ŽILINSKÁ UNIVERZITA V ŽILINE

FAKULTA RIADENIA A INFORMATIKY

DIPLOMOVÁ PRÁCA

Bc. ANDREJ MICHALEK

Riadenie robotického systému na základe obrazových dát v simulačnom prostredí

Vedúci práce: Ing. Peter Tarábek, PhD.

Registračné číslo: 2723/2023

Žilina, 2024

ŽILINSKÁ UNIVERZITA V ŽILINE

FAKULTA RIADENIA A INFORMATIKY

DIPLOMOVÁ PRÁCA

ŠTUDIJNÝ ODBOR: INFORMATIKA

Bc. ANDREJ MICHALEK

Riadenie robotického systému na základe obrazových dát v simulačnom prostredí

Žilinská univerzita v Žiline

Fakulta riadenia a informatiky

Katedra matematických metód a operačnej analýzy

ŽILINSKÁ UNIVERZITA V ŽILINE, FAKULTA RIADENIA A INFORMATIKY.

ZADANIE TÉMY DIPLOMOVEJ PRÁCE.

Študijný program: Inteligentné informačné systémy

Meno a priezvisko	**Osobné číslo**
Andrej Michalek	561561

Názov práce v slovenskom aj anglickom jazyku

Riadenie robotického systému na základe obrazových dát v simulačnom prostredí

Control of a robotic system based on image data in a simulation environment

Zadanie úlohy, ciele, pokyny pre vypracovanie (Ak je málo miesta, použite opačnú stranu)

Cieľ diplomovej práce:
Práca sa zaoberá riadením robotického systému (autonómneho vozidla) tak, aby vykonalo úlohu, ktorá si vyžaduje lokalizáciu určeného typu objektu, navigáciu k tomuto objektu a jednoduchú interakciu s ním. To všetko bez negatívnej interakcie so statickým okolitým prostredím. Tento objekt má svoju jasnú vizuálnu charakteristiku a je statický. Jeho poloha je iba približná. Predpokladom je, že tento objekt sa nachádza v blízkosti robota (v jeho vizuálnom rádiuse). Hlavným zdrojom informácií pri riadení budú obrazové informácie z virtuálnej kamery vo forme 2D, resp. 3D dát. Ako metóda riedenia sa predpokladá využite učenia posilňovaním. Celá úloha bude prebiehať vo vhodnom simulačnom prostredí, ktoré bude generovať potrebné senzorické dáta popisujúce stav systému (hlavne kamerové senzory) a posielať ich agentovi. Ten na ich základe bude voliť akcie, ktoré budú následne vykonané v simulačnom prostredí. Na komunikáciu simulačného prostredia s agentom sa predpokladá využitie ROS 2.

Obsah:
1. Analýza súčasného stavu
2. Výber vhodného simulačného prostredia a spôsobu komunikácie s agentom
3. Implementácia simulačných scenárov a ich validácia
4. Implementácia komunikácie medzi simulačným prostredím a agentom
5. Návrh a implementácia riešenia pre riadenie na báze učenia posilňovaním
6. Experimentálne overenie a záver

Meno a pracovisko vedúceho DP: Ing. Peter Tarábek, PhD., KMMOA, ŽU
Meno a pracovisko tútora DP:

31 OKT. 2023

garant štud. programu
(dátum a podpis)

Zadanie zaregistrované dňa 31.10.2023 pod číslom 2723/2023 podpis _____

Obsah

Zoznam obrázkov

Čestné vyhlásenie

Prehlasujem, že som túto prácu vypracoval samostatne s pomocou konzultácií so školiteľom, vedomostí nadobudnutých na fakulte a znalosťami obsiahnutými v literatúre. Všetky použité zdroje som uviedol.

V Žiline dňa ....

Bc. Andrej Michalek

Poďakovanie

Chcel by som sa poďakovať môjmu školiteľovi Ing. Petrovi Tarábkovi, PhD. za jeho pomoc, ústretovosť, odborné skúsenosti a rady počas celej tvorby práce. Takisto sa chcem poďakovať mojim rodičom za to, že mi umožnili študovať na vysokej škole.

ABSTRAKT V ŠTÁTNOM JAZYKU

MICHALEK, Andrej: *Riadenie robotického systému na základe obrazových dát v simulačnom prostredí*. [Diplomová práca]. – Žilinská univerzita v Žiline. Fakulta riadenia a informatiky; Katedra matematických metód a operačnej analýzy. – Školiteľ: Ing. Peter Tarábek, PhD. – Stupeň odbornej kvalifikácie: inžinier. – Žilina: FRI UNIZA, ŽU, 2024. 75s.

Táto práca skúma možnosti využitia metódy učenia posilňovaním pri riadení autonómneho vozidla. Predpokladom je, že vozidlo sa pohybuje v simulačnom prostredí, kde má identifikovať vopred určený objekt, dokázať k nemu prísť a vykonať s ním jednoduchú interakciu. Rozhodovací algoritmus by mal ako vstup primárne použiť obrazové dáta z kamery. V práci sme sa zamerali na variant úlohy, kde je vozidlom vysokozdvižný vozík a jeho úlohou je vedieť identifikovať paletu, presunúť sa k nej a vedieť do nej zasunúť lyžiny. Ako simulačné prostredie pre autonómne vozidlo sme použili Unity. Komunikácia medzi agentom a prostredím je sprostredkovaná pomocou ROS 2. V rámci experimentov sme overili možnosti riešenia úlohy pomocou agentov DQN a PPO a navrhli sme mechanizmus, ktorý agentovi umožňuje v rámci tréningovej fázy pracovať s viacerými autonómnymi vozidlami súčasne. Práca porovnáva niekoľko stratégií výpočtu odmeny v procese trénovania a ich vplyv na rýchlosť učenia ako aj kvalitu riešenia úlohy. Súčasťou experimentov je aj vyhodnotenie, do akej miery sú natrénovaní agenti schopní sa prispôsobiť na pozmenenú verziu úlohy, s ktorou sa predtým nestretli.

Kľúčové slová: umelá inteligencia, učenie posilňovaním, autonómne riadenie, robotika, simulácie, neurónové siete

ABSTRAKT V CUDZOM JAZYKU

This thesis examines the possibilities of using the reinforcement learning method in driving an autonomous vehicle. The assumption is that the vehicle moves in a simulation environment where it has to identify a predetermined object, be able to navigate to this object and perform a simple interaction with it. The decision algorithm should primarily use image data from the camera as input. In our work, we focused on a variant of the task, where the vehicle is a forklift and its task is to be able to identify a pallet, move to it and be able to insert skids into it. We used Unity as the simulation environment for the autonomous vehicle. Communication between the agent and the environment is mediated using ROS 2. In the course of the experiments, we verified the possibilities of solving the task using DQN and PPO agents, and we designed a mechanism that allows the agent to work with several autonomous vehicles simultaneously during the training phase. The thesis compares several reward calculation strategies in the training process and their impact on the speed of learning as well as the quality of solving the task. Part of the experiments is also an evaluation of the extent to which trained agents are able to adapt to a modified version of the task that they have not encountered before.

Key words: artificial intelligence, reinforcement learning, autonomous driving, robotics, simulations, neural networks

Úvod

Umelá inteligencia zaznamenáva veľký rozmach v oblasti robotiky a iných systémoch, ktoré vyžadujú okamžité rozhodovanie v reálnom čase. Prispelo k tomu viacero faktorov, medzi ktoré patrí rozvoj grafických kariet a iných paralelných prostriedkov nevyhnutných na trénovanie umelej inteligencie. Kvalitné kamery s vysokým dynamickým rozsahom a LIDAR senzory môžu pomocou vysokorýchlostného internetového pripojenia zbierať veľké množstvo dát nevyhnutných v procese učenia. Pravdepodobne najznámejším použitím umelej inteligencie z tejto kategórie je autonómne riadenie vozidiel v premávke. Úloha vodiča v tomto prípade prechádza z aktívnej roly priameho rozhodovania o smere a rýchlosti vozidla len na pasívnu, kedy musí dohliadať na systém, ktorý ovláda auto a až v prípade problémov prevziať riadenie.

Menej diskutovanou oblasťou použitia autonómneho riadenia je komerčná sféra, kde takisto vzniká potreba autonómneho riadenia vozidiel, ale v uzavretom prostredí. Príkladom je ovládanie vysokozdvižného vozíka v priestoroch skladu. Charakteristika úlohy je pritom odlišná v porovnaní s tradičným autonómnym riadením. Vstupom pre tradičné autonómne riadenie môže byť GPS poloha a cestná sieť. Tak isto cesty a dopravné značenie vyzerajú na rôznych miestach podobne, preto sa dajú ako tréningové dáta použiť napríklad existujúce údaje z kamier áut ovládaných skutočnými vodičmi. Rozmiestnenie objektov v sklade alebo v inom prostredí, kde vzniká potreba autonómneho riadenia vozidiel, sa môže meniť podľa polohy paliet a iných objektov a takisto sa môže meniť aj vizuálna podoba objektu, s ktorým treba interagovať, ako aj samotný charakter interakcie. Príkladom z reálneho sveta môže byť vysokozdvižný vozík, ktorý v sklade potrebuje identifikovať konkrétnu paletu, pričom nemá vopred k dispozícii informácie o jej presnej polohe. Pri presune k palete by mal byť schopný v reálnom čase na základe obrazových dát identifikovať prekážky a nenaraziť do nich.

Predpokladá sa, že zadávateľ takejto úlohy nebude mať k dispozícii tisíce hodín kamerových záznamov a údajov zo senzorov, ktoré by boli potrebné na natrénovanie modelu. Na dosiahnutie dobrých výsledkov by bolo takisto potrebné nechať autonómny systém riadiť vozidlo priamo v prostredí. V reálnom sklade by to však bolo časovo veľmi náročné a v rámci tohto procesu by mohlo vozidlo poškodiť seba aj tovar a bolo by nebezpečné pre zamestnancov. Z tohto dôvodu je lepšie pre potreby učenia nahradiť skutočné prostredie simulačným modelom, kde môže čas bežať oveľa rýchlejšie a učenie

bude prebiehať bez akýchkoľvek rizík. V rámci práce sa budeme zaoberať tým, ako vytvoriť vhodné simulačné prostredie a ako sprostredkovať komunikáciu riadiaceho prvku so simulačným prostredím tak, aby bolo možné simuláciu bez zásadných zmien vymeniť za skutočné autonómne vozidlo v skutočnom svete. Cieľom práce je zároveň zistiť, aké vstupné dáta zo senzorov vozidla by systém riadenia vyžadoval a ktorý učiaci algoritmus dosahuje najlepšie výsledky pri riešení tejto úlohy.

1 Analýza súčasného stavu

Práca ako metódu riadenia využíva učenie posilňovaním, ktoré sa v odbornej literatúre často uvádza pod anglickým pojmom reinforcement learning. V teoretickej časti práce priblížime túto metódu a takisto predstavíme technológie, ktoré sa využívajú na komunikáciu riadiacich výpočtových jednotiek a senzorov ako napríklad kamery. V rámci tréningu bude potrebné implementovať simuláciu robotického vozidla a samotného prostredia, preto sa pozrieme na to, aké prostredia sú k dispozícii a predstavíme niektoré kľúčové koncepty, ktoré za nimi stoja a sú dôležité pre správne nastavenie tréningu umelej inteligencie.

1.1 Reinforcement learning

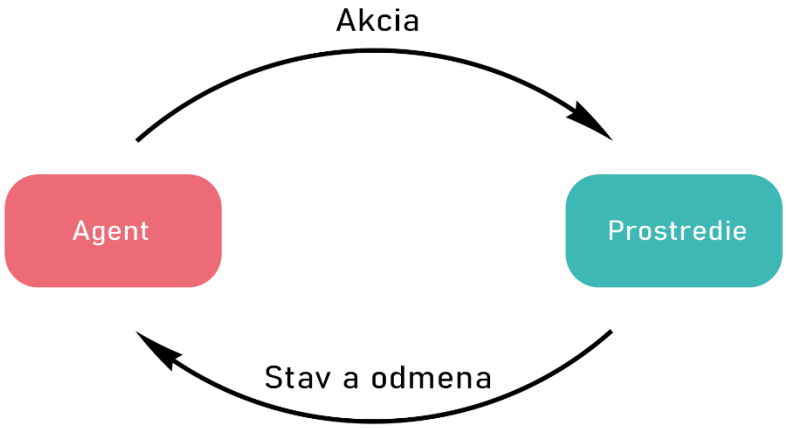
Umelá inteligencia predstavuje širokú oblasť aplikácií, ktoré sú schopné do určitej miery simulovať princípy skutočnej inteligencie. Metódy strojového učenia patria do kategórie umelej inteligencie. Vo všeobecnosti je vhodné použiť strojové učenie za predpokladu, že potrebujeme vyriešiť komplikovaný problém a nevieme, ako exaktne ho vyriešiť. Máme však k dispozícii množinu dát a pomocou algoritmu sa môžeme pokúsiť v dátach nájsť súvislosti, ktoré budú predstavovať riešenie problému alebo budú viesť k objaveniu riešenia.

Učenie posilňovaním (reinforcement learning) je súčasťou strojového učenia a využíva sa primárne v dynamických systémoch, teda v systémoch, kde existuje čas [1]. Množina problémov, ktorú je vhodné riešiť pomocou reinforcement learningu, má niekoľko spoločných charakteristík. Výsledkom, ktorý sa snažíme dosiahnuť, je vhodná stratégia (policy), ktorou sa má agent rozhodovať v prostredí, aby splnil cieľ alebo vopred stanovenú úlohu. Agent má v každom časovom okamihu vykonať rozhodnutie, pričom vyberá z množiny vopred definovaných rozhodnutí – akcií. Prostredie sa v každom časovom okamihu nachádza v určitom stave a agent má k dispozícii informácie o tom, v akom stave sa prostredie nachádza. Po vykonaní akcie prostredie prejde do nového časového okamihu a jeho stav sa zmení. Stratégia je mechanizmus, ktorý agent použije na výber konkrétnej akcie s prihliadnutím na stav systému.

Ak je úloha deterministická, tak správanie prostredia je vopred presne určené a stav prostredia sa mení len na základe akcií agenta. Pri stochastických úlohách však zohráva

úlohu aj náhodný alebo pseudonáhodný jav. Môžu ním byť napríklad rozhodnutia iných agentov, ktorí ovplyvňujú stav prostredia alebo iné vonkajšie vplyvy, ktoré sa riadia rozdelením pravdepodobnosti. Tieto úlohy sú zložitejšie, pretože to isté rozhodnutie v danom stave môže viesť k prechodu do rôznych stavov, čo zvyšuje náročnosť rozhodovania.

Pri väčšine úloh sa snaží agent dosiahnuť konkrétny cieľ – v prípade úlohy, ktorú rieši táto diplomová práca sa autonómne vozidlo pokúša identifikovať v prostredí objekt, prejsť k nemu a následne s ním vykonať interakciu. Postupnosť rozhodnutí agenta by tak mala pri úlohách reinforcement learningu smerovať k nejakému cieľu, a teda by mala byť konečná. Dosiahnutie cieľa je považované za úspech a postupnosť akcií, ktoré k tomu viedli, môžeme označiť ako vhodnú stratégiu na riešenie danej úlohy. Takúto postupnosť budeme označovať ako replikácia. Replikácia môže skončiť aj neúspechom – napríklad ak sa agentovi nepodarilo dosiahnuť požadovaný výsledok v stanovenom počte časových jednotiek. Agent by mal robiť takú postupnosť rozhodnutí, ktoré vedú k očakávanému výsledku a ideálne v čo najkratšom počte krokov.



Obrázok 1 – Komunikácia agenta s prostredím

V rámci procesu učenia pri reinforcement learningu zohráva dôležitú úlohu odmena, ktorá je spätnou väzbou pre agenta. Niektoré rozhodnutia zmenia stav systému, ale v novom stave nie je možné okamžite vyhodnotiť ich pozitívny alebo negatívny vplyv na riešenie úlohy. Iné rozhodnutia môžu viesť k úspešnému dokončeniu úlohy alebo naopak k prechodu do stavu, z ktorého už úlohu nie je možné úspešne vyriešiť. V oboch prípadoch by agent mal dostať spätnú väzbu, a to buď vo forme kladnej odmeny, alebo naopak zápornú odmenu, čo symbolizuje, že prechod do daného stavu nebol vhodný.

Agent na základe odmien upravuje svoj výber akcií tak, aby odmeny maximalizoval. V danom stave systému vyberá takú akciu, ktorá bude viesť k maximálnej odmene a vyhýba sa akciám, za ktoré v minulosti dostal trest v podobe zápornej odmeny. Výber akcií by mal reflektovať nie len aktuálnu odmenu, ale aj potrebu dosiahnutia dlhodobého cieľa. Agent by mal byť schopný vyberať také akcie, ktoré budú maximalizovať súčet odmien v rámci replikácie. Nastavenie efektívneho systému odmien pre konkrétnu úlohu môže byť náročné a má zásadný vplyv na rýchlosť tréningu ako aj kvalitu výslednej agentovej stratégie.

1.1.1 Princípy fungovania DQN a PPO agentov

Z hľadiska samotného fungovania reinforcement learning algoritmov je možné odlíšiť dva typy agentov. Prvou kategóriou sú Q Learning agenti, ktorí sú založení na princípoch dynamického programovania [2]. Snažia sa nájsť aproximáciu Q funkcie, na základe ktorej by sa mohli rozhodovať pri prechode medzi stavmi. Q funkcia je taká funkcia, ktorá pre každú možnú dvojicu akcie a a stavu s systému vráti ohodnotenie, ktoré predstavuje výhodnosť použitia akcie v danom stave. Keďže úlohou je získať čo najväčší súčet odmien, táto výhodnosť je reprezentovaná sumou budúcich odmien, ktoré môže agent získať, ak v stave s zvolí akciu a.

V rámci výberu akcie treba zobrať do úvahy aj množstvo prechodov medzi stavmi – v ideálnom prípade by sme chceli množstvo prechodov minimalizovať, aby sme úlohu vyriešili čo najrýchlejšie. Ak totiž dve akcie vedú k rovnakej sume odmien, ale jedna z nich k nej vedie skôr, jej použitie môže byť výhodnejšie. Preto vzťah na výpočet hodnoty Q funkcie obsahuje aj faktor zrážania γ (discount factor), ktorým sa násobí suma budúcich odmien [3]. Ide o parameter, ktorý môže nadobúdať hodnoty na intervale 0 až 1 a hovorí o tom, akú váhu prikladáme budúcim odmenám. Hodnota 0 znamená, že v rámci výpočtu hodnoty Q funkcie použijeme len aktuálnu odmenu a hodnota 1 znamená, že všetky budúce odmeny budú mať rovnakú váhu bez ohľadu na to, kedy ich získame. Hodnotu Q funkcie je možné vyjadriť nasledovne:

$$Q(s,a) = R(s,a) + \gamma \max_{a} Q(s',a) \tag{1}$$

Ak sú stav systému aj akcie diskrétne, Q funkciu je možné reprezentovať maticou, respektíve tabuľkou. V prípade, že by sme mali túto tabuľku pre danú úlohu k dispozícii, vedeli by sme ju riešiť exaktným spôsobom, lebo by sme vedeli nájsť taký prechod medzi stavmi, pri ktorom by sme dostali maximálnu odmenu voľbou takej akcie, ktorá má

najväčšiu sumu budúcich odmien. Pre väčšinu úloh Q tabuľku nemáme, a preto treba nájsť mechanizmus, ako ju vypočítať z pozorovaní v rámci prostredia. Agent môže v rámci prostredia vykonávať akcie a na základe odmien, ktoré dostane, aktualizuje hodnoty Q funkcie. Parameter α sa nazýva learning rate a hovorí o tom, s akou mierou budeme aktualizovať hodnoty Q funkcie vzhľadom na nové pozorovania. Hodnoty sa upravia podľa vzťahu:

$$Q\ (S_t, a^2) = (1 - \alpha\ ) * Q(S_t, a^2) + \alpha\ Q(s, a) \tag{2}$$

Na začiatku je tabuľka inicializovaná náhodným spôsobom, preto by agent v rámci preskúmavania prostredia v počiatočnej fáze nemal akcie voliť podľa nej. Neskôr, keď sú hodnoty relatívne dobrou aproximáciou reality, už je možné na základe tabuľky odhadnúť dobrú stratégiu, ktorú treba ešte zdokonaliť. Rozpor medzi fázou náhodného preskúmavania (exploration) a zdokonaľovania stratégie (exploitation) riešia parametre ε a ε *decay*. Hodnota ε sa volí z intervalu 0 až 1 a hovorí o tom, aký pomer akcií sa vykoná náhodne, zvyšné akcie sa vykonajú na základe hodnoty v Q tabuľke. Na začiatku sa preto ε nastaví na veľkú hodnotu blízku 1 a agent vykonáva prevažne náhodné akcie. V rámci tréningu sa hodnota priebežne znižuje o ε *decay* až po vopred stanovené minimum blízke 0, kedy už agent väčšinu akcií vykonáva na základe deterministickej stratégie.

Ďalší problém je, že stavový priestor je pre reálne úlohy obrovský, a preto aj keby sme Q funkciu vo forme matice vedeli vypočítať, trvalo by to veľmi dlhý čas a nezmestila by sa do pamäte. Najtypickejším zástupcom agenta v oblasti Q Learningu je DQN agent, ktorý sa snaží Q funkciu aproximovať neurónovou sieťou. Pre stabilizáciu tréningu používa DQN v rámci procesu učenia dve neurónové siete – target a online sieť. Do online siete sa pošle pozorovanie a výsledkom je predikcia Q hodnôt. Následne sa vypočítajú Q hodnoty podľa skutočnej odmeny, ktorú agent získal a ku nej sa pripočíta predikcia maximálnej Q hodnoty z target siete pre budúci stav. Rozdiel medzi týmito dvoma hodnotami je použitý ako loss funkcia na účely trénovania online siete (3). Raz za vopred určený počet krokov sa kópia online siete nastaví ako target sieť.

$$L(\theta) = [\ Y_{pred} - (R(s_t, a_t) + \ \gamma \max_a Q(s_{t+1}, a'))]^2 \tag{3}$$

Ak by sa sieť trénovala súčasne s tým ako agent vykonáva v prostredí akcie, medzi pozorovaniami z prostredia by bola silná korelácia, čo by mohlo viesť k tomu, že sieť sa namiesto generalizovania úlohy naučí predikovať Q hodnoty len pre konkrétnu replikáciu.

DQN agent namiesto toho ukladá pozorovania do experience replay buffera a tie sú z neho neskôr na účely tréningu náhodne vyberané [4].

Druhou kategóriou sú Policy optimization agenti využívajúci vlastnú stratégiu, prostredníctvom ktorej vykonávajú v systéme akcie [5]. Na rozdiel od Q learning agentov sa nesnažia ohodnocovať výhodnosť akcií v jednotlivých stavoch, ale celú úlohu vnímajú ako optimalizačný problém. Na základe odmien svoju aktuálnu stratégiu priebežne upravujú na takú stratégiu, ktorá bude viesť k lepším odmenám. Jednou z výhod oproti DQN je možnosť použitia spojitých akcií, keďže pri prechodoch do nového stavu nepotrebujeme vyberať z konečného zoznamu dostupných akcií takú akciu, ktorá bude mať maximálnu sumu budúcich odmien. Príkladom agenta z tejto kategórie je Proximal policy optimization.

PPO patrí do kategórie action-critic algoritmov. Výstup z neurónovej siete, ktorá určuje aktuálnu politiku správania sa agenta v prostredí, je rozdelený na dve časti – actor a critic. Výstupom z časti actor je rozdelenie pravdepodobnosti použitia jednotlivých akcií v danom stave. Critic má na výstupe hodnotu *V*. Táto hodnota by sa dala vyjadriť ako suma jednotlivých pravdepodobností vykonania akcie vynásobených budúcou sumou odmien pri výbere danej akcie *Q(a)*. Dá sa povedať, že ide o priemernú kumulatívnu odmenu, ktorú môžeme v danom stave očakávať vzhľadom na rozdelenie pravdepodobností jednotlivých akcií. Keďže stratégia prechodov medzi stavmi nie je deterministická, ale riadi sa rozdelením pravdepodobnosti, hodnotu Q funkcie pre akciu a v stave s môžeme vyjadriť podľa vzťahu:

$$Q(s,a) = R(s,a) + \gamma V(s') \tag{4}$$

V rámci tréningu agent vykonáva akcie podľa aktuálnej stratégie, a to tak, že na výstup z actor časti sa aplikuje softmax funkcia a následne sa zvolí finálna akcia podľa multinomického rozdelenia pravdepodobnosti. Za vykonanie akcie agent dostane odmenu a na základe nej je možné vypočítať výhodu A použitia danej akcie a v stave s podľa vzťahu:

$$A(s,a) = Q(s,a) - V(s) \tag{5}$$

S využitím získanej hodnoty výhody môžeme upraviť váhy neurónovej siete tak, aby akcie s malou výhodou mali menšiu pravdepodobnosť a naopak, aby boli preferované akcie s veľkou výhodou. Keďže PPO agent sa učí na základe aktuálnej stratégie, aj v tomto prípade by sme mali zamedziť korelácii dát, a to tak, že pozorovania, na ktorých sa učí, by

mali byť získavané z viacerých prostredí. Takisto sa treba vyhnúť prudkým zmenám v stratégii, ktoré môžu spôsobiť nestabilitu tréningu. Na tento účel agent využíva parameter ε, ktorý predstavuje maximálnu mieru zmeny stratégie oproti predchádzajúcej stratégii v jednom kroku.

