VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

HLUBOKÉ NEURONOVÉ SÍTĚ PRO POSILOVANÉ UČENÍ

DEEP NEURAL NETWORKS FOR REINFORCEMENT LEARNING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **TOMÁŠ LUDVÍK**
AUTHOR

VEDOUCÍ PRÁCE **MICHAL HRADIŠ, Ing. Ph.D.**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2021/2022

Zadání bakalářské práce

Student: **Ludvík Tomáš**
Program: Informační technologie
Název: **Hluboké neuronové sítě pro posilované učení**
Deep Neural Networks for Reinforcement Learning
Kategorie: Zpracování obrazu

Zadání:

1. Prostudujte základy neuronových sítí a posilovaného učení.
2. Vytvořte si přehled o současných metodách využívají neuronové sítě a posilované učení se zaměřením na hry více hráčů.
3. Vyberte konkrétní metodu aplikovatelnou na určitý problém posilovaného učení.
4. Obstarejte si simulační prostředí vhodné pro experimenty.
5. Implementujte navrženou metodu a proveďte experimenty ve zvoleném prostředí.
6. Porovnejte dosažené výsledky a diskutujte možnosti budoucího vývoje.
7. Vytvořte stručné video prezentující vaši práci, její cíle a výsledky.

Literatura:

- Ian Goodfellow and Yoshua Bengio and Aaron Courville, Deep Learning, MIT Press, 2016.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Hradiš Michal, Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 1. listopadu 2021

Abstrakt

Cílem této práce je použití hlubokých neuronových sítí na problém v posilovaném učení. Používám moji úpravu 2D hry Tuxánci jako testovací prostředí. Jedná se o úpravu, která zajišťuje možnosti využití hry jako prostředí pro strojového učení. Následně řeším problémy s naučením agenta pomocí posilovaného učení algoritmem Double DQN. Pomocí experimentů si prokazuji správné nastavení funkce odměn.

Abstract

The aim of this thesis is to use deep neural networks for task in reinforcement learning. I use my modification of 2D game Tuxánci for the purposes of the test environment. This modification provides the possibility of using the game as an environment for machine learning. Subsequently, Iam solving the task of learning the agent by using reinforcement learning with the Double DQN algorithm.

Klíčová slova

Posilované učení, DQN, Tuxánci, Tensorflow, strojové učení, testovací prostředí

Keywords

reinforcement learning, DQN, Tuxánci, Tensorflow, machine learning, testing enviroment

Citace

LUDVÍK, Tomáš. *Hluboké neuronové sítě pro posilované učení.* Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Michal Hradiš, Ing. Ph.D.

Hluboké neuronové sítě pro posilované učení

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Michala Hradiše, Ph.D Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Tomáš Ludvík
11. května 2022

Poděkování

Rád bych poděkoval vedoucímu mé bakalářské práce panu Ing. Michalu Hradišovi, Ph.D. za odborné vedení a poskytnuté konzultace při tvorbě mé práce.

Obsah

Kapitola 1

Úvod

Posilované učení nalézá každým dnem více a více uplatnění a to především díky hlubokým neuronovým sítím. Společně dokáží tyto algoritmy pojmou velké množství informací, díky kterým jsou schopné fungovat i ve složitých prostředí. Ale přesto je zde spoustu překážek pro využití plného potenciálu, jako například omezený výpočetní výkon dnešních stojů. Oblast posilovaného učení je inspirována člověk a to tím jak se rychle učí pomocí předešlých zkušenosti a tím, že je odměňován za požadované chování.

V této práci se snažím zjistit co metody pro vylepšení posilovaného učení přináší, přesněji se zaměřuji na oblast Q-učení a jeho vylepšení algoritmus DQN. Součástí cíle této práce je implementace komunikace a ovládání počítače hry Tuxánci pro potřeby obecně strojového učení. A toto prostředí následně použít společně s algoritmem DQN pro naučení agenta logice hry.

Práce je rozdělená do sedmi kapitol. Ve druhá kapitole je popis existujících řešení problémů podobných tomu mému. V kapitole třetí jsou základní pojmy posilované učení. Ve čtvrté kapitole je popsáno prostředí ve kterém se agent vyskytuje. V páté kapitole jsou informace o implementaci prostředí, komunikaci, reprezentaci stavu a použitých algoritmů. Šestá kapitola informuje o experimentech na daných problémech, jak se algoritmu dařilo učení a jak byli nastaveny různé parametry. Poslední kapitola shrnuje práci a nabízí možná vylepšení do budoucnosti.

Kapitola 2

Existující řešení podobných problémů

Hlavním průlomem v oblasti využití hlubokých neuronových sítích pro posilovaného učení byla publikace od výzkumníků v DeepMind o vytvoření DQN agenta, který je schopen hrát mnoho klasických Atari 2600 her. A dokonce 29 z nich s lepším výsledkem než lidský hráč.[1] Před tím byli pokusy o naučení agentů hrá hry jako například Quake III Arena pomocí základního Q-učení ale nic revolučního.[10] Existují také vylepšení pro DQN jako Prioritized experience replay nebo Double DQN, které se snaží urychlit a zpřesnit tento algoritmus.

Dokonalým příkladem multi-agent učení je úspěch od výzkumníků v DeepMin, kde dokázali vytvořit agenta s výkonem na lidské úrovni ve hře Quake III Arena v módu obsaď vlajku. Dalším příkladem řešeni složitých her pro více hráčů je multi-agent AlphaStar který se naučil hrát StarCraft II na vysoké úrovni. K řešení složitosti a herních teoretických výzev StarCraftu využívá AlphaStar kombinaci nových a stávajících univerzálních technik od architektury neuronových sítí, imitation learning, učení posilování až po multi-agent učení [12].

Kapitola 3

Posilované učení

V této kapitole se zaměřím na základy posilovaného a obecně strojového učení, jelikož je to nutné pro pochopení záměrům této práce.

3.1 Rozdělení podle typu učení

Strojové učení

Strojové učení patří pod oblast umělé inteligence, která se zabývá algoritmy se schopnostmi učení. Při procesu učení probíhá vylepšování modelu opakovaným průchodem získaných a nových zkušenosti. Pro správné naučení algoritmů je zapotřebí obrovské množství dat. Naučený model je po-té schopný vyhodnocovat i nové vstupy bez lidské pomoci[11].

Učení s učitelem

Podstatou učení s učitelem je naučit model pomocí dvojice vstupních dat a správných výstupů. Tento typ učení se dá například využít při rozpoznání čísel z obrázku. V prvním kroku člověk vytvoří soubor obrázků s čísly a k ním přiřadit správný popis o jaké číslo se jedná. Model poté vezme obrázek na vstup a vyhodnotí výstup. Ten se porovná se správným výsledkem a provede se korekce modelu.

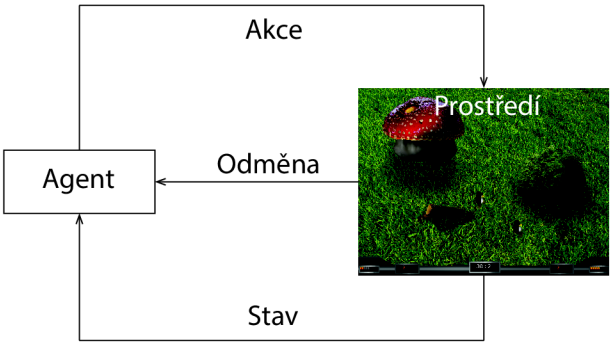
Učení bez učitelem

Při učení bez učitele není předem známý správný výstup. Model se učí jen pomocí vstupních dat a to pomocí principu pokus – omyl. Tímto způsobem se dá řešit problém shlukování dat.

Posilované učení

Posilovaného učení je odvětví strojového učení, ve kterém se model učí posloupnost akcí, které přiblíží agenta k jeho cíli v daném prostředí. Prostředí může být všechno od jednoduché hry (například snake) až po simulační prostředí robotických systémů. U složitějších prostředí bývá většinou posilované učení jediné možné řešení v oblasti strojového učení. Tento typ učení je inspirován lidským učením, kde lidé dostávají zpětnou vazbou z okolí. Model se učí pomocí odměn které získá z prostředí, které se snaží maximalizovat. Používá při problémech u kterých můžeme vyjádřit skupinu možných akcí a odměnu za dosažený cílů. Například autonomní řízení auta, kde agent ovládá rychlost a otáčení auta. Za každých

ujetých 100m dostane kladnou odměnu a pokud vyjede z silnice nebo narazí do překážky obdrží velkou negativní odměna. Pro posilované učení existuje spoustu algoritmů. Všechny



Obrázek 3.1: Základní cyklus při učení algoritmu v posilovaném učením

algoritmy se dají rozdělit do dvou kategorii podle toho zda si algoritmus vytváří model prostředí (model-free / model-based).

