TECHNICKÁ UNIVERZITA V KOŠICIACH
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Formálne metódy a strojové učenie vo virtuálnom priestore: autonómna entita

Bakalárska práca

2019 **Daniel Slinčák**

TECHNICKÁ UNIVERZITA V KOŠICIACH
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Formálne metódy a strojové učenie vo virtuálnom priestore: autonómna entita

Bakalárska práca

Študijný program:	Informatika
Študijný odbor:	Informatika
Školiace pracovisko:	Katedra počítačov a informatiky (KPI)
Školiteľ:	Ing. Štefan Korečko, PhD.

2019 Košice **Daniel Slinčák**

Abstrakt

Hlavnou náplňou bakalárskej práce je prepojenie strojového učenia, formálnych metód a ich evaluácia vo virtuálnom prostredí. To je dosiahnuté vytvorením agenta strojovým učením s posilňovaním. Správanie tohto agenta je kontrolované pomocou formálne verifikovaného komponentu napísaného v B-metóde. Tento agent je následne vložený do virtuálneho prostredia na testovanie. Virtuálnym prostredím je hra vyvinutá v hernom rámci Unity. Výsledky testovania sú spracované a je z nich vyvodený záver.

Kľúčové slova

strojové učenie, formálne metódy, virtuálne prostredie

Abstract

The main purpose of this bachelor thesis is connecting machine learning with formal methods and their evaluation in a virtual environment. This is achieved by creating an agent using reinforcement machine learning. The agent's behavior is controlled by a formally verified component written in B-method. This agent is inserted into the virtual environment for testing. The virtual environment represents a game that is developed in the Unity engine. The test results are then processed and a conclusion is made.

Keywords

machine learning, formal methods, virtual environment

TECHNICKÁ UNIVERZITA V KOŠICIACH

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Katedra počítačov a informatiky

ZADANIE BAKALÁRSKEJ PRÁCE

Študijný odbor: **Informatika**
Študijný program: **Informatika**

Názov práce:

Formálne metódy a strojové učenie vo virtuálnom priestore: autonómna entita

Formal Methods and Machine Learning in Virtual Space: Autonomous Entity

Študent: **Daniel Slinčák**
Školiteľ: **Ing. Štefan Korečko, PhD.**
Školiace pracovisko: **Katedra počítačov a informatiky**
Konzultant práce:
Pracovisko konzultanta:

Pokyny na vypracovanie bakalárskej práce:

1. Analyzovať súčasné prístupy k prepojeniu formálnych metód a strojového učenia.
2. Na základe analýzy navrhnúť a implementovať autonómnu entitu, využívajúcu strojové učenie a formálne metódy, ktorej učenie bude prebiehať počas hrania za týmto účelom vyvinutej hry.
3. V spolupráci s riešiteľom súvisiacej záverečnej práce vyvinutú entitu integrovať do uvedenej hry, experimentálne realizovať proces učenia sa a vyhodnotiť ho.
4. Vypracovať dokumentáciu podľa pokynov vedúceho práce.

Jazyk, v ktorom sa práca vypracuje: slovenský
Termín pre odovzdanie práce: 24.05.2019
Dátum zadania bakalárskej práce: 31.10.2018

doc. Ing. Jaroslav Porubän, PhD.
vedúci garantujúceho pracoviska

TECHNICKÁ UNIVERZITA V KOŠICIACH FAKULTA ELEKTROTECHNIKY A INFORMATIKY -10-

prof. Ing. Liberios Vokorokos, PhD.
dekan fakulty

Čestné vyhlásenie

Vyhlasujem, že som celú bakalársku prácu vypracoval samostatne s použitím uvedenej odbornej literatúry.

Košice, 24. mája 2019 .....

vlastnoručný podpis

Poďakovanie

Poďakovanie patrí vedúcemu práce Ing. Štefanovi Korečkovi, PhD. za jeho pomoc a vedenie od samého začiatku, kolegovi a bratrancovi Róbertovi Rauchovi, s ktorým táto práca tvorí kompletné riešenie a všetkým kamarátom a spolužiakom, s ktorými som sa mohol poradiť o každej maličkosti a podporovali ma počas celej cesty, až do samého konca. A nakoniec chcem poďakovať aj každému čitateľovi tejto práce.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Zoznam symbolov a skratiek

PPO – Proximal Policy Optimatization

Úvod

Predtým, ako je softvér vypustený do prevádzky, musí byť dostatočné otestovaný. Či už ide o softvér používaný na bežných počítačoch alebo o softvér, ktorý bude vložený do reálneho zariadenia, kde bude mať fyzickú interakciu s ľuďmi. V takomto prípade je potrebné vytvoriť prototyp zariadenia a testovať jednotlivé fázy softvéru počas jeho vývoja. Toto úsilie môže cenovo aj časovo neúmerne rásť, čo má potom dôsledky na samotný vývoj softvéru. Jedno z riešení, ako obísť tieto problémy, je testovanie vo virtuálnom prostredí. Pod virtuálnym prostredím si môžeme predstaviť jednoduchú simuláciu reality, ktorá obsahuje zjednodušené fyzikálne vlastnosti potrebné pre správne otestovanie softvéru. Virtuálnym prostredím môže byť aj už existujúca hra, ktorej použitie bude mať viacero výhod pre testovanie a pre vývojárov softvéru.

Popularita hier rastie, herný priemysel má každým rokom väčší a väčší finančný obrat. Ak sa použije hra ako virtuálne prostredie pre testovanie, môžeme si byť istí, že do kontaktu so softvérom sa dostane dostatočné množstvo rozdielnych osôb, či už pohlavím, vekom a pracovnými skúsenosťami. Myšlienka testovania softvéru v hre tak, že hráči si neuvedomujú jeho prítomnosť bola sformulovaná v článku [28]. Vraví, že správanie hráčov bude v tom prípade veľmi blízke ich prirodzenému správaniu, vďaka čomu softvér môže byť otestovaný takým spôsobom, akým ho budú budúci zákazníci používať. Ak si predstavíme softvér na ovládanie robota, ľudia ktorí sú o ňom vedomí, sa budú správať inak ako ľudia, ktorí si myslia že to je len ďalší hráč, alebo len nehrateľná postava v hre. Ak je náhodou aj softvér takého charakteru, že pri jeho testovaní by mohlo dôjsť k úrazu na zdraví, testovanie vo virtuálnom prostredí nás zbaví týchto nebezpečenstiev. Príkladom je testovanie autonómnych automobilov, kde testovanie počas reálnej dopravnej premávky by mohlo viesť k veľkým škodám na majetku a hlavne zdraví.

1. Formulácia úlohy a cieľ práce

Cieľom práce je spojiť spomínané testovanie softvéru vo virtuálnom prostredí s ďalšími metódami vývoja softvéru, ako sú napríklad strojové učenie a použitie formálnych metód. Strojové učenie je odvetvím, ktoré za posledné roky rýchlo napreduje, či už v rámci kvalitného aplikovania na rôzne účely, alebo rozšírenia medzi verejnosťou. Náš projekt aplikuje strojové učenie na softvér pre agenta predstavujúceho robotického kuriéra, ktorý sa bude musieť bezpečne dostať cez bojisko. Virtuálnym prostredím bude nami vytvorená hra pomocou herného rámca Unity. Aplikáciou formálnych metód na verifikáciu komponentu bude ustrážené korektné a bezpečné fungovanie agenta.

2. Formálne metódy

Systémy v dnešnej dobe sú stále viac a viac komplikovanejšie [10] a s tým rastie aj výskyt chýb, ktoré môžu mať seriózne následky. Samotné strojové učenie je veľmi náchylné [9] na výskyt chýb už od začiatku implementácie, kde môže byť vybratý zlý spôsob strojového učenia, nesprávny algoritmus na učenie, použitie dát, ktoré nijak nezlepšujú kvalitu učenia, ale môžu mať dokonca aj negatívny vplyv a mnoho ďalších. Jedným z riešení ako znížiť pravdepodobnosť ich výskytu je aplikovanie formálnych metód [20], ktoré predstavujú matematické jazyky, metódy a nástroje určené na verifikovanie a overenie softvéru od návrhu až po implementáciu.

2.1. Testovanie a verifikácia

Podľa autorov článku [18] je dôležité vykonať verifikáciu systému použitím formálnych metód. Pri testovaní softvéru sú dodávané rôzne vstupy, na základe ktorých sa kontroluje správanie systému. Toto testovanie sa vykonáva v predvídateľnom prostredí, kde dané použité vstupy sú prirodzené. Keďže systémy aplikujúce strojové učenie alebo umelú inteligenciu sú častejšie aplikované v nepredvídateľných prostrediach, môže to viesť neprirodzeným vstupom. V takých prípadoch je potrebné, aby systém vedel ako zareagovať a aby jeho správanie bolo zároveň bezpečné a aj predvídateľné. Na základe tvrdení autorov použitie formálnych metód neznamená, že softvér nie je potrebné testovať. Ich aplikovanie závisí od správnej analýzy požiadaviek na systém a tiež použitím správnej metódy. Ak je jedno z toho nesprávne, vďaka testovaniu bude možné včas nájsť chybu a opraviť ju. Práve preto je pre vytvorenie najbezpečnejšieho systému dôležitá ich kombinácia.

2.2. Formálne metódy a strojové učenie

Strojové učenie a umelá inteligencia, ktoré sa zdali a stále zdajú ako ďaleká budúcnosť, sa v skutočnosti za posledné roky nenápadne rozšírili do nášho každodenného života. Toto rozšírenie ovplyvnilo viacero faktorov, ako napríklad vysoký nárast výpočtovej sily, výrazné uľahčenie práce pri vytváraní systémov a rôzne iné možnosti ich aplikovania. Či už ide o používanie vyhľadávacích nástrojov, používanie osobných asistentov, samo-riadiace vozidlá alebo analýza videozáznamov, obrazov a následná detekcia z nich. Pre všetky tieto systémy je veľmi dôležité, aby ich chybovosť bola čo najnižšia, pretože v opačnom prípade

by mohli ohrozovať aj životy ľudí. Pre systémy používané vo vozidlách a lietadlách je taktiež nevyhnutné mať aj dôkaz o ich bezpečnosti. Preto je pri vývoji systémov so strojovým učením dobrým nápadom uvažovať o aplikovaní [17] formálnych metód, ktoré môžu navýšiť korektnosť fungovania systému.

O tejto problematike sa za posledné roky konalo viacero seminárov [12, 13, 14, 15] s expertmi z oboch odvetví, kde diskutujú o súčasných bezpečnostných riešeniach a možnostiach, ako ich vylepšiť. Závery vyvodené z týchto seminárov ohľadom kombinácie formálnych metód a strojového učenia sú pozitívne. Je to niečo, čo je ešte len v zárodku, nie je to veľmi rozšírené medzi osobami v samostatných oblastiach problematiky. Výsledky kombinovania by však boli prospešné pre obe strany.

2.2.1. Formálne metódy aplikované na algoritmus strojového učenia

Pod týmto spojením sa chápe verifikácia vybraných vzorcov, pravidiel a akcií v algoritmoch, s cieľom minimalizovať chybu vo formulácií algoritmu a kontrole jeho efektivity v štádiu učenia sa. V súčasnosti sú verifikované hlavne vstupné dáta do algoritmov, ktoré môžu byť nepriateľského charakteru. Dodávanie takýchto dát sa nazýva nepriateľské správanie (adversarial behavior) a je bližšie opísané v podkapitole 3.3.1.

Takéto spojenie formálnych metód a strojového učenia je možné, ale nie veľmi efektívne. Ako bolo povedané v úvode tejto kapitoly, vstupy pri strojovom učení sú nepredvídateľné. Definovanie zakázaných hodnôt nemusí zachytiť všetky vstupy a definovanie povolených hodnôt môže veľa užitočných vstupov zahodiť.

