Súhrn projektu

Tento projekt sa zaoberá skúmaním a implementovaním umelej inteligencie v simulátore Webots a tiež vytvorením webstránky, na ktorej si môže hocikto vyskúšať trénovať nášho robota pomocou posilňovaného učenia (RL) a porovnať svoje výsledky s ostatnými.

1. Webots

Tento softvér poskytuje prostredie na simuláciu robotov, ktorí sa v ňom môžu pohybovať. Dajú sa použiť preddefinovaní roboti (napríklad Nao - malý humanoid, youBot - mobilné rameno, Spot - robot podobný psovi) ale aj vytvoriť vlastní. Toto prostredie je kompatibilné s niekoľkými programovacími jazykmi, čo poskytuje flexibilitu. Medzi tieto jazyky patria Java, C, C++ a Python (ktorý sme si vybrali). Okrem toho je možno používať spolu s ním frameworky, ako napríklad ROS3, PyBullet4, Deepbots5 a OpenAI Gym6, vďaka čomu je možné porovnať výsledky viacerých prístupov k RL a tiež zefektívniť vývoj v tomto prostredí.

Jeho nevýhodou je napríklad to, že na spustenie trénovania RL algoritmov a zložitých simulácií sú vysoké nároky na výpočtové zdroje.

1.1 Naši roboti

V našom projekte sme si vytvorili dvoch vlastných robotov. Jediný rozdiel medzi nimi je vo farbe. Ich rozmery sú šírka/dĺžka/výška 5x5x10 centimetrov. Robotom sme dali kameru (aby videli loptu, bránky, druhého robota). Šírka a výška kamery v pixeloch je 640 x 640.

Aby to vyzeralo, že kopú do lopty, aj keď nemajú nohy - keď sú veľmi blízko lopte, zdvojnásobia svoju rýchlosť, čím loptu odkopnú ďalej od seba. Majú nastavenú hmotnosť na 0.75 kilogramu, aby sa neprevrátili po tom, ak budú rýchlo po sebe kopať do lopty (napríklad keď si ju kopú o mantinel).

1.2 Naše ihrisko

Takisto sme si vytvorili vlastné ihrisko. Jeho šírka je 1.3 metra a dĺžka 1.5 metra. Skladá sa z niekoľkých častí:

1. Mantinely - sú to steny okolo ihriska slúžiace na to, aby mala lopta a roboti obmedzený priestor na svoj pohyb a tiež na to, aby nevypadli z ihriska. Každá stena pozostáva z 22 vektorov
2. Bránky - tvar každej bránky definuje 18 vektorov. Sú vysoké 12 centimetrov, 30 centimetrov široké a dlhé 10 centimetrov. Modrá bránka patrí bielemu robotovi a fialová žltému
3. Samotné ihrisko s niekoľkými čiarami (hraničné čiary - okraje hracej plochy, bránkové čiary, stred určuje stredová čiara a pri bránkach sa nachádzajú pokutové územia)

1.3 Použitá lopta

Ako loptu sme použili preddefinovanú loptu priamo z Webotsu. Nastavili sme jej polomer na 2.1 centimetra a váhu na 0.45 kilogramu.

2. RL v projekte

Aby sme poriadne preskúmali možnosti Webotsu a RL v ňom, rozdelili sme sa do 3 skupín a každá z nich preskúmala iný spôsob uplatnenia tohto učenia. Tieto skupiny sa venovali:

- Implementácii RL bez použitia frameworkov a predtrénovania (čistý RL)
- RL s použitím frameworku Deepbots
- RL s predtrénovaním

2.1 Čistý reinforcement learning

Na začiatku sa implementovali dve neurónové siete - jedna pre bieleho a jedna pre žltého robota. Žltého robota označíme ako agenta. Tento agent je reprezentovaný triedou YellowRobot, ktorý dedí z tried Robot a gym.env. Trieda gym.env definuje sadu funkcií, ktoré sú použité na vytváranie, konfiguráciu a spustenie prostredia pre RL.