3.2 Základní pojmy

Epizoda

Epizoda je definovaná jako posloupnost akcí které jsou zakončeny koncovým stavem. Epizoda bývá ukončená agentem po dosažení konečného stavu. Aby se nestalo, že agent nikdy nedosáhne konečného stavu je nastaven limit kroků/akcí, po kterém se epizoda sama ukončí.

Policy

Policy se říká funkci, která přiřazuje k stávajícímu stavu akcí. Policy se dají rozdělit na dvě skupiny deterministické a stochastické. Deterministická policy má vždy jednu výslednou akci.

$$\pi(s) = a \tag{3.1}$$

U stochastických je výsledkem funkce rozložení pravděpodobnosti všech možných akcí.

$$\pi(a, s) = P[a, s] \tag{3.2}$$

Odměna

Základní učícím prvkem je odměna (reward), kterou dostává agent od prostředí a tím ho motivuje. Odměny je důležité správně nastavit aby se agent naučil zamýšlené chování. Samotná odměna vyjadřuje zda zvolená akce, která vedla ke změně na nový stav, byla výhodná.

Akce

Akcí se rozumí jeden příkaz, který má agent vykonat v daném prostředí. Akce mohou být dvou typů. Prvním typem jsou akce spojité, u kterých je akce vyjádřená určitou hodnotou v předem daném rozsahu. Příkladem je ovládání úhlu natočení volantu u auta nebo otočení kloubu u robota.

Druhý typ se nazývá diskrétní akce. U tohoto typu je každá akce individuální příkaz, který má agent v prostředí vykonat. Například pohyb postavy ve hře čtyřmi směry, kde pohyb na daný směr je individuální akce.

Gradientní sestup

Gradientní sestup (Gradient Descent) je algoritmus, který se snaží minimalizovat funkci, tím že se pohybuje opačným směrem vůči Gradientu v daných bodech. Výsledkem tohoto sestupem je lokální minimum funkce. Pokud je funkce konvexní, potom je výsledkem globální maximum. Parametry tohoto algoritmu jsou:

- **Rychlost učení** určuje délku jednotlivého kroku při provádění algoritmu. Menší rychlost znamená přesnější nalezení minima, ale je zase časově náročné. Při velké rychlost je naopak rizikem nedosažení přesného minima, jelikož je šance že ho překročí. Je proto důležité zvolit správnou rychlost.
- **Maximální počet iterací**
- **Tolerance** určuje jak blízko se algoritmus musí přiblížit minimu aby došlo k jeho ukončení.

3.3 Markovův rozhodovací proces

MDP (Markov decision process) je nejrozšířenějším způsobem popisu problémů v posilovacím učení [14]. Interakce agenta s prostředím probíhá v daných časových krocích i. V každém kroku učiní agent akci a a obdrží nový stav prostředí S_i a odměnu R_i. Další stav je vždy závislí jen na stávajícím stavu a akci, kterou agent provedl. Pro zjednodušení problému se budeme bavit jen konečnými MDP. Výsledkem jedné epizody konečného MDP je trajektorie, kterou agent prošel. A ten poslední stav se nazývá terminální. Trajektorie se skládá ze seznamu stavů, akcí a odměn, který popisuje interakci agenta s prostředím.

Očekávaná návratnost

Množství, které se snažíme maximalizovat při řešení popsaného problému podle MDP je očekávání náhodné proměnné zvané návratnost. V časovém kroku t pro posloupnost odměn je návrat celkem intuitivně definován jako

$$Gt = R_{t+1} + R_{t+2} + R_{t+3} + \ldots + R_T \tag{3.3}$$

Kde T je index posledního kroku. U mnoho problémů řešených posilovacím učením jsou takzvané epizodické úkoly, což znamená, že jsou prováděny v epizodách skládajících se z konečného počtu kroků 1 (Příkladným úkolem by byly šachy, kde každá partie končí po určitém počet tahů s jedním vítězným hráčem nebo nerozhodný stav) [14].

3.4 Exploration / Exploitation

Tento problém vzniká v posilovaném učení tím, že není jasně dáno jak zvolit správný poměr mezi prozkoumávaní nových stavů a následování již naučené cesty. Pokud zvolíme příliš prozkoumávaní hrozí, že se agent nic nenaučí, jelikož pořád získává nové stavy, které ještě

neviděl. Naopak při příliš důsledným nasledováním naučené politiky, se může stát že se agent zasekne v lokálním minimu a už se dál učit nebude.

Proto existuje ϵ-greedy strategie, která s pravděpodobností ϵ zvolí náhodnou akci. Hodnota samotného epsilonu v průběhu učení klesá. Může se nastavit jeho minimální hodnota, pod kterou neklesne.

3.5 Policy-Based metody

Hlavní myšlenkou u policy-based je umět ve stavu s určit, jakou akci podniknout, aby se maximalizovala odměna.[8]

3.6 Value-Based metody

Metody založené na hodnotách mají za cíl najít optimální hodnotovou funkci, ze které lze získat policy.

3.7 Q-Learning

Q-učení má za cíl naučit se jaká akce je v daném stavu nejlepší. K tomu využívá tabulku Q hodnot, která obsahuje hodnoty pro kombinaci všech stavů a všech možných akcí. Čím vyšší Q hodnota, tím je akce v daném stavu výhodnější [9].

Tabulka Q Hodnot		Akce						
		Akce 1	Akce 2	Akce 3	Akce 4	Akce 5	Akce 6	Akce 7
Stavy	1	-1,21	-1,25	-0,21	1,25	-1,81	-2,21	-1,81
	2	0	0	0	0	0	0	0
	3	-1,12	-3,21	-2,91	1,25	-1,68	-2,22	-0,29
	4	1,88	-0,65	0,12	-2,34	-0,87	-1,01	-0,54
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	n	0	0	0	0	0	0	0

Obrázek 3.2: Q tabulka s n stavy a sedmi akcemi s již naučenými hodnotami

Tato tabulka se aktualizuje při každé provedené akce a to podel toho jakou odměnu agent obdržel. Tato aktualizace se provádí podle této rovnice:

$$Q(s,a) = Q(s,a) + \alpha \cdot (r + \gamma \cdot max(Q(s',a)) - Q(s,a)) \tag{3.4}$$

Kde:

- a je akce kterou agent provedl.
- s je stav ze kterého tuto akci provedl.

- s' je stav, ve kterém se agent po provedení akce nachází.
- α je rychlost učení. Tento parametr rozhoduje o kolik se Q hodnota upraví. Nabývá hodnot v rozsahu od nuly po jedna včetně.
- r značí odměnu kterou agent právě obdržel.
- γ se nazývá slevový faktor (discount factor) $\gamma \epsilon < 0, 1 >$. Hodnota tohoto parametru udává jak jsou důležité odměny v budoucích krocích. Pokud je slevový faktor 0, úprava Q hodnoty je závislá jen na okamžité odměně. Kdežto při slevovém faktoru 1 mají všechny budoucí odměny stejnou váhu jako odměna okamžitá.
- $max(Q(s', a))$ vyjadřuje akci s největší Q hodnotu pro stávající stav.

DQN

DQN (Deep Q Network) je běžně používaný algoritmus v posilovaném učení, poprvé využitý pro hraní Atari her [1]. Algoritmus DQN je založen na Q-Learning. Problém u Q-Learning je u prostředí s velkým množství stavů a akcí je teměř nemožné uložit a aktulizovat tak velkou tabulku Q hodnot. Tento problém je u DQN vyřešený využítím hluboké neuronové sítě pro aproximaci Q hodnot namísto Q tabulky. U DQN se ale objevuje nový problém. Při vyhodnocování Q hodnot je vyhodnocení závislé na měnících se cílových Q hodnotách a to způsobuje nekoneční stíhání nestatického cíl. Tento problém je možno minimalizovat použití záznamové paměti (Experience replay).

