Projektni prijedlog iz kolegija Strojno učenje, ak.god. 2019./2020.

MALI MATE

KARLO LONČAREVIĆ, MATEJA SAVIĆ, LOVRO SINDIČIĆ

1. Opis problema

Pac-Man je jedna od najpoznatijih *vintage* video igrica koju je razvila japanska kompanija Namco 1980. godine. Svoju popularnost iz ranih dana održala je i dan danas te se sve češće javljaju zanimljivi gifovi i memeovi temeljeni na Pac-Manu što dodatno govori i o tome da su mlađe generacije upoznate s ovom video igricom.
Kao budući nastavnici informatike u toku svojeg studija susreli smo se s metodičkim pristupom programiranju robota[1]. Naišli smo na problem nedostataka raznih aktivnosti i zato smo si postavili pitanje možemo li koristeći dosadašnje znanje napraviti vlastitu verziju Pac-Mana pomoću robota.
Ovaj projekt realizirat ćemo pomoću robota Codey Rocky s određenim odmakom od tradicionalnih pravila igre s obzirom na to da nam konstrukcija robota ne omogućava neke od realizacija koje su moguće u video verziji igrice (jedenje igrača i sl.), ali i za njih ćemo pronaći jednako zanimljiva i kreativna rješenja.

2. Opis robota

Codey Rocky proizvod je privatne kineske firme Makeblock iz 2017.
Robot je namijenjen pomoći djeci koja uče osnove programiranja i umjetne inteligencije. Sastavljen je od dva odvojiva dijela. Glava imena Codey je programirljivi kontroler koji sadrži više od 10 elektroničkih modula. Dok je tijelo odnosno, Rocky, vozilo koje može izbjeći prepreke, prepoznati boje i slijediti crte. Za svoje kretanje koristiti dva DC Geared motora, a moguće ga je programirati koristeći se blokovskim programiranjem u programu Mblock ili pisanjem koda u programskom jeziku Python.
Codey Rocky kombinira hardver sa softverom, omogućujući djeci da uče o programiranju dok se igraju i stvaraju.
U eksploratornoj analizi smo pokazali da roboti neće davati iste rezultate za isti skup naredbi. Tako primjerice mogu prijeći različite udaljenosti ili se mogu okrenuti za različite kutove. Prema našem mišljenju nekih od mogućih razloga za to su prljavština na motorima i razina baterije. To može biti veliki problem prilikom

[1]Prema novome kurikulumu [1] za OŠ neki od ishoda koji to obuhvaćaju su C.1.3, B.2.1, A.4.2, B.5.2, B.6.2.

Slika 1. robot Codey Rocky

demonstracije. Zato ako nam trenutna epidemiološka situacija to dopusti proveli bi testiranje na sva tri robota u istim uvjetima kako bi pokušali ustanoviti koje od značajka utječu na ishod naredbi. Ako to nažalost neće biti moguće provesti, ideja je jednostavno napraviti kalibraciju jednog Codey Rockya u odnosu na druge.
Dakle, za naredbu `rocky.forward(50, 1)` prvi će Codey prijeći put duljine x_1, a drugi y_1. Također, za naredbu `rocky.forward(100, 1)` prvi će Codey prijeći put duljine x_2, a drugi y_2. Povlačenjem pravca kroz dvije točke (x_1, y_1) i (x_2, y_2) dobivamo funkciju $f(x) = kx + l$. Na taj način, ako je veza stvarno linearna, možemo izračunati duljinu puta koju bi prešao drugi Codey, ako je prvi prešao put duljine x.

3. Upravljanje robotom

Budući da je plan na računalu naučiti kontrolnu strategiju, a tek kasnije na naučenom modelu provesti demonstraciju pomoću robota. Sada ćemo opisati kako ćemo to napraviti. Kao što smo u ranijem odlomku naveli robote je moguće programirati u Pythonu, pa za početak navedimo popis biblioteka koje ćemo koristiti.

```
1 import codey, event, time
```