Trieda gym.env poskytuje štandardizované rozhranie pre interakciu s prostrediami RL a definuje sadu tried a funkcií, ktoré sa používajú na vytváranie, konfiguráciu a spúšťanie prostredí RL. S týmto prostredím agent interaguje a pozostáva zo stavového priestoru, priestoru pre akcie a z odmeňovacích funkcií. Stavový priestor popisuje rôzne stavy, v ktorých sa agent môže nachádzať. Priestor pre akcie načrtáva možné akcie, ktoré agent môže vykonať. Odmeňovacia funkcia dáva číselnú hodnotu agentovým akciám a agent chce tieto hodnoty maximalizovať. To znamená, že sa naučí takému správaniu, za ktoré dostane najvyššiu odmenu.

Tento spôsob trénovania vyžadoval vytvorenie prostredia so špecifickými pozorovaniami, ktorými sa robot (agent) riadil. Medzi tieto pozorovania patria aktuálna pozícia robota a jeho rýchlosť a tiež objekty, ktoré vidí na kamere (lopta, bránky, oponent). V každom okamihu agent dostal nový vektor s pozorovaniami. Na základe tohto vektora sa agent rozhodne, aká bude jeho ďalšia akcia. Každá akcia ovplyvní jeho pozíciu alebo rýchlosť.

Toto prostredie vyžadovalo implementáciu troch základných funkcií: action, step a reset.

1. **action -** vykoná akciu, ktorú si agent vyberie na základe aktuálneho stavu prostredia. Napríklad otočiť sa, zrýchliť, spomaliť
2. **step -** funkcia zodpovedná za aktualizáciu stavu prostredia na základe akcie, ktorú spravil agent. Vypočíta tiež odmenu, ktorú dostane agent za vykonanú akciu. Je to dôležitá funkcia, pretože umožňuje agentovi učiť sa z interakcií s prostredím a ďalej sa zlepšovať

3. **reset -** obnovuje prostredie do svojho pôvodného stavu, aké bolo na začiatku simulácie. Používa sa na začiatku každej epochy alebo po tom, čo agent dokončí svoju úlohu (strelí gól)

2.1.1 Tréning a výsledky

Agent bol trénovaný využitím Stable Baselines3 knižnice, ktorá bola vybraná kvôli tomu, že podporuje prostredie OpenAI Gym (gym.env) a taktiež ponúka množstvo funkcií a nástrojov na zjednodušenie tréningového procesu, ako je automatické ladenie hyperparametrov a vizualizácia tréningového procesu. Táto knižnica okrem toho poskytuje rozhranie pre algoritmy RL, ako sú PPO, A2C, TD3. V našej implementácii sme sa rozhodli použiť algoritmus Proximal Policy Optimization (PPO), pretože sa vie naučiť z obmedzeného počtu interakcií s prostredím, čo je kľúčové pre našu implementáciu. Agent ukázal schopnosť robiť pokroky vo svojom pohybe a strieľaní gólov. Avšak, niekedy sa vyskytol problém, že agent narazil do svojho oponenta. Výsledkom toho bolo niekoľko neúspešných epôch, pretože po tomto náraze sa už nevedeli oddeliť od seba.

V trénovaní sa vyskytlo aj pretrénovanie v niekoľkých epochách. Ako príklad môžeme uviesť, keď sa agent naučil otáčať sa pri lopte, pretože vďaka tomu bol v jej tesnej blízkosti. Táto jeho stratégia bola efektívna pri dosahovaní cieľa (vidieť loptu), avšak obmedzovala agentovu schopnosť skúmať ďalšie stratégie (kopanie do lopty či hľadanie súperovej bránky). Hoci výsledky trénovania boli sľubné, je tu miesto pre zlepšenie. Týka sa to napríklad systému odmeňovania alebo toho, aby agent dostával viac relevantné informácie o prostredí. Lepšiemu správaniu agenta by mohlo pomôcť predĺženie doby trénovania. Z dôvodu obmedzeného času sme nemali čas vykonať ďalšie iterácie tréningového procesu.