2.2.2. Syntéza formálnych metód a strojového učenia

Ako už bolo povedané na začiatku kapitoly, systém sa verifikuje pomocou metód postavených na matematickom základe. Na verifikáciu sa dajú použiť rôzne nástroje, ktoré však očakávajú fixný a nemenný systém alebo algoritmus. Pri použití strojového učenia sa však systém a algoritmy menia a vyvíjajú vďaka učeniu. Kvôli tomu nie je možné použiť tradičné metódy a nástroje na formálnu verifikáciu, ale ich prispôsobiť alebo nanovo vytvoriť pre dané požiadavky systému. Jedna klasifikácia na základe fáze učenia predstavuje offline a online učenie.

Verifikácia pre offline učenie, alebo offline verifikácia, pracuje so systémom, ktorý sa už neučí a používa výslednú, optimálnu formu (policy). Na každý rovnaký vstup reaguje rovnako, správa sa deterministicky.

Naopak verifikácia pre online učenie, online verifikácia [16], pracuje so systémom, ktorý sa stále učí, aj v čase keď má byť verifikovaný. Keďže sa systém mení, verifikácia sa musí vykonávať za jeho behu, kde monitory strážia, snímajú a reagujú na zmenu stavov alebo vstupov, ktoré sú či už správne alebo nesprávne. Kontrola celého systému je vzhľadom na jeho časté menenie výrazne náročná, preto je výhodnejšie verifikovať len niektoré časti. Vhodné na verifikáciu sú tie časti systému, ktoré sa môžu meniť vplyvom strojového učenia, napríklad mapovanie stavov na akcie, sekvencie akcií, monitorovanie predpokladov a ich porušenia a kontrola vstupných údajov, ktoré budú súčasťou trénovania.

3. Strojové učenie

Umelá inteligencia je významným odvetím v technologickej sfére už od 50. rokov minulého storočia. Ako je povedané v článku [2], jej cieľom je vyvinúť stroje, ktoré sa správajú a rozhodujú ako ľudia a pritom majú rovnakú alebo aj väčšiu inteligenciu. Jedným z prístupov k dosiahnutiu umelej inteligencie je strojové učenie. Základným princíp je použitie zložitých algoritmov na spracovanie dát, naučiť sa z nich a tieto vedomosti využiť na analýzu alebo predikciu udalostí, javov alebo podobností medzi nimi.

Tri hlavné typy strojového učenia sú

- Samostatné učenie (unsupervised learning).
- Učenie s dozorom/učiteľom (supervised learning).
- Strojové učenie s posilňovaním (reinforcement learning).

3.1. Samostatné učenie

Pri samostatnom učení [8] má systém k dispozícií len vstupné dáta, ktoré sám analyzuje a kategorizuje. Toto učenie je používané vtedy, keď nám nie je jasné na základe akých parametrov a vlastností existuje súvislosť medzi skupinou dát.

Pre náš projekt je tento spôsob strojového učenia neaplikovateľný, keďže nám nejde o hľadanie súvislostí medzi dátami, ale naučiť agenta robiť najefektívnejšie rozhodnutia v reakcii na zmeny prostredia.

3.2. Učenie s dozorom

Učenie s dozorom [3] je postavené na dostupnosti údajov patriacich do dvoch kategórií - vstupy a výstupy z nich. Tie sa rozdelia na tréningové a testovacie. Systém sa učí z tréningových, analyzuje vstupné údaje, naučí sa z nich pravidlá a vytvorí k nim daný výstup. Následne tieto získané vedomosti aplikuje na testovacie údaje a z nich zistí úspešnosť predikcie. Nízka úspešnosť predikcie môže nastať z viacerých dôvodov. Tréningové dáta mohli byť nedostatočne účinné alebo neboli relevantné k učeniu sa na predikcie. Príkladom by bolo učenie sa rozpoznania tvári len na jednej osobe a nie na viacerých. Z takýchto vstupných údajov by sa systém naučil rozpoznávať len danú osobu, ale pri testovaní na niekom inom, by už mal veľkú chybovosť. Ďalším dôvodom môže byť

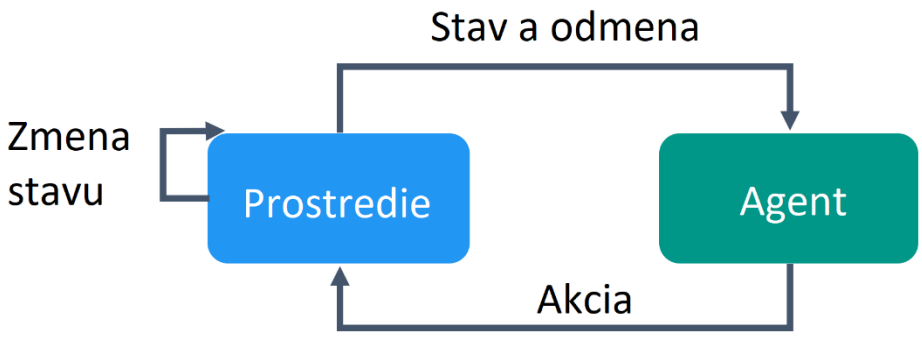
nesprávne vybratý algoritmus. Nie je jedno, či sa použije lineárna alebo exponenciálna funkcia pri výpočtoch.

Aplikovanie strojového učenia pod dozorom pre náš projekt by bolo možné, ale nebolo by ideálnym spôsobom riešenia. Takéto učenie si vyžaduje dostupnosť relevantných vstupov a výstupov systému, kde pre nášho agenta nevieme presne určiť k akému výsledku bude viesť daný vstup.

3.3. Učenie s posilňovaním

Základným prvkom strojového učenia je agent, ktorý predstavuje hocijaký objekt, ktorý je schopný vnímať svoje prostredie a vykonávať nad ním akcie.

Princíp strojového učenia s posilňovaním [5] spočíva v tom, že agent nevie čo má robiť, ale dostáva rôzne odmeny za akcie, ktoré vykoná. To znamená, že agent musí vyskúšať všetky dostupné akcie, aby našiel tú, ktorá mu dodá najväčšiu odmenu. Na obrázku 3.1 je zobrazená interakcia agenta s prostredím, v ktorom sa nachádza.



Obrázok 3.1 - Strojové učenie s posilňovaním [7]

Nevýhodou tohto učenia je, že určenie cieľu a spôsobu ako sa k nemu dostať nemusí byť veľmi jasné. Postupnosť rozhodnutí, pre ktoré sa agent rozhodne môže značne ovplyvniť výšku odmien pre nasledujúce rozhodnutia. Preto, aby agent našiel to úplne najlepšie rozhodnutie, musí prejsť každou možnou vetvou, čo je časovo aj hardvérovo náročné. Takisto nemusí prejsť cez všetky rozhodnutia, ak si myslí že dané rozhodnutie mu dalo vysokú odmenu. Bude toho názoru, že vyššia odmena už nie je možná, pretože si nemusí uvedomovať, že iná sekvencia rozhodnutí môže viesť k optimálnemu výsledku.

Pre veľkú časť komplexných úloh je strojové učenie s posilňovaním jediným možným spôsobom, ako dôjsť k riešeniu daného problému. Príkladom je naučenie agenta hrať šach. Aby bol agent čo najlepší, potreboval by odozvu na každý ťah ktorý vykonal. Vďaka tejto odozve by vedel či sa rozhodol pre správny alebo nesprávny ťah. Nie je však možné manuálne nastaviť odozvu pre tieto ťahy. Práve preto je na riešenie úloh tohto typu strojové učenie s posilňovaním najideálnejšie. Cieľom je nájsť optimálnu formu (policy), ktorá ovplyvňuje správanie agenta. Na základe tejto formy má agent viacero rôznych pozitívnych a negatívnych odmien. Tieto odmeny dostáva na základe jeho vykonaných akcií a dosiahnutých stavov. Ďalší faktor, ktorý by usmerňoval výšku odmien, môže byť časová dĺžka vyriešenia problému.

3.3.1. Nepriateľské správanie (Adversarial Behavior)

Je správanie, pri ktorom sa do systému niekto alebo niečo snaží vložiť nesprávne alebo zavádzajúce dáta za účelom negatívneho vplyvu na použitý algoritmus, čo má zároveň vplyv na robustnosť celého systému.

3.3.2. Typ prostredia

Prostredie, v ktorom sa agent nachádza môže mať rozličné vlastnosti, ktoré výrazne menia správanie sa agenta a algoritmu učenia. Delenie je prehľadne opísané v publikácií [23].

Prostredie môže byť definované počtom v ňom nachádzajúcich sa agentov. Prostredie s jediným agentom sa nazýva **Single Agent**, s viacerými **Multiagent**. Každý agent má svoj cieľ, ku ktorému sa snaží dostať. Viacerí agenti môžu mať tento cieľ rovnaký, kde cesta k nemu vedie buď spoluprácou, kde sú agenti kooperatívni, alebo naopak agenti navzájom súperia, kde pozitívny zisk jedného agenta predstavuje negatívny zisk druhého. Príkladom je šach, kde ťah, ktorý má kladný vplyv na stav hry pre jedného hráča, má naopak záporný pre druhého hráča.

Ak nasledujúci stav prostredia závisí priamo od stavu a akcií agenta, také prostredie sa označuje ako **deterministické**. Opakom je prostredie **stochaistické**, ktoré sa môže meniť bez vplyvu agenta.

Podľa spôsobu ako vníma prostredie agent, existuje **úplne vnímateľné** prostredie a **čiastočné vnímateľné**. V úplne vnímateľnom prostredí má agent v každom momente

informácie o každom jednom aspekte prostredia. Šach je v tomto prípade úplne vnímateľný, keďže agent pozná pozíciu každej hracej figúrky.

Ak nasledujúci stav agenta nie je ovplyvnený predošlými stavmi, v ktorých sa nachádzal vo svojom životnom cykle, ide o **epizodické** prostredie. Ak práve naopak, nasledujúci stav závisí od kombinácie všetkých predošlých stavov, takéto prostredie je prostredím **sekvenčným**.

Agent predtým, ako vstúpi do ďalšieho stavu na základe vykonanej akcie, si musí premyslieť a vybrať danú akciu. Ak sa počas tohto rozhodovania prostredie nemení, ale ostáva stále rovnaké, prostredie sa nazýva **statické**. Keď však musí agent zareagovať počas rozhodovania na zmenu stavu prostredia, prostredie je **dynamické**.

Prostredie môže byť pre agenta **známe** alebo **neznáme**. V známom prostredí má agent už nejaké predstavy a informácie o pravidlách a správaní prostredia. V neznámom prostredí však nemá žiadne informácie a počas svojho života sa učí na základe vykonaných akcií a ich vplyvov na svoj stav a stav prostredia. Z týchto vplyvov si vytvára predstavy o tom, aké pravidlá môžu platiť v danom prostredí.

Posledné delenie prostredia je na **diskrétne** a **kontinuálne**, ktoré hovorí o stavoch, akciách, vnemoch a vplyvu času na agenta. V kontinuálnom prostredí prechádzajú informácie plynule z jednej hodnoty na druhú. Diskrétne prostredie sa dá predstaviť ako také prostredie, kde sa nachádza konečný počet stavov a akcií agenta.

