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

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — Bc. Václav Pastušek

AUTHOR

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

SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová 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: 5.2.2024 ***Termín odevzdání:*** 21.5.2024

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

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

ABSTRAKT

Tato diplomová práce se zabývá optimalizací řízení tankového robota v prostředí Robocode za využití zpětnovazebního učení. Komplexita tohoto problému spadá do třídy EXPSPACE, což představuje výzvu, kterou nelze podcenit. Teoretická část práce pečlivě zkoumá platformu Robocode, koncepty zpětnovazebního učení a příslušné algoritmy, zatímco praktická část se zaměřuje na optimalizaci agenta, implementaci zpětnovazebních algoritmů a vytvoření uživatelsky přívětivého rozhraní pro snadné trénování a testování modelů. V rámci práce bylo natrénováno a otestováno celkem 64 modelů, jejichž data a parametry jsou vzájemně srovnávány a prezentovány v přiložených databázích a grafech. Nejlepší výsledky v průměrném počtu zásahů na epizodu dosáhly modely s označením *v0.8.0* a *v1.0.0*. U prvního z nich se projevila určitá schopnost vyhýbání se střelám, zatímco u druhého byly pozorovány úspěšnější zásahy.

KLÍČOVÁ SLOVA

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

ABSTRACT

This master's thesis focuses on optimizing the control of a tank robot in the Robocode environment using reinforcement learning. The complexity of this problem falls into the EXPSPACE class, presenting a challenge that cannot be underestimated. The theoretical part of the thesis meticulously examines the Robocode platform, concepts of reinforcement learning, and relevant algorithms, while the practical part focuses on optimizing the agent, implementing reinforcement learning algorithms, and creating a user-friendly interface for easy training and testing of models. A total of 64 models were trained and tested as part of the thesis, with their data and parameters compared and presented in accompanying databases and graphs. The best results in terms of average hits per episode were achieved by models labeled *v0.8.0* and *v1.0.0*. The first model exhibited a certain ability to evade shots, while the second model showed more successful hits.

KEYWORDS

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

PASTUŠEK, Václav. *Optimalizace řízení s pomocí zpětnovazební učení na platformě Robocode*. Diplomová 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:	Diplomová 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 a návrh částí kódů a komentářů, konkrétně ChatGPT od společnosti OpenAI ve verzi 3.5. Veškeré výstupy z tohoto nástroje byly pečlivě manuálně ověřeny a upraveny v souladu s potřebami této práce. Prohlašuji, že za výslednou podobu a spolehlivost přebírám veškerou zodpovědnost. Využití těchto nástrojů je v souladu s pravidly Vysokého učení technického v Brně, která jsou k dispozici na stránce: `https://www.vut.cz/uredni-deska/ai/vzdelavani`.

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 práce je ukázat, že i v komplexních úlohách lze nalézt smysluplný výzkum a poskytnout studentům příležitost k praktickému zapojení do této dynamické a perspektivní oblasti. Je však důležité zdůraznit, že řešení této problematiky není jednoduché, protože co se týče počtu stavů a akcí, problém spadá do třídy EXPSPACE.

Diplomová práce je rozdělena na teoretickou, praktickou část, výsledky s diskuzí a závěr. 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 se zabývá třídou složitosti algoritmů a zkoumá, do kterých složitostních kategorií spadají algoritmy zpětnovazebního učení, jako je například

hluboká Q síť. Třetí podkapitola teoretické části se pak zaměřuje na analýzu metod zpětnovazebního učení a jejich možné implementace v prostředí Robocode.

Praktická část této práce je zaměřena na optimalizaci virtuální arény a agenta. Pro trénování byla zvolena hluboká Q-síť (DQN) s epizodickou 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. Pro usnadnění manipulace s modely. jejich tréninku, testování a úprav vylo vytvořeno uživatelské rozhraní spolu s ukázkou databáze modelů a generování grafů.

V rámci práce bylo natrénováno a otestováno celkem 64 modelů, které představují různá nastavení hyperparametrů DQN s cílem optimalizovat řízení tanků. Tyto modely postupně akumulují zkušenosti, zdokonalují své schopnosti v reakci na nepřátele, zaměřování na cíle a střílení s co nejlepšími výsledky. Data a parametry těchto modelů jsou důkladně porovnány a prezentovány v přiložených databázích a grafech. Nejlepší výsledky v průměrném počtu zásahů na epizodu dosáhly modely s označením *v0.8.0* a *v1.0.0*. U prvního z nich se projevila určitá schopnost vyhýbání se střelám, zatímco u druhého byly pozorovány úspěšnější zásahy.

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]

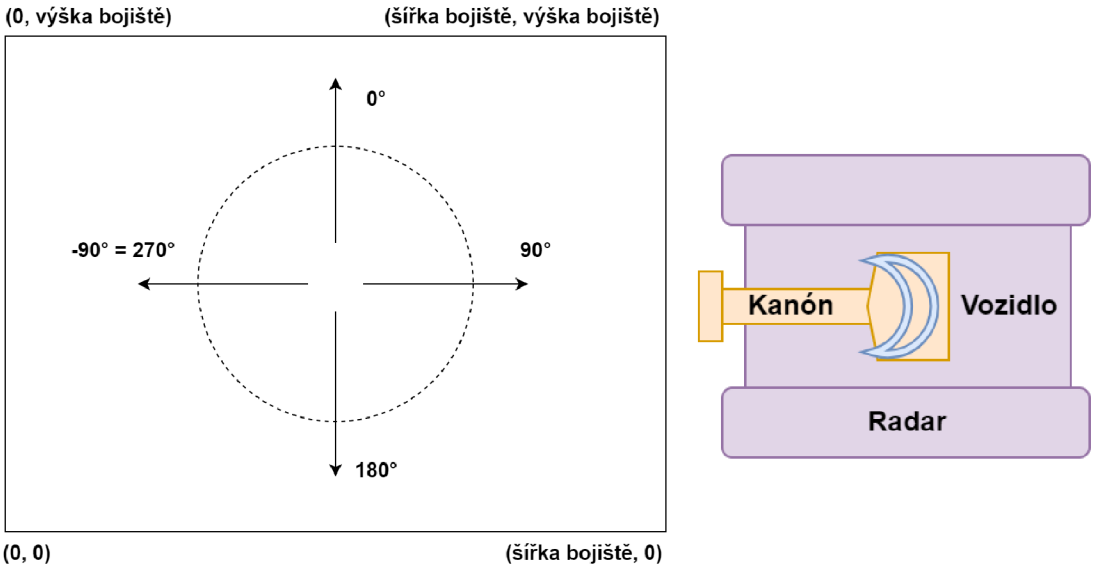
Hlavním záměrem hry je poskytnout zábavný způsob, jak se naučit programovací jazyk Java. Robocode je kompatibilní s různými operačními systémy, jako jsou Windows, MacOS a Linux, díky své platformě napsané v Javě. [3]

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).

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 počkat na ochlazení děla. [7, 8]



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

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

[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`

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].

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.

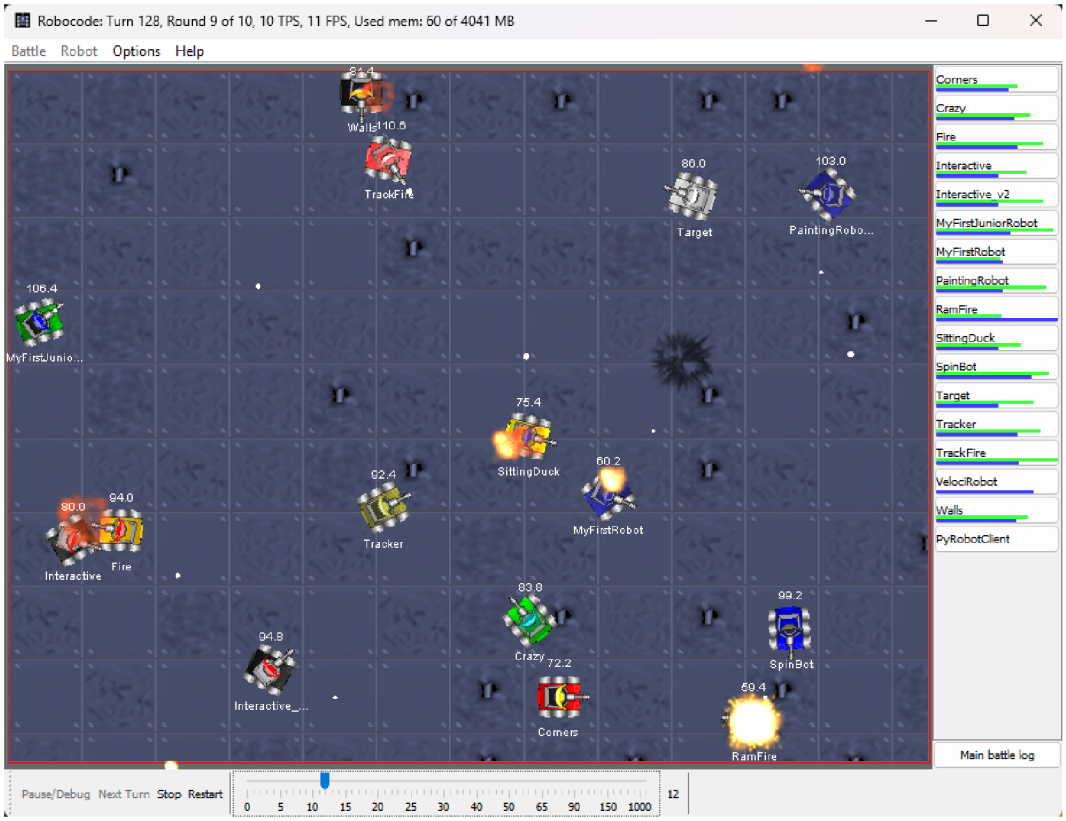
1.1.2 Modifikovaná verze platformy Robocode

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.

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

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).



Obr. 1.2: GUI platformy Robocode

1.2 Třída složitosti algoritmů

Když hovoříme o složitosti algoritmů, mluvíme o tom, jak se časová nebo prostorová náročnost algoritmu mění s velikostí vstupních dat. Složitostní třídy nám pomáhají klasifikovat algoritmy podle jejich náročnosti a umožňují nám porovnat různé algoritmy mezi sebou. Tyto třídy poskytují formální rámec pro analýzu výpočetní náročnosti algoritmů a problémů, což je klíčové pro návrh efektivních algoritmů a pochopení jejich limitací. [9, 10]

1.2.1 Základní složitostní třídy

Zde jsou některé základní složitostní třídy, které se běžně používají při analýze algoritmů. Tyto třídy nám pomáhají klasifikovat algoritmy podle jejich časové a prostorové náročnosti a umožňují nám porovnat různé algoritmy mezi sebou.

Deterministický prostor

Třída problémů, které lze vyřešit na deterministickém Turingově stroji[6] s využitím určitého množství prostoru:

- DSPACE: $\mathcal{O}(f(n))$
- L: $\mathcal{O}(log(n))$
- PSPACE: $\mathcal{O}(p(n))$[7]
- EXPSPACE: $\mathcal{O}(2^{p(n)})$

Nedeterministický prostor

Třída problémů, které lze vyřešit na nedeterministickém Turingově stroji s využitím určitého množství prostoru:

- NSPACE: $\mathcal{O}(f(n))$
- NL: $\mathcal{O}(log(n))$
- NPSPACE: $\mathcal{O}(p(n))$
- NEXPSPACE: $\mathcal{O}(2^{p(n)})$

[6] Turingův stroj je abstraktní matematický model výpočtu, který manipuluje s páskou pomocí hlavy, která může číst a zapisovat symboly. Tento model je základem teorie výpočtů a složitosti algoritmů.

[7] Kde $p(n)$ představuje polynom.