Listing 1. Python biblioteke koje korisiti Codey Rocky

S ugrađenim Wi-Fi-jem[2], Codey Rocky brzo se povezuje na internet, ostvarujući svoju IoT[3] funkcionalnost. Tako da robot ima mogućnost spajanja na WiFi mrežu i prilikom spajanja na nju dobije vlastitu IP adresu.

```
1 def on_start():
2   codey.wifi.start('ime_WiFi_mreze', 'lozinka_mreze', codey.wifi.STA)
3   while not codey.wifi.is_connected():
4     pass
```

Listing 2. spajanje Codey Rockya na WiFi mrežu

[2]Wireless Fidelity
[3]Internet of things

Zato ćemo komunikaciju računala i Codey Rockya provesti preko Wi-Fi mreže. Na računalu ćemo pokrenuti lokalni server pomoću programa XAMPP. Svaki od robota imati će svoj direktoriji u kojem će biti jedna datoteka index.php i čitati svoju datoteku i ovisno o broju koji pročita provest će određenu akciju. Jedini dodatni posao simulacije koju bi vrtili na računalu bio bi da piše u navedene datoteke pozicije na koje se mora kretatati svaki od robota. Zbog vremena potrebnog da se podatak zapiše na disk i pročita s njega novi korak u igri provodili svake 4 sekunde. Dolje navodimo dio koda koji bi se izvršavao u robotima.

```
<?
    echo("2")
?>
```

Listing 3. index.php

```
while True:
  request_url = 'http://lovro-pc/mate/index.php'
  resp = requests.get(request_url)
  text = resp.text
  broj = int(text)
  if broj == 0:
    rocky.turn_left_by_degree(90)
  if broj == 1:
    rocky.turn_right_by_degree(90)
  if broj == 2:
    rocky.forward(50, 1)
  if broj == 3:
    rocky.backward(50, 1)
  if broj == 4:
    codey.speaker.play_melody('warning.wav')
          codey.emotion.dizzy()
          rocky.stop()
          codey.stop_all_scripts()
```

Listing 4. kretanje Codey Rockya

Kako je jedenje Pacmana jedno od pooznatijih obilježja igre potrudili smo se za to pronaći dostojnu zamjenu. Svaki Codey Rocky ima senzor trešenje. Tako da kada se lovci sudare s robotom koji glumi Pacmana, odnosno kada ga uhvate, pokrenut će se njegov senzor trešenje i Codey Rocky će ispusiti zvuk gašenja i ugasiti se. Dolje navodimo kod kako bi to proveli.

```
def on_shaked():
    codey.emotion.dizzy()
    rocky.stop()
    codey.stop_all_scripts()
```