1.1.2 Príklady použitia reinforcement learningu

Oblasť reinforcement learningu je pomerne nová, preto v súčasnosti dochádza k rozsiahlemu výskumu. Vzhľadom na charakter a množstvo dát, ktoré reinforcement learning vyžaduje, sa veľká časť výskumu týka počítačových hier. Tie poskytujú prostredie s vopred jasne definovanými pravidlami, ktoré môže bežať rýchlejšie ako reálny čas. Známym príkladom je model OpenAI Five [6], ktorý dokázal poraziť najlepších hráčov v hre Dota 2.

Existujú ale už aj príklady z výskumu pri použití na úlohách v reálnom svete. Jedným z nich je uplatnenie v oblasti astronomických pozorovaní. Teleskopy sú dôležitým nástrojom na zber údajov z vesmíru, ktoré môžu byť následne použité na objavovanie nových vesmírnych telies alebo overovanie fyzikálnych hypotéz. Nemenej dôležitú úlohu majú teleskopy aj pri sledovaní pohybu vesmírneho odpadu, vďaka čomu je následne možné zabrániť zrážkam odpadu s funkčnými objektmi na obežnej dráhe. Konkrétne pozorovania pritom vyžadujú dobré atmosférické podmienky, správne načasovanie, ako aj umiestnenie a nasmerovanie teleskopov. Na riešenie tohto problému je možné využiť práve učenie posilňovaním [7].

Zaujímavá je aj práca, ktorá sa zaoberá využitím učenia posilňovaním na trénovanie autonómnych vozidiel [8]. Myšlienka autonómneho riadenia ako taká nie je nová, avšak v minulosti sa ako stavebné bloky systémov pre autonómne riadenie používali konvenčnejšie postupy. Autori používajú simulátor CARLA. Ide o sofistikovaný simulátor, ktorý bol špeciálne vyvinutý len na trénovanie, porovnávanie a validáciu systémov autonómneho riadenia vozidiel v premávke. Obsahuje realistické situácie v rôznych prostrediach vrátane dopravného značenia a chodcov. Autori článku použili dvoch DQN agentov, ktorí sa spolu snažili naučiť stratégiu ako ovládať autonómne vozidlo. Jeden agent mal za úlohu riadiť vozidlo, pričom rozhodnutím bol smer jazdy a druhý sa staral o brzdenie. Z prostredia extrahovali viacero údajov ako vstup pre model – informáciu o stave semaforov pred vozidlom, informácie o polohe a rýchlosti vozidla, zoznam waypointov, segmentované objekty v blízkosti vozidla a hĺbkovú mapu najbližších objektov. Následne navrhli deterministický systém, ktorý rozhodoval o tom, akcia ktorého

agenta sa prioritne použije v závislosti od situácie. Napríklad ak extraktor príznakov z prostredia zaznamenal, že auto stojí na semafore na červenom svetle, aplikovalo sa brzdenie. V prípade prepnutia na zelenú, sa riadenie vozidla odovzdalo agentovi, ktorý vyberal smer jazdy. Výsledky ukázali, že natrénovaní agenti boli schopní vozidlo riadiť bez nárazu v 94% prípadov na štyroch vybraných trajektóriách pozostávajúcich zo 100 behov.

Existujú aj pokusy o uplatnenie reinforcement learningu vo finančnom sektore [9], konkrétne v oblasti obchodovania akcií. Zatiaľ čo bežné techniky v oblasti machine learningu sa snažia na základe dát z minulosti odhadnúť cenu v budúcnosti, reinforcement learning sa na vývoj ceny pozerá ako na dynamický systém. Agent môže v tomto systéme nakupovať a predávať rôzne druhy akcií a prirodzenou odmenou je zisk nadobudnutý danou stratégiou. Agent by sa mal zároveň snažiť minimalizovať rizikové investície. Zaujímavosťou je, že okrem informácie o aktuálne nakúpených akciách a vývoji cien akcií na trhoch bola vstupom do systému aj ďalšia premenná reprezentujúca sentiment správ v médiách ohľadom danej akcie. Tá bola získaná pomocou analýzy nadpisov správ v oblasti financií a investícií z daného obdobia. Autorom sa na historických dátach vývoja cien akcií medzi rokmi 2010 až 2018 podarilo natrénovať agenta, ktorý dokázal dosahovať výrazný zisk v porovnaní s priemerným výnosom súvisiacim s nákupom týchto akcií v danom období.

1.2 Sieť ResNet

Úloha riadenia autonómneho vozidla by sa dala rozdeliť do dvoch samostatných podproblémov. Jedným z nich je, aké rozhodovania robiť a tým druhým je samotná analýza obrazu. Autonómne vozidlo by malo používať ako primárny zdroj dát údaje z kamery. Kamera nahráva digitálny obraz, ktorý je reprezentovaný pixelmi. Tie sú vo svojej podstate trojrozmerná matica. Ľudský mozog sa naučil rozpoznávať základné vzory v obraze a spájať ich do zložitejších vzorov a vďaka tomu môžeme na videu jednoducho identifikovať jednotlivé objekty, ale pre počítač je aj táto úloha pomerne zložitá.

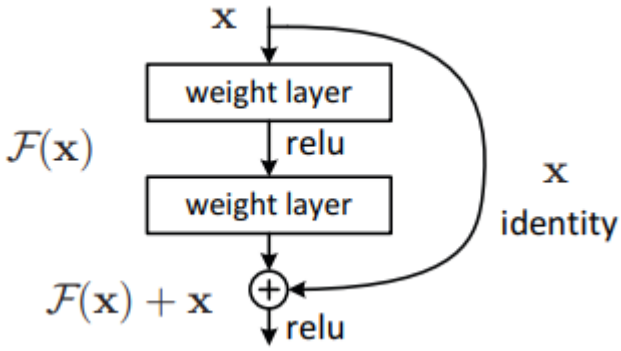
Najosvedčenejší spôsob analýzy obrazu v súčasnosti predstavujú konvolučné neurónové siete. Špecifikom analýzy obrazu v porovnaní s inými úlohami, v ktorých sa používajú iné architektúry sietí je, že je rozhodujúca poloha a blízkosť tvarov, ktoré chceme identifikovať. Konvolučné neurónové siete využívajú špeciálne konvolučné vrstvy,

ktoré dokážu efektívne spojiť informáciu z viacerých susedných pixelov alebo zo susedných vzorov na predchádzajúcej vrstve, čím identifikujú jednoduché tvary. Zreťazením viacerých konvolučných vrstiev je možné prejsť od identifikácie primitívnych tvarov až k identifikácii celých objektov.

S pridávaním konvolučných vrstiev rastie počet voliteľných parametrov, a teda by sa mal zlepšovať aj výkon neurónovej siete. Prax však ukázala, že to platí len do obmedzenej miery. Od určitého počtu konvolučných vrstiev dosahujú tieto siete horšie výsledky na trénovacej aj testovacej množine ako siete s menej vrstvami. Je to spôsobené problémom miznúcich gradientov. Tréning sietí začína obvykle s náhodnou inicializáciou váh. Signál z nultej vrstvy (teda informácie o originálnom obrázku) prechodom sieťou slabne, čo ovplyvňuje výpočet gradientov. Pri spätnej propagácii majú vstupy len veľmi malý vplyv na výstupy a takúto sieť je preto ťažké natrénovať.

Tento problém sa darí efektívne riešiť architektúre Deep Residual network [10], ktorú jej autori označujú aj ako ResNet. ResNet rieši problém slabnúcich gradientov pomocou tzv. reziduálnych spojení (residual connections), ktoré umožňujú priame preskočenie (skip connection) cez jednotlivé vrstvy v sieti. Tým sa umožňuje tok informácií a gradientov priamo od vstupu k výstupu bloku siete. To jednak zlepšuje intenzitu gradientov a pre sieť je jednoduchšie sa natrénovať, pretože ak nejaký blok neprispieva k riešeniu danej úlohy, pre mechanizmus učenia je jednoduchšie daný blok odignorovať a použiť vstup z predchádzajúceho bloku v nezmenenej podobe.

ResNet dosahuje v testoch lepšie výsledky ako predchádzajúce siete s inou architektúrou a jeho výkon rastie so zvyšujúcim sa počtom vrstiev. Výhodou ResNetu je aj fakt, že dosahuje dobré výsledky na viacerých typoch úloh, čo môže byť vhodné aj pre riešenie našej úlohy. Okrem klasifikácie, kedy je potrebné zaradiť objekt na obrázku do jednej z tried, sa ResNet používa ako extraktor príznakov na ďalších úlohách, ako sú detekcie a segmentácia objektov. Pri riadení autonómneho vozidla je táto vlastnosť dôležitá, pretože nezáleží len na tom, aký objekt sa na obraze nachádza, ale aj v ktorej časti obrazu sa nachádza.



Obrázok 2 – Základná architektúra siete ResNet **[10]**

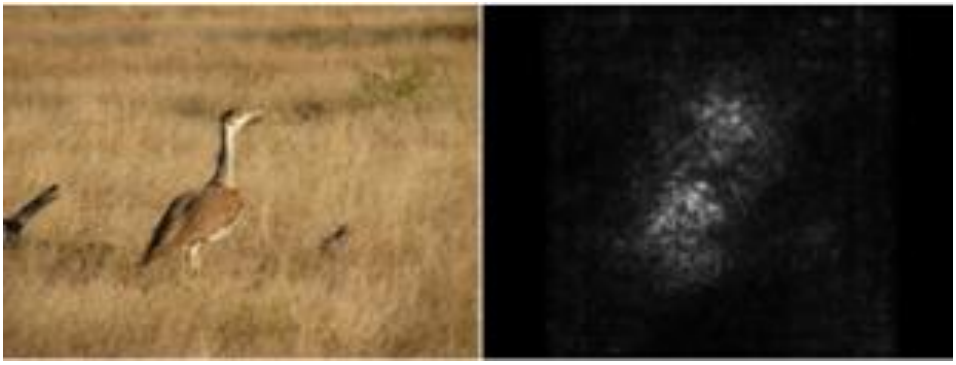
1.3 Vanilla gradient

Väčšina agentov v rámci učenia posilňovaním využíva hlboké neurónové siete ako nástroj rozhodovania, pričom sa medzi sebou líšia hlavne v spôsobe, ako neurónovú sieť natrénovať. Hlavnou výhodou neurónových sietí je schopnosť učiť sa, ale z hľadiska vnútorného fungovania ide o komplikovaný mechanizmus. Pre človeka je možné pochopiť, ako neurónová sieť funguje z teoretického hľadiska a rozumieť tomu, ako a prečo fungujú princípy, ktoré všeobecná neurónová sieť využíva na to, aby sa naučila riešiť daný problém. Ak by sme chceli ale pochopiť, ako funguje konkrétna neurónová sieť pri riešení vybranej úlohy a porozumieť tomu, prečo majú jednotlivé váhy konkrétne nastavené hodnoty, tak už pre siete s relatívne malou hĺbkou a malým počtom vstupov by to začal byť problém. Pri veľkom množstve vstupov a zložitých architektúrach je len skúmaním číselných hodnôt váh pre bežného človeka prakticky nemožné povedať, ako presne daná sieť funguje a prečo sa tak rozhoduje. Dá sa povedať, že funguje ako čierna skrinka, čo môže byť pre niektoré druhy aplikácií problematické.

Existuje preto niekoľko metód, ktoré sú zamerané na pokus o vysvetliteľnosť fungovania neurónových sietí. Jednou z takýchto metód je Vanilla gradient [11] [12]. S využitím tejto metódy je možné získať takzvanú saliency mapu, často označovanú aj ako heat mapu. Ide o maticu s identickými rozmermi ako je rozmer vstupov do neurónovej siete. Každá hodnota v tejto matici predstavuje relatívny vplyv daného vstupu na výsledné rozhodnutie neurónovej siete. Ak je nejaká hodnota v saliency mape nulová, znamená to, že daný vstup nemal žiadny vplyv na rozhodnutie. Naopak, veľká hodnota znamená, že vstup mal zásadný vplyv na vybraný výstup z neurónovej siete. Pre prípad reinforcement learningu je výstupom akcia, ktorú agent v prostredí vykoná. Vanilla gradient by nám mal

vedieť povedať, aký vplyv mali jednotlivé zložky stavu systému, ktoré boli posielané do neurónovej siete, na výber danej akcie.

Medzi výhody metódy Vanilla gradient patrí jej jednoduchosť. Neurónové siete sa trénujú s využitím metódy, ktorá sa volá spätná propagácia (backpropagation). Najskôr sa pošlú do neurónovej siete vstupné hodnoty a na základe aktuálne nastavených váh neurónov sa vypočítajú hodnoty výstupu. Následne sa kvôli aktualizácii váh jednotlivých neurónov vypočíta chybová funkcia a na základe nej sa počítajú gradienty od výstupnej vrstvy smerom k vstupom, pričom tento proces za bežných okolností zastaví na prvej vrstve. Tá je totiž pre spätnú propagáciu poslednou, kde chceme aktualizovať váhy neurónov. Ak však vypočítame gradienty aj voči nultej vstupnej vrstve, dostaneme hodnoty gradientov výstupov voči vstupom a zistíme tak, aký vplyv mali jednotlivé hodnoty na rozhodnutie. Príklad takejto saliency mapy je možné vidieť na nasledovnom obrázku.



Obrázok 3 – Saliency mapa na klasifikačnej úlohe **[11]**

1.4 Robotický operačný systém ROS 2

ROS2 je framework, ktorý bol primárne vytvorený na zjednodušenie a zjednotenie komunikácie medzi viacerými hardwarovými elementmi, z ktorých sa skladajú robotické systémy [13]. Autonómne vozidlo je možné považovať za robotický systém. Každý robotický systém má viacero samostatne fungujúcich prvkov, napríklad kameru, motor, ktorý ovláda kolesá, mechanizmus na natáčanie kolies, senzory teploty, Raspberry Pi a podobne.

V minulosti bolo potrebné vytvoriť zložitý program, ktorý bude komunikovať s drivermi týchto komponentov alebo priamo s týmito elementami. Tento prístup môže byť zložitý z hľadiska vývoja a ak za ním nestojí premyslený návrh architektúry systému a častí systému je veľa, systém sa stane ťažkopádnym a horšie rozšíriteľným. ROS 2 preto

poskytuje Data Distribution Service, teda jednotný spôsob komunikácie medzi elementmi robota.

ROS 2 umožňuje vytvárať Nody. Jeden Node by mal predstavovať samostatne fungujúci element v robotickom systéme, kam môže vývojár robota sústrediť funkčnosť zameranú na jeden konkrétny aspekt – napríklad riadenie vozidla. Dôležité je, že Node dokáže efektívne komunikovať s inými Nodmi, pričom nemusí byť nijako oboznámený s implementačnými detailami ostatných Nodov a nie je zaťažený ani implementačnými detailmi samotnej komunikácie. ROS 2 na to používa návrhový vzor publisher – subscriber. Ak chce Node odovzdať nejakú informáciu iným prvkom systému, vytvorí Topic. Každý Topic je jednoznačne identifikovaný svojím názvom a dátovým typom, ktorý podporuje – Node kamery by tak mohol vytvoriť „image_topic“ s dátovým typom Image. Vždy keď kamera zaznamená nový snímok, odošle ho do tohto Topicu. Iný Node sa dokáže subscribnúť na existujúci Topic a vždy keď do takéhoto Topicu príde nová správa, dokáže ju zaznamenať a spracovať. Dokumentácia Rosu odporúča používať jeden z veľkého množstva vstavaných dátových typov pre Topicy, čím sa zaistí maximálna kompatibilita s existujúcimi open-source Nodmi a hardwarovými komponentami.

Výhodou takto navrhnutej komunikácie je jednoduchá rozšíriteľnosť a bezpečnosť, keďže každý Node sa v systéme správa ako samostatne fungujúci zapuzdrený objekt a rozhranie komunikácie je jasne definované. ROS2 je takisto platformovo nezávislý, takže existujúce skripty je možné používať na robotoch nezávisle od toho, aký operačný systém a architektúru procesora používajú.

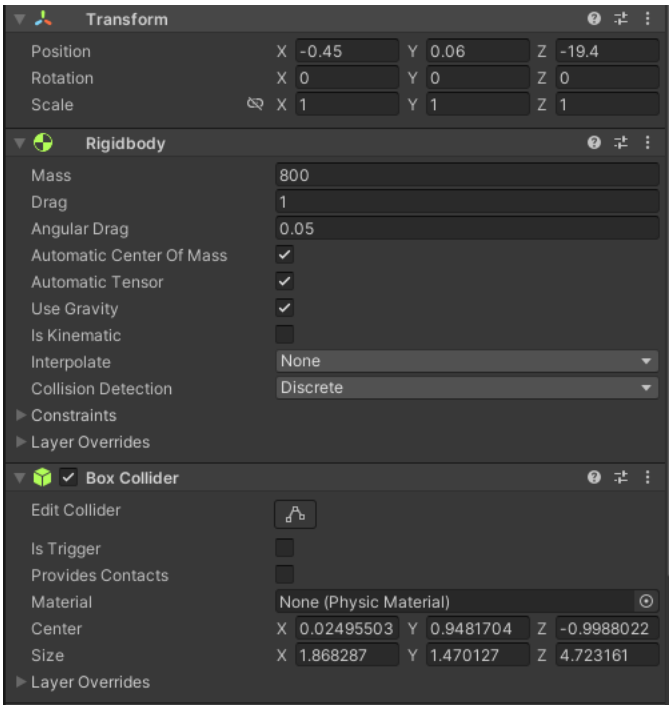
1.5 Unity engine

Unity je engine, ktorý bol primárne vytvorený na vývoj hier. Má však niekoľko vlastností, vďaka ktorým ho je možné použiť aj ako simulačné prostredie pre tréning umelej inteligencie na úlohu, ktorou sa zaoberá táto diplomová práca. Moderné 3D hry majú realistickú vizuálnu stránku. Prostredie tak môže byť relatívne vernou kópiou skutočného prostredia, kde by sa pohybovalo autonómne vozidlo a senzorické dáta v podobe údajov z kamery sa budú podobať skutočným dátam. Ďalšou výhodou Unity je podpora nástrojov, ktoré pridajú objektom relatívne realistické fyzikálne správanie, pričom sa programátor nemusí priamo starať o implementačné detaily na úrovni fyzikálnych zákonov. Reprezentácia objektov v rámci prostredia je oddelená od ich správania, vďaka

čomu je možné rýchlo a efektívne implementovať zmeny a pridávať správanie novým objektom. Keďže ide o jeden z najpoužívanejších herných enginov, je k dispozícií obrovské množstvo dokumentácie a návodov, čo prispieva k zjednodušeniu vývoja a ľahšiemu riešeniu problémov. Okrem Unity existujú aj iné enginy, ktoré by boli vhodné na simuláciu autonómneho vozidla v rámci jeho tréningu. Jedným z nich je Gazebo Sim [14].

1.5.1 Štruktúra projektu

Projekt v Unity sa skladá z niekoľkých základných prvkov. Jedným z nich je Scéna - ide o prostredie, v ktorom sa môžu pohybovať herné objekty a interagovať medzi sebou. V rámci vývoja hier by jedna scéna mohla predstavovať jeden svet alebo jeden level. Unity poskytuje nástroje ako za behu aplikácie prepnúť z jednej scény na inú. Do scén je možné umiestňovať Game Objecty, ktoré samé o sebe nemajú žiadne predvolené správanie. Ich správanie a ďalšie vlastnosti je možné zadefinovať pomocou Komponentov. Komponent je trieda, ktorá zapuzdruje určité správanie a vlastnosti herných objektov a dedí z triedy MonoBehaviour. Unity má niekoľko preddefinovaných komponentov, ktoré je možné použiť a v prípade potreby môže programátor vytvoriť vlastný komponent a použiť ho rovnako ako tie predvolené.



Obrázok 4 – Ukážka atribútov Unity komponentov

Základným komponentom je Transform – udržiava v sebe informácie o pozícii, rotácii a veľkosti objektu.

Medzi dôležité komponenty pre riešenie našej úlohy patrí Rigidbody, ktorý objektu pridáva fyzikálne správanie. Ak objektu pridáme Rigidbody, začne naň vplývať gravitácia a Unity bude reagovať na nárazy tohto objektu do iných objektov, ktoré majú tiež RigidBody. Prostredníctvom tohto komponentu je možné objektu nastaviť základné fyzikálne vlastnosti ako hmotnosť, mieru spomalenia v dôsledku odporu vzduchu alebo v dôsledku otáčania a ďalšie podobné vlastnosti.

Komponent BoxCollider slúži na zadefinovanie ohraničenia objektu pomocou kvádra. Do Unity je možné vložiť komplexný 3D model ako vizuálnu podobu Game Objectu. Ten môže mať veľmi komplikovaný tvar a bolo by výpočtovo náročné počítať fyzikálne interakcie medzi objektami pomocou tohto tvaru. Namiesto toho sa jeho vizuálna reprezentácia pre potreby výpočtov fyzikálnych interakcií nahradí aproximáciou – jedným alebo viacerými BoxCollidermi, čo vedie k jednoduchším výpočtom kolízií medzi objektami. RigidBody s objektom vždy počíta tak, akoby mal tvar svojich Box Colliderov a nie skutočný tvar, ktorý sa zobrazuje.

WheelCollider predstavuje koleso. Pre vernú reprezentáciu vozidla je potrebné vozidlu pridať wheelCollidery a nim je následne možné nastaviť priemer a hmotnosť. Pohyb vozidla je možné v Unity docieliť tým, že sa wheelColliderom nastaví aktuálny moment sily otáčania (motor torque), aktuálna sila brzdenia, samovoľná rýchlosť brzdenia, keď nie je aplikovaný moment otáčania, aktuálne natočenie v stupňoch a ďalšie vlastnosti.

Okrem použitia Unity komponentov je možné zadefinovať vlastný komponent ako triedu v jazyku C#, ktorá je potomok triedy MonoBehaviour. V rámci kódov vlastného komponentu je možné sa odkazovať na iné game objecty a pristupovať k ich komponentom. Vstavané komponenty majú okrem správania, prejavujúceho sa v rámci behu simulácie, často preddefinovanú skupinu atribútov a funkcií, ktoré je možné využiť pri programovaní vlastného komponentu.

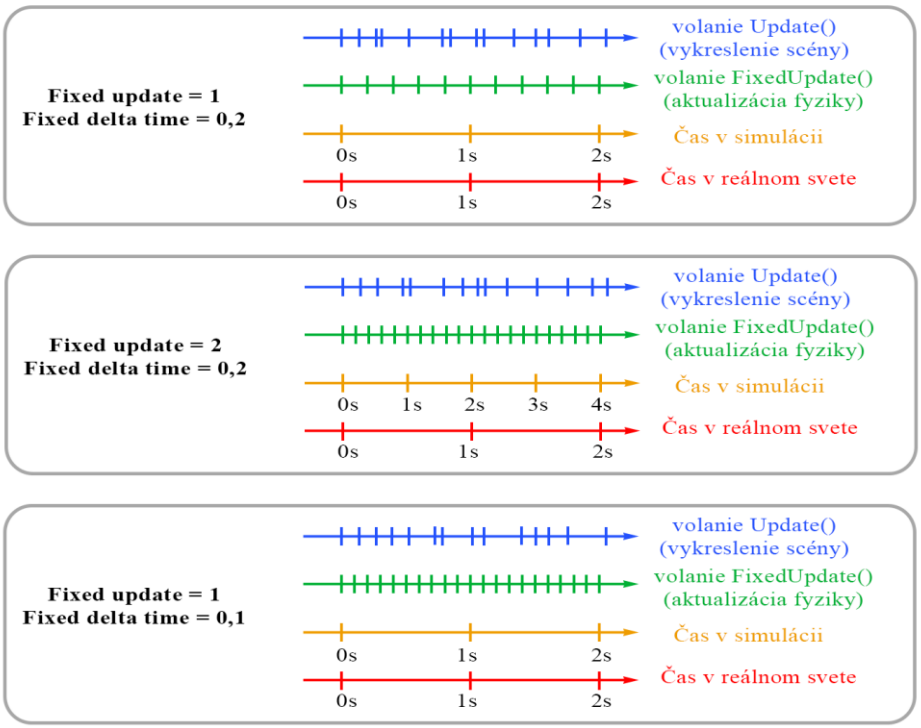
1.5.2 Koncepty času

Ak chceme Unity použiť ako nástroj na efektívne trénovanie agentov, simulácia by mala v rámci tréningu bežať rýchlejšie ako v reálnom čase. Preto je potrebné sa oboznámiť s tým, aké koncepty času sa v Unity používajú a ako fungujú [15].

Plynutie času v herných enginoch sa musí pre používateľa javiť ako spojité, ale z implementačného hľadiska bude samozrejme vždy diskrétne. Vo všeobecnosti je možné plynutie času v simulácii prostredia implementovať ako cyklus, pričom v každom volaní cyklu sa aktualizujú polohy objektov podľa stavu, v ktorom sa nachádzajú – napríklad podľa síl, ktoré na ne pôsobia a podľa interakcie s ostatnými objektami. Po aktualizácii polôh sa scéna zobrazí. Takáto jednoduchá implementácia má však problém – je hardwarovo závislá, a teda keby sme takto implementovali herný engine, na výkonnejšom počítači by čas v hre plynul rýchlejšie ako na pomalšom. Druhý problém je, že vyrenderovanie scény je v porovnaní s aktualizáciou polôh objektov pomerne drahá operácia a scénu nemusíme potrebovať zobrazovať vždy, keď potrebujeme vykonať aktualizáciu fyziky. Unity ako moderný herný engine je preto implementovaný o niečo sofistikovanejšie a ponúka vývojárom niekoľko prostriedkov, ktorými vedia ovplyvniť rýchlosť plynutia času z viacerých hľadísk.

