Elektronski Fakultet
Univerzitet u Nišu

Reinforcement learning (RL) agenti za igranje video igara

- Tehnički izveštaj -

Predmet:
Duboko učenje

Studenti:
Emilija Ćojbašić (1645)
Marko Stanković (1680)
Matija Špeletić (1672)

Mentor:
Prof. dr Aleksandar Milosavljević

Niš, jun 2024. godine

Sadržaj

1. Uvod

Duboko pojačano učenje (Deep Reinforcement Learning, DRL) predstavlja jednu od najzanimljivijih oblasti u domenu mašinskog učenja. Kombinujući moćne tehnike dubokog učenja (Deep Learning) sa strategijama pojačanog učenja (Reinforcement Learning), DRL omogućava kreiranje agenata sposobnih za donošenje odluka i učenje iz interakcije sa okolinom. Ovi agenti mogu da rešavaju kompleksne probleme i prilagođavaju se dinamičnim situacijama, što omogućava njihovu primenu u različitim domenima.

Jedna od najpopularnijih primena DRL metoda je upravo u obučavanju agenata koji igraju video igre. Video igre predstavljaju idealno okruženje za testiranje i usavršavanje algoritama pojačanog učenja, jer pružaju simulacije sa jasno definisanim pravilima, ciljevima i povratnim informacijama. Korišćenjem DRL u ovoj oblasti daje mogućnost kreiranja agenata koji ne samo da mogu da igraju igre, već i da postižu performanse koje prevazilaze ljudske igrače.

Cilj ovog projekta je istražiti i demonstrirati primenu DRL tehnika u obučavanju agenata za igranje različitih video igara. Projekat će se fokusirati na implementaciju, trening i evaluaciju agenata u nekoliko popularnih igara, koristeći savremene DRL algoritme. Kroz ovaj projekat, biće demonstrirano kako se teorijski koncepti DRL mogu primeniti u praksi.

2. Pojačano učenje: Teorijski osnovi

Pojačano učenje (engl. *Reinforcement learning*, skraćeno RL) je oblast mašinskog učenja koja se fokusira na preduzimanje odgovarajućih akcija radi maksimizacije nagrade u određenoj situaciji [1]. Koriste ga različiti softveri i mašine kako bi pronašli najbolje moguće ponašanje ili put koji bi trebalo da preduzmu u specifičnim okolnostima. Ova tehnika se razlikuje od nadgledanog učenja (engl. *supervised learning*) po tome što u nadgledanom učenju podaci za obuku dolaze sa ključem odgovora, tako da se model trenira sa tačnim odgovorom, dok u reinforcement learning-u nema unapred definisanih odgovora. Agenti uče na osnovu svog iskustva jer nemaju unapred pripremljen skup podataka za obuku.

Takođe, RL predstavlja nauku o donošenju odluka. Njegov cilj je da nauči optimalno ponašanje u nekom okruženju kako bi se postigla maksimalna nagrada. U RL-u podaci se akumuliraju iz sistema mašinskog učenja koji koriste metodu pokušaja i grešaka. Podaci nisu deo ulaza kao što je to slučaj kod nadgledanog ili nenadgledanog mašinskog učenja.

Ovaj pristup koristi algoritme koji uče iz ishoda i odlučuju koju akciju treba preduzeti sledeće. Nakon svake akcije, algoritam dobija povratne informacije koje mu pomažu da odredi da li je izbor bio tačan, neutralan ili pogrešan. Ovo je odlična tehnika za automatizovane sisteme koji moraju da donose mnogo malih odluka bez ljudskog nadzora. Rezultat je jedan samouki sistem koji uči metodom pokušaja i grešaka. Izvodi akcije s ciljem maksimizacije nagrada, odnosno uči kroz praksu kako bi postigao najbolje rezultate [2].

2.1. Glavne činjenice vezane za pojačano učenje

- *Ulaz:* Ulaz treba da bude početno stanje iz kojeg će model započeti.
- *Izlaz:* Postoji mnogo mogućih izlaza jer postoji mnogo različitih rešenja za određeni problem.
- *Obuka:* Obuka se zasniva na ulazu. Model će vratiti stanje, a korisnik će odlučiti da li će nagraditi ili kazniti model na osnovu njegovog izlaza.
- Model nastavlja da uči.
- Najbolje rešenje se određuje na osnovu maksimalne nagrade.

2.2. Tipovi Pojačanog učenja

Postoje dve osnovne vrste pojačanog učenja:

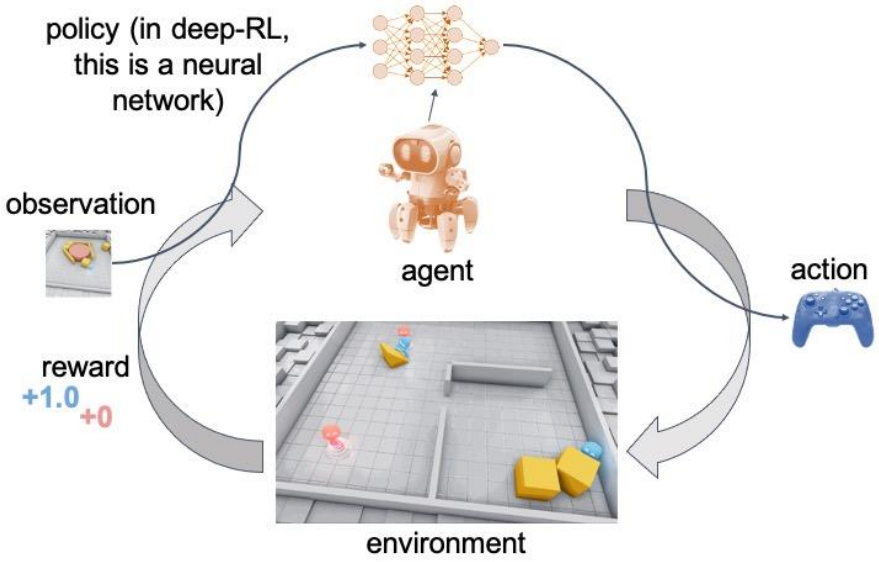
- *Pozitivno*: Pozitivno pojačanje se definiše kao događaj koji se dešava zbog određenog ponašanja i povećava snagu i učestalost tog ponašanja. Drugim rečima, ima pozitivan efekat na ponašanje.
    - Prednosti pozitivnog pojačanja su:
        - Maksimizuje performanse
        - Održava promenu tokom dugog vremenskog perioda
        - Previše pojačanja može dovesti do preopterećenja stanjima, što može smanjiti rezultate

- *Negativno*: Negativno pojačanje se definiše kao jačanje ponašanja jer se negativno stanje zaustavlja ili izbegava.
  - Prednosti negativnog pojačanja su:
    - Povećava učestalost ponašanja
    - Obezbeđuje postizanje minimalnog standarda performansi
    - Pruža samo dovoljno da se postigne minimalno ponašanje

2.3. Osnovni elementi pojačanog učenja

Osnovni elementi pojačanog učenja su:

1. Politika (engl. *Policy*)
2. Funkcija nagrade (engl. *Reward function*)
3. Funkcija vrednosti (engl. *Value function*)
4. Model okruženja (engl. *Model of the environment*)



Slika 1. Osnovni koncepti pojačanog učenja

Politika: Politika definiše ponašanje agenta učenja tokom određenog vremenskog perioda. To je mapiranje percipiranih stanja okruženja na akcije koje treba preduzeti u tim stanjima. Na osnovu politike, agent odlučuje koje akcije će preduzimati u različitim situacijama da bi postigao najbolje rezultate.

Funkcija nagrade: Funkcija nagrade se koristi za definisanje cilja u problemu pojačanog učenja. To je funkcija koja daje numerički skor zasnovan na stanju okruženja. Ovaj skor predstavlja trenutnu povratnu informaciju agentu o tome koliko je određeno stanje ili akcija korisna za postizanje cilja. Funkcija nagrade igra ključnu ulogu u vođenju agenta ka optimalnom ponašanju, jer agent nastoji da maksimizira ukupnu nagradu tokom vremena.

Funkcija vrednosti: Funkcije vrednosti određuju šta je dobro na duže staze. Vrednost stanja predstavlja ukupnu nagradu koju agent može očekivati da akumulira u budućnosti, počevši od tog stanja. Drugim rečima, funkcija vrednosti procenjuje koliko je korisno biti u određenom stanju u pogledu budućih nagrada koje agent može dobiti.

Model okruženja: Modeli se koriste za planiranje. Model okruženja je aproksimacija stvarnog sveta ili sistema u kojem agent deluje. Pomoću modela, agent može simulirati različite scenarije i predvideti ishode svojih akcija bez potrebe da ih zaista izvršava u stvarnom okruženju. Ovo omogućava agentu da unapred proceni posledice svojih odluka i izabere optimalne akcije koje će voditi ka maksimalnim nagradama.

2.4. Prednosti i mane pojačanog učenja

Neke osnovne prednosti i mane pojačanog učenja prikazane su u nastavku [3].

2.4.1. Prednosti pojačanog učenja

1. Pojačano učenje može se koristiti za rešavanje veoma složenih problema koji se ne mogu rešiti konvencionalnim tehnikama.
2. Model može ispraviti greške koje su se pojavile tokom procesa obuke.
3. U pojačanom učenju, podaci za obuku se dobijaju putem direktne interakcije agenta sa okruženjem.
4. Pojačano učenje može da se nosi sa okruženjima koja su nedeterministička, što znači da ishodi akcija nisu uvek predvidljivi. Ovo je korisno u realnim aplikacijama gde okruženje može da se menja tokom vremena ili je nesigurno.
5. Pojačano učenje može se koristiti za rešavanje širokog spektra problema, uključujući one koji se odnose na donošenje odluka, kontrolu i optimizaciju.
6. Pojačano učenje je fleksibilan pristup koji se može kombinovati sa drugim tehnikama mašinskog učenja, kao što je duboko učenje, radi poboljšanja performansi.