Deterministický čas

Třída problémů, které lze vyřešit na deterministickém Turingově stroji s využitím určitého množství času:

- DTIME: $\mathcal{O}(f(n))$
- P: $\mathcal{O}(p(n))$
- EXPTIME: $\mathcal{O}(2^{p(n)})$

Nedeterministický čas

Třída problémů, které lze vyřešit na nedeterministickém Turingově stroji s využitím určitého množství času:

- NTIME: $\mathcal{O}(f(n))$
- NP: $\mathcal{O}(log(n))$
- NEXPTIME: $\mathcal{O}(2^{p(n)})$

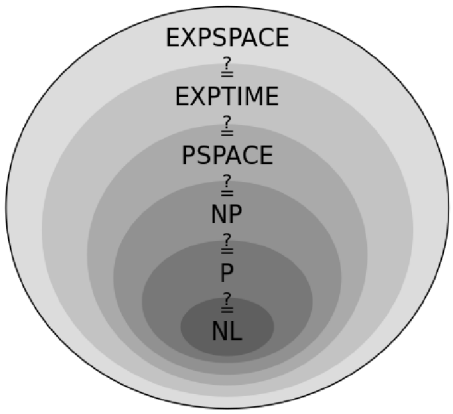
Problém P versus NP

Jedním z nejdůležitějších a nejznámějších otevřených problémů v informatice je otázka, zda třídy P a NP jsou stejné. P je třída problémů, které lze vyřešit na deterministickém Turingově stroji v polynomiálním čase, zatímco NP je třída problémů, které lze na nedeterministickém Turingově stroji ověřit v polynomiálním čase. Pokud by se prokázalo, že $P = NP$, znamenalo by to, že každý problém, jehož řešení lze rychle ověřit, lze také rychle vyřešit. Tento problém zůstává jednou z největších nevyřešených otázek teoretické informatiky.

Vztah mezi složitosními třídami

Vztah mezi složitostními třídami ukazuje, jak se jednotlivé třídy prostorově a časově náročných problémů vzájemně vztahují. Diagram na obrázku 1.3 zahrnuje problémy od těch nejjednodušších, řešitelných v logaritmickém prostoru, až po ty nejnáročnější, které vyžadují exponenciální množství prostoru. Tento formální rámec nám umožňuje lépe chápat výpočetní náročnost algoritmů a problémy, které se vyskytují v teoretické informatice.

$$L \subseteq NL \subseteq P \subseteq NP \subseteq PSPACE \subseteq EXPTIME$$
$$\subseteq NEXPTIME \subseteq EXPSPACE$$



Obr. 1.3: Diagram vztahů mezi složitostními třídami

Výzvy komplexity ve zpětnovazebním učení

Výzkum zaměřený na implementaci zpětnovazebního učení, jako je například DQN, se potýká s jednou zásadní výzvou: komplexitou stavového prostoru. Ten slouží jako vstupní data pro neuronovou síť a může být mimořádně rozsáhlý, často dosahující exponenciální složitosti (EXPSPACE) vzhledem k počtu možných stavů daného problému. Tento jev je patrný i při řešení bitvy v hře Robocode. Jasně vyplývá, že efektivní zpracování a aproximace hodnotových funkcí v takto vysokodimenzionálním prostoru vyžadují pokročilé algoritmy a optimalizace. Tato poznání jsou klíčová pro úspěšné řešení složitých problémů v reálném čase a s omezenými prostředky.

1.3 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. 1.4.

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.

Obr. 1.4: Diagram strojového učení [11]

Učení bez učitele – UL

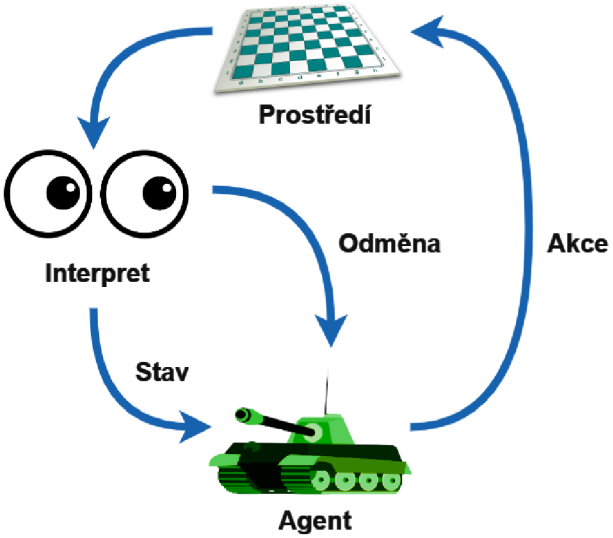
Učení bez učitele (unsupervised learning) se zaměřuje na shlukování dat bez přítomnosti cílových hodnot v datasetu. Algoritmus se snaží identifikovat vzory a struktury v datech, aby mohl seskupovat podobné příklady do shluků.

1.3.1 Zpětnovazební učení – RL

Zpětnovazební učení [12] (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. 1.5). Cílem agenta je získat vhodnou mapu, která spojuje jeho aktuální pozorování prostředí a vnitřní stav (paměť) s optimální akcí.

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



Obr. 1.5: Diagram zpětnovazebního učení [13]

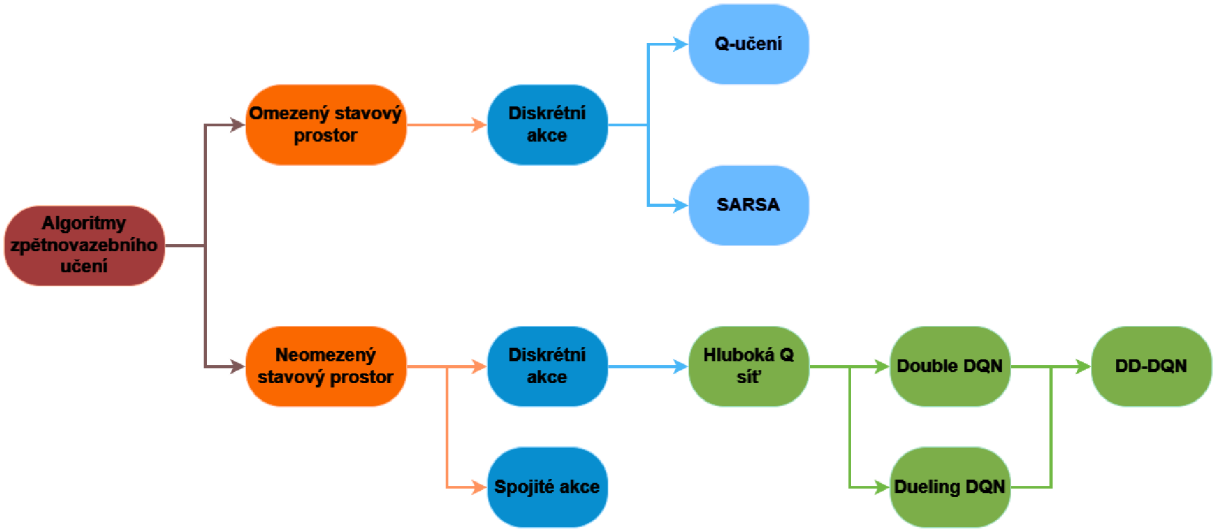
prostředí k učení, jak aktualizovat svou politiku.

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. [14]

1.4 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. 1.6)).



Obr. 1.6: Rozdělení zpětnovazebního učení [15]

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 [16] 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.

1.4.1 Q-učení

Q-učení [17] (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[8], 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:

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

[8] 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.

1.4.2 SARSA

Nyní, po seznámení s Q-learningem, přicházíme k dalšímu algoritmu posilovaného učení nazývanému SARSA [18]. 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}.

1.4.3 Hluboká Q síť – DQN

Hluboká Q síť [19, 20] (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:

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

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[9].
- 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.

[9] 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 [21, 22] 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 zaměřím na praktickou implementaci a optimalizaci projektu programovatelných tankových robotů v Robocode.

2.1 Optimalizace komunikace tank – aréna

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 se ještě zhoršil načítáním knihoven, což dále zpomalilo agenta. Jen samotné načtení knihovny TensorFlow trvá i 3 sekundy. Celkově to vedlo k náročné situaci, kdy byl můj tank pozastaven po dobu přibližně 1,5 sekundy při každém startu arény, a pokud se mu 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 dosažení optimalizace byla implementována socket komunikace uvnitř tanku, eliminující potřebu sdílené paměti. Tato komunikace probíhá mezi Java a Python částí tanku.

Nejprve je volný port zjištěn v Java části RobocodeRunner před spuštěním arény a je předán do Python části tanku jako vstupní argument. Tento port je také uložen

do souboru pro Java část tanku. Následně Python část tanku načte knihovny TensorFlow a Numpy. Po dokončení tohoto procesu se komunikace ukončí a aréna se spustí, čekajíc na druhou komunikaci od Java části tanku. Jakmile je spojení mezi částmi tanku navázáno, může probíhat komunikace a tank reaguje na stavy arény pomocí příslušných akcí.

Během měření bez modelu byla naměřena průměrná doba komunikace (tam a zpět) okolo desetiny milisekundy. Tento výsledek představuje více než tisícinásobné zrychlení ve srovnání s opakovaným načítáním knihovny TensorFlow a ostatních knihoven.

