Primjena algoritma potpomognutog učenja u razvoju računalnih igara

Kinkela, Dominik

Undergraduate thesis / Završni rad

2021

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Rijeka / Sveučilište u Rijeci**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:195:089317

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-05-06**

Sveučilište u Rijeci
Fakultet informatike i digitalnih tehnologija

Repository / Repozitorij:

Repository of the University of Rijeka, Faculty of Informatics and Digital Technologies - INFORI Repository

zir.nsk.hr

DIGITALNA KNJIŽNICA

DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

Sveučilište u Rijeci – Odjel za informatiku

Preddiplomski jednopredmetni studij informatike

Dominik Kinkela

Primjena algoritma potpomognutog učenja u razvoju računalnih igara

Završni rad

Mentor: izv. prof. dr. sc., Marina Ivašić-Kos

Rijeka, rujan 2021.

ODJEL ZA INFORMATIKU
SVEUČILIŠTE U RIJECI

Rijeka, 17.02.2021.

Zadatak za završni rad

Pristupnik: **Dominik Kinkela**

Naziv završnog rada: **Primjena algoritma potpomognutog učenja u razvoju računalnih igara**

Naziv završnog rada na eng. jeziku: **Application of reinforcement learning algorithm in computer game development**

Sadržaj zadatka:

Proučiti metode potpomognutog učenja koje se koriste prilikom razvoja računalnih igara. Objasniti osnovne pojmove kao što su funkcije vrijednosti, korisnost učenja, stopa učenja, stopa istraživanja i slično. Opisati različite metode potpomognutom učenja s naglaskom na Q-učenje.

Napraviti implementaciju algoritma Q-učenja u programskom jeziku Python uz pomoć OpenAI-jevog okruženja Gym. Opisati postupak izrade aplikacije i obrazložiti parametre i funkcije iz biblioteke Gym koje se koriste te ostale resurse korištene u razvoju aplikacije. Demonstrirati funkcionalnost Q-učenja na više jednostavnih primjera.

Mentor

Izv. prof. dr. sc. Marina Ivašić-Kos

Type text here

Voditelj za završne radove

Doc. dr. sc. Miran Pobar

Zadatak preuzet: 17.02.2021.

Sadržaj

1. Zadatak ..... **Error! Bookmark not defined.**
2. Sažetak 5
3. Potpomognuto učenje 6
   3.1. Općenito 6
   3.2. Povijest 7
   3.3. Notacija 8
   3.4. Epizode 8
   3.5. Sniženi povrat 9
   3.6. Politike 9
   3.7. Funkcije vrijednosti 10
   3.8. Optimalnost 11
   3.9. Učenje na i bez temelja modela 11
4. Q-Učenje 12
   4.1. Q-vrijednost 12
   4.2. Q-tablica 12
   4.3. Duljina učenja 13
   4.4. Istraživanje naspram iskorištavanja 13
   4.5. Stopa učenja 13
   4.6. Konačna formula 14
5. Implementacija 15
   5.1. Cilj 15
   5.2. Okolina 15
   5.3. Prostor stanja 16
   5.4. Prostor akcija 17
   5.5. Nagrade 17
   5.6. Kod 18
   5.7. Ispis 21
   5.8. Grafovi 22
   5.9. Usporedba parametara 24
   5.10. Prikaz igre 25
6. Zaključak 35
7. Popis slika 36

8. Literatura i izvori 37

2. Sažetak

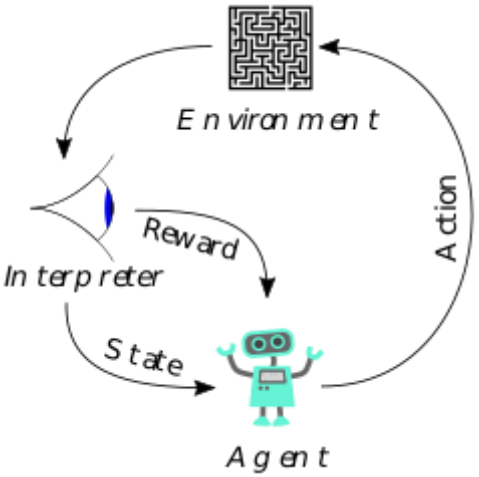
U ovome radu opisano je što je to potpomognuto učenje te kako ono funkcionira. Opisano je njegovo nastajanje te spomenute njemu prethodne, a slične tehnike. Opisani su pojmovi koji se u kontekstu potpomognutog učenja koriste i način kako se koriste. Spomenut ćemo više vrsta potpomognutog učenja te se fokusirati na q-učenje, kojeg ćemo još detaljnije opisati te ga kasnije implementirati i demonstrirati njegovu funkcionalnost na primjeru. Opisati ćemo algoritme i obrazložiti određene parametre i formule koje se koriste. Implementaciju ćemo napisati u programskom jeziku Python uz pomoć OpenAI-jevog okruženja Gym.

Ključne riječi: potpomognuto učenje, q-učenje, python, funkcije vrijednosti, stopa učenja, stopa sniženja, stopa istraživanja, OpenAI, Gym, učenje bez temelja modela

3. Potpomognuto učenje

3.1. Općenito

Potpomognuto učenje (engl. reinforcement learning) je jedna od tri vrste strojnog učenja (engl. machine learning) [1] koja omogućava agentu da uči postići cilj u potencijalno nepoznatom i kompleksnom okruženju [2]. Kao i kod nama poznatog treniranja pasa, učenje se vrši na temelju pozitivnog i negativnog podražaja za obavljenu akciju. To naravno zahtjeva od programera da odredi koja akcija je dobra i koliko je dobra, a koja je loša i koliko loša kako bi agent na temelju pokušaja i pogrešaka naučio optimalan postupak kojem je cilj maksimiziranja ukupne nagrade, Slika 1.



Slika 1: potpomognuto učenje [1]

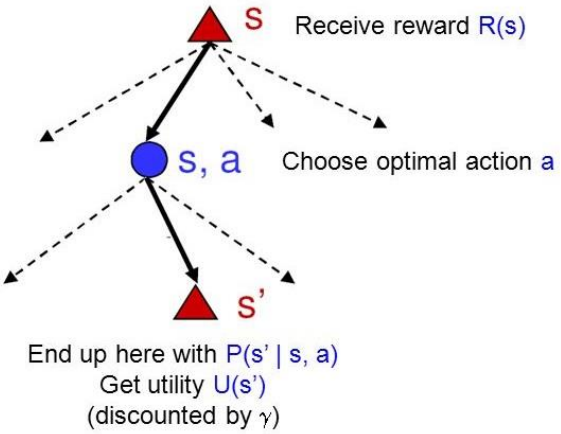
Učenje se najčešće vrši kroz niz iterativnih epizoda između kojih se okruženje resetira kako bi agent pokušao ponovo naći najbolji način izvršenja neke akcije. Nakon nekog vremena iako programer agentu nije dao konkretne instrukcije ili opis okruženja, agent sam u početku posve nasumičnim akcijama postaje sve bolji u onome što radi. Cijeli proces učenja ovisi o kompleksnosti okruženja, broju mogućih stanja i akcija, te naravno performansama računala na kojem agent trenira.

3.2. Povijest

Krajem pedesetih godina 20. stoljeća američki matematičar Richard Ernest Bellman pokušavao je riješiti takozvanu teoriju optimalne kontrole. To je grana matematičke optimizacije koja pokušava minimizirati ponašanje nekog sustava kroz neko vrijeme. Naposljetku je zajedno sa svojim kolegama došao do rješenja kojeg je nazvao Bellmanova jednadžba, koja je u današnje vrijeme opće poznata i korištena za brojne stvari poput dinamičkog programiranja [3].

Bellmanova jednadžba opisuje korisnu vrijednost nekog koraka (stanja) prilikom rješavanju problema s obzirom na vrijednost prošlih akcija i potencijalnu vrijednost sljedećih. Na taj način problem se rastavi na više jednostavnijih potproblema. Na slici 2. prikazana je Bellmanova jednadžba te kako se kroz svaki korak bira optimalna akcija dok se ne dođe do cilja.

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) U(s')$$



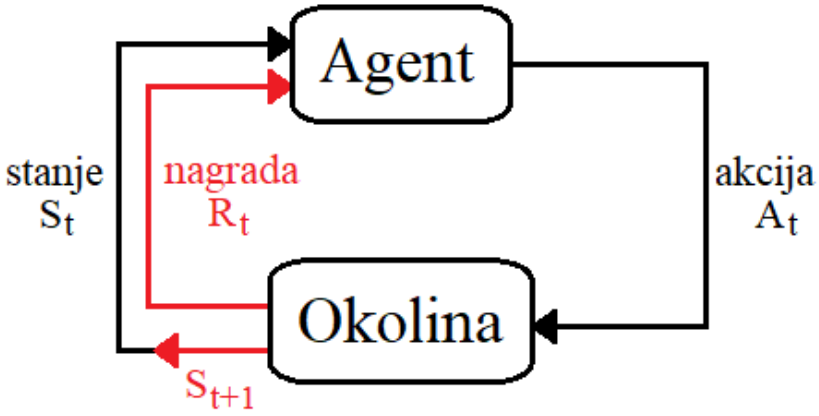
Slika 2: bellmanova jednadžba i rekurzivni odnos između korisnosti uzastopnih stanja [4]

S druge strane američki psiholog Edward Thorndike razvio je psihološki princip zvan „the law of effect“. Eksperimentirajući i promatrajući ponašanje mačke u kutiji zaključio je da „podražaji koji proizvode zadovoljavajući efekt u određenoj situaciji imaju veću vjerojatnost da će se ponoviti u toj situaciji, a podražaji koji proizvode negativan efekt imaju manju vjerojatnost da će se ponoviti u toj situaciji [5]. Danas nam se to čini trivijalno, no informatičar Chris Watkins je 1989. iskoristio upravo tu ideju kako bi ju spojio sa prije navedenom Bellmanovom jednadžbom, s ciljem da potencijalno napravi novi algoritam učenja na način da minimizira ponašanje sustava koje uči kroz metodu pokušaja i pogrešaka.