Fixed delta time ovplyvňuje, koľkokrát za 1 sekundu herného času sa zavolá základný herný cyklus, v ktorom dôjde k aktualizácii fyziky. Defaultná hodnota je 0,02, čo znamená, že aktualizácia polôh a ostatných atribútov objektov sa vykoná 50x za sekundu. Tento atribút zároveň nedokáže ovplyvniť rýchlosť plynutia času v hre ako by sa mohlo zdať, ovplyvňuje len presnosť fyziky. Ak by sme ho napríklad nastavili na 0,01, fyzika sa bude aktualizovať 100x za sekundu, ale v polovičných krokoch.

Na ovplyvnenie rýchlosti plynutia herného času voči reálnemu času je možné použiť Time Scale. Defaultná hodnota je 1 a znamená, že 1 časová jednotka simulačného času bude zodpovedať 1 časovej jednotke v reálnom svete. Ak by sme TimeScale nastavili napríklad na hodnotu 5, znamená to, že za 1 sekundu reálneho času ubehne v simulácii 5 sekúnd. Pri defaultnom Fixed delta Time 0,02 a tomto TimeScale to znamená, že sa herný cyklus aktualizujúci fyziku vykoná 250krát za sekundu. Treba si preto uvedomiť, že vysoká kombinácia hodnôt TimeScalu v kombinácii s nízkou hodnotou FixedDeltaTimu by mohla spôsobiť, že počítač nebude mať dostatočný výkon, aby simulácia bežala tak rýchlo ako požadujeme. Ak je výkon nedostatočný, presnosť fyziky zostáva garantovaná, ale zmení sa rýchlosť plynutia simulačného času voči reálnemu svetu.



Obrázok 5 – Príklady možnosti práce s časom v Unity

Target FPS je tretia dôležitá premenná, ktorá ovplyvňuje renderovanie snímok za sekundu. Defaultná hodnota je -1, čo pre beh hry na platforme PC znamená, že sa vyrenderuje toľko snímok, koľko počítač výpočtovo zvládne. Jej zmenou dokáže programátor ovplyvniť len vrchný limit. Aktualizácia fyziky sa tak vždy vykonáva prioritne tak, aby sa čo najviac priblížila hodnotám zadaným pomocou Time Scale a Fixed Delta Time, zatiaľ čo snímky sa renderujú len vtedy, keď na to zostáva čas.

Z hľadiska implementácie simulácie sú v komponentoch dostupné dve hlavné metódy, kde je možné umiestniť funkcionalitu súvisiacu s plynutím času. Metóda Update sa volá predtým, ako sa renderuje snímka – jej pravidelnosť volania voči hernému času teda vôbec nie je garantovaná. Pre funkčnosť, ktorá vyžaduje túto garanciu, slúži metóda FixedUpdate, ktorá sa volá pri každom aktualizovaní fyziky.

1.5.3 Senzory

Pre účely použitia tejto práce je nutné, aby simulácia bežala rýchlejšie ako reálny čas, avšak zároveň potrebujeme dosiahnuť, aby sme agentovi v procese učenia vždy poslali snímku zobrazujúcu najaktuálnejší stav scény. To by v rámci bežných mechanizmov nebolo garantované, keďže scéna sa renderuje len vtedy, keď na to ostáva čas. Unity však tento problém rieši a ponúka knižnicu MLAgents.Sensors [16], ktorá okrem iného

obsahuje aj mechanizmy na manuálne vyrenderovanie pohľadu z kamery. Aktuálna snímka zo scény sa dá takto získať aj v rámci volania FixedUpdatu, pričom je možné zadať rozlíšenie, v ktorom je potrebné obrázok vyrenderovať. Parametrizácia tejto metódy značným spôsobom šetrí strojový čas, pretože agentovi by mali vo väčšine prípadov stačiť dáta v nižšom rozlíšení, keďže sa snaží len riešiť danú úlohu a nepotrebuje zážitok z hry ako človek. Počas tréningu nás teda vôbec nemusí zaujímať, ako často a či vôbec dochádza ku zavolaniu Udpatu, a teda vyrenderovaniu scény v plnom rozlíšení, keďže tieto dáta agent nepoužíva.

1.5.4 Prepojenie s ROS 2

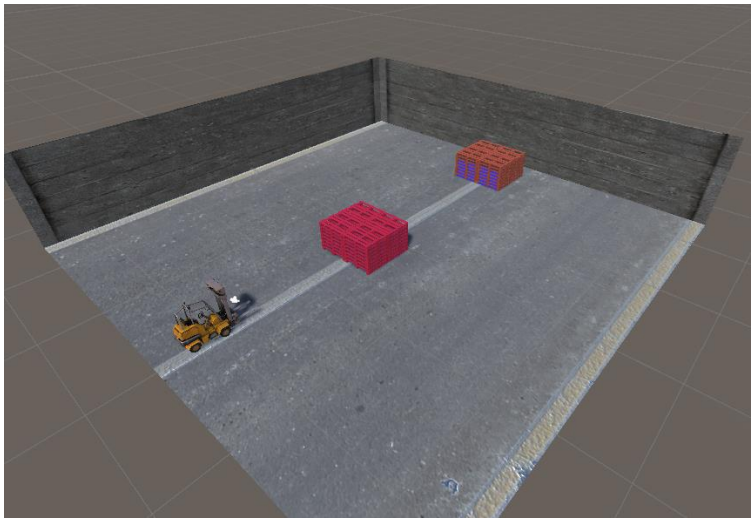
Pre dosiahnutie prepojenia Unity s Robotickým operačným systémom je možné využiť napríklad open source balík Ros2 For Unity [17]. Ten zapuzdruje vytváranie ROS2 Nodov a komunikáciu prostredníctvom Topicov do objektovej reprezentácie. Okrem toho, že podporuje všetky štandardné ROS2 formáty správ, je aj rýchly a do Unity sa dá pridať rovnakým spôsobom ako ostatné assety. Rozhodli sme sa ho preto použiť v našej implementácii.

2 Implementácia

Z hľadiska implementácie je možne túto prácu rozdeliť do viacerých častí. Prvá časť je prostredie v Unity, kde sa má agent pohybovať a vykonať jednoduchú úlohu. Táto časť je implementovaná v jazyku C#. V rámci prostredia dochádza k výpočtu odmeny a zberu štatistických dát, ktoré priamo súvisia so simuláciou. Druhá časť je samotný agent a architektúra siete, ktorú agent využíva. Dá sa povedať, že agent funguje ako samostatný systém a nevie o tom, že simulácia prebieha v Unity. Veľmi dôležitou časťou je teda aj tretia časť, ktorou je komunikácia agenta s Unity prostredím a potreba synchronizácie. Jadro agentov prispôsobených pre účely tejto práce pochádza od vedúceho práce.

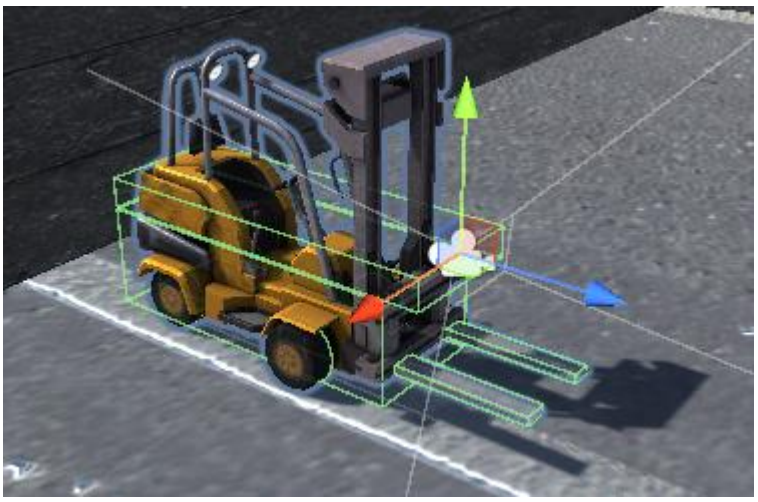
2.1 Návrh prostredia v Unity

Rozhodli sme sa, že autonómne vozidlo v Unity bude predstavovať vysokozdvižný vozík. Aby bol cieľ pre agenta relatívne jednoducho identifikovateľný, boli vybraté jednoduché textúry okolia, kde už aj pri nižšom rozlíšení obrazu je možné rozoznať jednotlivé objekty. V priebehu toho, ako boli na prostredí vykonané experimenty s natrénovaním agenta, vznikli nové požiadavky a dochádzalo k zmenám prostredia a správania niektorých objektov. Týmto zmenám sa budeme detailnejšie venovať v časti s experimentami.



Obrázok 6 – Ukážka prostredia v Unity

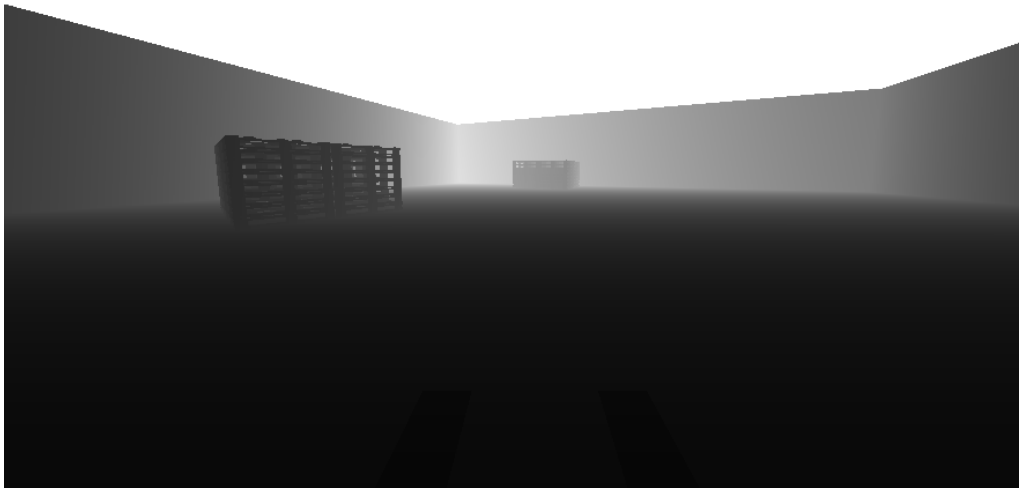
Ovládanie vozíka obstaráva WheelController. Je navrhnutý tak, aby bolo možné vozík ovládať šípkami napríklad pre potreby overenia správneho nastavenia fungovania systému odmien. Okrem toho dokáže táto trieda spracovať aj požiadavky od agenta.



Obrázok 7 – Box collidery autonómneho vozidla a umiestnenie kamery

Vozík obsahuje kameru umiestnenú v priestore nad lyžinami, tak aby videla pred vozíka a obraz zachytil aj samotné lyžiny. Má nastavené RigidBody a BoxCollidery (vyznačené zelenou farbou na obrázku), voči ktorým sa počíta fyzikálna interakcia vozíka s okolitými objektami, napríklad pri nárazoch do stien.

Pre niektoré experimenty sme potrebovali z kamery získať obraz podobný tomu, aký by produkoval LIDAR senzor. Ten v skutočnom svete funguje na princípe vyslania laserových bodov do priestoru. Na základe odrazov tohto mračna bodov vie odhadnúť vzdialenosť objektov v rámci priestoru a ich skutočnú vzdialenosť vo vzťahu ku kamere. Unity aktuálne nepodporuje nastavenie kamery, ktoré by umožňovalo jednoduchým spôsobom získať obraz, kde by pixely reprezentovali ich relatívnu vzdialenosť od kamery. V rámci shaderu, ktorý sa za bežných okolností používa pre špeciálne efekty, je možné zachytiť informáciu o hĺbke, ktorá sa počíta pre účely renderovania. Pri použití takto upraveného shaderu sa renderuje v režime RGBD, teda výstupom je bežný obraz a na štvrtom rozmere hĺbková mapa. Nasledujúci obrázok znázorňuje len hĺbkovú mapu pre lepšiu predstavu, ako obraz z nej vyzerá.



Obrázok 8 – Obraz hĺbkovej mapy z kamery autonómneho vozidla

2.1.1 Riziká trénovania agentov v Unity

V rámci prvých experimentov s Unity sme narazili na zásadný problém, kedy ihneď po spustení simulácie prostredia začala na počítači lineárne narastať spotreba operačnej pamäte, čo naznačuje memory leak. Triedy, ktoré zapuzdrujú senzorické dáta ako napríklad Vector3 z ROS2 sú v C# označené interfacom IDisposable, čo znamená, že vstavaný správca pamäte ich neodstráni a o uvoľnenie pamäte je potrebné sa postarať manuálne zavolaním metódy Dispose alebo využitím klauzuly using. Tento prístup by bol ale zbytočne pomalý, pretože by to znamenalo, že pri každom poslaní správy agenta by bolo potrebné vytvoriť novú inštanciu tejto triedy a následne ju zahodiť. Lepší prístup je vytvoriť len jednu inštanciu triedy Vector3 a následne pri každom poslaní správy agenta zmeniť hodnoty atribútov zodpovedajúce aktuálnemu stavu prostredia.

Druhý a oveľa ťažšie identifikovateľný memory leak sa týkal objektu Texture, do ktorého zapisuje virtuálny senzor kamery. Tento objekt nie je označený ako IDisposable, ale aj napriek tomu ho memory manager v Unity ignoruje. Na začiatku sme ho vytvárali pri každom získaní dát zo senzoru kamery, čo pri vyššom rozlíšení viedlo k rapídnemu nárastu spotreby operačnej pamäte (nad 100GB) – po vyčerpaní swapového priestoru na disku nedošlo k pádu aplikácie, ale k pádu celého operačného systému na chybe System out of memory exception.

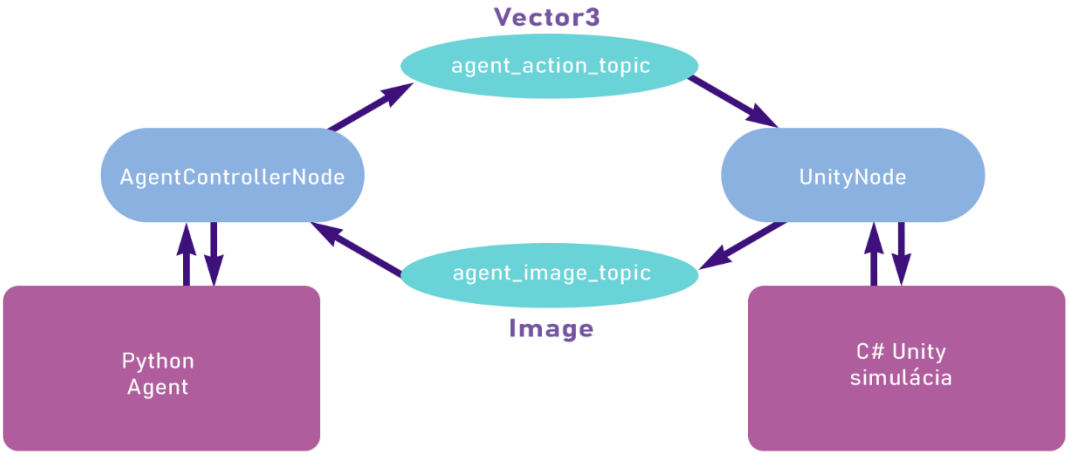
Memory manažment je pomerne častým problémom vývoja v Unity a pri prehliadnutí krátkej poznámky v dokumentácii môže dôjsť k takýmto problémom, pričom

konkrétne miesto chyby je ťažké odhaliť. Tento problém sa dá pri simuláciách čiastočne obísť (prípadne sa dá získať čas na vyriešenie problému, počas ktorého už môže simulácia fungovať), a to reštartovaním celej scény každých X replikácií. Reštartovanie scény uvoľní väčšinu objektov z pamäti, a to aj takých, ktoré by memory manager v rámci behu nedealokoval. Pre simulácie s dĺžkou behu nad 10 hodín odporúčame raz za čas preventívne reštartovať scénu aj keď boli všetky problémy s pamäťou vyriešené. Ak sme to neurobili a nechali sme celú simuláciu bežať na jednej scéne, niekedy sa stalo, že po pokuse o vypnutie veľmi dlhej simulácie Unity zamrzlo a vyvolalo pád operačného systému.

Ďalšou nástrahou je systém ukladania zmien v scéne v Unity. Tento proces nemá tradičný autosave, takže každú zmenu je potrebné uložiť manuálne. Zmeny sa priebežne ukladajú do dočasného súboru, tento sa ale automaticky vymaže, ak sa Unity otvorí po páde ešte predtým, ako užívateľ tento súbor manuálne zálohuje.

2.2 Komunikácia agenta s prostredím pomocou ROS 2

Z hľadiska simulácie v Unity je komunikácia vozidla s agentom obsiahnutá v triede AgentController. Každému autonómnemu vozidlu, ktoré má ovládať agent, stačí pridať túto triedu ako komponent. Triedu sme implementovali tak, že na začiatku sa vytvorí Node reprezentujúci vozidlo. Pomocou volania metódy createPublisher sa následne vytvorí topic, do ktorého bude Unity posielať obrazové dáta z kamery. Tento topic sa bude pre agenta javiť ako skutočný kamerový senzor, odkiaľ si môže čítať dáta. Následne sa vytvorený Node subscribne na Topic, od ktorého očakáva, že bude dostávať akcie pre autonómne vozidlo. Tento Topic vytvára reprezentácia agenta v Pythone. Keď do agent_action_topicu príde nová akcia, vyvolá to spustenie metódy SpracujAkciu. Vyvolanie tejto metódy by malo spôsobiť nastavenie akcelerácie a natočenia kolies pre aktuálne vozidlo podľa akcie, ktorú poslal agent. Nie je to však možné vykonať priamo, pretože v Unity nie je možné meniť niektoré atribúty mimo vlákna, ktoré aktualizuje fyziku alebo vykresľuje scénu, teda mimo Updatu alebo FixedUpdatu. Akcia, ktorú agent poslal sa teda zapamätá do atribútov agenta a takisto sa zapamätá, že došlo k prijatiu novej správy a bude ju potrebné spracovať. Pri nasledovnom volaní FixedUpdatu sa údaje prijaté od agenta prenesú do WheelControllera, a teda pohnú autonómnym vozidlom.



Obrázok 9 – Komunikácia agenta pomocou ROS2

Spracovanie obrazových dát agentom na vytvorenie príslušnej akcie môže byť pomerne drahá operácia a takisto je zbytočné renderovať scénu a posielať ju agentovi, ak ešte nestihol odpovedať na poslednú fotku scény, ktorú sme mu predtým poslali. Bolo preto potrebné vytvoriť mechanizmus, ktorý by dokázal synchronizovať komunikáciu medzi prostredím a agentom.

V rámci komunikácie agenta s prostredím je možné nastavovať dve hlavné konštanty. Pošli raz za toľkoto fixed updatov hovorí o tom, koľko časových jednotiek musí prejsť na to, aby sme agentovi poslali ďalšiu správu. Počet fixed updatov sa začína rátať od momentu, kedy sme agentovi poslali predchádzajúci stav systému. Agentovi pošleme snímok scény v prípade, že prešiel dostatočný počet fixed udpatov od poslania predchádzajúceho snímku a zároveň, že na predchádzajúci snímok už dorazila odpoveď v podobe akcie, ktorú je potrebné vykonať. Druhá konštanta je počet fixed updatov pre platnosť správy. Agentova posledná akcia by totiž mala mať vopred stanovenú maximálnu platnosť. Keďže z výpočtových dôvodov nie je reálne zisťovať od agenta akciu pre každú časovú jednotku, akcia, ktorú pošle, trvá toľko časových jednotiek, na akú hodnotu je nastavená táto konštanta. V reálnom svete by sa mohlo stať, že dôjde k poruche riadiaceho systému a ak by autonómne vozidlo dlhý čas vykonávalo poslednú akciu, mohlo by naraziť do steny. V rámci tréningu beží agent aj prostredie v Unity ako dve samostatné aplikácie a môže sa stať, že operačný systém agenta pozastaví a správa nedokáže doraziť včas. Ak by vozidlo pokračovalo v poslednom pohybe, vyvolalo by to neobjektivitu v rámci tréningu, pretože by sa aplikovala akcia na iný stav systému, než na aký reagoval agent. Ak teda správa nestihne doraziť včas, motor prestane aplikovať silu na kolesá

z predchádzajúcej akcie. Pri praktickej aplikácii na skutočnom autonómnom vozidle by zrejme bolo treba pridať ešte tretiu časovú konštantu, a to okamih, od ktorého by malo vozidlo začať brzdiť, ak agent nepošle novú správu. Pomocou tohto synchronizačného mechanizmu je garantované, že vozidlo bude vykonávať len akciu platnú pre daný stav.

Agent musí v rámci tréningu dostávať okrem snímky zo scény aj odmenu a informáciu o tom, či replikácia skončila (napríklad v pozitívnom zmysle splnením úlohy alebo v negatívnom zmysle narazením do steny). Keď hra skončí, prostredie sa musí reštartovať do pôvodného stavu. Informácia o skončení replikácie má pre agenta vysokú dôležitosť, preto bolo potrebné v rámci synchronizácie ošetriť, aby sa nestalo, že sa prostredie reštartuje skôr, ako dostane agent informáciu o konci replikácie. Ak dôjde ku ukončeniu replikácie, prostredie pošle správu agentovi a čaká na potvrdenie – až keď dorazí potvrdenie, dôjde k reštartu scény a agentovi sa pošle počiatočná snímka z novej replikácie.

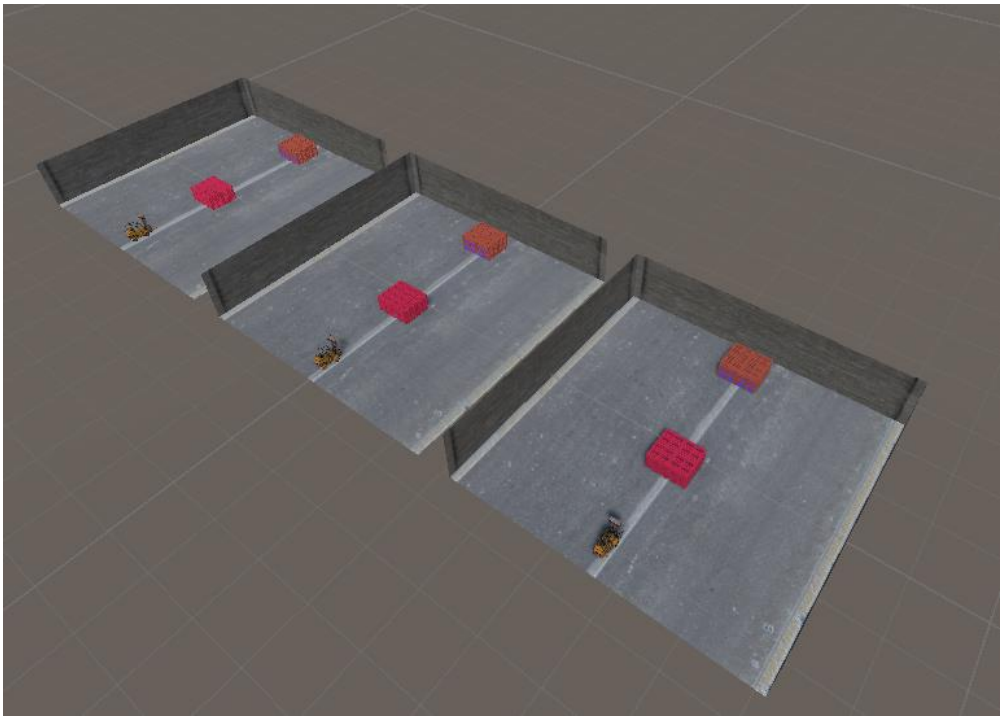
Z pohľadu Agenta funguje komunikácia podobným spôsobom. Na začiatku vytvorí agent vlastný Node a subscribne sa na agent_image_topic. Keď do tohto topicu Unity odošle snímku vo formáte Image, agent ju transformuje na použitie pre vstup do neurónovej siete. Textúra má lepšiu kvalitu, keď sa z Unity posiela vo forme PNG namiesto JPG, avšak alfa kanál nemá pre agenta prínos, takže sa odstráni (okrem experimentu s hĺbkovou mapu) a takto upravený obrázok sa transformuje na tensor. Pomocou konkrétnej implementácie aktuálne používaného agenta dôjde k transformácii stavu na index akcie a následne prípadne aj k úprave agentovej stratégie podľa získanej odmeny pre predchádzajúcu akciu. Z indexu novej akcie sa vytvorí pohybový objekt Vector3 a ten sa následne odošle do agent_action_topicu, z ktorého je spracovaný prostredím.

Prostredie Unity prečíta Vector3 a následne nastaví WheelControlleru natočenie kolies ako hodnota súradnice X vynásobená maximálnym natočením kolies a krútiaci moment motora aplikovaný na hnaciu nápravu kolies ako hodnota súradnice Y krát maximálne zrýchlenie motora autonómneho vozidla.