Náš projekt bude v tomto prípade predstavovať prostredie multiagentové, stochaistické, čiastočne vnímateľné, sekvenčné, dynamické, kontinuálne a známe. To predstavuje druhé najkomplikovanejšie prostredie, ktoré je zároveň najbližšie k reálnemu svetu. Zložitejším prípadom by bolo prostredie neznáme, avšak náš agent bude mať informácie aj o prostredí, aj o tom čo by mal docieliť. Práve takéto realistické prostredia sú hlavnou problematikou umelej inteligencie a strojového učenia, kde predstavujú obrovské výzvy. Existuje viacero projektov, ktoré sa snažia riešiť takéto problémy. V nasledujúcej kapitole bude opísaných zopár implementácií strojového učenia na hry, ktoré takisto spĺňajú tieto aspekty prostredia a obsahujú zložité herné pravidlá.

4. Aplikácia strojového učenia

Ako už bolo spomínané v 2.2, využitie strojového učenia je vo viacerých aspektoch nášho života. Využívané je na riešenie problémov, ktoré sú veľmi komplexné [21], na automatizáciu a vylepšovanie už vymyslených postupov a spôsobov riešenia. Takisto sa využíva aj v hernej sfére, kde je aplikované na rôzne logické hry [22] a následne postavené voči majstrom týchto hier, ako napríklad šachy alebo dámy. Tieto hry sú príťažlivé pre vývojárov, ktorí pracujú so strojovým učením z dôvodu ich komplexnosti. Aby bolo strojové učenie efektívne, malo by v danom momente dokázať prejsť všetkými možnými nasledujúcimi stavmi a z nich vybrať práve ten, ktorý bude viesť k výhre. To sa dá opísať vzorcom b^d [19], kde b predstavuje počet možných platných ťahov a d hĺbku, čiže dĺžku hry. Pri šachu platí b=35 a d=80, čo je dosť vysoké číslo. Použitie vyhľadávacích algoritmov na prehľadanie týchto stavov nemusí byť možné, alebo bude príliš náročné a časovo zdĺhavé. Využitie strojového učenia s posilňovaním môže efektívne uľahčiť vyhľadávanie.

4.1. OpenAI

Strojové učenie s posilňovaním je základom projektu OpenAI [6], ktorý využíva PPO (Proximal Policy Optimatization) algoritmus. OpenAI má viacero zameraní, jedným z ktorých je vytvorenie botov do hry Dota 2, kde hrajú proti sebe dva tímy po päť hráčov. Obsahuje cez 100 hrateľných postáv, kde každá z nich má tri a viac jedinečných schopností a rôzne vlastnosti ako silu, agilitu a inteligenciu. Z týchto vlastností vyplývajú ďalšie, pre každú postavu jedinečné aspekty, ako úroveň života, kúziel a získavanie peňazí. Za tieto peniaze je taktiež možné kupovať veľké množstvo rôznych predmetov, ktoré ovplyvňujú schopnosti aj vlastností postáv.

Skúšať hneď aplikovať strojové učenie na takúto komplikovanú hru nebolo možné. Preto prvým krokom bolo vytvoriť agenta, ktorý hral len proti jednej ďalšej postave. Tento agent predstavuje bota, čo je postava v hre, správaním pripomínajúca hráča, ale nie je ovládaná hráčom. Na začiatku sa tento agent učil hraním sám so sebou, kde výška odmien za jeho rozhodnutia spočívala napríklad v množstve zlata získaného zabíjaním príšer alebo veľkosti poškodenia spôsobeného nepriateľovi. Agent však nepoznal žiadne pravidlá hry, nevedel čo je jej cieľom a čo všetko mohol robiť. Všetky tieto aspekty sa učil sám podľa veľkosti odmien, ktoré dostával. Po určitom čase bol tento agent postavený voči reálnym

osobám, zo začiatku vývojárom z OpenAI a neskôr aj profesionálnym hráčom, kde sa naučil nové spôsoby hrania. Následne boli vedomosti získané od tohto jedného agenta použité na natrénovanie celého tímu, ktorý sa skladá až z piatich agentov. Takýto typ učenia sa dá kategorizovať ako curriculum learning, kde na začiatku sú akcie agenta a prostredie samotné zjednodušené a postupne sa zvyšuje ich náročnosť pre plynulejšie a efektívnejšie učenie.

4.2. AlphaStar

Rovnako ako Dota 2, hra StarCraft 2 je ďalšia veľmi populárna hra medzi bežnými a profesionálnymi hráčmi. Jej komplexnosť a zložitosť je vysoká oproti iným online hrám. Princíp hry spočíva v piatich kolách, jeden proti jednému, kde si obaja hráči vyberú jednu z troch rás. Každá rasa má svoje špecifické charakteristiky a schopnosti. Hráči začnú s určitým počtom jednotiek, tie získavajú suroviny na výskum a výstavbu nových budov a technológií, vďaka ktorým neskôr môžu získavať viac komplikované schopnosti. Z toho vyplýva, že ak hráč chce vyhrať, musí počas hry sústrediť na dve aspekty - ovládanie samostatných jednotiek a plánovanie svojej ekonómie do budúcnosti.

Aplikovanie strojového učenia na StarCraft 2 na takej úrovni, že je schopné hrať a dokonca aj poraziť profesionálnych hráčov, je základom projektu AlphaStar [1]. Učenie musí byť schopné prekonať viaceré komplikácie vyplývajúce z takejto zložitej hry. Takisto v tomto type hier neexistuje žiadna jediná najlepšia stratégia, ale naopak mnoho rôznych ciest, ktoré môžu viesť k výhre. Jedna hra môže trvať aj hodinu, takže rozhodnutia na začiatku nemusia mať hneď odozvu, či sú správne alebo nie. Hráči majú informácie len o svojich jednotkách a nie súperových. Majú však možnosť ich získať posielaním skautov k nepriateľským oblastiam. Hra je hraná sa v reálnom čase, čiže obaja hráči zároveň robia svoje rozhodnutia a v každom momente existuje veľké množstvo možných akcií, z ktorých si vie hráč vybrať.

AlphaStar je založený na hlbokej neurónovej sieti, ktorá dostáva informácie z herného rozhrania. Učiaci algoritmus bol zo začiatku učený pomocou imitácie zo záznamov hier hraných hráčmi. Následne použili učenie s posilňovaním pre viacerých agentov, ktorí súperili proti sebe. Využili technológiu Cloud TPU od Google, ktorá umožňuje v jednom momente paralelne učenie pomocou tisícky inštancií hrania. Po dvoch týždňoch učenia sa

hraním proti rôznym verziám s rôznymi cieľmi, bol AlphaStar postavený proti najlepším profesionálnym hráčom, kde ich oboch porazil 5-0.

5. Strojové učenie v Unity

Herný rámec Unity [11] ponúka výhodné možnosti aplikovania strojového učenia. ML-Agents Toolkit je open-source nástroj, v ktorom je možné vytvoriť prostredia a agentov na učenie a s úplnou podporou funkcionalít Unity.

5.1. Komponenty

Učenie je použité na natrénovanie neurónovej siete, ktorá sa skladá z viacerých vrstiev. Vstupná vrstva obsahuje informácie o prostredí a vstupoch, výstupná vrstva vplyvy na prostredie pre daný vstup. Vnútorné vrstvy slúžia na výpočty.

Na obrázku 5.1 sú zobrazené komponenty nástroja. Základnými komponentami sú Prostredie učenia, Externý komunikátor a Python API. Externý komunikátor slúži na komunikáciu prostredia učenia s vonkajším pythonovským prostredím, ktoré obsahuje všetky použité algoritmy strojového učenia pomocou TensorFlow knižnice.

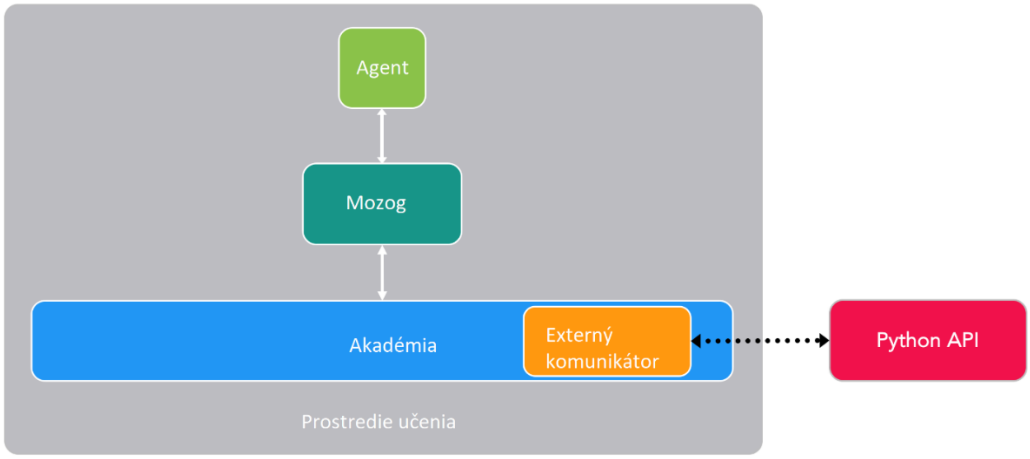
V prostredí učenia sa nachádza Akadémia, Mozgy a Agenti. Akadémia predstavuje skript, ktorý mení parametre prostredia a podáva tieto informácie mozgom.

Agent môže byť akýkoľvek herný objekt alebo postava, ktorá generuje pozorovania, vykonáva akcie a dostáva odmeny. Každý agent je napojený na práve jeden mozog, ktorý obsahuje logiku správania sa pre daného agenta. Na základe pozorovaní a odmien, ktoré pošle agent mozgu, mozog vygeneruje príslušnú akciu a tú pošle agentovi, ktorý ju vykoná. Sú tri typy mozgov- learning, player a heuristic.

Learning (učiaci) mozog spracúva informácie použitím TensorFlow modelu, ktorý predstavuje optimálnu formu (policy) a generuje akcie pre agenta.

Player (hráč) mozog je priame ovládanie agenta reálnou osobou v reálnom čase.

Heuristic (heuristický) mozog je natvrdo naprogramované správanie sa agenta. Všetky tieto mozgy môžu komunikovať s externým Python prostredím, vďaka čomu je na výber viacero možných typov učenia agentov.



Obrázok 5.1 - Prostredie učenia [7]

5.2. Typy učenia

Všetky mozgy môžu komunikovať s externým Python prostredím, vďaka čomu je na výber viacero možných typov učenia agentov. Výsledkom každého učenia je **TensorFlow model**, ktorý obsahuje všetky naučené dáta.

5.2.1. Klasické učenie s posilňovaním

Pri aplikácií klasického učenia s posilňovaním je prvým krokom definovanie troch entít, ktoré tvoria základ strojového učenia s posilňovaním - pozorovania, akcie a odmeny.

Pozorovania predstavujú všetky informácie, ktoré má agent o danom prostredí. Tieto informácie nepredstavujú celkový stav prostredia, ale iba podskupinu vlastností, ktoré sú dostupné pre agenta. Môžu byť číselné alebo vizuálne.

Akcie sú všetky akcie, ktoré môže agent vykonať v danom prostredí. Môžu byť diskrétne, napríklad pohyb do jedného zo štyroch smerov alebo kontinuálne, kde má na výber napríklad aj rýchlosť.

Odmeny- číselná hodnota vyjadrujúca či daná akcia bola pozitívna alebo negatívna.

Nástroj ponúka aj možnosť zobrazenia rozhodovacieho procesu agenta v reálnom čase, vďaka čomu bude výrazne uľahčené optimalizovanie systému odmien. Takisto sa dá aj postupne zvyšovať komplexnosť testovacieho prostredia, kde na začiatku sa bude agent trénovať na jednoduchých verziách.

5.2.2. Imitačné učenie

Druhým veľmi efektívnym spôsobom učenia agenta v nástroji je pomocou imitačného učenia. Jeho princíp spočíva v tom, že niekedy je jednoduchšie znázorniť sekvenciu požadovaných akcií a ideálnych stavov, ako len zoširoka opísať cieľ a nechať agenta skúšať a hľadať cestu k nemu.