3.3. Notacija

Potpomognuto učenje koristi Markov sustav odlučivanja (engl. Markov decision process - MDP) kao notaciju. MDP je stohastički proces upravljanja diskretnim vremenom, koji pruža matematički okvir za modeliranje donošenja odluka u situacijama gdje su ishodi djelomično nasumični, a djelomično pod utjecajem donositelja odluka. Tog donositelja odluka nazivamo agentom. On vrši interakcije sa okolinom u kojoj se nalazi na temelju danih informacija. Imamo skup stanja **S**, skup akcija **A** i skup nagradi **R**.

Svakog vremenskog koraka $t = 0, 1, 2, \ldots$ agent dobije reprezentaciju stanja okoline $S_t \in \boldsymbol{S}$ na temelju kojeg vrši akciju $A_t \in \boldsymbol{A}$. To nazivamo stanje-akcija par. Nadalje pri sljedećem koraku agent dobije nagradu $R_{t+1} \in \boldsymbol{R}$ za prijašnji postupak (S_t, A_t).



Slika 3: stanje, akcija, nagrada

3.4. Epizode

U potpomognutom učenju agent uči iz koraka u korak. No postoje definirane akcije koje terminiraju vršenje programa. Kada agent izvrši akciju koja vodi u terminalno stanje, okolina se resetira te agent nastavi s treniranjem.

Sekvencu (inter)akcija, sa određenom duljinom, simulacije od početnog stanja do terminalnog stanja nazivamo epizodom. Duljina treniranja se često izražava u epizodama.

3.5. Sniženi povrat

U slučaju da se interakcije agenta i okoliša ne mogu lako rastaviti na epizode, moramo preraditi način na koji agentu dajemo nagrade. S beskonačnim brojem koraka $\boldsymbol{T} = \infty$ možemo potencijalno imati beskonačno nagrada, stoga uvodimo sniženi povrat. Za razliku od očekivanog povrata koji je samo zbroj kumulativnih nagrada, sniženi povrat u jednadžbu dodaje stopu sniženja γ, broj između 0 i 1 koji se koristi u formuli na sljedeći način.

Očekivani povrat: $G_t = R_{t+1} + R_{t+2} + \cdots + R_T$

Sniženi povrat: $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$

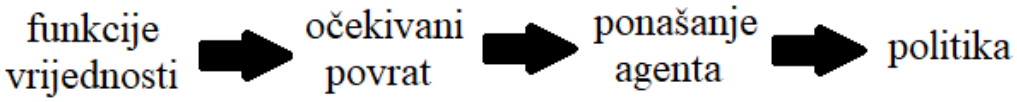
Sniženi povrat tjera agenta da sve manje cijeni kasnije nagrade i daje veći prioritet rješenjima koji zahtijevaju manje koraka s obzirom da će nagrade kasnijih akcija biti sve manje.

3.6. Politike

Politika je funkcija koja definira ponašanje agenta u određenom stanju. Označava se sa simbolom π. Kažemo da agent prati politiku na način da u koraku t, $\pi(a|s)$ prezentira vjerojatnost vršenja akcije a u stanju s. Za svako stanje s, π je vjerojatnost distribucije po $a \in \boldsymbol{A}(s)$.

3.7. Funkcije vrijednosti

Funkcije vrijednosti su funkcije stanja ili stanje-akcija para koje procjenjuju koliko je agentu povoljno da ode u neko stanje ili obavi neku akciju. Ta vrijednost je izražena u obliku očekivanog povrata. S obzirom da je očekivani povrat baziran na tome kako se agent ponaša, vidimo da funkcije vrijednosti ovise o politici agenta.



Slika 4: odnos funkcija vrijednosti i politika

Razlikujemo dvije vrste funkcija vrijednosti, stanje-vrijednost i akcija-vrijednost. Stanje-vrijednost funkcija nam kaže koliko je dobro određeno stanje za agenta koji prati politiku π i označava se sa slovom *v*.

$$v_\pi(s) = E^1{}_\pi[G_t|S_t = s] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s]$$

Akcija-vrijednost funkcija nam kaže koliko je dobro da agent izvrši određenu akciju u nekom stanju dok prati politiku π i označava se sa slovom q.

$$q_\pi(s,a) = E_\pi[G_t|S_t = s, A_t = a] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a]$$

Termin q_π naziva se q-funkcijom dok se povratna vrijednost funkcije za bilo koji stanje-akcija par naziva q-vrijednost gdje u oba slučaja slovo q prezentira kvalitetu istih.

[1] E_π označuje očekivani povrat ako agent prati politiku π, počevši iz stanja *s*

3.8. Optimalnost

S obzirom da je potpomognutom učenju cilj postići što bolje rezultate traže se optimalne verzije prethodno navedenih funkcija. U nastavku će biti opisan postupak određivanja optimalnih funkcija.

Politika π je bolja ili jednaka politici π' ako i samo ako je očekivani povrat politike π veći ili jednak politici π' za sva stanja. Politika koja nije lošija od nijedne druge politike naziva se optimalna politika.

$$\pi \geq \pi' \; akko \; v_\pi(s) \geq v_{\pi'}(s) \; \forall \; s \in S$$

Kao što je prethodno navedeno funkcija vrijednosti ovisi o politici. Optimalna funkcija se označava sa indeksom *. v_* vraća najveći mogući očekivani povrat od ikoje politike za sva stanja. q_* vraća najveći mogući povrat od ikoje politike za svaki stanje-akcija par.

$$v_*(s) = \max_\pi v_\pi(s) \qquad\qquad q_*(s,a) = \max_\pi q_\pi(s,a)$$

Ako uključimo Bellmanovu jednadžbu u prethodnu, dobit ćemo formulu zvanu Bellmanova jednadžba optimalnosti.

$$\boxed{q_*(s,a) = E[R_{t+1} + \gamma \max_{a'} q_*(s',a')]}$$

q_* mora zadovoljavati ovu jednadžbu. Ona nam kaže da za bilo koji stanje-akcija par u koraku t, očekivani povrat u stanju s nakon vršenja akcije a će biti jednak zbroju akcije a u stanju s (R_{t+1}) i maksimalnom sniženom povratu sljedećeg optimalnog stanje-akcija para.

3.9. Učenje na i bez temelja modela

Potpomognuto učenje može donositi odluke na jedan od dva načina. Na temelju modela ili bez temelja modela. Na temelju modela, agent iz svojih opažanja nauči model funkcioniranja okoline, a zatim pomoć tog modela planira rješenje tako da postavi pitanje što će se dogoditi ako napravi neku akciju. No agent ne mora imati model okoline da bi mogao naći dobru politiku. Jedan od najpoznatijih primjera učenja bez temelja modela je q-učenje.

4. Q-Učenje

Q-učenje je algoritam potpomognutog učenja. S obzirom da je q-učenje bazirano na učenju bez modela, agent počinje učiti bez ikakvih pretpostavki ili znanja, već uči samo na temelju pokušaja i pogrešaka.

4.1. Q-vrijednost[2]

Algoritam iterativno ažurira q-vrijednosti svakog stanje-akcija para koristeći prethodno spomenutu Bellmanovu jednadžbu sve dok trenutna q-funkcija ne dođe do optimalne q-funkcije q_*. Ovaj proces se zove iteracija vrijednosti, te je demonstrirana u kasnijem poglavlju.

4.2. Q-tablica

Sve se te q-vrijednosti spremaju u tablicu zvanu q-tablica, kako bi im agent mogao lako pristupati. Tablica se sastoji od konačnog broja redova i stupaca. Broj redova reprezentira broj mogućih stanja okoline, na način da svako jedinstveno stanje okoline zauzme jedan redak tablice. Broj stupaca reprezentira broj mogućih akcija koje agent može izvršiti u svakom stanju. S obzirom da agent nema prijašnjih znanja, sva polja tablice početno su inicijalizirana na nulu. Potom učeći kroz mnogi broj koraka i epizoda agent popunjava tablicu novim ažuriranim q-vrijednostima tako da vrijednost R_{t+1} pohrani u red reprezentiran stanjem s_t i stupac reprezentiran prethodno izvršenom akcijom a_t.

Stanja	Akcije			
	Akcija 1	Akcija 2	Akcija 3	Akcija 4
Stanje 1	0	0	0	0
Stanje 2	-3	0	4	0
Stanje 3	0	0	0	0
Stanje 4	0	-15	0	0
Stanje 5	0	0	0	0
Stanje 6	0	0	-10	7

Slika 5: q-tablica

Nakon nekog vremena kako tablica postaje sve više puna agent može očitati vrijednosti pojedinih akcija u određenom stanju i prema njima „bolje“ reagirati.

[2] definicija u poglavlju 3.7. Funkcije vrijednosti

4.3. Duljina učenja

Kao što je prethodno spomenuto duljina učenja se često izražava u broju epizoda koje će agent odigrati. Taj broj je proizvoljan jer agent uvijek može učiti još više ili manje. Na temelju performansi agenta nakon učenja možemo odlučiti hoćemo li produžiti učenje ili ga možemo skratiti bez velikog utjecaja na rezultat. Također možemo ograničiti broj koraka unutar epizode kako agent ne bi zauvijek zapeo u epizodi bez dolaska u terminalno stanje.