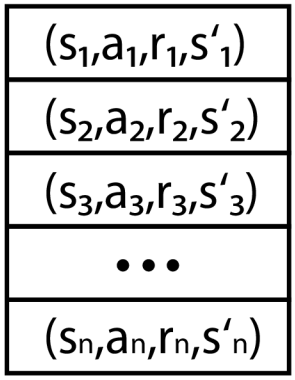
Algoritmus 1: `DQN`

```
Inicializace záznamové paměti pro minulé stavy
Vytvoření modelu neuronové sítě a náhodná inicializace jeho vah
for Každou episodu do
    Restartování prostředí
    Inicializace prvního stavu s
    while Episoda neskončila do
        Vybrání akce a pomocí modelu a = argmax(Q(s)) nebo náhodná s
          pravděpodobností ϵ
        Provedení akce prostředím
        Získání nového stavu s' a odměny r
        Uložení zkušenost jako čtveřice (s, a, r, s') do paměti
        Náhodně vybrat soubor i zkušeností (s, a, r, s') z paměti
        Vyhodnocení souboru zkušeností v_i = Q(s_i)
        t_i = r_i pro koncové stavy
        t_i = r_i + γ · amax(Q(s')) pro ostatní
        Trénování se stavy s_i jako daty a t_i jako jejich štítky
    end
end for
```

Experience replay

Hlavní funkcí je uložení minulých zkušeností do paměti. Z této paměti jsou následné náhodně vybrané soubory zkušeností, nad kterými se následně provede učení. Toto zajistí, že vybrané

zkušenosti mají menší pravděpodobnost, že jsou na sobě závislé.Na tomto obrázku 3.3 je znázorněné uložení minulých zkušeností.



Obrázek 3.3: Příklad uložení zkušeností v paměti, každá položka obsahuje stávající stav s, akci kterou agent provedl a, odměna za akci r, příští stav s'

Prioritized experience replay

Běžný experience replay používá jednotné vzorkování, které je dostačující pro lehké úkoly. Pro složitější problémy je ale lepší vybírat ty zkušenosti, které mají větší informační hodnotu pro učení. Upřednostňování některých zkušeností při jejich vzorkování, je proto dalším trikem jak vylepšit proces učení. Při ukládání zkušeností se ukládá také priorita nastaví jako největší. Při vybrání a naučení nad zkušenosti se aktualizuje priorita podle chyby, kterou model udělal při vyhodnocení nejlepší akce. Při vybírání zkušeností musí mýt každá zkoušenost nenulovou pravděpodobnost vybrání. U algoritmu DQN můžeme jako ohodnocení důležitosti použít TD error, který se spočítá následovně [7]:

$$\delta = r + \gamma \cdot max(Q(s', a)) - Q(s, a) \tag{3.5}$$

a pro Double DQN

$$\delta = r + \gamma \cdot Q'(s', argmax(Q(s', a))) - Q(s, a) \tag{3.6}$$

Double DQN

Tento problém se snaží zlepšit verze DDQN (Double DQN) zlepšuje stabilitu učení pomocí rozdělení aproximace Q hodnoty na dvě neuronové sítě hlavní a cílová.Používá dva identické modely neuronové sítě. Řekněme-li, že pro stav s akce a je vyšší hodnota než akce b, pak akce a bude vybrána pokaždé pro stav s. Může se stát, že pro nějakou paměťovou zkušenost akce b stane lepší akcí pro stav s. pak protože neuronová síť je trénována způsobem, který dává mnohem vyšší hodnotu pro akci a, když je dán stav s, je obtížné trénovat síť, aby se naučila, že akce b je za určitých podmínek lepší akcí. A to se vyřeší tím, že jednu sít využijeme k vybrání akce a druhou na zhodnocení vybrané akce. [2] Po několika epizodách se zkopírují váhy neuronové sítě z hlavní sítě na cílovou.

Imitation learning

Podstatou IL(Imitation learning) je naučení agenta podle nahraných zkušeností lidským expertem. Tato metoda se dá spojit s posilovaným učení. Teorie je taková, že se agent v

Algoritmus 2: `Double DQN`

Inicializace záznamové paměti pro minulé stavy
Vytvoření modelu neuronové sítě a náhodná inicializace jeho vah
Zkopírování modelu neuronové sítě jejích vah a nazvat jí cílová síť
for *Každou episodu* **do**
    Restartování prostředí
    Inicializace prvního stavu s
    while *Episoda neskončila* **do**
        Vybrání akce a pomocí modelu $a = argmax(Q(s))$ nebo náhodná s pravděpodobností ϵ
        Provedení akce prostředím
        Získání nového stavu s' a odměny r
        Uložení zkušenost jako čtveřice (s, a, r, s') do paměti
        Náhodně vybrat soubor i zkušeností (s, a, r, s') z paměti
        Vyhodnocení souboru zkušeností $v_i = Q(s_i)$
        $t_i = r_i$ pro koncové stavy
        $t_i = r_i + \gamma \cdot Q'(s', argmax(Q(s')))$ pro ostatní
        Trénování se stavy s_i jako daty a t_i jako jejich štítky
        Každých n kroků zkopírování vah z hlavní do cílové sítě
    end
end for

první fázi naučí s využití gradientního sestupu pomocí zkušenostech, které mu jsou dodány. Po dobu první fáze si neukládá žádné nové stavy.

V druhé fázi už sám agent provádí akce a ukládá si zkušenosti do paměti. Při vybírání zkušeností lze zajistit aby část byla vybíraná z demonstračních dat a zbytek z vlastních. V této fáze agent používá ϵ-greedy policy, která ale začíná na nízké hodnotě (např. $\epsilon = 0.01$) [4]. Existují i jiná sofistikovanější řešení, ale pro potřeby této práce, bude toto řešení dostačující.

3.8 Multi-agent posilované učení

U jednoduchých agentů v posílení učení je modelován tak, aby prováděl postupné rozhodování při interakce s okolím. Prostředí je obvykle formulováno jako Markovův rozhodovací proces s konečný (MDP). Také MARL (multi-agent RL) také řeší sekvenční problémy rozhodování, ale s více než jedním zapojeným agentem. Samotný vývoj stavu systému, tak odměna obdržená každým agentem jsou ovlivněny společnými akcemi všech agentů. Ještě zajímavější je, že každý agent má svou vlastní dlouhodobou odměnu, kterou může optimalizovat, což se sdílí se všemi ostatními agenty. V MARL se rozlišuje několik typů problémů podle typu vazby mezi agenty [5].

Kooperativní

V plně kooperativním prostředí všichni agenti obvykle sdílejí společnou funkci odměn.Pří kooperativním modelu je Q-funkce totožná u všech agentů, což umožňuje použít jednoagentové RL algoritmy, pokud jsou všichni agenti koordinováni jako jeden subjekt. Kromě mo-

delu společné odměny existuje ještě jeden o něco obecnější model pro kooperativní MARL a ten počítá s týmovou průměrnou odměnu. Například agenti mohou mít různé funkce odměn, které mohou být jedinečné pro každého agenta, přičemž cílem spolupráce a optimalizování dlouhodobé odměny [5].

Kompetitivní

Pro zjednodušení se budeme bavit o dvou agentech, kteří proti sobě soutěží. Plně Kompetitivní prostředí v MARL má typicky nulovým součtem odměn mezi agenty, jelikož kde je jeden odměnou získá druhý agenta ji ztratí [5].

Kombinované

Kombinované nastavení je také známé jako nastavení hry s obecným součtem bez omezení je kladen na cíl a vztah mezi agenty. Každý agent je se snaží získat co nejvíce odměn pro sebe a při získaní odměny může sebrat odměnu ostatním. Dalším příkladem s může být dva týmy, které proti sobě soupeří a v každém týmu jsou dva hráči kteří spolupracují [5].

Překážky pro MARL

Navzdory obecnému modelu, který nachází široké uplatnění, MARL trpí několika dalšími problémy, kromě těch, které se objevují v RL s jedním agentem. Jedna z hlavních výzvev MARL spočívá ve skutečnosti, že se více agentů obvykle učí současně, což způsobuje, že prostředí, kterému čelí každý jednotlivý agent, je nestacionární. Při akci provedené jedním agentem ovlivní odměnu ostatních nepřátelských agentů. To způsobuje to, že agent se musí naučit jak se ostatní agenti chovají a přizpůsobují tomuto chování.

Aby bylo možné zvládnout nestacionaritu, každý jednotlivý agent možná bude muset zohlednit společnou akci.

3.9 Hluboké neuronové sítě

Hluboké neuronové sítě je označení pro síť neuronů, která má mnoho vrstev.