2.2.1 Paralelizácia s využitím viacerých prostredí súčasne

Pre agenta PPO je dôležité, aby dáta, na ktorých sa učí, pochádzali z aktuálnej stratégie. Jeho implementácia, ktorú sme sa rozhodli využiť, pracuje s viacerými Workermi súčasne. Vnútorne agent funguje tak, že postupne spracúva stavy systémov jednotlivých

Workerov v cykle. Vďaka tomu vie lepšie vyhodnotiť účinnosť aktuálnej stratégie vo viacerých situáciách v rámci prostredia. Preto sme sa rozhodli vytvoriť pre PPO agenta systém komunikácie, v rámci ktorého bude možné ovládať viacero autonómnych vozidiel – každé v samostatnom prostredí.



Obrázok 10 – Ukážka paralelných prostredí

Docielili sme to tak, že na začiatku sa vytvorí toľko AgentControllerNodov, koľko autonómnych vozidiel chceme ovládať súčasne. Každý Node si vytvorí vlastný agent_action_topic a subscribne sa na agent_image_topic príslušného vozidla. Samotný objekt PPO agenta, ktorý sprostredkuje tréning a výber akcií pre stav, sa otvorí v novom vlákne a pomocou synchronizačných prostriedkov postupne spracúva stavy systému z jednotlivých autonómnych vozidiel. Vďaka tomu môže mať mechanizmus tréningu prístup k dátam zo všetkých autonómnych vozidiel, ale komunikácia agenta voči jednotlivým vozidlám zostáva oddelená.

Výhodou tejto implementácie je aj efektívnejšie rozloženie využitia systémových prostriedkov. Unity dokáže pracovať s viacerými jadrami procesora súčasne, zatiaľ čo agentovým úzkym hrdlom je neurónová sieť, ktorá vyžaduje najmä paralelné hardwarové prostriedky, teda grafickú kartu. Pri jednom vozidle môže nastať situácia, že agent bude čakať zopár milisekúnd, kým sa vozidlo dostane do nového stavu a kým mu z tohto stavu dorazí správa cez ROS2. Pridaním viacerých vozidiel je možné tento čas využiť na to, aby sa spracovali stavy z ostatných vozidiel, ktoré sa trénujú súčasne. Vďaka spôsobu, akým

Unity pracuje s vláknami, pridanie viacerých vozidiel na viacjadrovom procesore nemusí viesť k spomaleniu simulácie. Vhodný počet paralelne trénovaných vozidiel v rámci tréningu odporúčame nastavovať podľa zaťaženia hardwarových prostriedkov v správcovi úloh alebo inom monitore aktivity podľa konkrétneho operačného systému, na ktorom tréning prebieha. My sme sa rozhodli tréningy vykonávať s troma vozidlami.

K dispozícii je viacero distribúcií prostredia ROS 2, ktoré by mali byť medzi sebou kompatibilné, ale líšia sa v podpore operačných systémov, pričom sa môžu líšiť aj v rýchlosti a stabilite. Na začiatku sme experimentovali s distribúciou ROS 2 Foxy určenou pre operačný systém Windows, avšak narazili sme na problémy so stabilitou. Takisto sa nám stalo, že niektoré Python balíčky nesúvisiace s Rosom nemali dobrú podporu s aktuálnou verziu Pythonu, takže bolo treba nainštalovať inú verziu Pythonu a do nej následne doinštalovať aj ROS2. Inštalácia ROSu na Windowse je pritom zložitý proces. Nakoniec sme sa rozhodli vyskúšať funkciu WSL (Windows Subsystem for Linux), vďaka ktorej je možné spustiť Linuxové jadro na Windowse. V rámci tohto prostredia sme nainštalovali ROS 2 distribúciu ROS 2 Humble Hawksbill [18]. Inštalácia v rámci Linuxu bola oveľa jednoduchšia a rýchlejšia, pozostávala len zo zadania niekoľko príkazov do príkazového riadku. Rovnako sme zaznamenali, že na Linuxe bolo oveľa jednoduchšie spravovať ostatné Python balíky. Jediná nevýhoda, ktorú sme pozorovali, bola vyššia spotreba operačnej pamäte, keďže samotná služba WSL musí alokovať časť RAM pre seba počas celej doby, kedy sa používa. Komunikácia agenta s prostredím v Unity fungovala bez problémov, hoci agent bežal v Linuxe a Unity simulácia vo Windowse, čo dokazuje výhody platformovej nezávislosti ROSu.

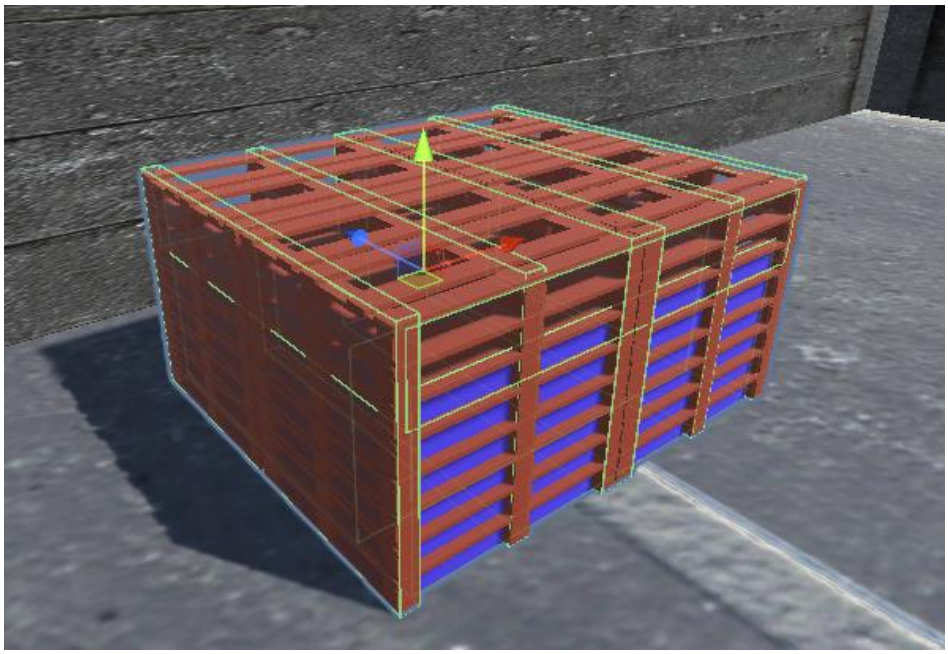
2.3 Systém odmien

Nastavenie mechanizmu na výpočet odmeny pre túto úlohu je pomerne zložité. Nevhodne zvolená odmena môže spôsobiť neželané správanie agenta alebo sa agent dokonca vôbec nenatrénuje. Výpočet odmeny zabezpečuje komponent OdmenaScript. Do výpočtu finálnej odmeny reprezentovanej jedinou hodnotou môže vstupovať viacero faktorov a OdmenaScript zabezpečuje ich meranie.

Prvým faktorom je náraz do stien. Jeho efekt je vždy pre agenta negatívny a pri skutočnom autonómnom vozidle by sme chceli zamedziť aj tomu, aby sa pohybovalo v tesnej blízkosti stien. Preto počítame vzdialenosť k najbližšej stene a od určitého

priblíženia začne OdmenaScript zaznamenávať zápornú negatívnu odmenu, ktorá lineárne klesá od 0 až k hodnote -1. Hodnotu -1 dosiahne v prípade, že vozidlo narazí do steny alebo do prekážky. Prekážka má iný charakter ako stena – v jej prípade nevieme vopred odhadnúť umiestnenie v rámci skladu, a preto môže byť vhodné ju obísť aj v tesnej blízkosti. Preto za priblíženie k nej nevypočítavame zápornú odmenu ako v prípade stien. Pre potreby výpočtu niektorých stratégií odmeny zaznamenávame agentovu aktuálnu rýchlosť a informáciu o tom, či sa pohybuje dopredu.

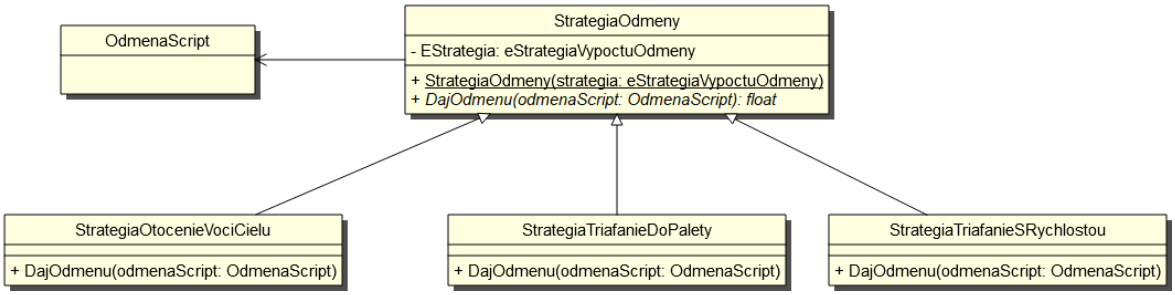
Z hľadiska interakcie s cieľom existuje niekoľko faktorov, ktoré sa skúmajú. Cieľom úlohy je, aby autonómne vozidlo zasunulo lyžiny do palety a aby ich zasunulo v čo najväčšom rozsahu – ideálne v takmer celej dĺžke. Chceme sa pritom vyhnúť negatívnej interakcii s paletou vo forme nárazu zboku alebo z prednej časti mimo otvorov určených na zasunutie lyžín.



Obrázok 11 – Box collidery palety

Ak nejaká časť vozíka narazí do časti palety, ktorá je vyznačená zeleným orámovaním, zvýši sa o 1 počítadlo CielCollidedCount. Ak sa vozík odtiahne a kontakt so stenou palety sa stratí, hodnota sa zníži o 1. K obdobnej interakcii dochádza aj v prípade, že lyžiny prejdú cez plochu vyznačenú modrou farbou. V tom prípade sa zvyšuje, respektíve znižuje počítadlo EnteredCount. Táto plocha má len vizuálny efekt, ale nemá žiadny vplyv na fyziku a lyžiny cez ňu môžu voľne prejsť. Za ňou sa hlbšie v palete nachádza ďalšia virtuálna plocha približne vo vzdialenosti dĺžky lyžín. Interakcia s ňou je naviazaná na premennú CielDorazCount. Táto plocha je umiestnená tak, aby s ňou vozidlo

vedelo vytvoriť kontakt len v prípade, že sú do palety zasunuté lyžiny skoro v celej ich dĺžke.



Obrázok 12 – Návrh mechanizmu na systém odmien

OdmenaScript sa stará primárne o zber dát v súvislosti s odmenou. V rámci experimentov sme potrebovali dynamicky skúšať viacero typov samotného výpočtu odmeny z týchto dát, preto bolo potrebné navrhnúť mechanizmus, pomocou ktorého bude možné bez kopírovania kódov prepínať medzi viacerými stratégiami výpočtu odmeny. Vytvorili sme preto triedu StrategiaOdmeny, ktorej atribútom je enumerát eStrategiaVypoctuOdmeny označujúci konkrétnu stratégiu a abstraktná metóda DajOdmenu, ktorá berie ako parameter celý OdmenaScript. V prípade pridania odmeny tak stačí vytvoriť potomka tejto triedy a implementovať túto metódu, pričom všetky údaje potrebné k vypočítaniu odmeny sú v rámci danej metódy dostupné prostredníctvom OdmenaScriptu.

Konkrétne stratégie spolu s vplyvom na priebeh tréningu a výsledky agenta budú popísané v časti Experimenty.

2.4 Zber dát v procese učenia

V rámci úvodných experimentov nám na posúdenie kvality agenta stačilo pre jednotlivé replikácie ukladať terminálnu odmenu a sumárnu odmenu pre jednotlivé replikácie. Cieľom týchto experimentov bolo naučiť agenta identifikovať cieľ a priblížiť sa k nemu v pomerne jednoduchom prostredí. Ako sme prechádzali ku komplexnejšej úlohe a prostrediu, bolo potrebné definovať odmenu komplexnejšie. Okrem samotnej odmeny bolo potrebné zbierať aj ďalšie údaje z prostredia, ktoré nám umožňovali lepšie pochopiť správanie sa agenta a identifikovať prípadné problémy. Preto sme sa rozhodli vytvoriť systém na ukladanie viacerých dát popisujúcich správanie sa agenta v prostredí z jednotlivých replikácií.

Simulácia v Unity ukladá pre každú replikáciu jej ID, počet nárazov do cieľovej palety z vonkajšej strany, počet nárazov do cieľovej palety z vnútornej strany, dĺžku narazenia z vonkajšej a vnútornej strany v počte volaní FixedUdpate, maximálnu rýchlosť pri náraze a čas trvania replikácie. V rámci agenta sa ukladá terminálna odmena, sumárna odmena za celú replikáciu a počet použitia jednotlivých akcií. Pre jednoduchšie spracovanie týchto dát sme urobili systém, ktorý dokáže načítané dáta vizualizovať vo forme grafov.

3 Experimenty

Experimenty prebiehali iteratívnym spôsobom. Prvé experimenty sme začali na veľmi jednoduchej úlohe, aby sme vedeli odhadnúť, aké zložité je pre agenta sa na ňu natrénovať. Na základe nameraných dát a pozorovania jázd autonómneho vozidla sme sa pokúšali identifikovať problémy pre aktuálneho agenta a aktuálne prostredie. Keď sa zdalo, že agent vie už daný rozsah úlohy efektívne riešiť, úlohu sme spravili zložitejšou a vykonali ďalšiu sadu experimentov.

Keďže táto úloha nemá presnú matematickú definíciu, je možné sa v rámci experimentov zamerať na viacero faktorov, ktoré môžu mať vplyv na správanie agenta. Od systému odmien, cez rozsah a prvky obsiahnuté v stavovom priestore, synchronizačné konštanty, typy agentov, hyperparametre agentov a architektúry sietí až po rozloženie prvkov v rámci prostredia. Pre dosiahnutie stopercentných výsledkov by bolo možné urobiť medzi týmito faktormi karteziánsky súčin experimentov, čo by bolo ale z časového hľadiska prakticky nemožné. Preto sme sa rozhodli experimenty vykonávať iteratívnym spôsobom. Na danej úlohe sme vždy vykonali niekoľko experimentov zameraných na jeden aspekt simulácie a na základe výsledkov z týchto experimentov sme sa rozhodli pozorovania a nastavenia z nich preniesť do ďalších experimentov.

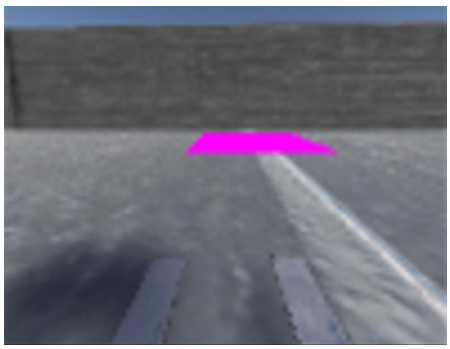
Všetky experimenty prebiehali na notebooku DELL Precision 17 5750 s procesorom Intel Core i7 10750H (6 jadier, 12 vlákien, turbo boost 4,4GHz), 40GB RAM DDR4 2993MHz a grafickej karte NVIDIA Quadro T2000 4GB.

3.1 Experimenty s DQN a DDQN agentom

Prvé experimenty využívali DQN agenta. Stav systému bol reprezentovaný obrazovými dátami s rozmermi 50x80 pixelov, celková vstupná matica tak mala rozmer 50x80x3. Agent mal k dispozícii jednu zo 6 akcií reprezentujúce pohyb autonómneho vozidla: dopredu, dopredu doprava, dopredu doľava, dozadu, dozadu doľava a dozadu doprava. Architektúra neurónovej siete pozostávala z troch konvolučných vrstiev kvôli analýze obrazu – výstup z nich sa preformátoval na jednorozmerný vektor a poslal do dvoch plne prepojených vrstiev.

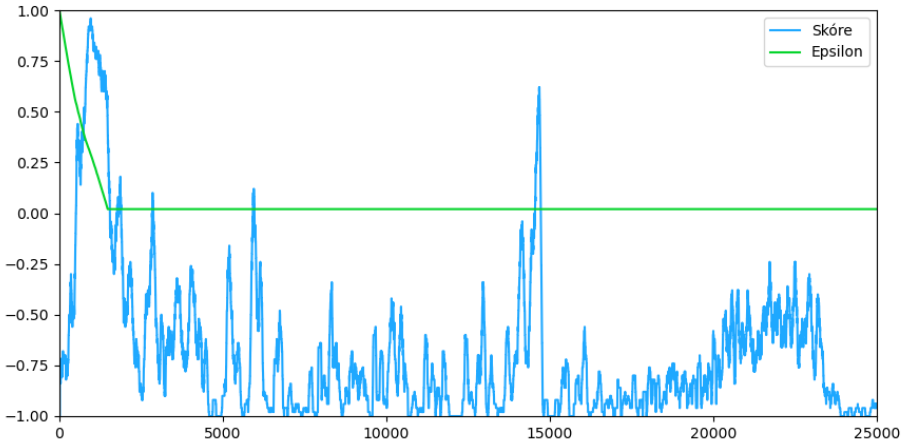
Z hľadiska prostredia bol prvý experiment veľmi jednoduchý – autonómne vozidlo aj cieľ sa pri každej replikácii vždy zjavili na rovnakom mieste a vozidlo sa malo presunúť

do cieľa. Cieľ bol reprezentovaný štvorcom na podlahe, ktorého farba bola zvolená tak, aby bola kontrastná s okolitým prostredím.



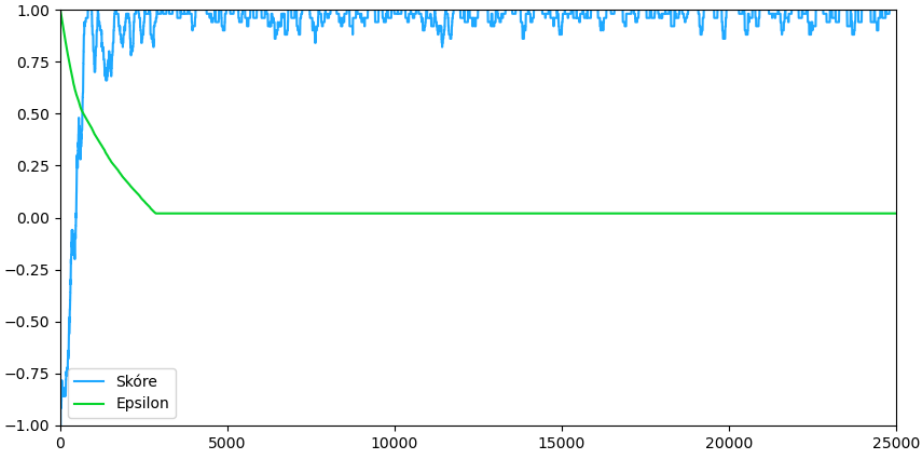
Obrázok 13 – Ukážka stavového priestoru pre základné experimenty

V tomto experimente by malo stačiť, aby sa agent naučil ísť dopredu, tento agent teoreticky ani nepotreboval videnie na riešenie danej úlohy. Skóre z jednej replikácie pre účely analýzy výsledkov sme počítali len ako terminálnu odmenu – ak agent nabúral do steny alebo mu došiel čas, dostal terminálnu odmenu -1, naopak ak sa dokázal presunúť do vyznačenej oblasti včas, dostal terminálnu odmenu 1. Terminálna odmena je odmena v poslednom stave, teda v stave, kedy replikácia skončila. Terminálnu odmenu budeme v rámci vyhodnotenia experimentov označovať aj ako skóre. Okrem nej v rámci tohto experimentu agent dostával aj neterminálne, teda čiastkové odmeny počas replikácie, ak sa priblížil k cieľu. Hodnota odmeny lineárne narastala medzi intervalom 0 a 1 podľa znižujúcej sa vzdialenosti agenta voči cieľu. Agent dostával aj malú zápornú odmenu, ak sa priblížil k stene na vzdialenosť menšiu ako vopred definovaná malá konštanta. Táto odmena klesala na intervale 0 až -1 s klesajúcou vzdialenosťou k najbližšiemu bodu steny. Grafy zobrazujú závislosť počtu replikácií voči priemernému skóre za 100 posledných replikácií a voči hyperparametru epsilon DQN agenta. Čas bol limitovaný ako 10 násobok času, ktorý by trvalo prejsť do cieľa človeku, keby ovládal autonómne vozidlo.



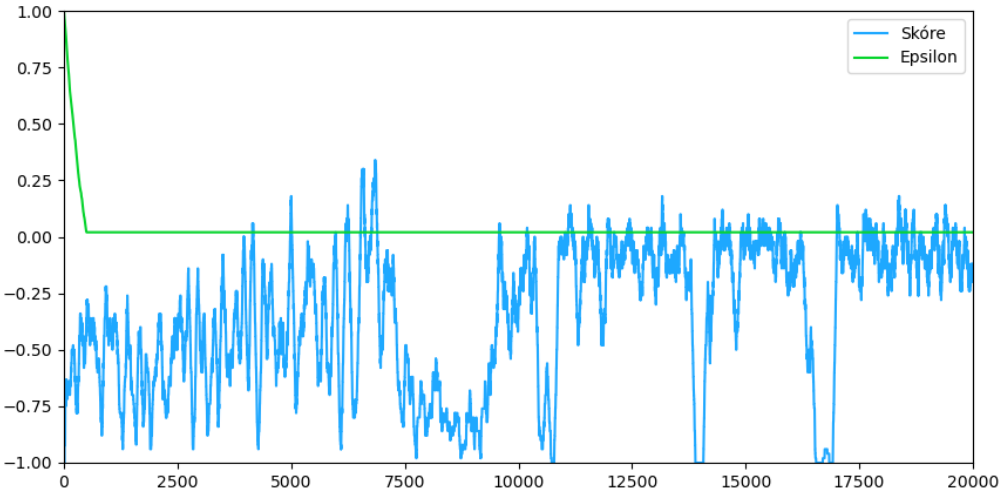
Obrázok 14 – Experiment so statickým cieľom a lineárnou odmenou

Keď agent dostával lineárnu odmenu okolo cieľa, pomerne rýchlo sa natrénoval, ale následne skóre kleslo (**Obrázok 14**). Zistili sme, že agent úmyselne spomaľuje pred cieľom, pretože suma odmien za replikáciu bola v tomto prípade väčšia ako keby priamo prišiel do cieľa. Preto sme vykonali experiment, kde sme úplne vynechali čiastkové odmeny a agent dostal len jednu terminálnu odmenu. V tomto prípade sa rýchlo natrénoval a dokázal stabilne riešiť túto úlohu (Obrázok 15).



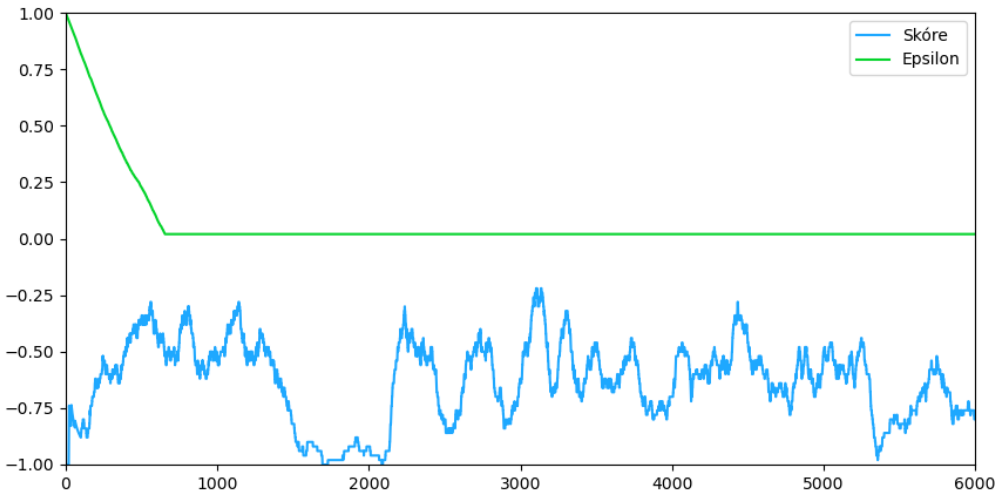
Obrázok 15 - Experiment so statickým cieľom a terminálnou odmenou

Keďže sme dokázali riešiť úlohu so statickým cieľom, rozhodli sme sa prejsť na komplikovanejšiu úlohu. V tomto prípade sa poloha cieľa generovala náhodne medzi ľavou a pravou stenou – vždy na takom mieste, aby sa cieľ vygeneroval v zornom poli kamery. Na riešenie tejto úlohy už je potrebné, aby neurónová sieť vedela identifikovať na základe obrazu polohu cieľa a následne k nemu navigovala autonómne vozidlo.