4.4. Istraživanje naspram iskorištavanja

Kako agent ne bi svaki puta izabrao istu akciju koja je trenutno u određenom stanju najpovoljnija, te time propustio potencijalno bolju akciju, mora se uvesti pojam istraživanja. Napraviti ćemo to pomoću takozvane epsilon pohlepne strategije. U toj strategiji simbolom ε definirana je stopa istraživanja, inicijalizirana na 1. Ona predstavlja vjerojatnost da agent zanemari najpovoljniju q-vrijednost iz q-tablice i odabere nešto drugo, odnosno da radije istraži nego iskoristi okolinu. S početnom vrijednosti 1, postoji 100%-tna šansa da će agent prvotno istražiti okolinu. Ta se vrijednost s vremenom eksponencijalno smanjuje kako agent sve više uči. Ova strategija osigurava da agent isproba i druge akcije te ne propusti niti jednu akciju i stanje zbog toga što trenutno ima nešto bolje.

Način na koji agent odabere hoće li istražiti ili iskoristiti okolinu u svakom koraku je taj da se generira nasumični broj između 0 i 1 te usporedi sa vrijednosti epsilona. Ukoliko je nasumično generirani broj veći od vrijednosti epsilona, agent će iskoristiti okolinu, i na isti princip vrijedi suprotno.

```
threshold = np.random.uniform(0, 1)
if threshold > exploration_rate:
    action = np.argmax(q_table[state])  # iskorištavanje
else:
    action = env.action_space.sample()  # istraživanje
```

Slika 6: odabir istraživanja ili iskorištavanja

4.5. Stopa učenja

Uspoređujući staru i novu q-vrijednost agent ne smije uvijek zapisati veću, jer se može desiti da će ju kasnije morati smanjiti zbog novo otkrivene bolje politike. Stoga agent koristi staru i novu vrijednost u sprezi. Koliku važnosti pridodaje kojoj, određuje stopa učenja. To je broj označen sa **α**, u vrijednosti između 0 i 1. Što je veća stopa učenja, agent će brže mijenjati q-vrijednosti.

4.6. Konačna formula

Naposljetku dolazimo do formule za izračunavanje nove q-vrijednosti. Uzevši sve prethodno u obzir formula je sljedeća.

$$q^{new}(s,a) = (1-\alpha)q(s,a) \; + \; \alpha[R_{t+1} + \gamma \max_{a'} q(s',a')]$$

Nova q-vrijednost je ponderirana suma prošle q-vrijednosti i naučene q-vrijednosti. Staru vrijednost stanje-akcija para pomnožimo sa (1-α), dok novu vrijednost množimo sa α.

Ovisno o planiranoj duljini učenja možemo podešavati parametre stope učenja i stope sniženja u našu korist. Ukoliko planiramo kraće učenje možemo postaviti veću stopu učenja te smanjiti stopu sniženja kako bi agent mogao više naučiti u kraćem vremenu.

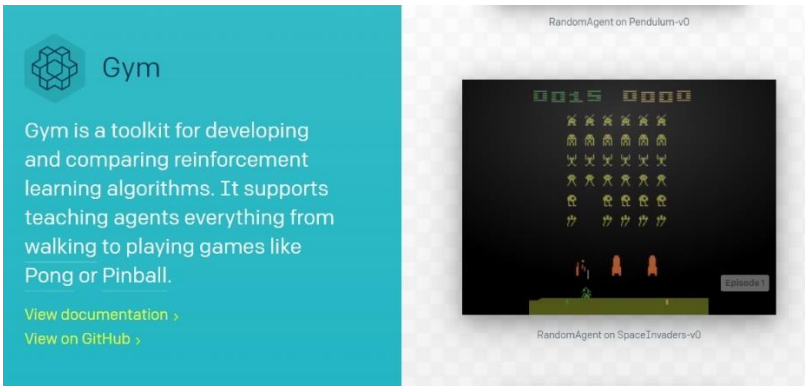
5. Implementacija

5.1. Cilj

Q-učenje će biti pokazano na primjeru vozača taksija kojemu je cilj preuzeti putnika i ispustiti ga na njegovom odredištu u što kraćem roku. Agent započne epizodu sa taksijem na jednom od polja postavljene mreže. Postoje 4 moguće lokacije koje mogu biti mjesto preuzimanja ili isporuke putnika. Alternativno putnik može u početku epizode već biti u taksiju.

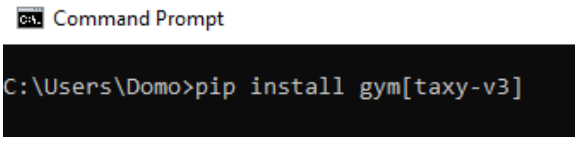
5.2. Okolina

Kao okolinu koristiti ćemo OpenAI-jevu Gym okolinu [6]. Gym je alat za razvoj i usporedbu algoritama potpomognutog učenja. Dok se može postaviti vlastita okolina, za ovaj primjer koristit ćemo gym „Taxi-v3“ zbog jednostavnosti pristupa te ljepšeg prikaza izlaza programa.



Slika 7: gym alat [6]

Počet moramo sa instalacijom gym-a.



Slika 8: instaliranje gym-a

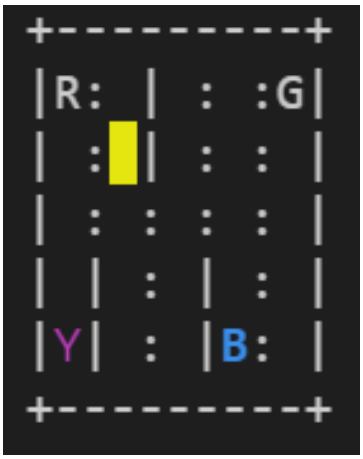
Potom ga moramo uvesti u program te postaviti okolinu na onu koju želimo. Također će nam trebati i par drugih biblioteka pa ćemo uvesti i njih.

```
import gym
import numpy as np
import matplotlib.pyplot as plt
import time

env = gym.make('Taxi-v3')
```

Slika 9: uvoženje biblioteka

5.3. Prostor stanja

Prostor stanja je ukupni broj mogućih stanja. Mreža kretanja je kvadratna i veličine 5 x 5. Dakle taksi ima 25 različitih početnih pozicija. Postoji 5 lokacija gdje može biti početno mjesto putnika (4 lokacije na mreži i u taksiju) te 4 lokacije destinacije putnika (4 lokacije na mreži).



Slika 10: primjer stanja

R, G, Y, B – lokacije

Plava boja – lokacija putnika

Ljubičasta boja – destinacija putnika

Žuti pravokutnik – lokacija taksija

Zeleni pravokutnik – lokacija taksija koji je pokupio putnika

| – zid kroz kojeg taksi ne može proći

: – služi kao vizualni pomagač za odjeljenje

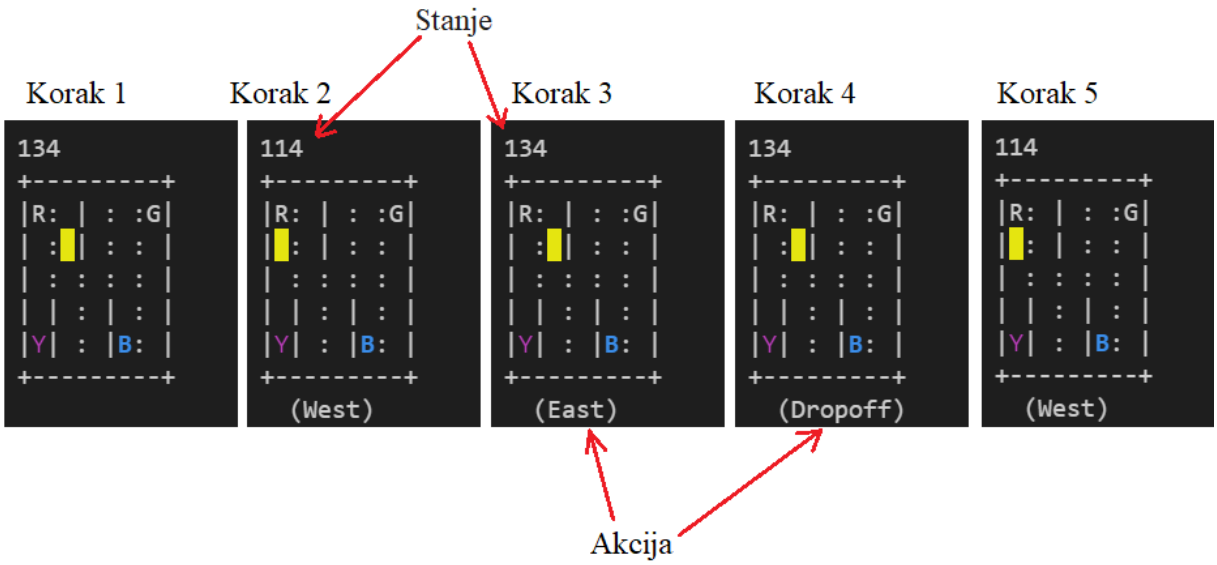
Prazno polje – cesta po kojoj taksi može voziti

Zaključujemo da postoji 5 x 5 x 5 x 4 = 500 mogućih stanja.

5.4. Prostor akcija

Prostor akcija je ukupni broj mogućih akcija u svakom stanju. U svakom koraku agent ima 6 različitih akcija:

1. pomaknuti se južno
2. pomaknuti se sjeverno
3. pomaknuti se istočno
4. pomaknuti se zapadno
5. pokupiti putnika
6. iskrcati putnika



Slika 11: nasumične akcije agenta

5.5. Nagrade

Nagrade koje će agent dobivati po koraku ovisno o akciji koju izvrši su sljedeće. Svakim korakom nagrada je -1 bod, za svako neispravno preuzimanje ili iskrcavanje nagrada je -10 bodova, te za uspješno iskrcavanje putnika nagrada je +20 bodova.