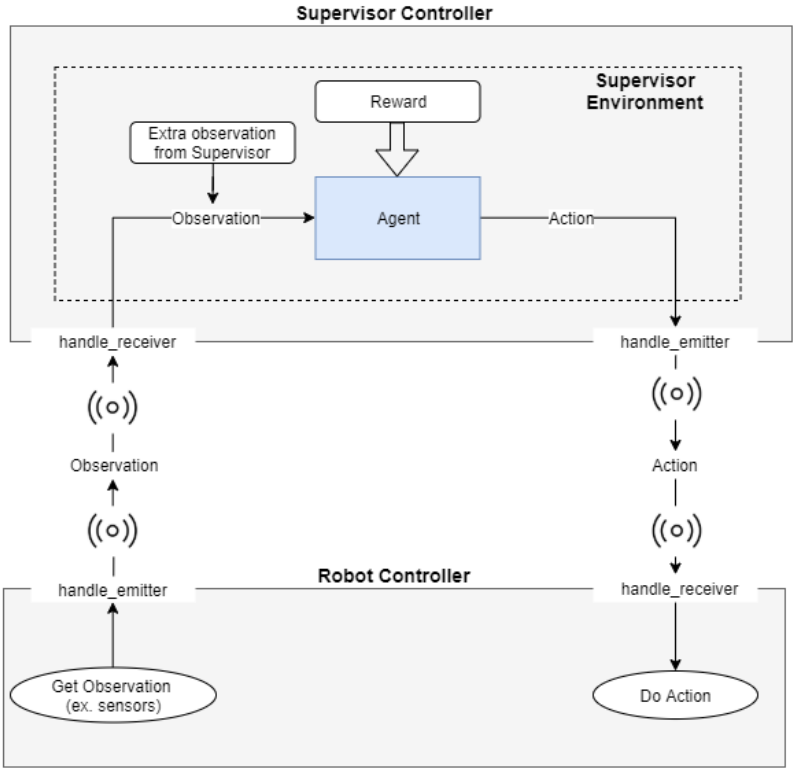
2.2 RL s využitím frameworku Deepbots

Deepbots je jednoduchý open-source framework, ktorý sa zameriava na zjednodušenie tréningu agentov v simulátore Webots. Obsahuje niekoľko pred-programovaných ovládačov pre robotov. Poskytuje rozhranie na vysokej úrovni medzi Webotsom a RL algoritmami, založené na OpenAI gym. Takýto middleware pomáha záujemcom sústrediť sa na riešenie problémov a preskočiť integráciu s prostredím. Podobne ako pri čistej implementácii agenta RL (predchádzajúca časť) sme potrebovali definovať prostredie, akcie, pozorovania, odmeny a časové okamihy (tiky) pre agenta. Tréningový proces je taktiež veľmi podobný, ale z hľadiska architekúry Deepbots poskytuje dve schémy implementácie RL agenta: Emitter-Receiver a Combined Supervisor.

2.2.1 Emitter-Receiver

Táto schéma oddeľuje RL agenta od robota k samotnému ovládaču. Komunikácia medzi RL agentom a robotom je dosiahnutá pomocou vysielacích a prijímacích modulov - robot zbiera pozorovania a posiela ich RL agentovi, ktorý rozhodne o tom, akú ďalšiu akciu robot vykoná.

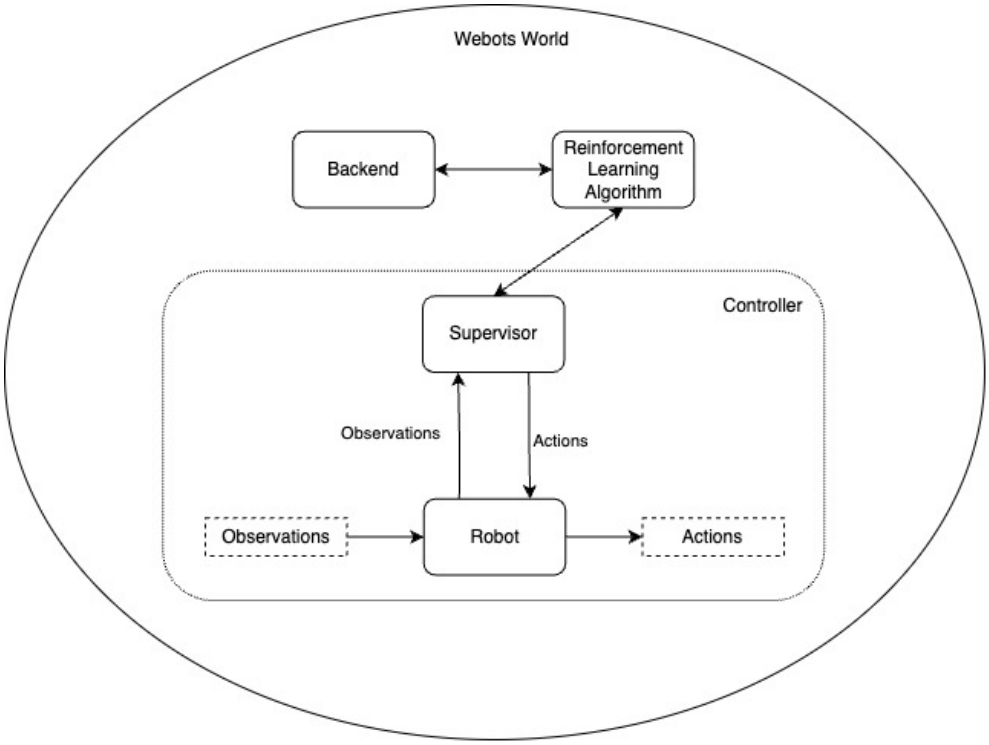
Je to užitočné z niekoľkých dôvodov, napríklad vtedy, keď jeden RL agent musí ovládať viac robotov a získavať pozorovania od každého z nich. V našom projekte sme museli vytvoriť dvoch samostatných a nezávislých RL agentov. Keď sme použili túto schému bez úprav, zistili sme problém s rozhodovaním. Na vyriešenie tohto problému sme vytvorili viacrozmerné prostredie, odmeny a akcie. PPO algoritmus sme nastavili tak, aby sa zdieľali výsledky medzi agentmi, a tým sa rýchlejšie trénovali.