Obrázok 16 – Experiment s pohyblivým cieľom a terminálnou odmenou

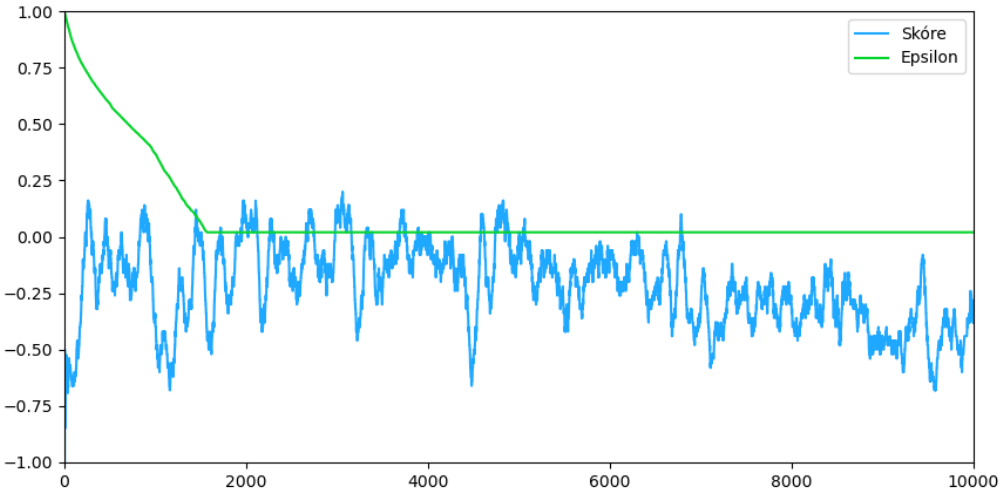
Domnievali sme sa, že stavový priestor, ktorý posielame agentovi, už bude nedostatočný na riešenie tejto úlohy, pretože je rozdiel, či sa vozidlo pohybuje rýchlo alebo pomaly a akým smerom sa pohybuje, pričom tieto informácie nie sú obsiahnuté vo fotke. Do vstupu prvej plne prepojenej vrstvy sme preto pridali informáciu o aktuálnej rýchlosti a aktuálnom natočení kolies. Agent sa dokázal natrénovať tak, že trafil k cieľu približne v 50% prípadoch, čo znamená, že priemerná odmena bola 0 (Obrázok 16). Chceli sme overiť, či to nie je spôsobené zle nastavenou odmenou, pretože v rámci tohto experimentu sme mu dávali len terminálnu odmenu 1 za dosiahnutie cieľa. Ten istý experiment sme tak vykonali ešte s lineárnou odmenou v okolí cieľa predelenou dvojnásobkom maximálneho počtu akcií v rámci danej replikácie. Tým sme chceli zabezpečiť, aby agent dostával spätnú väzbu aj počas replikácie a nie len na konci, ale aby sa zároveň nemohol naučiť uprednostňovať viacero malých odmien pred finálnou odmenou a kvôli tomu nedokončovať úlohu. Toto nastavenie však nepomohlo zlepšiť výsledky. Domnievali sme sa, že by mohlo pomôcť, ak by agent dostával spätnú väzbu za každú vykonanú akciu, tak sme odmenu skúsili upraviť tak, že sa vypočíta vzdialenosť k cieľu pred vykonaním akcie a po vykonaní akcie a odmena, ktorú agent dostane, bude ich rozdiel normovaný z intervalu -1 po 1. Za týchto okolností by mal agent dostávať kladnú odmenu, ak sa bude pohybovať smerom k cieľu a zápornú, ak sa bude od cieľa vzďaľovať. Tento experiment (Obrázok 17) však nepriniesol zlepšenie. Mohlo to byť spôsobené aj tým, že vzdialenosť voči cieľu sa zmenšuje napríklad aj vtedy, keď sa autonómne vozidlo pohybuje smerom dopredu, ale nie priamo smerom k cieľu.



Obrázok 17 – Experiment s pohyblivým cieľom a približovacou odmenou

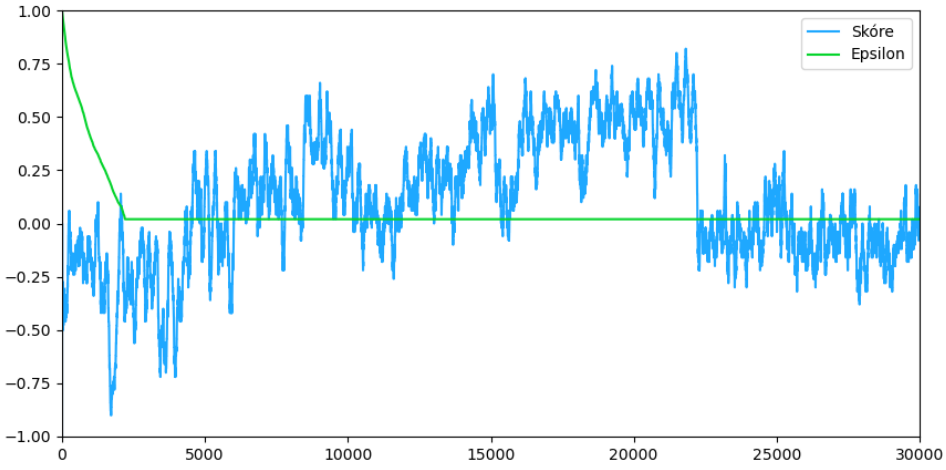
Z pozorovania priebehu hier agenta po natrénovaní sa nám zdalo, že problém je videnie. Autonómne vozidlo sa naučilo prechádzať v každej replikácii podobnú trasu pohybom dopredu a následne do jednej strany v úrovni, kde sa generoval cieľ, aby maximalizovalo pravdepodobnosť, že dorazí do cieľa. V rozhodnutiach však nebolo badateľné, že by akýmkoľvek spôsobom vnímalo polohu cieľa. Na začiatku sme skúsili cieľ na zemi nahradiť skutočnou paletou, pričom úlohou vozidla bolo naraziť do palety. Paleta má oproti cieľu na zemi trojrozmerný charakter a navyše je na obraze oveľa vyššia, a teda tvorí väčšiu časť obrazu. Táto zmena však neviedla k zlepšeniu výsledkov. Aj napriek tomu sme sa rozhodli nasledujúce experimenty trénovať už s paletou, nakoľko ide o presnejšiu reprezentáciu reality.

Keďže sme sa domnievali, že problém bude v absencii schopnosti rozpoznávať obraz z dát, rozhodli sme sa zmeniť architektúru siete na sofistikovanejšiu. Tri konvolučné vrstvy v pôvodnej architektúre sme nahradili sieťou ResNet 18 predtrénovanou na datasete ImageNet. Poslednú klasifikačnú vrstvu z ResNetu sme odstránili a výstup z predposlednej vrstvy sme transformovali na jednorozmerný vektor a poslali do dvoch plne prepojených vrstiev.



Obrázok 18 – Experiment RESNET a 2 plne prepojené vrstvy

V tomto prípade výsledky v grafovej reprezentácii neboli výrazne lepšie (Obrázok 18), avšak z priebehu hry agenta sa zdalo, že autonómne vozidlo sa snaží zatáčať do smeru, kde sa nachádza cieľ, ale jeho pohyb bol nemotorný. Preto sme chceli overiť, či je použitá architektúra dostatočná z hľadiska rozhodovania. Hypotéza bola, že dve plne prepojené vrstvy nestačia na to, aby sa dokázali naučiť transformovať príznaky z ResNetu na rozhodnutia agenta. Do architektúry siete sme preto pridali ešte ďalšie dve plne prepojené vrstvy a keďže sme videli priebežný rast skóre, nechali sme agenta natrénovať na väčšom množstve replikácií.



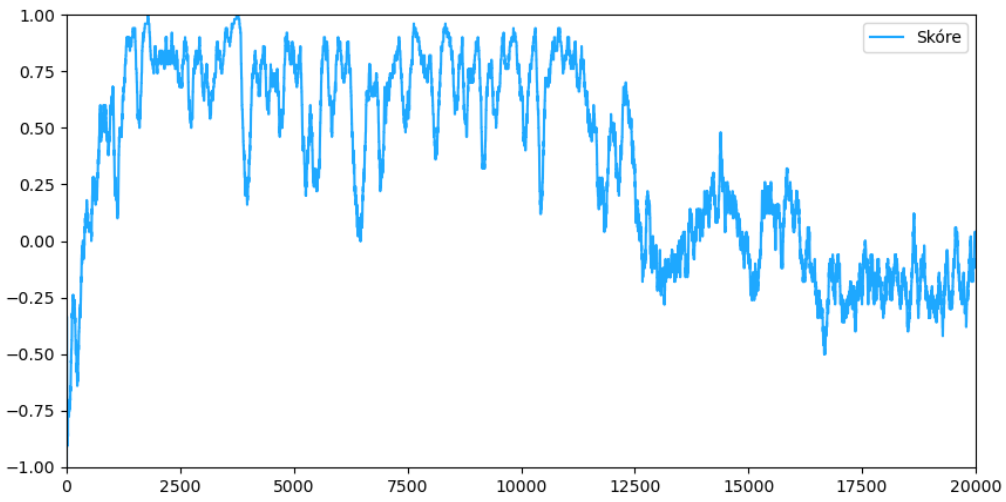
Obrázok 19 – Experiment RESNET a 4 plne prepojené vrstvy

Agent sa po zmene architektúry dokázal natrénovať tak, že jeho priemerné skóre pri posledných sto replikáciách bolo v najlepšej časti tréningu až 0,75, čo po preškálovaní znamená, že dokázal trafiť do cieľa v približne 88% prípadoch (Obrázok 19). Skóre však malo tendenciu padať. Vzhľadom na to, že ku pádu vždy došlo v momente, kedy sa

prepínala trénovaná sieť na online sieť, sme si mysleli, že to súvisí so všeobecnosťou vlastnosťou DQN agenta, ktorou je nestabilita tréningu. Keď sme skúsili načítať sieť z posledného miesta s dobrým skóre a dotrénovať agenta nanovo, tak sa skóre priebežne zvyšovalo, ale výkon agenta opäť po nejakom počte replikácií prudko spadol. Vyskúšali sme aj DDQN agenta, ktorý by mal byť v porovnaní s DQN agentom o niečo stabilnejší, ale uňho sme pozorovali podobné výsledky. Rozhodli sme sa preto vyskúšať PPO agenta.

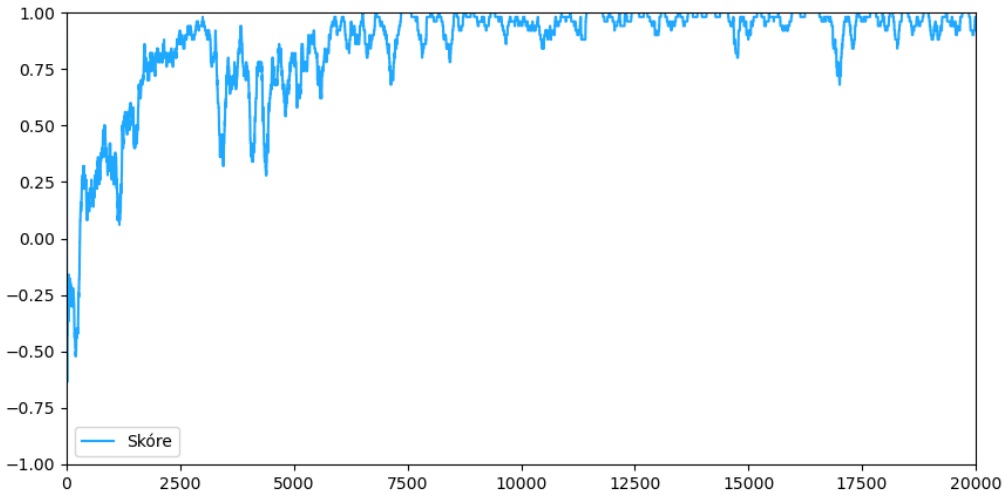
3.2 Experimenty s PPO agentom

PPO agent sa dokázal oveľa rýchlejšie natrénovať z pohľadu počtu replikácií (Obrázok 20). Kdežto v prípade DQN dosahoval najlepšie skóre na úrovni 0,75 a natrénovať sa na tú úroveň trvalo až 20000 replikácií, PPO agent sa dokázal natrénovať na úroveň blízku hodnote 1 už po približne 2000 replikáciách, teda 10x rýchlejšie. Stále však pretrvávali problémy so stabilitou tréningu.



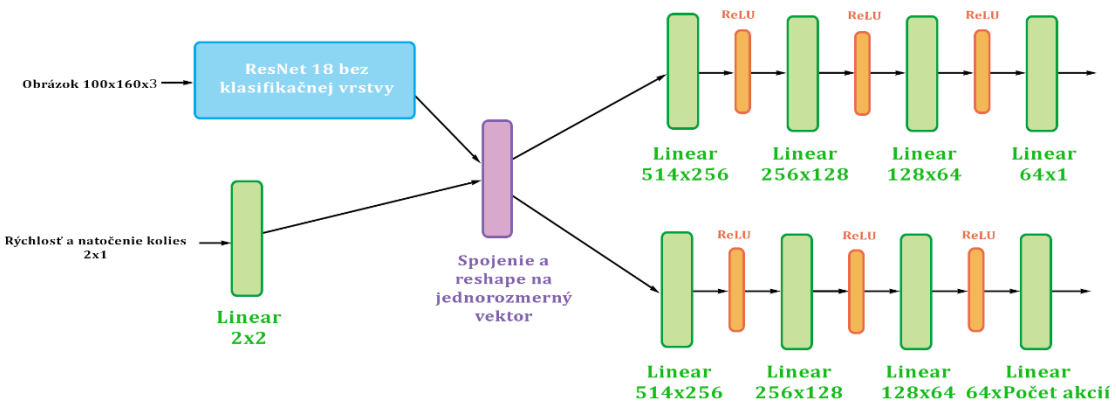
Obrázok 20 – PPO agent prvý experiment

Vzhľadom na to, že PPO agent sa trénoval oveľa rýchlejšie, chceli sme overiť, či je ResNet18 naozaj potrebný v architektúre a či by sa tento agent nedokázal natrénovať aj so 6 konvolučnými vrstvami doplnenými o 4 plne prepojené vrstvy. V tejto konfigurácii sa však agent nenatrénoval, a preto sme aj pri ďalších experimentoch používali ResNet.



Obrázok 21 – PPO po zmene hyperparametrov

Výkyvy v stabilite tréningu by mohli spôsobovať problémy pri vyhodnocovaní ďalších experimentov, preto sme sa rozhodli overiť, či by sme ich nevedeli eliminovať zmenou hyperparametrov agenta. Pred touto zmenou mal agent hodnoty hyperparametrov nastavené nasledovne: Gama = 0,9, Learning rate = 0,0001, Beta entropy = 0,001, Value loss koeficient = 0,5, Epsilon = 0,2. Parameter Gama sme zmenili z 0,9 na 0,999 a Epsilon na 0,1. Po zmene sme dosiahli oveľa stabilnejší priebeh tréningu a lepšie výsledky (Obrázok 21).

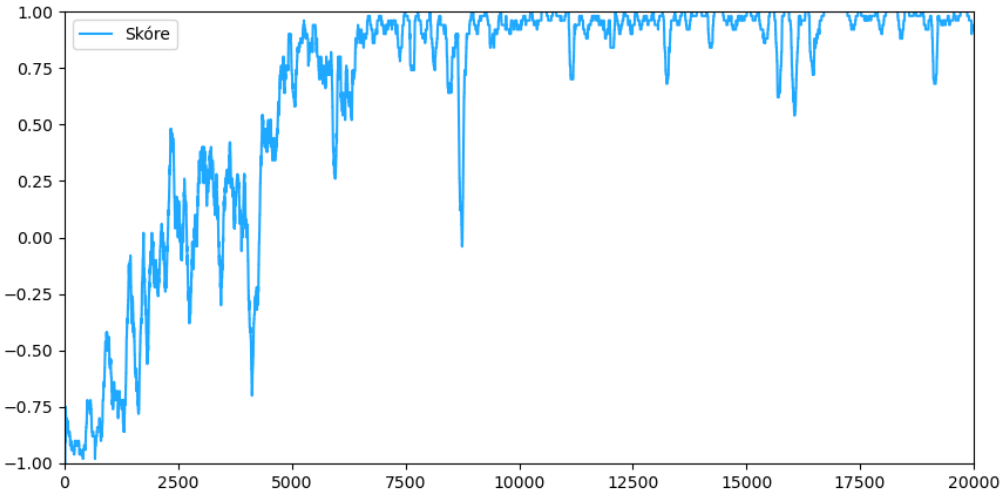


Obrázok 22 – Architektúra siete použitá pri trénovaní PPO agenta

3.3 Experimenty s odmenami

Keď sme už dokázali nájsť spôsob ako agenta stabilne natrénovať na úlohe nárazov do palety, ktorá sa pri každej replikácii vygeneruje vždy na novom mieste, mohli sme prejsť na ťažšiu úlohu. V tomto prípade sme chceli, aby agent lyžinami trafil do otvorov v palete, čím by autonómne vozidlo vedelo v realite paletu uchopiť. Pri riešení tejto úlohy zohráva rolu viacero faktorov, nie len vzdialenosť vozidla od prekážky, preto bolo potrebné prehodnotiť aktuálny systém odmien a navrhnúť nové odmeny pre túto úlohu.

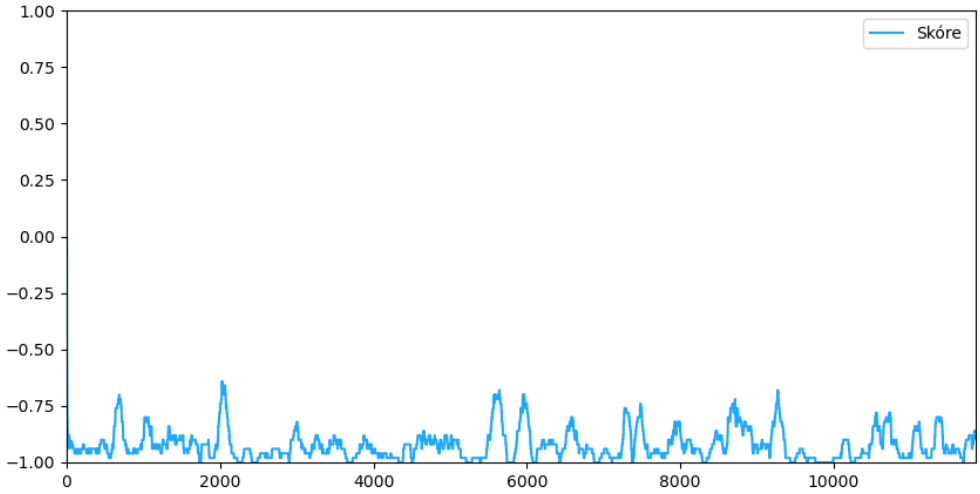
V rámci prvého experimentu sme agentovi dávali pozitívnu odmenu, ak bol zasunutý aspoň jednou lyžinou do palety. Počítala sa lineárne od zasunutia až po hodnotu 1. Maximálnu odmenu, teda 1, agent dostal v prípade, ak boli lyžiny zasunuté aspoň na 60%, pretože sme chceli overiť, ako bude agent reagovať na novú úlohu. Čiastkové kladné odmeny sme delili dvojnásobným maximálnym počtom akcií v rámci replikácie, aby sme zabránili problémom so snahou o uprednostnenie čiastkových odmien pred terminálnou odmenou.



Obrázok 23 – Zapichnutie do palety, lineárna odmena

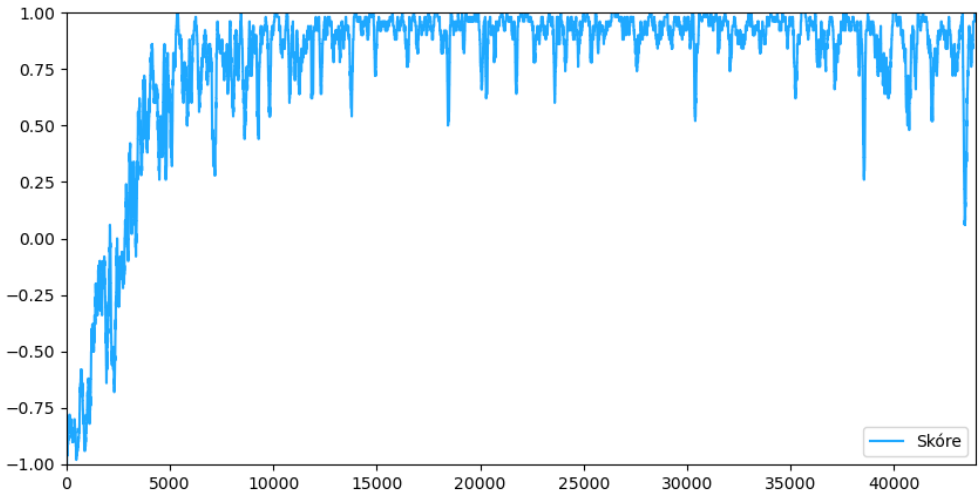
Pre agenta bola táto úloha v porovnaní s predchádzajúcou úlohou o niečo zložitejšia (Obrázok 23). Natrénovanie trvalo až približne 7500 replikácií v porovnaní s 2500 replikáciami u predchádzajúcej úlohy. Aj napriek tomu sa ju ale stabilne naučil riešiť. Z pozorovaní agentovho správania sme vypozorovali, že pohyb nie je taký, ako by sme si predstavovali. Agent často narážal do stien palety z boku a do miesta určeného na

zasunutie lyžín sa mu podarilo trafiť až na viacero pokusov, pričom to bolo často z uhla, takže lyžiny neboli zasunuté ideálne.



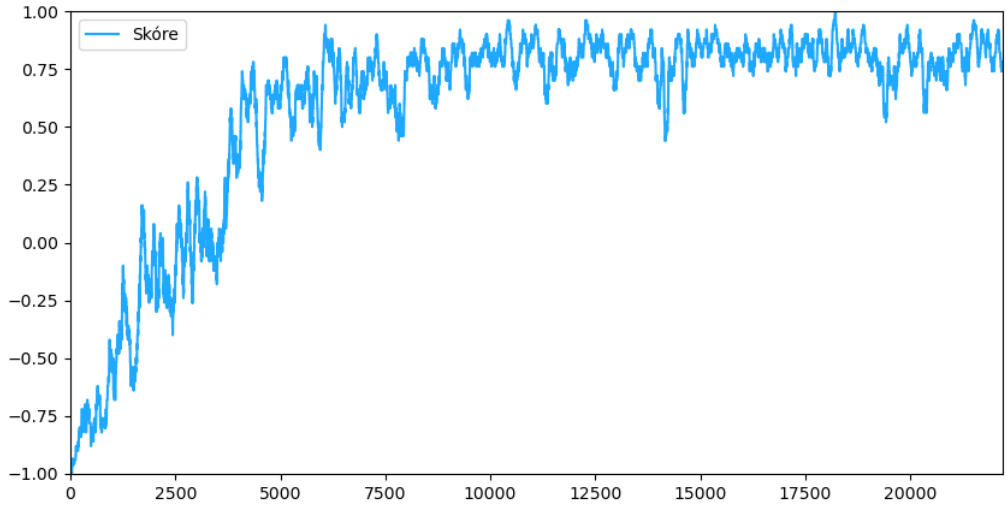
Obrázok 24 – Zapichnutie do palety, terminálna záporná odmena za náraz

Tento problém sme chceli vyskúšať vyriešiť tým, že ak agent narazí do palety, dostane okamžite terminálnu odmenu -1 a replikácia skončí. Ako je vidieť z výsledkov tréningu (Obrázok 24), tento spôsob nie je možné použiť. Odôvodňujeme si to tým, že v rámci procesu učenia, keď agent skúša náhodnú stratégiu, je oveľa väčšia šanca, že narazí do palety nevhodným spôsobom a dostane terminálnu odmenu -1. Naopak na získanie terminálnej odmeny 1 za správne zasunutie do palety, musí byť vozidlo voči palete správne otočené a lyžiny musia byť v správnej polohe, aby nenabúrali do hrany palety. Nevhodný náraz a záporná odmena sú teda oveľa pravdepodobnejšie v rámci prieskumnej (exploration) fázy a agent si asociuje pohyb k palete so zápornou odmenou, takže sa mu vyhýba. Bolo to vidieť aj na priebehu samotnej replikácie, kedy chodil dopredu a dozadu, aby nenarazil do stien, ale ani do palety. Ten istý experiment sme vyskúšali aj s odmenou 1 za oveľa plytšie zasunutie do palety, ale výsledky boli rovnaké.



Obrázok 25 - Zapichnutie do palety, malá záporná odmena za náraz

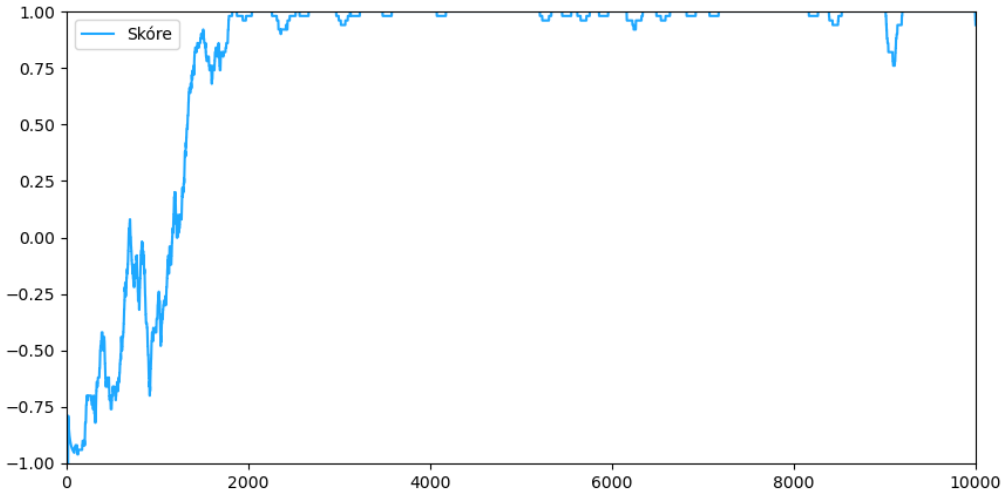
Experiment sme skúsili upraviť tak, že záporná odmena za náraz do palety nebude terminálna, ale bude malá ako v prípade mierneho zasunutia. Ak agent narazil do palety, získal odmenu vypočítanú ako -1 podelenú dvojnásobným maximálnym počtom akcií v rámci replikácie. S touto odmenou agent dosiahol dobré výsledky (Obrázok 25) a dokázal sa dokonca natrénovať rýchlejšie ako keď sme mu dávali len odmenu za zasunutie do palety. Skóre blízke hodnote 1 dosiahol už po približne 5500 replikáciách.