5.6. Kod

U nastavku ćemo proći i opisati ostatak koda na temelju prethodno naučenih koncepata. Potom ćemo pokazati rad programa i analizirati neke dobivene rezultate ovisno različitim vrijednostima parametara.

Inicijalizirat ćemo q-tablicu sa vrijednostima 0, veličine prostor stanja (500) x prostor akcija (6).

```
state_space_size = env.observation_space.n
action_space_size = env.action_space.n

q_table = np.zeros((state_space_size, action_space_size))
```

Slika 12: inicijalizacija q-tablice

Odlučimo se za dugo i točno učenje. Odaberemo relativno velik broj epizoda/igra koje će agent odigrati. S obzirom na velik broj epizoda stopa učenja neće imati velik utjecaj na konačni rezultat dokle god je u rasponu od 0.2 do 0.9. Postavimo ju 0.75 jer u par testova daje konstantno najveće rezultate. Postavimo stopu sniženja na visoku vrijednost jer planiramo dugo učiti.

```
games_to_play = 10000

learning_rate = 0.75
discount_rate = 0.95
```

Slika 13: postavljanje broja epizoda, stopa učenja i sniženog povrata

Potom moramo postaviti stopu istraživanja. Kao što smo prethodno rekli postavimo ju na najveću vrijednost, odnosno na 1 jer želimo da agent u početku dok još ništa ne zna više istražuje nego iskorištava okolinu. Postavimo maksimalnu i minimalnu vrijednost epsilona jer će nam trebati u kasnijoj formuli. Također definirajmo stopu smanjenja.

```
exploration_rate = 1
max_exploration_rate = 1
min_exploration_rate = 0.01
exploration_rate_decay = 0.0005
```

Slika 14: postavljanje vrijednosti epsilona

Navedene varijable iskoristit ćemo u sljedećoj formuli koja će nam omogućiti računanje nove vrijednosti epsilona pri svakom novom koraku. Ovom formulom uz povećanje broja gotovih epizoda vrijednost epsilona će biti funkcija eksponencijalnog smanjenja.

```
exploration_rate = min_exploration_rate + \
                   (max_exploration_rate - min_exploration_rate) * \
                    np.exp(-exploration_rate_decay * episode)
```

Slika 15: formula promjene epsilona

Glavna for petlja iterira po epizodama do ukupnog broja epizoda koje će se odigrati kojeg smo odabrali. Na početku svake epizode resetira se okolina te se postavi nagrada epizode na 0. Nagrada epizode služi samo za ispis informacija te nije ključna za vršenje učenja ili performanse agenta. Isto vrijedi i za varijablu stope istraživanja.

Zatim ulazimo u petlju koja nije ograničena brojem koraka već samo terminalnim stanjem. Unutar petlje pri svakom koraku nalazi se odluka istraživanja naspram iskorištavanja koju smo već spomenuli. Nakon toga dolazimo do glavnog djela algoritma. Poduzmemo korak s prethodno određenom akcijom te nam to vrati 4 vrijednosti, redom: novo stanje, nagradu, podatak jeli epizoda završila te dijagnostičke informacije korisne za otklanjanje pogrešaka. Potom izračunamo novu q-vrijednost uz pomoć već poznate formule.

$$q^{new}(s,a) = (1-\alpha)q(s,a) \; + \; \alpha[R_{t+1} + \gamma \max_{a'} q(s',a')]$$

Naposljetku postavimo trenutno stanje na novo stanje. Ukoliko je podatak je li epizoda završila potvrdan, while petlja će biti prekinuta. Zatim će se osvježiti vrijednost epsilona kao što je prethodno objašnjeno i nadodati nagrade epizode u listu svih nagrada koja također služi samo za ispis.

```
for episode in range(games_to_play):
    state = env.reset()
    episode_reward = 0

    exploration_rates.append(exploration_rate)

    while True:
        threshold = np.random.uniform(0, 1)
        if threshold > exploration_rate:
            action = np.argmax(q_table[state])  # iskorištavanje
        else:
            action = env.action_space.sample()  # istraživanje

        new_state, reward, done, info = env.step(action)

        q_table[state, action] = (1 - learning_rate) * q_table[state, action] +
        learning_rate * (reward + discount_rate * np.max(q_table[new_state]))

        state = new_state
        episode_reward += reward

        if done:
            break

    exploration_rate = min_exploration_rate + (max_exploration_rate -
    min_exploration_rate) * np.exp(-exploration_rate_decay * episode)

    rewards.append(episode_reward)
```

Slika 16: glavna for petlja

5.7. Ispis

Kako bi procijenili kvalitetu agentovog znanja, kao ispis koristit ćemo sumu nagrada u određenom broju epizoda. Ovisno o preciznosti koju želimo možemo postaviti varijablu koja to određuje na 10, 100, 1000 za ispis nagrada po ekvivalentnom broju epizoda ili pak staviti neku drugu vrijednost kako ne bi imali ispis.

```
if nagradePo == 1000:
    #Rewards per 1000 episodes
    rewards_per_thousand_episodes = np.split(np.array(rewards), games_to_play /
    1000)
    count = 1000

    print('\n--- Average reward per thousand episodes ---\n')
    for r in rewards_per_thousand_episodes:
        print(f'{count}: {sum(r/1000)}')
        count += 1000

elif nagradePo == 100:
    #Rewards per 100 episodes
    rewards_per_hundred_episodes = np.split(np.array(rewards), games_to_play / 100)
    count = 100

    print('\n--- Average reward per hundred episodes ---\n')
    for r in rewards_per_hundred_episodes:
        print(f'{count}: {sum(r/100)}')
        count += 100

elif nagradePo == 10:
    #Rewards per 10 episodes
    rewards_per_ten_episodes = np.split(np.array(rewards), games_to_play / 10)
    count = 10

    print('\n--- Average reward per ten episodes ---\n')
    for r in rewards_per_ten_episodes:
        print(f'{count}: {sum(r/10)}')
        count += 10
```

Slika 17: ispis nagrada po broju epizoda

Za primjer ispisa dobrog rezultata učenja uzeli smo parametre:

Broj igri za odigrat = 10000

Stopa učenja = 0.75

Stopa sniženja = 0.95

te dobili sljedeći rezultat. Sve blizu nuli i preko je već relativno dobar rezultat kao što ćemo kasnije vidjeti.

```
--- Average reward per thousand episodes ---

1000: -320.1540000000003
2000: -43.8680000000001
3000: -14.32599999999998
4000: -2.770000000000002
5000: 1.7559999999999958
7000: 5.310999999999982
8000: 6.274999999999979
9000: 6.460999999999973
10000: 7.114999999999963
```

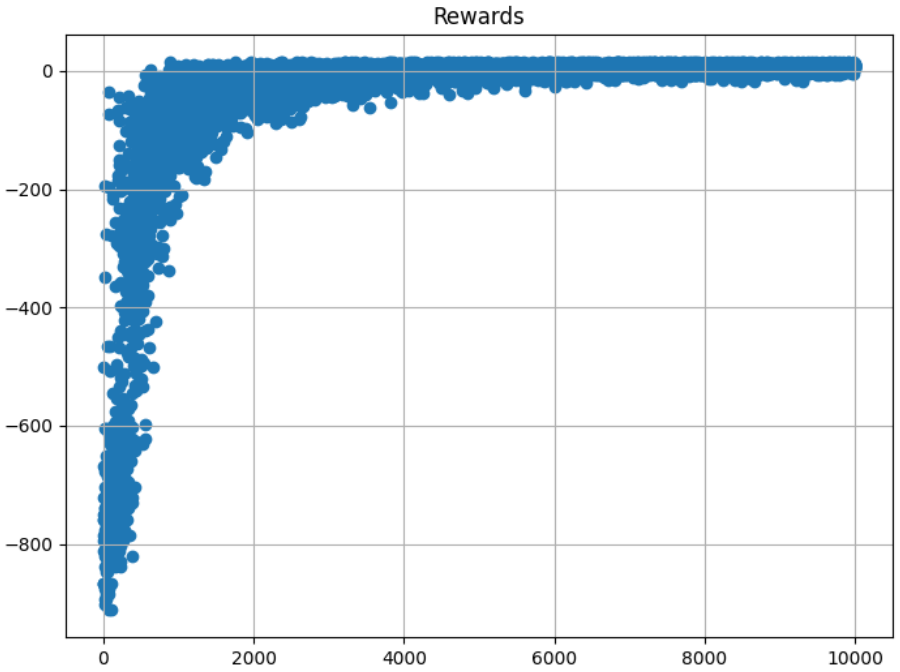
Slika 18: dobar rezultat

5.8. Grafovi

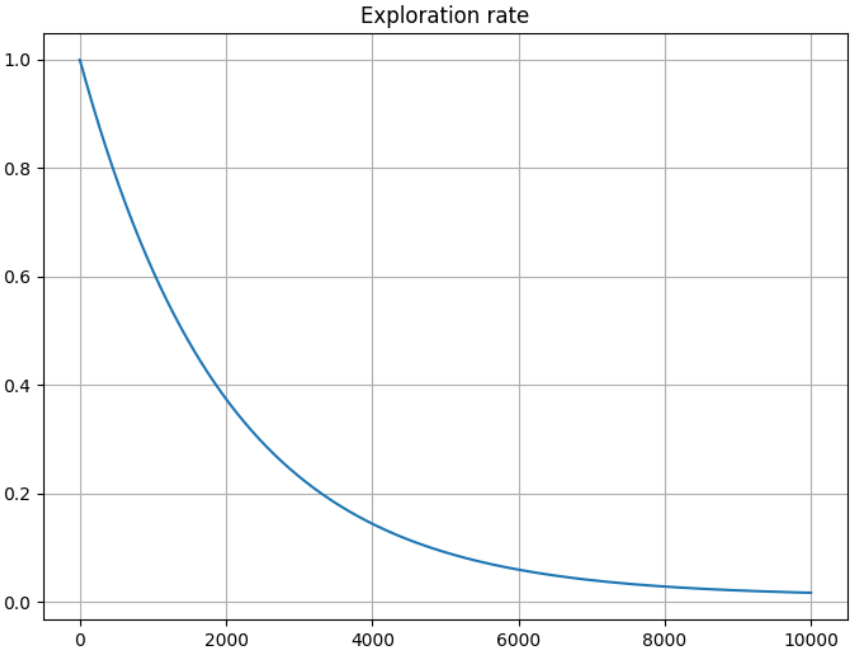
Osim prethodnog ispisa iskorištena je biblioteka matplotlib za prikaz vrijednosti nagrada i epsilona na grafu po broju epizoda, sa istim parametrima kao prije.

```
plt.figure(figsize = (18, 6))

plt.subplot(121)
plt.scatter(range(games_to_play), rewards)
plt.title('Rewards')
plt.grid()

plt.subplot(122)
plt.plot(range(games_to_play), exploration_rates)
plt.title('Exploration rate')
plt.grid()

plt.show()
```

