLOG RJEŠENJA

Programsko rješenje implementirano je koristeći programski jezik Python i biblioteku *tf-learn*. Okolina predstavlja vozilo s jednostavnim fizikalnim modelom [4] koje mora obići definirane točke u 2D ravnini. Osnovni izgled implementacije jedne okoline dan je isječkom koda 3.1.

3.1. Definicija okoline i vozila

Okolina je zasnovana na pojednostavljenom modelu [4] 2D vozila (slika 3.1.). Svaki korak simulacije (engl. *step, timestep*) provodimo sljedeći postupak kako bismo odredili novu poziciju vozila na osnovu koje izračunavamo nagradu i provjeravamo uvjet završenosti simulacije:

1. Izračunamo poziciju prednje i stražnje osovine koristeći definirani osovinski razmak i poznati vektor kretanja vozila
2. Pomaknemo stražnju osovinu u smjeru kretanja vozila, a prednju osovinu u smjeru kretanja vozila + kut skretanja
3. Izračunamo novu poziciju vozila kao sredinu između pozicija prednje i stražnje osovine.



Slika 3.1 – prikaz fizike vozila

Model okoline u svakom koraku prima jednu od tri moguće akcije:

- Vozilo skreće lijevo
- Vozilo nastavlja voziti ravno
- Vozilo skreće desno

Okolina kao rezultat vraća slijedeće vrijednosti, normalizirane u intervalu [-1, 1]:

- X, Y koordinate trenutnog cilja
- X, Y koordinate trenutne pozicije vozila
- X, Y elementi vektora koji predstavlja trenutni smjer kretanja.
- Kut između vektora kretanja i vektora smjera prema trenutnom cilju

Simulacija se prekida u sljedećim slučajevima:

- Vozilo izađe izvan definiranih granica simulacije
- Vozilo dosegne traženu točku
- Dosegnut je maksimalan broj koraka (*frameova*) simulacije.

U radu je korišten kut skretanja od 15°, brzina od 20m/s, te veličina okoline 100m x 100m. Maksimalan broj koraka je 50. Točka se smatra posjećenom ako je udaljenost manja od 5m.

3.2. Implementacija

Rješenje je implementirano koristeći programski jezik Python, te biblioteke *tensorflow, tf-agents* i *numpy*. Uz skripte za treniranje i testiranje modela koje koriste *tensorflow checkpoint* sustav, implementirane su i dvije verzije okoline (*train_environment i test_environment)*. Kod okoline za treniranje, u jednoj iteraciji postoji samo jedna ciljana točka, dok se u okolini za testiranje, nakon što vozilo dosegne željenu točku, generira nova točka i brojač koraka se resetira.

3.2.1. Implementacija okoline

Okolina (engl. environment) nasljeđuje klasu *PyEnvironment* i pruža sučelje (isječak koda 3.1.) koje je potrebno za izgradnju modela i treniranje mreže. Metode *action_spec()* i *observation_spec()* vraćaju informacije o tome u kojem formatu i obliku mreža može očekivati opažanja i akcije. Te su vrijednosti potrebne kako bi se mogli konstruirati neuroni ulaznog i izlaznog sloja.

```
class MyEnv(py_environment.PyEnvironment):
    def __init__(self):
        ...
        self._reset()

    def observation_spec(self):
        return self._observation_spec

    def action_spec(self):
        return self._action_spec

    def _reset(self):
        ...
        return time_step.restart(observation)

    def _step(self, action):
        ...
        return time_step...(observation, reward)
```

Isječak koda 3.1. – sučelje environment klase

Za generiranje pseudo-nasumičnih vrijednosti za početne pozicije vozila i točaka koristi se *random* funkcija *numpy* biblioteke (isječak koda 3.2.). Sve se vrijednosti generiranju u definiranom rasponu koji je zadan u *__init__()* funkciji. Nasumične se vrijednosti generiraju u *reset()* metodi, pa se zbog toga ona poziva nakon instanciranja klase.

```
self.target = (
    numpy.random.uniform(-0.25 * self.width, 0.25 * self.width),
    numpy.random.uniform(-0.25 * self.height, 0.25 * self.height))
self.car_location = (
    numpy.random.uniform(-0.25 * self. width, 0.25 * self. width),
    numpy.random.uniform(-0.25 * self. height, 0.25 * self. height))
self.car_heading = numpy.random.uniform(-math.pi, math.pi)
```

