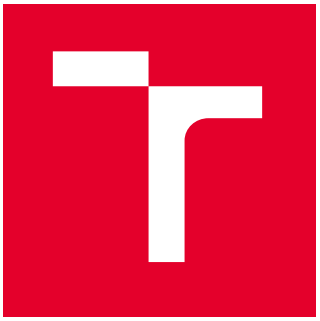


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

OPTIMALIZACE ŘÍZENÍ S POMOCÍ ZPĚTNOVAZEBNÍHO UČENÍ NA PLATFORMĚ ROBOCODE

OPTIMIZATION OF CONTROL USING REINFORCEMENT LEARNING ON THE ROBOCODE PLATFORM

SEMESTRÁLNÍ PRÁCE
SEMESTRAL THESIS

AUTOR PRÁCE — **Bc. Václav Pastušek**
AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. Radim Burget, Ph.D.**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Semestrální práce

magisterský navazující studijní program **Telekomunikační a informační technika**

Ústav telekomunikací

Student: Bc. Václav Pastušek ***ID:*** 204437

Ročník: 2 ***Akademický rok:*** 2023/24

NÁZEV TÉMATU:

Optimalizace řízení s pomocí zpětnovazebního učení na platformě Robocode

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s problematikou optimalizace řízení robotů a s využitím zpětnovazebního učení na platformě Robocode. Proveďte rešerši ohledně současného stavu vědy a techniky v této oblasti, včetně algoritmů Q-learning a Deep Q Networks. Vytvořte trénovací a testovací datovou množinu simulací bojových robotů v prostředí Robocode. Navrhněte několik metod založených na různých strategiích zpětnovazebního učení, s pomocí kterých budou roboti schopni efektivněji se vyhýbat nepřátelům, cílit a střílet. Navržené metody vhodným způsobem srovnejte a dosažené výsledky diskutujte za účelem dosažení optimálního řízení bojových robotů.

DOPORUČENÁ LITERATURA:

podle pokynů vedoucího práce

Termín zadání: 2.10.2023 ***Termín odevzdání:*** 13.12.2023

Vedoucí práce: doc. Ing. Radim Burget, Ph.D.

prof. Ing. Jiří Mišurec, CSc.
předseda rady studijního programu

ABSTRAKT

Tato semestrální práce se zaměřuje na optimalizaci řízení tankového robota v prostředí Robocode pomocí zpětnovazebního učení. Teoretická část se věnuje důkladnému zkoumání platformy Robocode, zpětnovazebního učení a příslušných algoritmů. V praktické části byly vytvořeny dva modely, z nichž model se ztrátovou funkcí MSE dosáhl lepších výsledků ve srovnání s modelem využívajícím funkci MAE. Tyto modely byly úspěšně natrénovány na původním robotu SpinBot, který vykazoval nejlepší výkony ze všech dostupných pokročilých robotů.

KLÍČOVÁ SLOVA

Robocode, strojové učení, zpětnovazební učení, epizodická paměť, Q-učení, hluboké učení, hluboká Q síť, TCP/IP, klient-server, Java, Python, TensorFlow

ABSTRACT

This semester project focuses on optimizing the control of a tank robot in the Robocode environment using reinforcement learning. The theoretical part involves a thorough examination of the Robocode platform, reinforcement learning, and relevant algorithms. In the practical section, two models were created, with the MSE loss function model achieving better results compared to the model using the MAE function. These models were successfully trained on the original SpinBot robot, which exhibited the best performance among all available advanced robots.

KEYWORDS

Robocode, machine learning, reinforcement learning, episodic memory, Q-learning, deep learning, deep Q network, TCP/IP, client-server, Java, Python, TensorFlow

PASTUŠEK, Václav. *Optimalizace řízení s pomocí zpětnovazební učení na platformě Robocode*. Semestrální práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2023. Vedoucí práce: doc. Ing. Radim Burget, Ph.D.

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Bc. Václav Pastušek
VUT ID autora:	204437
Typ práce:	Semestrální práce
Akademický rok:	2023/24
Téma závěrečné práce:	Optimalizace řízení s pomocí zpětnovazební učení na platformě Robocode

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

Prohlášení o použití nástrojů

Pro tvorbu této práce byl využit pokročilý softwarový nástroj pro stylizaci textu či návrh částí kódů, jako je OpenAI ChatGPT. Veškeré takto získané výstupy byly manuálně validovány, uzpůsobeny pro potřeby této práce a prohlašuji, že za výslednou podobu a spolehlivost přebírám veškerou zodopovědnost.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu doc. Ing. Radimu Burgetovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů

Úvod

V dnešní době, kdy umělá inteligence a strojové učení dominují diskuzím v technologickém světě, není pochyb o vzrůstajícím významu těchto oblastí. Jejich potenciál překračuje hranice a zdá se, že jsou stále více v centru pozornosti. V této době rychlého technologického pokroku vzniká nejen poptávka po odbornících na umělou inteligenci a strojové učení, ale také po efektivních výukových metodách, které tuto poptávku mohou uspokojit.

V rámci tohoto trendu se objevuje výzva v oblasti praktické výuky, zejména co se týče metod strojového učení. Nalézt zajímavou problematiku pro studenty, kterou by měli řešit, a zároveň umožnit komplexní zhodnocení kvality jejich práce, představuje nelehký úkol. Zvláště při složitějších úkolech je obtížné jednoznačně určit, zda je dané řešení správné či nikoliv. Navíc porovnávat více různých přístupů a hodnotit, které z nich je nejefektivnější, představuje další vrstvu náročnosti.

V této souvislosti se tato práce zabývá konkrétním výzkumem v oblasti optimalizace řízení tankových robotů v prostředí modifikované platformy Robocode, napsané v jazyce Java, pomocí zpětnovazebního učení. Jedná se o 2D simulovanou bojovou hru, která hráčům umožňuje programovat vlastní bojové roboty a posléze je nasazovat do virtuálních arén. Tímto se snažíme nejen přinést nové poznatky v rámci této konkrétní problematiky, ale také reagovat na aktuální potřeby vzdělávání v oblasti umělé inteligence a strojového učení. Cílem této úráce je ukázat, že i v komplexních úlohách lze nalézt smysluplný výzkum a poskytnout studentům příležitost k praktickému zapojení do této dynamické a perspektivní oblasti.

Semestrální práce je rozdělena na teoretickou a praktickou část. První podkapitola teoretické části se věnuje historii platformy Robocode a její modifikované verzi, kterou vytvořil autor Vladimír Peňáz v rámci své bakalářské práce pro hodnocení studentských projektů u předmětu MPC-PDA [1]. Tato upravená verze umožňuje každému studentskému robotovi vidět všechny tanky a střely na herní mapě, což jim poskytuje stejné informace o bitvě jako lidským hráčům. Kromě toho obsahuje bezpečnou komunikaci mezi serverem a klienty s TCP/IP zabezpečením, které bylo implementováno s cílem chránit učitelský počítač před neoprávněným přístupem ze strany studentských robotů. Tato funkce je také využitelná mimo školní prostředí prostřednictvím propojení klientů a serveru pomocí virtuální privátní sítě (VPN), například za použití aplikace Radmin VPN, která nevyžaduje registraci.

Druhá podkapitola teoretické části se zabývá analýzou metod zpětnovazebního učení a jejich možné implementace v prostředí Robocode. Zároveň zkoumáme výpočetní složitost těchto algoritmů.

Praktická část této práce je zaměřena na optimalizaci virtuální arény a tanků vytvořených studenty. Pro trénování byla zvolena hluboká Q-síť (DQN) s epizodic-

kou pamětí, která vyniká svou jednoduchostí a schopností eliminovat možné oscilace během učení. Implementace této metody probíhá v jazyce Python pro jeho flexibilitu a kompatibilitu s knihovnou TensorFlow. V rámci práce budou vytvořeny testovací a trénovací modely, které postupně akumulují zkušenosti, zdokonalují své schopnosti v reakci na nepřátele, zaměřování na cíle a střílení s co nejlepšími výsledky.

Následně provedeme série experimentů s různými parametry epizodické DQN s cílem optimalizovat řízení tanků. Výsledky těchto experimentů budou pečlivě testovány, porovnány a diskutovány s cílem hodnotit dosaženou úroveň řízení bojových robotů v simulátoru Robocode.

V závěru této práce byly úspěšně natrénovány 2 modely tanků na původním robotu SpinBot, který vykazoval nejlepší výkony ze všech dostupných pokročilých robotů. Bylo zjištěno, že model využívající ztrátovou funkci MSE dosáhl lepších výsledků ve srovnání s modelem, který využíval funkci MAE.

1 Teoretická část

V této kapitole bude prozkoumán teoretický základ, který nám poskytne hlubší vhled do platformy Robocode a zároveň nás zavede do problematiky zpětnovazebního učení.