2.4.2. Mane pojačanog učenja

1. Pojačano učenje nije poželjno koristiti za rešavanje jednostavnih problema.
2. Pojačano učenje zahteva mnogo podataka i puno računarske snage.
3. Pojačano učenje je veoma zavisno od kvaliteta funkcije nagrade. Ako je funkcija nagrade loše dizajnirana, agent možda neće naučiti željeno ponašanje.
4. Pojačano učenje može biti teško za debagovanje i interpretaciju. Nije uvek jasno zašto se agent ponaša na određeni način, što može otežati dijagnostikovanje i rešavanje problema.

2.5. *Deep Q Learning*

Deep Q-Learning je vrsta algoritma za pojačano učenje koji koristi duboku neuronsku mrežu za aproksimaciju Q-funkcije, koja se koristi za određivanje optimalne akcije u datom stanju. Q-funkcija predstavlja očekivanu kumulativnu nagradu za preduzimanje određene akcije u određenom stanju i praćenje određene politike. U Q-Learningu, Q-funkcija se iterativno ažurira

dok agent interaguje sa okruženjem. Deep Q-Learning nalazi primene u različitim oblastima, kao što su video igre, robotika i autonomna vozila [4].

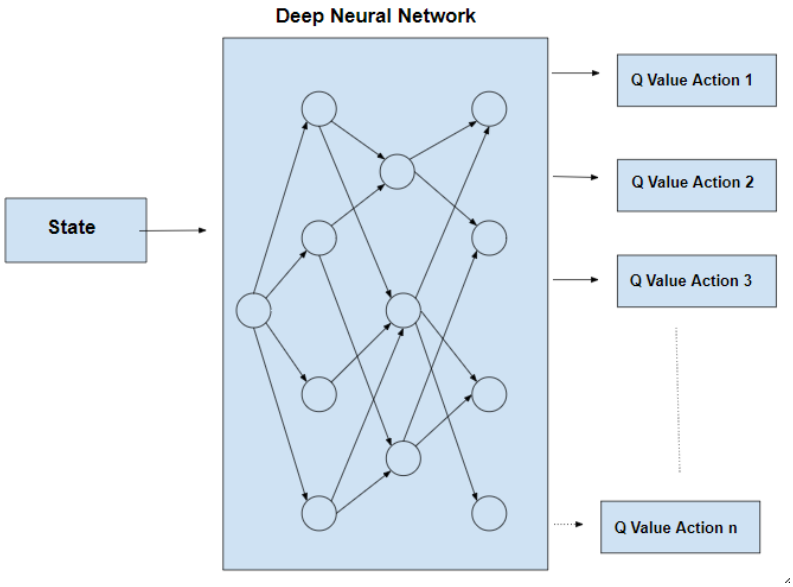
Deep Q-Learning je varijanta Q-Learninga koja koristi duboku neuronsku mrežu za predstavljanje Q-funkcije, umesto jednostavne tabele vrednosti. Ovo omogućava algoritmu da se nosi sa okruženjima sa velikim brojem stanja i akcija, kao i da uči iz visoko-dimenzionalnih ulaza kao što su slike ili podaci sa senzora.

Jedan od ključnih izazova u implementaciji Deep Q-Learninga je da je Q-funkcija tipično nelinearna i može imati mnogo lokalnih minimuma. Ovo može otežati konvergenciju neuronske mreže ka tačnoj Q-funkciji. Da bi se ovo prevazišlo, predloženo je nekoliko tehnika, kao što su iskustveno ponavljanje (experience replay) i ciljane mreže (target networks).

Iskustveno ponavljanje je tehnika gde agent čuva podskup svojih iskustava (stanje, akcija, nagrada, sledeće stanje) u memorijskom baferu i uzima uzorke iz ovog bafera kako bi ažurirao Q-funkciju. Ovo pomaže da se podaci dekorelišu, kako bi se proces učenja učinio stabilnijim. Ciljane mreže, s druge strane, koriste se za stabilizaciju ažuriranja Q-funkcije. U ovoj tehnici, odvojena mreža se koristi za izračunavanje ciljnih Q-vrednosti, koje se zatim koriste za ažuriranje mreže Q-funkcije.

Deep Q-Learning je primenjen na širok spektar problema, uključujući igranje igara, robotiku i autonomna vozila. Na primer, korišćen je za obučavanje agenata koji mogu igrati igre kao što su Atari i Go, kao i za kontrolu robota za zadatke kao što su hvatanje i navigacija.

Deep Q-Learning predstavlja značajan napredak u pojačanom učenju, jer omogućava agentima da efikasno uče i donose odluke u složenim i dinamičnim okruženjima, koristeći snagu dubokih neuronskih mreža za obradu visoko-dimenzionalnih podataka.



Slika 2. Koncept funkcionisanja Deep Q Learning metode

2.6. PPO (*Proximal Policy Optimization*) algoritam

Proximal Policy Optimization (skraćeno PPO) je nedavni napredak u oblasti pojačanog učenja, koji donosi poboljšanje u odnosu na *Trust Region Policy Optimization* (skraćeno TRPO) [5]. Ovaj algoritam je predložen 2017. godine i pokazao je izvanredne performanse kada ga je implementirao *OpenAI*. Osnovni pojam vezan za PPO algoritam pojačanog učenja je pojam politike (engl. *policy*).

Politika, u terminologiji pojačanog učenja, je mapiranje od prostora akcija do prostora stanja. Može se zamisliti kao uputstva za RL agenta, u smislu koje akcije treba da preduzme u zavisnosti od trenutnog stanja okruženja. Kada se govori o evaluaciji agenta, obično se misli na evaluaciju funkcije politike kako bi se utvrdilo koliko dobro agent izvršava zadatke prateći datu politiku. Ovde metode gradijenta politike igraju ključnu ulogu. Kada agent "uči" i ne zna koje akcije daju najbolje rezultate u odgovarajućim stanjima, to se čini izračunavanjem gradijenata politike. To funkcioniše kao arhitektura neuronske mreže, pri čemu se gradijent izlaza, tj. logaritam verovatnoća akcija u tom određenom stanju, uzima u odnosu na parametre okruženja, a promena se odražava u politici na osnovu tih gradijenata [6].

Iako ovaj proveren metod dobro funkcioniše, glavni nedostaci ovih metoda su njihova preosetljivost na podešavanje hiperparametara, kao što su izbor veličine koraka, brzine učenja itd., kao i njihova loša efikasnost uzorka. Za razliku od nadgledanog učenja koje ima garantovan put do uspeha ili konvergencije uz relativno manje podešavanja hiperparametara, pojačano učenje je mnogo složenije sa različitim pokretnim delovima koje treba uzeti u obzir. PPO ima za cilj da postigne ravnotežu između važnih faktora kao što su jednostavnost implementacije, jednostavnost podešavanja, složenost uzorka, efikasnost uzorka i pokušaj da se izračuna ažuriranje pri svakom koraku koje minimizira funkciju troška dok osigurava da odstupanje od prethodne politike bude relativno malo. PPO je, zapravo, metod gradijenta politike koji takođe uči iz online podataka. On samo osigurava da ažurirana politika ne bude previše različita od stare politike kako bi se osigurala niska varijansa u obuci. Najčešća implementacija PPO-a je putem tzv. *Actor-Critic* modela koji koristi dve duboke neuronske mreže, jednu za preduzimanje akcija (glumac) i drugu za rukovanje nagradama (kritičar). Matematička jednačina PPO-a je prikazana ispod:

$$L^{CLIP}(\theta) = \bar{E}_t[min(r_t(\theta)\bar{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\bar{A}_t)]$$

$where,$
$\theta\ is\ the\ policy\ parameter$
$\bar{E}_t\ denotes\ the\ empirical\ expectation\ over\ timesteps$
$r_t denotes the ratio of the probabilities under the new and old policies respectively (also known as Importance Sampling Ratio)$
$\bar{A}_t\ is\ the\ estimated\ advantage\ at\ time\ t$
$\epsilon\ is\ a\ hyperparameter,\ usually\ 0.1\ or\ 0.2.$

Sledeći važni zaključci mogu se izvući iz PPO jednačine:

- To je algoritam za optimizaciju gradijenta politike, tj. u svakom koraku dolazi do ažuriranja postojeće politike kako bi se postiglo poboljšanje određenih parametara.

- Osigurava da ažuriranje nije previše veliko, tj. stara politika nije previše različita od nove politike (to čini tako što "klipuje" oblast ažuriranja na veoma uski opseg).
- Funkcija prednosti je razlika između buduće diskontovane sume nagrada za određeno stanje i akciju i funkcije vrednosti te politike.
- Odnos važnosti uzorkovanja, odnosno odnos verovatnoće prema novim i starim politikama, koristi se za ažuriranje.
- ε je hiperparametar koji označava granicu opsega unutar kojeg je ažuriranje dozvoljeno.

3. Eksperimentalna evaluacija

Kao glavni predstavnik algoritama pojačanog učenja u ovom projektu će biti korišćen PPO algoritam. Optimizacija proksimalne politike (PPO) je jedan od savremenijih algoritama pojačanog učenja koji je razvila kompanija OpenAI. PPO je dizajniran da održi pouzdanost i stabilnost ažuriranja politike, pružajući robustan metod za obuku agenata. Koristi isečeni surogat cilj da ograniči korak ažuriranja politike, sprečavajući velika odstupanja koja bi mogla da destabilizuju proces učenja. Ova ravnoteža između istraživanja i eksploatacije čini PPO veoma efikasnim za širok spektar zadataka, od jednostavnih igara do složenih, dinamičnih okruženja.