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

```python
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. Java část RobocodeRunner, která spouští arénu, funguje jako server, zatímco Python část tanku přebírá roli klienta. Po načtení všech potřebných knihoven se komunikace uzavře a Python část čeká na Java část tanku. Poté se role obrátí, a Python se stává serverem, zatímco Java část tanku funguje jako klient. Klient získává stavy z arény a posílá je serveru. Server tyto stavy přijme, použije je pro neuronovou síť k získání akcí a následně je zpětně posílá klientovi. Velikost stavů je upravena pro neuronovou síť, tedy 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 stavový prostor, který závisí na vlastnostech tanků a střel. Každý tank disponuje 8 parametry a každá střela 4 parametry typu double ($D = 2^{64}$). Počet střel m primárně závisí na počtu tanků n, což lze vyjádřit jako $m \leq k \cdot n; k \in \mathbf{N}$. Složitost tohoto stavového prostoru je následující:

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

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

Akce jsou spojité, typu double a mohou být provedeny současně. Proto je nezbytné tyto akce přizpůsobit pro použití s DQN, což vyžaduje jejich transformaci na sadu diskrétních akcí:

- První akce řídí pohyb tanku dopředu a dozadu. Pro tyto účely byly vybrány následující hodnoty: $-10, -5, 0, 5, 10$.
- Druhá akce řídí otočení tanku doleva a doprava ve stupních. Pro tyto účely byly vybrány následující hodnoty: $-10, -5, 0, 5, 10$.
- Třetí akce řídí sílu střely. Pro tyto účely byly vybrány následující hodnoty: $0, 1, 2, 3$.
- Poslední čtvrtá akce řídí pohyb děla doleva a doprava ve stupních. Pro tyto účely byly vybrány následující hodnoty: $-5, -1, 0, 1, 5$.

Pokud jsou všechny akce použity současně, vytvoří se $5 \cdot 5 \cdot 4 \cdot 5 = 500$ diskrétních akcí. Avšak tato kombinace je příliš rozsáhlá, a proto byla omezena na situace, kdy se tank buď pohybuje ve všech směrech, nebo střílí s točením kanónu. To vede k celkovému počtu $5 \cdot 5 + 4 \cdot 5 - 1 = 44$ diskrétních akcí. Odečtení jedné akce je z důvodu toho, že jedna diskrétní akce je stejná pro obě množiny akcí.

Převod spojitých akcí na diskrétní je pro pohyb zobrazen v tabulce 2.1 a pro střelbu a pohyb děla v tabulce 2.2.

Tab. 2.1: Převod spojitých akcí na diskrétní akce (pro pohyb)

Pohyb tanku	Otočení tanku	Síla tanku	Otočení kanónu	Diskrétní akce
-10	-10	0	0	1
-10	-5	0	0	2
-10	0	0	0	3
-10	5	0	0	4
-10	10	0	0	5
-5	-10	0	0	6
-5	-5	0	0	7
-5	0	0	0	8
-5	5	0	0	9
-5	10	0	0	10
0	-10	0	0	11
0	-5	0	0	12
0	0	0	0	13
0	5	0	0	14
0	10	0	0	15
5	-10	0	0	16
5	-5	0	0	17
5	0	0	0	18
5	5	0	0	19
5	10	0	0	20
10	-10	0	0	21
10	-5	0	0	22
10	0	0	0	23
10	5	0	0	24
10	10	0	0	25

Tab. 2.2: Převod spojitých akcí na diskrétní akce (pro střelbu)

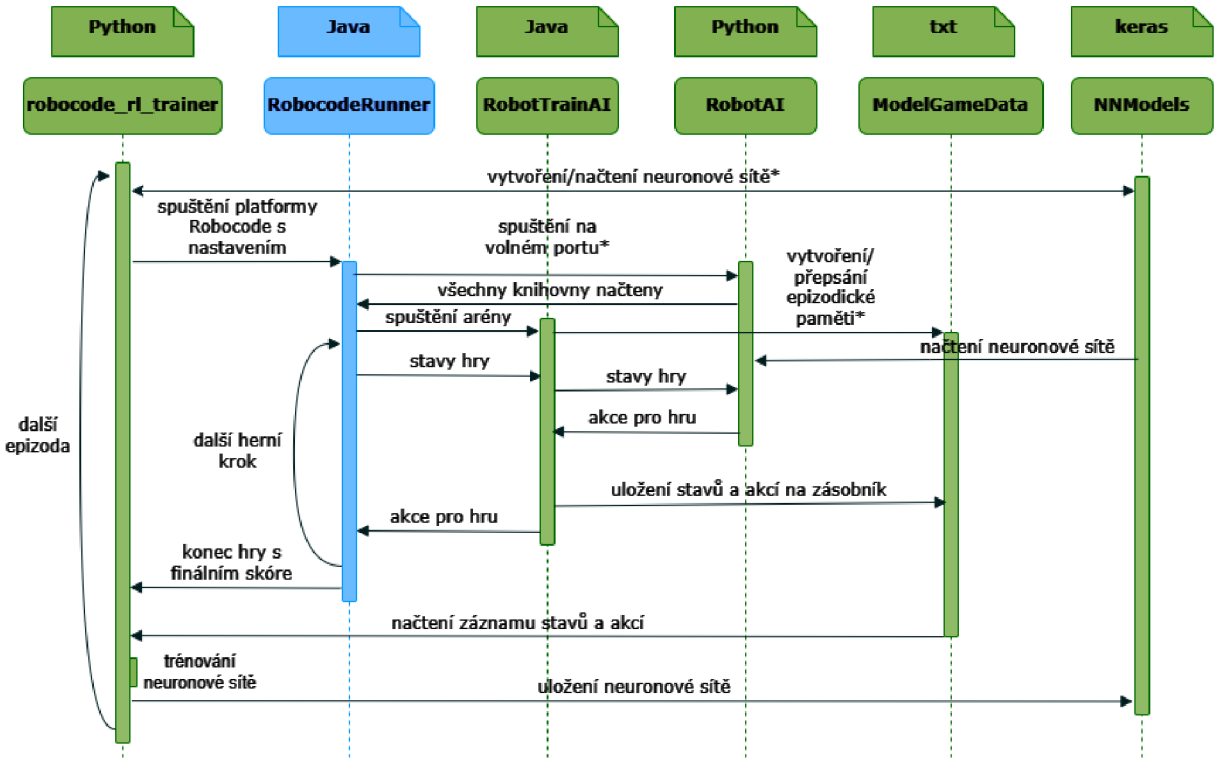
Pohyb tanku	Otočení tanku	Síla tanku	Otočení kanónu	Diskrétní akce
0	0	0	-5	26
0	0	0	-1	27
0	0	0	1	28
0	0	0	5	29
0	0	1	-5	30
0	0	1	-1	31
0	0	1	0	32
0	0	1	1	33
0	0	1	5	34
0	0	2	-5	35
0	0	2	-1	36
0	0	2	0	37
0	0	2	1	38
0	0	2	5	39
0	0	3	-5	40
0	0	3	-1	41
0	0	3	0	42
0	0	3	1	43
0	0	3	5	44

Každá epizoda spustí jedno kolo arény, během kterého se vykoná několik stovek stavů a akcí. Po skončení arény je vygenerováno finální skóre. Pro zpracování těchto stavů, akcí a výsledného skóre byl zvolen algoritmus EMDQN s epizodickou pamětí. Implementace tohoto algoritmu je obsažena v souboru *robocode_rl_trainer.py*.

Pro zvýšení přehlednosti jsem vytvořil sekvenční diagram, který ukazuje nově přidané části označené zeleně a částečně modifikovanou část RobocodeRunner v modré barvě (viz obr. 2.1).

První větší optimalizace byla dosažena přechodem z výpočtů na CPU na výpočty na GPU pomocí cuDNN. Díky tomu se doba trvání epizody zkrátila z několika minut na přibližně jednu minutu. Pro tuto konfiguraci je však nezbytné mít specifické verze všech potřebných komponent. Aktuální kombinace pro systém Windows[1] zahrnuje knihovnu tensorflow-gpu-2.10, Python 3.10, cuDNN 8.1 a CUDA 11.2. Podrobnější informace k nastavení trénování přes GPU lze nalézt na stránce: `https://www.tensorflow.org/install/source_windows`.

[1]Nejnovější verze jsou primárně vyvíjeny pro Linux.



Obr. 2.1: Sekvenční diagram projektu

Další optimalizace byla provedena v Pythonové části tanku, kde došlo ke změně metody predikce vstupu z model.predict() na pouhou invokaci model(). Zpočátku to vypadalo, že tato úprava přinesla až pětinásobné zrychlení, což vedlo k době trvání epizody na pouhých 8 sekund. Nicméně, později jsem zjistil, že tento rychlejší výkon byl pozorován pouze v Eager Execution mode, a po jeho vypnutí je o pár desetin milisekund rychlejší metoda model.predict().

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 byla nastavena na 160. Velikost výstupní vrstvy je stanovena na 44, přičemž každá diskrétní akce s největší číselnou hodnotou je vykonána. Kvůli tomuto chování je výstupní vrstva sítě definována jako lineární.

2.2.3 Implementace uživatelského rozhraní

Pro usnadnění procesu vytváření, konfigurace, trénování a testování neuronové sítě jsem vytvořil uživatelské rozhraní obsahující devět možností, zastupujících devět funkcí. V případě, že neexistuje žádný model neuronové sítě, jsou dostupné pouze možnosti pro vytvoření nové sítě, zobrazení nápovědy a ukončení programu. Po vytvoření prvního modelu jsou automaticky zpřístupněny všechny ostatní funkce. Možnosti výběru zahrnují:

- **Vytvoření neuronové sítě:** umožňuje vytvořit novou neuronovou síť pro trénování tanků s pomocí 12 nastavitelných parametrů, které jsou ukládány do souboru *_neural_network_parameters_data.csv*. Je důležité všechny parametry zadávat bez diakritiky. Uživatel má také možnost napsat „help“ pro vypsání nápovědy nebo „end“ pro ukončení programu.
- **Trénování neuronové sítě:** umožňuje trénovat již vytvořené neuronové sítě s možností nastavení počtu epizod, rychlosti arény a její viditelnosti. (Při nastavování počtu epizod je doporučeno zvolit menší hodnoty, aby se předešlo možnému pádu programu při trénování na GPU po přibližně 150 epizodách. Nicméně po opětovném spuštění programu není problém v trénování pokračovat). Pro dosažení maximální rychlosti arény lze nastavit hodnotu na -1. Po dokončení trénování jsou informace uloženy do souboru *_neural_network_training_data.csv* a jsou také vygenerovány grafy, které zobrazují energetickou úroveň tanků, získané skóre tanků, zásahy tanků, frekvenci akcí a ztrátovou hodnotu s přesností.
- **Testování neuronové sítě:** umožňuje provést testování výkonnosti již natrénovaných neuronových sítí. Nastavení této funkce je podobné trénování, s výjimkou zpětné vazby a grafu se ztrátovou hodnotou s přesností. Po dokončení testování jsou informace uloženy do souboru *_neural_network_testing_data.csv*.
- **Kopie neuronové sítě:** umožňuje vytvořit kopii existující neuronové sítě s možností ponechat původní váhy nebo vygenerovat nové.
- **Úprava neuronové sítě:** umožňuje úpravu parametrů vybrané neuronové sítě. Každý parametr je dostupný pomocí odpovídajícího indexu. S kombinací s funkcí kopie neuronové sítě, lze vytvářet nové modely bez nutnosti opakovaného zadávání všech parametrů.
- **Ukázka databáze neuronových sítí:** umožňuje zobrazit přehled databáze všech neuronových sítí v interaktivním prostředí PandasGUI. Uživatel zde může prozkoumat parametry sítí a informace o jejich trénování a testování, stejně jako si vytvořit vlastní grafy pro detailnější analýzu dat.

- **Vymazání neuronové sítě:** umožňuje odstranit existující neuronovou síť z databáze. Po provedení tohoto kroku je síť odstraněna ze všech záznamů, s výjimkou grafů.
- **Nápověda:** poskytuje uživateli informace a nápovědu k používání různých funkcí a možností rozhraní. Je možné ji volat i napsáním příkazu *help*.
- **Ukončení programu:** slouží k ukončení a uzavření aplikace. Je možné ji volat i napsáním příkazu *end*.