Obrázok 26 – Zapichnutie do palety, odmena s rýchlosťou

Do tejto chvíle sme v odmene nemali odzrkadlené, ako rýchlo agent do palety narazil. U skutočného autonómneho vozidla by na tom ale záležalo. Ak by sa totiž okolo

palety len jemne obtrelo, je to iná situácia, ako keď do palety narazí v maximálnej rýchlosti. Na základe pokusu v simulačnom prostredí sme určili, aká rýchlosť by bola príliš veľká a túto rýchlosť sme použili ako konštantu pri výpočte odmeny. Ak vozidlo narazilo nad túto rýchlosť, replikácia skončila s odmenou -1. Druhá rýchlosť bola určená na základe toho, aká rýchlosť by bola príliš veľká, ale nie dostatočne veľká na to, aby replikácia skončila. Ak vozidlo narazilo nad túto rýchlosť, dostalo rýchlostnú pokutu. Ak došlo k zasunutiu lyžín, najskôr sa vypočítala odmena za zasunutie a od nej sa odpočítala rýchlostná pokuta. Ako je vidieť na grafe (Obrázok 26), aj s takto nastavenou odmenou sa vozidlo dokázalo pomerne rýchlo natrénovať, ale už nie na 100%.



Obrázok 27 – Zapichnutie do palety, presnejšie riadenie

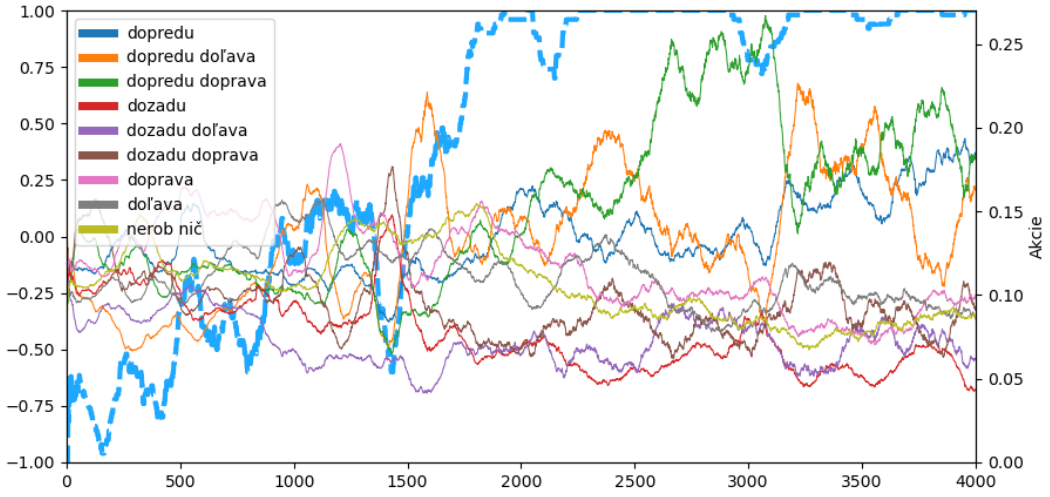
Z pozorovania tréningu sa zdalo byť problematické, že vozidlo nedokáže dosiahnuť požadovanú presnosť, preto sme skúsili zvýšiť presnosť väčšou frekvenciou rozhodnutí. Predtým bola synchronizácia nastavená tak, že rozhodnutie bolo po agentovi vyžadovaných každých 15 fixed updatov, túto frekvenciu sme znížili na 5 fixed updatov. Po vykonaní tohto nastavenia sme dostali oveľa lepšiu stabilitu tréningu – hodnota skóre v rámci tréningu sa stabilizovala takmer na hodnote 1. Skúsili sme teda experiment s týmto nastavením spustiť znova, ale tentokrát sa vozidlo na začiatku každej replikácie umiestnilo oveľa ďalej od prekážky. Aj s týmto nastavením sa vozidlo dokázalo na riešenie úlohy natrénovať rýchlejšie, a to už po 2000 replikáciách. (Obrázok 27).

3.4 Experimenty s priebehom simulácie

V predchádzajúcich experimentoch sme našli spôsob, ako agenta efektívne natrénovať na zapichnutie lyžín do palety z hľadiska terminálnej odmeny. Pre potreby ďalších experimentov sme sa rozhodli zbierať a vyhodnocovať oveľa viac dát ako len terminálnu odmenu. Tieto dáta nám mali pomôcť lepšie pochopiť správanie sa agenta, odhaliť možné problémy a v neposlednom rade tiež komplexnejšie porovnávať výsledky jednotlivých experimentov medzi sebou. V rámci ďalších pokusov sme preto pre každý experiment sledovali atribúty popísané v kapitole 2.4. Okrem samotného dokončenia úlohy sme sa snažili nájsť také nastavenie, ktoré by dosiahlo najplynulejší a najprirodzenejší pohyb agenta.

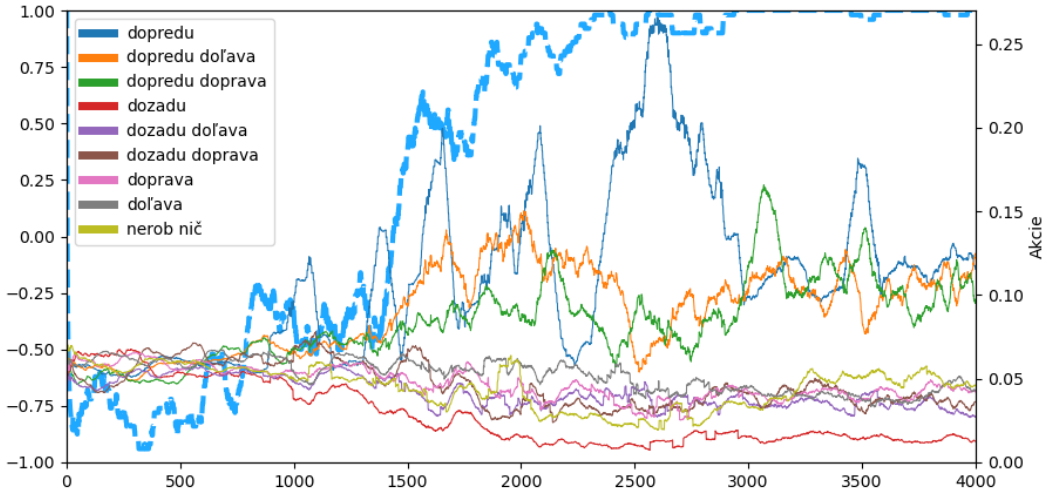
3.4.1 Experimenty s akciami

V rámci tohto experimentu sme pridali tri nové akcie – zatočenie doprava bez použitia motora, zatočenie doľava bez použitia motora a nerob nič – v tom prípade autonómne vozidlo vyrovnalo kolesá a ani nepoužilo motor. Ak by vozidlo riadil človek, mal by tieto akcie k dispozícii, preto sme chceli zistiť, či sa ich agent naučí používať a aký budú mať vplyv na tréning. Na nasledujúcich grafoch prerušovaná modrá krivka reprezentuje skóre, zatiaľ čo ostatné krivky reprezentujú použitie jednotlivých akcií (Obrázok 28). Agent sa dokázal natrénovať za približne 2000 replikácií, čo z hľadiska skóre nepredstavuje zásadné zlepšenie. Akcie pre nerob nič mali malé použitie približne na úrovni akcií pre cúvanie. Agent pritom z logického hľadiska preferuje akcie chodu dopredu, keďže cieľ sa nachádza pred ním. Cúvanie používa len ako spomaľovanie alebo pri manévroch pred paletou.

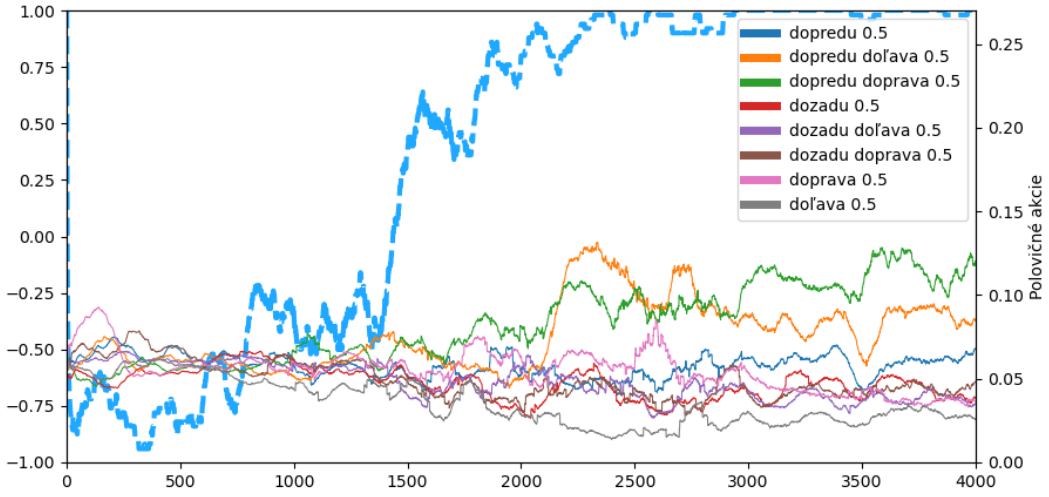


Obrázok 28 – Experiment - pridanie akcie pre nerob nič

V rámci ďalšieho experimentu sme chceli overiť, či by pomohlo pridať akcie s polovičnou presnosťou. Bežná akcia napríklad pohyb dopredu vyvolá maximálnu akceleráciu autonómneho vozidla daným smerom. V prípade polovičných akcií by vozidlo mohlo dosiahnuť lepšiu presnosť. Vzhľadom na ich počet je pre lepšiu prehľadnosť prehľad použitia jednotlivých akcií rozdelený do dvoch samostatných grafov (Obrázok 29, Obrázok 30). Rozdelenie polovičných akcií sa po natrénovaní podobalo na štandardné akcie, ale v rámci tohto experimentu sme nespozorovali vplyv na rýchlosť tréningu ani na iné sledované premenné.

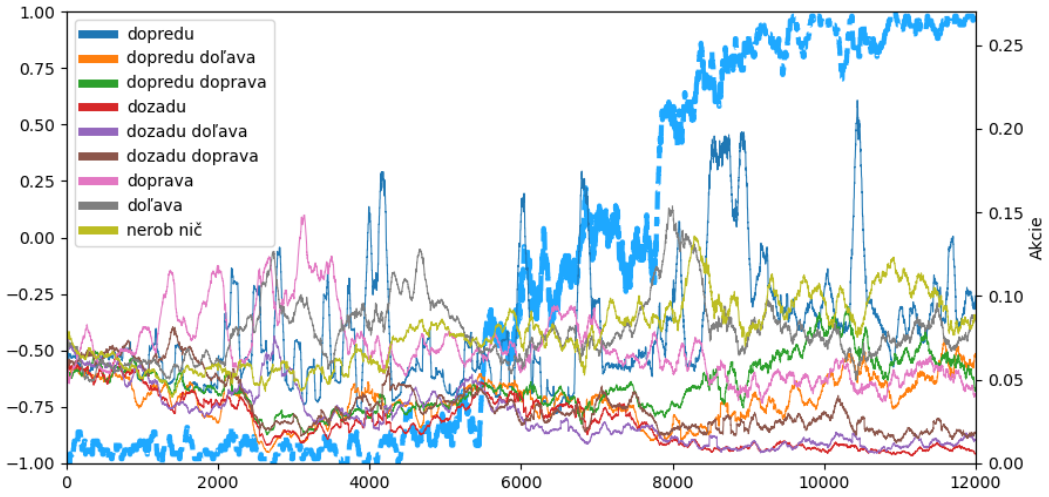


Obrázok 29 – Experiment – pridanie polovičných akcií – základné akcie



Obrázok 30 - Experiment – pridanie polovičných akcií

Chceli sme tiež overiť, či by dokázal agent vykompenzovať menšiu presnosť z hľadiska frekvencie akcií za jednotku času polovičnými presnejšími akciami. Preto sme zmenili synchronizáciu a umožnili mu robiť trikrát menej akcií za rovnaký čas. Agent sa nakoniec dokázal natrénovať aj v tomto nastavení, ale neprinieslo to požadovaný benefit, pretože na to potreboval až štyrikrát viac replikácií (Obrázok 31).



Obrázok 31 – Experiment – polovičné akcie s menšou presnosťou

3.4.2 Väčšie rozlíšenie, zatáčanie vzadu a zmena uhla otáčania

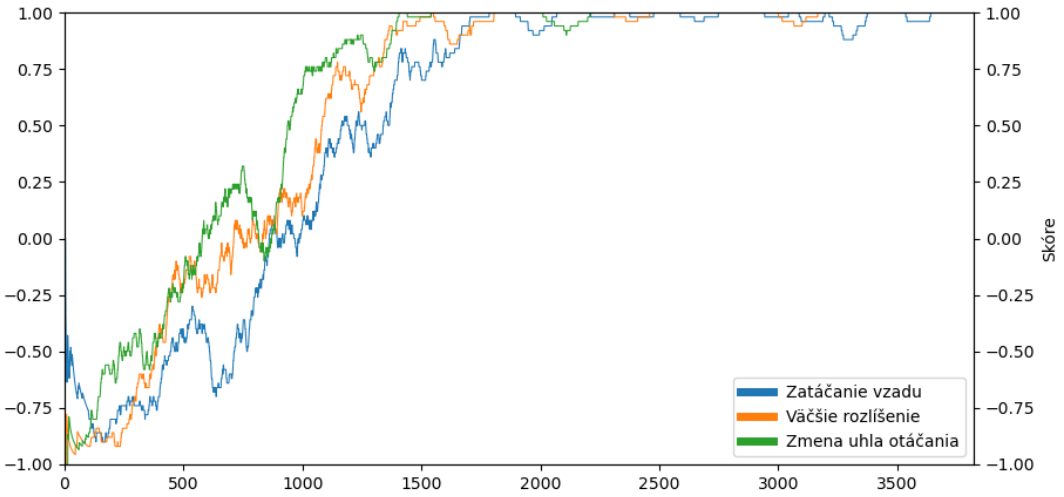
V tejto fáze experimentov sme sa dostali do bodu, kedy sa agent dokázal naučiť nami definovanú úlohu riešiť efektívnym spôsobom po približne 2000 replikáciách. Jeho pohyb však nebol ideálny – vozidlo malo tendenciu cúvať aj keď existoval spôsob ako úlohu vyriešiť efektívnejšie bez cúvania a pohyb stále nebol úplne prirodzený. Zároveň existovalo niekoľko premenných, ktoré sme na začiatku nastavili odhadom, ale v rámci riešenia úlohy je možné ich meniť. Chceli sme preto experimentálne overiť, aký vplyv na riešenie danej úlohy bude mať zmena týchto faktorov a či by nemohli viesť napríklad k natrénovaniu za menší počet replikácií alebo prirodzenejšiemu pohybu autonómneho vozidla.

Prvý experiment sa týkal rozlíšenia. Pôvodné rozlíšenie 100x160 bolo stanovené na základe toho, že to bolo najnižšie rozlíšenie, z ktorého bolo možné ľudským pohľadom identifikovať paletu ako trojrozmerný objekt a zachytiť aj jej detaily ako geometrický tvar, otočenie v priestore a miesta na zapichnutie lyžín, čo pri pôvodnom rozlíšení 50x80 použitom na primitívnejšej úlohe s cieľom vyznačeným na zemi, nebolo možné. Rozlíšenie sme v rámci tohto experimentu zvýšili na 200x320, teda zoštvornásobili počet pixelov. Vybrať správne rozlíšenie vstupného obrázka pre úlohu, ktorá je závislá na dobrej analýze obrazu, môže byť neľahká úloha. Na jednej strane kvalita obrazu musí byť dostatočná, aby na obrázku agent dokázal identifikovať všetky príznaky, ktoré potrebuje pre rozhodnutie. Na druhej strane, vyššie rozlíšenie prináša viac voliteľných parametrov do neurónovej siete a urobiť konvolúciu nad väčším obrázkom a aj vyrenderovať ho, je časovo náročnejšie na výpočet, čo spomaľuje tréning z časového hľadiska.

Ďalej sme chceli overiť, či by sme nevedeli docieliť prirodzenejšie správanie agenta tým, že zmeníme vlastnosti autonómneho vozidla. To až do tejto chvíle využívalo zatáčanie na predných kolesách. V našej úlohe je vozidlo reprezentované vysokozdvižným vozíkom. Existuje viacero modelov týchto vozíkov a niektoré majú zatáčanie na zadných kolesách, čo by malo teoreticky uľahčiť trafenie lyžinami do palety. Vo všetkých predchádzajúcich experimentoch bol maximálny uhol otáčania kolies agenta 15 stupňov. Pri sledovaní agentových hier sa nám zdalo, že je to limitujúci faktor a takisto pre vysokozdvižný vozík pomerne malá hodnota, a tak sme skúsili zvýšiť uhol otáčania na 35 stupňov.

Výsledky ukázali, že tieto faktory nemali žiadny zásadný vplyv na počet replikácií, ktoré agent potrebuje na natrénovanie (Obrázok 32). Rozdiely sme nezaznamenali ani pri

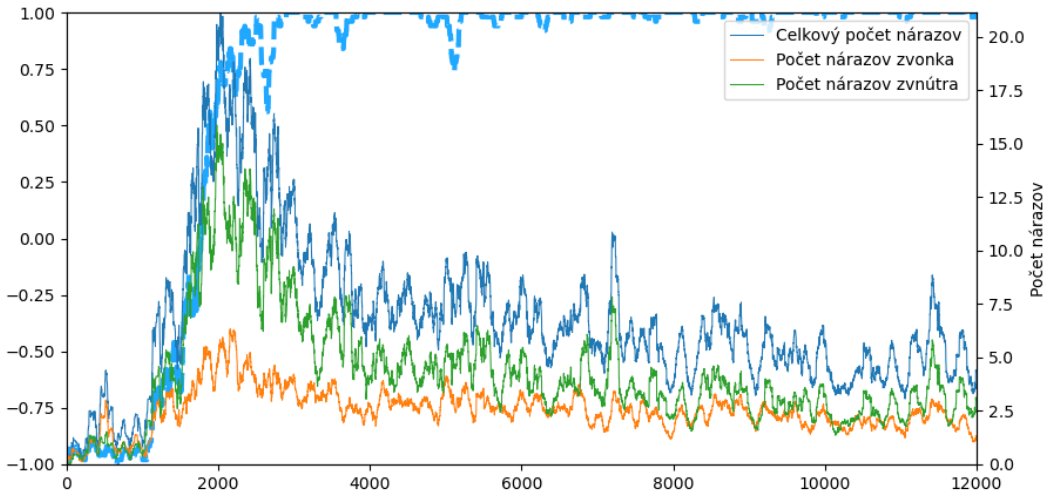
porovnaní priemerného počtu nárazov alebo priemernej rýchlosti pri náraze. Vizuálne sledovaním replikácií sa však zdalo, že agent s 35 stupňovým zatáčaním má prirodzenejší pohyb.



Obrázok 32 – Experiment so zmenou rozlíšenia a zatáčaním

Keď sme si pozreli viacero replikácií, zistili sme, že s týmto nastavením odmeny vozidlo dostáva terminálnu odmenu za relatívne plytké zasunutie. V týchto experimentoch stačilo 60% zasunutie lyžín, aby replikácia skončila s výsledkom 1. Rozhodli sme sa preto opäť upraviť odmenu, a to tak, že vozidlo dostane odmenu len ak sú zasunuté do palety obe lyžiny a ak sú zasunuté takmer celé. Aby dosiahlo takéto hlboké zasunutie, musí byť zároveň natočené relatívne kolmo voči palete. Ako je vidieť na grafe (Obrázok 33), natrénovanie po tejto zmene trvalo o niečo dlhšie.

Graf zároveň zachytáva priemerný počet nárazov. V rámci grafov s počtom nárazov je potrebné si uvedomiť, že ide o priemerný počet nárazov. Našli sa teda aj replikácie, kde bolo nárazov oveľa viac, ale aj také, kde nebol žiadny. To spôsobilo pohyb tejto premennej v intervale približne medzi 2 až 5. Vozidlo po zmene odmeny častejšie narážalo, avšak priebehy replikácií vyzerali lepšie ako pred zmenou odmeny, preto sme sa rozhodli ďalšie experimenty vykonávať s touto odmenou.



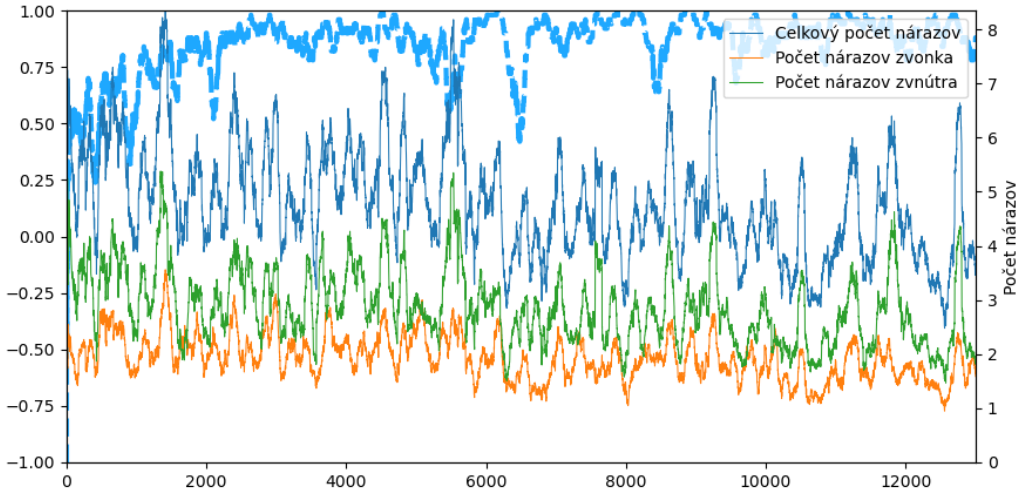
Obrázok 33 – Experiment uhol zatáčania 35 stupňov, odmena za hlbšie zasunutie lyžín

Väčší uhol otáčania kolies autonómneho vozidla v spojení so zatáčaním na zadných kolesách je lepšou aproximáciou reálneho vysokozdvižného vozíka, preto sme sa rozhodli v ďalších experimentoch používať túto konfiguráciu. Pridanie polovičných akcií síce nemalo pozitívny efekt z hľadiska rýchlosti natrénovania, avšak ani nijako nepredĺžilo tréningový čas a agentovi by tak mohlo do budúcna len pomôcť, preto sme sa rozhodli mu v ďalších úlohách umožniť tieto akcie používať.

3.5 Experimenty s prekážkou

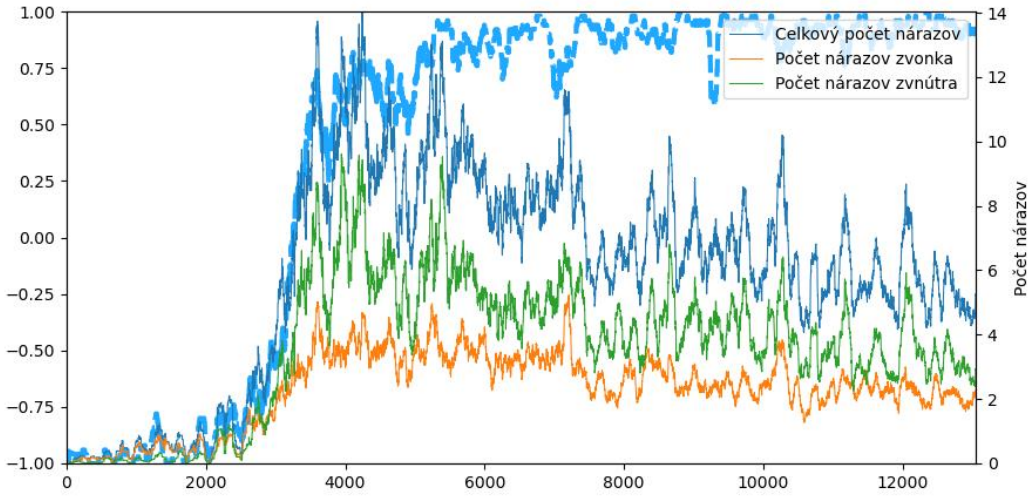
Keďže agent dokázal efektívne riešiť stanovenú úlohu, rozhodli sme sa prejsť na ťažšiu úlohu. Do prostredia sme pridali prekážku, ktorá sa náhodne generuje v strednej časti mapy vždy na náhodnej pozícii medzi pravým a ľavým okrajom. Aby bola úloha ešte komplikovanejšia, pridali sme náhodné natáčanie. Pri každej replikácii sa na začiatku pre agenta, cieľ aj prekážku samostatne vygeneruje náhodný uhol z intervalu -15 až 15 stupňov a o tento uhol sa natočia. Ak agent nabúra do palety, ktorá reprezentuje prekážku, replikácia končí s odmenou -1.