Za testiranje i ocenu performansi PPO algoritma, biće iskorišćene tri različite igre: Flappy Bird, Bomberman i Car Racing. Svaka od ovih igri nudi jedinstvene izazove i različite nivoe složenosti za agenta. Kako bismo na adekvatan način proverili kako se PPO algoritam snalazi u različitim scenarijima, biće izvršeno merenje maksimalnih rezultata koje je agent postigao u odgovarajućoj igrici u zavisnosti od broja koraka za treniranje [7].

3.1. Pokretanje RL okruženja: *Gymnasium*

Gymnasium je popularan alat koji se koristi u oblasti pojačanog učenja (Reinforcement Learning - RL) i razvijen je od strane OpenAI. Njegova glavna svrha je da standardizuje i olakša proces treniranja i evaluacije RL agenata kroz različita simulacijska okruženja. Gymnasium pruža jednostavan, intuitivan interfejs za interakciju između RL algoritama i simulacijskih okruženja, što omogućava istraživačima i inženjerima da efikasno eksperimentišu sa različitim algoritmima i upoređuju njihove performanse.

Jedna od ključnih karakteristika Gymnasium-a je širok spektar dostupnih okruženja koja pokrivaju razne domene i nivoe složenosti. Ova okruženja uključuju jednostavne zadatke kao što su balansiranje stuba na pokretnoj osnovi (CartPole) ili pomeranje automobila uzbrdo (MountainCar), kao i složenije simulacije poput Atari igara, robotike i 3D simulacija. Ova raznolikost omogućava korisnicima da testiraju svoje RL agente u različitim uslovima i na različitim zadacima.

Interfejs Gymnasium-a je dizajniran da bude jednostavan za korišćenje. Proces rada sa Gymnasium-om obično počinje kreiranjem instance okruženja pomoću funkcije gym.make(). Nakon toga, agent može interaktivno birati akcije koje će preduzeti na osnovu trenutnog stanja okruženja. Svaka akcija ažurira stanje okruženja i agent dobija povratne informacije u vidu novog stanja, nagrade, i informacije da li je epizoda završena.

Jedna od važnih prednosti Gymnasium-a je njegova sposobnost da se nosi sa izazovima pojačanog učenja, kao što su složena okruženja sa velikim brojem stanja i akcija. Gymnasium omogućava agentima da uče iz visokodimenzionalnih ulaza, kao što su slike ili podaci sa senzora, koristeći napredne RL algoritme poput Deep Q-Learning (DQN) i Proximal Policy Optimization (PPO).

Dodatno, Gymnasium podržava tehnike kao što su "experience replay" i "target networks" koje pomažu u stabilizaciji procesa učenja i poboljšanju performansi agenata. "Experience replay" omogućava agentima da čuvaju i ponovo koriste svoja iskustva, što pomaže u de-korelaciji podataka i stabilizaciji učenja. "Target networks" se koriste za stabilizaciju ažuriranja Q-funkcije, smanjujući oscilacije u vrednostima koje agent uči.

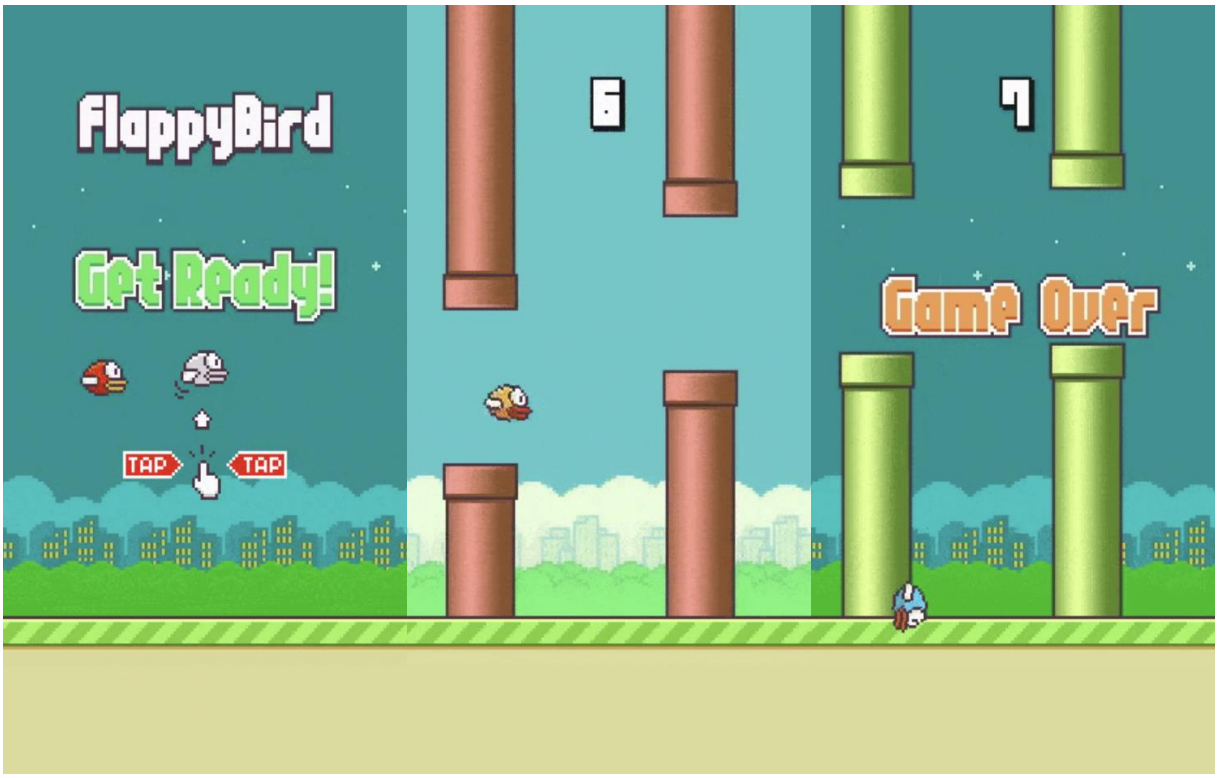
Gymnasium je postao nezaobilazan alat u zajednici pojačanog učenja, jer omogućava brzi razvoj, treniranje i testiranje RL agenata. Njegov standardizovani interfejs olakšava reprodukciju i poređenje rezultata različitih istraživača i timova, čineći ga ključnim alatom za napredak u oblasti pojačanog učenja.

3.2. *Flappy Bird*

Flappy Bird je jednostavna, ali veoma izazovna arkadna igra sa pikselizovanom grafikom koja podseća na stare retro video igre. Igra se odvija u okviru neprekidnog okruženja i centralni element igre su cevi između kojih ptica mora da prolazi, bez da ih dodirne i bez da padne na zemlju. U okviru ove igrice, igrač upravlja ovom malom pticom i jedini potez koji igrač može da napravi je skok (tačnije, zamahivanje krilima), koji povećava trenutnu visinu na kojoj se ptica nalazi. Bez skokova, ptica pada na zemlju pod dejstvom gravitacije.

Cilj igre je da igrač provede pticu kroz što više parova cevi bez da udari u njih ili padne na zemlju. Cevi su postavljene sa gornje i donje strane ekrana, ostavljajući mali prostor između za prolazak ptice. Ptica mora da leti kroz ove otvore, a izazov je u tome što je prostor za prolazak vrlo uzak i cevi se nasumično postavljaju na različitim visinama.

Poeni se skupljaju svaki put kada ptica uspešno prođe kroz par cevi. Broj poena se prikazuje u gornjem delu ekrana i raste za jedan sa svakim uspešnim prolaskom. Ukoliko ptica udari u cev ili padne na tlo, igra se završava i igrač mora da počne ispočetka. Drugim rečima, ova igra nema kraja i u teoriji se može igrati do beskonačnosti.



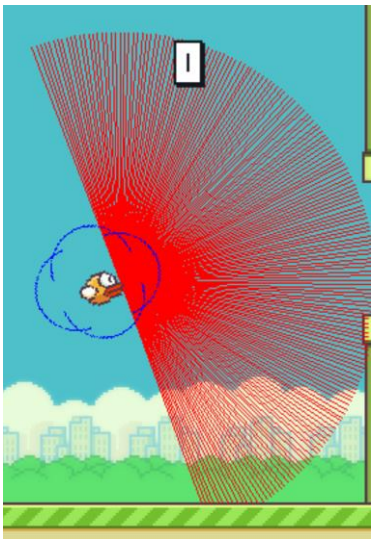
Slika 3: Izgled Flappy Bird igrice (početni ekran - levo, igra u toku - srdina, kraj igre - levo)

3.2.1. Modelovanje okruženja

3.2.1.1. Prostor obzervacija

Kako bismo ovu igricu mogli da iskoristimo za obučavanje agenta pojačanog učenja, pre svega je neophodno modelovati prostor obzervacija. Prostor obzervacija podrazumeva skup korisnih informacija o tome gde se igrač nalazi, u kom je stanju i šta se nalazi oko njega. Postoji više načina kako se ovo može uraditi. Dva često korišćena načina za modelovanje prostora obzervacija u okviru ove igrice su:

- **Korišćenjem informacija odbijenih od virtuelnog LIDAR senzora** koji se nalazi na samoj ptici - Ova opcija koristi 180 očitavanja LIDAR senzora. LIDAR senzor meri udaljenost od prepreka na različitim uglovima, omogućavajući detaljan uvid u okolinu oko igrača. Ova opcija je korišćena u radu [8]. Izgled virtuelnih zraka LIDAR senzora za ovakav način modelovanja je prikazan na slici 4.