Parametry neuronové sítě

- **Název neuronové sítě:** nedá se nastavit, je generován automaticky podle aktuálního datumu a času a jedná se v databázi o primární klíč.
- **Přezdívka neuronové sítě:** může být libovolný text omezený na 50 znaků.
- **Vnitřní vrstva neuronové sítě:** definuje velikost neuronové sítě, typ aktivační vrstvy a hodnotu dropout. Příklad zápisu dvou skrytých vrstev: *[[64, relu, 0.01], [128, gelu, 0]]*. První závorka zde představuje první skrytou vrstvu s 64 neurony, s aktivační funkcí ReLU a dropoutem s hodnotou $0,01$. Druhá závorka zde představuje druhou skrytou vrstvu se 128 neurony, s aktivační funkcí GELU a dropoutem 0.
- **Zapnutí/vypnutí masky:** představuje zapnutí nebo vypnutí masky, která se aplikuje u mrtvých protivníků, aby po smrti nevytvářeli falešný dojem, že jsou stále v aréně.
- **Diskontní faktor (gama):** určuje, jak moc jsou preferovány okamžité odměny oproti odměnám získaným v budoucnosti.
- **Epsilon:** určuje pravděpodobnost, s jakou agent vybere náhodnou akci namísto akce, která byla vybrána na základě stávajících znalostí a odhadů hodnoty Q. Tato strategie je známá jako epsilon-greedy strategie. Epsilon má tří parametrů: počáteční hodnota, konečná hodnota a krok. Příklad zápisu: *0.1, 0.02, 0.01*.
- **Typ optimizéru:** určuje konkrétní typ optimalizačního algoritmu použitého při tréninku neuronové sítě. (U tohoto a dalších parametrů je možné zadat část textu pro ukázku všech možností optimizéru.)
- **Rychlost učení:** určuje míru, jak rychle se parametry modelu mění během tréninku v závislosti na gradientu ztrátové funkce.
- **Typ ztrátové funkce:** určuje konkrétní typ ztrátové funkce, která je použita při trénování neuronové sítě.
- **Typ metriky:** určuje konkrétní typ metriky použité při hodnocení výkonu neuronové sítě.

- **Počet epoch:** určuje, kolik epoch má být použito při zpětnovazebním učení neuronové sítě.
- **Velikost dávky:** určuje, jaká velikost dávky má být použita při zpětnovazebním učení neuronové sítě.
- **Poznámky:** mohou obsahovat libovolný text omezený na 200 znaků, které slouží k uložení dodatečných informací nebo poznámek k neuronové síti.

2.3 Trénování a testování modelů tanků

Pro arénové souboje je k dispozici celkem 16 přednastavených tanků. Pro účely tréninku a testování bylo vybráno pět nejlepších tanků z této škály. Z této selekce bylo následně náhodně vybráno s opakováním devět protihráčů, kteří sloužili k tréninku modelu mého tanku. Tento přístup je záměrně založen na náhodnosti, aby se předešlo přílišnému specializování modelu tanku.

Pro selekci nejlepších pěti tanků byl využit hierarchický turnajový systém, kde každý z tanků absolvoval 10 000 bitev v jednotlivých fázích turnaje. Nejprve proběhl turnaj nazvaný „všichni proti všem“, následně se utkali ti nejlepší z desetice a nakonec byla vybrána pětice nejlepších tanků, jak ukazuje 2.3. V této sérii soubojů se zvlášť výrazně prosadil tank s označením Walls, který se pohyboval podél vnějšího okraje arény s kanónem namířeným směrem do středu. Na druhém místě se umístil SpinBot, jenž se pohyboval po kruhové dráze a intenzivně pálil, jakmile zaznamenal nepřátelskou přítomnost. Třetí pozici obsadil Tracker, jenž si vybíral určitého robota, přibližoval se k němu a v momentě blízkosti provedl palbu. Čtvrté místo získal tank TrackFire, který čekal na nejbližšího viditelného protivníka a okamžitě na něj vystřelil. A na pátém místě se umístil tank s názvem Crazy, jehož pohyb po aréně byl nepředvídatelný. Pro trénování byla testována možnost úpravy pokročilých tanků[2] tak, aby poskytovaly veškeré potřebné informace pro vstup do modelu. Avšak v konečném důsledku tato možnost nebyla využita. Pro oznamování dokončení trénování a testování bylo implementováno zvukové upozornění, které lze vypnout pomocí globální proměnné NOTIFICATION_SOUND.

Pro hodnocení modelu jsou zohledněny následující faktory:

- **Frekvence provádění akcí:** čím častěji je opakovaná stejná akce provedena, tím nižší je přidělená odměna. To zabraňuje modelu zaujímat příliš repetitivní chování.
- **Změna energie:** odměna je ovlivněna změnou energie hráče. Pozitivní změna energie je odměněna, zatímco negativní změna energie je penalizována. Tím

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

se modelu poskytuje podnět k účinnému vyhýbání se poškození a podnět k aktivnímu střílení.

- **Stav tanku:** Odměna je také upravena podle stavu tanku. Například, pokud tank narazil do stěny nebo do jiného tanku, což modelu pomáhá vyhýbat se rizikovým situacím a preferovat strategie, které minimalizují poškození.

Tab. 2.3: 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			

2.3.1 Optimalizace parametrů modelu tanku

V této části se zaměřím na postupné upravování jednotlivých parametrů s cílem dosáhnout optimalizace modelu tanku. Pro tyto účely budu používat přezdívky modelů, které budou představovat jednotlivé změny parametrů podobou *v0.x.y*, kde x značí upravovaný parametr a y konkrétní úpravy.

Pro modely DQN se často používají 2 až 4 skryté vrstvy, přičemž počet neuronů v těchto vrstvách se obvykle pohybuje mezi počtem vstupních a výstupních neuronů. Často se jako aktivační funkce používá ReLU. Použití dropoutu u neuronů ve skrytých vrstvách je volbou, která závisí na konkrétní architektuře sítě a na komplexnosti dat, která jsou zpracovávána. Malé DQN modely často vynechávají použití dropout, protože tyto sítě mají tendenci k menšímu přeučení a jejich parametry jsou obvykle méně citlivé na šum. Epsilon-greedy strategie se často uplatňuje v případech, kdy je potřeba vyvážit průzkumné a explozivní chování modelu při učení z interakce s prostředím. Pro regresní problémy může být vhodná ztrátová funkce *mean squared error* (MSE) nebo *mean absolute error* (MAE). Počet epoch a dávek je také závislý na konkrétním problému a architektuře modelu, ale obvykle se volí hodnoty, které zajistí dostatečné pokrytí dat a stabilní učení modelu. Pro účely srovnání parametrů, tréninkových a testovacích dat byla vytvořena trojice databází v adresáři *NNModels*, přičemž grafy jsou ukládány do složky *graphs*.

Počet skrytých vrstev neuronové sítě

Pro začátek provedu úpravu velikosti skryté vrstvy a otestuji modely s 0 až 4 skrytými vrstvami. Počet neuronů v každé vrstvě bude nastaven na 128 a jako aktivační funkce bude použita ReLU. Dropout a maskování budou prozatím vypnuty. Diskontní faktor bude nastaven na 0,9, zatímco epsilon-greedy strategie bude dočasně deaktivována. Toto rozhodnutí je motivováno přítomností náhodných protivníků a variabilitou startovních pozic, což může ovlivnit efektivitu strategie průzkumu prostředí. Jako optimalizační algoritmus bude použit Adam s rychlostí učení 0,0001. Ztrátová funkce bude MSE (Mean Squared Error). Jako metrika bude sledována přesnost (accuracy). Prozatím budu trénovat model pouze po jedné epoše s dávkou velikosti 1.

Tab. 2.4: Testování velikosti skryté vrstvy

Přezdívka NN – počet vrstev	Ztrátová hodnota na epizodu	Počet zásahů na epizodu	Skóre tanku na epizodu	Celkem epizod
v0.0.0 – 0	73	1,22	140,9	500
v0.0.1 – 1	51	1,48	127,2	500
v0.0.2 – 2	13	1,07	120,5	500
v0.0.3 – 3	9	0,80	113,4	500
v0.0.4 – 4	4	1,03	139,6	500

Z analýzy tabulky 2.4 vyplývá, že s přidáváním skrytých vrstev klesá ztrátová hodnota. U sítí s žádnou a jednou skrytou vrstvou lze pak z přiložených grafů pozorovat, že pokles ztrátové hodnoty probíhá pomaleji, než u ostatních konfigurací. První model z nich začne oscilovat mezi hodnotami 20 a 80, zatímco druhá se ustálí v rozmezí hodnot 2 až 8, což odpovídá trendu ostatních sítí. Analýza četnosti akcí u sítě se čtyřmi skrytými vrstvami ukazuje nežádoucí trend, kdy je jedna pohybová akce znatelně preferována před ostatními. Na základě těchto zjištění se ukázalo, že sítě označené jako *v0.0.1* a *v0.0.2* dosáhly nejlepších výsledků.

Počet neuronů ve skryté vrstvě neuronové sítě