Toto učenie sa dá implementovať ako offline aj online učenie. Pri offline učení sa nahrá skupina ilustračných záberov, stavov alebo akcií, ktoré chceme aby dokázal. Táto nahrávka sa dodá mozgu a ten spustí trénovanie. Ak chceme učiť agenta online, musí učebné prostredie obsahovať dvoch agentov- jeden napojený na player mozog a druhý na learning. Oba mozgy musia mať definované kompatibilné akcie a parametre, aby bolo schopné imitovať naše akcie agentom. Takto v reálnom čase vidíme ako sa agent učí presne podľa našich krokov a keď sme spokojný s jeho správaním, vieme uložiť stav naučeného agenta.

5.3. TensorBoard grafy

Pri učení pomocou TensorFlow knižnice sa dá použiť vizualizačný nástroj zvaný Tensorboard. Vďaka tomuto nástroju sa dajú zobraziť parametre a štatistiky pre učenie v danom čase. Z týchto vygenerovaných grafov sa dá potom určiť, či zvolené hyperparametre alebo aj odmeny vedú k dobrému učeniu, alebo rozhodovanie agenta je len čisto náhodné. Všetky grafy zobrazujú zmenu v čase, reprezentovaného počtom krokov a sú rozdelené do troch kategórií – Environment, Loss a Policy.

Enviroment grafy hovoria o priemernej výške odmien a dĺžke epizódy. Dobré učenie má mať rastúci graf pre odmeny a klesajúci graf pre dĺžku epizódy.

Policy grafy hovoria o naučenom modeli a spôsobe rozhodovania. Graf o entropii ukazuje náhodnosť rozhodnutí, ktorá má po čase klesať.

Príklady grafov budú zobrazené v kapitole 10, pojednávajúcej o výsledkoch testovania naučených modelov.

6. Virtuálne prostredie

Testovanie softvéru je veľmi dôležitá časť jeho životného cyklu. Môže prebiehať na reálnom zariadení v reálnom svete, čo znamená vytvorenie nového prototypu po neúspechu predošlého testovania. Preto sa častejšie začína využívať testovanie vo virtuálnom prostredí.

Podľa publikácie [30] je testovanie vo virtuálnom prostredí už dlho využívane pri bezpilotných vozidlách. Práve v tomto odvetví je testovanie náchylné na množstvo chýb, ktoré môžu viesť k úplnému zničeniu vozidla. Tieto vozidlá sa budú nachádzať aj v blízkosti osôb, na ktorých pohyby budú musieť vedieť v adekvátnom čase zareagovať.

Testovanie automobilov môže prebiehať v rôznych fázach vývoja a existuje nato viacero nástrojov, napríklad pomocou rozšírenej platformy dSpace [4].

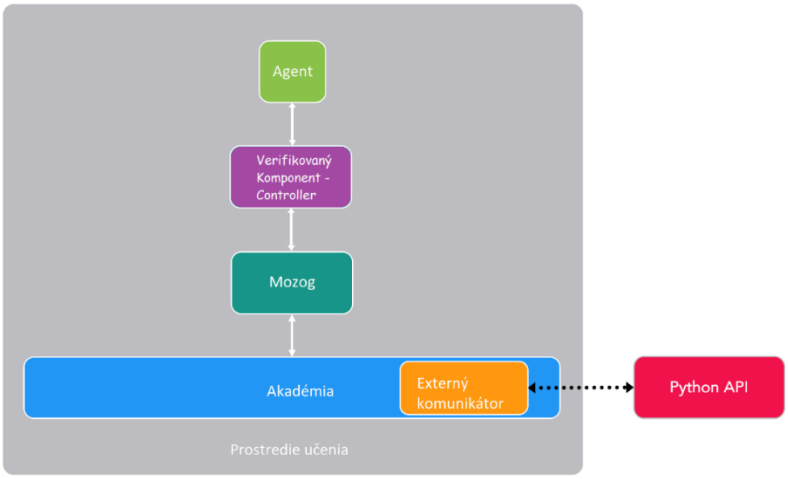
Vývoj herného prostredia pri vývoji hier je náročná činnosť, ktorá si vyžaduje veľké množstvo úsilia. Nejde iba o vymodelovanie hernej mapy a vloženia zopár objektov, medzi ktorými budú hráči pobehovať. Od vývojárov je očakávané prostredie, ktoré bude veľmi podobné až nerozoznateľné od nášho skutočného sveta. To neznamená len po grafickej stránke, kde to dnes môže byť už aj fotorealistické, ale aj podľa správania a pravidiel daného prostredia. Takým pravidlom môže byť fyzika v prostredí, pohyb hráčov a postáv, takisto ich interakcia s predmetmi v prostredí. Všetky tieto aspekty by mali čo najviac zodpovedať realite. Každé herné štúdio si vytvára svoje vlastné herné prostredie a často sa nové vytvára aj pre iné hry od rovnakého herného štúdia. Z toho sa dá usúdiť, že vývoj takého prostredia bude výrazne náročný a zaberie veľa času aj financií, ktoré nie sú obmedzené. Ak by však bolo toto herné prostredie využité aj na niečo iné, ako len hranie, mohlo by to vyriešiť problémy vývojárov. Ak by niektorá firma vytvárala softvér, ktorý je potrebné testovať aj vo virtuálnom aj reálnom prostredí, vedeli by sa spojiť s herným štúdiom a využiť ich vytvorené prostredie práve na testovanie. Hráči by si nemuseli ani uvedomovať že hra, ktorú hrajú, je využitá na takýto účel.

Virtuálne prostredie, použitie existujúcich hier alebo herných rámcov na vytvorenie prostredia je detailne opísane v bakalárskej práci [29], venovanej interaktívnemu virtuálnemu prostrediu.

7. Návrh riešenia

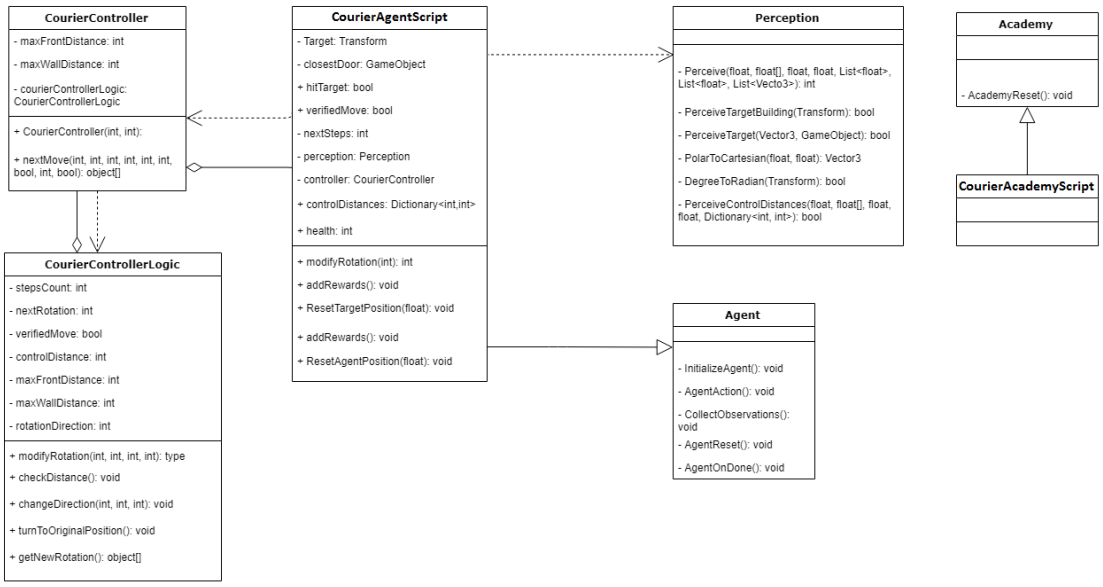
V hernom rámci Unity bude vytvorené herné prostredie, ktoré bude hrateľné viacerými hráčmi. Toto herné prostredie je vytvorené a opísané v bakalárskej práci [29] kde bol kladený dôraz na výber najlepšieho vývojového prostredia pre náš projekt. Server bude spúšťaný na našom počítači, na ktorý sa následne pripoja ostatní hráči. Na tomto serveri bude spustený buď už natrénovaný, alebo stále učiaci sa agent pomocou algoritmov strojového učenia. Agent bude vytvorený pomocou Unity nástroja pre strojové učenie, ktorého výhody boli detailne opísané v predošlej kapitole. Tento nástroj používa strojové učenie s posilňovaním s algoritmom PPO, kde agent v hre bude robiť rozhodnutia na základe odmien, ktoré dostáva po vykonaní danej akcie a stavu, v ktorom sa nachádza svet aj on samotný. Hra bude vo forme strielačky pre štyroch až ôsmych hráčov a úlohou agenta bude bezpečne sa dostať z jednej náhodne vybratej pozície mapy do druhej, rovnako náhodnej. Základné odmeny budú na základe času, za ktorý splnil túto úlohu a vzdialenosti priblíženia sa k cieľu. Pridané môžu byť aj rôzne ďalšie odmeny, ako napríklad či ho vidia hráči, výšky zranenia, ktoré utrpel, vzdialenosti od objektov v hre a mnoho iných.

Veľmi dôležitým aspektom správania sa agenta bude vzdialenosť priblíženia sa k prekážkam. Tieto prekážky môžu predstavovať rôzne objekty vo svete ako budovy, barikády, stromy alebo aj hráči. Funkcia robota nemusí byť iba aplikovanie v bojovej zóne, ale napríklad na normálnej mestskej ulici s viacerými chodcami a autami. V takom prípade by nízka vzdialenosť medzi agentom a objektami v prostredí znamenala potencionálne ublíženie na zdraví osôb alebo poškodenia iných objektov a samého seba. Táto situácia bude riešenia použitým formálnych metód, kde bude vytvorený verifikovaný komponent. Na obrázku 7.1 je zobrazené, ako bude fungovať komunikácia medzi všetkými komponentmi systému. V momente, kedy sa agent ocitne v nebezpečnej situácií, napríklad spomínaná nízka vzdialenosť, sa prestane rozhodovať na základe strojového učenia a odmien, ale prevezme nad ním kontrolu verifikovaný komponent vytvorený formálnymi metódami.



Obrázok 7.1 – Komunikácia komponentov v prostredí učenia

Na obrázku 7.2 vidíme diagram tried, kde hlavnou triedou je **CourierAgentScript**, ktorá dedí od abstraktnej triedy Agent a využíva dve ďalšie triedy. Prvou z nich je **Perception**, ktorej funkcie slúžia na posielanie lúčov od agenta na zisťovanie jeho vzdialenosti od objektov. Druhou triedou je **CourierController**, ktorá predstavuje vygenerovaný C# kód z implementácie abstraktného stroja B-metódy. CourierController volá funkcie **CourierControllerLogic**, ktorá bola takisto vygenerovaná. Poslednou triedou je **CourierAcademyScript**, dediaca od Academy. Jej úlohou je inicializovanie herného prostredia za účelom učenia alebo použitia natrénovaného modelu.



Obrázok 7.2 – Diagram tried

8. Riešenie strojového učenia

Na vytvorenie autonómnej entity agenta bolo použité strojové učenie riešené pomocou ML-Agents Toolkit pre Unity. Tento nástroj podporuje viaceré spôsoby učenia, ktoré boli vysvetlené v kapitole 5. Autonómna entita agenta bola natrénovaná pomocou strojového učenia s posilňovaním, využívajúc PPO (Proximal Policy Optimization) algoritmus.

8.1. Herné objekty

Do herného prostredia v Unity boli vložené objekty CourierAgent, CourierAcademy a CourierTarget. CourierAgent má priradený mozog na strojové učenie s posilňovaním nazvaný CourierBrain.