Isječak koda 3.2 – generiranje pseudo nasumičnih vrijednosti

Opažanje stvaramo kao *numpy.array* (isječak koda 3.3), u koji učitavamo koordinate cilja i vozila, vektor koji opisuje kretanje te kut između smjera kretanja i trenutnog cilja. Sve su vrijednosti radi lakšeg rada odmah normalizirane u interval [-1, 1]. Smjer kretanja se normalizira tako da ga pretvaramo iz polarnih u kartezijeve koordinate uz duljinu vektora = 1, dok se koordinate normaliziraju tako da ih podijelimo s vrijednostima koje predstavljaju veličinu/granicu okoline.

```
vel_cart = polar_to_cartesian((1, self.car_heading))
observation = numpy.array([
    self.target[0] / (0.5 * self. width), self.target[1] / (0.5 * self. height),
    self.car_location[0] / (0.5 * self. width), self.car_location[1] / (0.5 * self.height),
    vel_cart[0], vel_cart[1],
    angle_difference_end / math.pi
])
observation = observation.astype(numpy.float32)
```

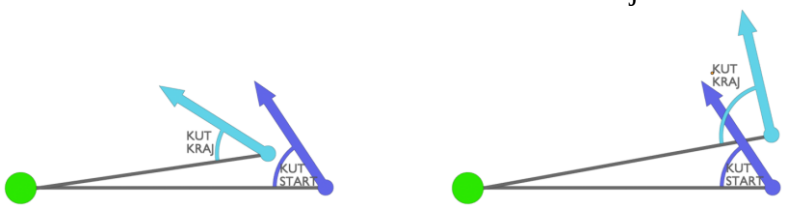
Isječak koda 3.3 – generiranje opažanja

Uz sve navedeno, implementirano je i nekoliko pomoćnih (engl. *utility*) funkcija koje olakšavaju rad s vektorima, uključujući pretvorbu iz polarnih u kartezijeve koordinate, računanje kuta, euklidske udaljenosti i sl. Funkcije su implementirane koristeći ugrađenu Python biblioteku *math*.

3.2.2. Implementacija funkcije nagrade

Nagrade se dodjeljuju prema slijedećoj funkciji:

- $\frac{početni\ kut - krajnji\ kut}{max.\ kut\ skretanja}$ za svaki korak gdje se vozilo odmakne od trenutnog cilja
- $25 + 75 \cdot \left(1 - \frac{tren.\ korak}{max\ \ korak}\right)$ svaki put kada vozilo dođe do ciljane točke
- -100 ako vozilo izađe izvan definiranih granica simulacije
- $25 \cdot \left(1 - \frac{trenutna\ udaljenost}{početna\ udaljenost}\right)$ na posljednjem koraku simulacije
- 2.5 ako se vozilo stvori unutar cilja



Slika 3.2 – dobra akcija(lijevo) je ona gdje je kut na kraju koraka manji nego na početku

3.2.3. Agent i neuronska mreža

U projektu je korišten DQN agent uz mrežu sa 2 *fully connected* sloja od 32 neurona. Za ulazni sloj korišteno je 7 neurona, što odgovara broju elemenata u vektoru opažanja, dok se izlazni sloj sadrži od 3 neurona, koji predstavljaju 3 moguće akcije. Brojevi neurona ulaznog i izlaznog sloja dohvaćaju se koristeći *action_spec()* i *observation_spec()* kao što je prikazano na isječku koda 3.4.

```
q_net = q_network.QNetwork(
    train_env.observation_spec(),
    train_env.action_spec(),
    fc_layer_params=(32,32) )
```