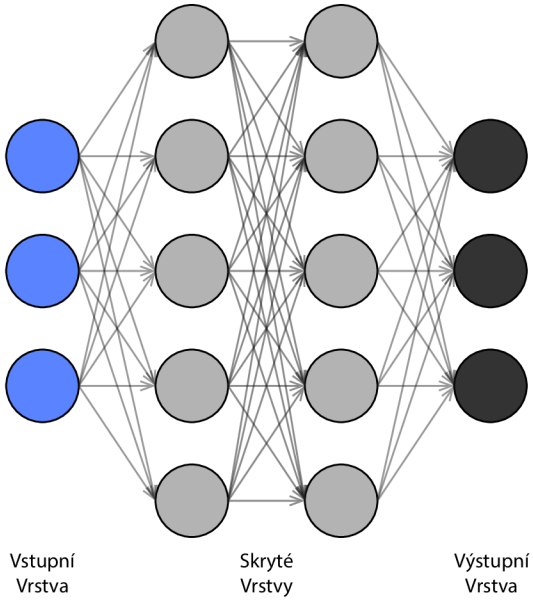
Neuronové sítě

Neuronové sítě (neural networks) jsou jedním z nástrojů strojového učení. Někdy se jim říká umělé neuronové sítě (artificial neural networks). Jejich název a chování je inspirováno lidským mozkem a jeho neurony, které posílají signály mezi sebou.

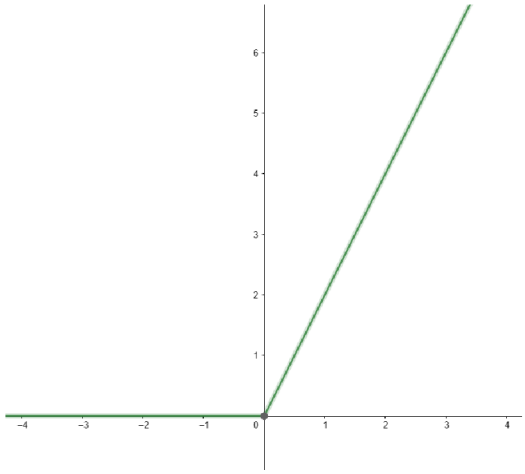
Nejjednodušším typem neuronů v umělých neuronových sítích je neuron s jedením výstupem a více vstupů z předešlých neuronů. Každý vstup má také přiřazenou váhu. Dalším parametr neuronu je práh, po jeho překročení je na výstup poslán výsledek přechodové funkce [6]. Topologie takové sítě je možné vidět na obrázku 3.4. Většinou se jako přechodová funkce používá ReLu (rectified linear unit), sigmoid nebo lineární. Fukce ReLu vypadá takto 3.5.

Vrstvy sítí

V umělých neuronových sítích jsou neurony uspořádány do jednotlivých vrstev. První vrstva je vstupní, poslední se nazývá výstupní a všechny vrstvy mezi nimi se obecně nazývají skryté



Obrázek 3.4: Základní struktura neuronové sítě



Obrázek 3.5: Relu (rectified linear unit) aktivační funkce

(hidden). Samotné vrstvy mohou být různých typů. Pokud je výstupy z předešlé vrstvy připojen na všechny vstupy neuronů následné vrstvy nazývá se Dense nebo plně připojená vrstva. Další běžná vrstva je vrstva Konvoluční. Tato vrstva se je schopná detekovat různých rysů ve vstupním vektoru a to nezávisle na poloze ve vektoru. Nejčastěji se využívá pro detekci nebo klasifikaci objektů v obrázcích.

Kapitola 4

Prostředí agenta

Prostředí je důležitou součástí posilovaného učení. Je to místo ve kterém agent provádí svoje akce. Prostředím může být reálný svět, ale většinou se používá umělé prostředí. Mezi virtuální prostředí patří všechno přes počítačovou hru, simulace reálného světa, simulace robotový systém. Já jsem jako své prostředí využil počítačovou hru Tuxánci.

4.1 Tuxánci

Tuxánci je akční hra pro jednoho až dva hráče inspirovanou českou hrou Bulánci [13]. Prostředí hry je 2D z vrchním pohledem kamery. Jedná se hru s licencí GNU GPL s dostupným zdrojovým kódem na veřejném repositáři[1]. Zdrojový kód je napsaný v jazyce C. Pro vykreslování grafiky a přehrávání zvuků využívá knihovnu SDL. Je dostupná pro operační systém Microsoft Windows a pro Linux. Hráč ovládá jednu postavu Tuxe (tučňáka). Tux se může pohybovat po mapě čtyřmi směry (nahoru, dolů, vpravo, vlevo). Daným směrem se začne pohybovat, jen pokud je na daný směr otočen. Pokud hráč zvolí směr, na který není postava otočená, zůstane na místě a otočí se na zvolený směr. Zápas vyhrává ten hráč, který první získá nastavené skóre (základně 15).

Předměty

Každý hráč má základní zbraň, která má pět nábojů v zásobníku. Po vyprázdnění zásobníku se základní zbraň začne přebíjet po dobu 2,5 sekund. V menu nastavení se dá jednotlivě zapnout dostupnost speciálních zbraní, min, granátu a speciální vylepšení. Seznam předmětu:

- **Dvojitá pistole** - Střílí dva souběžné projektily, které se pohybuji po linii
- **Brokovnice** - Střílí salvu pěti projektilů, které se oddalují od sebe
- **Samopal** - Vystřelí deset střel, pauza mezi jednotlivými vystřelí projektilů je 100 ms
- **Laser** - Vystřelí paprsek světla po dobu 0,5 sekund
- **Mina** - Položí minu pod Tuxe. Při další kolizi s Tuxem mina vybuchne
- **Bomba** - Hodí granát, který se při kolizi s objektem odrazí pod náhodným úhlem zpět. Pokud koliduje s hráčem vybuchne

[1] https://repo.or.cz/tuxanci.git

Obrázek 4.1: Rozehraný souboj ve hře Tuxánci na mapě for_a_good_night

- **Bonus Rychlost** - Zdvojnásobí rychlost pohybu Tuxe
- **Bonus Nekonečná munice** - Po dobu účinku se neodečítá munice
- **Bonus Nesmrtelnost** - Pokud by Tuxe s tímto bonusem měl zasáhnout projektil, přemístí se na jinou volnou část mapy
- **Bonus 1 střela do 4 stran** - Zbraně střílejí do všech 4 stran
- **Bonus neviditelnosti** - Funguje jen ve sítové hře a druhý hráč nevidí protivníka
- **Bonus Průchod zdí** - Tux může procházet skrz objekty a protivníka. Jeho střely také procházejí skrz objekty. A navíc při kontaktu s minou jí neaktivuje

Po smrti jakéhokoliv hráče se na náhodném místě objeví jeden z povolených bonusů/zbraní. Všechny bonusy trvají 25 sekund a všechny zbraně mají 5 výstřelů.

Mapy

Hra obsahuje pět základních map. Čtyři z nich jsou běžné celo-obrazové (800x600) arény a poslední je velká aréna(2500x2500) při které se pohybuje kamera společně s hráčem. Na mapě může být dva typy základních objektů buď s kolizí, nebo bez ní. Na mapě construction_site jsou požité trubky. Pokud projektil vstoupí do trubky, objeví se okamžitě na druhém konci trubky a pokračuje trajektorii ve směru konce trubky. Dalším prvek map jsou teleporty, které se nachází na dvou základních mapách. Jednou z funkcí teleportu je teleportování hráče na pozici jiného v pořadí dalšího teleportu na mapě. Sekundární funkcí je teleportování projektilů, které do něj vstoupí. Při výstupu projektilu z cílového portálu se náhodně zvolí jeho směr.

Uživatel je si schopen vytvořit vlastní mapu. Neexistuje editor, uživatel si ji musí ručně vytvořit. Výsledná mapa je uložená v archívu zip. V tomto archívu musí být textový soubor *arena.map*, další soubory jako textury a hudba jsou volitelné. V samotném *arena.map* souboru je definovaný název, pozadí a velikost mapy. Dále se zde definují textury a hudba, která se má načíst z archívu:
loadMusic file="in_a_captivity_of_8-bit.ogg"
loadImage file="bal3.nic.png"name="nic"alpha="no"
Pomocí příkazu *wall* je možné přidat na mapu kolizní objekt s texturou. Příkazem *teleport* se vytvoří jeden teleport (je potřeba definovat alespoň dva) a *pipe* vytvoří jedna strana trubky, u trubek se udává id druhé strany, takže je nutné vytvořit obě strany.