8.1.1. CourierBrain

Predstavuje mozog, ktorý sa bude učiť a vytvárať TensorFlow model. V mozgu sa nastavuje počet pozorovaní agenta a počet akcií, ktoré bude schopný vykonať. Ak má mať agent pozorovania z kamier, treba takisto určiť ich počet a rozlíšenie na spracovanie obrazu. Ak už existuje nejaký natrénovaný model pre daný mozog a nevyžaduje sa ďalšie učenie, treba ho vložiť v Unity editore do mozgu ako model.

Počet pozorovaní agenta je 125, neobsahuje žiadne vizuálne pozorovania z kamier a má jednu skupinu diskrétnych akcií o veľkosti 9. Význam a priradenie jednotlivých pozorovaní a akcií bude vysvetlený v časti 8.2.

8.1.2. CourierAcademy

CourierAcademy je prázdny objekt predstavujúci akadémiu. Herné prostredie je v celku statické, kde objekty, až na zopár výnimiek, sa nijak nemenia ani dynamicky nepridávajú za behu do prostredia. Z toho dôvodu akadémie nijak neovplyvňuje učenie agenta a jej zdrojový kód je prázdny. Avšak jej parametre menia nastavenia prostredia.

- **Width** – Šírka obrazu hry
- **Height** – Výška obrazu hry
- **Quality** – Kvalita grafiky hry
- **Time Scale** – Týmto parametrom nastavíme zrýchlenie hernej slučky. Ak nie je dôležité pozorovať agenta a hru v reálnom čase, ale chceme čo najviac urýchliť

proces učenia, nastavíme vyššiu hodnotu. Keď budú hrať hru aj skutoční hráči, nastaví sa hodnota na 1.

- **Target Frame Rate** – Počet snímok za sekundu, ktoré sa má hra snažiť udržiavať.
- **Max Step** – Rovnako ako u CourierAgent, tento parameter určuje maximálny počet krokov, po ktorého dosiahnutí sa resetuje akadémia.

Tieto parametre sa dajú nastaviť zvlášť pre fázu učenia a pre fázu po učení, nazývanú Inference. Takisto sa môžu parametre meniť bez ohľadu na nastavenia natrénovaného modelu.

Pre autonómnu entitu je len jeden mozog, ktorý je priradený do Broadcast Hub, kde sa definujú všetky mozgy, ktoré sa nachádzajú v prostredí a majú komunikovať s agentom. Ak už je model natrénovaný a ďalšie učenie sa nevyžaduje, je potrebné buď odobrať mozog z akadémie, alebo vypnúť políčko Control. V takom prípade musí však mozog obsahovať niektorý natrénovaný model.

8.1.3. CourierTarget

Tento objekt je jednoduchá kocka, ktorá predstavuje cieľovú pozíciu agenta. Jej začiatočná poloha je v blízkosti agenta a keď sa agent dostane ku nej v dostatočnej blízkosti, zmení sa pozícia v určitej náhodnej vzdialenosti od agenta. Táto vzdialenosť sa postupne zvyšuje, aby zo začiatku agent vedel čo je jeho cieľom a ako sa k nemu dostať.

8.1.4. CourierAgent

CourierAgent je objekt v scéne predstavujúci autonómnu entitu robotického kuriéra, ďalej ako agent. Tento agent má priradené rovnaké komponenty ako ostatné objekty v hernom prostredí, ako napríklad RigidBody, ktorý mu pridáva simuláciu gravitácie a Collider, vďaka ktorému nebude môcť prechádzať cez iné objekty. Dôležitým komponentom je CourierBrain, ktorý predstavuje učiaci sa mozog pomocou posilňovania s učením, ktorého učenie ovplyvňuje niekoľko parametrov.

- **Max step** – Určuje maximálny počet krokov agenta v jednej epizóde učenia. Ak sa dosiahne táto hodnota, všetky nadobudnuté odmeny a pozorovania sú poslané do TensorFlow knižnice a spracované na upravenie naučeného modelu. Odporúčaná hodnota je pod 1000, aby sa urýchlilo učenie agenta. Je dôležité, aby agent

dokázal za tento počet krokov získať dostatočné množstvo užitočných pozorovaní aj odmien. Nastavená hodnota je 600.

- **Reset on Done** – Ak je tento parameter zaškrtnutý, po dosiahnutí Max Step alebo po dokončení úlohy agenta je zavolaná funkcia AgentReset(), kde sa nastaví agent pre novú epizódu učenia sa. Parameter je zaškrtnutý, avšak funkcia je prázdna, keďže cieľ môže byť vybraný v takej vzdialenosti od agenta, že by ho nemusel stihnúť nájsť za daný počet krokov. Namiesto toho máme definované vlastné funkcie na resetovanie agenta a pripravenia ho na novú epizódu. Tieto funkcie sa líšia spôsobom, akým nastal koniec epizódy – či agent dosiahol cieľ, či dosiahol maximálny povolený počet krokov alebo dosiahnutie nami stanovených podmienok. Jednou z týchto podmienok je maximálne vzdialenie agenta od jeho cieľa.
- **On Demand Decision** – Týmto parametrom určujeme, či si má agent pýtať akcie po určitom počte krokov alebo v nami definovaných momentoch. Tento parameter je nezaškrtnutý, čiže agent si pýta akcie podľa nasledujúceho parametra.
- **Decision Frequency** – Určí po koľkých krokoch si má agent pýtať nové akcie od modelu. Pri vyššej hodnote bude agent vykonávať rovnaké kroky, aj keď už má nové informácie o prostredí, ktoré môžu byť pre neho kritické. Preto je tento parameter nastavený na hodnotu 5.
- **Agent Cameras** – Agentovi môžu byť priradené kamery, ktoré mu budú zobrazovať prostredie z prvej osoby, alebo napríklad aj zhora. Tieto obrazy z kamier budú následne spracované a použité pri učení. Momentálne nemá agent priradenú žiadnu kameru, keďže spracovanie obrazov je veľmi náročné na výkon, spomaľujú učenie a v našom prípade neposkytovali dostatočne vysoké zlepšenie učenia sa.

Až na Agent Cameras a On Demand Decision, môžeme tieto parametre meniť aj ak už máme nejaký natrénovaný model a pokračujeme v jeho učení.

8.2. Odmeny, pozorovania a akcie

V 8.1.1. bolo povedané, že mozog obsahuje určitý počet pozorovaní a akcií. Tieto dva parametre spolu s odmenami sú základ strojového učenia s posilňovaním.

8.2.1. Akcie

CourierBrain má definovanú jednu skupinu diskrétnych akcií o veľkosti 9. To znamená, že v momente, keď si agent pýta akciu od mozgu, ako odpoveď dostane jednu hodnotu z rozsahu 0 – 8. Hodnota 0 určí, že agent stojí na mieste. Podľa hodnôt 1 - 8 je priradený jeden z ôsmych smerov, do ktorého sa bude agent hýbať. To sa robí nastavením rotácie agenta, kde smery predstavujú násobky 45 stupňov v rozsahu 0-360.

Agentovi sa dajú pridať viaceré skupiny akcií, napríklad druhá skupina by určovala jeho rýchlosť – či má spomaliť, zrýchliť alebo ísť rovnakou rýchlosťou. Prvé testovania ukázali, že pre nášho agenta to nie je dôležitá akcia, keďže stále bolo výhodnejšie ísť čo najrýchlejšie. Existencia tejto skupiny akcie potom len spomaľovala učenie, pretože agent skúšal rôzne kombinácie akcií, aby zistil ktorá je tá najlepšia. Preto bola vybraná konštantná rýchlosť agenta a ponechaná iba jedna skupina diskrétnych akcií.

8.2.2. Pozorovania

Pozorovania sú vektory, ktoré agent vníma o svete, objektoch vo svete alebo sám o sebe. Tieto pozorovania sú spolu s odmenami spracované TensorFlow modelom a ten následne upravuje svoju naučenú formu. Boli skúšané a testované rôzne počty, kombinácie a typy pozorovaní pre agenta. Základné vektory boli získané posielaním lúčov, v Unity nazývaných Raycast, od stredu agenta do viacerých uhlov. Vďaka týmto lúčom zisťujeme v akej vzdialenosti od agenta sú objekty, ako napríklad budovy, stromy, kamene, hráči, barikády, iní agenti a autá. Na začiatku boli lúče posielané do uhlov, ktoré predstavovali zorné pole agenta. Následne boli zmenené tak, že idú od neho do každého smeru. Okrem toho je dôležitý aj rozptyl uhlov pre lúče, keďže čím ďalej sú od agenta, tým vznikajú väčšie medzery medzi nimi, kde agent nezachytí nejaký objekt. Vybraný rozptyl lúčov je každých päť stupňov v rozsahu 0 až 360. Okrem vzdialeností objektov od agenta si môže agent pamätať ich poslednú polohu, konkrétny počet objektov v nebezpečnej vzdialenosti a napríklad aj typy objektov. Najideálnejšie riešenie bolo získavanie len vzdialenosti objektov od agenta. Pri lepších hardvérových podmienkach mohli byť tieto pozorovania lúčmi nahradené alebo doplnené o pozorovania z obrazov zachytených kamerami.

Momentálne mozog obsahuje 125 pozorovaní. Medzi tieto pozorovania patrí pozícia agenta a cieľa, vzdialenosť a výskyt nejakej prekážky medzi nimi, rozdiel medzi ich X súradnicou, rozdiel medzi ich Y súradnicou, otočenie agenta, jeho život, zmena vo

vzdialenosti agenta a jeho cieľa od poslednej pýtanej akcie a celkové oddialenie agenta od cieľa v danej epizóde. Zvyšných pozorovania sú získané posielaním lúčov, kde sa zisťujú vzdialenosti objektov od agenta, vzdialenosti a pozície hráčov.

Výhodné bolo normalizovať určité hodnoty, aby každé pozorovanie malo rovnakú váhu pri rozhodovaní.

8.2.3. Odmeny

V každom momente, keď si agent pýta akciu, dostáva aj odmeny za svoj momentálny stav. Funkcia odmien je veľmi dôležitým aspektom trénovania, kde je potrebné vybrať správne hodnoty a situácie, za ktoré odmeniť agenta.

Podľa testovania agentovho správania a analyzovania výsledkov sa tieto odmeny menili, kde veľa vyskúšaných bolo úplne odobratých alebo boli výrazne modifikované. Najdôležitejším zistením bolo, že agent sa lepšie učí z kladných odmien, ako zo záporných. Preto bolo potrebné nájsť správnu distribúciu odmien, aby sa agent len nevyhýbal každej akcií, pretože viedla k záporným odmenám a kvôli tomu sa nikdy nedostal k tým pozitívnym.

- Za čas riešenia úlohy, čiže stále keď si pýta akciu, dostane -0,02.
- Posun od poslednej akcie. Ak sa priblížil, odmena je kladná, ak sa vzdialil, odmena je záporná.
- Ak je už blízko cieľa, odmena za posun sa zdvojnásobí.
- Odmena za jeho život.
- Nízka negatívna odmena za každého hráča, ktorý je v malej vzdialenosti od agenta a vyššia pozitívna odmena za neprítomnosť hráčov v okolí agenta.
- Podobné odmeny aj za ostatné objekty vo svete v blízkosti agenta.
- Ak sa vzdialil o maximálnu vzdialenosť od cieľa alebo vykonal maximálny počet povolených krokov, odmena -5 a ukončenie epizódy.
- Dosiahnutie cieľa, odmena sa nastaví na 50f a ukončí sa epizóda.