Slika 4: Zraci virtuelnog LIDAR senzora

- **Parametarsko modelovanje stanja** igrača korišćenjem informacija o trenutnoj poziciji igrača i cevi. Za ovakav način modelovanja se koriste sledeće informacije:
    - Horizontalna pozicija poslednje cevi
    - Vertikalna pozicija poslednje gornje cevi
    - Vertikalna pozicija poslednje donje cevi
    - Horizontalna pozicija sledeće cevi
    - Vertikalna pozicija sledeće gornje cevi
    - Vertikalna pozicija sledeće donje cevi
    - Horizontalna pozicija preko sledeće cevi
    - Vertikalna pozicija preko sledeće gornje cevi
    - Vertikalna pozicija preko sledeće donje cevi
    - Vertikalna pozicija igrača
    - Vertikalna brzina igrača
    - Rotacija igrača

Za potrebe modela koji će biti obučavan u okviru ovog projekta će biti korišćeno modelovanje prostora obzervacija uz pomoć informacija dobijenih od virtuelnog LIDAR senzora.

3.2.1.2. Prostor akcija

Prostor akcija u igri Flappy Bird je vrlo jednostavan. Sastoji se od samo dve akcije, a to su:

- **0 - ne radi ništa**: Igrač ne pravi nikakvu akciju i ptica se spušta zbog gravitacije.
- **1 - zamahni krilima:** Igrač tapne na ekran, što uzrokuje da ptica zamahne krilima i podigne se naviše.

3.2.1.3. Modelovanje nagrade

Kao glavni deo modelovanja okruženja za agenta pojačanog učenja, nagrada u igri Flappy Bird se dodeljuje na osnovu sledećih pravila:

- ***Ptica ostaje živa***: +0.1 za svaki frejm u kojem ptica ostaje živa, agentu se dodeljuje mala nagrada od +0.1 kako bi se podsticale akcije koje uzrokuju da ptica ostane u letu što duže.

```
reward = 0.1  # reward for staying alive
```

- ***Osvajanje poena***: Kako je osvajanje što većeg broja poena glavni cilj ove igre, agentu se dodeljuje relativno velika nagrada od +1.0 svaki put kada uspešno osvoji poen, odnosno svaki put kada prođe između dve cevi bez kolizije.

```
# check for score
player_mid_pos = self._player_x + PLAYER_WIDTH / 2
for pipe in self._upper_pipes:
    pipe_mid_pos = pipe["x"] + PIPE_WIDTH / 2
    if pipe_mid_pos <= player_mid_pos < pipe_mid_pos + 4:
        self._score += 1
        reward = 1  # reward for passed pipe
```

- ***Smrt ptice***: Kako je osnovni cilj igrice da ptica ostane što duže živa i time osvoji što veći broj poena, za smrt ptice se agentu daje relativno velika kazna od -1.0, obzirom da ovo označava da je igra gotova. Ovo se dešava kada ptica padne na zemlju ili udari u cev.

```
# check for crash
if self._check_crash():
    reward = -1  # reward for dying
    self._player_vel_y = 0
```

- ***Dodir vrha ekrana***: Kako bi se izbegla situacija da agent ostane prilepljen uz vrh ekrana kao posledica konstantnog izbora akcije skoka dodeljuje se kazna od -0.5 za dodirivanje vrha ekrana. Dodirivanje vrha ekrana ne uzrokuje direktno smrt i zato je kazna manja, ali

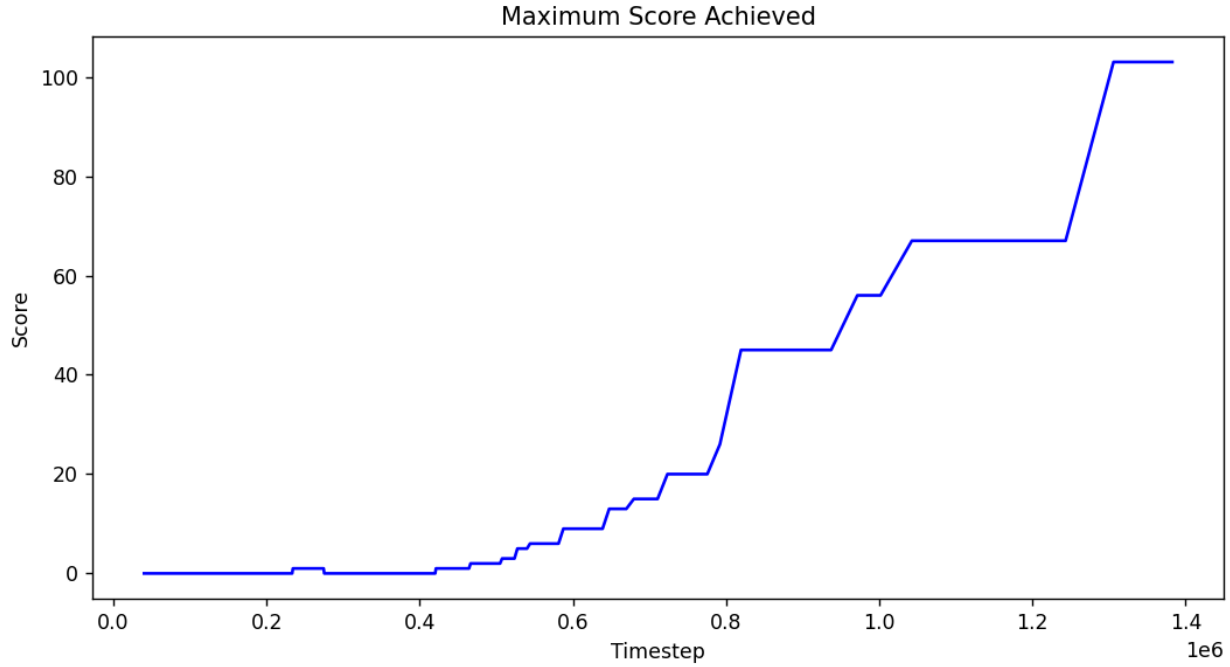
ostajanje uz vrh retko dovodi do dobrih rezultata i zbog toga je poželjno kazniti ovakvo ponašanje agenta, ali uz manju kaznu u odnosu na smrt ptice.

```
# agent touch the top of the screen as punishment
 if self._player_y < 0:
     reward = -0.5
```

Nagrada u ovoj igrici je modelovana tako da se agent podstiče da ostane što duže živ i da osvoji što veći broj poena i tako da se izbegavaju akcije koje izazivaju kraj igre.

3.2.2. Rezultati i diskusija

Obučavanje modela je vršeno korišćenjem vektorskih okruženja dostupnih u okviru *Gymnasium* biblioteke. Radi bržeg obučavanja, paralelno je pokrenuto 16 okruženja i vršeno je merenje performansi agenta za svako od njih. Rezultati obučavanja kroz vreme su prikazani na slici 5.



Slika 5: Maksimalna distanca koju agent prelazi (broj zaobiđenih cevi) tokom obučavanja.

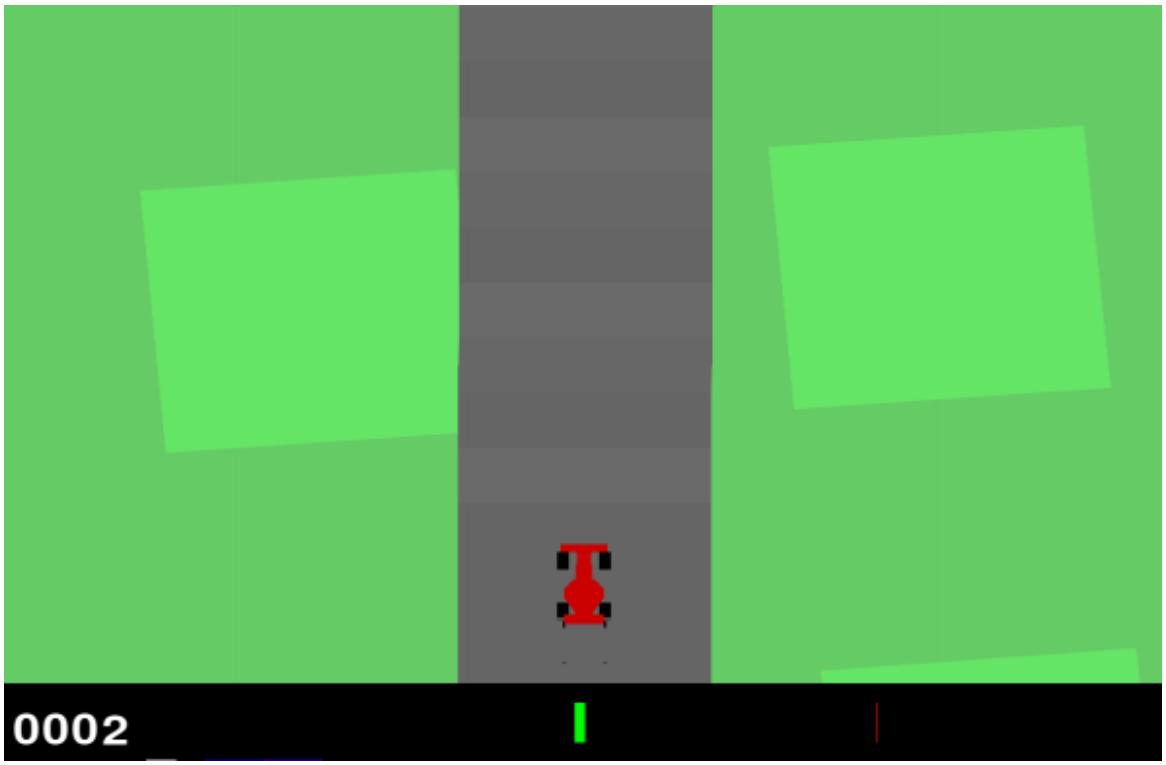
Obučavanjem agenta koji igra igricu Flappy Bird korišćenjem PPO algoritma smo pokazali da ovaj algoritam može dati dobre rezultate u relativno jednostavnim okruženjima, kod kojih je prostor akcija diskretan. Može se primetiti da tokom obučavanja, maksimalna distanca koju agent pređe u igrici (u vidu broja zaobiđenih cevi) postepeno raste u toku obučavanja. Obzirom da se igrica u teoriji može igrati do beskonačnosti, obučavanje je prekinuto nakon što je agent uspeo da dostigne rezultat od 100 (što se za konkretnu igricu može smatrati prilično visokim rezultatom obzirom na njenu težinu).