Obrázok 1 - Emitter-Receiver schéma

2.2.2 Combined Supervisor scheme

Táto kombinovaná schéma udržuje RL agenta a jeho ovládač ako rovnakú entitu. Každá inštancia ovládača je úplne nezávislá a spolieha len sama na seba. Tiež odstraňuje potreby emitter a receiver modulov, pretože všetka komunikácia prebieha v rovnakom ovládači. Nedostatočná komunikácia medzi robotmi však vedie k problémom so synchronizáciou. Táto schéma poskytla jednoduchší spôsob implementovania nezávislých ovládačov, ale počas tréningu sme čelili problému so synchronizáciou resetovania prostredia, keď jeden z robotov dokončil trénovaciu epochu skôr ako druhý. Tento problém bol kritický, pretože ten druhý robot mal potom mix pozorovaní a odmien z rôznych epôch. Našťastie sa nám podarilo vyriešiť tento problém pridaním ďalšieho komunikačného kanála do ovládača, ktorý informuje druhého robota o konci epochy.



Obrázok 2 - Combined Supervisor schéma

2.2.3 Tréning a výsledky

Pretože sme sa dohodli, že budeme používať rovnaké pozorovania, odmeny a akcie počas tréningu RL agentov, tak aj tréningový proces bol takmer identický. RL agent ukázal schopnosť pohybovať sa, otáčať sa a ísť smerom k lopte. Niekedy sa im aj podarilo streliť gól. RL agenti čelili podobným problémom pri trénovaní, ako napríklad: otáčanie sa okolo lopty, zastavenie priamo pred loptou, sledovanie druhého robota. Deepbots nám uľahčil tréningový proces aj keď sme čelili problémom s podporou viacerých RL agentov.

2.3 RL s predtrénovanou sieťou

Táto skupina sa zamerala na tréning robotov pomocou vopred natrénovanej siete. Rozdiel medzi touto metódou a metódami bez predtrénovania je v odmenách. Pri takomto tréningu dokážeme "predpovedať budúcnosť" a rozdávame odmeny podľa toho, kto vyhral hru. Najprv sme zhromaždili a vyčistili dáta. Potom sme týmto vyčisteným dátam pridali odmeny. Následne sme postupovali dvomi rôznymi spôsobmi. Jeden z nich bol natrénovať RL model na dátach, uložiť ho a ďalej načítať do Webotsu a použiť na predpovede. Ďalšou možnou cestou bolo použitie PPO Behavior tréningu. Tu sa simuloval cyklus na vyčistených dátach a nechali sme PPO agenta sa učiť. Potom sme agenta uložili a načítali ho do Webotsu. Tam sme ho použili na predikcie a urobili ďalšie tréningy založené na odmenách.