Dalším krokem bude testování různých počtů neuronů ve skryté vrstvě. Při volbě vhodného počtu neuronů je běžné preferovat hodnoty, které jsou násobky dvou. Pro tuto analýzu budu vycházet z různých verzí modelů označených jako *v0.0.1* až *v0.0.3*. Pro každou z těchto verzí budu zkoumat efektivitu neuronové sítě při použití dvou různých velikostí skryté vrstvy: 64 a 256 neuronů.

Tab. 2.5: Testování počtu neuronů ve skryté vrstvě

Přezdívka NN – počet neuronů	Ztrátová hodnota na epizodu	Počet zásahů na epizodu	Skóre tanku na epizodu	Celkem epizod
v0.1.0 – 64	46	1,45	114,7	500
v0.1.1 – 256	30	0,72	126,3	500
v0.1.2 – 64	18	1,13	121,7	500
v0.1.3 – 256	11	0,77	122,1	500
v0.1.4 – 64	14	0,8	110,3	500
v0.1.5 – 256	5	0,74	119,9	500

Z analýzy tabulky 2.5 vyplývá, že modely s 64 neurony ve skrytých vrstvách dosahují lepšího průměrného počtu zásahů za epizodu než modely s 256 neurony, avšak jejich celkové skóre je naopak horší. Porovnáním s předešlou tabulkou 2.4 vynikly nejlépe modely *v0.0.1* s jednou vrstvou a 128 neurony a *v0.1.2* se dvěma vrstvami a 64 neurony v každé vrstvě.

Aktivační funkce pro jednu a dvě skryté vrstvy

Dalším krokem bude testování různých aktivačních funkcí pro skryté vrstvy. Jejich podrobné vlastnosti lze najít na stránce: https://keras.io/api/layers/activations/. Pro tyto účely budu vycházet z verze *v0.0.1* s jednou skrytou vrstvou a následně z verze *v0.1.2* se dvěma skrytými vrstvami.

Tab. 2.6: Testování aktivační funkce u modelu s 1 skrytou vrstvou

Přezdívka NN - aktivační funkce	Ztrátová hodnota na epizodu	Počet zásahů na epizodu	Skóre tanku na epizodu	Celkem epizod
v0.2.0 – sigmoid	1	0,42	150,9	200
v0.2.1 – softmax	1,2	1	160,6	200
v0.2.2 – softplus	101	0,67	116,9	200
v0.2.3 – softsign	0,7	0,73	112,7	200
v0.2.4 – tanh	0,7	0,63	157,1	200
v0.2.5 – selu	97	1,4	132	200
v0.2.6 – relu6	1,4	0,84	111,3	200
v0.2.7 – silu	98,8	1,65	113,5	200
v0.2.8 – hard sigmoid	0,6	0,65	153,2	200
v0.2.9 – linear	200	1,28	128,7	200
v0.2.10 – elu	90	1,56	121,4	200
v0.2.11 – leaky relu	64,1	1,25	108,5	200
v0.2.12 – gelu	82,1	0,79	113,3	200

Z analýzy tabulky 2.6 vyplývá, že modely používající aktivační funkce jako *silu*, *elu*, *selu*, *linear* a *leaky relu* dosáhují nejlepších výsledků z hlediska průměrného počtu zásahů za epizodu. Porovnáním s modelem *v0.0.1* lze pozorovat, že aktivační funkce *relu* byla překonána funkcemi *elu* a *selu* v počtu zásahů. Nejlepší skóre dosáhly modely s aktivačními funkcemi *softmax*, *tanh*, *hard sigmoid* a *sigmoid*. Nicméně analýza grafů četností akcí u těchto modelů ukazuje nežádoucí trend, kdy je jedna pohybová akce výrazně preferována před ostatními.

Dalším krokem bude testování různých aktivačních funkcí pro dvě skryté vrstvy. Pro tyto účely budu vycházet z verze *v0.1.2*, nad kterou budu zkoušet pět nejlepších aktivačních funkcí z tabulky 2.6.

Tab. 2.7: Testování aktivační funkce u modelu se 2 skrytými vrstvami

Přezdívka NN – aktivační funkce	Ztrátová hodnota na epizodu	Počet zásahů na epizodu	Skóre tanku na epizodu	Celkem epizod
v0.3.0 – silu	28	1,59	129,6	200
v0.3.1 – elu	42,8	1,03	118,4	200
v0.3.2 – selu	42,6	1,62	123,3	200
v0.3.3 – linear	153,7	0,73	113,5	200
v0.3.4 – leaky relu	64,8	1,74	142,8	200

Z analýzy tabulky 2.7 vyplývá, že modely využívající aktivační funkce *leaky relu*, *selu* a *silu* dosáhly nejlepších výsledků z hlediska průměrného počtu zásahů za epizodu. Porovnáním s modelem *v0.1.2* lze pozorovat, že všechny tyto funkce překonali model s aktivační funkcí *relu*. Model *leaky relu* kromě největšího počtu zásahů ze všech předchozích modelů, také dosáhl největšího skóre.

Dropout a vstupní maska

Dalším krokem bude testování různých hodnot dropoutu a vlivu vstupní masky na model. Pro tyto účely budu vycházet z verze *v0.3.4* s aktivační funkcí *leaky relu*. Nejprve vyzkouším zapnutou masku a poté budu experimentovat s různými hodnotami dropoutu: nejprve 0.1 a 0.3 bez masky a poté s maskou.

Tab. 2.8: Testování dropout a vstupní masky

Přezdívka NN – dropout – maska	Ztrátová hodnota na epizodu	Počet zásahů na epizodu	Skóre tanku na epizodu	Celkem epizod
v0.4.0 – 0 – ano	8,7	1,54	114	200
v0.4.1 – 0,1 – ne	80,2	1,68	161,2	200
v0.4.2 – 0,3 – ne	249,2	1,89	119	200
v0.4.3 – 0,1 – ano	0,9	0,21	173,1	200
v0.4.4 – 0,3 – ano	0,7	0,21	191,1	200

Z analýzy tabulky 2.8 lze vyvodit, že modely bez vstupní masky dosahují lepších výsledků v průměrném počtu zásahů za epizodu. Naopak modely s maskou dosahují nejvyššího skóre, avšak za cenu mnohem menšího počtu zásahů. Tento nedobrý

trend je patrný i z analýzy grafů četností akcí, kde se ukazuje, že jeden typ pohybové akce je výrazně preferován před ostatními. Porovnáním s modelem *v0.3.4* lze pozorovat, že model bez masky a s dropout hodnotou $0,1$ dosáhl vyššího skóre než jeho předchůdce, ovšem za cenu mírně menšího průměrného počtu zásahů. Při analýze grafů četností akcí obou modelů bylo zjištěno, že model *v0.4.1* preferuje dvě pohybové akce a jednu střelovou akci s otočením, přičemž jedna pohybová akce je výrazně více používána, než ostatní. Naopak u starší verze je četnost rozdělena do více než deseti různých akcí.

Diskontní faktor (gama)

Následujícím krokem bude testování diskontního faktoru (gama), který ovlivňuje budoucí odměny. Pro snížení budoucích odměn na polovinu, platí rovnice: $\gamma^n = 0,5$, kde γ je diskontní faktor a n představuje číslo kroků, za kterou se budoucí odměna sníží o polovinu. Pro tyto účely budu vycházet z verze *v0.3.4* s aktivační funkcí *leaky relu* a vyzkouším gama hodnoty $0,95$ a $0,99$.

Tab. 2.9: Počet kroků k dosažení poloviční hodnoty budoucích odměn

γ	0,9	0,95	0,99
n	6,6	13,5	69

Tab. 2.10: Testování diskontního faktoru

Přezdívka NN – gama	Ztrátová hodnota na epizodu	Počet zásahů na epizodu	Skóre tanku na epizodu	Celkem epizod
v0.5.0 – 0,95	9	1,23	115,4	200
v0.5.1 – 0,99	3,7	1,68	120,2	200

Z analýzy tabulky 2.10 je patrné, že modely s vyšší hodnotou gama stále dosahují celkem dobrého průměrného počtu zásahů za epizodu. Podobně i jejich průměrné skóre není zanedbatelné, avšak ani v jednom případě nepřevyšují výsledky verze *v0.3.4*.

Epsilon

Dalším krokem bude testování epsilon-greedy strategie, která rozhoduje, jak často se agent rozhodne prozkoumat nové akce náhodně (epsilon) namísto toho, aby volil nejlepší akci modelu (greedy). Pro tyto účely budu vycházet z verze *v0.3.4*. Nejprve

nastavím epsilon na hodnotou 0, 5 a postupně se bude snižovat s krokem 0, 005 až na 0. Poté se provede druhý test s epsilon nastaveným na hodnotu 1, a postupně se bude snižovat s krokem 0, 01 až na 0.

Tab. 2.11: Testování epsilon-greedy strategie

Přezdívka NN – epsilon	Ztrátová hodnota na epizodu	Počet zásahů na epizodu	Skóre tanku na epizodu	Celkem epizod
v0.6.0 – 0,5;0;0,005	8,3	1,53	123,2	200
v0.6.1 – 1;0;0,01	32,3	1,79	115,6	200

Z analýzy tabulky 2.11 je patrné, že modely s různým nastavením epsilon stále dosahují celkem dobrých průměrných počtů zásahů za epizodu. Nejvýraznějším z nich je model *v0.6.1* s nejširším rozsahem epsilon-greedy strategie, který v počtu zásahů na epizodu předčí všechny předchozí modely.

Optimizér

Dalším krokem bude testování optimizéru, který ovlivňuje proces optimalizace modelu. Jejich podrobné vlastnosti lze najít na stránce: https://keras.io/api/optimizers/. Pro tyto účely budu vycházet z verze *v0.6.1*.

Tab. 2.12: Testování optimizéru

Přezdívka NN – optimizér	Ztrátová hodnota na epizodu	Počet zásahů na epizodu	Skóre tanku na epizodu	Celkem epizod
v0.7.0 – rmsprop	3,7	1,91	114,6	200
v0.7.1 – adadelta	3,7	1,53	119	200
v0.7.2 – adagrad	4,4	1,78	125,2	200
v0.7.3 – adamax	3,9	1,88	130,1	200
v0.7.4 – nadam	4,3	1,7	120,5	200
v0.7.5 – ftrl	8,7	1,55	118,3	200

Z analýzy tabulky 2.12 je patrné, že všechny modely s různým nastavením optimizéru dosahují průměrně nad 1,5 zásahů na epizodu. Nejvýraznějším z nich je model *v0.7.0* s optimizérem *rmsprop*, který v počtu zásahů na epizodu předčil všechny předchozí modely.