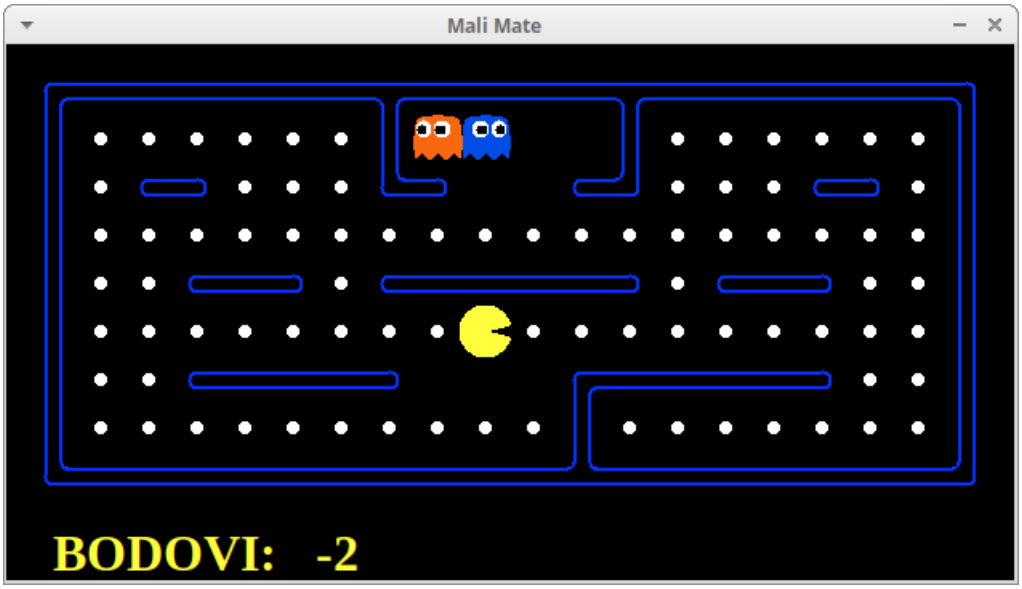
Listing 5. kraj igre

4. Pravila i okruženje

Sve metode učenja s podrškom koja ćemo implementirati biti će bazirane na emulatoru za Pacman igru izrađenog na sveučilištu Berkeley u Kaliforniji [4]. Pravila biti će jednaka originalnoj igri Pac-Man, izuzev toga što će agent imati samo jedan život i neće biti debelih bijelih točkica. Prisjetimo se ako ih agent pojede tada neometano može pojesti lovce. Za navedenu promjenu pravila odlučili smo se zbog jednostavnije implementacije cijele igre s robotima.
Dobivene nagrade od strane agenta odgovaraju novim pravilima igre. Odnosno, malom nagradom za jedenje točkica i visokom pozitivnom nagradom kada pobijedi. Gubitak igre rezultira velikom negativnom nagradom, a za svaku sekundu trajanja igre dodjeljuje mala negativna nagrada u cilju zagovaranja brzih rješenja. U donjoj tablici pišu sve akcije, za koje našem agentu dajemo nagradu.

Pobjeda (pojeo sve točkice)	+500
Pojeo jednu točkicu	+10
Svake sekunde(Time punishment)	-1
Poraz	-500

Agent i lovci se u toku igre kreću po diskretnoj koordinatnoj mreži po slobodnim poljima. Na donjoj slici navodimo primjer vlastito izrađenog labirinta kojeg ćemo koristiti prilikom demonstracije projekta s robotima. U daljnjem tekstu naveden labirint nazivamo Sunce.



Svaki trenutak igre jedinstveno je opisan matricom koja sadrži zid, točke, Pac-Mana i lovce.

```
%%%%%%%%%%%%%%%%%%%%
%......% GG %......%
%.%%...%%  %%...%%.%
%..................%
%..%%%.%%%%%%.%%%..%
%........P.........%
%..%%%%%   %%%%%%..%
%..........%.......%
%%%%%%%%%%%%%%%%%%%%
```

Listing 6. labirint Sunce

Testiranja ćemo provoditi na većem broju različitih labirnata koji su poznati iz igre. Gornji labirint dimenzija je $20 \cdot 9$ što nam daje ukupan broj različitih situacija $92^3 \cdot 4 \cdot 2^{82} = 150620164579852218046976444433408$. S obzirom na veliki broj raznih mogućnosti trebat ćemo naći algoritme koji će iz manjeg broja naučenih primjera mogu zaključiti na većem skupu koji prije nisu vidjeli.

5. Pregled dosadašnjih istraživanja

Posljednjih desetak godina proveden je veći broj istraživanja u kojima se analizirala igra Pac-Man. Neki od tih istraživanja su [7] i [8]. Problemu su pristupali korištenjem algoritma Q-learning. Primjerice u [7] su rješavali igru Ms. Pac-Man vs. Ghosts, pa su umjesto Q tablice koristili su tzv. *case base*. Koristili su *Case-based reasoning* kojem je ideja je da slični problemi imaju slična rješenja. Što je inaće česta ideja u rješavanju problema korištenjem umjetne inteligencije. Prema njihovim snimljenim rješenjima[4] zaključujemo da uspješno rješili problem. U [8] su za rješavanje igre Ms. Pac-Man uz Q-learning algoritam koristili i skup značajki s kojima su opisivali zadani skup stanja. Prema njihovim rezultatima navedeni postupak bio je uspješan i Pac-Man je u testiranju pobjedio u svim igrama. Djelo Mhina i suradnika [11], gdje koriste dubinsko Q-učenje (DQL) za obuku igrača u Atari igrama. Budući da je dokazano da DQN može imati stvarno dobre performanse u Atari igrama, čak i bolji od nečovječnih performansi u nekim igrama, DQN je privukao puno pažnje i predložena su mnoga poboljšanja. Nakon uspijeha s DQN u detaljnom radu [12] njemački stručnjaci uspjevaju kombinacijom neuronskih mreža i učenja s podrškom riješiti problem.