Isječak koda 3.4 – primjer Q-Mreže koristeći tf-agents

4. REZULTATI

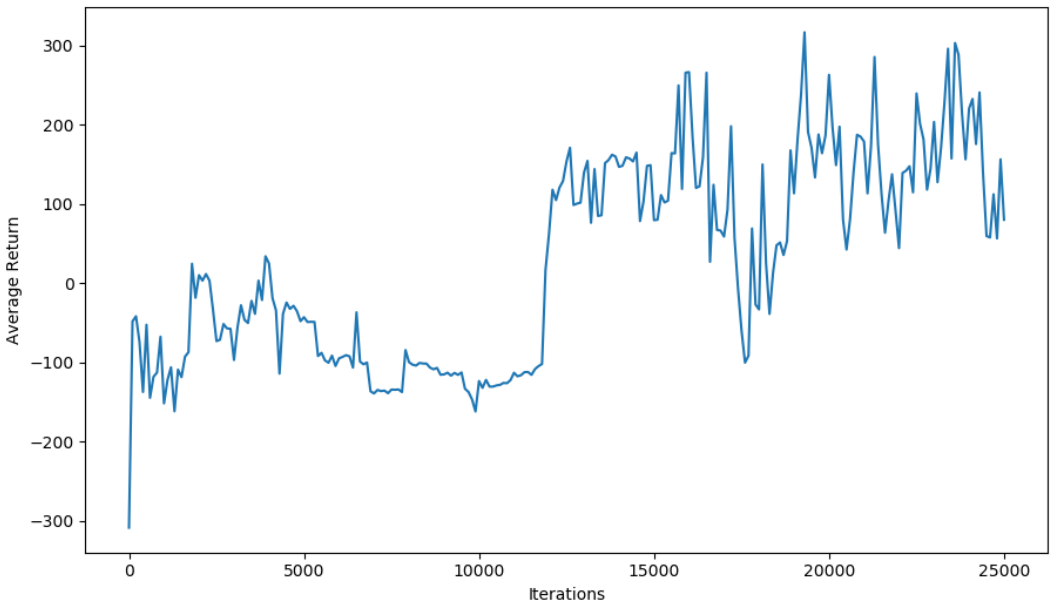
4.1. Učenje agenta

Agent se trenira u petlji, gdje u svakoj iteraciji prikupimo iskustvo(engl. *Experience*) određeni broj epizoda (nizovi opažanja, akcije i nagrade) koje spremimo u *replay buffer*. Nakon toga se agent trenira na tom iskustvu i prilagođava težine neuronske mreže. Uz to, u određenim intervalima provjeravamo prosječnu nagradu kao ocjenu kvalitete mreže i spremamo *checkpoint* (trenutno stanje) mreže. Ovakav oblik učenja se naziva petlja treniranja/učenja (engl. *training loop).* Primjer petlje za treniranje agenta dan je isječkom koda 4.1.

```
for _ in range(num_iterations):
    collect_data(train_env, agent.collect_policy, collect_episodes_per_iteration)
    experience, unused_info = next(iterator)
    train_loss = agent.train(experience)
    step = agent.train_step_counter.numpy()
    if step % eval_interval == 0:
        numpy.random.seed(10)
        avg_return = compute_avg_return(eval_env, agent.policy, num_eval_episodes)
        print('step = {0}: Average Return = {1}'.format(step, avg_return))
        numpy.random.seed()
    if step % checkpoint_interval == 0:
        checkpoint = tensorflow.train.Checkpoint(q_net=q_net, optimizer=optimizer)
        checkpoint.save(file_prefix=CHECKPOINT_PREFIX)
```

Isječak koda 4.1 – petlja učenja

Agent je treniran 25000 iteracija, a kao metrika je korištena prosječna nagrada kroz 10 iteracija. Pri računanju prosječne nagrade, svaki puta je korištena ista vrijednost *seeda* za generator pseudo nasumičnih brojeva - *numpy.random.seed().* Na slici 4.1 prikazana je ovisnost prosječne nagrade o broju iteracija petlje učenja.