Slika 19: matplotlib kod



Slika 20: graf nagrada



Slika 21: graf stope istraživanja

5.9. Usporedba parametara

Ukoliko probamo sa istim parametrima trenirati agenta puno kraće vrijeme, poput 100 epizoda vidjet ćemo da rezultati neće biti najbolji što mogu biti. Prebacit ćemo ispis po broju epizoda na 10.

```
--- Average reward per ten episodes ---

10: -715.1
20: -777.8
30: -767.9000000000001
40: -812.0
50: -761.6
60: -776.0000000000001
70: -728.9
80: -770.6
90: -725.9999999999999
100: -767.0
```

Stopa učenja = 0.75

Stopa sniženja = 0.95

Slika 22: učenje sa lošim parametrima

```
--- Average reward per ten episodes ---

10: -750.7
20: -758.0
30: -788.6
40: -759.8000000000001
50: -808.3999999999999
60: -779.6000000000001
70: -727.0000000000001
80: -754.4000000000001
90: -678.4
100: -669.6
```

Stopa učenja = 0.92

Stopa sniženja = 0.07

Slika 23: učenje sa dobrim parametrima

Kao što vidimo iz rezultata za kraće treniranje nam je isplativije povećati stopu učenja kako bi agent brže mijenjao vrijednosti, te smanjiti stopu sniženja kako bi agent više vrijednosti pridodao vrijednostima neposrednih akcijama.

5.10. Prikaz igre

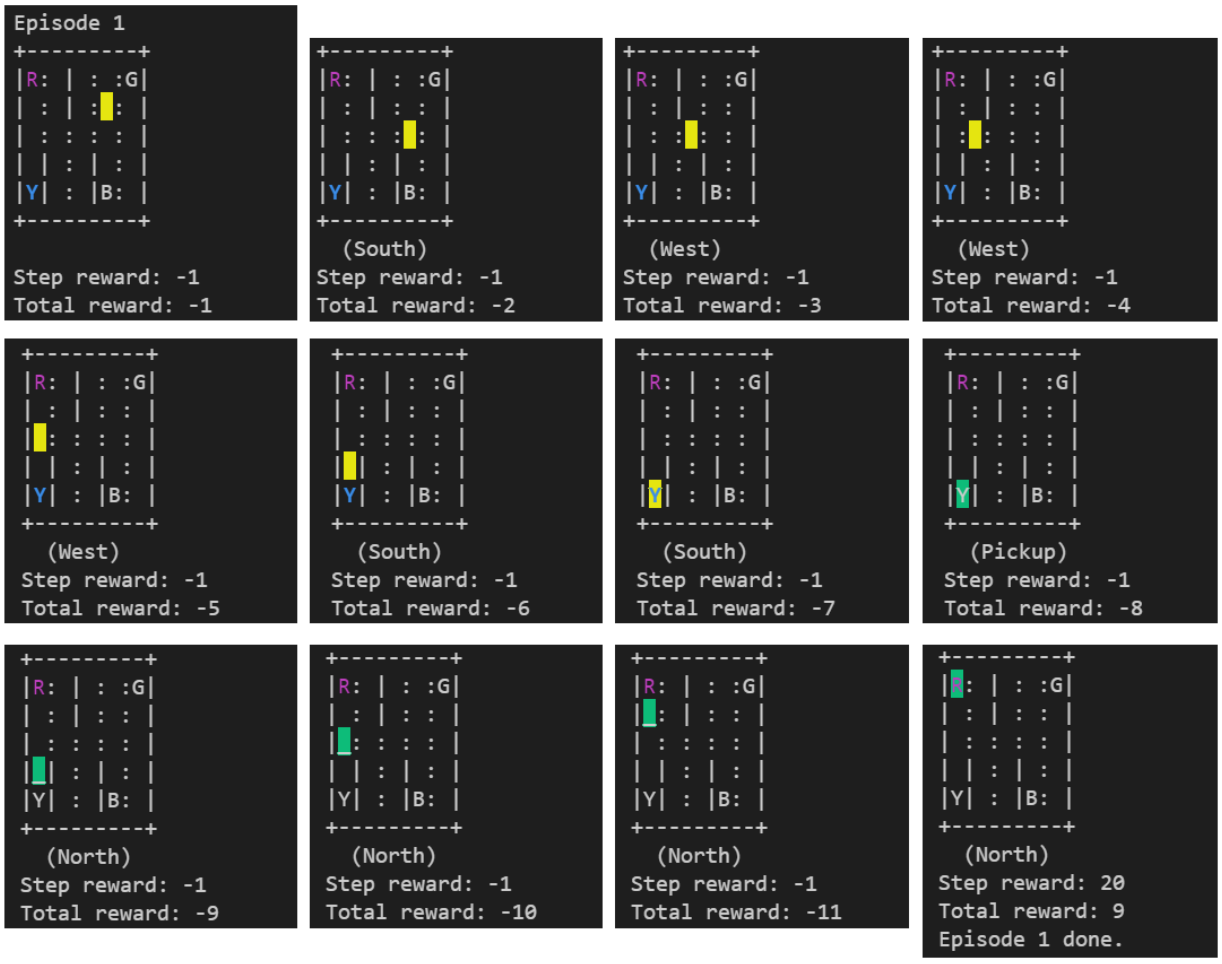
Nakon što je agent završio učenje uz pomoću već popunjene q-tablice može zaigrati još par epizoda te nam vizualno demonstrirati svaki korak.

U vanjskoj for petlji postavimo broj epizoda koje želimo vidjeti. Ponovo se na početku svake epizode okolina mora resetirati te ostale varijable rade isto što i u prethodnom primjeru. Unutarnja petlja je ograničena na 18 koraka jer to je maksimalni broj koraka koji agentu trebaju da obavi epizodu sa najvećim mogućim brojem koraka, optimalno. Ukoliko uspije imat ćemo ispis uspjeha, a ukoliko mu bude trebalo više od toga ispis neuspjeha. Vremena spavanja su dodana zbog lakšeg pregleda pri ispisu u terminalu. Također ćemo pri svakom koraku imati ispis trenutne i sveukupne nagrade.

```
for i in range(2):
    total_reward = 0
    state = env.reset()
    print(f'Episode {i+1}')

    for j in range(18):
        env.render()
        action = np.argmax(q_table[state])
        state, reward, done, debug = env.step(action)
        total_reward += reward
        print(f'Step reward: {reward}')
        print(f'Total reward: {total_reward}')

        time.sleep(0.2)

        if done:
            print(f'Episode {i+1} done.')
            time.sleep(1)
            break
    if not done:
        print(f'Episode {i+1} failed.')
        time.sleep(1)
```

Slika 24: kod za prikazivanje igre

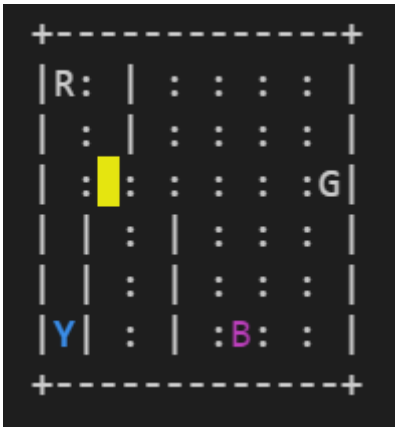
Ovdje vidimo jednu epizodu gdje agent vrši optimalne akcije te sa početne pozicije pomiče taksi do putnika, ukrcava ga te onda nastavi do destinacije gdje ga i iskrca. Na kraju vidimo da je epizoda uspješno završila.



Slika 25: prikaz epizode

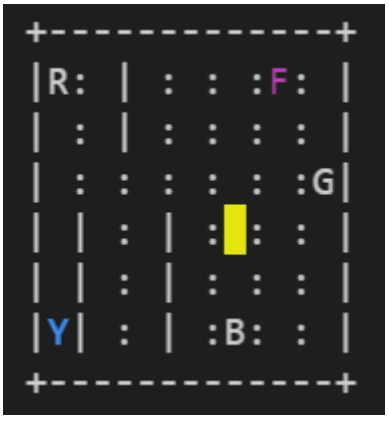
Kada agent nije dovoljno istreniran onda radi nasumične akcije kao što se vidi na slici 11. Razinu agentovog znanja možemo provjeriti pomoću ispisa rezultata.