2.3.1 Zbieranie dát

Najprv sme potrebovali mať dataset, na ktorom sme mohli vykonať predtrénovanie našej siete. Aby sme to dosiahli, museli sme získať dáta. Dvaja ľudskí hráči ovládali robotov pomocou klávesnice z pohľadu prvej osoby robota. Jeden (hlavný) počítač zobrazoval pohľad z kamery obidvoch robotov. Každý z hráčov videl iba to, čo videl jeho robot na kamere. Hráči mohli vykonávať štyri akcie - ísť dopredu, dozadu, doľava a doprava. Používali klávesnicu na svojich počítačoch a ako obrazovka slúžil už spomínaný hlavný počítač. To znamená, že ich počítače slúžili ako ovládače pre akcie robota. Tieto akcie boli ďalej odoslané do hlavného počítača, kde boli vykonané. Zbierali sme tri typy údajov: od bieleho robota, žltého robota a od supervízora, ktorý bol nazvaný "logger". Hlavnou úlohou tohto supervízora bolo určovanie toho, ktorý tím vyhral (dal gól). Pretože roboti sa nie vždy pozerajú na loptu, nebolo by bez neho možné určiť, ktorý tím vyhral. Tento supervízor nekomunikoval s robotmi (bol niečo ako rozhodca), takže agenti boli ako normálni futbalisti.

2.3.2 Odmeny

Výhodou predtrénovania je hlavne to, že vidíme budúcnosť. Vieme, ktorý tím vyhral, a na základe týchto informácií vieme pridať odmeny akciám, ktoré predchádzali víťazstvu (alebo prehre). Vďaka tomu vieme pridať ďalšie metriky.

Stručne povedané, ak tím vyhral zápas, bol spätne pozitívne odmenený. Ak tím prehral, bol odmenený negatívne. V tejto metrike nájdeme najkratšiu hru v datasete a normalizujeme výsledky na toto číslo. Celková váha metriky je SUPER VYSOKÁ, čo predstavuje 4/22 v celom rozsahu metriky. Potom sa výsledok vynásobí buď -1, keď tím prehral hru, alebo 1 v prípade výhry. Tento predtrénovací skript, ktorý poskytuje odmeny, sa spúšťa pre každého z robotov nezávisle, pretože každý robot má svoju vlastnú RL neurónovú sieť a nekomunikujú spolu.

2.3.3 Trénovanie v Tensorflowe

V tejto časti sme načítali dataset s odmenami a spustili RL s Q-modelom. Tento model bol uložený do súboru .npz. Potom sme v skripte ovládača Webots načítali uložený model, spustili cyklus a v každom okamihu (ticku) sme použili model na predpovedanie akcie. V tomto riešení bolo RL použité iba na predtréning. V samotnom tréningu sa už model netrénoval, skôr bol použitý na predpovedanie akcií. Táto metóda bola horšia ako druhá metóda.

2.3.4 Tréning s PPO's Behavior Cloning

Na tento typ predtréningu sme použili Behavior Cloning (BC) z frameworku Stable Baselines. S pomocou BC môžeme predtrénovať politiky pre PPO agenta a urýchliť tréning. V tejto metóde bolo RL použité v predtréningu a aj v samotnom tréningu. To znamená, že predikčná

schopnosť a presnosť modelu sa každým cyklom zvyšovala. Preto bola táto metóda lepšia ako predchádzajúca.

3. Konzistencia medzi všetkými skupinami

Veľkou výzvou pre náš tím bolo navrhnúť vhodný systém odmien. Aby sme zabezpečili konzistentnosť všetkých troch skupín a vyhli sa nezrovnalostiam v systéme odmeňovania, vytvorili sme skript, ktorý obsahuje všetky možné odmeny pre agenta. V našej implementácii sme pridelili pozitívnu odmenu napríklad za to, že agent úspešne posunul loptu bližšie k bráne, bol blízko súperovej bránky, nezasiahol súpera. Negatívne odmeny dostane vtedy, keď dlhší čas nevidí loptu alebo zasiahol protihráča. Odmeny sú prideľované v intervale od 0 do 1. Pochopiteľne, najvyššiu odmenu dostane vtedy, keď strelí gól. Naopak, najnižšiu keď gól dostane. Pri použití predtréningu sa používajú aj ďalšie metriky.