Obrázek 4.2: Seznam základních map

Uložené hry

Rozehraná hra je možná uložit v jakémkoliv stavu pomocí stlačení klávesy F2. Po zadání názvu souboru pro uložení se do něho nahrají následující informace:

- **Mapa/aréna** a její parametry. Mezi parametry patří maximální a současný počet kol.
- **Tuxi** a jejich momentální stav. Ukládá se jejich pozice, rotace, počet nábojů, skóre, vlastněná zbraň, ID, stav a název.
- **Projektily** a všechny jejich informace.
- **Předměty** - U nich se ukládá jejich pozice a typ

Kapitola 5

Implementace

Pro implementaci neuronových sítí jsem si zvolil knihovnu TensorFlow. TensorFlow je open source framework pro strojové učení vyvíjený společností Google. Prostředí hra Tuxanci jsem upravil pro potřeby posilovaného učení. Tuxanci i moje rozšíření je implementované v jazyce C.

5.1 Prostředí

Důležitou součástí je vytvoření ovládání prostředí pro potřeby posilovaného učení. Základní smyčka probíhá minimální každý 50 milisekund, pokud smyčka trvá déle, tak další iterace proběhne po jejím ukončení. Smyčku jsem upravil tak aby čekala na příjem paketu z klienta. Tuxánci fungují jako server, takže čekají na zprávu od klienta. Každý požadavek je označený unikátním číslem. Klient může poslat jeden z následujících požadavků:

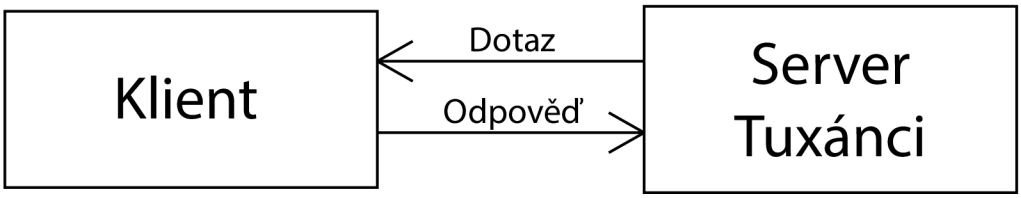
- **Načtení uložené hry (0).** Je nutné poslat název souboru s uloženou hrou.
- **Požádání o poslání aktuálního stavu (1).** Vrátí informace obou hráčů (pozici, rotaci, počet nábojů, typ zbraně a stav Tuxe jestli je naživu), všech neprůchodných stěn (pozici, výška a šířka), všech projektilů (pozice, směr a velikost) a předmětů (pozice, velikost a typ).
- **Odeslání akce (2)**, kterou má agent vykonat a následně čeká na vrácení nového stavu a odměny. Nebo odeslání dvou akcí, každou pro jednoho z agentů
- **Ukončení hry (4)**
- **Požádání o zadání dalšího lidského vstupu (5).** Tento požadavek slouží pro nahrávání zkušeností pomocí lidského hráče.

Pro ušetření času jsem vypnul vykreslování grafiky hry na obrazovku. Pro spouštěni programu s prostředím je nutné zadání příkazu *tuxanci*. Program se může spustit s těmito parametry:

- **-RL** spustí se verze připravená pro komunikaci s klientem a čeká na akce.
- **-Screen** spustí se verze s vykreslováním obrazovky
- **-Control** zajistí, že hra čeká na zadání akce od hráče a následně jí i se stavem pošle klientovy.

5.2 Komunikace s prostředím

Prostředí a klient kde probíhá učení agenta, jsou dva oddělené programy/procesy. Kvůli tomu je potřeba zajistit komunikaci mezi nimi. Já jsem se rozhodl pro komunikaci pomocí síťového soketu. Existuje mnoho implementací sítových soketů. Jelikož hra Tuxánci je napsaná jazyce C a klient pomocí pythonu, kvůli tomu jsem si zvolil multijazykovou knihovnu ZeroMQ, která má API v obou jazycích. Má řadu jazykových API a běží na většině operačních systémů. Tato knihovna poskytuje sokety, které přenášejí atomické zprávy přes různé přenosy, jako jsou mezi-procesové, TCP a vícesměrové vysílání. Komunikace je implementovaná jednoduchým vzorem žádostí a odpovědí (Request-Reply) viz obrázek 5.1. Server je implementovaný na straně hry Tuxánci.



Obrázek 5.1: Diagram síťové komunikace mezi Tuxánci a klientem

5.3 Reprezentace stavu

Přijaté informace o prostředí jsou přeformátované do 2D tensoru o velikost 80x60 s hodnotami v rozsahu $< -1, 5 >$. Pozadí mapy je reprezentováno pomocí 0. Všechny neprůchodné objekty jsou reprezentované pomocí hodnotou 1. Jelikož originální velikost prostředí desekrát větší než jeho reprezentace (800x600) to znamená ,že u objektů, které nezabírají celý 10x10 čtverec je výsledná hodnota h vypočtená následně:

$$h = p/100 \tag{5.1}$$

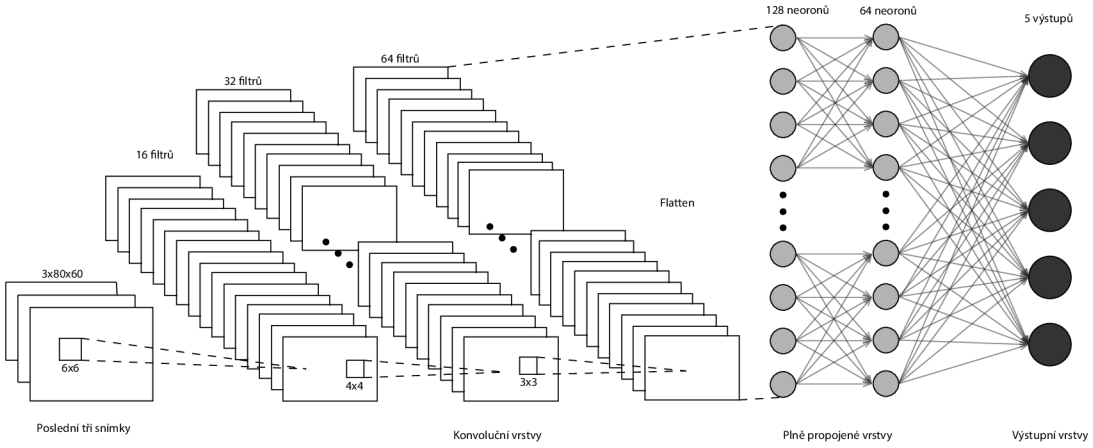
Kde p je počet pixelů, které obsahují objekt. Agentem ovládaný Tux je reprezentován 3x3 pixely s hodnotou 2. Nepřátelský Tux je reprezentován taktéž 3x3 pixely ale s hodnotou 3. U obou se podle směru, na který je Tux otočený, vymění tři přední pixely na hodnotu -1. Jako poslední jsou projektily, které mají hodnotu pixelu 5. Vizuální ukázka viz obrázek A.1. Celý stav je pak reprezentovaný jako tensor s posledními třemi takovými snímky.

5.4 Neuronové sítě

Topologii sítě je na obrázku 5.2. Inspiroval jsem se následující řešením [3]. Vstupem je tensor s tvarem 3x80x60, který obsahuje poslední tři stavy. Následují tři konvoluční vrstvy. První z nich má 16 filtrů s oknem 6x6 a s kroky 4x4. Druhá vrstva má 32 filtrů s výběrovým oknem 4x4 a s kroky 2x2. Poslední konvoluční vrstva má 64 filtrů, její výběrové oknem má velikost 3x3 a má kroky velikosti 1x1. Následující vrstva je Flatten, která výsledek konvolučních vrstev převede do jednorozměrného vektoru. Vrstva Flatten je nutná pro správné fungování následující Dense vrstvy, která vyžaduje 1D vektor. První Dense vrstva má 128 neuronů, které mají jako svou aktivační funkci ReLu. Následující Dense vrstva má poloviční počet

neuronů 64 a také se aktivují podle funkce ReLu. Poslední vrstva má pět neuronů s lineární aktivací.

Hodnoty výstupů těchto neuronů korespondují s jednotlivými akcemi. První neuron určuje zda se má zvolit akce pohyb nahoru, druhý pohyb dolů, třetí pohyb vpravo, čtvrtý pohyb vlevo a pátý určuje zda má Tux vystřelit.



Obrázek 5.2: Topologie implementované neuronové sítě

5.5 Prioritized experience replay

Prioritized experience replay jsem implementoval pomocí prioritní fronty. Přesněji jsem využil stukturu Max Binary heap, která je časově nenáročná. Max Binary heap zaručuje, že kořen stromu má vždy největší hodnotu, a že potomci uzlu jsou stejné nebo menší. Max Binary heap jsem implementoval jako binární strom, který je uložen v lineárním poli. Pole je přibližně sestupně seřazené, což je pro naše účely vyhovující, protože je důležitější časová náročnost než přesnost. Časová náročnost vložení i aktualizace je $O(log\ n)$. Aby nedošlo přílišné nevyváženosti stromu, tak každých 10 epizod proběhne sestupné seřazení pole. Další optimalizaci jsem provedl tím, že jsem samotné data zkušeností jsem uložil do samostatného neměnícího pole a v Max Binary heap jsem uložil jen jejich index. To zajistí menší náročnost při seřazení.