1.1 Platforma Robocode

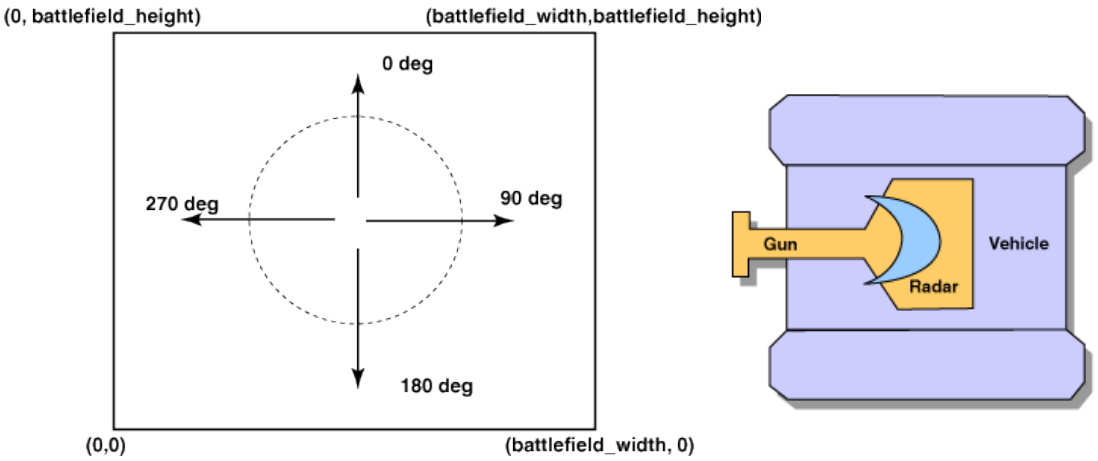
Robocode představuje programovací hru, ve které vytváříte bojové tanky v programovacím jazyce Java a necháváte je soupeřit v reálném čase [2].

Hráč není schopen ovládat tanky přímo během hry, místo toho může napsat umělou inteligenci pro svého robota, určující jeho chování a reakce na události, které se mohou v bojové aréně objevit. Název „Robocode“ samo o sobě naznačuje, že jde o kódování robotů. [3]

Hlavním záměrem hry je poskytnout zábavný způsob, jak se naučit programovací jazyk Java. Robocode je kompatibilní s různými operačními systémy, jako jsou Windows, MacOS a Linux, díky své platformě napsané v Javě. [3]

Souboje ve hře Robocode se odehrávají na bitevním poli, kde spolu bojují malí automatizovaní šestikoloví roboti, dokud nezůstane pouze jeden. Tato hra neobsahuje násilí ani politiku a je určena pouze pro soutěžní zábavu. [3]

Bitevní pole je dvojrozměrné a má základní velikost 800x600 buněk, ale lze ji nastavit od 400x400 až po 5000x5000 buněk. Jak můžeme vidět na obrázku č. 1.1 (vlevo), nulová dvojice začíná v dolním levém rohu GUI, což je jiné než na počítačové obrazovce, kde toto platí pro levý horní roh. Tank lze rozdělit na tři části: vozidlo, zbraň a radar (viz obrázek č. 1.1 vpravo).



Obr. 1.1: Souřadnicový systém (vlevo) [4] a anatomie tanku (vpravo) [5]

Hra Robocode omezuje hráče na tři hlavní akce: pohyb tanku po celém herním bojišti, otáčení věže a výstřel. Hrací plocha umožňuje tanku pohyb vpřed a vzad, přičemž tank se může otáčet, avšak není schopen pohybu bokem. Dělová věž tanku má plný záběr 360 stupňů a umožňuje střelbu. Na věži je také radar, který detekuje nepřátelské tanky v okruhu 120 pixelů. Při střelbě má hráč kontrolu nad sílou výstřelu, kde silnější výstřel způsobuje větší poškození, ale spotřebuje více energie. [7]

Hra probíhá v několika kolech, přičemž hráči začínají se 100 body energie, které představují jejich životy. Klesne-li energie na nulu, hráčův tank je buď vypnut nebo zničen. Energie se snižuje při zásahu střelou, nárazu do stěny (kde vyšší rychlost znamená větší poškození) nebo do tanku (kde poškození je konstantní), a při nepřesných střelbách. Při střelbě se generuje také teplo, nad určitou mez se nedá střílet a tank musí se počkat na ochlazení děla. [7, 8]

Hráč může získat energii primárně úspěšným zásahem nepřítele zničením nepřátelských tanků. Cílem hry není pouze přežít jako poslední, ale získat co nejvyšší bodové skóre v průběhu všech kol. [7]

Platforma vyžaduje JDK ve verzi 11 nebo novější [2] a můžete ji stáhnout z platformy SourceForge[1] nebo z GitHubu[2]. Pro začátečníky poskytuje přehledný manuál a návody na RoboWiki[3]. Na webu je také k dispozici sekce FAQ[4] s nejčastěji kladenými otázkami ohledně Robocode.

1.1.1 Historie platformy Robocode

Následující historické informace jsou parafrázovány ze zdroje [6].

Hra Robocode vznikla jako soukromý projekt Matheze A. Nelsona koncem roku 2000. Svůj profesionální charakter získala až po představení v IBM prostřednictvím Alphaworks v červenci 2001.

Firma IBM projevila zájem o Robocode s cílem propagovat ho jako zábavný nástroj pro učení programování v Javě. Inspirací pro vytvoření Robocode byla hra Robot Battle napsaná Bradem Schickem v roce 1994, která byla inspirována hrou RobotWar pro Apple II+ z počátku 80. let. Robocode se stalo populárním díky článkům od IBM a komunitě za RoboWiki[3]. Po mnoho let bylo a stále je využíváno pro výuku a výzkum na školách a univerzitách po celém světě.

Na začátku roku 2005 přesvědčil Mathew IBM, aby Robocode uvolnilo jako open-source na platformě SourceForge[1].

[1] `https://sourceforge.net/projects/robocode/files/`

[2] `https://github.com/robo-code/robocode/releases`

[3] `https://robowiki.net/wiki/Main_Page`

[4] `https://robowiki.net/wiki/Robocode/FAQ`

Flemming N. Larsen následně převzal projekt Robocode na platformě SourceForge v červenci 2006 a pokračoval vývojem oficiální verze Robocode 1.1 s mnoha vylepšeními. Následně bylo vydáno mnoho nových verzí Robocode s rozšířenými funkcemi a příspěvky komunity.

V květnu 2007 byl do Robocode integrován klient RoboRumble, který slouží komunitě k vytváření aktuálního žebříčku robotů pro různé soutěže.

V roce 2012 uživatel Robocode jménem Julian („Skilgannon“) vytvořil systém LiteRumble, určený k běhu na platformě Google App Engine, s cílem poskytnout lehký a snadno nasaditelný systém pro RoboRumble. Verze na GitHubu[5].

V květnu 2010 byl pro Robocode poskytnut zásuvný modul .Net umožňující vývoj robotů pro .Net Framework verze 3.5 vedle vývoje robotů v Javě, díky Pavelu Savarovi. Bohužel, v dubnu 2021 byl tento modul zrušen z důvodu problémů s .Net Framework a nástroji potřebnými pro jeho vývoj a udržování.