3.3. *Car Racing*

Car Racing je simulaciona igrica razvijena korišćenjem Box2D fizičkog okvira, koja pruža realistično iskustvo vožnje automobila po trkačkoj stazi. Box2D okvir omogućava detaljnu simulaciju različitih fizičkih aspekata vožnje, uključujući trenje između guma i staze, reakcije automobila na ubrzanje i kočenje, kao i uticaje sudara i prepreka. Ovo čini igricu ne samo izazovnom, već i izuzetno realističnom u pogledu fizike vožnje.

Cilj igre je da agent uspešno vodi automobil kroz kompleksnu trkačku stazu, izbegavajući različite prepreke, savladavajući oštre krivine, i optimizujući brzinu i kontrolu vozila kako bi što brže završio trku. Staze su dizajnirane sa različitim nivoima težine, uključujući ravne deonice za maksimalno ubrzanje, kao i tehnički zahtevne krivine koje testiraju sposobnosti agenta u upravljanju vozilom.

Pored toga, igra zahteva od agenta da balansira između brzine i preciznosti. Prebrza vožnja može dovesti do gubitka kontrole i izlaska sa staze, dok previše oprezna vožnja produžava vreme potrebno za završetak trke. Efikasno upravljanje automobilom, uključujući pravovremeno kočenje i ubrzanje, ključni su faktori za uspeh.



Slika 6: Izgled Car Racing igrice

3.3.1. Modelovanje okruženja

3.3.1.1. Prostor obzervacija

Kako bismo igricu Car Racing mogli da iskoristimo za obučavanje agenta pojačanog učenja, pre svega je neophodno modelovati prostor obzervacija. Prostor obzervacija podrazumeva skup korisnih informacija o tome gde se automobil nalazi, u kom je stanju i šta se nalazi oko njega. Postoji više načina kako se ovo može uraditi. Dva često korišćena načina za modelovanje prostora obzervacija u okviru ove igrice su:

- Korišćenjem mreže stanja okoline - Ovaj način modelovanja koristi mrežu koja predstavlja stanje cele staze, uključujući informacije o poziciji automobila, prepreka, ivica staze i različitih objekata. Mreža pruža kompletnu sliku okoline i omogućava agentu da sagleda celu situaciju u svakom trenutku.

- Parametarsko modelovanje stanja automobila i staze - Za ovakav način modelovanja se koriste sledeće informacije:
    - Horizontalna i vertikalna pozicija automobila (x, y)
    - Brzina automobila
    - Ugao rotacije automobila
    - Udaljenost do najbliže prepreke
    - Udaljenost do ivica staze
    - Stanje točkova (klizanje, kontakt sa stazom)

Za potrebe modela koji će biti obučavan u okviru ovog projekta, biće korišćeno modelovanje prostora obzervacija uz pomoć mreže stanja okoline, koja pruža sveobuhvatan pregled situacije na stazi.

3.3.1.2. Prostor akcija

Prostor akcija u igri Car Racing je jednostavan i sastoji se od sledećih akcija:

- 0 - bez akcije: Automobil održava trenutnu brzinu i pravac.
- 1 - skretanje levo: Automobil se okreće ulevo.
- 2 - skretanje desno: Automobil se okreće udesno.
- 3 - ubrzanje: Automobil povećava brzinu.
- 4 - usporavanje: Automobil smanjuje brzinu.

3.3.1.2. Modelovanje nagrade

U igrici Car Racing, sistem nagrađivanja je dizajniran da podstakne agenta da vozi što brže i da prelazi što veći deo staze. Postoje sledeći aspekti nagrađivanja:

- ***Nagrada za posećivanje novih pločica staze****:* Svaki put kada auto poseti novu pločicu staze koja nije ranije posećena, dobija nagradu proporcionalnu delu ukupne staze. Ovo se daje pomoću 1000.0 / len(self.env.track). Ovo podstiče auto da istražuje stazu i kompletira krugove.

```
if begin:
    obj.tiles.add(tile)
    if not tile.road_visited:
        tile.road_visited = True
        self.env.reward += 1000.0 / len(self.env.track)
        self.env.tile_visited_count += 1
```

- ***Penalizacija po okviru (frame):*** Auto dobija malu kaznu od -0.1 za svaki okvir (frame). Ovo podstiče auto da brzo kompletira stazu.

```
if action is not None:
       self.reward -= 0.1
```

- ***Nagrada za kompletiranje:*** Kada auto kompletira dovoljan procenat staze (definisan sa lap_complete_percent), smatra se da je kompletirao krug i okruženje se resetuje za novi krug.

```
if (
    tile.idx == 0
    and self.env.tile_visited_count / len(self.env.track)
    > self.lap_complete_percent
):
    self.env.new_lap = True
```

- ***Penalizacija za izlazak sa staze****:* Ako auto pređe previše daleko sa staze (izvan igrališta), dobija kaznu od -100 i epizoda se završava. Ovo odvraća auto od napuštanja staze.

```
if abs(x) > PLAYFIELD or abs(y) > PLAYFIELD:
    terminated = True
    step_reward = -100
```

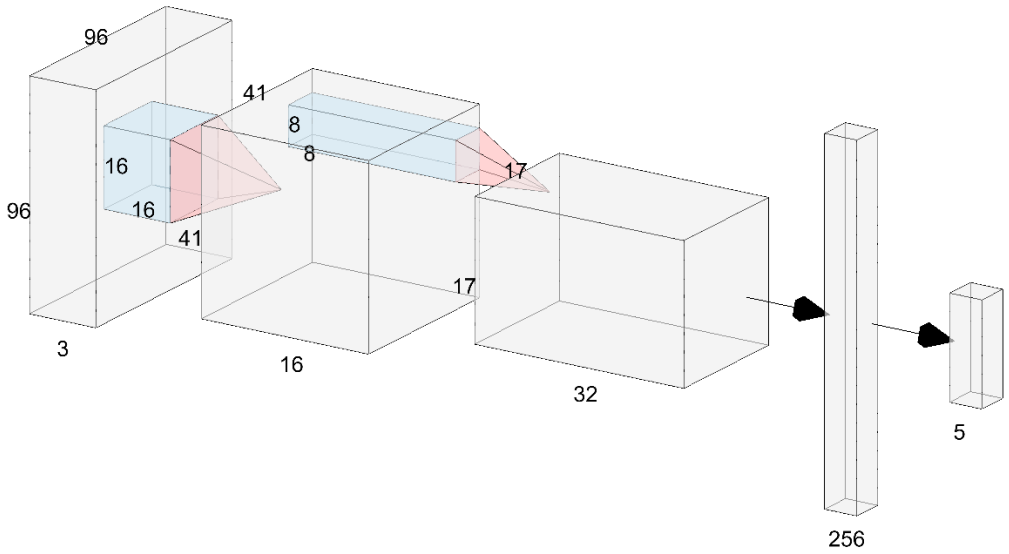
- ***Kod za korak****:* U koraku se izračunava ukupna nagrada i proverava se da li je epizoda završena zbog kompletiranja kruga ili izlaska sa staze.

```
def step(self, action: Union[np.ndarray, int]):
    ...
    step_reward = 0
    terminated = False
    truncated = False
    if action is not None:  # Prvi korak bez akcije, pozvan iz reset()
        self.reward -= 0.1
        self.car.fuel_spent = 0.0
        step_reward = self.reward - self.prev_reward
```

```
            self.prev_reward = self.reward
            if self.tile_visited_count == len(self.track) or self.new_lap:
                truncated = True
            x, y = self.car.hull.position
            if abs(x) > PLAYFIELD or abs(y) > PLAYFIELD:
                terminated = True
                step_reward = -100
        ...
        return self.state, step_reward, terminated, truncated, {}
```

3.3.2. CNN Policy

Konvoluciona Neuronska Mreža (CNN) u ovom okruženju za pojačano učenje je kreirana tako da služi kao ekstraktor karakteristika (feature-a) u okviru algoritma PPO algoritma. CNN prihvata ulazne slike visoke dimenzionalnosti iz igre (sami pikseli ekrana igrice), izvlačeći ključne prostorne karakteristike koje su presudne za donošenje odluka agenta. Arhitektura CNN (slika 7) počinje sa konvolucionim slojevima koji detektuju osnovne karakteristike kao što su ivice i teksture i napreduje ka prepoznavanju složenijih obrazaca. Ovi slojevi su praćeni potpuno povezanim slojevima koji dodatno obrađuju i smanjuju dimenzionalnost karakteristika. Ovo smanjenje je ključno za obradu ulaznih slika visoke dimenzionalnosti i čini proces učenja koji sledi računarski izvodljivim i efikasnijim. Konačni izlaz CNN-a je vektor karakteristika koji je prilagođen na određenu dimenziju pre nego što bude prosleđen u mrežu politike algoritma PPO.