Rychlost učení

Dalším krokem bude testování různých hodnot rychlosti učení, což je klíčový parametr optimalizačního algoritmu. Správně zvolená rychlost učení ovlivňuje efektivnost učení neuronové sítě. Pro tyto účely budu pracovat s verzí *v0.7.0* jako výchozím bodem a budu zkoušet rychlosti 10^{-3} a 10^{-5}.

Tab. 2.13: Testování diskontního faktoru

Přezdívka NN – rychlost učení	Ztrátová hodnota na epizodu	Počet zásahů na epizodu	Skóre tanku na epizodu	Celkem epizod
v0.8.0 – 10^{-3}	91,3	2,41	126,1	200
v0.8.1 – 10^{-5}	22,3	1,66	104,2	200

Z analýzy tabulky 2.13 je patrné, že model *v0.8.0* s nejnižší rychlostí učení 10^{-3} dosáhl nejvyššího počtu zásahů na epizodu než všechny předchozí modely.

Ztrátová funkce

Dalším krokem bude testování různých ztrátových funkcí, které mají vliv na výsledky učení neuronové sítě. Jejich podrobné vlastnosti lze najít na stránce: `https://keras.io/api/losses/`. Pro tyto účely budu pracovat s verzí *v0.8.0* jako výchozím bodem. Postupně budou vyzkoušeny pravděpodobnostní, regresní a hinge ztrátové funkce.

Tab. 2.14: Testování různých ztrátových funkcí

Přezdívka NN – ztrátová funkce	Ztrátová hodnota na epizodu	Počet zásahů na epizodu	Skóre tanku na epizodu	Celkem epizod
v0.9.0 – binary crossentropy	-2015,4	2,28	111,1	200
v0.9.1 – categorical crossentropy	681.049,6	2,14	114,5	200
v0.9.2 – kl divergence	0	1,84	109,3	200

v0.9.3 – mean absolute error	2,9	2,02	118,7	200
v0.9.4 – mean absolute percentage error	26.074.612,9	2,08	120,4	200
v0.9.5 – mean squared logarithmic error	0,2	1,87	120,4	200
v0.9.6 – cosine similarity	-1	2,19	128,7	200
v0.9.7 – huber	2,7	1,78	115,3	200
v0.9.8 – log cosh	2,7	2,34	123,9	200
v0.9.9 – hinge	0,4	2,01	131,3	200
v0.9.10 – squared hinge	304,8	1,77	133,5	200
v0.9.11 – categorical hinge	0,7	1,99	159,9	200

Z analýzy tabulky 2.14 je patrné, že změna ztrátové funkce může v některých případech výrazně ovlivnit ztrátovou hodotu během trénování, a to ne vždy ve prospěch zlepšení modelu. Ztrátová funkce určuje, jak velká je chyba mezi predikcemi modelu a skutečnými hodnotami v trénovacích datech. Nejlépe si vedly modely s funkcemi *log cosg* a *binary crossentropy*, pokud jde o průměrný počet zásahů na epizodu. Nicméně ani jeden z těchto modelů nepřekonal v tomto ohledu model *v0.8.0* se ztrátovou funkcí *mean squared error*.

Počet epoch

Dalším krokem bude testování různých počtů epoch, což ovlivňuje účinnost učení neuronové sítě. Epochy představují počet průchodů trénovacími daty během učení. Prozkoumám efekt učení sítě při použití 10 a 50 epoch. Pro tyto účely budu vycházet z verze *v0.8.0* jako výchozího bodu.

Tab. 2.15: Testování počtu epoch

Přezdívka NN – počet epoch	Ztrátová hodnota na epizodu	Počet zásahů na epizodu	Skóre tanku na epizodu	Celkem epizod
v0.10.0 – 10	209,8	1,79	115,98	200
v0.10.1 – 50	2.677.044,8	2,14	134,56	200

Z analýzy tabulky 2.15 lze vyvodit, že model s větším počtem epoch dosáhl většího průměrného počtu zásahů na epizodu, avšak na úkor vysoké ztrátové hodnoty. Tento jev je potvrzen i z grafu četností akcí, kde se výrazně preferují pouze 2 akce ze 44 možných. Tento vzorec naznačuje, že model byl přetrénován na specifické akce, což může omezit jeho schopnost adekvátně reagovat na různorodé situace. Nicméně ani jeden z těchto modelů nepřekonal v tomto ohledu model *v0.8.0*, který dosáhl vyšší účinnosti a stabilnějšího chování při zachování nižší ztrátové hodnoty.

Velikost dávky

Posledním krokem bude testování různé velikosti dávky, která určuje, kolik trénovacích vzorků je použito při jednom kroku optimalizace. Nižší hodnoty obvykle zvyšují stabilitu učení, zatímco vyšší hodnoty mohou vést k rychlejší konvergenci, ale také k větší fluktuaci. Pro tyto účely budu vycházet z verze *v0.8.0* jako výchozího bodu a budu testovat dávky o velikostech 32 a 64.

Tab. 2.16: Testování velikosti dávky

Přezdívka NN – velikost dávky	Ztrátová hodnota na epizodu	Počet zásahů na epizodu	Skóre tanku na epizodu	Celkem epizod
v0.11.0 – 32	59,2	1,93	111,4	200
v0.11.1 – 64	126,3	2,11	120,5	200

Z analýzy tabulky 2.16 je zřejmé, že model s vyšší velikostí dávky dosáhl většího průměrného počtu zásahů na epizodu. Nižší hodnoty dávky obvykle přispívají ke stabilnějšímu učení a mohou vést k lepší schopnosti generalizace, což se odráží i na nižší ztrátové hodnotě. Nicméně ani jeden z těchto modelů nedosáhl výkonnosti modelu *v0.8.0*, který prokázal vyšší účinnost a lepší výkon při učení.

Nastavení odměny

Dále jsem trénoval model *v0.8.0* dalších 1000 epizod, ale nedošlo k výraznému zlepšení výkonu. Ani při testování proti 6 nejhorším robotům z tabulky 2.3 nebyly pozorovány významné změny. Při analýze problému jsem zjistil, že negativní odměny mohou nabývat mnohem nižších hodnot než pozitivní, proto jsem modifikoval odměny tak, aby byly pozitivní odměny pětkrát vyšší. Pro toto nové nastavení jsem upravil model *v0.8.0* na verzi *v1.0.0*. Hlavní změny zahrnují návrat k optimalizátoru *adam* a úpravu epsilon-greedy strategie: během prvních 500 epizod epsilon klesal z 100% na nulu, a poté z 50% na nulu.

Tab. 2.17: Testování různé odměny

Přezdívka NN	Průměrné pořadí	Počet zásahů na epizodu	Skóre tanku na epizodu	Celkem epizod
v0.8.0	8,24	1,54	111,4	1200
v1.0.0	5,45	6,34	303,8	1000

Z analýzy tabulky 2.17 je zřejmé, že zvýšení pozitivní odměny vedlo k celkovému zlepšení modelu. Model *v1.0.0* dosáhl nejlepšího průměrného pořadí a mezi předprogramovanými tanky se umístil lehce nad průměrem, který je pro 10 tanků $(1 + 10)/2 = 5,5$. Ve srovnání s předchozími modely se výrazně zlepšil také v počtu zásahů na epizodu, což vedlo k dosažení nejvyššího skóre.

3 Výsledky a diskuze

V rámci práce byla úspěšně provedena optimalizace agenta, který dosahuje 1000 herních tiků za sekundu. Tento agent je schopen ukládat informace o aréně do souboru *ModelGameData.txt*, který slouží jako jeho epizodická paměť. Současně zajišťuje soketovou komunikaci mezi Java a Python částmi, které jsou označeny jako *RobotTrainAI.java* a *RobotAI.py*.

Dále byl vytvořen hlavní skript v jazyce Python pro konfiguraci a spouštění platformy Robocode pro účely trénování. Tento skript obsahuje uživatelské rozhraní umožňující různé funkce pro práci s neuronovou sítí tanku, včetně ukázky jejich spojené databáze. Data o nastavení parametrů pro modely, stejně jako data z trénování a testování, jsou ukládána do tří oddělených databází ve složce *NNModels*, která obsahuje i samotné modely neuronových sítí. Podle potřeby lze tyto databáze spojit a zobrazit v prostředí PandasGUI, kde je možné provádět další analýzy a manipulace, například vytváření různých statistik nebo tvorba vlastních grafů.

Tento skript má okolo 2600 řádků kódu a obsahuje komentáře a typovou anotaci. Byl kladen důraz na dodržení standardů PEP, a proto kód neobsahuje žádné varování. Trénování probíhá primárně na grafické kartě a samotná trénovací epizoda trvá 7 až 8 sekund. Z toho 3 až 5 sekund zabere načítání knihovny TensorFlow. Následně se načte herní aréna, která se nastaví přibližně za sekundu. Celá bitva v aréně, při nastavení maximální rychlosti, se provede pod sekundu, a nakonec dochází k zpětnému trénování, které trvá 1 až 2 sekundy.

Celkem bylo vytvořeno 64 modelů, přičemž bylo testováno 13 různých hyperparametrů neuronové sítě. Z jejich trénování a testování vzniklo přes 300 grafů, které jsou uložené ve složce *graphs*. Nejlepší modely mají označení *v0.8.0* a *v1.0.0*. Trénování modelů pomocí DQN není jednoduché kvůli složitosti prostoru možných stavů a akcí, který patří do třídy EXPSPACE. Předprogramované tanky mohou využívat přesně definované funkce, které například umožňují okamžitě zaměřit kanón na nepřítele. Naše DQN modely se musí naučit aproximovat složité matematické funkce, jako jsou goniometrické funkce, aby dosáhly podobné úrovně přesnosti a efektivity.

Analýza výkonu modelu *v0.8.0*