Ovo je relativno mala mreža te su rezultati, neovisno o početnoj konfiguraciji lokacija putnika, destinacije i taksija, poprilično isti po pitanju broja koraka jer agent može vrlo brzo kroz mali broj epizoda doći do optimalne politike. Stoga ćemo modificirati mrežu koju smo do sada koristili tako što ćemo joj dodati još 2 stupca i 1 red te prerazmjestiti lokacije.



Slika 26: povećana mreža

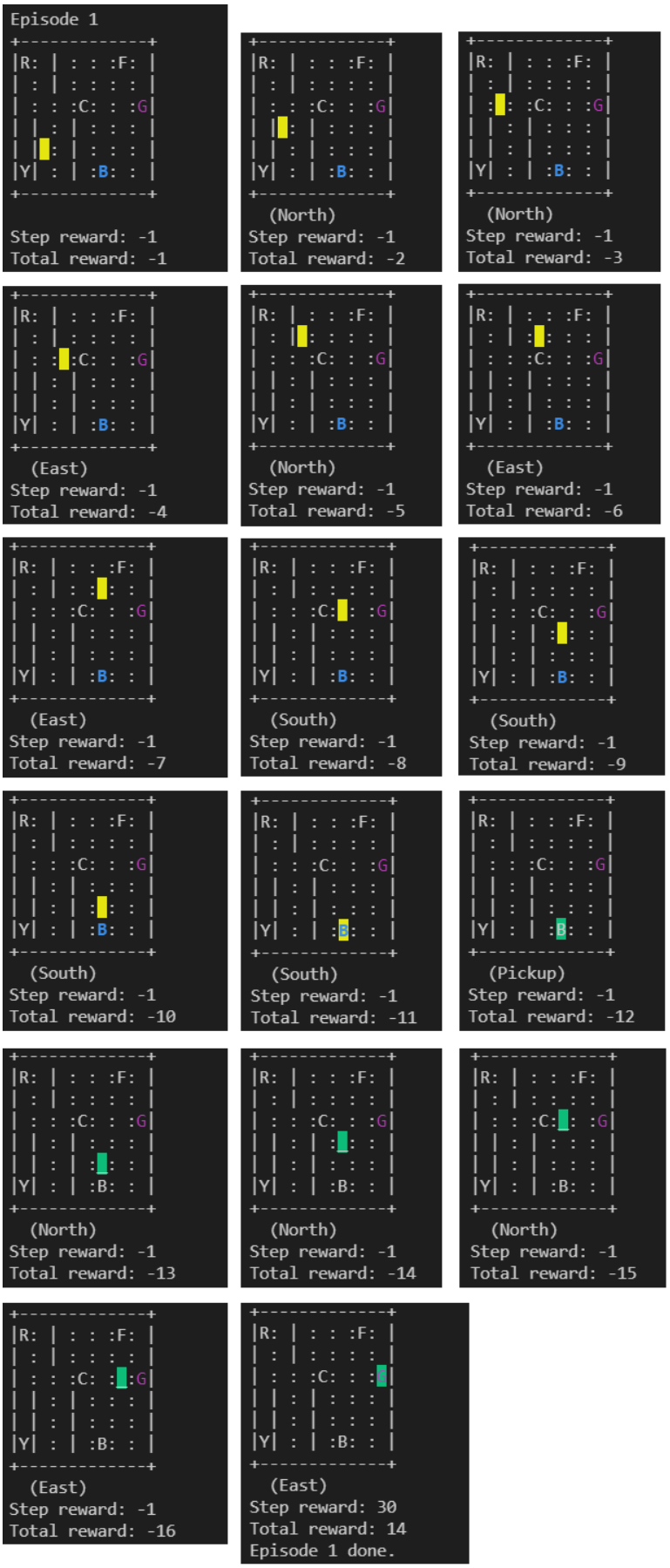
Ovime smo promijenili ukupni broj mogućih stanja sa 500, na 840 (6 x 7 x 5 x 4). Dok je osnovnoj mreži da istrenira 10000 epizoda prosječno trebalo 6.46 sekundi, novoj proširenoj mreži prosječno treba 8.72 sekunde. To je porast od 35%. Razlika je toliko velika zbog dodatnih akcija koje agent vrši unutar epizode s obzirom da ga nismo ograničili brojem koraka. Dodajemo još jednu lokaciju F, što će povećati broj stanja na 1260 (6 x 7 x 6 x 5).



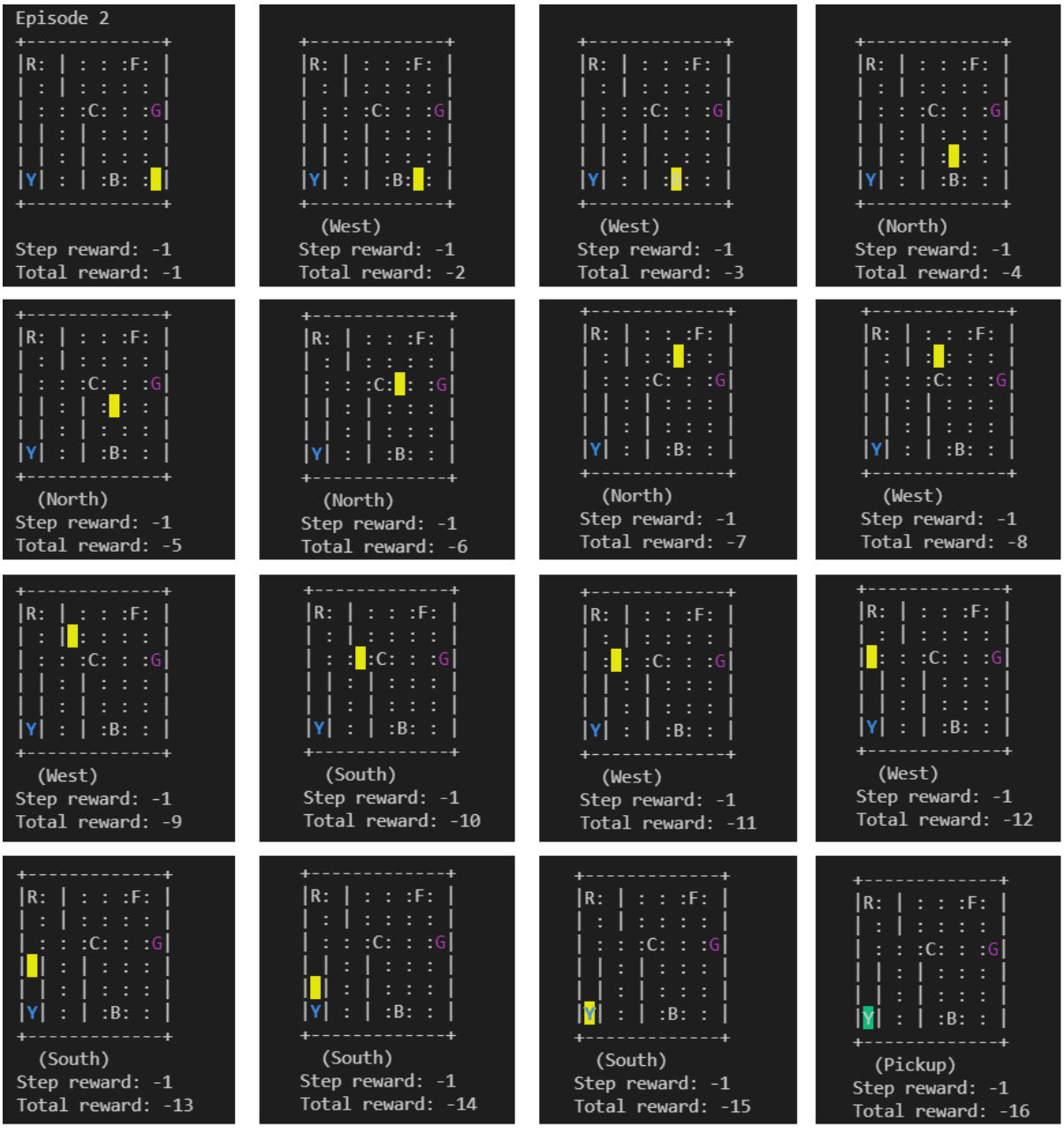
Slika 27: dodana nova lokacija

Za treniranje sa novom lokacijom trebalo samo 4% više vremena, sa prosjekom od 9.14 sekundi. Iako se broj stanja opet znatno povećao vrijeme izvođenja programa nije. Agent trenira na bazi epizoda te mu zato broj stanja u kojima će on učiti ne igra veliku ulogu po pitanju vremena izvođenja programa, već u kvaliteti učenja. Time on pri kraju učenja u prosjeku ima 8% lošije rezultate za isti broj epizoda.