Prvotný experiment sme spravili tak, že sme najskôr agenta natrénovali na predchádzajúcej úlohe s fixnou polohou agenta, bez prekážky a bez natáčania na 10 000 replikáciách. S takto natrénovaným agentom sme následne vyskúšali nový tréning na novej úlohe. Ako je vidieť na grafe (Obrázok 34), agent sa dokázal pomerne rýchlo naučiť riešiť túto úlohu.



Obrázok 34 – Experiment s prekážkou – použitie natrénovanej siete na úlohe bez prekážky

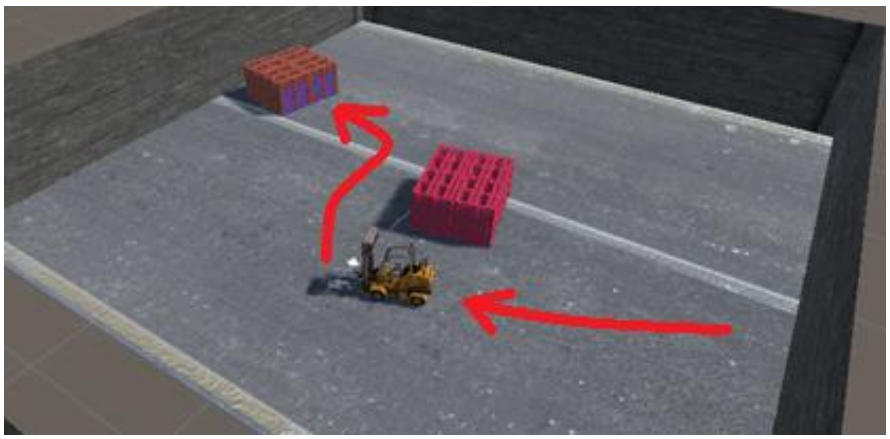
Zaujímalo nás, ako bude vyzerať tréning v prípade, že ho spustíme v tomto nastavení, ale s náhodne inicializovaným agentom. Na predchádzajúcej úlohe trvalo priemerné natrénovanie približne 2000 replikácií (Obrázok 33). Na úlohe s prekážkou trvá natrénovanie na priemerné skóre (približne 0,8 - čo zodpovedá 90% úspešnosti) až 6000 replikácií (Obrázok 35).



Obrázok 35 – Experiment s prekážkou – tréning s náhodne inicializovanou sieťou

Nižšie skóre môže byť spôsobené aj tým, že v rámci úlohy s prekážkou dochádza k situáciám, ktoré je pre agenta oveľa ťažšie riešiť – napríklad sa môže vygenerovať paleta reprezentujúca prekážku tak, že agent z počiatočnej pozície vôbec nevidí cieľ (Obrázok 36). Druhým problémom je, že prekážku je pomerne ťažké obísť a následne sa agent musí

na úzkom priestore narovnať tak, aby dokázal kolmo trafiť do otvorov v palete. Pri tomto manévri je pomerne ľahké zadnou časťou vozíka nabúrať do prekážky.



Obrázok 36 – Experiment s prekážkou, príklad problematickej úlohy

3.6 Experimenty s dvoma obrázkami a hĺbkovou mapou

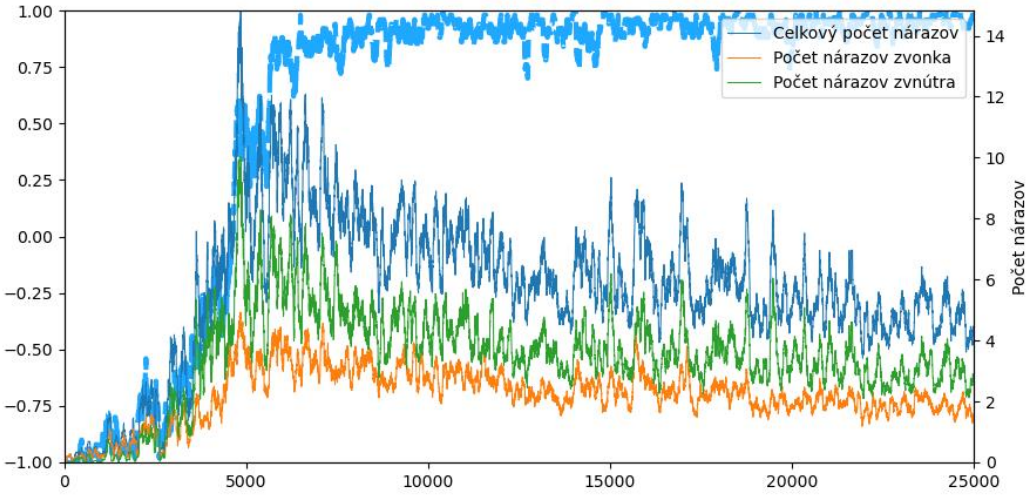
Aj úlohu s prekážkou sa nakoniec agentovi podarilo riešiť pomerne efektívne, avšak nedosahoval stopercentnú úspešnosť. Zároveň z pozorovania priebehov hry sme zistili, že pohyby agenta nie sú úplne plynulé. Chceli sme preto zistiť, aký vplyv by malo na výsledky, ak by sme mu poskytli viac informácií o aktuálnom stave. V prípade prvého experimentu sme agentovi neposielali len informáciu o aktuálnom stave – teda aktuálnu snímku z kamery, rýchlosť a natočenie kolies, ale aj snímku, natočenie a rýchlosť z predchádzajúceho stavu.

Aby mali pre agenta dodatočné informácie zmysel, bolo dôležité, aby bol medzi snímkami vždy identický časový interval. Rozhodli sme sa preto detailnejšie zamerať na fungovanie synchronizácie. Zistili sme, že na začiatku synchronizácia funguje dobre a komunikácia medzi agentom a prostredím funguje podľa nastavenia, avšak v neskorších častiach tréningu dochádza k tomu, že agent nestíha odpovedať prostrediu v stanovenom časovom intervale, a preto sa časový interval medzi rozhodnutiami môže líšiť. Je to zrejme spôsobené prehriatím grafickej karty počas tréningového procesu, ktorá následne zníži svoj výkon. Tento problém je možné riešiť viacerými spôsobmi. Najjednoduchšie je znížiť zrýchlenie simulácie voči reálnemu času tak, aby agent stíhal odpovedať včas a interval medzi správami tak zostal identický alebo použiť výkonnejší hardware. V týchto prípadoch by bol problém vyriešený na hardwarovej úrovni a bolo by treba manuálne sledovať, či synchronizácia počas celého behu funguje dobre. Ďalšou možnosťou, kedy aplikácia zaistí stopercentnú synchronizáciu automaticky, by bolo zastaviť čas pre daný vozík, ak správa

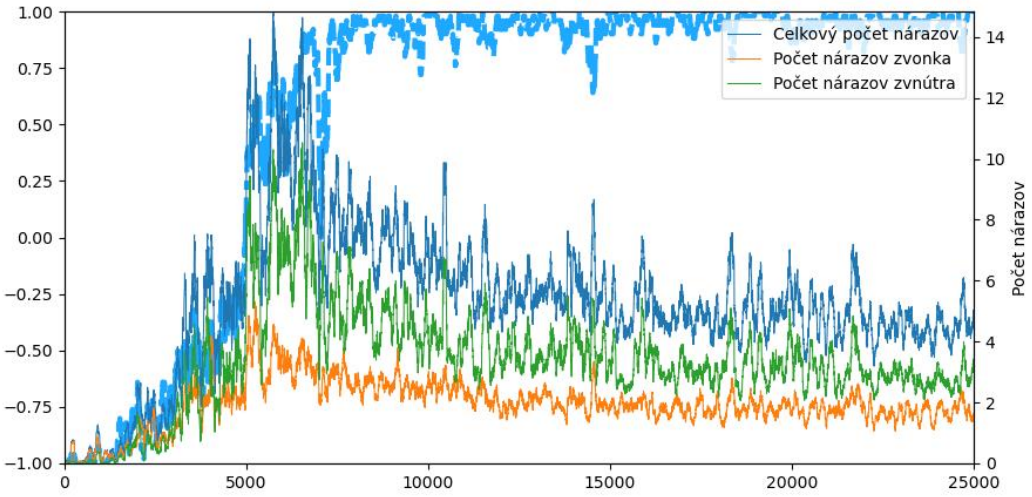
od agenta nedorazí včas. V našom prípade sa však trénuje viacero vozíkov súčasne a je potrebné zastaviť čas len pre predmetný vozík, ktorý čaká na správu. Toto je možné vyriešiť tak, že ak správa nedorazí včas, aktuálna poloha a rýchlosť autonómneho vozidla sa zapamätajú do dočasnej premennej. Následne, keď správa dorazí, pred jej spracovaním sa vozíku nastaví poloha a rýchlosť z tohto časového intervalu. Vozík sa tak vráti v čase do okamihu, kedy mala správa doraziť a z pohľadu tréningu by sa tak mala javiť synchronizácia tak, akoby bola dokonale presná. V praktickom použití by tento problém nenastával ani na stredne výkonnom hardwari, pretože hardware agenta by spracovával na nezrýchlenom čase oveľa menšie množstvo správ.

V rámci synchronizácie sme vykonali viacero experimentov a ukázalo sa, že pri dokonalej synchronizácii má frekvencia správ zásadný vplyv na rýchlosť tréningu agenta. Ak sú správy príliš časté, potom pri náhodnej inicializácii váh neurónovej siete vozidlo dostáva náhodné povely, čo spôsobí, že prejde len veľmi malú trasu a má len malú šancu trafiť do cieľa a získať tak od prostredia cennú spätnú väzbu. Naopak pri príliš veľkom časovom rozpätí medzi akciami vozidlo vykonáva jednu akciu príliš dlho a agent nemá dostatočnú presnosť, aby vedel úlohu spoľahlivo riešiť. Toto vedie k sekanému pohybu vozidla.

Vývoj skóre v rámci experimentu s pridaním druhej snímky do stavového priestoru je veľmi podobný ako v prípade jednej snímky (Obrázok 37). V rámci riešenia našej úlohy sa neukázalo, že by priniesol zásadné zlepšenie. Domnievame sa, že je to spôsobené tým, že sa v priestore simulácie nenachádzajú žiadne dynamické objekty, pri ktorých by agent s využitím dvoch snímok mohol lepšie odhadnúť smer ich pohybu. Pôvodne sme si mysleli, že dve informácie o rýchlosti mu pomôžu určiť jeho aktuálne zrýchlenie alebo spomalenie, rýchlosť autonómneho vozidla pre túto úlohu však zrejme nie je dostatočne vysoká na to, aby to malo vplyv na priebeh tréningu.



Obrázok 37 – Experiment so stavovým priestorom, ktorý obsahoval dva obrázky



Obrázok 38 – Experiment so stavovým priestorom, pridanie hĺbkovej mapy

Podobné výsledky je možné pozorovať aj pri pohľade na tréning s pridaním štvrtého rozmeru obrázka obsahujúceho hĺbkovú mapu (Obrázok 38). Po natrénovaní sa správanie agenta s využitím jedného obrázka, dvoch obrázkov alebo obrázka spolu s hĺbkovou mapou zásadne nelíšia, ale konfigurácie s hĺbkovou mapou a dvoma obrázkami potrebovali na natrénovanie väčší počet replikácií.

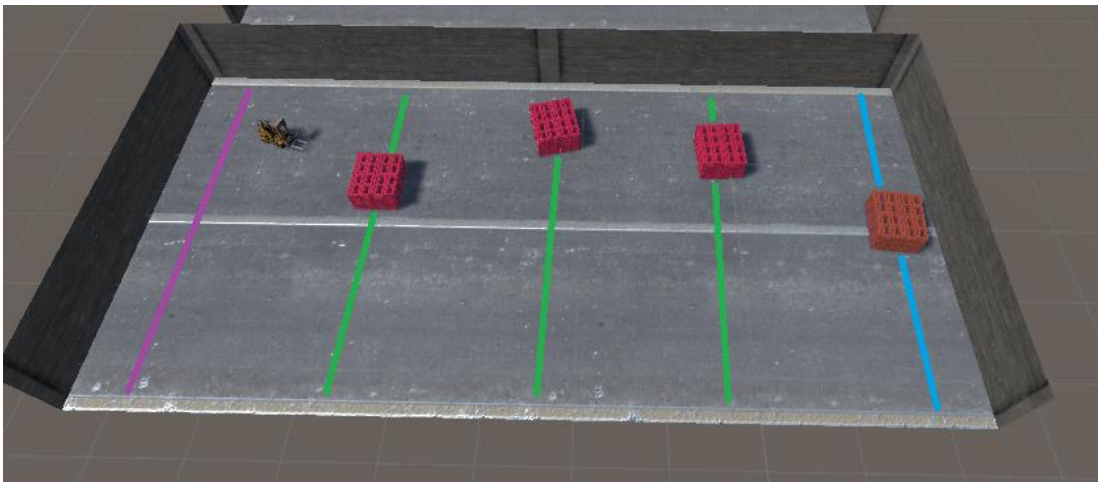
3.7 Adaptácia agenta na nové prostredie

Natrénovaný agent dosahoval na nami stanovenej základnej úlohe dobré výsledky. V záverečnej časti práce sme sa rozhodli otestovať, ako bude daný agent reagovať na

zmenu podmienok. Simulátor môže byť prvým krokom trénovania agentov určených do reálneho prostredia. Schopnosť generalizovať preto môže zvýšiť praktické využitie takto trénovaných agentov.

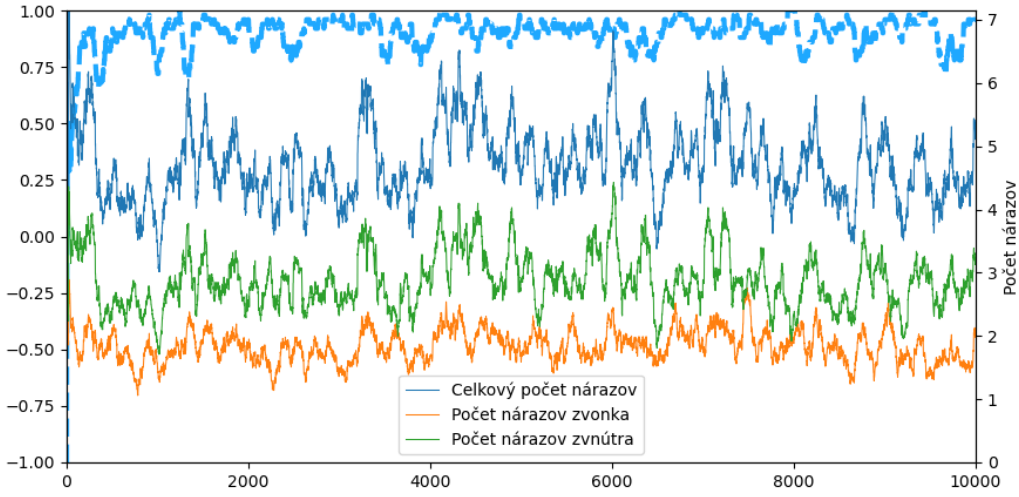
3.7.1 Pridanie viacerých prekážok

Úloha, ktorú sa agent naučil riešiť je pomerne jednoduchá. V praxi by zrejme autonómne vozidlo potrebovalo prejsť dlhšiu vzdialenosť a vyhnúť sa viacerým prekážkam, preto nás zaujímalo, aké výsledky by agent dosahoval na zložitejšej úlohe. Pridali sme preto dve ďalšie palety ako prekážky. Poloha všetkých troch prekážok sa generuje pre každú úlohu náhodne na zelených úsečkách. Poloha agenta sa generuje náhodne na fialovej úsečke a poloha cieľovej palety, do ktorej má agent trafiť lyžinami, sa generuje na modrej úsečke. Okrem náhodne zvolenej polohy sa každému objektu na začiatku vygeneruje náhodné natočenie z intervalu <-15, 15> stupňov.



Obrázok 39 – Úloha s pridaním viacerých prekážok

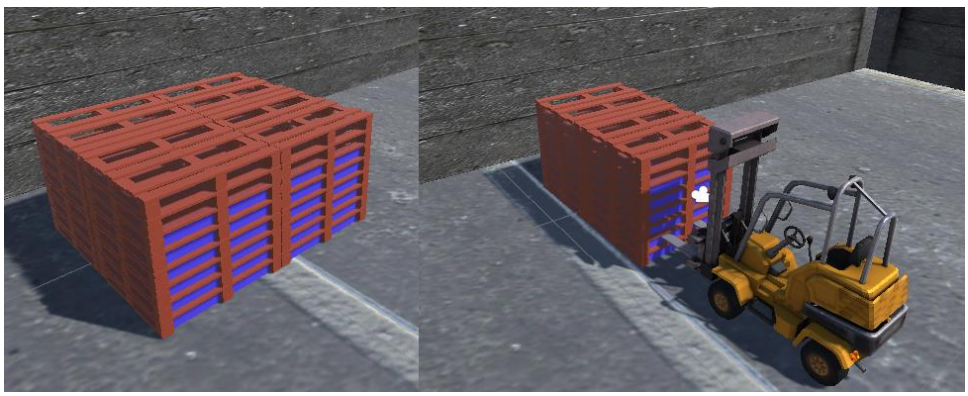
Do agenta bola načítaná neurónová sieť predtrénovaná na úlohe s jednou prekážkou. Následne sme s týmto agentom spustili dotrénovanie. Experiment ukázal, že pridanie nových prekážok nemá vplyv na výkonnosť agenta (Obrázok 40). Ten je schopný už od prvých replikácií riešiť upravenú úlohu rovnako efektívne ako pôvodnú úlohu.



Obrázok 40 – Experiment s troma prekážkami

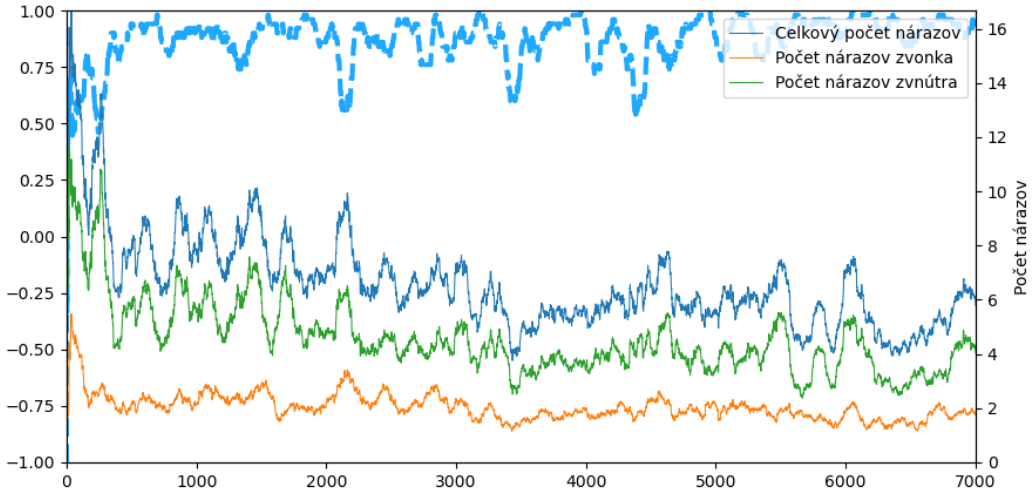
3.7.2 Zúženie palety

Pri doteraz vykonaných úlohách platilo, že ak chcel vozík dokončiť úlohu úspešne, musel zasunúť lyžiny do palety. V skutočnosti bol cieľ zložený z dvoch paliet vedľa seba. Existovali tak tri možnosti, ako mohol lyžiny zasunúť – do ľavej časti, do stredu alebo do pravej časti. V praxi by zrejme bolo potrebné identifikovať jednu paletu a zasunúť lyžiny do nej, preto nás zaujímala aj táto situácia.



Obrázok 41 – Vizuálny vzhľad palety pred zúžením a po zúžení

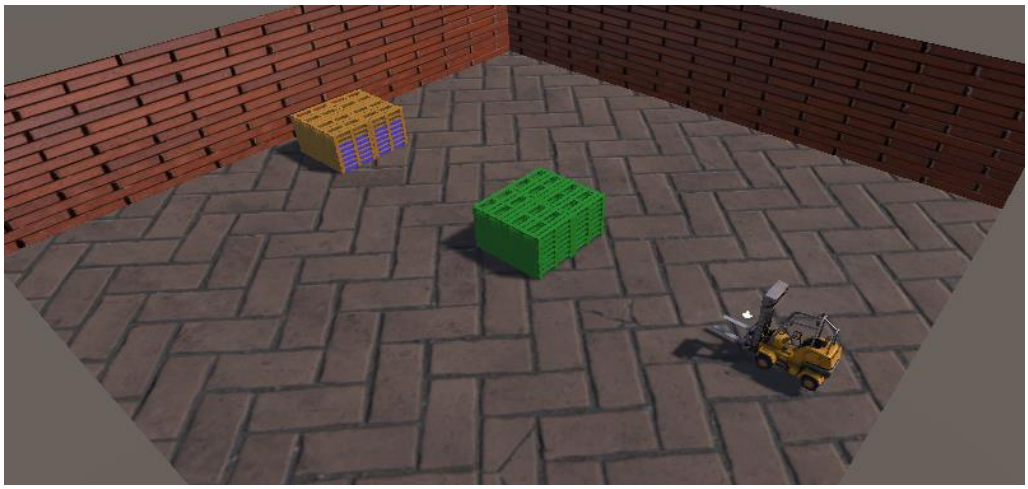
Experiment prebiehal tak, že sme spustili tréning, kde bol agent inicializovaný s neurónovou sieťou natrénovanou na úlohe so širšou paletou. Ako je vidieť na nasledujúcom grafe (Obrázok 42), agent sa vedel veľmi rýchlo adaptovať na takto pozmenenú úlohu s užšou paletou a jeho výsledky boli už po približne 300 replikáciách v podstate identické ako na úlohe so širšou paletou.



Obrázok 42 – Experiment so zúžením palety

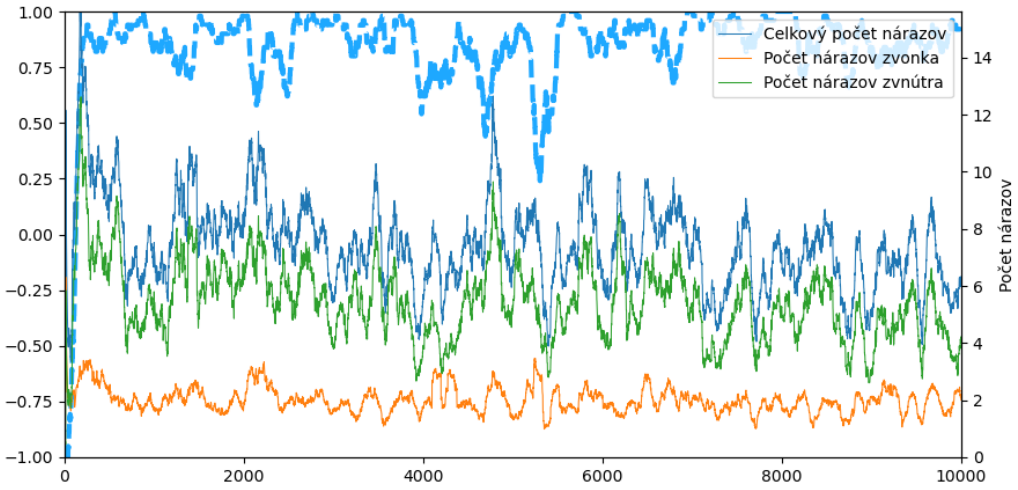
3.7.3 Zmena vizuálnej podoby prostredia

Väčšina vstupov pre agenta, ktoré slúžia na reprezentáciu stavu prostredia, v ktorom sa autonómne vozidlo nachádza, sú obrazové dáta. V praxi sa môže vizuálna podoba prostredia meniť – napríklad sa zmení osvetlenie skladu alebo vznikne potreba použiť vozidlo v inom sklade, kde budú mať steny alebo podlaha inú farbu, prípadne inú povrchovú úpravu.



Obrázok 43 – Prostredie s novými textúrami

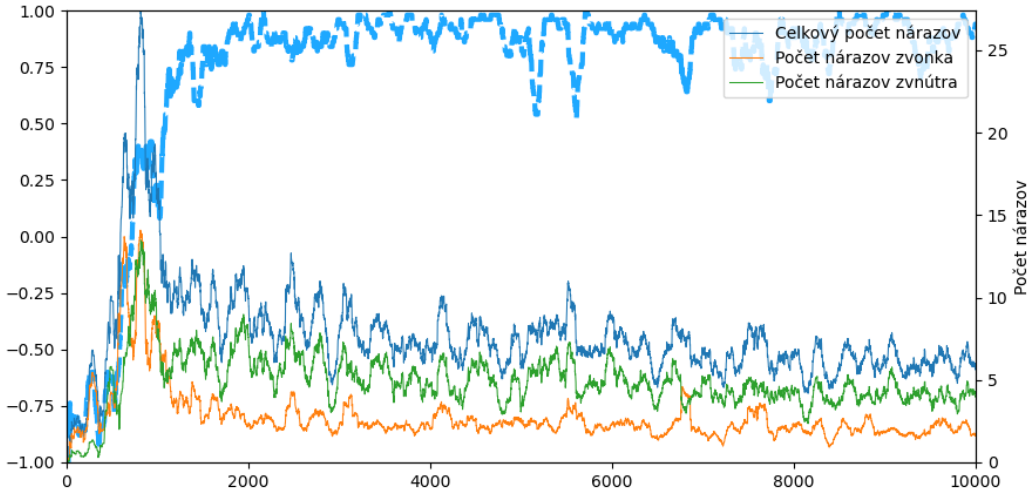
Skúsili sme preto overiť, aké výsledky bude agent dosahovať na úlohe, kde sú zmenené textúry objektov. Agent sa dokázal na zmenenú úlohu pomerne rýchlo adaptovať (Obrázok 44), avšak predpokladom bolo, aby vyznačený vstup do palety, kam sa pokúša trafiť, zostal bez zmeny.