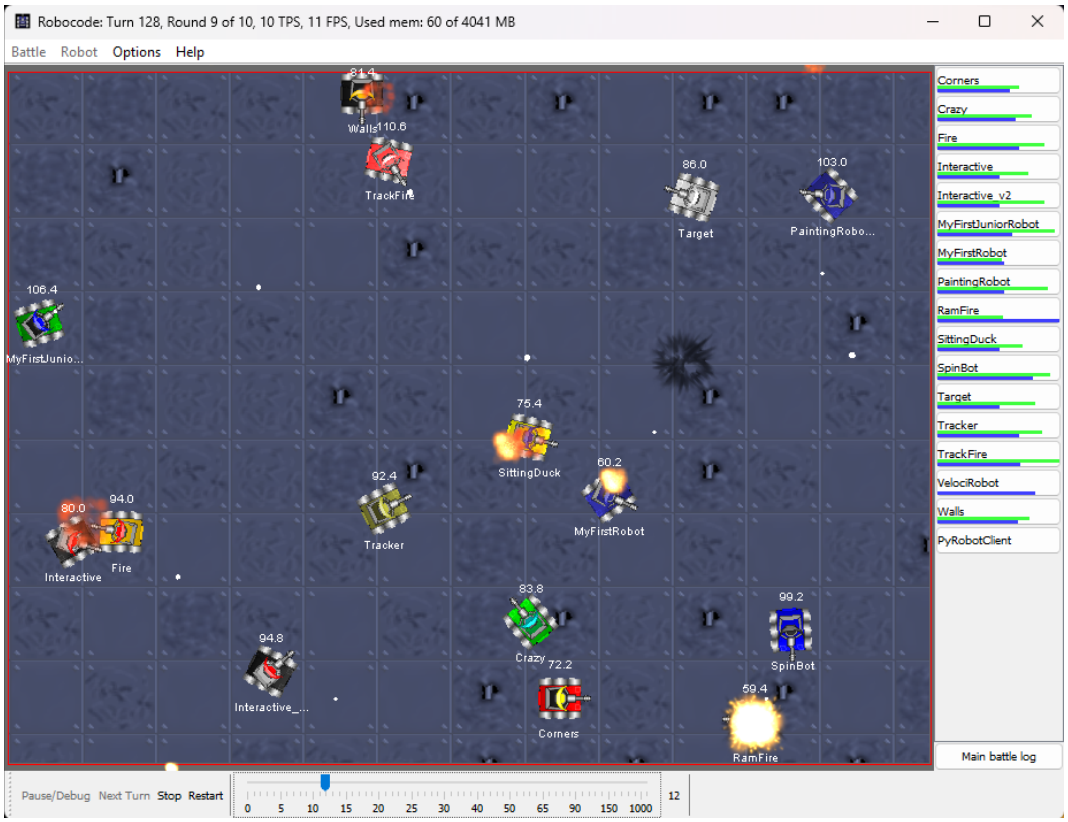
1.1.2 Modifikovaná verze platformy Robocode

Tato práce navazuje na modifikovanou verzi platformy Robocode, která byla upravena za účelem využití při hodnocení studentských projektů v rámci předmětu MPC-PDA autorem Vladimírem Peňázem v rámci jeho bakalářské práce [1]. Tato modifikace přináší dvě hlavní změny do platformy Robocode.

První z těchto změn spočívá v implementaci bezpečného testovacího prostředí. Hlavním cílem je zajištění toho, že studenti budou moci soutěžit na herním serveru s minimálním rizikem poškození učitelské výpočetní stanice. Před spuštěním soutěže je nutné provést konfiguraci správných IP adres a portů pro studentské tanky v souboru game.properties. Poté jsou spuštěny studentské klienty a následně i učitelský server. Začne turnajový souboj, na jehož konci je výpis skóre pro každý tank. Server i klient je napsaný v jazyce Java. Klient následně kontinuálně volá Python, který získává stavové informace a odesílá akce tanku. Nicméně tato metoda může být neefektivní, protože vyžaduje opakované vytváření virtuálního prostředí (interpretu) pro Python a načítání knihoven, jako je Tensorflow (což může trvat až 3 sekundy). Pro tuto část bude vypracována optimalizace, jak je popsáno v sekci Návrh řešení 2.2.

Druhá změna spočívá v modifikaci radaru. V této úpravě získává trénovací tank informace o všech ostatních tancích a střelách, namísto toho, aby obdržel informace jen z dané výseče od radaru. Stejné informace mohou být viděny i prostřednictvím grafického uživatelského rozhraní (viz obrázek č. 1.2).

[5] `https://github.com/jkflying/literumble`



Obr. 1.2: GUI platformy Robocode

1.2 Strojové učení – ML

Strojové učení (machine learning) je odvětvím umělé inteligence, které se zaměřuje na vývoj algoritmů a modelů, které umožňují počítačům automaticky se učit a zlepšovat své výkony na základě zkušeností. Učení s učitelem, podobně jako učení bez učitele a zpětnovazební učení, patří mezi základní formy strojového učení. Rozdělení strojového učení viz obr. č. 1.3.

Obr. 1.3: Diagram strojového učení [9]

Učení s učitelem – SL

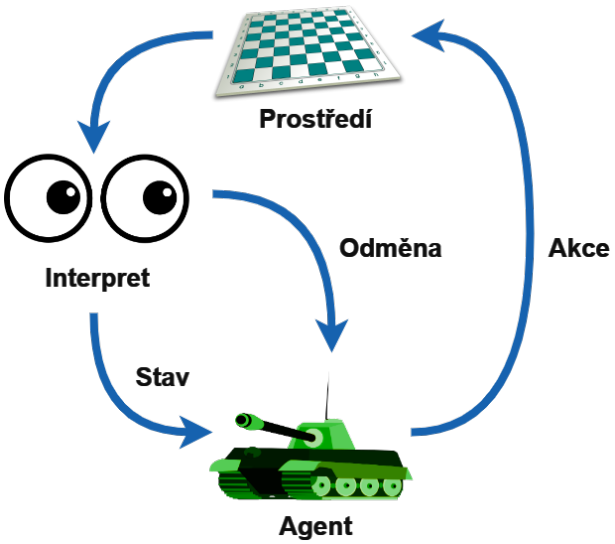
V případě učení s učitelem (supervised learning) jsou příklady v datasetu označeny cílovými hodnotami, což umožňuje algoritmu porovnávat jeho výstupy s očekávanými hodnotami a upravovat své chování na základě rozdílu mezi nimi.

Učení bez učitele – UL

Učení bez učitele (unsupervised learning) se zaměřuje na shlukování dat bez přítomnosti cílových hodnot v datasetu. Algoritmus se snaží identifikovat vzory a struktury v datech, aby mohl seskupovat podobné příklady do shluků.

1.2.1 Zpětnovazební učení – RL

Zpětnovazební učení [10] (reinforcement learning), na rozdíl od učení s učitelem a učení bez učitele, se specializuje na online kontrolní úlohy, jako je ovládání robotů nebo hraní her. V těchto scénářích se agent snaží naučit politiku pro své akce na základě odměn a trestů, které obdrží za své interakce s prostředím (viz obrázek č. 1.4). Cílem agenta je získat vhodnou mapu, která spojuje jeho aktuální pozorování prostředí a vnitřní stav (paměť) s optimální akcí.



Obr. 1.4: Diagram zpětnovazebního učení [11]

V rámci zpětnovazebního učení je časté provádění v reálném čase, kde je agent vypuštěn do prostředí, aby experimentoval s různými strategiemi. Začíná s potenciálně náhodnou politikou a postupně aktualizuje svou politiku v reakci na odměny, které obdrží za své akce. V případě pozitivní odměny se posiluje mapování mezi pozorováním, stavem a danou akcí, zatímco v případě negativní odměny se oslabuje.

Zpětnovazební učení představuje specifické výzvy, neboť učení probíhá v situaci, kde jsou fáze tréninku a inference vzájemně propletené a probíhají současně. Agent musí neustále odhadovat, jakou akci podniknout, a využívá zpětnou vazbu z prostředí k učení, jak aktualizovat svou politiku.

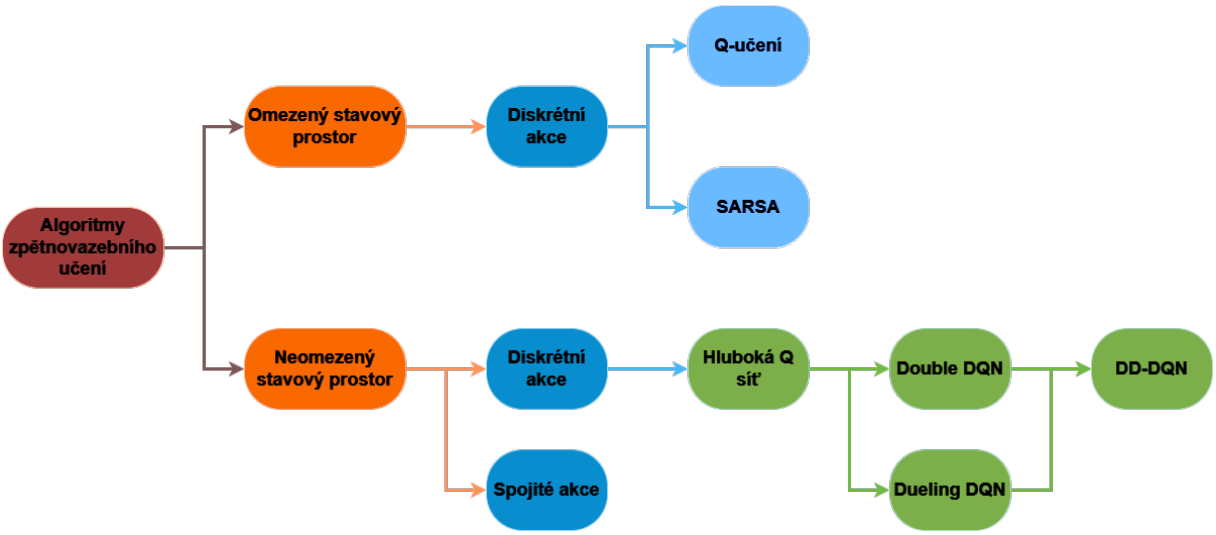
Jedním z charakteristických rysů zpětnovazebního učení je oddělení cílového výstupu naučené funkce (akce agenta) od mechanismu odměn. Odměna může záviset na více akcích, a nemusí být dostupná ihned po provedení akce. Například v šachovém scénáři může být odměna $+1$, pokud agent vyhraje, a -1, pokud prohraje, ale tato zpětná vazba bude k dispozici až po dokončení posledního tahu hry. Toto představuje výzvu v návrhu tréninkových mechanismů, které správně distribuují odměnu

zpětně přes sekvenci akcí, aby politika mohla být vhodně aktualizována.