Při pozorování prvního z nich, model vykazoval možnost hbitě se pohybovat s občasným zastavením, díky čemuž byl částečně nepředvídatelný pro ostatní tanky. Také občas vykazoval schopnost vyhnutí se jedné nebo více střel najednou. Ale jakmile je obklíčen vícero tanky, má problém se z tohoto scénáře dostat ven. Kvůli nastavené odměně je zřejmé, že se tank neodnaučil střílet a snaží se tyto akce používat, avšak bez možnosti přesného zamíření na protivníka je těžší některého trefit, ať už na jeho aktuální pozici nebo do predikce.

Jeho parametry jsou zobrazeny na obrázku 3.1. Tento model prošel tréninkem v průběhu 200 epizod, během nichž byl systematicky testován každých 100 epizod. A poté po dalších 1000 epizodách, pro porovnání s modelem *v1.0.0*.

Jak ukazuje graf 3.2, během prvních 100 epizod docházelo k postupnému poklesu ztrátové hodnoty a zároveň k nárůstu přesnosti, což naznačuje, že model se zlepšuje a učí se. Nicméně ve druhé polovině tréninku, jak je patrné z grafu 3.3, pokles ztrátové hodnoty zpomaluje a dochází k periodickým výkyvům, zatímco přesnost začíná klesat. Tento vzorec naznačuje, že model se přibližuje svému predikovanému limitu a začíná se zhoršovat.

Z grafu 3.4 lze vidět, že model se pohybuje mezi nejhorším a průměrným skóre. Toto skóre není závislé pouze na posledním přeživším, ale také na počtu zásahů, jak je patrné z grafu 3.5.

Z grafu 3.6 je patrné, že model preferuje 6 akcí, přičemž převládá pohyb tanku před střílením s otáčením kanónu v poměru 68 ku 32. Při detailnější analýze vidíme, že model nejvíce preferoval samotné otáčení, následované jízdou dopředu, otáčením kanónu a nakonec střelbou. Nicméně žádná z možných akcí nebyla zcela vyloučena.

Graf 3.7 zobrazuje nejúspěšnější epizodu tohoto modelu, ve které tank dokázal zasáhnout ostatní tanky pětkrát. Tento úspěch je patrný z přírůstku energie a faktu, že tank přežil až do posledního okamžiku.

Pokud porovnáme tento výsledek s modelem *v0.1.2*, což je předchozí verze současného modelu, lze jasně vidět dvojnásobné zlepšení počtu zásahů na epizodu.

Analýza výkonu modelu *v1.0.0*

Druhý model byl trénován na šesti nejhorších předprogramovaných robotech uvedených v tabulce 2.3. Při sledování výkonu modelu bylo zaznamenáno výrazné zlepšení oproti všem předchozím modelům. Zvláště se to projevilo v přesnosti střelby a v celkovém dosaženém skóre. Nicméně vzhledem k diskrétním akcím bylo patrné, že tank nedokázal střílet ve všech úhlech a občas minul o malý kousek cíl. Co se týče pohybu, byl oproti předchozí verzi modelu méně aktivní. Přesto tento model dosáhl umístění 5,45, což je lehce nad průměrem mezi předprogramovanými tanky. Z grafu 3.8 je patrné, že model dosahuje průměrného skóre mezi předprogramovanými tanky.

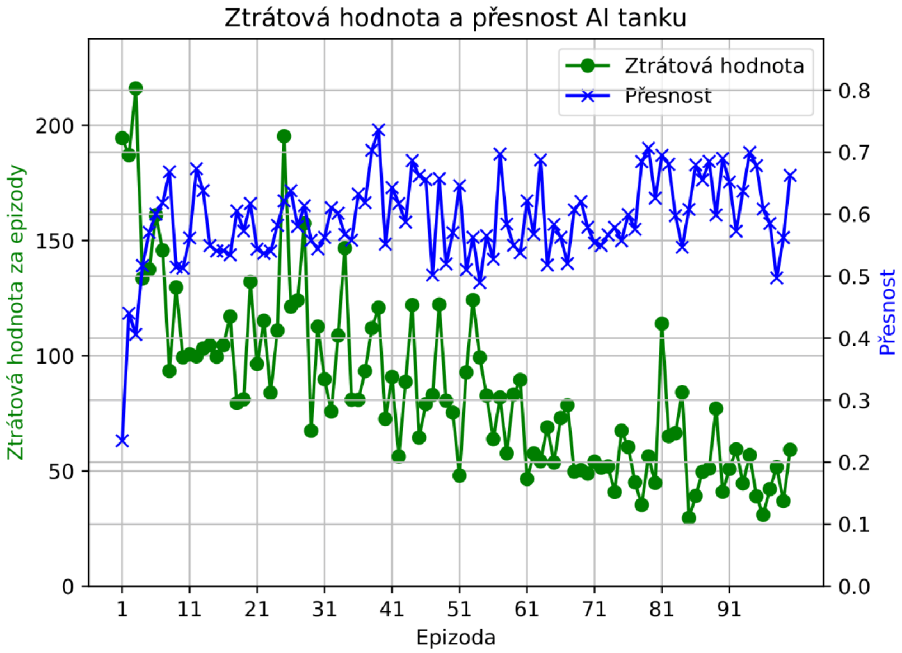
Analýza grafu 3.9 ukazuje, že model preferuje čtyři akce, přičemž nejčastěji používanou akcí je střelba s otáčením kanónu, která převažuje nad pohybem v poměru 63 ku 37. Podrobnější analýza odhalila, že model nejčastěji používá střelbu bez pohybu při síle 2 a 3, a z pohybových akcí preferuje otáčení doprava a stání na místě. Je však důležité poznamenat, že žádná z možných akcí není zcela vyloučena.

Zajímavá epizoda tohoto modelu je zobrazena na grafu 3.10, kde tank dokázal zasáhnout ostatní tanky šestnáctkrát. Co se týče pořadí, skončil na třetí pozici.

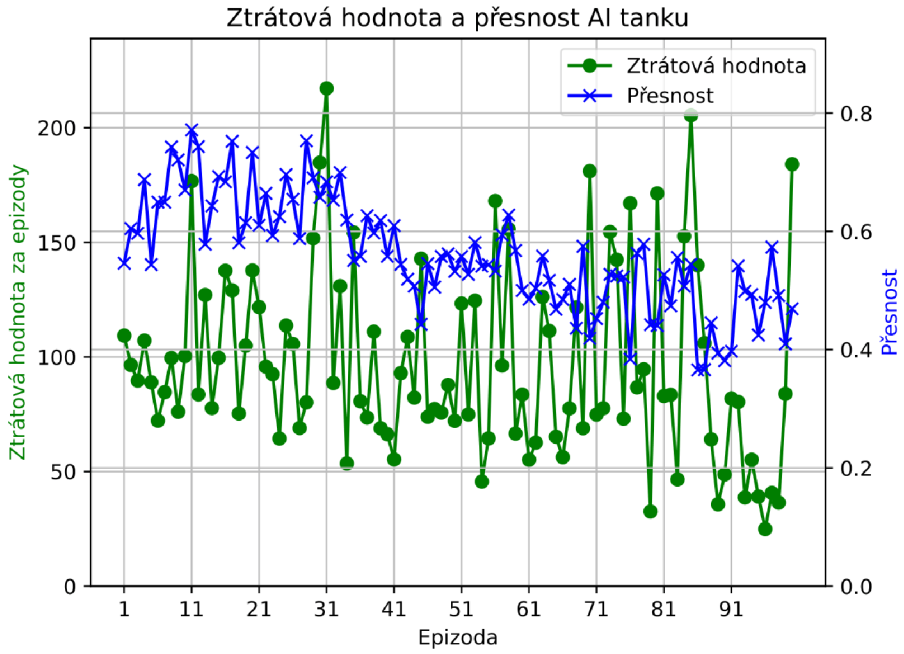
Při srovnání tohoto výsledku s modelem *v0.8.0*, viz tabulka 2.17, je patrné zlepšení ve všech sledovaných kategoriích. Průměrné pořadí mezi předprogramovatelnými tanky se zlepšilo z 8,24 na 5,45, počet zásahů se zčtyřnásobil a průměrné skóre se téměř třikrát zvýšilo.

Parametry NN:	
NN name	2024-05-11-23-21-40
NN nickname	v0.8.0
NN hidden layers	[[64,leaky_relu,0],[64,leaky_relu,0]]
input mask	False
gamma	0.9
epsilon	[1,0,0.01]
optimizer type	rmsprop
learning rate	0.001
loss function type	mean_squared_error
metric type	accuracy
epochs	1
batch size	1
notes	learning rate

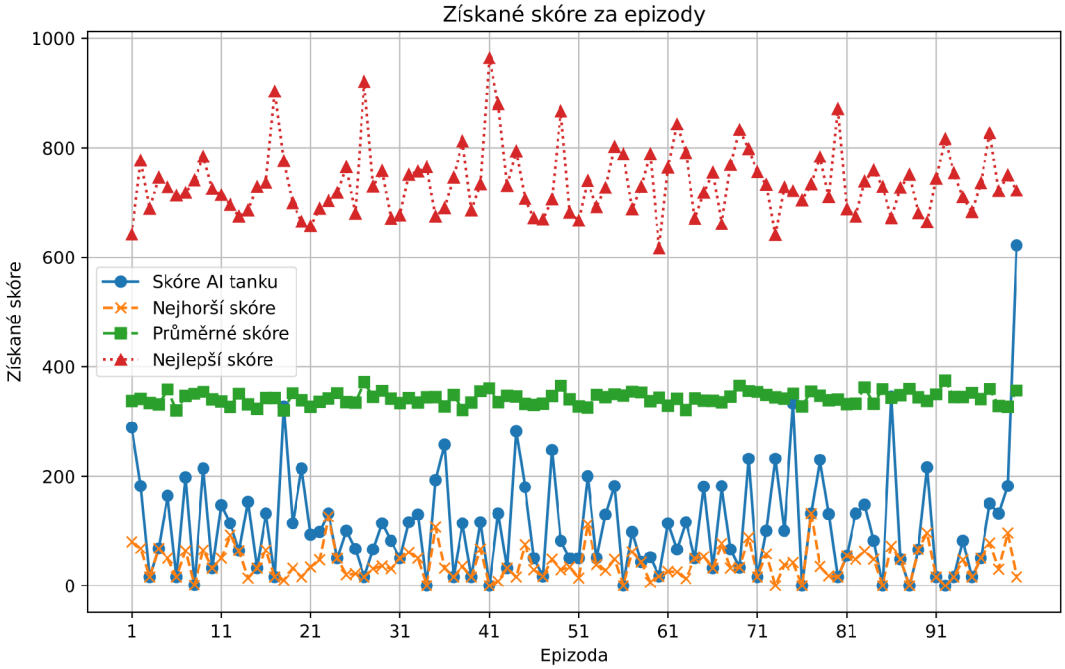
Obr. 3.1: Parametry modelu *v0.8.0*



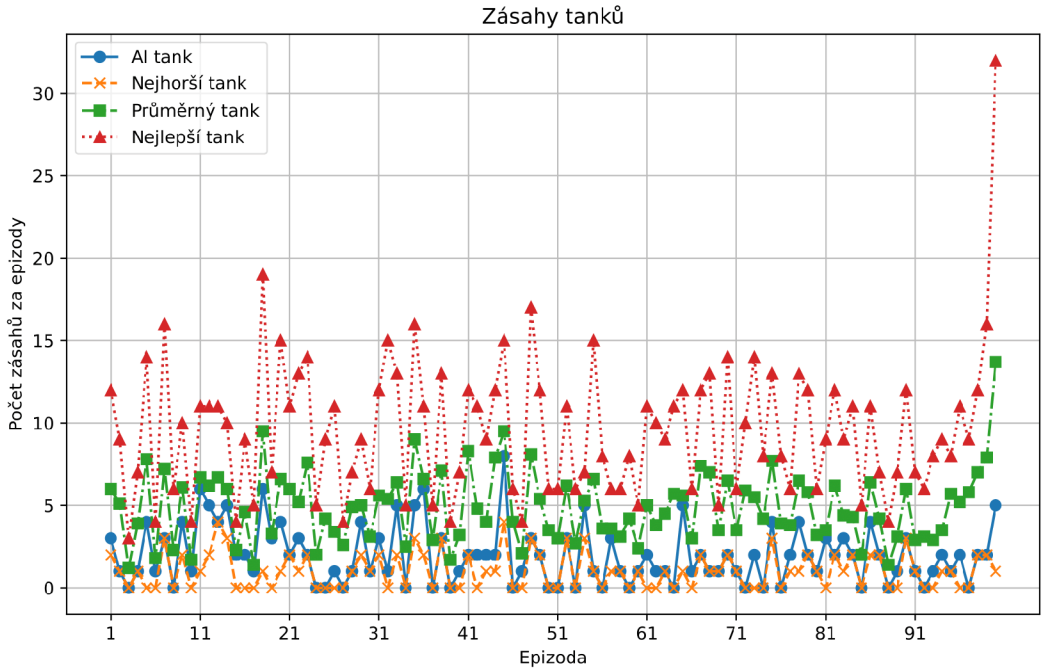
Obr. 3.2: Graf ztrátové hodnoty a přesnosti během prvních 100 epizod pro *v0.8.0*



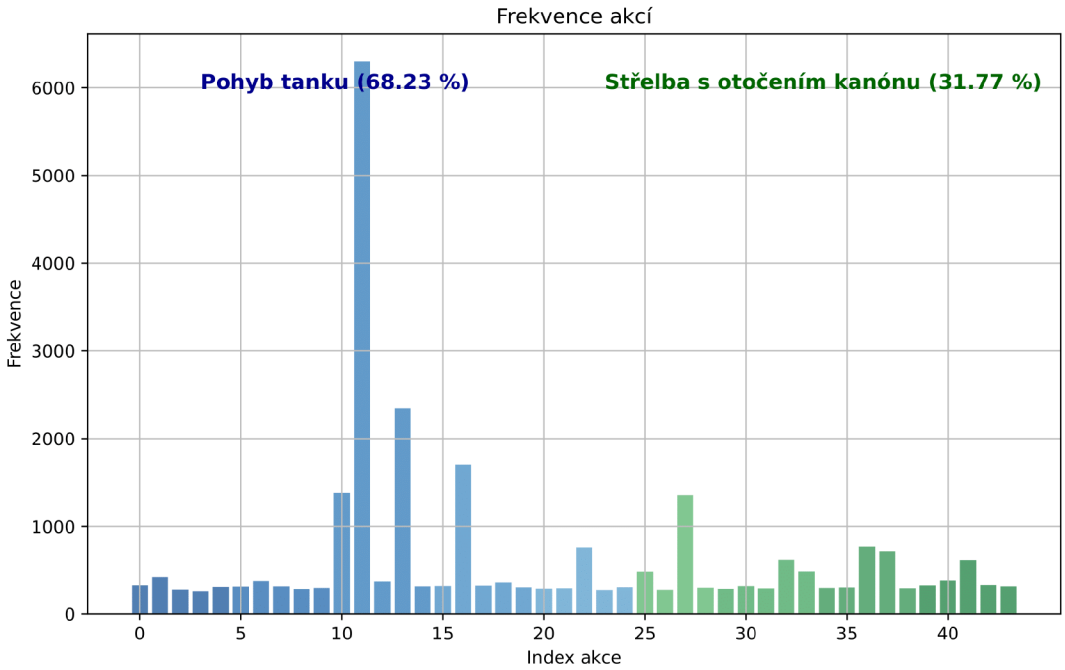
Obr. 3.3: Graf ztrátové hodnoty a přesnosti během druhých 100 epizod pro *v0.8.0*



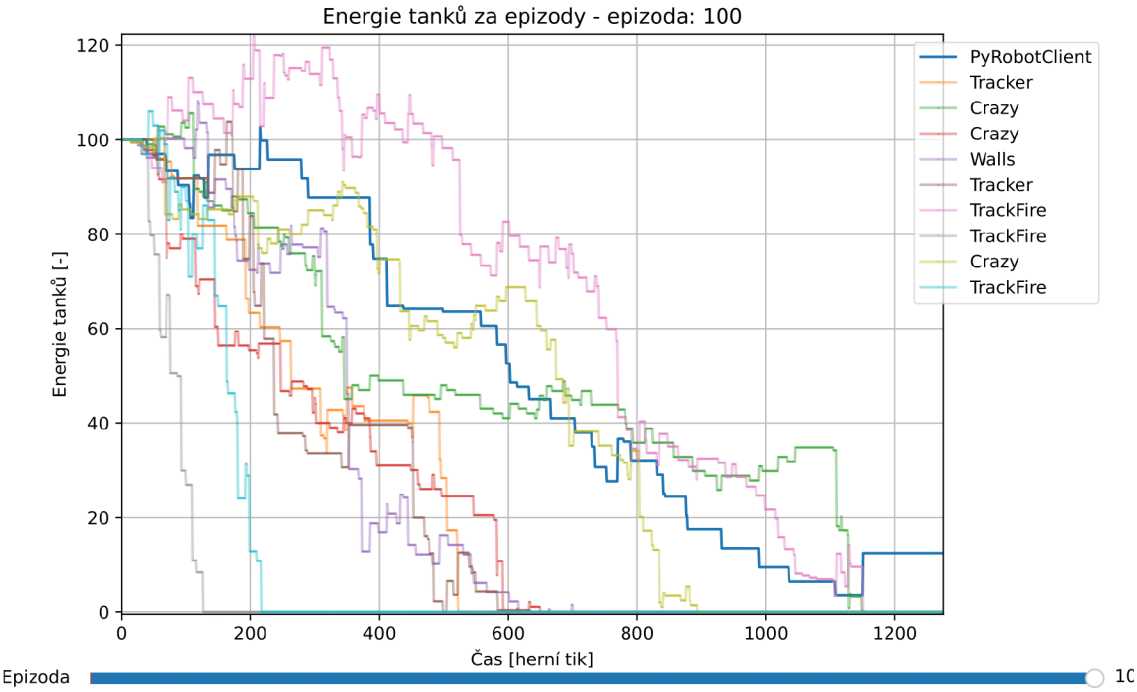
Obr. 3.4: Graf získaného skóre pro druhých 100 epizod pro *v0.8.0*



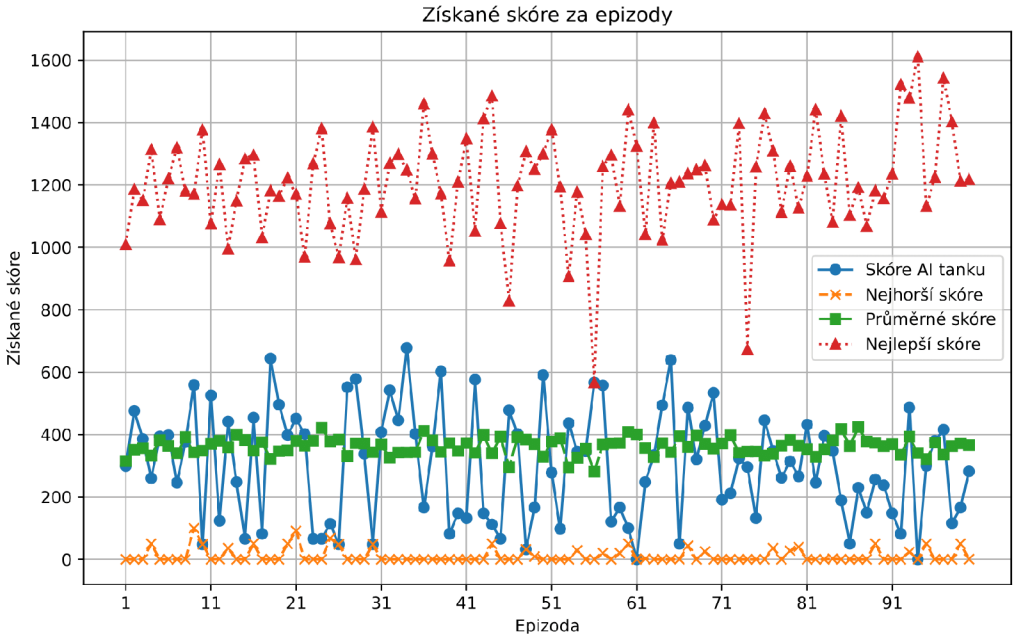
Obr. 3.5: Graf zásahů tanků pro druhých 100 epizod pro *v0.8.0*



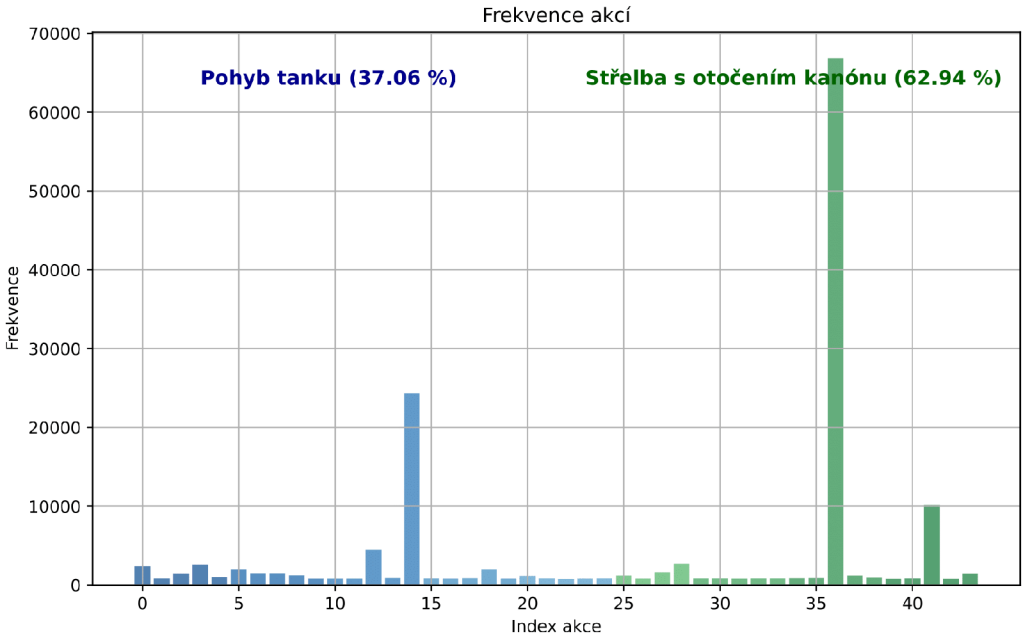
Obr. 3.6: Graf frekvence akcí pro druhých 100 epizod pro *v0.8.0*



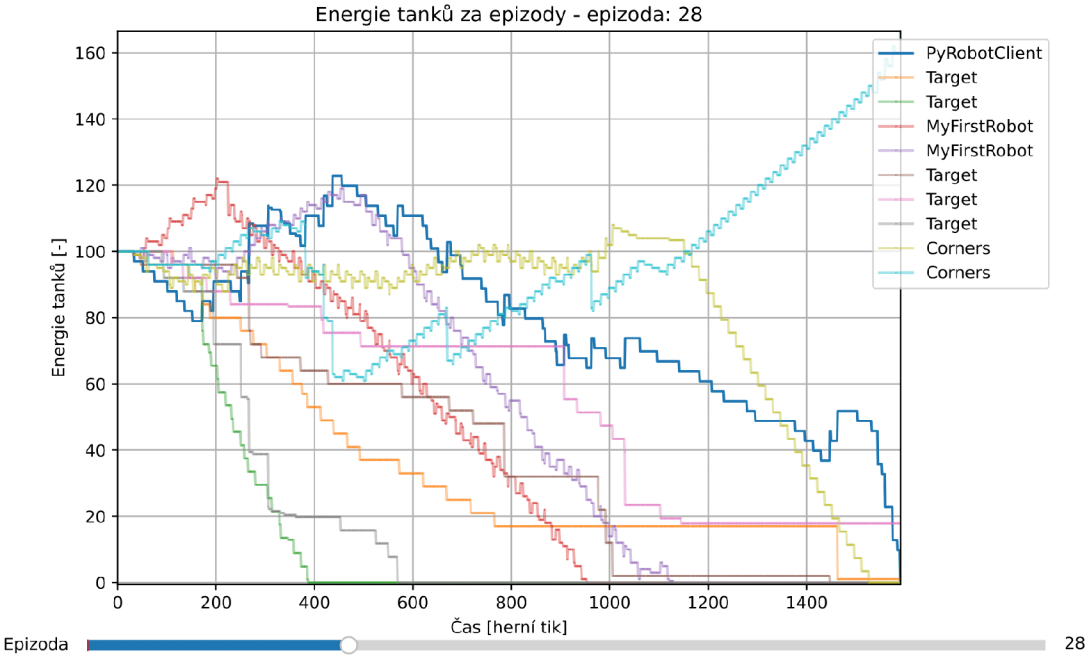
Obr. 3.7: Graf energií tanků pro epizodu 200 pro *v0.8.0*



Obr. 3.8: Graf získaného skóre pro *v1.0.0*



Obr. 3.9: Graf frekvence akcí pro *v1.0.0*



Obr. 3.10: Graf energií tanků pro *v1.0.0*

3.1 Možnosti rozšíření a vylepšení modelu

Současný stav implementace agenta vyžaduje načítání knihovny TensorFlow před každou epizodou, což je efektivnější než v původní verzi, kde se knihovna musela načítat při každém herním tiku. Nicméně je zde prostor pro další optimalizaci. Hlavní Python skript by mohl být přepracován na výpočetní stanici pro tanky, což by umožnilo načíst knihovnu TensorFlow pouze jednou před spuštěním trénování. Toto však vyžaduje, aby Python část agenta komunikovala jak s Java částí tanku, tak s výpočetní stanicí. Další možností rozšíření je podpora více modelů současně, což by vyžadovalo práci s více vlákny.

Pro další zlepšení trénování by bylo zajímavé zredukovat nebo upravit vstupní stavy pro neuronovou síť a zkoumat alternativní diskrétní akce z výstupu sítě. Taktéž by bylo zajímavé upravit model tak, aby místo volby konkrétní akce volil strategii nebo funkci, buď vlastní nebo z předprogramovaných tanků. Další možností pro pokročilé zlepšení výkonu by mohlo být použití hierarchické hluboké Q sítě (H-DQN). Hlavní DQN by se zaměřila na volbu strategie, jako je například rozhodnutí, zda je vhodnější se pohnout nebo začít střílet, zatímco druhá úroveň obsahující několik DQN by tato strategická rozhodnutí realizovala na nižší úrovni.

Další možností pro vylepšení je přidání vlastního ovládání tanku pomocí myši a klávesnice, což by umožnilo testování v duelu a další experimenty s interakcí tanků.

Závěr

Tato práce se zaměřuje na optimalizaci řízení bitevních robotů pomocí zpětnovazebního učení v prostředí Robocode. To zahrnuje navržení skriptu využívajícího vhodný algoritmus, vytvoření a optimalizaci agenta a natrénování a otestování modelů, které jsou následně porovnávány mezi sebou. Kromě toho bylo nezbytné vyřešit problém opakovaného volání arény a navrhnout a optimalizovat komunikaci mezi jazyky Java a Python, které jsou součástí agenta a hlavního trénovacího skriptu.

Vzhledem k velikosti stavového prostoru a množství akcí jsem pro trénování vybral algoritmus DQN s epizodickou pamětí. Avšak trénování modelů pomocí DQN není jednoduché kvůli složitosti tohoto prostoru, který spadá do třídy EXPSPACE. Předprogramované tanky mají často k dispozici přesně definované funkce, které jim umožňují okamžitě zaměřit kanón na nepřítele. Naše DQN modely se musí naučit aproximovat složité matematické funkce, například goniometrické funkce, aby dosáhly podobné úrovně přesnosti a efektivity.

Během diplomové práce jsem úspěšně optimalizoval komunikaci agenta a provedl trénování a testování 64 modelů tanků, přičemž jsem testoval 13 různých hyperparametrů neuronové sítě. Z těchto tréninků a testů vzniklo více než 300 grafů. Pro trénování těchto modelů jsem vyvinul speciální skript *robocode_rl_trainer.py*, který umožňuje efektivní trénink. Po každém tréninku a testování je automaticky generován a uložen graf do adresáře *graphs*, a záznam je ukládán do databáze ve složce *NNModels*, kde jsou také uloženy jednotlivé modely.