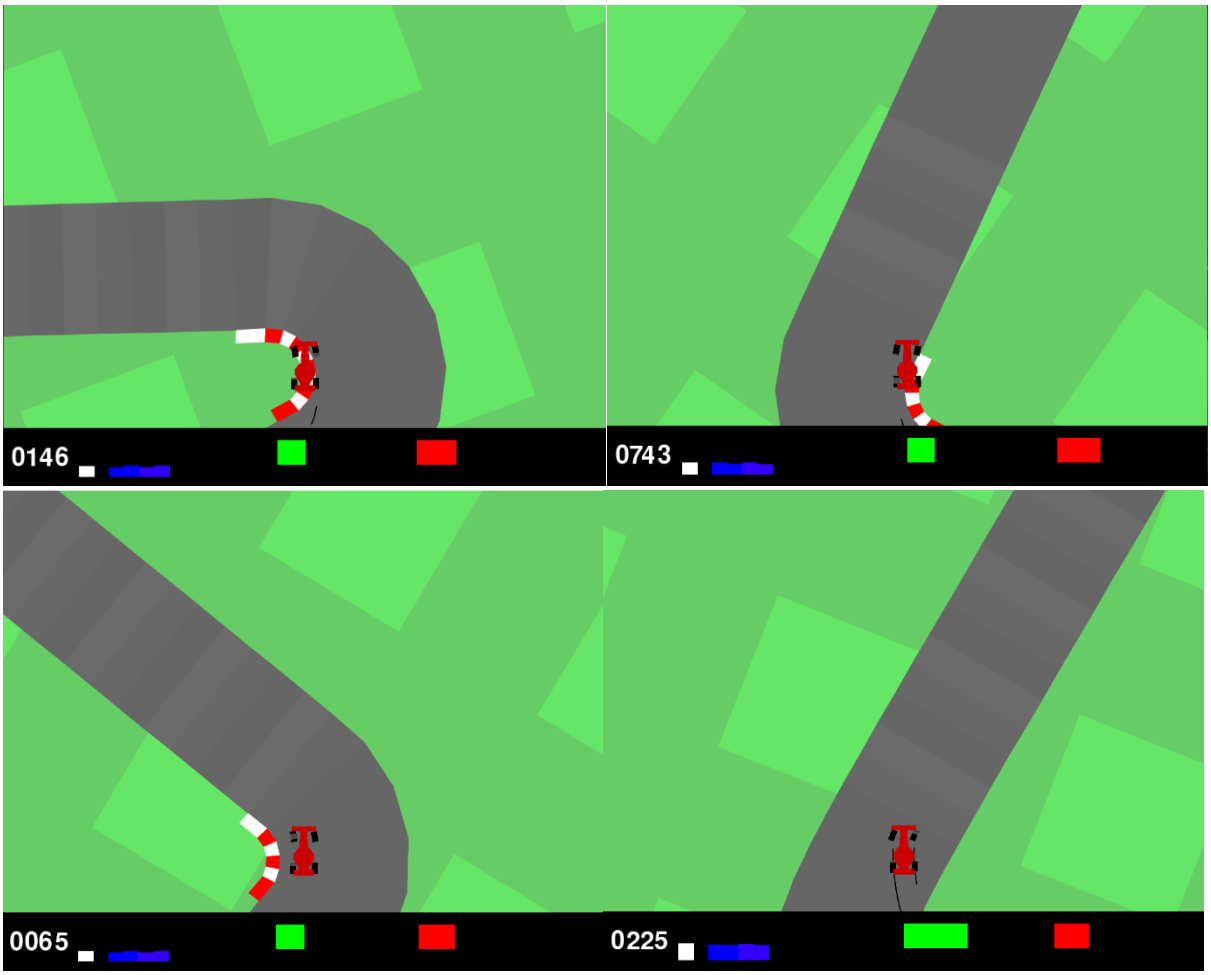


Slika 7: Arhitektura korišćene CNN.

Korišćenje ovakve CNN kao deo PPO algoritma obezbeđuje da mreža politike prima visokokvalitetne, značajne karakteristike, poboljšavajući efikasnost učenja i ukupne performanse agenta za učenje pojačanjem. Fokusirajući se na relevantne aspekte okruženja, CNN omogućava agentu da efikasno interpretira vizuelne informacije, što dovodi do boljeg donošenja odluka i performansi u datoj igrici. Ovaj proces prilagođenog izvlačenja karakteristika ključan je za sposobnost agenta da uči i optimalno funkcioniše u dinamičnom i složenom okruženju.

3.3.3. Rezultati i diskusija

Obučavanjem agenta koji igra igricu *Car Racing* pokazali smo da algoritam može postići dobre rezultate u kompleksnijim okruženjima. Tokom obučavanja, uočeno je da agentova sposobnost da ostane na stazi, uspešno skreće i vrati se na stazu nakon grešaka postepeno raste. S obzirom na to da se igrica može igrati teoretski beskonačno, obučavanje je prekinuto nakon što je agent uspeo da vozi stabilno i bez većih grešaka tokom dužeg vremenskog perioda, što se može smatrati dobrim rezultatima. Na slici 8 možemo videti nekoliko različitih situacija u kojima se agent uspešno snalazi.



Slika 8. Naučeno ponašanje agenta za igranje Car Racing igre.

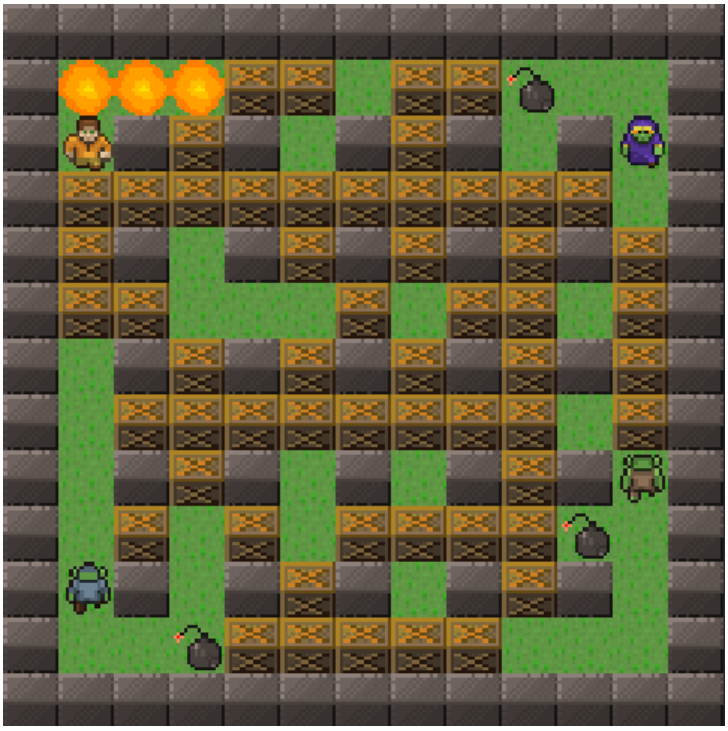
3.4. *Bomberman* igrica

Bomberman je klasična arkadna igra u kojoj igrači preuzimaju ulogu hrabrog lika koji postavlja bombe kako bi razneo prepreke i eliminisao protivnike. Igra se odvija u lavirintu prepunom drvenih kutija i drugih prepreka koje igrači moraju uništiti kako bi pronašli put do neprijatelja i eliminisali ih.

Cilj igre je jednostavan, ali izazovan: postavljati bombe na strateškim mestima kako bi eksplozije uništile drvene kutije, otvarajući nove prolaze kroz lavirint. Dok se kreću kroz lavirint, igrači nailaze na razne protivnike koje takođe treba eliminisati bombama. Eksplozije su opasne i za igrače, pa je ključno pažljivo tempiranje kako bi se izbeglo da budu uhvaćeni u vlastitoj eksploziji ili eksplozijama koje postavljaju protivnici.

Uništavanjem kutija, igrači mogu otkriti skrivene bonuse koji poboljšavaju njihove sposobnosti, kao što su dodatne bombe koje mogu postaviti odjednom, povećanje radijusa eksplozije. Ovi bonusi su od velike pomoći jer nivoi postaju sve izazovniji, sa sve složenijim lavirintima i opasnijim protivnicima.

Kako igra napreduje, igrači se suočavaju sa sve većim izazovima, pokušavajući da očiste svaki nivo od neprijatelja i prežive opasnosti koje dolaze sa postavljanjem bombi. Bomberman zahteva brze reflekse, strateško razmišljanje i precizno kretanje, pružajući sati zabave i uzbuđenja dok igrači istražuju lavirinte, skupljaju bonuse i eliminišu sve neprijatelje na svom putu ka pobedi.



Slika 9. Izgled Bomberman igrice

3.4.1. Modelovanje okruženja

3.4.1.1. Prostor obzervacija

Kako bismo igricu Bomberman mogli da iskoristimo za obučavanje agenta pojačanog učenja, pre svega je neophodno modelovati prostor obzervacija. Prostor obzervacija podrazumeva skup korisnih informacija o tome gde se igrač nalazi, u kom je stanju i šta se nalazi oko njega. Postoji više načina kako se ovo može uraditi. Dva često korišćena načina za modelovanje prostora obzervacija u okviru ove igrice su:

- **Korišćenjem mreže stanja okoline** - Ovaj način modelovanja koristi mrežu koja predstavlja stanje cele table u igrice, uključujući informacije o poziciji igrača, neprijatelja, bombi, eksplozija i različitih objekata. U ovom slučaju, mreža je 13x13 matrica gde svaka ćelija ima vrednost koja označava tip objekta u toj ćeliji. Ova mreža pruža kompletnu sliku okoline i omogućava agentu da sagleda celu situaciju u svakom trenutku.

- **Parametarsko modelovanje stanja igrača i neprijatelja** - Za ovakav način modelovanja se koriste sledeće informacije:
    - Pozicija igrača (x, y)
    - Broj preostalih bombi
    - Broj uništenih sanduka
    - Pozicije neprijatelja (x, y) za svakog neprijatelja
    - Stanja bombi (pozicije i preostalo vreme do eksplozije)
    - Stanja eksplozija (pozicije i trajanje)
    - Pozicije i tipovi pojačanja na tabli

Za potrebe modela koji će biti obučavan u okviru ovog projekta, biće korišćeno modelovanje prostora obzervacija uz pomoć mreže stanja okoline, koja pruža sveobuhvatan pregled situacije na tabli.