Zpětnovazební učení zaznamenalo v posledních letech výrazný rozmach, přičemž jedním z významných příkladů je demonstrace společnosti DeepMind Technologies of Googlu. Ta předvedla efektivní využití zpětnovazebního učení při trénování modelu hlubokého učení pro ovládání strategií v sedmi různých počítačových hrách na platformě Atari. Systém využíval surové hodnoty pixelů ze hry, přičemž ovládací strategie specifikovaly, jaké akce by měl agent provádět na joysticku v každém okamžiku hry. Tato inovativní metoda ukázala svou účinnost zejména v prostředí počítačových her, kde agent odehrál tisíce her proti počítačovému hernímu systému, a to bez nutnosti vytváření nákladného datasetu s označenými příklady situací a akcí na joysticku. [12]

1.3 Algoritmy zpětnovazebního učení

Zpětnovazební učení (RL) zahrnuje širokou škálu algoritmů, které umožňují agentovi učit se chování v interaktivním prostředí. Tato sekce poskytuje krátký přehled o kategoriích algoritmů a jejich klíčových aspektech v rámci rozdělení RL na omezené a neomezené stavy a na diskrétní a spojité akce (viz obrázek č. 1.5)).



Obr. 1.5: Rozdělení zpětnovazebního učení [13]

Omezené a neomezené stavy

Zpětnovazební učení může být rozděleno podle typu stavů prostředí, ve kterém agent operuje. Omezené stavy odkazují na situace, kdy agent má k dispozici omezený a předem definovaný soubor stavů. Naopak neomezené stavy umožňují agentovi vnímat širší a komplexnější spektrum prostředí, což může zahrnovat nekonečný počet možných stavů.

Diskrétní a spojité akce

Dalším důležitým kritériem pro klasifikaci algoritmů zpětnovazebního učení je charakter akcí, které agent může podniknout. Diskrétní akce znamená, že agent má omezený počet diskrétních akcí, které může vykonat v každém okamžiku. Naopak spojité akce umožňují agentovi provádět akce v kontinuálním rozsahu, což je relevantní zejména v prostředích, kde jsou akce plynulé a mohou nabývat nekonečného množství hodnot.

Epizodické a kontinuální úlohy

Epizodická úloha [14] je typ úlohy ve zpětnovazebním učení, kde interakce mezi agentem a prostředím tvoří sérii oddělených epizod. Naopak kontinuální úloha nemá přirozené rozdělení na epizody a může trvat nekonečně dlouho. Epizodické úlohy jsou matematicky jednodušší, protože každá akce agenta ovlivňuje pouze konečný počet odměn během jedné epizody.

K epizodickým úlohám bývá často přidávána epizodická paměť, která uchovává informace o každé jednotlivé epizodě. Tato paměť je užitečná při tréninku agenta, zejména při zpracování informací o průběhu jednotlivých epizod.

Kontinuální úloha nikdy neskončí, což znamená, že odměna na konci není definována. Naopak epizodická úloha má konečnou dobu trvání, například jedno kolo hry Go. V kontinuální úloze mohou být odměny přiděleny s diskontováním, kde nedávné akce obdrží větší odměnu než akce z minulosti, a to pomocí faktoru diskontování λ.

1.3.1 Q-učení

Q-učení [15] (Q-learning) je algoritmus, který slouží k učení optimální strategie rozhodování pro agenta v interaktivním prostředí. Tento přístup patří do kategorie Temporal-Difference (TD) learning a byl vyvinut Christopherem J.C.H. Watkinsem v roce 1989.

Cílem Q-učení je naučit funkci hodnoty akce, známou jako Q-funkce. Tato funkce přiřazuje každému možnému stavu a akci očekávanou kumulativní odměnu[6], kterou agent získá, když se nachází ve stavu a provede danou akci.

Pro reprezentaci a aktualizaci Q-funkce se často používá Q-tabulka, což je tabulka, kde každý řádek odpovídá stavu a každý sloupec odpovídá akci. Hodnoty v tabulce reprezentují odhady kumulativních odměn pro dané kombinace stavů a akcí. Q-tabulka může být chápána jako forma Look-Up Table (LUT).

Aktualizace Q-funkce v Q-učení je prováděna pomocí Temporal-Difference update pravidla, což umožňuje agentovi adaptovat své rozhodování na základě nových zkušeností z interakce s prostředím. Tento proces je založen na Bellmanově rovnici, která poskytuje teoretický rámec pro aktualizaci hodnoty akce na základě odhadů budoucích odměn.

Formální vyjádření aktualizace Q-funkce v Q-učení s Bellmanovou rovnicí je následující:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]. \tag{1.1}$$

[6] Celkový součet odměn získaných agentem při provedení dané akce ve stavu a následných akcích a stavech v průběhu času. Tato hodnota reflektuje celkový očekávaný přínos této akce v daném kontextu.

Kde:

- $Q(S_t, A_t)$ je hodnota Q-funkce pro stav S_t a akci A_t,
- α je velikost kroku (learning rate),
- R_{t+1} je odměna získaná za provedení akce A_t ve stavu S_t,
- γ je diskontní faktor,
- $\max_a Q(S_{t+1}, a)$ představuje maximální odhad hodnoty Q-funkce pro stav S_{t+1} přes všechny možné akce.

Q-učení s Q-tabulkou se často využívá v prostředích s diskrétními stavy a akcemi, kde Q-tabulka může efektivně reprezentovat hodnoty pro všechny možné kombinace stavů a akcí.

1.3.2 SARSA

Nyní, po seznámení s Q-learningem, přicházíme k dalšímu algoritmu posilovaného učení nazývanému SARSA [16]. Obě tyto metody patří do kategorie Temporal-Difference (TD) learning a sdílí několik základních principů. Stejně jako Q-learning se i SARSA zaměřuje na učení optimální strategie rozhodování pro agenta v interaktivním prostředí s diskrétními stavy a akcemi.

Jméno „SARSA“ vychází z charakteristického způsobu, jakým algoritmus aktualizuje hodnoty Q-funkce. Každá písmena v názvu představují jednu část kvintupletu událostí: Stav (**S**tate), Akce (**A**ction), Odměna (**R**eward), Stav (**S**tate) a Akce (**A**ction). Tato pětice udává, co se děje od jednoho páru stavu-akce k následujícímu. Takovýmto způsobem se algoritmus učí a aktualizuje své odhady hodnot akcí.

Hlavním rozdílem mezi Q-learningem a SARSA je způsob, jakým tyto hodnoty odhadují hodnotu akce. Q-learning se snaží předvídat maximální hodnotu pro následující akci ve stavu, což může být výhodné v situacích, kde odměny jsou stabilní a jednoznačné. Na druhé straně SARSA přímo odhaduje hodnotu aktuální akce podle aktuální politiky. Tato vlastnost může být výhodná v prostředích s nejednoznačnými nebo náhodnými odměnami, kde se odhady hodnot akcí mohou lépe přizpůsobovat aktuálním podmínkám prostředí.

Formální vyjádření aktualizace Q-funkce v SARSA s Bellmanovou rovnicí je následující:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \tag{1.2}$$

Kde:

- $Q(S_t, A_t)$ je hodnota Q-funkce pro stav S_t a akci A_t,
- α je velikost kroku (learning rate),
- R_{t+1} je odměna získaná za provedení akce A_t ve stavu S_t,
- γ je diskontní faktor,

- $Q(S_{t+1}, A_{t+1})$ představuje hodnotu Q-funkce pro nový stav S_{t+1} a novou akci A_{t+1}.

1.3.3 Hluboká Q síť – DQN

Hluboká Q síť [17, 18] (deep Q network) je osvědčenou metodou pro řešení úloh zpětnovazebního učení, kde klíčovou roli hraje volba Q-funkce. DQN úspěšně využívá hluboké neuronové sítě (DNN) k aproximaci hodnot akcí $Q(s, a; \theta)$, kde θ jsou parametry sítě a (s, a) představují pár stav-akce.