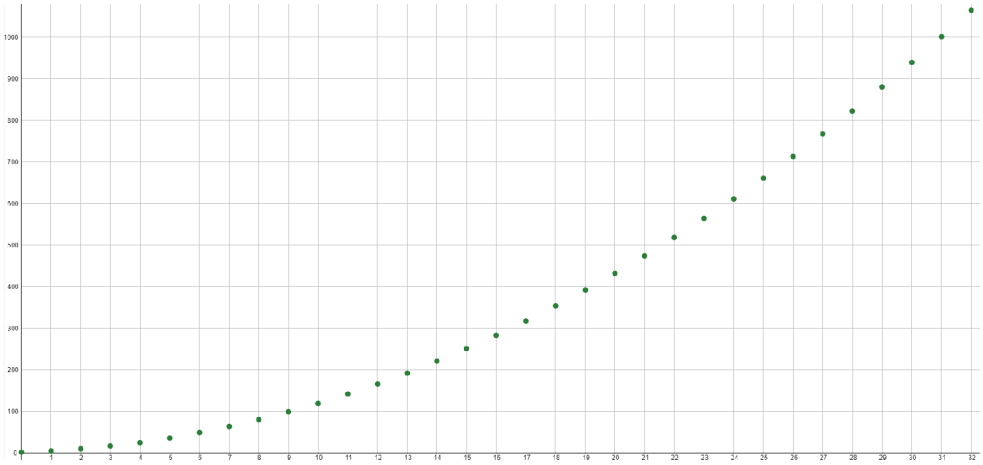
Vybíraní skupiny uložených probíhá následovně. Celkový rozsah se rozloží na n úseků, kde n je požadovaná velikost skupiny. Rozdělení na úseky není rovnoměrné. První úsek bývá n-krát menší než poslední. Názorně zobrazené na obrázků 5.3. Z každého úseků se následně náhodně vybere jeden záznam.

5.6 Algoritmus

Vybral jsem si a implementoval jsem algoritmus Double DQN s Prioritized experience replay. Při učení využívá Prioritized experience replay, který je implementovaný v *PER.py*. Tento algoritmus jsem následně upravil pro potřeby MARL a tato implementace je uložená v souboru *muliagentDQN.py*. Hlavní změnou je vyhodnocení akcí pro oba agenty a posílání a příjem dvou odměn.

Algoritmus 3: `Double DQN s PER`

Inicializace upřednostněné záznamové paměti pro minulé stavy
Vytvoření modelu neuronové sítě a náhodná inicializace jeho vah
Zkopírování modelu neuronové sítě jejích vah, která je známá jako cílová síť
for *Každou episodu* **do**
    Restartování prostředí
    Inicializace prvního stavu s
    while *Episoda neskončila* **do**
        Vybrání akce a pomocí modelu $a = argmax(Q(s))$ nebo náhodná s pravděpodobností ϵ
        Provedení akce prostředím
        Získání nového stavu s' a odměny r
        Uložení zkušenost jako čtveřice (s, a, r, s') do paměti
        Náhodně vybrat soubor i zkušeností (s, a, r, s') z paměti
        Vyhodnocení souboru zkušeností $v_i = Q(s_i)$
        $t_i = r_i$ pro koncové stavy
        $t_i = r_i + \gamma \cdot Q'(s', argmax(Q(s')))$ pro ostatní
        Trénování se stavy s_i jako daty a t_i jako jejich štítky
    end
    if *Každých 10 epizod* **then**
        Zkopírování vah z hlavní do cílové sítě
        Seřazení upřednostněné záznamové paměti
    end
end for



Obrázek 5.3: Velikost úseků ve výběru zkušeností pro $n = 32$ a počet všech záznamů je 1000

Kapitola 6

Experimenty

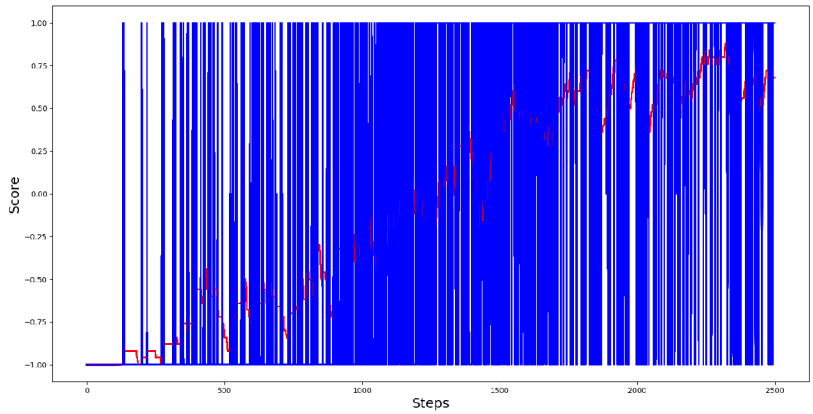
Pro své experimenty jsem použil prostředí, které jsem pro tyto účely upravil. Cílem experimentů bylo zjistit, zda implementovaný algoritmus Double DQN je schopný natrénovat agenta v jednotlivých případech, ve kterých jsou různě definované odměny. Natrénovaný model by měl zvládnout vyhýbání se projektilům a zabití nepřátele. Při provádění experimentů jsem nastavil pamět pro ukládání zkušeností na maximální kapacitu 200000 vzorků a trénování vždy probíhalo s 32 vzorcích z paměti. Hodnotou discount faktoru jsem nastavil na hodnotu 0,98, zkoušel jsem i nižší hodnoty, ale u nich bylo učení neefektivní. Parametr rychlost učení jsem nastavil na 0,001. Ve všech experimentech mají agenti jen základní zbraně a na mapě se žádné bonusy ani předměty neobjevuji. Dalším společným rysem skrz všechny experimenty je použití zákldaní mapy tuxánků „for a good night“.

6.1 Hraní proti AI

V prvním experimentu jsem testoval zda se agent naučí hrát proti umělé inteligence hry, která je vcelku primitivní. Úkolem agenta bylo zabití nepřátelského hráče a za to obdržel pozitivní odměnu +1. Pokud však byl zabit nepřítelem odměna byla negativní a to -1 . Při začátku nové episody byl vždy nahrána stejná uložená hra. Toto zaručilo, že agent začínal na stejném místě na mapě. Jak jde vidět na obrázku 6.1 , agent se úspěšné ve většině epizod získal pozitivní odměnu již po 1500 epizod.

6.2 Zabíjení statického cíle v náhodných pozicích

V tomto experimentu jsem testoval častější rozdávání odměn. Úkolem agenta bylo zabití nepřátelského hráče a to buď vlastním projektilem nebo dotknutí se ho. Za každý pohyb směrem k protivníkovy obdržel pozitivní odměnu +0,1. Pokud se však vzdálil od cíle byl potrestán negativní odměnou a to v hodnotě -0,5. Pokud však byl schopen zabit nepřítele odměna +1. Pozice Tuxů se nezměnila dokud agent nedosáhl pozitivní průměrné odměny za posledních 50 episod. Poté se náhodně vybrala nová pozice. Jak jde vidět na obrázku 6.2, agent úspěšné dosáhl cíle průměrně za 250 epizod. V druhém obrázku 6.3 je dlouhá stagnující část, která trvala přes 1250 epizod. Důvodem pro tuto stagnaci byla překážky, která stála v cestě. Agent byl nucen překážku obejít a jelikož vzdalování se od cíle způsobí velké negativní odměny, trvalo učení delší dobu.