3.4.1.2. Prostor akcija

Prostor akcija u igri Bomberman je složeniji od prostora akcija u igri Flappy Bird. U Bombermanu, igrač može birati između šest različitih akcija, a to su:

- 0 - Kretanje gore: Igrač pomera lika za jedno polje prema gore.
- 1 - Kretanje dole: Igrač pomera lika za jedno polje prema dole.
- 2 - Kretanje levo: Igrač pomera lika za jedno polje prema levo.
- 3 - Kretanje desno: Igrač pomera lika za jedno polje prema desno.
- 4 - Postavljanje bombe: Igrač postavlja bombu na trenutnoj poziciji.
- 5 - Ostajanje u mestu: Igrač ne pravi nikakvu akciju.

Ovaj prostor akcija omogućava igraču da se kreće po mreži, postavlja bombe za uništavanje prepreka i protivnika, kao i da strateški odlučuje kada će ostati u mestu. U poređenju sa Flappy Bird, Bomberman pruža mnogo više opcija i kompleksniju dinamiku igre, što zahteva promišljeno planiranje i brzo donošenje odluka.

3.4.1.3. Modelovanje nagrade

U igri Bomberman, funkcija nagrade (engl. *reward function*) je pažljivo dizajnirana kako bi podstakla agenta na optimalne akcije i ponašanje [9]. U nastavku je dat detaljan opis elemenata koji su uzeti u obzir prilikom modelovanja nagrade:

- ***Kazna za smrt***: Ako igrač umre, igra se završava i dobija se značajna kazna od -0.5.

```
if self.player.life:
    # (Ovaj deo koda se ne izvršava jer igrač nije živ)
else:
    self.done = True
    reward -= 0.5  # Penalty for player death
```

- ***Ubijanje neprijatelja***: Ubijanje neprijatelja predstavlja jedan od važnijih koraka ka pobedi u igrici i zbog toga je poželjno agentu dodeliti visoku nagradu za ovo. Svako ubistvo neprijatelja povećava nagradu po +1.0.

```
# Reward for killing each enemy
reward += (1.0 * self.player.kills)
```

- ***Uništavanje kutija***: Uništavanje kutija motiviše agenta da aktivno učestvuje u igri, kreće se po terenu i ide ka pobedi. Zbog ovoga se za svaku uništenu kutiju agentu dodeljuje manja nagrada od 0.1 po uništenoj kutiji.

```
# Reward for destroying crates
reward += (0.1 * self.player.crates_destroyed)
```

- ***Blizina bombe*:** Agentu se dodeljuje nagrada u zavisnosti od toga da li se nalazi na bezbednoj poziciji ili ne. Bezbednost pozicije podrazumeva proveru dometa bombe i eksplozije. Ukoliko se agent nađe unutar dometa eksplozije, dodeljuje mu se manja kazna od -0.000666, a ukoliko se nađe na bezbednom mestu, dodeljuje mu se manja nagrada od 0.02.

```
# Bomb vicinity
for bomb in self.bombs:
    distance = math.sqrt((bomb.pos_x - px)** 2 + (bomb.pos_y - py)** 2)
    if distance > bomb.range:
        continue

    if [px, py] in bomb.sectors:
        reward -= 0.000666
    else:
        reward += 0.002  # Reward for standing on safety near bomb
```

- ***Blizina eksplozije***: Eksplozija ostaje aktivna neko vreme nakon što bomba eksplodira i predstavlja opasnost za agenta. Identična logika za modelovanje nagrade u slučaju bombe važi i ovde.

```
# Explosion vicinity
for explosion in self.explosions:
    distance = math.sqrt((explosion.sourceX-px)**2+(
        explosion.sourceY-py)**2)
    if distance > explosion.range:
        continue
    if [px, py] in explosion.sectors:
        reward -= 0.000666  # Punish for standing on explosion path
    else:
        reward += 0.002  # Reward for standing on safety near explosion
```

- ***Kazna po iteraciji***: U svakoj iteraciji se agentu dodeljuje manja kazna od -0.01 čime se agent podstiče da završi igru što pre, jer bi u suprotnom agent mogao da čeka da ostali protivnici završe igru.

```
# Iteration penalty
reward -= 0.001 * self.steps
```

Logika nagrađivanja u igrici Bomberman je osmišljena tako da balansira između podsticanja igrača na proaktivne i strateške akcije (kao što su kretanje, postavljanje bombi i skupljanje bonusa) i kazne za rizična ili neoptimalna ponašanja (kao što su blizina bombama, zarobljenost ili smrt). Ovaj pristup omogućava igračima da uče iz svojih grešaka i poboljšavaju svoje performanse kroz kontinuiranu interakciju sa okruženjem igre.

Još neke od strategija za modelovanje nagrade koje su isprobane u okviru ovog projekta, koje nisu dale značajna poboljšanja u performansama data su u nastavku:

- ***Kazna za zarobljenost***: Ako je igrač zarobljen i ne može da se pomeri (okružen preprekama ili zidovima), dobija kaznu od -0.2.

```
px = self.player.pos_x // self.player.TILE_SIZE
py = self.player.pos_y // self.player.TILE_SIZE
# Punish if trapped
if self.state[px - 1][py] != 0 and self.state[px + 1][py] != 0 and
self.state[px][py - 1] != 0 and self.state[px][py + 1] != 0:
    reward -= 0.2
```

- ***Pokreti***: Igrač dobija malu nagradu (+0.01) za svaki uspešan pokret.

```
if movement:
    reward += 0.01  # Small reward for moving
```

- ***Postavljanje bombi***: Kada igrač postavi bombu, povećava se nagrada. Ako bombe unište neprijatelje ili prepreke, igrač dobija dodatne nagrade.

```
elif action == 4:  # Plant bomb
    if self.player.bomb_limit > 0:
        temp_bomb = self.player.plant_bomb(self.state)
        self.bombs.append(temp_bomb)
        self.state[temp_bomb.pos_x][temp_bomb.pos_y] = 3
        self.player.bomb_limit -= 1
        # reward += 0.1  # Reward for planting a bomb

for bomb in self.bombs:
    if isinstance(bomb.bomber, Player):
        unique_sectors = list(set(tuple(sector)
                                  for sector in bomb.sectors))
        unique_sectors = [list(sector)
                          for sector in unique_sectors]
        for sector in unique_sectors:
            sx = sector[0]
            sy = sector[1]
           if self.state[sx][sy] == 2 or self.state[sx][sy] == 5:
                reward += 0.025  # Small reward for each bomb
```

- ***Postavljanje svih bombi***: Kada agent postavi sve bombe koje ima, dodeljuje mu se manja nagrada.

```
# Reward for placing all the bombs
if self.player.bomb_limit == 0:
      reward += 0.05
```

- ***Neprijatelji brži***: Agentu se dodeljuje kazna za kutije koje neprijatelji unište, kako bi se podsticao da bude aktivniji i brži od neprijatelja.

```
# Punish if enemies are faster at destroying crates
for enemy in self.enemy_list:
      reward -= (0.05 * enemy.crates_destroyed)
```

- ***Bonusi***: Broj sakupljenih bonusa povećava nagradu (+0.05 po bonusu).

```
# Additional rewards or penalties based on power-ups and other factors
reward += self.player.num_power_ups * 0.0
```

3.4.2. Integracija u *Gymnasium* okruženje

Kako bi se osigurala kompatibilnost koda sa Gymnasium bibliotekom, moraju biti ispunjeni uslovi navedeni u nastavku.

Klasa okruženja mora da bude *child* klasa klase *Env* i da sadrži sledeća polja:

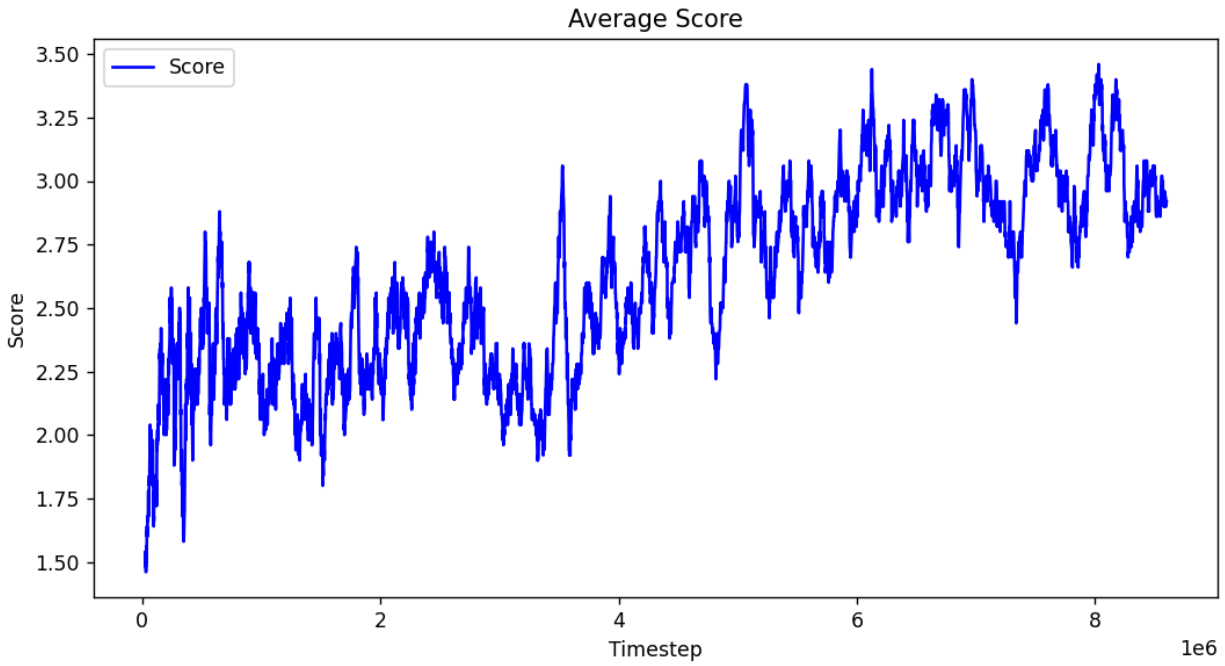
- ***done*** - označava da li je epizoda završena
- ***cumulative_reward*** - vrednost koja označava *reward* u okviru jednog koraka ili epizode
- ***render_mode*** - može imati vrednost *None* ili “human”, što označava mod prikaza prilikom treninga
- ***observation_space*** - odeđuje skup svih mogućih stanja okruženja u bilo kom vremenu
- ***action_space*** - određuje skup svih mogućih akcija koje agent može da izvede u okviru okruženja