S tehničke strane stručni rad [6] u kojem se pomoću robota implementira Pacman je zaista impresivan. Neovisno o tome što se radi o potpuno drugačijoj robotskoj platformi gdje roboti imaju kamere, a način komunikacije ostvaruje se preko Socket APIa. Napomenimo da smo i mi također pokušali komunikaciju provesti na sličan način, ali su na Codey Rockyu svi portovi zatvoreni.

[4] https://www.youtube.com/watch?v=4phEjPAGrbk

Dakle naš problem je proučavan i rješavan već godinama, ali vrijedi ga pogledati iz malo više kuteva i probati nekolicinu različitih metoda rješavanja.

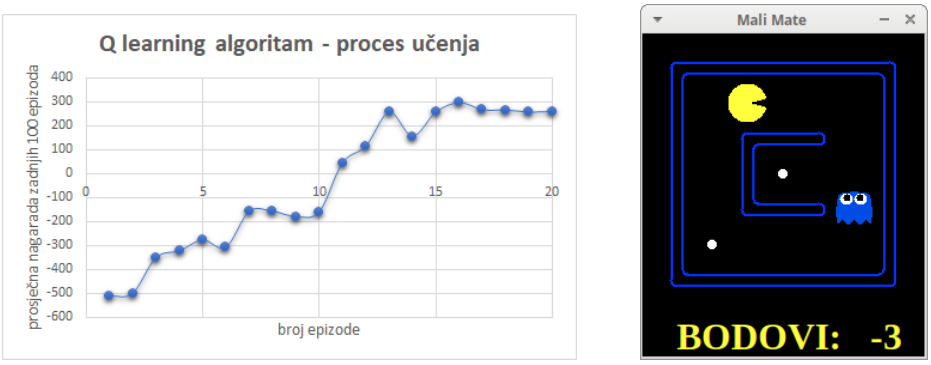
6. Materijali, metodologija i plan istraživanja

Problem koji pokušavamo riješiti pripada kategoriji problema koji se rješavaju koristeći se metodama iz *reinforcement learninga*. Pretpostavljamo da će naš problem biti moguće modelirati korištenjem konačnog Markovljevog lanca. Skup stanja S biti će sve moguće konfiguracije igre uključujući poziciju igrača i lovaca, dok će skup mogućih akcija A biti sveden na svega 5 mogućnosti: lijevo, desno, gore, dolje ili ostani na mjestu. Cilj je naučiti igrača kontrolnu strategiju $\pi : S \rightarrow A$.

Jedan od načina kako se može naučiti kontrolna strategija je uporabom algoritma Q-learning. Osnovna ideja algoritma je koristiti Bellmanovu jednadžbu kao jednostavno ažuriranje iteracije vrijednosti korištenjem težinskog prosjeka stare vrijednosti i novih informacija.

$$Q^{new}(s_t, a_t) \leftarrow Q(s_t, a_t) + (r_t + \alpha(Q(s_{t+1}, a)) - Q(s_t, a_t)) \tag{1}$$

Dokazano je da za konačni Markovljev lanac Q-vrijednost uvijek konvergira [2]. Međutim, ovaj teorijski dokaz konvergencije ne znači da je konvergencija zagarantirana u praktičnim okruženjima, jer obično postoje ograničenja u vremenu treniranja modela. Nažalost navedeni algoritam nije dovoljno dobar za ovaj tip problema niti na puno manjem labirintu. To pokazuje donji graf u kojem je prikazan tijek prosječno sakupljenih bodova svaki 100 epizoda od njih ukupno 2000. Graf prikazuje rast istreniranosti modela, ali s obzirom na limitiran broj mogućih stanja donjeg labirinta to je bilo i za očekivati.



Slika 2. graf učenja i labirint na kojem je provedeno učenje

Naveden algoritam proveden je za fiksirane parametre $\epsilon = 0.05, \alpha = 0.2, \gamma = 0.8$. Prvi i osnovni problem ovog algoritma prilikom primjene na početni labirint je prevelik broj stanja.