Obrázok 44 – Experiment so zmenou textúr

3.7.4 Odobratie zvýraznenia vstupu do palety

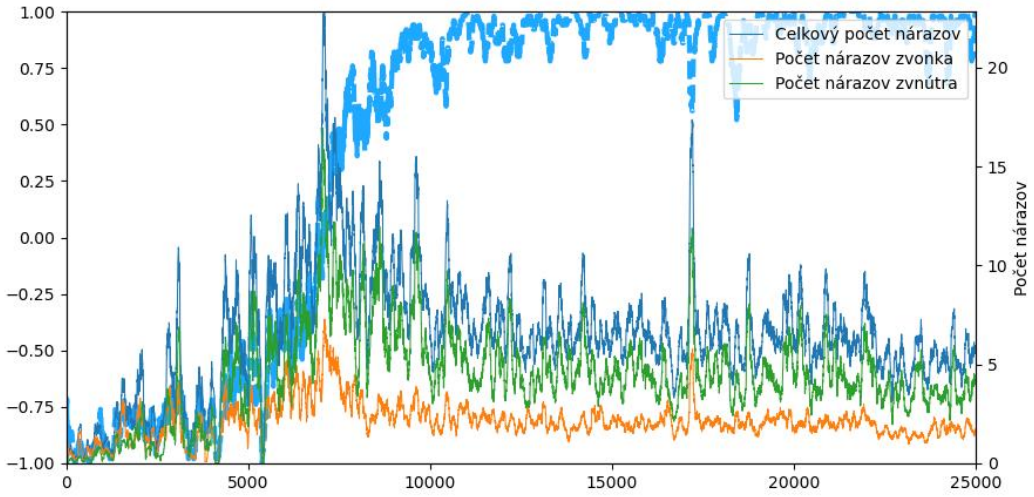
Skúsili sme vykonať tento experiment aj v upravenej verzii, kde sa zvýraznenie vstupu do palety úplne odstránilo. Výsledok experimentu spusteného s agentom predtrénovaným na pôvodnej úlohe pred zmenou textúr je vidieť na nasledujúcom grafe (Obrázok 42).



Obrázok 45 – Experiment s odstránením zvýraznenia otvoru v palete – predtrénovaná sieť

Agent na začiatku dosahoval veľmi zlé skóre a prispôsobiť sa na novú úlohu mu trvalo až necelých 2000 replikácií. Vzhľadom na to, aké komplikované bolo prispôsobiť sa na túto úlohu pre predtrénovaného agenta, sme chceli vyskúšať, koľko replikácií by potreboval nepredtrénovaný agent, aby sa naučil riešiť takto zmenenú úlohu. Najskôr sme

to skúsili s agentom, ktorý používa len jeden obrázok. Ten ani po 10 000 replikáciách nedokázal prekonať skóre -0,8. Následne sme skúsili agenta, ktorý má k dispozícii hĺbkovú mapu. Tento agent sa úlohu naučil riešiť, ale trvalo mu to až 10 000 replikácií, čiže skoro dvakrát toľko ako keby bol vstup zvýraznený (Obrázok 46). Z časového hľadiska len 10 000 replikácií trvalo viac ako 24 hodín, keďže replikácie so zlým skóre trvajú dlhšie ako replikácie s dobrým skóre, kde vozidlo hneď trafí do cieľa. Zaujímavý je aj počet nárazov, ktorého maximum počas trénovania bolo skoro dvakrát vyššie v porovnaní s úlohami so zvýrazneným vstupom do palety. Po natrénovaní sa už počet nárazov nelíšil.



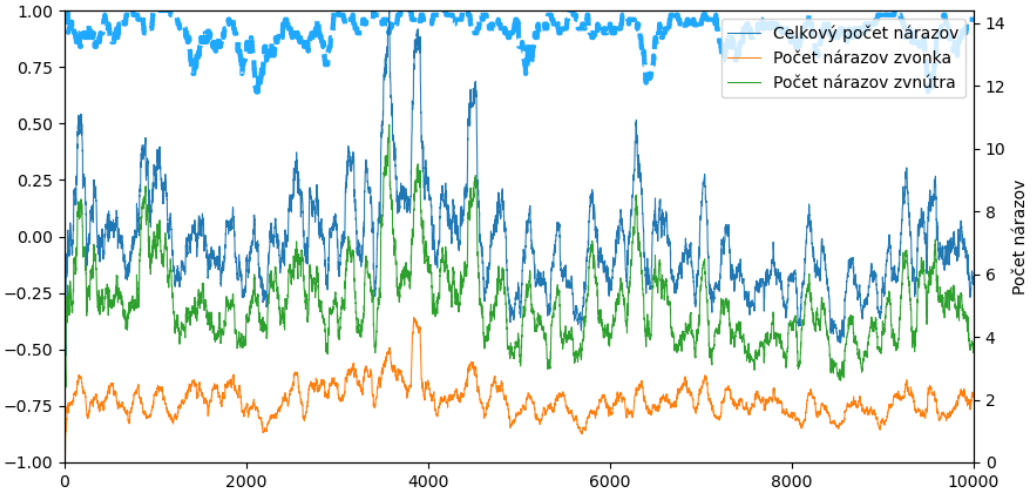
Obrázok 46 – Experiment s odstránením zvýraznenia otvoru v palete

Z tohto experimentu vyplýva, že ak by sme chceli v praxi natrénovať agenta ovládajúceho autonómne vozidlo na interakciu s nejakým objektom, je vhodné agenta natrénovať na jednoduchšej úlohe. Napríklad ak prax vyžaduje interakciu s objektom bez označenia, je ľahšie najskôr natrénovať agenta na jednoduchšej úlohe s označením a následne ho dotrénovať na úlohe bez označenia ako ho od nuly trénovať na ťažšej úlohe.

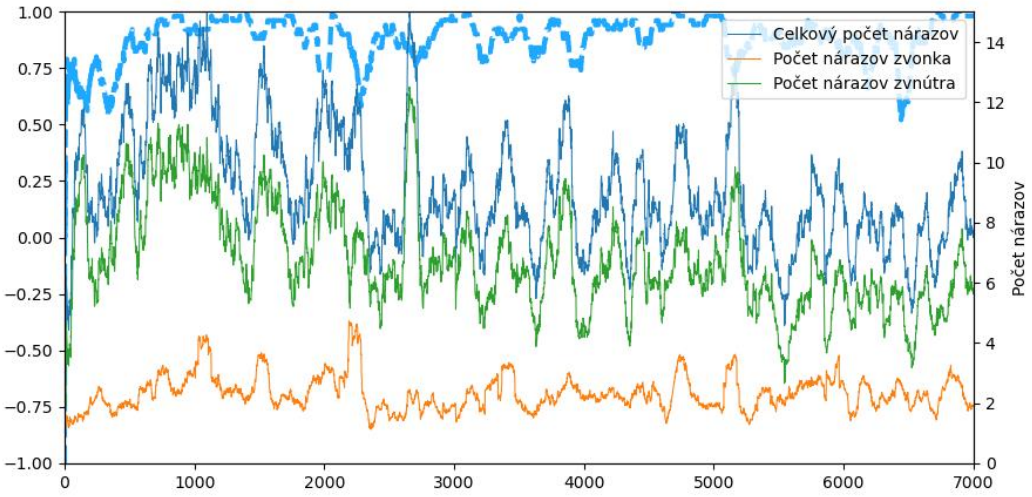
3.7.5 Zmena vlastností vozidla

Simulácia, v ktorej trénujeme autonómne vozidlo, je len aproximáciou reality a môže sa stať, že niektoré vlastnosti vozidla budú v simulácii iné ako v realite. Zaujímalo nás preto, ako sa bude natrénovaný agent správať pri riešení úlohy, ak pozmeníme niektorú vlastnosť vozidla. Tento set experimentov sa na prvý pohľad môže javiť podobný experimentom z časti 0. Hlavný rozdiel je však v tom, že u týchto experimentov chceme overiť citlivosť natrénovaného agenta na zmeny v správaní autonómneho vozidla a jeho schopnosť sa týmto zmenám prispôsobiť, neoverujeme priebeh tréningu náhodného agenta

pri rôznych nastaveniach autonómneho vozidla. Namiesto toho overujeme tréning najlepšieho natrénovaného agenta na mierne upravenej verzii úlohy.



Obrázok 47 - Experiment – zvýšenie akcelerácie vozidla o 30%



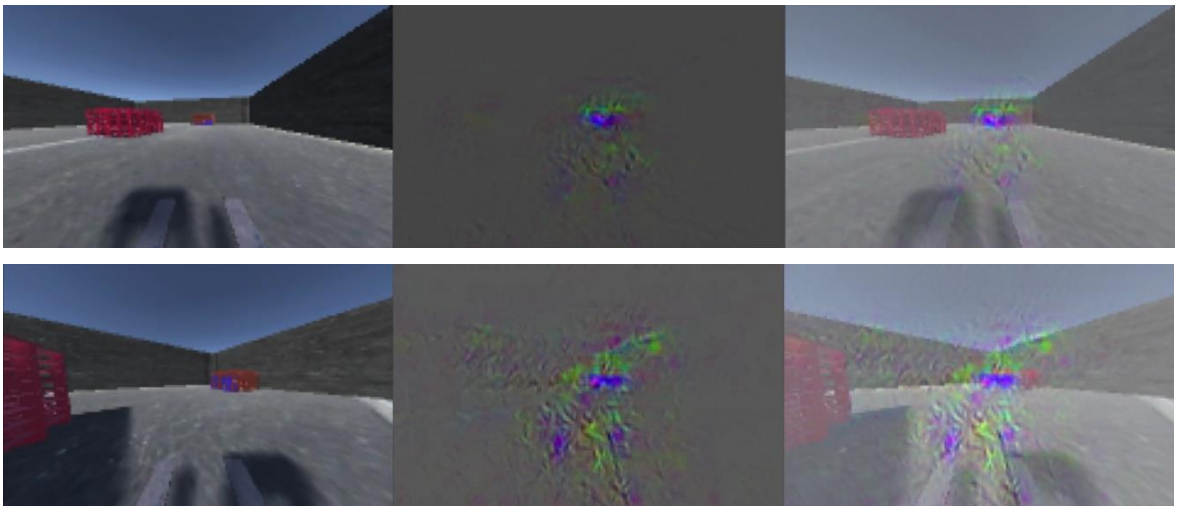
Obrázok 48 – Experiment – zmena zatáčania zo zadnej nápravy na prednú nápravu

Zvýšenie akcelerácie o 30% nemalo na výkonnosť agenta žiaden vplyv (Obrázok 47), viedlo len k tomu, že vozidlo sa pohybovalo o niečo rýchlejšie. V rámci druhého experimentu sme natrénovali agenta so zatáčaním kolies na zadnej náprave a chceli sme vidieť, ako efektívne bude vedieť agent ovládať vozidlo, keď bude zatáčanie na prednej náprave. Aj v tomto prípade vedel upravený agent úlohu riešiť efektívne (Obrázok 48). Zmena fyziky vozidla ale nemôže byť príliš výrazná. Skúšali sme vykonať aj experiment,

kde sa akcelerácia zvýšila o 150%. V tomto prípade už agent nebol schopný úlohu riešiť a nenatrénoval sa ani po 10 000 replikáciách.

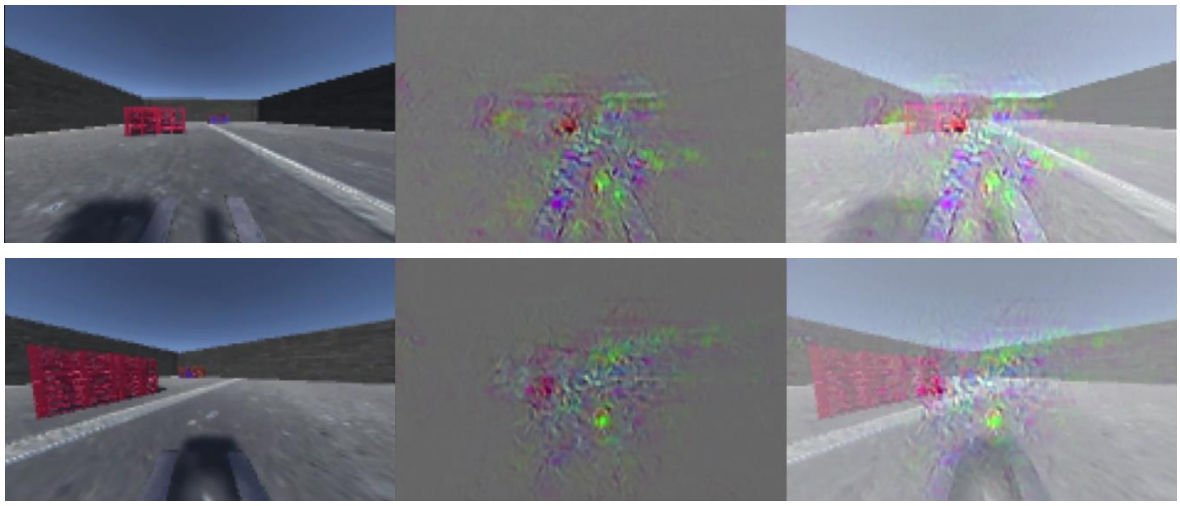
3.8 Vysvetliteľnosť cez Vanilla gradient

Hoci séria experimentov so zafarbením vstupu do palety naznačila, na aké príznaky v rámci stavového priestoru sa agent po natrénovaní zameriava, rozhodli sme sa využiť v rámci vysvetliteľnosti aj metódu Vanilla gradient. Gradienty sa v každom stave menia, preto sme do práce vybrali len niekoľko typových situácií z prostredia. Na ľavom obrázku je možné vidieť vstup do neurónovej siete, teda pôvodný obrázok, ktorý vidí agent. V strede sa nachádza zobrazenie samotných gradientov vo forme saliency mapy a na obrázku vpravo je pre lepšie pochopenie pôvodný obrázok v bledších farbách a nad ním saliency mapa.



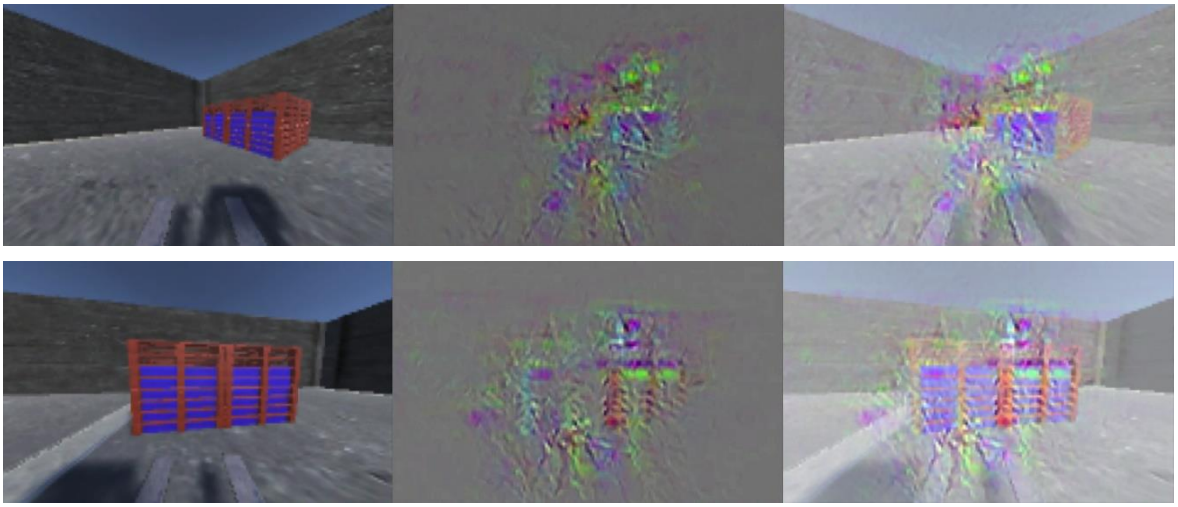
Obrázok 49 – Vanilla gradient – identifikácia cieľa z diaľky

Prvá situácia (Obrázok 49) zachytáva okamih, kedy agent smeruje k cieľovej palete, ktorá je relatívne ďaleko a prekážka mu nestojí v ceste. Vybratá akcia pre tento stav je pohyb dopredu a gradienty sú na cieli.



Obrázok 50 – Vanilla gradient – obchádzanie prekážky

Druhá situácia (Obrázok 50) reprezentuje moment, kedy agent vidí cieľ, ale zároveň si „uvedomuje“, že by mohol na ceste k nemu naraziť do prekážky, takže vybral akciu zatočiť doprava. Silné gradienty sú v tomto prípade na hrane prekážky.



Obrázok 51 – Vanilla gradient – identifikácia cieľa v blízkosti

Tretia situácia (Obrázok 51) zachytáva moment, kedy je už agent v blízkosti palety. V tomto okamihu už rozhodujú detaily, pretože autonómne vozidlo sa musí natočiť podľa otvorov v palete. Z obrázku je zreteľné, že najsilnejšie gradienty sú na otvoroch v palete, do ktorých sa agent snaží trafiť. Gradienty sa však objavujú aj na iných objektoch v rámci obrazu - napríklad na hrane medzi stenou a podlahou, ale aj na lyžinách.

Záver

V rámci práce sme sa zamerali na využitie reinforcement learningu pri riadení autonómneho vozidla v simulačnom prostredí na základe obrazových dát. Vytvorili sme prostredie v engine Unity, ktoré predstavuje zjednodušenú podobu skladu, v ktorom sa môže pohybovať vysokozdvižný vozík. Následne sme implementovali komunikáciu reinforcement learning agenta s prostredím pomocou systému ROS 2. Základnú úlohu sme skúsili natrénovať pomocou DQN agenta a niektoré nedostatky sme vyriešili prechodom na agenta na báze PPO. Pre lepšiu rýchlosť tréningu sme pridali paralelizáciu, aby sa mohlo trénovať viacero agentov súčasne.

V rámci experimentov sme overili, aký vplyv majú rôzne spôsoby výpočtu odmeny pre agenta na rýchlosť natrénovania a kvalitu ovládania autonómneho vozidla, pričom sme dospeli k záveru, že ideálne je agentovi posielať veľkú terminálnu odmenu a veľmi malé čiastkové odmeny. Takisto je dôležité aby boli situácie s veľkou kladnou a zápornou odmenou dostatočne odlíšiteľné, inak to vedie k neschopnosti agenta sa natrénovať ako v prípade, keď sme dávali odmenu -1 aj za narazenie do steny palety vedľa otvoru, kam mali byť zasunuté lyžiny.

Agent bol po natrénovaní na nami definovanej úlohe schopný ovládať vysokozdvižný vozík tak, aby prešiel k cieľovej palete a zasunul do nej lyžiny. V rámci navigácie k tejto cieľovej palete sa dokázal vyhnúť inej palete predstavujúcej prekážku a takisto dokázal nenabúrať do steny. Ďalšie experimenty preukázali, že natrénovaný agent nie je vyslovene naučený na riešenie konkrétnej úlohy, ale dokáže efektívne riešiť aj jej pozmenené verzie, napríklad po pridaní viacerých rovnako vyzerajúcich prekážok. Zistili sme, že v prípade drobných zmien v textúrach dokáže agent úlohu riešiť stále efektívne, čo znamená, že aj ak by tréning v simulačnom prostredí nedokázal potenciálne úplne nahradiť tréning v reálnom sklade, simulačné prostredie by bolo vhodné minimálne na predtrénovanie.

Z experimentov vyplýva, že v prípade trénovania ovládania autonómneho vozidla na konkrétnej úlohe, je lepšie začať trénovať na zjednodušenej úlohe a takto natrénovaného agenta následne využiť na dotrénovanie pri zložitejších úlohách. Príkladom je pridanie viacerých prekážok, kedy by bolo pre agenta veľmi ťažké náhodnými pohybmi dosiahnuť cieľ pri dlhom sklade a troch paletách a potreboval by na to oveľa väčšie množstvo

replikácií, zatiaľ čo pri natrénovaní na kratšom sklade a jednej prekážke dokázal automaticky riešiť aj úlohu s viacerými prekážkami. Ďalším príkladom môže byť zvýraznenie vstupu do palety, kedy trvalo oveľa kratší čas najskôr naučiť agenta na zvýraznenom vstupe a následne ho dotrénovať na tej istej úlohe bez farebne zvýrazneného vstupu do palety.

V budúcnosti by bolo možné vykonať ďalšiu sadu experimentov, kde by sa do stavového priestoru okrem obrázkov a informácií o rýchlosti a natočení kolies mohli pridať aj ďalšie informácie, ktoré by mohli potenciálne zlepšiť plynulosť pohybu agenta. Mohlo by ísť napríklad o informáciu o tom, aká akcia bola vykonaná naposledy. Takisto by bolo možné experimentálne overiť, či by bol agent natrénovaný na statických prekážkach schopný riešiť aj upravenú úlohu, v rámci ktorej by sa prekážky pohybovali.

Do budúcna by bolo zaujímavé preskúmať, do akej miery by bol schopný nami natrénovaný agent riešiť ovládanie skutočného autonómneho vozidla. Pre tento prípad by sme ale v ideálnom prípade potrebovali prístup do skutočného skladu, kde by bolo potrebné urobiť fotografie a z nich extrahovať textúry, rozmery objektov a ich polohu aby testovacie prostredie bolo čo najpresnejšou reprezentáciou skutočného prostredia. Takisto by bolo vhodné správanie vozidla v simulácii upraviť tak, aby jeho fyzikálne vlastnosti ako akcelerácia a zatáčanie kolies čo najvernejšie reflektovali skutočné vlastnosti vozidla.

Zoznam použitej literatúry

[1] OpenAI, „Introduction to RL,“ [Online]. Available: https://spinningup.openai.com/en/latest/.

[2] C. J. Watkins, „Technical Note Q-Learning,“ 1992. [Online]. Available: https://link.springer.com/content/pdf/10.1007/BF00992698.pdf.

[3] R. S. Sutton a A. G. Barto, Reinforcement Learning: An Introduction, Cambridge: MIT Press, 2018.

[4] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, Martin Riedmiller, „Playing Atari with Deep Reinforcement Learning,“ 2013. [Online]. Available: https://arxiv.org/pdf/1312.5602.pdf.

[5] J. Schulman, F. Wolski, D. Prafulla, A. Radford a O. Klimov, „Proximal Policy Optimization Algorithms,“ 2017. [Online]. Available: https://arxiv.org/abs/1707.06347.

[6] OpenAI, „OpenAI Five defeats Dota 2 world champions,“ [Online]. Available: https://openai.com/research/openai-five-defeats-dota-2-world-champions.

[7] Franco Terranova, M. Voetberg, Brian Nord, Amanda Pagul, „Self-Driving Telescopes: Autonomous Scheduling of Astronomical Observation Campaigns with Offline Reinforcement Learning,“ 2023. [Online]. Available: https://arxiv.org/pdf/2311.18094.pdf.

[8] Ghadi Nehme, Tejas Y. Deo, „Safe Navigation: Training Autonomous Vehicles using Deep Reinforcement Learning in CARLA,“ 2023. [Online]. Available: https://arxiv.org/pdf/2311.10735.pdf.

[9] Taylan Kabbani, Ekrem Duman, „Deep Reinforcement Learning Approach for Trading Automation in The Stock Market,“ 2022. [Online]. Available: https://arxiv.org/pdf/2208.07165.pdf.

[10] Kaiming He a kolektív, „Deep Residual Learning for Image Recognition,“ 2015. [Online]. Available: https://arxiv.org/abs/1512.03385.

[11] S. Andrew, „Saliency Maps for Deep Learning: Vanilla Gradient,“ 20 8 2019. [Online]. Available: https://andrewschrbr.medium.com/saliency-maps-for-deep-learning-part-1-vanilla-gradient-1d0665de3284.

[12] Karen Simonyan, Andrea Vedaldi a Andrew Zisserman, „Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps,“ [Online]. Available: https://arxiv.org/pdf/1312.6034.pdf.

[13] Steve Macenski, Tully Foote, Brian Gerkey, Chris Lalancette a William Woodall, „Robot Operating System 2: Design, Architecture, and Uses In The Wild,“ 2022. [Online]. Available: https://arxiv.org/pdf/2211.07752.pdf.

[14] GAZEBO, „Gazebo Sim,“ [Online]. Available: https://gazebosim.org/features.

[15] No Such Dev, „All about time in Unity Game Engine,“ 2020. [Online]. Available: https://nosuchstudio.medium.com/all-about-time-in-unity-game-engine-4c290d2772c7.

[16] Unity, „Namespace Unity.MLAgents.Sensors,“ [Online]. Available: https://docs.unity3d.com/Packages/com.unity.ml-agents%401.0/api/Unity.MLAgents.Sensors.html. [Cit. 5 10 2023].

[17] RobotecAI, „https://github.com/RobotecAI/ros2-for-unity,“ [Online]. [Cit. 3 10 2022].

[18] Open Robotics, „ROS2 Documentation: Humble,“ 2 11 2023. [Online]. Available: https://docs.ros.org/en/humble/Installation/Ubuntu-Install-Debians.html.

Zoznam príloh

Príloha A: DVD so zdrojovými kódmi a prácou v elektronickej podobe