V mnoha implementacích DQN se často používají dvě hluboké sítě: hlavní síť Q_θ a cílová síť $Q_{\bar{\theta}}$. Hlavní síť slouží pro predikci hodnot Q-funkce, zatímco cílová síť je využívána pro stabilizaci učení. Tato dualita sítí pomáhá eliminovat oscilace a zlepšuje stabilitu trénování. Parametry hlavní sítě Q_θ jsou aktualizovány ve směru gradientu pomocí odměn a predikcí cílové sítě $Q_{\bar{\theta}}$, což vede k lepší konvergenci a efektivnějšímu učení.

Ztrátová funkce DQN

Parametry neuronové sítě jsou optimalizovány pomocí stochastického gradientního sestupu k minimalizaci následující ztrátové funkce:

$$L(\theta) = E\left[(r_t + \gamma \cdot \max_{a'} Q_{\bar{\theta}}(s_{t+1}, a') - Q_\theta(s_t, a_t))^2\right] \tag{1.3}$$

Kde:

- $L(\theta)$: Loss funkce, kterou chceme minimalizovat optimalizací parametrů θ.
- θ: Parametry modelu, které chceme optimalizovat, například váhy neuronové sítě.
- E: Očekávaná hodnota (expectation), která reprezentuje průměrnou hodnotu přes všechny možné hodnoty náhodné veličiny[7].
- r_t: Odměna získaná za provedení akce a_t ve stavu s_t
- γ Diskontní faktor.
- $\max_{a'} Q_{\bar{\theta}}(s_{t+1}, a')$: Nejvyšší očekávaná hodnota akce a' ve stavu s_{t+1} podle cílového Q-hodnotového modelu.
- $Q_\theta(s_t, a_t)$: Očekávaná hodnota Q-funkce pro akci a_t ve stavu s_t podle aktuálních parametrů modelu θ.
- $(r_t + \gamma \cdot \max_{a'} Q_{\bar{\theta}}(s_{t+1}, a') - Q_\theta(s_t, a_t))^2$: Čtverec rozdílu mezi aktuální Q-hodnotou a cílovou Q-hodnotou.

[7] Průměr přes všechny možné budoucí stavy a akce, kde s_{t+1} představuje budoucí stav a a' představuje budoucí akci.

Strategie pro stabilizaci učení v DQN

Pro stabilizaci učení v hluboké Q-síti (DQN) se využívají dvě klíčové strategie: cílová síť a paměťový buffer zážitků. Cílová síť minimalizuje fluktuace v aktualizacích vah hlavní sítě prostřednictvím aktualizace parametrů $\bar{\theta}$ plynulým průměrem. Cílová síť slouží jako cíl pro odhad budoucích Q-hodnot, což podporuje stabilní učení tím, že minimalizuje fluktuace v aktualizacích vah hlavní sítě a předejde oscilacím nebo divergencím během trénování.

Aktualizace vah hlavní sítě

Aktualizace vah neuronů hlavní sítě probíhá podle gradientního sestupu:

$$\theta \leftarrow \theta - \alpha \cdot \nabla_\theta L(\theta), \tag{1.4}$$

kde α je rychlost učení. Celkově použití dvou sítí umožňuje stabilnější a efektivnější učení v rámci algoritmu DQN.

Cílová síť v DQN

Cílová síť minimalizuje fluktuace v aktualizacích vah hlavní sítě pomocí aktualizace parametrů $\bar{\theta}$ plynulým průměrem:

$$\bar{\theta} \leftarrow \bar{\theta} \cdot \beta + (1 - \beta) \cdot \theta, \tag{1.5}$$

kde β je malá konstanta. Cílová síť slouží jako cíl pro odhad budoucích Q-hodnot, což podporuje stabilní učení tím, že minimalizuje fluktuace v aktualizacích vah hlavní sítě a předejde oscilacím nebo divergencím během trénování.

Rozšíření DQN: Double DQN, Dueling DQN a EMDQN

Hluboká Q síť (DQN) vstoupila na scénu jako úspěšná metoda pro řešení úloh zpětnovazebního učení. Nicméně, s průběhem času se objevily varianty a rozšíření DQN, které se snaží překonat některé jeho omezení a vylepšit výkon. Následující tři rozšíření patří mezi ty nejznámější.

Double DQN (DDQN): Standardní DQN může často nadhodnocovat hodnoty akcí, což může vést k chybným rozhodnutím. Double DQN řeší tuto otázku tím, že používá dvě síťové struktury: jednu pro volbu akce a druhou pro hodnocení této akce. Tím se eliminuje přehodnocování hodnot a zlepšuje přesnost odhadu Q-hodnot.

Dueling DQN: Dueling DQN se zaměřuje na rozdělení odhadu hodnot Q-funkce na dvě části: jednu pro hodnotu stavu (V-stav) a druhou pro výhodu akce (A-akce). Tato struktura umožňuje síti efektivněji odhadovat hodnoty akcí pro různé stavy, což může výrazně zvýšit efektivitu učení.

Dueling Double DQN (DDDQN): V některých situacích se kombinují výhody obou přístupů a vytváří se Dueling Double DQN (DDDQN). Tato kombinace zkoumá, jak může být spojení duelingu a double strategií přínosné pro zlepšení stability učení a přesnosti odhadu hodnot akcí.

Epizodická paměťová DQN (EMDQN): EMDQN [19, 20] přináší do DQN koncept epizodické paměti. Místo tradičního paměťového bufferu, kde se ukládají náhodné vzorky z průběhu celého trénovacího průběhu, EMDQN ukládá vzorky pouze z jedné epizody. Tím se snaží lépe zachytit závislosti mezi stavy v rámci jedné epizody a umožnit efektivnější učení v situacích s dlouhými časovými závislostmi nebo vzory.

Tato rozšíření DQN představují jen malou část bohaté rodiny algoritmů pro hluboké Q-učení. Jejich použití závisí na povaze konkrétní úlohy a požadavcích učení. Experimentování s různými variantami může vést k nalezení nejefektivnější strategie pro konkrétní problém.

2 Praktická část

V této části se zaměříme na praktickou implementaci a optimalizaci našeho projektu programovatelných tankových robotů v Robocode.

2.1 Optimalizace komunikace tank – server

Původní implementace obsahovala TCP/IP připojení tanku v jazyce Java k serveru RobocodeRunner. Toto připojení zahrnovalo opakované volání Python skriptu, který vyhodnocoval stavy hry a vracel akce z Q-tabulky. Každé volání tohoto skriptu trvalo přibližně 10 ms, zatímco samotné Java roboty se vykonávaly pod desetinu milisekundy. Tento rozdíl byl způsoben opakovaným spouštěním interpretu Pythonu, který je pomalejší než kompilované jazyky, například Java.

Problém byl zdvojen načítáním knihoven, což dále zpomalilo proces. U knihovny TensorFlow to mohlo trvat až 3 sekundy. Celkově to vedlo k náročné situaci, kdy můj tank byl pozastaven po dobu přibližně 1,5 sekundy při startu arény a pokud se nepodařilo připojit do další sekundy, trénovací tank explodoval.

Pro řešení tohoto problému existuje několik možností. První variantou je mít model tanku v kompilovaném jazyku, jako jsou Java nebo C++. Tato metoda by byla efektivní, avšak v případě volání knihoven pro tvorbu neuronových sítí, jako jsou TensorFlow nebo PyTorch, je lepší variantou Python, pro který byly tyto varianty primárně vytvářeny a kromě velké komunity mají i větší integritu. Navíc i zde není zaručeno, že načítání knihovny TensorFlow nepřekročí povolenou mez. Další metody jsou buď použití Dockeru pro Python, komunikace přes sdílenou paměť nebo přes sokety na localhost s volným portem mezi Java a Python částí tanku. Zde je myšlenka taková, že se prvně spustí Java část tanku, která poté spustí Python část tanku, kde se všechno nastaví, importují se všechny potřebné knihovny a pak bude probíhat komunikace bez nežádoucího zpomalení.

2.2 Implementace řešení

2.2.1 Implementace optimalizace komunikace

Pro optimalizaci byla implementována socket komunikace uvnitř tanku, kde není potřeba alokovat sdílenou paměť. K tomu byl využit socket komunikace mezi Java a Python částí tanku. Pro zajištění komunikace na volném portu je volný port zjištěn v Java části a poté je přes vstupní argument poslán do Python části při jejím spuštění. Po navázání spojení může probíhat komunikace.

Během měření bez modelu byla naměřena průměrná doba komunikace (tam a zpět) okolo desetiny milisekundy. Tímto bylo dosaženo více než tisícinásobného zrychlení oproti opakovanému načítání knihovny Tensorflow a jemu podobným.