Ukončenie epizódy predstavuje spracovanie odmien, akcií a pozorovaní a aktualizáciu modelu. Poloha cieľa sa mení len pri úspešnom ukončení epizódy. Pri ukončení epizódy kvôli dosiahnutiu Max Step, sa len aktualizuje model učenia. Pri ukončení epizódy kvôli dosiahnutiu maximálnej povolenej vzdialenosti od cieľa, sa ukončí epizóda a agentova

poloha sa nastaví na polohu, akú mal na začiatku epizódy. To je veľmi výhodné na začiatku učenia, keďže herný priestor je veľmi rozsiahly a agent by mohol ísť celý čas opačným smerom a nikdy by sa nedostal k cieľu a tým pádom by sa nikdy nenaučil, že za jeho dosiahnutie je najväčšia odmena.

8.3. Hyperparametre

Ako pri každom type strojového učenia, nastavovanie a kombinácie hyperparametrov je základným krokom učenia. Ovplyvňujú kvalitu, rýchlosť a hlavne aj úspešnosť učenia. Tieto hyperparametre sú definované v konfiguračnom súbore trainer_config.yaml, z ktorého číta TensorFlow knižnica pri zapnutí učenia. Ak sa pokračuje učenie z nejakého natrénovaného modelu, väčšina parametrov sa už nedá dodatočne meniť. Preto si je potrebné poriadne premyslieť, ako ich nastaviť, pretože učenie môže trvať dlhšiu dobu. Natrénovanie modelu pre agenta s pozitívnymi výsledkami zabralo štyri hodiny učenia. Pri zložitejších systémoch môže trvať aj niekoľko dní, aby bolo vidno nejaký pokrok.

PPO algoritmus použitý v ML-Agents Toolkit obsahuje trinásť povinných a dve dvojice doplňujúcich hyperparametrov. V tejto kapitole je opísané, aký je vplyv každého hyperparametra na proces učenia a aká hodnota je nastavená pre agenta. Podrobný popis sa nachádza v dokumentácií [11].

Gamma – ovplyvňuje očakávanie odmien agenta v budúcnosti. Ak jeho akcia priamo vedie k odmene, odporúčajú sa nižšie hodnoty. Náš agent však musí myslieť viac do budúcnosti, keďže približovanie sa k cieľu priamou cestou nemusí viesť k najvyššej odmene.

Nastavená hodnota je 0.98.

Lambda – určuje ako veľmi agent dôveruje svojej funkcie očakávanej odmeny. Keď je nastavená vyššia hodnota, agent sa viac spolieha na získané reálne odmeny z prostredia, ako na svoju naučenú funkciu. 0.95

Buffer Size – koľko informácií, tvorené počtom pozorovaní, akcií a odmien, má agent nazbierať pred tým, ako aktualizuje svoj naučený model.

Nastavená hodnota je 10 240.

Batch Size – počet informácií, ktoré budú použité v jednej iterácii pri výpočtoch poklesu gradientu (gradient descent) [27] algoritmu PPO. Pod poklesom gradientu si môžeme

predstaviť optimalizačnú funkciu. Tento hyperparameter by mal byť stále deliteľom buffer size. Hodnota sa upravuje podľa typu priestoru akcií, kde pre diskrétne akcie sa odporúča nižšia hodnota a pre kontinuálne akcie naopak vyššia.

Náš agent dostáva diskrétne akcie, z toho dôvodu je hodnota nastavená na 256.

Number of Epochs – počet prechodov uloženými informácií pri poklese gradientu. Zvýšenie vedie k urýchlenému učeniu, avšak ak je nízky batch size, nemusí to nijak pomôcť. Nastavená hodnota: 4.

Learning Rate – aký je vplyv každého poklesu gradientu. Nastavená hodnota: 3e -4.

Time Horizon – koľko informácií sa má zozbierať pred ich uložením. Ak je hodnota príliš malá, nemusí zachytiť všetky dôležité body v agentovom správaní. Preto ak počas jednej epizódy učenia dostáva agent nízke množstvo odmien, je lepšie nastaviť vyššiu hodnotu. Ak sa dosiahne jej hodnota pred ukončením epizódy, použite sa funkcia očakávanej odmeny na výpočet celkovej odmeny z agentovho súčasného stavu. Použitá hodnota je 512.

Max Steps – počet krokov počas učenia. Čím je komplexnejší problém, tým je potrebná vyššia hodnota, aby sa agent dokázal naučiť. Po dosiahnutí hodnoty sa ukončí učenie, ale ak výsledky nie sú dostatočne uspokojujúce, dá sa hodnota zvýšiť a ďalej pokračovať v učení. Agentovo správanie viedlo k pozitívnym výsledkom po dosiahnutí 100 000 krokov učenia, preto je nastavená hodnota 1e6.

Beta – parametrom Beta ovplyvníme entropiu, čiže náhodnosť vybratých akcií agenta. To je veľmi výhodné, keďže aby agent vedel vyberať najlepšie akcie, musí najprv tieto akcie vôbec vyskúšať. So zvyšujúcimi sa získanými odmenami bude entropia klesať. To sa dosiahne pri dobrej kombinácii hyperparametrov a funkcie odmien. Nastavená hodnota je 3e-3.

Epsilon – Týmto parametrom určíme maximálnu odchýlku medzi starým a novým modelom. Nastavená hodnota je 0.2.

Normalize – určuje, či získané vektory pozorovania od agenta budú normalizované. Nastavená hodnota: false.

Number of Layers – definuje, koľko skrytých vrstiev sa nachádza v neurónovej sieti pri spracovaní pozorovaní. Nastavená hodnota je 2.

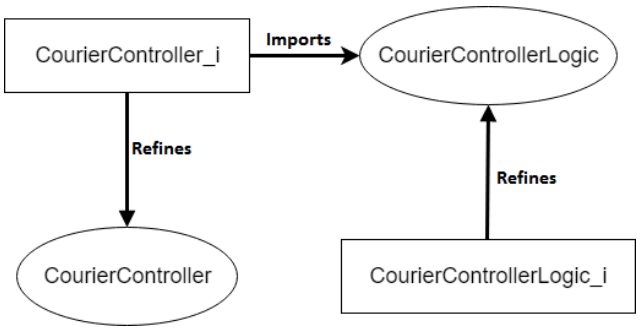
Hidden Units – určuje počet jednotiek v jednej vrstve neurónovej siete. Hodnota sa ma odvíjať od komplexnosti problému, kedy sa odporúča vysoká hodnota. Nastavená hodnota je 128.

Okrem týchto povinných hyperparametrov sa dajú nastaviť ďalšie dva – **Use Curiosity** a **Use Recurrent**. Ak ich hodnota je true, obe majú ďalšie dva parametre ktoré ich ďalej upravujú. Use Curiosity určuje, či má agent skúšať čo najviac rôznych kombinácií akcií na začiatku učenia, bez ohľadu na získanú odmenu. Povolením Use Recurrent si bude agent pamätať určite množstvo predošlých akcií, stavov a odmien a tie budú použité pri nasledujúcom rozhodovaní. To je výhodné použiť vtedy, keď agent musí aplikovať nejaký poznatok, ktorý zistí až počas svojho fungovania. V tomto prípade agent nemá žiadne takéto poznatky, preto je používanie Recurrent pamäte vypnuté.

9. Verifikovaný komponent pomocou formálnych metód

Na vývoj formálne verifikovanej časti softvéru bola použitá B-Metóda pomocou nástroja AtelierB [26]. Inšpiráciou pre tento komponent bol riadiaci program čistiaceho robota z článku [28]. Komponent kontroluje stav agenta a je volaný pri každej požiadavke agenta o novú akciu. Ak sa zistí platnosť podmienok, ktoré hovoria o nebezpečnej situácii pre agenta, agent sa rozhoduje na základe informácií obdržaných od tohto verifikovaného komponentu a nie od naučeného modelu strojového učenia.

Na obrázku 9.1. sú zobrazené základné komponenty tohto riešenia. Oválmi sú znázornené abstraktné stroje a obdĺžnikmi ich implementácie.



Obrázok 9.1 – Diagram komponentov verifikovanej časti softvéru

9.1. Abstraktné stroje

Prvým krokom bolo vytvorenie abstraktných strojov, ktoré sa dajú porovnať s triedami v objektovo orientovanom programovaní. Operácie v jednom abstraktom stroji sú od seba nezávislé a všetky premenné sú dostupné a modifikovateľné iba operáciami daného stroja. Preto boli vytvorené dva abstraktné stroje, CourierController a CourierControllerLogic, ktoré tvoria tento komponent.

9.1.1. CourierControllerLogic

Abstraktný stroj CourierControllerLogic predstavuje základnú logiku, na základe ktorej sa bude upravovať agentovo správanie. Dostáva dva parametre:

`maxFrontDistance` – maximálna povolená vzdialenosť agenta od steny, ku ktorej je otočený.

`maxWallDistance` – maximálna povolená vzdialenosť agenta od steny, popri ktorej bude prechádzať.

Tieto parametre sú ošetrené v klauzule CONSTRAINTS. V klauzule SETS je definovaná množina DIRECTIONS. Prvky tejto množiny určujú smer, do ktorého sa agent otočil kvôli prekážke. Členské premenné sa nachádzajú v klauzule CONCRETE_VARIABLES a ich typy a obmedzenia sú v klauzule INVARIANT. Tieto premenné sú:

stepsCount – počet krokov, ktoré agent ešte vykoná po prekonaní prekážky.

nextRotation – nová hodnota rotácie pre agenta.

rotationDirection – typu DIRECTIONS, smer otočenia sa agenta.

verifiedMove – určuje, či verifikovaný komponent prevzal kontrolu nad agentom.

controlDistance – kontrolná vzdialenosť prekážky od agenta.

Inicializácia premenných je v klauzule INITIALISATION.

Poslednou klauzulou je OPERATIONS, kde sa nachádzajú definície operácií stroja:

- **modifyRotation** – dostáva 4 vstupné parametre, kde prvý určuje súčasnú rotáciu agenta a zvyšné vzdialenosti objektov pred ním . Ak je rotácia v uhle inom ako násobok 45 a zároveň predná vzdialenosť menšia ako maxFrontDistance, natočí sa o 45° podľa hodnôt predných bočných vzdialeností.
- **changeDirection** – dostáva 3 vstupné parametre predstavujúce vzdialenosti prekážok vpred, naľavo a napravo od agenta. Výsledkom tejto operácie je nová rotácia agenta v newRotation a nová hodnota controlDistance podľa jeho smeru natočenia v rotationDirection.
- **checkDistance** – kontroluje, či kontrolná vzdialenosť prekročila maximálnu povolenú a podľa toho nastaví verifiedMove a nextStepsCount.
- **turnToOriginalPosition** – natočenie agenta do pôvodnej polohy po ukončení jeho ovládania verifikovaným komponentom.
- **getNewRotation** – operácia na získanie hodnôt premenných.

9.1.2. CourierController

Tento abstraktný stroj predstavuje triedu, s ktorou komunikuje hlavná trieda agenta. Prijíma rovnaké parametre ako predošlý abstraktný stroj, ktorý aj bude následne vytvárať. Obsahuje definíciu jednej funkcie **nextMove**, ktorá vracia kolekciu štyroch objektov.

Prijíma 9 parametrov, kde prvý parameter je aktuálna rotácia agenta. Ďalších päť parametrov sú vzdialenosti objektov od agenta. Zvyšné tri spolu s parametrom `frontDistance` sú dôležité v hlavnej podmienke stroja, ktorá určuje či má komponent prevziať kontrolu nad agentom.