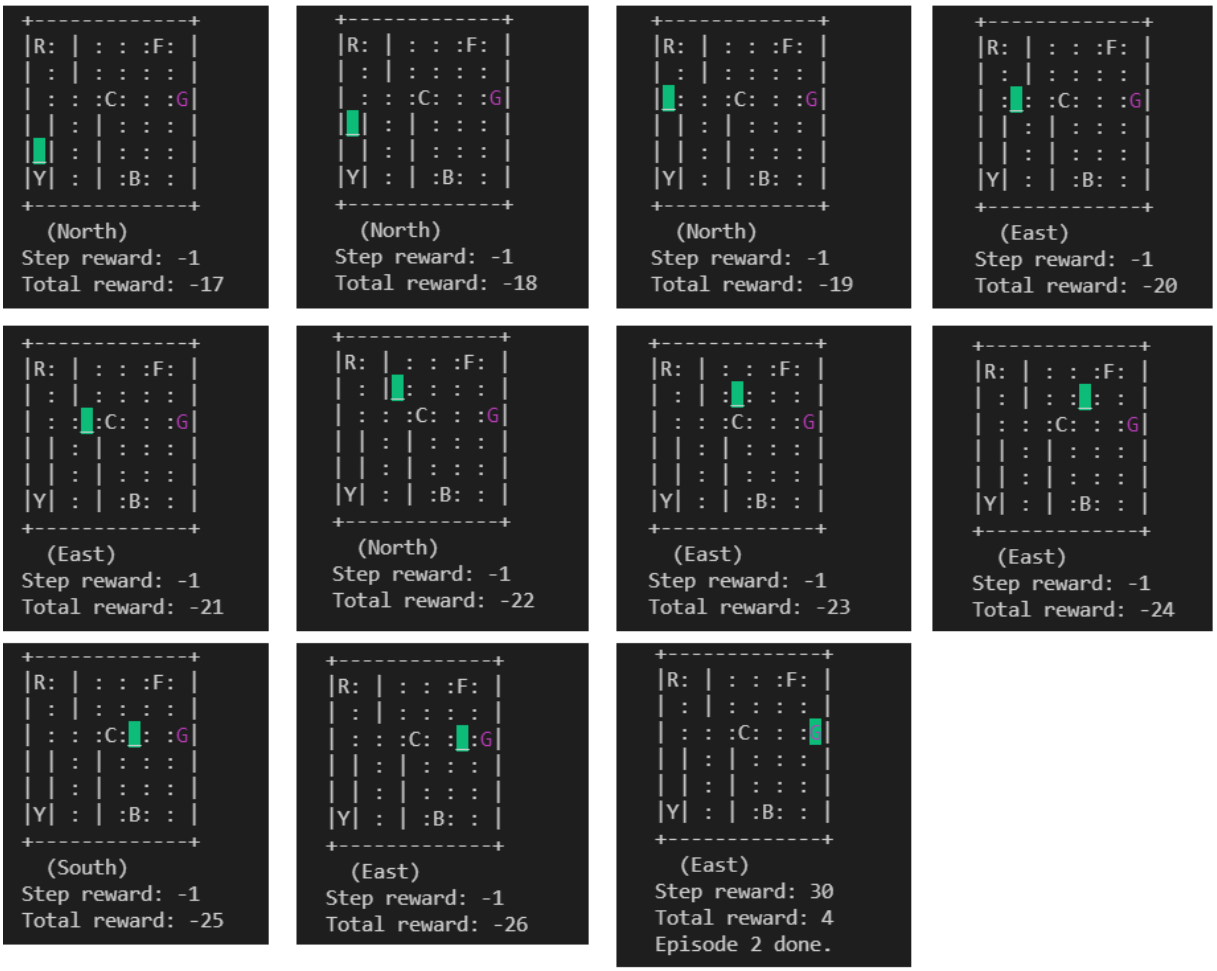
Naposljetku dodati ćemo još jednu lokaciju C, no nju ne tretiramo kao i ostale, već ona predstavljati naplatu cestarine za taksi ukoliko on prođe kroz tu lokaciju. Broj stanja ostaje isti, nagrada za naplatu cestarine je -10 bodova, a finalna nagrada je povećana na 30 bodova kako bi zbog većeg ukupnog broja koraka rezultat ostao pozitivan u svim situacijama gdje agent koristi optimalnu politiku. Kao primjer slijede dvije epizode različitih duljina.



Slika 28: kratka vožnja



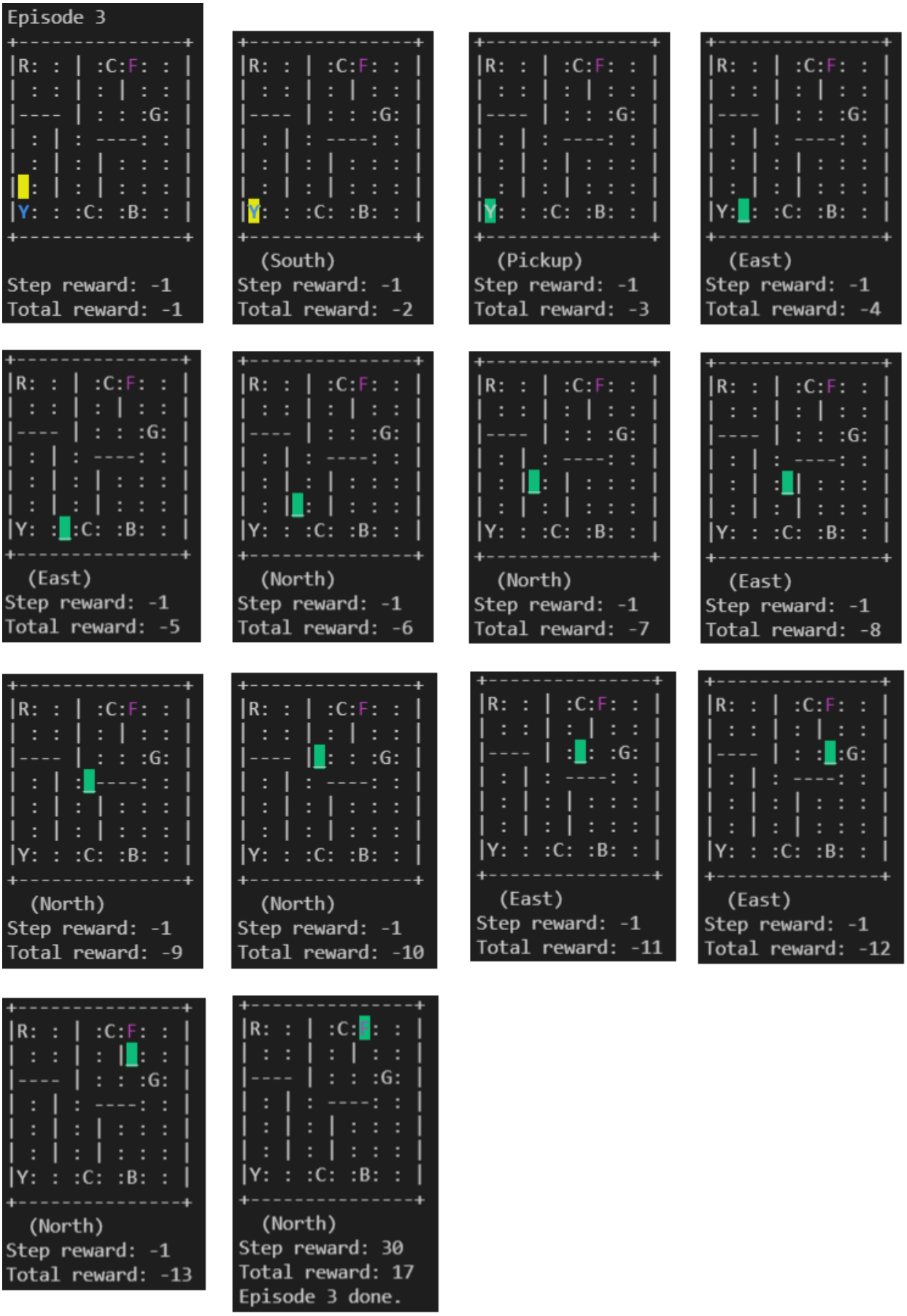
Slika 29: duga vožnja (1. dio)



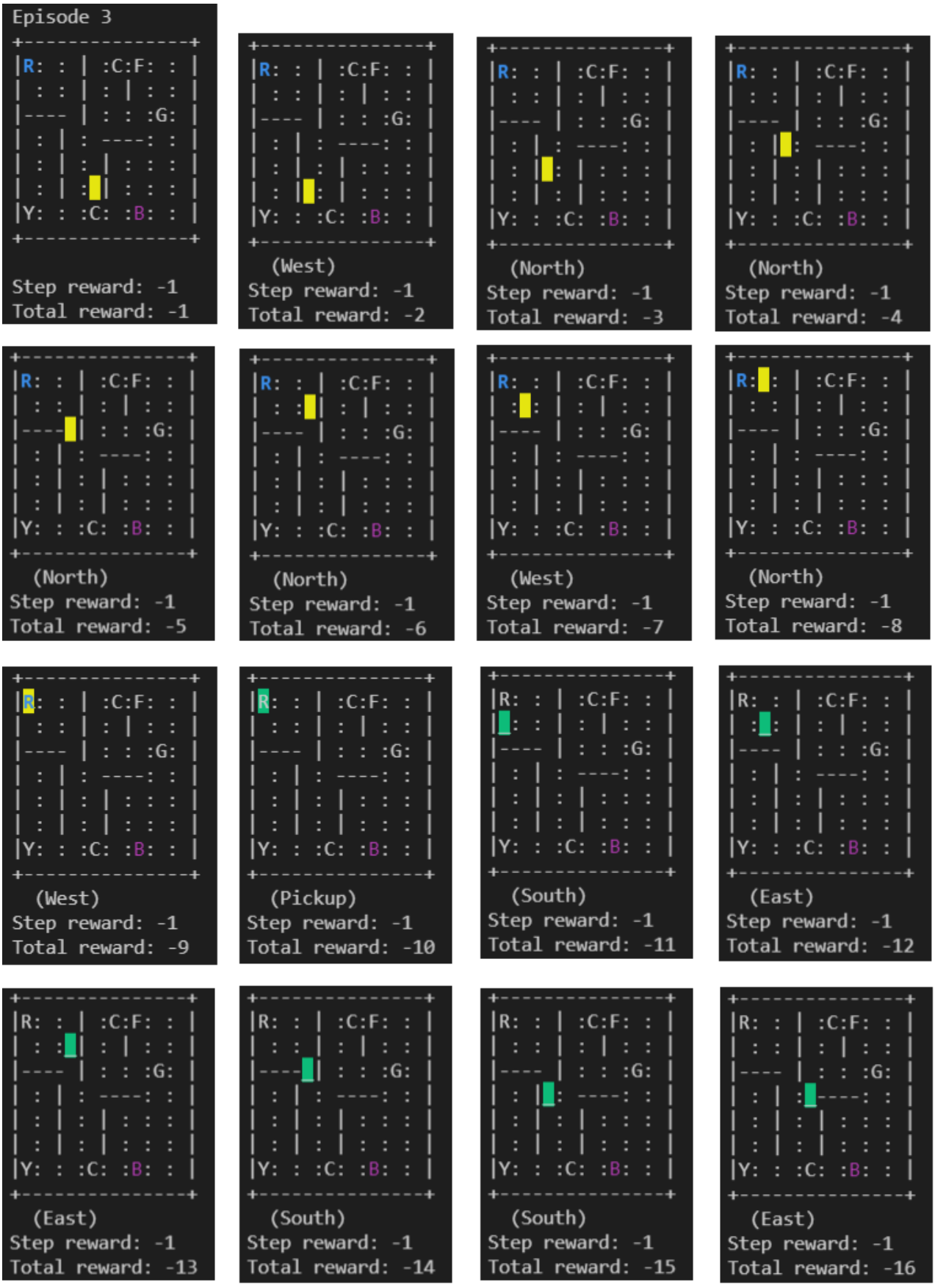
Slika 30: duga vožnja (2. dio)