Slika 3. slike 1, 2 i 3

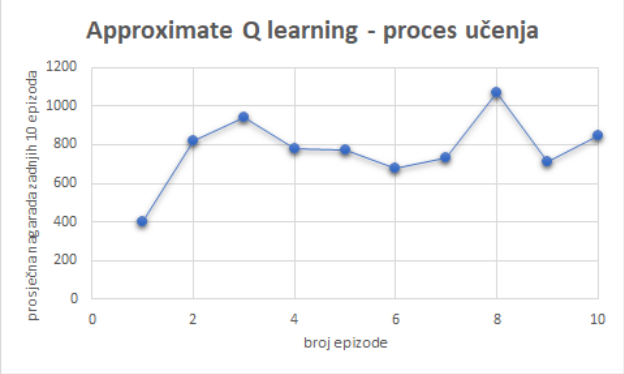
Primjerice 1 i 2 slika su nama jednake, ali algoritmu nisu i dva puta mora učiti istu stvar. Tako primjerice slike 1 i 3 razlikuje samo 1 točka i to je za algoritam jedna potpuno nova okolnost koju nije vidio i ponovno ju mora učiti. Zato je ideja pokušati s određenim skupom značajki riješiti gore navedeni problem. Značajke su funkcije koje stanju pridružuju realne vrijednosti koje opisuju bitna svojstva stanja. Neke od značajki s kojima bi pokušali riješiti problem su: udaljenost od najbliže hrane, minimalna udaljenost od lovca, broj duhova koji su udaljeni od igrača na 2 koraka i sl. Ideja je da značajka opisuje što općenitiji skup mogućih situacija, a ne da se koristi npr. značajka ušao u tunel koja bi se koristila samo u limitiranom broju slučaja. Prednost te ideje da iskustvo agenta možemo svesti u nekoliko bitnih brojeva, dok je nedostatak da određena stanja mogu dijeliti značajke, ali u biti suštinski različita. Izbor bitnih značajki za naš model radili bi metodom *backward feature selection* preko uspoređivanja prosječnog broj skupljenih bodova. Algoritam Approximate Q-learning pretpostavlja postojanje funkcije značajki $f(s,a)$ na skupu stanja i akcija koja za komponentne funkcije ima $f_1(s,a), f_2(s,a), \ldots, f_n(s,a)$. Approximate Q-learning funkcija poprima sljedeći oblik:

$$Q(s,a) = w_0 \cdot f_0(s,a) + w_1 \cdot f_1(s,a) + w_2 \cdot f_2(s,a) + \cdots + w_n \cdot f_n(s,a) \tag{2}$$

Pri čemu je svaka težina w_i povezana s određenom značajkom f_i, a $f_0(s,a)$ je *bias* i iznosi 1. Težine se ažuriraju prema sljedećem pravilu:

$$w_i \leftarrow w_i + \alpha \cdot [r_t + \gamma \cdot \max_{a \in A}(Q(s_{t+1}, a)) - Q(s_t, a_t)] \cdot f_i(s,a) \tag{3}$$

Iz donjeg grafa vidimo kako je primjena ovog algoritma na našem početnom labirintu Sunce puno učinkovitija nakon samo 100 epizoda učenja.



Korištene su samo dvije značajke u ovom testiranju: broj lovaca na udaljenosti 1 i udaljenost do najbliže hrane. Udaljenosti smo pomoću algoritma A* s heurističkom funkcijom Manhattan udaljenost[5]. U nastavku rada na projektu pokušat ćemo uključiti i neke druge značajke koje bi mogle dobro opisati traženi skup stanja.
Ručni odabir značajki nije vrlo učinkovit, jer uvijek postoji mogućnost da propustimo važnu značajku. Stoga ćemo implementirati konvolucijsku neuronsku mrežu (CNN) koja može implicitno ekstrahirati značajke i iznijeti Q-vrijednosti za različite moguće akcije za dano stanje. Konvolucijska neuronska mreža (CNN) varijacija je neuronske mreže *feed forward*. Arhitektura CNN-a osmišljena je da iskoristi lokalnu povezanost koja se obično nalazi u podacima visoke dimenzije, kao što su slike. Neuronske mreže za Deep Q-learning obično se sastoje od konvolucionarnih slojeva koje prate potpuno povezani skriveni slojevi. U ovom slučaju $Q(s,a) = Q(s,a;w)$ gdje je $Q(s,a;w)$ funkcija aproksimirana od strane neuronske mreže, a w uključuje *bias* izraza i težina neuronske mreže. Za implementaciju CNN koristit ćemo Python biblioteku TensorFlow[5], a zbog velikog broja stanja planiramo izvršavanja na GPU[6] preko programa Google Colab. Svi slojevi CNN koristit će aktivacijsku funkciju ReLU (Rectified Linear Unit). Navedenu aktivacijsku funkciju jednostavno je implementirati korištenjem funkcije $g(x) = \max(0,x)$ i zbog svojeg linearnog ponašanja i to da u aktivnom stanju propušta signal unaprijed, a gradijent unatrag. Također za CNN s aktivacijskim funkcijama ReLu se pokazalo da treniraju nekoliko puta brže od mreža s drugim aktivacijskim funkcijama, poput $\tanh(x)$[10]. Budući da CNN kao ulazni parametar prima samo brojeve odlučili smo se stanje u igri zapisati koristeći *one hot encoding*.