Klasa okruženja mora da implementira i sledeće funkcije:

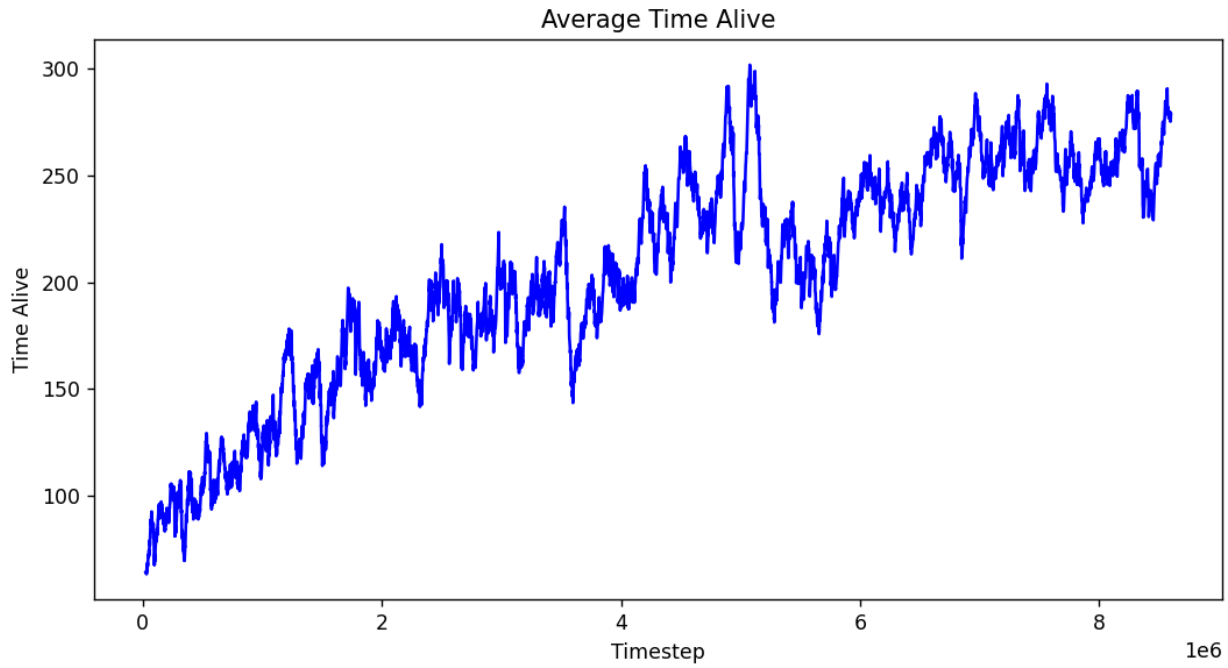
- ***step(action)*** - ova funkcija vrši ažuriranje stanja i vrednosti *reward* promenljiove na osnovu prosleđene akcije *action*. Povratne vrednosti ove funkcije moraju da budu: ažurirano stanje nakon izvšene akcije, vrednosti *reward* i *done* promenljivih, vrednost koja označava da li je epizoda završena ispunjenjem uslova za prekidanje epizode i *info* promenljiva koja sadrži dodatne informacije koje su relevantne za trenutnu epizodu ili korak
- ***reset()*** - ova funkcija se koristi za restartovanje okruženja. Povratne vrednosti ove funkcije su novo stanje okruženja i *info* promenljiva
- ***render()*** - ova funkcija ima ulogu da prikaže trenutno stanje okruženja u obliku koji je razumljiv za čoveka

3.4.3. Rezultati i diskusija

Obučavanjem agenta pomoću PPO algoritma smo pokazali sposobnost ovog algoritma da obradi i okruženja sa većom kompleksnošću. Životni vek agenta i povećan broj uništenih kutija jasno ukazuju na efikasnost algoritma u postizanju dobrih performansi i u ovakvim okruženjima. Velika kompleksnost okruženja uvodi i dve velike prepreke: modelovanje nagrade i vreme obučavanja agenta. Zbog velikog broja mogućih stanja potrebno je odabrati odgovarajuće vrednosti nagrade kako bi agent što lakše učio koje akcije treba da izvrši. Kako je vreme obučavanja ovog agenta veliko, nije moguće proceniti da li su vrednosti nagrade odgovarajuće pre završetka treninga, što dalje otežava izbor prave vrednosti nagrade. Potencijalno poboljšanje obučavanja ovog agenta se ogleda u izboru odgovarajuće vrednosti nagrade.



Slika 10: Broj uništenih kutija tokom procesa obučavanja.



Slika 11: Životni vek agenta tokom procesa obučavanja.

4. Zaključak

U ovom izveštaju smo obradili *Deep-Q Learning* i *Proximal Policy Optimization* algoritme pojačanog učenja i prikazali primenu PPO algoritma obučavanjem agenata za igranje tri različite video igre: Flappy Bird, Car Racing i Bomberman. Svaka od ovih igara poseduje određeni nivo kompleksnosti, gde Flappy Bird ima najjednostavnije okruženje i mali broj akcija, dok Car Racing i Bomberman imaju znatno kompleksnije okruženje i veći broj akcija.

Flappy Bird, kao najjednostavnija igra sa dve akcije (ne radi ništa, zamahni krilima), je omogućila PPO algoritmu da postigne značajan uspeh i konvergenciju za relativno kratko vreme, što pokazuje sposobnost ovog algoritma da efikasno upravlja jednostavnim okruženjima.

Car Racing je predstavljao veći izazov zbog znatno većih prostora obzervacija i akcija. Trening agenta ove igre je trajao značajno duže od treninga agenta Flappy Bird igre, što je posledica veće kompleksnosti. Za trening ovog agenta bilo je ključno projektovanje odgovarajuće CNN arhitekture koja će efikasno da interpretira vizuelne informacije u cilju povećanja performansi i olakšavanja učenja.

Obučavanje agenta za igru Bomberman je bio najizazovniji, kako zbog još veće kompleksnosti okruženja, tako i zbog osiguravanja kompatibilnosti koda sa Gymnasium bibliotekom. Modelovanje nagrade je predstavljalo još jednu veliku prepreku u procesu učenja ovog agenta, što je zajedno sa velikim vremenom obučavanja znatno otežalo brz napredak.

Zaključno, PPO je pokazao odlične rezultate kod okruženja sa manjom kompleksnošću i adekvatne rezultate kod okruženja sa većim prostorom obzervacija i akcija. Trajanje obuke bilo je direktno proporcionalno složenosti igre, pri čemu je za Bomberman trebalo najduže, a za Flappy Bird najkraće vreme.

5. Literatura

[1] Wang, H.N., Liu, N., Zhang, Y.Y., Feng, D.W., Huang, F., Li, D.S. and Zhang, Y.M., 2020. Deep reinforcement learning: a survey. Frontiers of Information Technology & Electronic Engineering, 21(12), pp.1726-1744.

[2] AlMahamid, F. and Grolinger, K., 2021, September. Reinforcement learning algorithms: An overview and classification. In 2021 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE) (pp. 1-7). IEEE.

[3] Li, Y., 2022. Reinforcement learning in practice: Opportunities and challenges. arXiv preprint arXiv:2202.11296.

[4] Özalp, R., Varol, N.K., Taşci, B. and Uçar, A., 2020. A review of deep reinforcement learning algorithms and comparative results on inverted pendulum system. Machine Learning Paradigms: Advances in Deep Learning-based Technological Applications, pp.237-256.

[5] Engstrom, L., Ilyas, A., Santurkar, S., Tsipras, D., Janoos, F., Rudolph, L. and Madry, A., 2019, September. Implementation matters in deep rl: A case study on ppo and trpo. In International conference on learning representations.

[6] Mohammad Hasani Zade, B. and Mansouri, N., 2021. PPO: a new nature-inspired metaheuristic algorithm based on predation for optimization. Soft Computing, pp.1-72.

[7] Jayaramireddy, C.S., Naraharisetti, S.V.V.S.S., Nassar, M. and Mekni, M., 2022, October. A survey of reinforcement learning toolkits for gaming: applications, challenges and trends. In Proceedings of the Future Technologies Conference (pp. 165-184). Cham: Springer International Publishing.

[8] Dirgová Luptáková, I., Kubovčík, M. and Pospíchal, J., 2024. Playing Flappy Bird Based on Motion Recognition Using a Transformer Model and LIDAR Sensor. Sensors, 24(6), p.1905.

[9] Ícaro Goulart, Aline Paes, Esteban Clua. Learning How to Play Bomberman with Deep Reinforcement and Imitation Learning. 1st Joint International Conference on Entertainment Computing and Serious Games (ICEC-JCSG), Nov 2019, Arequipa, Peru. pp.121-133