Představme si jednoduchý příklad soketové komunikace mezi Java skriptem (Server viz 2.2.1) a Python skriptem (Klient viz 2.2.1) v kontextu programovatelných tankových robotů.

```
public class Server {
    public static void main(String[] args) {
        try {
            ServerSocket serverSocket = new ServerSocket(12345);
            serverSocket.setSoTimeout(20000); // 20s
            Socket clientSocket = serverSocket.accept();
            serverSocket.setSoTimeout(500); // 0.5s

            InputStream inputStream = clientSocket.getInputStream();
            OutputStream outputStream = clientSocket.getOutputStream();

            int data = 0;
            for (int i = 0; i < 1000; i++) {
                outputStream.write((data + "\n").getBytes()); // Out

                BufferedReader in = new BufferedReader(new
                    InputStreamReader(inputStream)); // In
                int receivedData = Integer.parseInt(in.readLine());
                System.out.println("Java received: " + receivedData);
                data = receivedData + 1;
            }
            serverSocket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Výpis 2.1: Server v jazyce Java

```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(("localhost", 12345))

for i in range(1000):
    data = s.recv(1024).decode()
    print(f"Python received: {data}")
    data = int(data) + 1
    s.send((str(data)+"\n").encode())

s.close()
```

Výpis 2.2: Klient v jazyce Python

Tato ukázka názorně ilustruje princip server-klient komunikace, kde Java skript funguje jako server a Python skript jako klient. Server vytvoří soket na portu 12345 a čeká 20 sekund na připojení klienta. Po úspěšném připojení se doba vypršení sníží na 0.5 sekundy a zahájí se vzájemná komunikace. Server začíná posíláním nuly klientovi, který ji přijme, zvýší o 1 a pošle zpět. Podobně poté server přijme číslo, zvýší ho o 1 a opět jej odešle zpět. Tento proces se opakuje celkem 1000krát, ilustrujíc tak stabilní a opakovaný tok komunikace mezi oběma skripty.

Můj kód pro tank pracuje na podobném principu. Server si zjistí volný port a vytvoří klienta, kterému přes argument předá volný port. Po připojení server posílá stavy hry do klienta. Klient tyto stavy přijme, skrze neuronovou síť získá akce a pošle je zpět na server. Velikost stavů je upravena pro neuronovou síť. Aktuálně na 10 tanků a 20 střel. Touto optimalizací se podařilo dosáhnout až 1000 TPS (Tick per second). Server také ukládá stavy a akce jako epizodickou paměť do souboru ModelGameData.txt pro EMDQN viz níže 2.2.2.

2.2.2 Implementace algoritmu zpětnovazebního učení

Pro výběr správného algoritmu zpětnovazebního učení je nutné zjistit typ a velikost stavů a akcí. V našem případě máme diskrétní akce a stavový prostor závisí na vlastnostech tanků a střel. Každý tank má 8 parametrů a každá střela 4 parametry typu double $D = 2^{64}$. A počet střel závisí primárně na počtu tanků $m = k \cdot n, k \in N$. Složitost tohoto stavového prostoru je následovná:

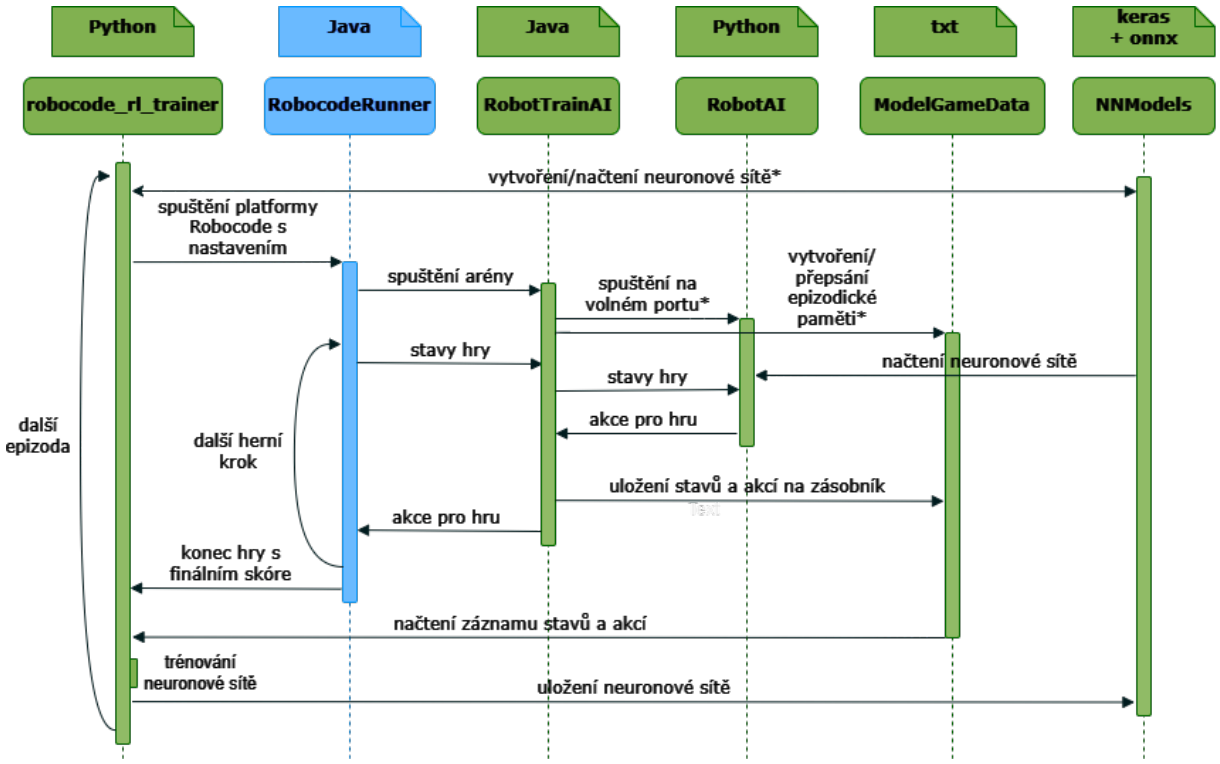
$$\mathcal{O}\left((D^8)^n \cdot (D^4)^m\right) = \mathcal{O}\left(D^{4(2n+m)}\right) = \mathcal{O}\left(2^{n \cdot 256(2+k)}\right) = \mathcal{O}\left(2^{n \cdot \alpha}\right) = \mathcal{O}\left(2^{p(n)}\right), \alpha \in N$$

Jak můžeme vidět, tak složitost spadá do kategorie EXPSPACE, kde p představuje polynom proměnné n. Z tohoto důvodu je efefktivnější použít algoritmus pro neomezený stavový prostor jako DQN.

V rámci platformy Robocode se po skončení epizody (několik kol, dle nastavení) vytvoří finální skóre. Pro tyto potřeby byl nakonec zvolen algoritmus EMDQN s epizodickou pamětí. Tento algoritmus je implementován v robocode_rl_trainer.py a obsahuje menší uživatelské rozhraní v terminálu, kde ptá se na počet epizod u učení. Jiné parametry je potřeba změnit přímo v kódu jako nastavení pro platformu Robocode apod.

V rámci sekvenčního diagramu (viz obr. 2.1) můžeme vidět zeleně nově vytvořené části a modře částečně modifikovanou část RobocodeRunner, která spouští arénu.

Pro další optimalizaci používám kombinaci knihoven TensorFlow a ONNX, kdy pro načtení neuronové sítě v tanku používám ONNX, u které načtení knihovny bývá i 3krát rychlejší.



Obr. 2.1: Sekvenční diagram projektu

Návrh a implementace neuronové sítě

Výhodou neuronové sítě je, že vstupem mohou být vektory, což celý stavový problém zlogaritmuje. Počet tanků byl stanoven na 10 a počet střel na 20, ostatní tanky nebo střely jsou ignorovány.

$$S = D^{4(2n+m)} = D^{4(2\cdot 10+20)} = D^{160} \rightarrow S_{nn} = 160$$

Velikost vstupní vrstvy neuronové sítě byl nataven na 160. Velikost výstupní vrstvy je nastavena na 4, kdy může nabývat i záporných hodnot, proto má lineární aktivační funkci LINEAR. Skrytých vrstev je 4 a všechny mají velikost 1024 a aktivační funkci RELU.

2.2.3 Trénování a testování

Pro trénování bylo zvažováno několik metod jako epsilon-greedy strategie nebo možnost uživatelského prostředí. Nakonec byla vybrána metoda učení od nejlepšího tanku. Pro výběr nejlepšího tanku bylo vytvořeno několik arén. Každá z nich měla 10 000 kol. Jak můžeme vidět v tabulce 2.1, tak byl prvně zápas všichni proti všem, poté bylo vybráno 10 nejlepších a nakonec 5 nejlepších. Jak můžeme vidět, tak nejlepé si vedl tank Walls, který se snaží držet u stěny, nicméně pro trénování byl použit nejlepší pokročilý robot[1] SpinBot, který byl pro potřeby posílání stavů modifikován. Jeho taktika je primárně točení a střílení.

Dále byly vytvořeny 2 modely, které byly trénovány a testovány viz tabulky 2.2 a 2.3. Oba modely používají optimizér „adam“ a metriku „přesnost“. První má ztrátovou funkci MSE (střední kvadratická chyba) a druhý má MAE (střední absolutní chyba). Jak můžeme vidět, tak první model dosahuje téměř 2krát lepších výsledků než druhý model s MAE. U obou modelů pro trénování bylo použito 200 epizod a pro testování 100 epizod. Každá epizoda byla nastavena na 10 kol.

[1] R (Robot) robot, AR (AdvancedRobot) pokročilý robot

Tab. 2.1: Přehled výsledků a statistik nejlepších robotů v souboji

Pořadí	Název a skóre tanku	Název a skóre tanku z 10 nejlepších	Název a skóre tanku z 5 nejlepších	Typ tanku
1.	Walls | 10 460 994	Walls | 6 790 597	Walls | 3 658 982	R
2.	SpinBot | 9 381 288	SpinBot | 5 009 592	SpinBot | 2 181 133	AR
3.	Tracker | 7 558 361	Tracker | 3 931 535	Tracker | 1 630 302	R