Vzorce odmien:

- **check_ball_distance()** - vráti odmenu založenú na vzdialenosti medzi robotom a loptou

$$reward_1 = \frac{1 - \frac{dist([0,0][position\_1_{ball}, position\_2_{ball}])}{football\_pitch\_max\_length}}{6}$$

- **both_ball_own_goal()** - odmenu dostane podľa toho, či vidí loptu a svoju vlastnú bránu

$$reward_2 = -\frac{if(\exists_{ball} \& \exists_{goal_{own}})}{9}$$

- **both_ball_another_teams_goal()** - vráti odmenu ak robot vidí loptu a aj súperovu bránku

$$reward_3 = \frac{if(\exists_{ball} \& \exists_{goal_{opposite}})}{18}$$

- **collision_the_ball()** - ak je robot tak blízko k lopte, že kopne do nej

$$reward_4 = \frac{position\_1_{ball} <= colision\_dist_{ball}}{9}$$

- **ball_visibility()** - koľko okamihov (tickov) robot nevidel loptu

$$reward_5 = -\frac{min(1, \frac{tick_{current} - tick_{last\_seen\_ball}}{max\_not\_seen\_ball})}{6}$$

$$max\_not\_seen\_ball = 100$$

- **collision_opposite_player()** - či dvaja roboti do seba narazili. Zistíme to podľa ich relatívnej vzdialenosti od seba. Odmena závisí od toho, či narazili alebo nenarazili do seba

$$reward_6 = -\frac{position\_1_{opponent} <= colision\_dist_{opponent}}{18}$$

- **ball_and_goal_overlap():** vráti priesečník medzi priamkou prechádzajúcou stredom lopty a stredom hráča a úsečkou, ktorá predstavuje cieľ. Ak neexistuje žiadny priesečník, funkcia vráti nulu

$$reward_7 = \frac{line\_segment_{goal} \cap line_{ball \to player}}{6}$$

$$line\_segment_{goal} = |position\_on\_image\_1_{ball}, position\_on\_image\_1_{ball} + size\_on\_image\_1_{goal}|$$

$$line_{ball \to player} = \overrightarrow{center_{ball}\ , center_{player}}$$

$$center_{player} = \frac{image\_width}{2}$$

$$center_{ball} = \frac{position\_on\_image\_1_{ball} + size\_on\_image\_1_{ball}}{2}$$

$$image\_width = 640$$

- **closest_to_opponent_goal()** - vráti normalizované skóre ako zlomok celkovej vzdialenosti medzi hráčom a oponentovou bránou

$$reward_8 = \frac{1 - \frac{dist([0,0][position\_1_{opponent\_goal}, position\_2_{opponent\_goal}])}{football\_pitch\_max\_length}}{6}$$

Podobne bol vytvorený aj skript s konštantami obsahujúci premenné, ktoré sú rovnaké pre všetky skupiny. Medzi tieto konštanty patria:

- `num_of_sides = 4` – počet strán, do ktorých sa môže robot otáčať
- `wheel_radius = 0.5` – polomer kolies

- `distance_between_wheels = 0.4 –` vzdialenosť stredu dvoch kolies
- `football_pitch_max_length = 2` – maximálna dĺžka futbalového ihriska v metroch
- `football_pitch_min_length = 0` – minimálna dĺžka futbalového ihriska v metroch
- `ball_nearby_in_picture = 0.05` – vzdialenosť medzi robotom a loptou na ihrisku, odmeraná senzormi robota, ktorá definuje dotyk robota s loptou. Hodnota je v nameraná v metroch (čiže 5 centimetrov)
- `distance_of_other_player = 0.05` – vzdialenosť medzi robotom a druhým hráčom na ihrisku, odmeraná senzormi robota, ktorá definuje, že roboty sa zrazili. Hodnota je v metroch (čiže 5 centimetrov)
- `ball_mass = 0.6` – hmotnosť lopty v kilogramoch
- `ball_radius = 0.021` – polomer lopty v metroch
- `h_robot = 0.1` – výška robota v metroch
- `w_robot = 0.05` – šírka robota v metroch
- `w_image = 640` – šírka obrázku z kamery v pixeloch

4. Úspešnosť RL tímov

Všetky tri skupiny dokázali integrovať agentov do simulátoru Webots a trénovať ich pomocou simulačného prostredia. Implementácia PPO agenta si však vyžiadala prispôsobenie a vyladenie, aby sa mohol trénovať vo Webotse.