Slika 4.1 - ovisnost prosječne nagrade o broju iteracija

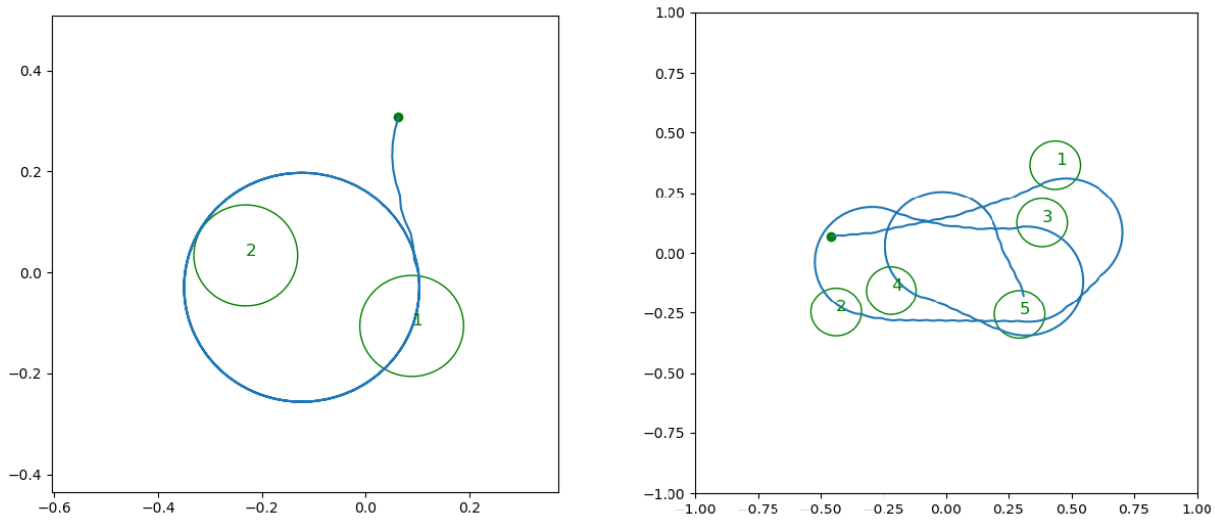
4.2. Testiranje agenta

Nakon što vozilo posjeti ciljanu točku u okolini za testiranje, trenutni korak simulacije postavljamo na nulu, te nasumično generiramo novu točku. To radimo za proizvoljan broj točaka, a kada vozilo posjeti sve generirane točke, simulacija se završava. Q-Mrežu učitavamo koristeći *checkpoint* sustav.

Svako opažanje spremamo u listu, koju na kraju zapisujemo u *.csv* datoteku, te prikazujemo putanju kroz točke koristeći biblioteku *matplotlib*. Primjer petlje za testiranje dan je isječkom koda 4.2., dok je primjer putanje kroz 5 točaka dan na slici 4.2. Petlja se izvršava sve dok ne dođemo do posljednjeg koraka simulacije (što može biti zbog izlaska vozila izvan granica, dolaska do maksimalnog broja koraka, ili zbog uspješnog prolaženja kroz točke).

```
time_step = eval_env.reset()
result.append(time_step.observation[0].numpy())
while not time_step.is_last():
    action_step = agent.policy.action(time_step)
    time_step = eval_env.step(action_step.action)
    result.append(time_step.observation[0].numpy())
with open("trained_outputs.csv","w", newline='') as outfile:
    csv_out=csv.writer(outfile)
    for row in result:
        csv_out.writerow(row)
```

Isječak koda 4.2 – petlja za testiranje agenta



Slika 4.2 – primjeri generirane putanje. Zelene točke predstavljaju početnu lokaciju vozila, dok zeleni krugovi predstavljaju ciljeve, označene redim brojem.

Na slici 4.2. prikazane su generirane putanje vozila. Desno vidimo primjer dobrog prolaska kroz pet različitih točaka. S lijeve strane vidljiv je slučaj kada agent ne može uspješno posjetiti određenu točku. Zbog relativno male udaljenosti agent nije u mogućnosti dovoljno jako skrenuti kako bi došao do cilja. Zbog toga što agent prioritizira brzinu, nije naučio da se u ovom slučaju mora udaljiti od cilja kako bi mogao dovoljno skrenuti. Ovo je jedan od rubnih slučajeva koji se ne pojavljuju često u okolini, pa zbog toga agent ne može naučiti kako se ponašati u tom slučaju.

Agent je evaluiran na tri različite točke u treniranju (nakon 5000, 15000 i 25000 iteracija). Kao mjera kvalitete uzima se broj ponavljanja(od 100) u kojima agent uspješno posjećuje niz ciljeva. Također, mjereno je vrijeme i standardna derivacija. Rezultati evaluacije prikazani su u tablici

Trenirano na:	Broj ciljeva	1 cilj	2 cilja	5 ciljeva	10 ciljeva
5k iteracija	uspjeh	100/100	37/100	0/100	0/100
	pr. vrijeme	3.39s	4.66s	N/A	N/A
	st. dev.	2.1021	1.8621	N/A	N/A
15k iteracija	uspjeh	100/100	55/100	16/100	0/100
	pr. vrijeme	5.44s	9.46s	18.54s	N/A
	st. dev.	3.6152	4.0826	4.2720	N/A
25k iteracija	uspjeh	100/100	91/100	79/100	59/100
	pr. vrijeme	3.10s	5.37s	12.83s	25.95s
	st. dev.	2.2897	2.1890	3.0395	4.9754

Tablica 4.1 – rezultati evaluacije

Vidljivo je kako je najmanje prosječno vrijeme za posjećivanje 2 cilja potrebno agentu koji je treniran 5000 iteracija, ali taj agent također uspijeva posjetiti najmanji broj točaka. Iz toga možemo zaključiti da agent uspijeva riješiti samo jednostavnije slučajeve, za koje u prosjeku treba manje vremena, dok ne uspijeva riješiti kompleksne slučajeve.