Dále byl vytvořen agent, jehož klíčové části jsou obsaženy v souborech *RobotTrainAI.java* a *RobotAI.py*. První z nich umožňuje ukládání stavů a akcí do epizodické paměti, která je zaznamenávána v textovém souboru *ModelGameData.txt*. Druhý soubor slouží k vyhodnocování akcí na základě stavového vstupu načteného modelu. Pro zvýraznění mého tanku mezi ostatními jsem provedl úpravu jeho radaru, který nyní bliká postupně ve všech barvách. Dále byly provedeny optimalizace ve skriptu *RobocodeRunner.java*, který spouští arénu a umožňuje nastavení různých parametrů prostřednictvím konfiguračních souborů umístěných ve složce *config*.

Ze všech zkoumaných modelů tanků vynikly dva modely s označením *v0.8.0* a *v1.0.0*, které dosáhli nejvyššího průměrného počtu zásahů na epizodu ve srovnání s ostatními modely. Při pozorování prvního z nich bylo zjištěno, že model projevuje schopnost rychle se pohybovat s občasným zastavením, což ho činí částečně nepředvídatelným pro ostatní tanky. Také občas vykazoval schopnost vyhýbat se jedné nebo více střelám současně. Nicméně, když je obklíčen několika soupeři, má problém se z tohoto scénáře vymanit. I přestože tank nestřílí tak často jako se pohybuje, zdá se, že se to zcela neodnaučil, a je zřejmé, že se snaží tyto akce používat. Avšak bez možnosti přesného zamíření na protivníka je těžší některého zasáhnout,

ať už na jeho aktuální pozici nebo do predikce. V nejlepší epizodě dokázal rozdat 5 zásahů, viz graf 3.7.

Druhý model byl trénován na základě chování šesti nejhorších předprogramovaných robotů uvedených v tabulce 2.3. Při sledování výkonu tohoto modelu bylo zaznamenáno výrazné zlepšení oproti všem předchozím modelům, zejména v přesnosti střelby a v celkovém dosaženém skóre. Nicméně kvůli diskrétním akcím, které se získávají z DQN, bylo patrné, že tank nedokázal střílet ve všech úhlech a občas minul cíl o malý kousek. Co se týče pohybu, byl oproti předchozí verzi modelu méně aktivní. Přesto tento model dosáhl umístění 5,45, což je lehce nad průměrem mezi předprogramovanými tanky. Z grafu 3.8 je patrné, že model dosahuje průměrného skóre mezi předprogramovanými tanky a v nejlepší epizodě dokázal rozdat 16 zásahů, viz graf 3.10.

Veškerý kód a vývojová dokumentace jsou k dispozici v repozitáři na platformě GitHub: https://github.com/wenca611/Diplomova-Prace. Repozitář obsahuje jak zdrojový kód, tak i natrénované modely a grafy, které byly vytvořeny během práce. Z důvodu omezeného rozsahu příloh však nebyly přímo začleněny do přílohy k této práci, s výjimkou nejlepších modelů.

Při vývoji epizodické DQN jsem se inspiroval kódem popisujícím DQN na hře had, který naleznete na následujícím odkazu: https://github.com/Jaswar/Snake-CNN-Deep-Q-Learning.

Výhodou mého programu je využití GPU pro zpětné trénování modelu, což vedlo k několikanásobnému zrychlení procesu trénování. Jeden tréninkový cyklus, který zahrnuje několik stovek až tisíců stavů a akcí, spolu s načtením knihoven, spuštěním arény a zpětným učením, trvá pouze 7 až 8 sekund. Dále nabízím uživatelské prostředí s databází neuronových sítí, které usnadňuje práci s modely a umožňuje snadné vytváření jejich grafů.

Tato práce otevírá cestu pro další rozvoj a zdokonalení modelů tanků. Možnosti pro budoucí práci jsou široké a mohou vést k ještě lepším výsledkům. Jeden potenciální směr je přechod od predikce diskrétních akcí k strategickému rozhodování. Tím by tank získal schopnost vybírat z různých strategií, jako je přesné zamíření na protivníka a následné vystřelení nebo definovaný úhyb. Tato inovace by mohla významně zvýšit efektivitu tanku v boji a přinést nové perspektivy do oblasti řízení bitevních robotů.

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] Time Complexity Classes. In: *Automata, Computability and Complexity: theory and applications*. Upper Saddle River: Pearson / Prentice Hall, c2008, s. 490. ISBN 978-0-13-228806-4.

[10] *CSE 431 Theory of Computation*. Online. THE UNIVERSITY OF WASHINGTON. Https://www.cs.washington.edu/. 2014. Dostupné z: https://courses.cs.washington.edu/courses/cse431/14sp/scribes/. [cit. 2024-05-18].

[11] 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.

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

[13] 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.

[14] 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.

[15] 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].

[16] 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.

[17] 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.

[18] 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.

[19] 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-q-learning-6ffe3b0a9062

[20] 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/`

[21] 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`

[22] 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

α	Velikost kroku
γ	Diskontní faktor
O	Velká O notace – prostorová složitost algoritmu
r_t	Odměna získaná za provedení akce a_t ve stavu s_t
CPU	Centrální procesorová jednotka – Central Processing Unit
CUDA	Jednotná architektura pro výpočty na zařízení – Compute Unified Device Architecture
cuDNN	Knihovna pro hluboké neuronové sítě na zařízení s architekturou CUDA – CUDA Deep Neural Network library
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
DSPACE	Deterministický prostor – Deterministic Space
DTIME	Deterministický čas – Deterministic Time
EMDQN	Hluboká Q síť s epizodickou pamětí – Episodic Memory Deep Q Network
EXPSPACE	Exponenciální prostor – Exponential Space
EXPTIME	Exponenciální čas – Exponential Time
FAQ	Často kladené otázky – Frequently Asked Questions
GPU	Grafický procesor – Graphics Processing Unit
GUI	Grafické uživatelské rozhraní – Graphical User Interface
H-DQN	Hierarchická hluboká Q síť – Hierarchical Deep Q Network
IBM	International Business Machines Corporation
IDE	Vývojové prostředí – Integrated Development Environment

JDK Java vývojový balík – Java Development Kit

L Logaritmický prostor – Logarithmic Space

LUT Vyhledávací tabulka – Look-Up Table

MAE Střední absolutní chyba – Mean absolute error

ML Strojové učení – Machine Learning

MSE Střední kvadratická chyba – Mean squared error

NEXPSPACE Nedeterministický exponenciální prostor – Nondeterministic Exponential Space

NEXPTIME Nedeterministický exponenciální čas – Nondeterministic Exponential Time

NL Nedeterministický logaritmický prostor – Nondeterministic Logarithmic Space

NP Nedeterministický polynomiální čas – Nondeterministic Polynomial Time

NPSPACE Nedeterministický polynomiální prostor – Nondeterministic Polynomial Space

NSPACE Nedeterministický prostor – Nondeterministic Space

NTIME Nedeterministický čas – Nondeterministic Time

P Polynomiální čas – Polynomial Time

PEP Návrhy na vylepšení Pythonu – Python Enhancement Proposal

PSPACE Polynomiální prostor – Polynomial Space

ReLU Usměrňovač (rampa) – Rectified Linear Unit

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. 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 Python kódu ve třídě *RobocodeRunner*. 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 daly přenastavit z Pythonu. Před spuštěním je potřeba mít nastavenou cestu k Java knihovnám pro Robocode, toto nastavení je popsáno v souboru *REDME.md* ve složce RoboCode.

Celkem se dá nastavit 34 proměnných. Ty nejdůležitější jsou nastavění přidání tanků (i podle indexu), velikosti mapy, počtu kol, zapnutí/vypnutí GUI, nastavení zvuku, viditelnost 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* nebo zavolat příkaz *pip install -r requirements.txt*,

- `/` ..... kořenový adresář: RoboCodeProject
  - `doc` ..... text semestrální práce
    - `xpastu02_semestralni_prace.pdf`
  - `graphs` ..... grafy modelů po trénování a testování - 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_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 — databáze a 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