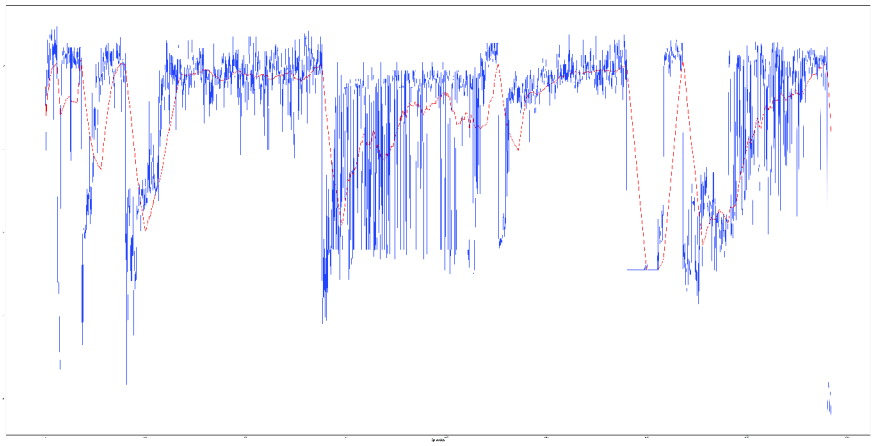
Obrázek 6.1: Graf zobrazuje průběh získaných odměn za každou epizodu (modrá) a průměr za posledních 50 epizod (červená) při učení modelu proti AI.

6.3 Hraní proti AI s pomocí IL

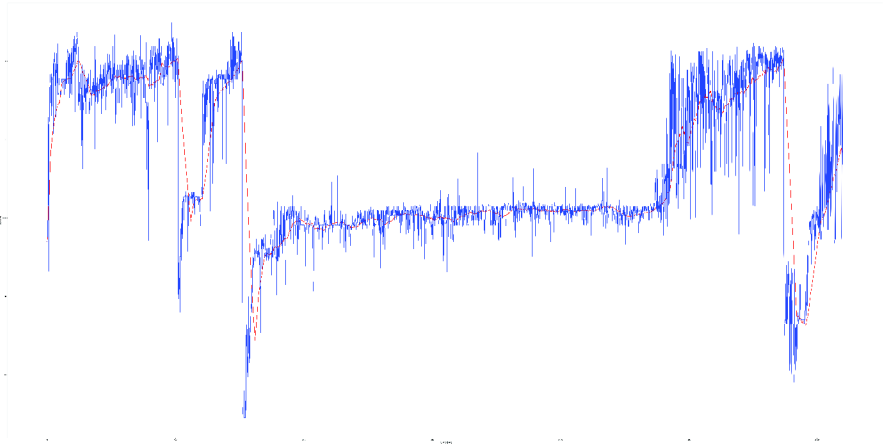
Tímto experimentem jsem chtěl otestoval zda se agent naučí rychleji zabit nepřítele, který je ovládaný pomoci umělé inteligence hry pokud využije zkušenosti, které jsem nahrál já. Úkolem agenta bylo zabití nepřátelského hráče a to vlastním projektilem. Pokud tak učinil obdržel odměnu +1 pokud však byl zabit protivníkem odměna byla -1. Na začátku trénování byly obě neuronové sítě (hlavní, cílová) inicializovány stejnými hodnotami vah. Stejně jak v prvním experimentu se pozice Tuxů se nezměnila. Obrázek 6.4 ukazuje agenta, který úspěšné získal ve většině epizod pozitivní odměnu již po 200 epizod. Při učení tohoto agent bylo využito nahraných zkušeností. Další obrázek 6.5 ukazuje výsledek agenta, který se učil podle ϵ-greedy policy bez žádných předehraných zkušeností. S tímto agentem se odměna maximalizovala asi o 100 epizod později.

6.4 Multiagent self-play

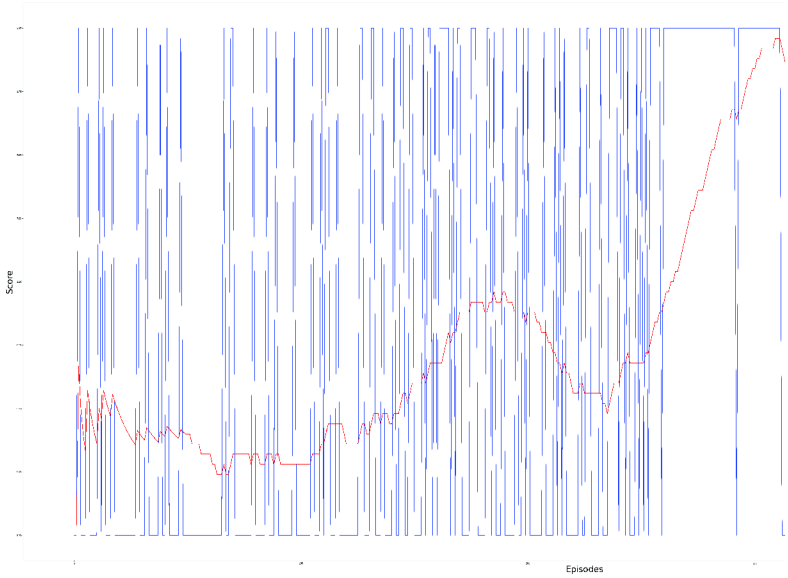
Další experiment se zabývá učení modelu pomoci dvou soupeřících agentům. Tito agenti se učí navzájem. Sdílí spolu paměť událostí, které zažili. Úkolem agenta bylo zabití nepřátelského hráče a to vlastním projektilem. Pokud tak učinil obdržel odměnu +1 pokud však byl zabit protivníkem odměna byla -1. Pokud by se agenti nezabili do konce epizody (v tomto případě to bylo 100 kroků), tak by oba dostaly stejnou negativní odměnu byla -1. Obrázek 6.6 ukazuje shluknuté odměny obou agentů a to tak, že pokud oba dostali zápornou odměnu výsledek je -1, pokud však jeden obdržel odměnu pozitivní výsledná hodnota na grafu je 1. Z grafu lze vyčíst, že po 500 episodách průměrně každá druhá skončí smrtí jednoho z agentů.



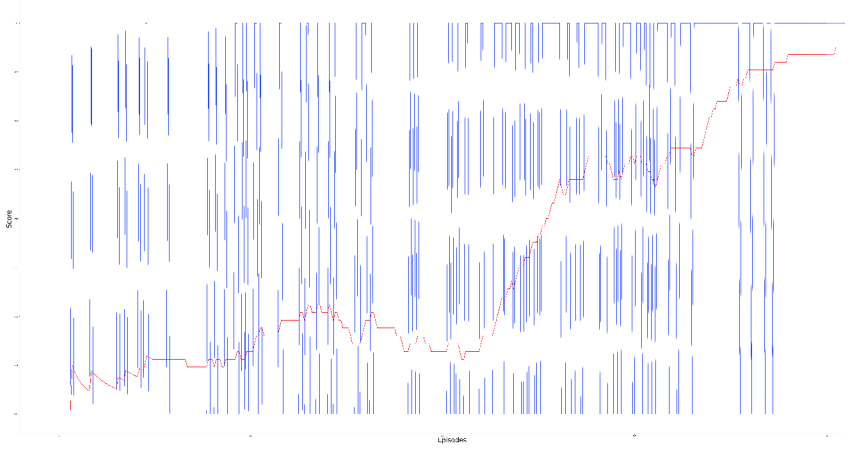
Obrázek 6.2: Graf zobrazuje průběh získaných odměn za každou epizodu (modrá) a průměr za posledních 50 epizod (červená) při učení modelu na zabíjení statického cíle.



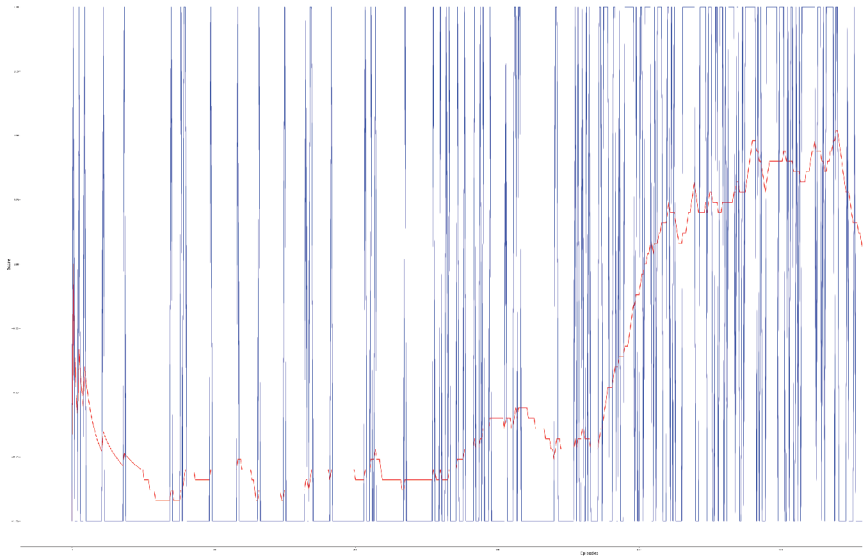
Obrázek 6.3: Graf zobrazuje průběh získaných odměn za každou epizodu (modrá) a průměr za posledních 50 epizod (červená). Tento graf je výsledkem učení modelu pro zabíjení/přibližování se k statickému cílů.