9.2. Implementácia abstraktných strojov

Po vytvorení abstraktných strojov je ďalším krokom ich implementácia. Implementácia sa líši od špecifikácie abstraktných strojov v zopár aspektoch. Prvým rozdielom je možnosť sekvenčného vykonávania príkazov definovaného (;) namiesto paralelného (||). Druhým je zámena klauzuly PRE v operáciách za podmienený príkaz IF. Posledným a najdôležitejším rozdielom je konkrétne priradenie hodnôt. V implementácií abstraktného stroja CourierController sa nachádza referencia na stroj CourierControllerLogic v klauzule `IMPORTS`. Volané sú jeho operácie a ich návratová hodnota je následne poslaná do hlavnej triedy agenta CourierAgent.

Zo zdrojových kódov implementácií strojov je následne možné generovanie kódu do iného programovacieho jazyka.

9.3. Verifikácia

Tieto komponenty boli verifikované nástrojom pomocou metódy Automatic Proof (Force 1), kde sa kontroluje, či boli zachované podmienky definované v klauzulách `INVARIANT`, `CONSTRAINTS` a `INITIALIZATION`. Na obrázku 9.2 vidíme výsledky verifikácie každého komponentu. Stĺpec Proof Obligations uvádza počet povinných dôkazov. Stĺpce Proved a Unproved udávajú, koľko dôkazov sa podarilo a nepodarilo dokázať. Pre tento komponent boli všetky povinné dôkazy úspešne dokázané. Spôsob verifikácie bol nastavený na Legacy.

Component	TypeChecked	POs Generated	Proof Obligations	Proved	Unproved
CourierController	OK	OK	0	0	0
CourierControllerLogic	OK	OK	49	49	0
CourierControllerLogic_i	OK	OK	68	68	0
CourierController_i	OK	OK	4227	4227	0

Obrázok 9.2 – Dôkazy v nástroji AtelierB

9.4. Aplikácia do autonómnej entity

Z abstraktného stroja bola vytvorená jeho implementácia a z implementácie bol pomocou nástroja BKPI [24] vygenerovaný zdrojový kód v jazyku C#. Vygenerovaný zdrojový kód bol vložený do Unity projektu medzi ostatné zdrojové kódy. Hlavná trieda agenta následne vytvorí inštanciu triedy `CourierController` a volá nad ním funkciu `nextMove`, ktorá mu vráti kolekciu objektov. Podľa hodnôt týchto objektov sa následne určí, či komponent prevezme kontrolu nad agentom. Ak neprevzal, agent pokračuje s akciami obdržanými od natrénovaného modelu strojového učenia.

Nástroj BKPI je stiahnuteľný zo zdroja [25].

10. Testovanie implementácie

V tejto kapitole sú vysvetlené a opísané pozorované výsledky po implementácií a testovaní natrénovaného modelu pre agenta vo virtuálnom prostredí. Najprv je opísaný priebeh a podmienky procesu učenia

10.1. Podmienky pre proces učenia

Učenie prebiehalo za rovnakých podmienok, ako bolo spomínané v kapitole 8. Cieľ sa generoval v pomerne malej vzdialenosti od agenta.

Do prostredia bolo vložených 5 agentov, každý v inej časti prostredia a s rôznym okolím. Každý z agentov mal svoj vlastný cieľ, nezávislý od ostatných. Všetci agenti boli pripojený k rovnakému mozgu, teda ich akcie, stavy a pozorovania boli použité na aktualizovanie natrénovaného modelu spoločne. To bolo výhodne z dvoch strán. Prvou je výrazné urýchlenie učenia, keďže existencia piatich agentov oproti jednému nepredstavuje viditeľne zvýšené hardvérové nároky. Čiže v danom momente dostával mozog päť skupín informácií skladajúcich sa z akcií, stavov a pozorovaní. S týmto súvisí druhá výhoda. Agenti sa nachádzali na rôznych pozíciách v prostredí. Tým pádom mozog dostával viaceré rôzne informácie, ktoré by nemusel dostať len pri použití jedného agenta. Napríklad jeden agent sa nachádzal na rovinke bez prekážok, druhý na rovinke s prekážkami a tretí sa nachádzal priamo v budove. Vďaka tomu malo učenie tieto rôzne podnety dostupné od samého začiatku. Aby pri použití jedného agenta došlo k dodaniu rovnakých podnetov, musel by sa on sám presunúť v prostredí na dané lokality, čo však nemuselo nastať vôbec alebo až oveľa neskôr v procese učenia.

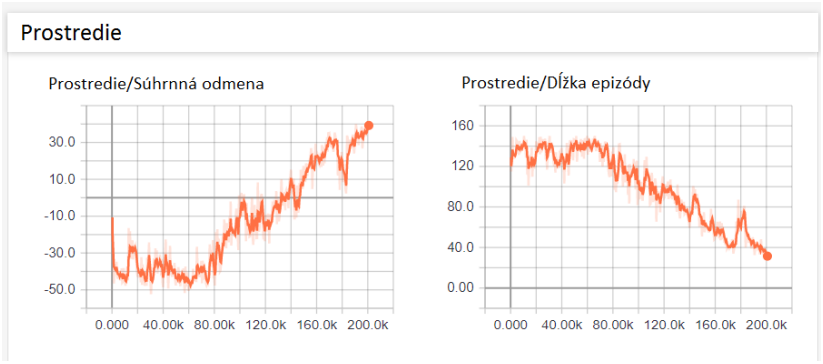
Ak sa agent dostal do cieľa reprezentovaného kockou, bola mu nastavená odmena rovná hodnote 40 a ukončená epizóda. Ak sa nedostal do cieľa za 5000 krokov, bola mu nastavená negatívna odmena -5 a resetovaná jeho pozícia na rovnakú, ako mal na začiatku. V oboch prípadoch bol znova vygenerovaný cieľ v menšej vzdialenosti agenta.

Tento proces učenia bol zrýchlený 15 násobne.

10.2. Strojové učenie bez formálnych metód

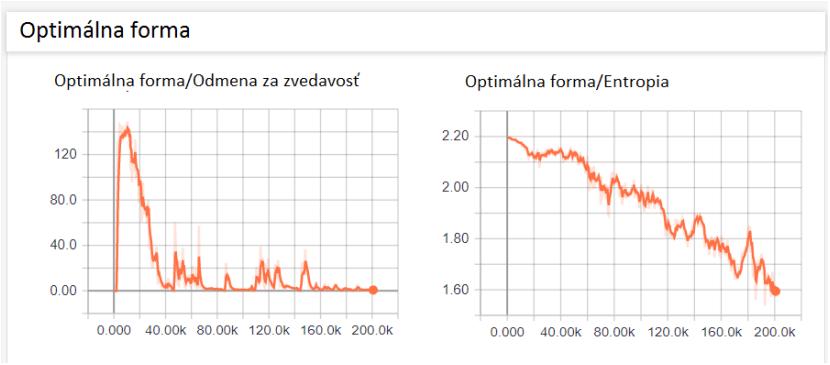
Proces učenia bol zapnutý bez aplikácie verifikovaného komponentu a ukončený po približne 200 000 krokoch. Na obrázku 10.1 vidíme dva grafy – graf Prostredie/Súhrnná

odmena zobrazuje, ako rástla dosiahnutá odmena agenta v čase a graf Prostredie/Dĺžka Epizódy ukazuje klesanie dĺžky epizódy v čase. Pozitívna odmena za agentovo správanie nastala po približne 135 000 krokoch.



Obrázok 10.1 – Grafy prostredia pre strojové učenie

Na obrázku 10.2 vidíme grafy pojednávajúce o optimálnej forme. Graf Optimálna Forma/ Odmena za zvedavosť ukazuje, že agent na začiatku ešte nevie aké akcie môže vykonávať a aké odmeny môže za tieto akcie očakávať. Druhý graf Optimálna forma/Entropia vyjadruje náhodnosť akcií agenta.



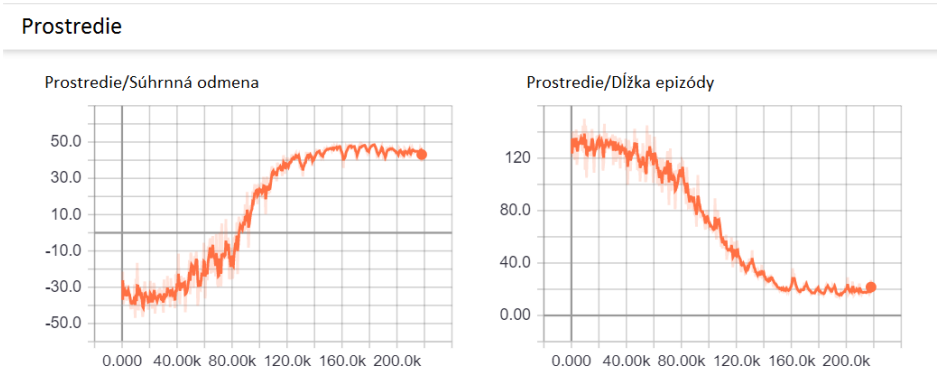
Obrázok 10.2 – Grafy pre optimálnu formu strojového učenia

Všetky 4 grafy zodpovedajú úspešnému procesu učenia, kde stúpanie súhrnnej odmeny znamená, že agent vykonával akcie a dostával sa do stavov, ktoré boli definované ako pozitívne. Pokles dĺžky epizódy, odmeny za zvedavosť a entropie znamená, že agent sa naučil vyberať akcie, ktoré viedli k najvyšším odmenám a dokázal splniť svoju úlohu v stále kratšom čase.

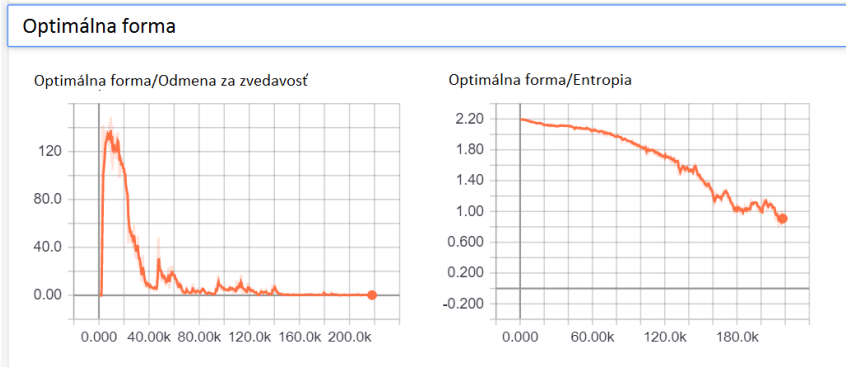
10.3. Strojové učenie s formálnymi metódami

Proces učenia bol spustený za rovnakých podmienok, až na jeden rozdiel – prítomnosť verifikovaného komponentu opísaného v kapitole 9. Pred tým, ako agent aplikuje rozhodnutia z mozgu, pošle informácie o svojom stave do tohto verifikovaného komponentu. Komponent sa na základe obdržaných informácií rozhodne, či bude meniť agentovo správanie, alebo sa budú používať rozhodnutia mozgu.

Výsledky po procese učenia vidíme na grafoch zobrazených v obrázkoch 10.3 a 10.4. Obe dvojice grafov ukazujú úspešný proces učenia. Takisto zobrazujú rýchlejšie dosiahnutie pozitívnej odmeny už po 90 000 krokoch a dosiahnutie optimálnej odmeny po 120 000 krokoch.



Obrázok 10.3 – Grafy prostredia pre strojové učenie s formálnymi metódami



Obrázok 10.4 – Grafy pre optimálnu formu strojového učenia s formálnymi metódami

10.4. Porovnanie výsledkov agenta

Tieto natrénované modely boli otestované dvoma spôsobmi – s použitím verifikovaného komponentu a bez jeho použitia, čím boli vytvorené 4 verzie agenta. Prvá verzia je agent používajúci model, ktorý bol natrénovaný bez použitia verifikovaného komponentu

a testovanie prebiehalo tiež bez komponentu. Druhá verzia má rovnaký model, ale pri testovaní bol použitý verifikovaný komponent. Tretia verzia používa model natrénovaný spolu s verifikovaným komponentom, ale testovanie prebiehalo už bez neho. Posledná verzia má model natrénovaný s komponentom a pri testovaní bol takisto použitý.