Vidimo kako agent u okolini gdje je dodana naplata cestarine radije odabire ići dužim putem okolo te lokacije nego „platiti cestarinu“. On to čini u oba smjera s obzirom da je povoljnije napraviti par koraka okolo te lokacije i dobiti par negativnih bodova nego proći kroz C i dobiti 10 negativnih bodova. Prosječno vrijeme treniranja 10000 epizoda iznosi 9.93 sekundi. To je porast od 9% naspram okoline bez carine te se može pripisati dodatnom koraku provjere kojeg agent mora napraviti da vidi je li u planiranom smjeru kretanja lokacija naplate cestarine.

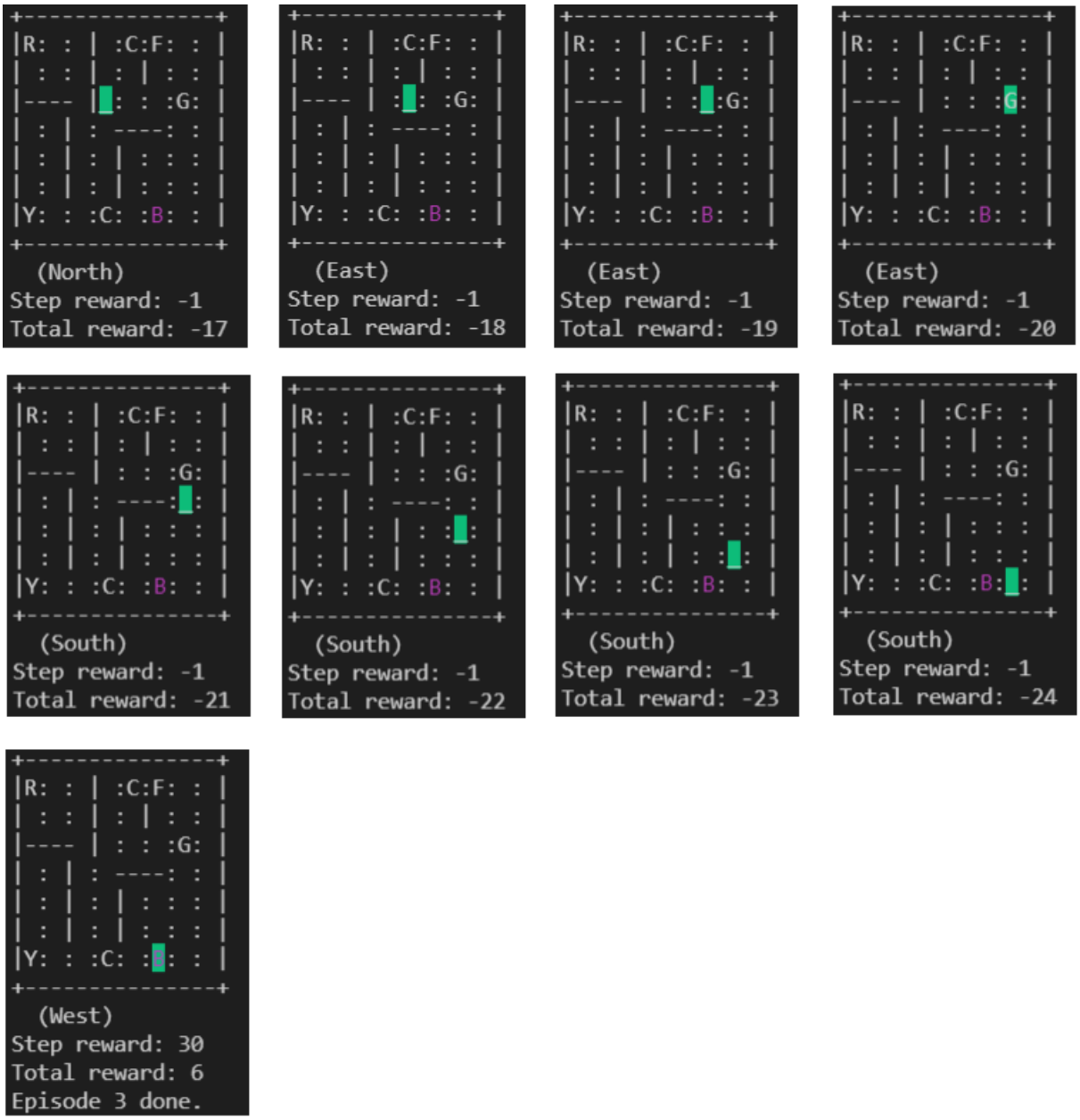
Još malo zakomplicirajmo mrežu, proširimo ju za još jedan redak i stupac, prerazmjestimo lokacije i zidove, dodajmo horizontalne zidove te još jednu lokaciju carine. Na ovoj kompliciranijoj mreži prikazane su još dvije odigrane epizode.



Slika 31: kompliciranija mreža



Slika 32: kompliciranija mreža 2 (1. dio)



Slika 33: kompliciranija mreža 2 (2. dio)

6. Zaključak

Potpomognuto učenje je vrlo interesantna metoda strojnog učenja koja ima široke primjene, a da u isto vrijeme nije pretjerano apstraktna da je netko sa malom količinom učenja ne može shvatiti te primijeniti na svom primjeru.

Pri razvoju novi računalnih igara znatno skraćuje vrijeme razvoja osnovnog bot-a, a može se i nadograditi u kompliciraniji i napredniji oblik.

U kombinaciji sa drugim tehnologijama poput vizualne detekcije objekata potpomognuto učenje može imati mnogo znatnije primjene za razliku od samostalne.

7. Popis slika

Slika 1: potpomognuto učenje [1] 6
Slika 2: bellmanova jednadžba i rekurzivni odnos između korisnosti uzastopnih stanja [4] 7
Slika 3: stanje, akcija, nagrada 8
Slika 4: odnos funkcija vrijednosti i politika 10
Slika 5: q-tablica 12
Slika 6: odabir istraživanja ili iskorištavanja 13
Slika 7: gym alat [6] 15
Slika 8: instaliranje gym-a 15
Slika 9: uvoženje biblioteka 16
Slika 10: primjer stanja 16
Slika 11: nasumične akcije agenta 17
Slika 12: inicijalizacija q-tablice 18
Slika 13: postavljanje broja epizoda, stopa učenja i sniženog povrata 18
Slika 14: postavljanje vrijednosti epsilona 18
Slika 15: formula promjene epsilona 19
Slika 16: glavna for petlja 20
Slika 17: ispis nagrada po broju epizoda 21
Slika 18: dobar rezultat 22
Slika 19: matplotlib kod 22
Slika 20: graf nagrada 23
Slika 21: graf stope istraživanja 23
Slika 22: učenje sa lošim parametrima 24
Slika 23: učenje sa dobrim parametrima 24
Slika 24: kod za prikazivanje igre 25
Slika 25: prikaz epizode 26
Slika 26: povećana mreža 27
Slika 27: dodana nova lokacija 27
Slika 28: kratka vožnja 29
Slika 29: duga vožnja (1. dio) 30
Slika 30: duga vožnja (2. dio) 31
Slika 31: kompliciranija mreža 32
Slika 32: kompliciranija mreža 2 (1. dio) 33
Slika 33: kompliciranija mreža 2 (2. dio) 34

8. Literatura i izvori

[1] https://en.wikipedia.org/wiki/Reinforcement_learning

[2] https://deepsense.ai/what-is-reinforcement-learning-the-complete-guide/

[3] https://en.wikipedia.org/wiki/Bellman_equation

[4] https://slidetodoc.com/markov-decision-processes-chapter-17-image-source-p/

[5] https://en.wikipedia.org/wiki/Law_of_effect

[6] https://gym.openai.com/

[7] https://www.youtube.com/watch?v=A5eihauRQvo&ab_channel=SirajRaval

[8] https://www.quora.com/What-is-the-difference-between-model-based-and-model-free-reinforcement-learning

[9] https://www.quora.com/What-does-the-term-%E2%80%9Cepisode%E2%80%9D-mean-in-the-context-of-reinforcement-learning-RL

[10] https://deeplizard.com/learn/video/rP4oEpQbDm4

[11] https://deeplizard.com/learn/video/nyjbcRQ-uQ8

[12] https://en.wikipedia.org/wiki/Markov_decision_process

[13] http://www.incompleteideas.net/book/ebook/node34.html

[14] https://bair.berkeley.edu/blog/2019/12/12/mbpo/

[15] https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/