4.	TrackFire | 6 564 960	Crazy | 3 184 889	TrackFire | 1 227 525	R
5.	RamFire | 5 731 362	TrackFire | 3 174 844	Crazy | 1 226 218	AR
6.	Crazy | 5 647 309	VelociRobot | 3 032 245		
7.	Fire | 5 416 632	RamFire | 2 970 407		
8.	MyFirstJuniorRobot | 5 070 921	Fire | 2 494 898		
9.	VelociRobot | 4 743 023	MyFirstJuniorRobot | 2 464 209		
10.	PaintingRobot | 4 738 179	PaintingRobot | 2 248 915		
11.	Corners | 4 592 922			
12.	MyFirstRobot | 3 415 619			
13.	Target | 2 928 196			
14.	Interactive | 2 314 150			
15.	SittingDuck | 2 312 700			
16.	Interactive_v2 | 2 302 200			

Tab. 2.2: Přehled trénování a testování modelu s MSE

Pořadí	Název a skóre tanku, trénování	Název a skóre tanku, testování
1.	Walls | 731 894	Walls | 378 538
2.	SpinBot | 434 462	Tracker | 186 285
3.	Tracker | 327 189	Crazy | 136 212
4.	Crazy | 243 861	TrackFire | 174 196
5.	TrackFire | 245 581	RobotTrainAI | 88 107

Tab. 2.3: Přehled trénování a testování modelu s MAE

Pořadí	Název a skóre tanku, trénování	Název a skóre tanku, testování
1.	Walls | 733 587	Walls | 387 501
2.	TrackFire | 250 908	TrackFire | 212 307
3.	SpinBot | 436 095	Tracker | 207 539
4.	Tracker | 319 473	Crazy | 148 509
5.	Crazy | 243 777	RobotTrainAI | 47 013

Závěr

V rámci semestrální práce byly provedeny úspěšné optimalizace komunikace a následně natrénovány dva modely tanků. Pro trénování modelů byl vyvinut skript robocode_rl_trainer.py, který umožňuje trénování modelů. Dále byl vytvořen samotný model RobotTrainAI.java, RobotAI.py a modifikace robota SpinBot, která umožňuje ukládání stavů a akcí do epizodické paměti v textovém souboru ModelGameData.txt. Taktéž byly provedeny úpravy skriptu RobocodeRunner.java, aby umožňoval různá nastavení prostřednictvím konfiguračních souborů ze složky config.

Veškerý kód a vývojová dokumentace jsou dostupné v repozitáři na platformě GitHub:
https://github.com/wenca611/Diplomova-Prace. K dispozici jsou zde také dva natrénované modely, které však nebyly začleněny do této práce z důvodu omezeného rozsahu příloh.

Porovnáním modelů s různými ztrátovými funkcemi bylo zjištěno, že tank s použitím funkce kvadratické chyby střední hodnoty (MSE) dosahuje téměř dvojnásobného skóre oproti tanku s funkcí střední absolutní chyby (MAE).

Do budoucna je plánována explorace různých modifikací modelů tanků za účelem dosažení optimálního řízení tankových robotů. V plánu je také vytvoření skriptu pro vizualizaci dat a rozšíření skriptu pro trénování o možnost výběru neuronové sítě na základě vytvořené přezdívky.

Literatura

[1] PEŇÁZ, Vladimír. *Robocode - zabezpečená platforma pro hodnocení studentských projektů*. Brno, 2023. [online]. Dostupné také z: https://www.vut.cz/www_base/zav_prace_soubor_verejne.php?file_id=255587. Bakalářská práce. Vysoké učení technické v Brně. Vedoucí práce Doc. Ing. Radim Burget, Ph.D.

[2] *Robocode* [online]. [21. století] [cit. 2023-10-29]. Dostupné z: https://robocode.sourceforge.io/

[3] *Robowiki: Robocode* [online]. [21. století], naposledy upraveno 20. října 2022 [cit. 2023-10-30]. Dostupné z: https://robowiki.net/wiki/Robocode. Licence: CC BY-SA 4.0.

[4] *Battlefield Measurements* [online]. [21. století] [cit. 2023-11-01]. Dostupné z: https://mark.random-article.com/weber/java/robocode/lesson2.html

[5] *Tank Anatomy* [online]. [21. století] [cit. 2023-11-01]. Dostupné z: https://mark.random-article.com/weber/java/robocode/lesson2.html

[6] LARSEN, Flemming N. *ReadMe for Robocode* [online]. [21. století], 29.7.2021 [cit. 2023-11-05]. Dostupné z: https://robocode.sourceforge.io/docs/ReadMe.html

[7] GADE, Morten, Michael KNUDSEN, Rasmus ASLAK KJÆR, Thomas CHRISTENSEN, Christian PLANCK LARSEN, Michael David PEDERSEN a Jens Kristian SØGAARD ANDERSEN. *Applying Machine Learning to Robocode* [online]. Aalborg, 2003, 148 s. [cit. 2023-11-05]. Dostupné z: https://www.dinbedstemedarbejder.dk/Dat3.pdf. DAT3 projekt. Aalborg University. Vedoucí práce Søren Holbech Nielsen.

[8] *Robowiki: FAQ* [online]. 2007, 25.10.2020 [cit. 2023-11-01]. Dostupné z: https://robowiki.net/wiki/Robocode/FAQ. Licence: CC BY-SA 4.0.

[9] What is machine learning?: Machine learning creates algorithms that get better the more data they work with. In: ISAZI CONSULTING. *What is machine learning?* [online]. c2015 [cit. 2023-12-03]. Dostupné z: https://www.isaziconsulting.co.za/machinelearning.html.

[10] KELLEHER, John D. *Deep learning.* Illustrated. Cambridge, Massachusetts: The MIT Press, [2019]. The MIT Press essential knowledge series. ISBN 978-026-2537-551.

[11] MEGAJUICE. Reinforcement learning diagram. In: WIKIPEDIA. *Reinforcement learning diagram* [online]. 2017 [cit. 2023-12-03]. Dostupné z: https://upload.wikimedia.org/wikipedia/commons/thumb/1/1b/Reinforcement_learning_diagram.svg/2119px-Reinforcement_learning_diagram.svg.png.

[12] MNIH, Volodymyr, Koray KAVUKCUOGLU, David SILVER, Alex GRAVES, Ioannis ANTONOGLOU, Daan WIERSTRA a Martin RIEDMILLER. *Playing Atari with Deep Reinforcement Learning* [online]. Toronto, Ontario, 2013, 9 s. [cit. 2023-12-03]. Dostupné z: https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf. NIPS Deep Learning Workshop. University of Toronto.

[13] ALMAHAMID, Fadi a GROLINGER, Katarina. *Reinforcement Learning Algorithms: An Overview and Classification* [online]. Publikace z oblasti elektroniky a informatiky. 1151 Richmond Street, London, Ontario, Canada: University of Western Ontario, září 2021. Dostupné z: https://ir.lib.uwo.ca/cgi/viewcontent.cgi?article=1559&context=electricalpub. [cit. 2023-12-02].

[14] SUTTON, Richard S. a Andrew G. BARTO. Unified Notation for Episodic and Continuing Tasks. In: *Reinforcement Learning: An Introduction.* Second edition. Cambridge, Massachusetts: The MIT Press, [2018], s. 79-80. Adaptive Computation and Machine Learning (The MIT Press). ISBN 9780262039246.

[15] SUTTON, Richard S. a Andrew G. BARTO. Q-learning: Off-policy TD Control. In: *Reinforcement Learning: An Introduction.* Second edition. Cambridge, Massachusetts: The MIT Press, [2018], s. 153-154. Adaptive Computation and Machine Learning (The MIT Press). ISBN 9780262039246.

[16] SUTTON, Richard S. a Andrew G. BARTO. Sarsa: On-policy TD Control. In: *Reinforcement Learning: An Introduction.* Second edition. Cambridge, Massachusetts: The MIT Press, [2018], s. 151-152. Adaptive Computation and Machine Learning (The MIT Press). ISBN 9780262039246.