Na druhej strane sa použite Deepbots frameworku ukázalo ako efektívnejšia metóda integrácie agentov do Webotsu. Framework poskytol riešenie, ktoré odstránilo potrebu prispôsobenia a vyladenia, čo ušetrilo čas a tiež aj námahu v procese implementácie. Okrem toho sa ukázalo, že použitie predtrénovaného PPO agenta je efektívnou alternatívou k trénovaniu od nuly, pretože to šetrí čas. Vďaka tomu agenti mohli rýchlejšie dosahovať dobré výsledky.

Zistili sme, že kombinovaním rôznych prístupov ako napríklad použitie Deepbotsu v kombinácii s predtrénovaným PPO agentom by mohlo ešte viac skrátiť čas implementácie a dosiahnuť strmšiu krivku učenia pri výcviku inteligentných agentov.

Výsledky simulácie boli analyzované hlavne na základe výsledkovej tabuľky, ktorá je rozdelená na dve časti - rollout a čas. Jednou z kľúčových metrík v rolloute je priemerná dĺžka učiacej epochy, v tabuľke reprezentovaná skratkou ep_len. Táto metrika dáva priemerný počet krokov vykonaných v každej epoche simulácie. V prvej tabuľke je táto hodnota 381, uatiaľ čo v druhej tabuľke je 424. Vyššia hodnota pre priemer môže naznačovať dlhšie epochy, čo môže znamenať, že priniesli lepšie výsledky. Ďalšou dôležitou metrikou je

priemerná odmena za epochu, ktorá má skratku ep_rew. Táto metrika zobrazuje priemernú kumulatívnu odmenu získanú v každej epoche simulácie. V prvej tabuľke je táto hodnota 39.7 a v druhej 7.7. Nižšia hodnota môže naznačovať nižšie priemerné odmeny za epochu, čo nie je pre nás žiaduce. Radšej máme vyššie odmeny, čo znamená, že robot sa učí a zlepšuje svoj výkon.

Rollout			**Rollout**	
ep_len_mean	381		ep_len_mean	424
ep_rew_mean	39.7		ep_rew_mean	7.7
Time			**Time**	
fps	108		fps	151
iterations	1		iterations	1
time_elapsed	18		time_elapsed	13
total_timesteps	2048		total_timesteps	2048

Tabuľka 1 – výsledková tabuľka

Časová časť tabuľky uvádza ďalšie metriky, ako fps (počet snímok za sekundu simulácie) a iterations, ktoré označujú počet aktualizácií vykonaných v simulácii. Time_elapsed meria čas v sekundách od začiatku simulácie a total_timesteps vyjadruje celkový počet "timesteps" vykonaných počas simulácie.

Okrem toho sme skúmali samotnú simuláciu a sledovali, ako sa roboti správali v priebehu tréningu, aby sme zistili, či bol ich tréning efektívny. Ak sa po určitom čase vedeli natočiť smerom k lopte a pohybovať sa s ňou smerom k súperovej bránke, naznačovalo to, že tréning je na správnej ceste.

5. Trénovanie neurónky pre všetkých

Na našej webovej stránke si môžete nastaviť ovládač, vybrať svet, pozorovania a začať trénovať neurónku na našich serveroch. Po natrénovaní bude neurónová sieť ohodnotená a umiestnená v rebríčku, kde si môžete porovnať svoje skóre s ostatnými.

Na výber je zatiaľ jeden **ovládač** - Deepbots.

Nasleduje **výber sveta** - sú na výber štyri:

1. Jednoduchý - je tu len bránka a lopta
2. Náhodná lopta - lopta sa objaví na náhodnom mieste na ihrisku
3. Prekážky - na ihrisku sa objavujú aj prekážky, ktorým sa musí robot vyhnúť
4. Tím vs Tím - na ihrisku sa vyskytuje aj druhý robot

Výber pozorovaní - na výber je až 32 rôznych pozorovaní, ktoré si môžete vybrať. Každé pozorovanie je zobrazené aj na obrázku s vysvetlením, čo presne znamená

Naša stránka vám umožňuje vytvoriť si svoj vlastný odmeňovací vzorec pre robotov (v spodnej časti stránky). Po natrénovaní a vyhodnotení vašej vyklikanej neurónovej siete uvidíte úspešnosť vášho vzorca. Túto možnosť nemusíte využiť, vtedy sa použije náš preddefinovaný vzorec. Tento kód je potrebné napísať v Pythone.