Testovanie prebiehalo za rovnakých podmienok ako učenie. V prostredí sa nachádzalo 5 agentov a každý bol rovnakej verzie. Pre každú verziu bol proces testovania spustený 4 krát v 10 minútových intervaloch. Na konci každého intervalu boli zaznamenané údaje o úspešnosti agenta o dosiahnutí jeho cieľa. Výsledky sú zobrazené v tabuľke 1, kde najlepšie dopadla štvrtá verzia agenta, kde bol model natrénovaný s verifikovaným komponentom a tento komponent bol takisto zapnutý pri testovaní. Tento agent mal aj najlepší pomer úspechu a zároveň mal aj najväčší počet dosiahnutých cieľov.

Typ modelu	Úspech	Neúspech	Pomer
Natrénovaný bez FM a spustený bez FM	3292	65	98.0255
Natrénovaný bez FM a spustený s FM	3175	61	98.0787
Natrénovaný s FM a spustený bez FM	4419	101	97.7144
Natrénovaný s FM a spustený s FM	4970	79	98.4105

Tabuľka 10.1 – Výsledky dosiahnutia cieľa agentom

10.5. Možné vylepšenia

Správanie agenta je v celku jednoduché, naučený je len pohyb pomocou diskrétnych akcií. Takisto verifikovaný komponent kontroluje vzdialenosti agenta od objektov len v 3 smeroch – vpredu, naľavo a napravo. Tieto dva aspekty by sa dali urobiť viac komplexnejšími, kde by agent zodpovedal viac robotickému kuriérovi. Okrem toho sa stále dajú testovať iné kombinácie odmien, pozorovaní a hyperparametrov. Nami zvolené viedli k pozitívnym výsledkom, avšak nemusia byť najideálnejšími.

11. Vyhodnotenie

Na začiatku boli analyzované súčasné trendy v odvetviach strojového učenia a formálnych metód a ich prepojenie. Zistilo sa, že táto téma sa začala rozoberať len za posledné roky a experti z oboch strán súhlasia, že môže viesť k lepšiemu návrhu a vývoju softvéru.

Ďalším krokom bolo porovnanie rôznych typov strojového učenia, ktoré by sa dali aplikovať do nami vytvorenej hry. Boli porovnané výhody aj nevýhody pre strojové učenie s dozorom, samostatné strojové učenie a učenie s posilňovaním. Učenie s posilňovaním sa používa na rôzne problémy, medzi ktoré patria aj vysoko komplexné hry. Tie by nebolo možné vyriešiť iným spôsobom a preto bolo strojové učenie s posilňovaním vybraté pre nášho agenta, reprezentujúceho robotického kuriéra.

Agent bol implementovaný v hernom rámci Unity pomocou ich nástroja ML-Agents. Na naučenie agenta bolo potrebné nájsť najlepšiu kombináciu odmien, pozorovaní a akcií. Dôležitým bodom bolo aj skúšanie rôznych hyperparametrov pre samotné strojové učenie. Po mnohých experimentoch bol výsledkom natrénovaný model pre agenta. Ten však mal stále nedostatky, z ktorých hlavnými boli priblíženie sa k objektom na príliš krátku vzdialenosť a hľadanie cieľa priamočiaro.

Na vyriešenie týchto problémov boli použité formálne metódy. V B-Metóde bol vyvinutý formálne verifikovaný komponent, ktorý kontroloval agentov stav a jeho okolie. Ak zistil, že agent sa nachádza v nebezpečnej situácii, prevzal nad ním kontrolu. Agent sa v tom prípade prestal rozhodovať na základe akcií z natrénovaného modelu, ale od príkazov z daného komponentu.

Následne bol agent vložený do herného prostredia a po otestovaní jeho rôznych verzií dosiahla najlepšie výsledky práve tá, ktorej natrénovaný model využíval formálne verifikovaný komponent od samého začiatku.

Záver

Výsledkom tejto práce je agent predstavujúci robotického kuriéra. Základným aspektom jeho vývoja je strojové učenie.

Strojové učenie a umelá inteligencia sú využívané v každej technologickej sfére a každodenne sa s nimi dostávame do styku, aj keď si to neuvedomujeme. Analyzované boli rôzne typy strojového učenia a dva projekty, OpenAI a AlphaStar, ktoré nasadili strojové učenie do komplexných hier. Podľa ich príkladu bolo použité strojové učenie s posilňovaním, ktorého podstatu tvorí definovanie odmien a pozorovaní agenta. Tieto údaje sú spracované naučeným modelom a ako odpoveď je vrátená odporúčaná akcia.

Pri tom môže nastať situácia, kedy si naučený model nemusí uvedomovať potenciálne riziko vyplývajúce z danej akcie. To môže nastať kvôli nedostatočnému procesu učenia alebo zlému definovaniu odmien a pozorovaní. Riešením tohto problému môže byť aplikovanie formálnych metód. Prepojenie týchto dvoch odvetví je diskutovaná téma, kde je viacero možností, ako ho docieliť. Vybraným riešením je použitie softvéru verifikovaného formálnymi metódami, ktorý by slúžil ako dozor nad ustrážením bezpečného správania agenta. Pomocou B-metódy bol vyvinutý takýto verifikovaný komponent, z ktorého bol následne vygenerovaný kód v jazyku C#.

Po kombináciou softvéru vytvoreného pomocou strojového učenia a formálne verifikovaného softvéru ho bolo potrebné otestovať. Nato je výhodné použiť virtuálne prostredie, ktoré ponúka viaceré výhody oproti testovaniu v reálnom svete. Virtuálne prostredie pre agenta bolo vytvorené v súvisiacej bakalárskej práci.

Nasadený agent bol otestovaný rôznymi spôsobmi, kde najlepšie výsledky mal práve ten agent, ktorý využíval aj strojové učenie, aj formálne metódy.

Zoznam použitej literatúry

[1] VINYALS, O. et al. *AlphaStar: Mastering the Real-Time Strategy Game StarCraft II* [https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii] (2019).

[2] COPELAND, M.: *What's the Difference Between Artificial Intelligence, Machine Learning, and Deep Learning?* [https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/] (2019).

[3] MAGLOGIANNIS I. G.: *"Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in EHealth, HCI, Information Retrieval and Pervasive Technologies". (2007*).

[4] dSpace: *Virtual Validation with dSPACE: Benefits the entire ECU development process* [https://www.dspace.com/shared/data/pdf/2018/dSPACE-Virtual-Validation_Business_field_brochure_2018_English.pdf] (2019).

[5] SUTTON, R. S.; BARTO, A. G.: *"Reinforcement Learning: An Introduction".* MIT Press, (1998).

[6] SCHULMAN, J; KLIMOV, O.; WOLSKI, F.; DHARIWAL, P.; RADFORD, A.: *Proximal Policy Optimization* [https://openai.com/blog/openai-baselines-ppo/] (2019).

[7] JULIANI, A., BERGES, V., VCKAY, E., GAO, Y., HENRY, H., MATTAR, M., LANGE, D.: Unity: *A General Platform for Intelligent Agents*. *arXiv preprint arXiv:1809.02627.* [https://github.com/Unity-Technologies/ml-agents] (2018).

[8] FUMO, D.: *Types of Machine Learning Algorithms You Should Know* [https://towardsdatascience.com/types-of-machine-learning-algorithms-you-should-know-953a08248861] (2019).

[9] SELSAM, D., LIANG, P., & DILL, D.L.: *"Developing Bug-Free Machine Learning Systems With Formal Mathematics"*. *ICML*. (2017).

[10] CLARKE E. M.; WING J. M.; et al. *"Formal Methods: State of the Art and Future Directions". (1996).*

[11] NIGRETTI, A.: *Using Machine Learning Agents Toolkit in a real game: a beginner's guide* [https://blogs.unity3d.com/2017/12/11/using-machine-learning-agents-in-a-real-game-a-beginners-guide/] (2019).

[12] Formal Verification Meets Machine Learning [https://moves.rwth-aachen.de/teaching/ss-18/fvmml/] (2018).

[13] Summit on machine learning meets formal methods [https://www.turing.ac.uk/events/summit-machine-learning-meets-formal-methods] (2018).

[14] PLATZER A.: Safe reinforcement learning via formal methods [https://easychair.org/smart-slide/slide/jTdT#] (2018).

[15] SESHIA, S. A.; ZHU, X.; KRAUSE, A.; JHA, S.: Machine Learning and Formal Methods [http://drops.dagstuhl.de/opus/volltexte/2018/8430/pdf/dagrep_v007_i008_p055_17351.pdf] (2018).

[16] WESEL, P.; GOODLOE A. E.: *"Challenges in the verification of reinforcement learning algorithms"*. Technical report, NASA, (2017).

[17] MUEHLHAUSER, L.: *Emil Vassev on Formal Verification* [https://intelligence.org/2014/01/30/emil-vassev-on-formal-verification/] (2019).

[18] GOODFELLOW, I.; PAPERNOT, N.: *The challenge of verification and testing of machine learning* [http://www.cleverhans.io/security/privacy/ml/2017/06/14/verification.html] (2019).

[19] SILVER, D.; HUANG, A. et al. *"Mastering the game of Go with deep neural networks and tree search"*. Nature, (2016).

[20] KOREČKO, Š.; HUDÁK, Š.: *"Formálne metódy pre diskrétne systémy: Petriho siete a B-metóda"*. Technická univerzita v Košiciach, (2015).

[21] AREL, I.; LIU, C.; URBANIK, T.; KOHLS A.*: "Reinforcement learning-basedmulti-agent system for network traffic signal control".* IET Intelligent Transport Systems, (2010).

[22] SILVER, D.; HUANG, A. et al*. "Mastering the game of go without human knowledge".* Nature, (2017).

[23] RUSSEL, S.; NORVIG, P.: *"Artificial Intelligence. A modern approach". Prentice-Hall*, Egnlewood Cliffs 25, (1995).

[24] KOREČKO, Š.; HUDÁK, Š.; DOBOŠ, J.; ai.: *"Decompositional Mw Automaton-Based Reachability Algorithm". Egyptian Computer Science Journal*, ročník 37, č. 6, (2013).

[25] Web Stránka nástroja BKPI. [http://hron.fei.tuke.sk/~korecko/FMInGamesExp/resources/BKPICompiler.zip] (2019).

[26] Web Stránka nástroja Atelier B. [https://www.atelierb.eu/en/] (2019).

[27] DONGES, N.: *Gradient Descent in a Nutshell* [https://towardsdatascience.com/gradient-descent-in-a-nutshell-eaf8c18212f0] (2018).

[28] KOREČKO, Š.; SOBOTA, B.; ZEMIANEK, P.: *"Jadex/JBdiEmo Emotional Agents in Games with Purpose: a Feasibility Demonstration"*. SNE (2016): (195).

[29] RAUCH, R.: "Formálne metódy a strojové učenie vo virtuálnom priestore: Interaktívne virtuálne prostredie". Technická univerzita v Košiciach, (2019).

[30] ARNALDI, B.; COZOT, R.; DONIKIAN, S.: *"Simulating Automated Cars in a Virtual Urban Environment"*. Göbel M. (eds) Virtual Environments '95. Eurographics. (1995).

Zoznam príloh

Príloha A - Systémová príručka.

Príloha B - Používateľská príručka.

Príloha C - CD médium – záverečná práca v elektronickej podobe.