Obrázek 6.4: Tento graf obsahuje průběh získaných odměn za každou epizodu (modrá) a průměr za posledních 50 epizod (červená) při učení modelu na zabíjení statického cíle s použitím imitation learning.



Obrázek 6.5: Graf zobrazuje průběh získaných odměn za každou epizodu (modrá) a průměr za posledních 50 epizod (červená). Tento graf je výsledkem učení modelu pro zabíjení/přibližování se k statickému cílů.



Obrázek 6.6: Graf zobrazuje průběh získaných pozitivních odměn za každou epizodu (modrá) a průměr za posledních 50 epizod (červená). Tento graf je výsledkem učení modelu pro Multiagent self-play.

Kapitola 7

Závěr

Tato práce se zaměřila na posilované učení společně s hlubokými neuronovými sítěmi. Rozebral jsem základy imitation learning a hlavně jsem popsal fungování Q učení. Popsal jsem algoritmus DQN a Double DQN, který jsem využil při implementaci.

Cílem práce bylo vytvořit agenta, který bude ovládat postavu Tuxe. Tento agent měl být schopný se pohybovat v prostředí a zabíjet protivníka. A dále tento agent měl soupeřit s dalším agent, který se v prostředí nachází.

Implementoval jsem komunikaci mezi mnou upraveným prostředím a klientem. Vytvořil jsem komunikační interface pro hru Tuxanci, který slouží pro účely strojového učení. Na experimentech jsem ukázal fungování učícího modelu.

Do budoucna by bylo možné naučit agenty sbíraní a používání předmětů, zbraní a bonusů. Také by bylo potřeba další čas na učení multi-agent modelu. Dalším možným rozšířením může být náhodná generace map, nebo alespoň jejich střídání.

Literatura

[1] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I. et al. *Playing Atari with Deep Reinforcement Learning* [online]. arXiv, 2013. Dostupné z: https://arxiv.org/abs/1312.5602.

[2] Moghadam, P. H. *Deep Reinforcement learning: DQN, Double DQN, Dueling DQN, Noisy DQN and DQN with Prioritized Experience Replay* [online]. 2019 [cit. 2022-10-05]. Dostupné z: https://medium.com/@parsa_h_m/deep-reinforcement-learning-dqn-double-dqn-dueling-dqn-noisy-dqn-and-dqn-with-prioritized-551f621a9823.

[3] AL Ma'amari, M. *Deep Reinforcement Learning With Python* [online]. 2020 [cit. 2022-09-05]. Dostupné z: https://towardsdatascience.com/deep-reinforcement-learning-with-python-part-2-creating-training-the-rl-agent-using-deep-q-d8216e59cf31.

[4] Seita, D. *Combining Imitation Learning and Reinforcement Learning Using DQfD* [online]. 2019 [cit. 2022-09-05]. Dostupné z: https://danieltakeshi.github.io/2019/04/30/il-and-rl/.

[5] Zhang, K., Yang, Z. a Başar, T. *Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms*. 2019 [cit. 2019-10-02]. Dostupné z: https://arxiv.org/abs/1911.10635.

[6] Durčák, P. *Neuronové sítě a princip jejich fungování* [online]. 2017 [cit. 2022-08-05]. Dostupné z: https://www.napocitaci.cz/33/neuronove-site-a-princip-jejich-fungovani-uniqueidgOkE4NvrWuNY54vrLeM670eFNQh552VdDDulZX7UDBY/.

[7] Seita, D. *Understanding Prioritized Experience Replay* [online]. 2019 [cit. 2022-09-05]. Dostupné z: https://danieltakeshi.github.io/2019/07/14/per/.

[8] SALLOUM, Z. *Policy Based Reinforcement Learning, the Easy Way* [online]. 2019 [cit. 2022-10-05]. Dostupné z: https://towardsdatascience.com/policy-based-reinforcement-learning-the-easy-way-8de9a3356083.

[9] Kansal, S. a Martin, B. *Reinforcement Q-Learning from Scratch in Python with OpenAI Gym* [online]. 2018 [cit. 2022-09-05]. Dostupné z: https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/.

[10] Bonse, R., Kockelkorn, W., Smelik, R., Veelders, P. a Moerman, W. *Learning Agents in Quake III* [online]. 2008 [cit. 2022-10-05]. Dostupné z: https://www.researchgate.net/publication/250809440_Learning_Agents_in_Quake_III#pf9.

[11] Koďousková, B. *CO JE STROJOVÉ UČENÍ A JAK SOUVISÍ S UMĚLOU INTELIGENCÍ?* [online]. 2021 [cit. 2022-09-05]. Dostupné z: https://www.rascasone.com/cs/blog/strojove-uceni-ml-metody-klasifikace.

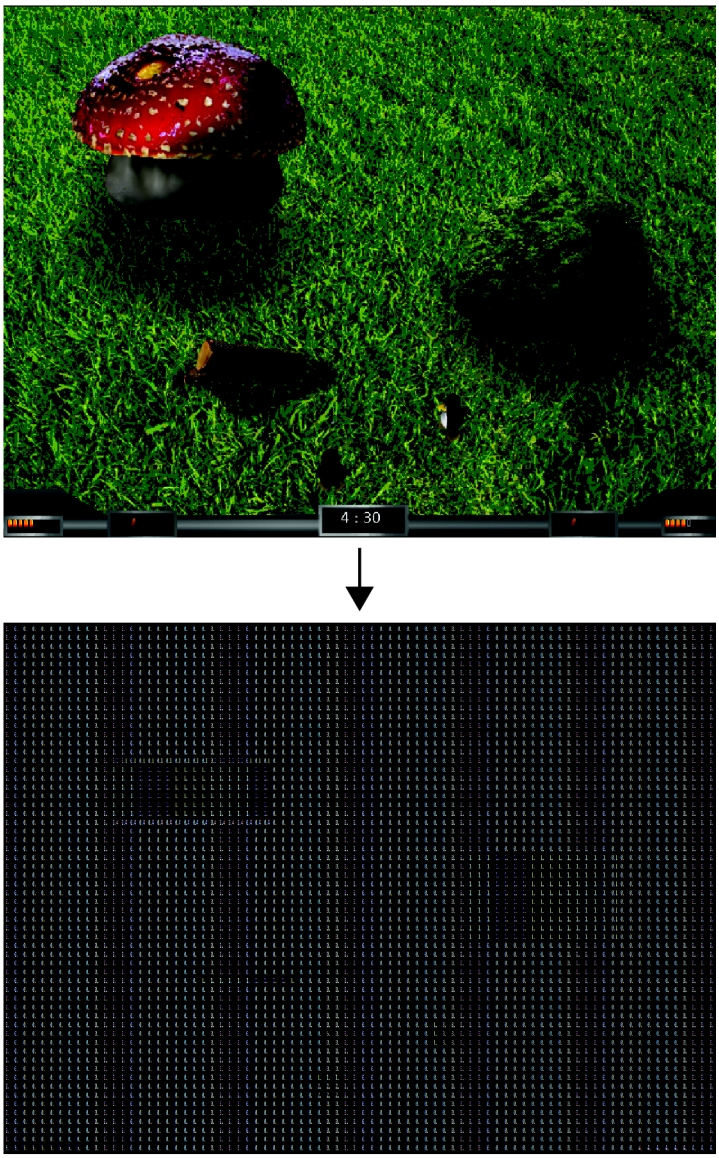
[12] Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A. et al. *Grandmaster level in StarCraft II using multi-agent reinforcement learning* [online]. 2019 [cit. 2022-10-05]. Dostupné z: https://www.nature.com/articles/s41586-019-1724-z.

[13] VRLÍK, F. *Tuxánci: předělávka slavných Bulánků* [online]. 2007 [cit. 2022-10-05]. Dostupné z: https://www.root.cz/clanky/tuxanci-predelavka-slavnych-bulanku/.

[14] Volný, A. *Deep Reinforcement Learning in Complex Structured Environments.* Praha, CZ, 2018. Diplomová práce. Czech Technical University in Prague Faculty of Electrical Engineering Department of Computer Science. Dostupné z: https://dspace.cvut.cz/bitstream/handle/10467/76484/F3-DP-2018-Volny-Adam-thesis.pdf?sequence=-1&isAllowed=y.

Příloha A

Reprezentace stavu



Obrázek A.1: Názorná ukázka reprezentace jednoho snímku hry