U slučaju većeg broja ciljeva agenti trenirani nad 5000 i 15000 iteracija ne postižu dobre rezultate, jer je vjerojatnost generiranja jednostavnih, trivijalnih slučajeva manja.

Također, vidljivo je kako je prosječno vrijeme za posjećivanje jednog cilja manje s porastom broja ciljeva koje agent treba uzastopno posjetiti, što znači da agent uspješno uzima u obzir moguće buduće lokacije cilja. Na primjer, ako se trenutni cilj nalazi uz donji(-y) rub okoline, vjerojatnost je veća da slijedeći cilj bude prema gore u odnosu na trenutni, dok takve veze nije moguće uspostavljati u slučaju jednog cilja.

Testiranjem agenta zapisuje se *.csv* datoteka se koristi za generiranje animacije u programskom paketu Blender. Blender je alat za 3D modeliranje, koji pruža mogućnost automatiziranja jednog dijela procesa koristeći Python skripte. Skripta za svaki *frame* animacije postavlja lokaciju i rotaciju vozila, te objekta koji predstavlja trenutni cilj. Primjer *renderane* animacije dostupan je u repozitoriju.

5. ZAKLJUČAK

Pojačano učenje je dobar postupak u slučajevima gdje okolinu (*engl. environment*) možemo promatrati kao crnu kutiju (*engl. black box*). Primjenjiv je u slučajevima gdje je moguće dozvoliti agentu da izvodi kritične greške, poput računalnih simulacija(npr. Igara, sustava upravljanja i slično). Pri stvaranju takvih simulacija bitno je da one što bolje predstavljaju stvarne uvjete. Dobro ga je primijeniti u slučajevima gdje nije moguće analitički doći do dobrog rješenja, ali je poznat željeni cilj/stanje sustava prema kojemu agent treba težiti.

Najveći problem kod algoritama pojačanog učenja je oblikovanje funkcije nagrade. U slučajevima kada za svaki korak ne dajemo nagradu agent će rijetko doći do zadovoljavajućeg stanja, te će zbog toga konvergencija biti gotovo nemoguća. Pri oblikovanju funkcije nagrade također je bitno da agenta ne ograničimo previše, jer će se u tom slučaju prilagoditi funkciji nagrade i neće naći neko drugo (možda optimalnije) rješenje. Velik problem predstavljaju i rubni slučajevi (*engl. edge cases*), koji će se rijetko pojavljivati pri treniranju, ali kasnije svojom pojavom mogu dovesti do grešaka.

Zbog toga je još uvijek dobro koristiti konvencionalne algoritme. Iako konvencionalni algoritmi možda nisu optimalna rješenja, nude prednosti pri obradi rubnih slučajeva, lakšu proširivost i često bolju brzinu izvođenja, pa su zbog toga još uvijek dobar izbor. Te je algoritme moguće proširiti algoritmima pojačanog učenja kako bi se pronašla moguća druga rješenja.

6. LITERATURA

[1] S. Guadarrama, A. Korattikara, O. Ramirez, P. Castro, E. Holly, S. Fishman, K. Wang, E. Gonina, N. Wu, E. Kokiopoulou, L. Sbaiz, J. Smith, G. Bartók, J. Berent, C. Harris, V. Vanhoucke i E. Brevdo, »TF-Agents: A library for Reinforcement Learning in Tensorflow,« 2018. [Mrežno]. Available: https://github.com/tensorflow/agents. [Pokušaj pristupa January 2020].

[2] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang i W. Zaremba, *OpenAI Gym,* 2016.

[3] V. Mnih, K. Kavukcuoglu, D. Silver i e. al., »Human-level control through deep reinforcement learning,« *Nature,* svez. 518, pp. 529-533, 2015.

[4] »Simple 2D car steering physics in games,« 2010. [Mrežno]. Available: http://engineeringdotnet.blogspot.com/2010/04/simple-2d-car-physics-in-games.html. [Pokušaj pristupa January 2020].