[5] $d(A,B) = |A_x - B_x| + |A_y - B_y|$

[6] Graphics processing unit - Grafička procesorska jedinica

```
import numpy as np
if char == '%':
  r = np.concatenate([r, [0, 0, 0, 0, 1]])
if char == ' ':
  r = np.concatenate([r, [0, 0, 0, 1, 0]])
if char == 'P':
  r = np.concatenate([r, [0, 0, 1, 0, 0]])
if char == 'G':
  r = np.concatenate([r, [0, 1, 0, 0, 0]])
if char == '.':
  r = np.concatenate([r, [1, 0, 0, 0, 0]])
```

Listing 7. *one hot encoding* ključnih elemenata igre

Poznati problem u RL *exploration vs exploitation* rješavat ćemo ϵ-greedy metodom s početnom vrijednosti 1 i lagani spuštanjem u toku treniranja do $\frac{1}{10}$. Uspješnost svakog od modela mjeriti ćemo preko broja sakupljenih bodova, kao i gledanja broja na koliko je epizoda agent uspješno izvršio zadatak.

7. Očekivani rezultat projekta

Očekujemo da ćemo u završnom izvješću predati rješenje korištenjem Deep Learninga i da ćemo uspjeti bolje razumjeti razloge različitih kretanja robota.
Glavni cilj ovog projekta je koristeći učenje s podrškom naučiti robote da mogu obavljati određene radnje koje bi kasnije vrlo lagano mogli iskoristiti kao pokazni primjer u nastavi informatike u osnovnoj ili srednjoj školi.

Literatura

[1] Ministarstvo znanosti i obrazovanja, *Odluka o donošenju kurikuluma za nastavni predmet Informatike za osnovne škole i gimnazije u Republici Hrvatskoj*, NN, Zagreb, 2019

[2] C. J. Watkins, P. Dayan, *Machine learning*, 1992

[3] C. J. Watkins, Doktorski rad: *Learning from Delayed Rewards*, King's College, 1989

[4] Berkeley Pacman project, `http://ai.berkeley.edu/project_overview.html`

[5] Google, Tensorflow, `https://www.tensorflow.org/`, 2019

[6] R. Madhav, *An Implementation of Pacman game using robots*, Indian Journal of Computer Science and Engineering. 2011

[7] F. Dominguez-Estevez, *Training Pac-Man bots using Reinforcement Learning and Case-based Reasoning* ,Complutense University of Madrid, 2017

[8] L. Bom, R. Henken, M. Wiering, *Reinforcement learning to train ms. pac-man using higher-order action-relative inputs. In: ADPRL. pp. 156–163. IEEE*, 2013

[9] S. Šegović, materijali za kolegiji Duboko učenje, `http://www.zemris.fer.hr/~ssegvic/du/`

[10] A. Krizhevsky, I. Sutskever, G. E. Hinton, *Advances in neural information processing systems*, 2012

[11] V. Mnih, K. Kavukcuoglu, D. Silver i dr., Playing atari with deep reinforcement learning, 2013

[12] J. Peters, R. Calandra, G. Neumann, *Deep Learning for Reinforcement Learning in Pacman*, 2014

[13] Z. Vondraček, *Markovljevi lanci, predavanja*, PMF, 2012