[17] KARUNAKARAN, Dhanoop. Applying Neural Network as a functional approximation in Q-learning. A MEDIUM CORPORATION. *Deep Q Network(DQN)* [online]. Sep 21, 2020 [cit. 2023-12-07]. Dostupné z: https://medium.com/intro-to-artificial-intelligence/deep-q-network-dqn-applying-neural-network-as-a-functional-approximation-in-

[18] CHOUDHARY, Ankit. Deep Q-Networks. *A Hands-On Introduction to Deep Q-Learning using OpenAI Gym in Python* [online]. c2013-2023, August 21st, 2023 [cit. 2023-12-07]. Dostupné z: https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/

[19] LIN, Zichuan, Tianqi ZHAO, Guangwen YANG a Lintao ZHANG. TSINGHUA UNIVERSITY, MICROSOFT, MICROSOFT RESEARCH. Deep Q-Networks, Episodic Memory Deep Q-Networks. *Episodic Memory Deep Q-Networks* [online]. July 2018 [cit. 2023-12-07]. Dostupné z: `https://www.researchgate.net/publication/326202263_Episodic_Memory_Deep_Q-Networks`

[20] GEEKSFORGEEKS. Episodic Memory. *Episodic Memory and Deep Q-Networks* [online]. 2021, 20 May, 2021 [cit. 2023-12-07]. Dostupné z: `https://www.geeksforgeeks.org/episodic-memory-and-deep-q-networks/`

Seznam symbolů a zkratek

DDDQN	Dvojitá zdvojená hluboká Q síť – Double Dueling Deep Q Network
DDQN	Dvojitá hluboká Q síť – Double Deep Q Network
DNN	Hluboká neuronová síť – Deep Neural Network
DQN	Hluboká Q síť – Deep Q Network
EMDQN	Hluboká Q síť s epizodickou pamětí – Episodic Memory Deep Q Network
FAQ	Často kladené otázky – Frequently Asked Questions
GUI	Grafické uživatelské rozhraní – Graphical User Interface
IDE	Vývojové prostředí – Integrated Development Environment
JDK	Java vývojový balík – Java Development Kit
MAE	Střední absolutní chyba – Mean absolute error
ML	Strojové učení – Machine Learning
MSE	Střední kvadratická chyba – Mean squared error
RL	Zpětnovazební učení – Reinforcement Learning
SARSA	Stav-Akce-Odměna-Stav-Akce – State-Action-Reward-State-Action
SL	Supervizované učení – Supervised Learning
STDERR	Standardní chybový výstup – Standard Error
STDIN	Standardní vstup – Standard Input
STDOUT	Standardní výstup – Standard Output
TCP/IP	Přenosový kontrolní protokol/Internetový protokol – Transmission Control Protocol/Internet Protocol
TPS	Počet herních tiků za vteřinu – Tick per second
TD	Učení s časovým rozdílem – Temporal-Difference learning
UL	Nesupervizované učení – Unsupervised Learning
VPN	Virtuální privátní síť – Virtual Private Network

Seznam příloh

A Volání programu

Je potřeba mít nainstalovanou Javu ve verzi IBM JDK 1.8 a Python ve verzi 3.10 a vyšší. Dále je doporučeno používat PyCharm a IntelliJ, avšak není to podmínku.

Hra se dá spouštět ze 2 různých míst. V případě spuštění z RobocodeRunner.java v Java se spustí Robocode spouštěč. Pro variantu s TCP/IP připojení tanků je potřeba nastavit v configu jednotlivé ip adresy tanků, a pak je potřeba prvně zapnout tanky Student1.java a pak server RobocodeRunner.java. Takhle jde kombinovat studentské tanky s autonomními roboty.

Další možnost je spuštění robocode_rl_trainer.py v python, který slouží pro trénování robota. Před spuštěním je ale potřeba zjistit si z IntelliJ nebo Eclipse příkaz, který spouští celou platformu Robocode a opravit jej v kódu. Pozor, pokud se změní něco v kódu RobocodeRunner.java, tak se to neprojeví při spouštění z Pythonu, dokud se to znovu nespustí v IDE. Kvůli tomu byly některé nastavění vyndány z RobocodeRunner.java bokem do configu, aby se to daly přenastavit v Pythonu.

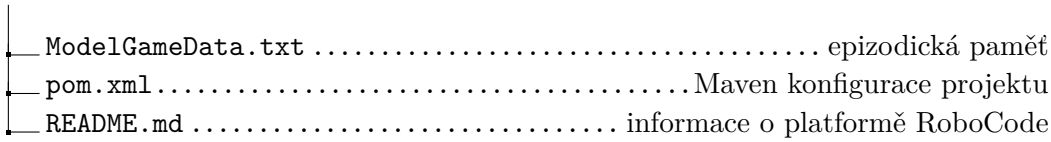
Celkem se dá nastavit 34 proměnných. Ty nejdůležitější jsou nastavění přidání tanků (i podle indexu), velikosti mapy, počtu kol, zapnutí/vypnutí GUI, nastavení zvuku, explozí apod.

B Obsah elektronické přílohy

Pro spuštění zpětnovazebního trénování slouží program robocode_data_analyzer.py. Pokud chybí některé knihovny, tak stačí spustit install_requirements.py, který stáhne všechny knihovny z requirements.txt.

- `/` ..... kořenový adresář: RoboCodeProject
  - `doc` ..... text semestrální práce
    - `xpastu02_semestralni_prace.pdf`
  - `RoboCode` ..... platforma RoboCode
    - `obsah složky viz níže`
  - `.gitignore` ..... odstranění nadbytečných souborů
  - `install_requirements.py` ..... skript pro instalaci knihoven z requirements.txt
  - `README.md` ..... informace o projektu
  - `requirements.txt` ..... instalační požadavky
  - `robocode_data_analyzer.py` ..... skript pro analýzu dat
  - `robocode_rl_trainer.py` ..... skript pro výcvik RL

- `RoboCode` — platforma RoboCode
  - `config` — konfigurační data
    - `compiler.properties` — konfigurace kompilace
    - `game.properties` — konfigurace hry
    - `robocode.properties` — konfigurace platformy
    - `window.properties` — konfigurace velikosti oken
  - `libraries` — knihovny pro hru RoboCode
    - ...
  - `manualImages` — manuální obrázky
    - ...
  - `NNModels` — modely neuronových sítí
    - ...
  - `robots` — roboti
    - `sample` — třídy robotů
    - ...
  - `src` — zdrojové soubory projektu
    - `cz/vutbr/feec/robocode`
      - `battle`
        - `BattleObserver.java` — bojový pozorovatel
        - `RobocodeRunner.java` — spuštění platformy Robocode (server)
        - `Student1.java` — studentský tank1 (klient)
        - ...
      - `data` — zpracovaní dat
        - `ProcessedBattleData.java` — zpracovaná data zápasu
        - `ProcessedBulletData.java` — zpracovaná data střely
        - `ProcessedTankData.java` — zpracovaná data tanku
      - `protocol` — definice paketů
        - `RobocodePacket.java` — paket
        - `RobocodeRequest.java` — požadavek
        - `RobocodeResponse.java` — odpověď
      - `socket` — implementace serveru a klienta
        - `AgentCommunicator.java` — komunikace mezi agentem a serverem
        - `SocketsHolder.java` — správa soketů
        - `TCPClientSocket.java` — implementace klienta soketu
        - `TCPServerSocket.java` — implementace serveru soketu
      - `studentRobot` — studenští roboti
        - `RobotAI.py` — robot s neuronovou sítí pro EMDQN
        - ...
    - `sample` — autonomní roboti
      - `RobotClient.java` — klientský tank
      - `RobotTrainAI.java` — trénovací tank v pythonu
      - ...
  - `test/cz/vutbr/feec/robocode/protocol` — testování TCP/IP protokolu
    - `TestRequest.java` — testování požadavku
    - `TestResponse.java` — testování odpovědi
  - `.gitignore` — odstranění nadbytečných souborů


ModelGameData.txt ..... epizodická paměť
pom.xml ..... Maven konfigurace projektu
README.md ..... informace o platformě RoboCode