**ŽILINSKÁ UNIVERZITA V ŽILINE**

FAKULTA RIADENIA A INFORMATIKY

BAKALÁRSKA PRÁCA

Jakub Dzurovčin

**Algoritmus učenia posilňovaním v úlohe sprievodcu**

Vedúci práce: Ing. Marek Baláž

Registračné číslo: 1393/2020

Žilina, 2021

**ŽILINSKÁ UNIVERZITA V ŽILINE**

FAKULTA RIADENIA A INFORMATIKY

BAKALÁRSKA PRÁCA

ŠTUDIJNÝ ODBOR:

Informatika

Jakub Dzurovčin

**Algoritmus učenia posilňovaním v úlohe sprievodcu**

Žilinská univerzita v Žiline

Fakulta riadenia a informatiky

Katedra matematických metód a operačnej analýzy

Žilina, 2021

****Čestné Vyhlásenie****

Čestne prehlasujem, že som prácu vypracoval samostatne s využitím dostupnej literatúry a vlastných vedomostí. Všetky zdroje použité v bakalárskej práci som uviedol v súlade s predpismi.

Súhlasím so zverejnením práce a jej výsledkov.

...........................................

V Žiline, dňa 25.04.2021 Jakub Dzurovčin

Poďakovanie

„*Moje poďakovanie patrí vedúcemu bakalárskej práce Ing. Marekovi Balážovi za odbornú pomoc, dobré rady, pripomienky a usmerňovanie pri tvorbe práce.*“

ABSTRAKT

„Dzurovčin, Jakub: *Algoritmus učenia posilňovaním v úlohe sprievodcu*. [Bakalárska práca] – Žilinská univerzita v Žiline, Fakulta riadenia a informatiky, Katedra matematických metód a operačnej analýzy. – Vedúci: Ing. Marek Baláž. – stupeň odbornej kvalifikácie: Bakalár v odbore Informatika. Žilina: FRI ŽU v Žiline, 2021.

Cieľom tejto bakalárskej práce je vytvoriť asistenta pre hranie hry Pacman pomocou agenta založeného na algoritme pre učenie posilňovaním. Teoretická časť obsahuje informácie potrebné pre pochopenie, vytvorenie a zlepšenie A2C agenta. To zahŕňa hlboké neurónové siete, princíp ich fungovania a optimalizácie, konvolučné neurónové siete pre spracovanie obrazu a prístupy pre učenie posilňovaním pomocou metód pre predikciu Q hodnoty a metód pre predikciu politky. Teoretická časť Ďalej obsahuje podrobný popis A2C agenta a možnosť jeho vylepšenia na PPO agenta. V praktickej časti je popísaná implementácia zvoleného agenta v jazyku Python, jeho trénovanie a aplikácia kde je agent implementovaný ako asistent.

**Kľúčové slová:** asistent, agent, A2C, hlboké neurónové siete, učenie posilňovaním

ABSTRAKT

„Dzurovčin, Jakub: *Algoritmus učenia posilňovaním v úlohe sprievodcu*. [Bakalárska práca] – Žilinská univerzita v Žiline, Fakulta riadenia a informatiky, Katedra matematických metód a operačnej analýzy. – Vedúci: Ing. Marek Baláž. – stupeň odbornej kvalifikácie: Bakalár v odbore Informatika. Žilina: FRI ŽU v Žiline, 2021. ... oof aj toto treba preložiť ? :D

Our goal in this bachelors thesis is to create an assistant for playing Pacman. Assistant is based on reinforcement learning agent. Theoretical part of this thesis contain information needed to understand, create and upgrade A2C agent. This includes deep neural networks, their mechanisms of function and optimalization, as well as convolutional neural networks for image processing and reinforcement learning approaches – Q value based methods and policy based methods. After which we explain A2C agent in detail and possible upgrade to PPO agent. In practical part we will describe our implementation of A2C agent using Python programming language, training and application with agent implemented as an assistant.

**Kľúčové slová:** assistant, agent, A2C, deep neural networks, reinforcement learning

Obsah

[ZOZNAM OBRÁZKOV 9](#_Toc69573193)

[Úvod 10](#_Toc69573194)

[1 Súčasný stav riešenia problematiky 11](#_Toc69573195)

[1.1 Strojové učenie 11](#_Toc69573196)

[1.1.1 Učenie bez učiteľa 11](#_Toc69573197)

[1.1.2 Učenie s učiteľom 11](#_Toc69573198)

[1.1.3 Učenie posilňovaním 11](#_Toc69573199)

[1.2 Učenie posilňovaním 12](#_Toc69573200)

[1.2.1 Základný princíp 12](#_Toc69573201)

[1.2.2 Tabuľkové metódy 12](#_Toc69573202)

[1.2.3 Hlboké neurónové siete 15](#_Toc69573203)

[1.2.4 Konvolučné neurónové siete 16](#_Toc69573204)

[1.2.5 Algoritmy využívajúce neurónové siete 18](#_Toc69573205)

[1.2.6 DQN 18](#_Toc69573206)

[1.2.7 Metódy predikujúce politiku 20](#_Toc69573207)

[1.2.8 A2C 23](#_Toc69573208)

[1.2.9 A3C 24](#_Toc69573209)

[1.2.10 PPO 24](#_Toc69573210)

[1.3 Knižnice pre neurónové siete 26](#_Toc69573211)

[1.3.1 TensorFlow 26](#_Toc69573212)

[1.3.2 Pytorch 27](#_Toc69573213)

[2 algoritmus a2c 28](#_Toc69573214)

[2.1 Agent 28](#_Toc69573215)

[2.1.1 Učenie 28](#_Toc69573216)

[3 Trénovanie algoritmu 32](#_Toc69573217)

[3.1 Prostredie 32](#_Toc69573218)

[3.2 Pracovník 35](#_Toc69573219)

[3.3 Sieť 36](#_Toc69573220)

[3.4 Trénovanie 38](#_Toc69573221)

[4 Program 43](#_Toc69573222)

[4.1 Hrací Agent 45](#_Toc69573223)

[4.2 Hra 45](#_Toc69573224)

[4.3 Konzultant 45](#_Toc69573225)

[4.4 Graf istoty 45](#_Toc69573226)

[4.5 Aplikácia 46](#_Toc69573227)

[Záver 47](#_Toc69573228)

[Zoznam použitej literatúry 48](#_Toc69573229)

[Prílohy 50](#_Toc69573230)

[Príloha A: 51](#_Toc69573231)

[Príloha B: 52](#_Toc69573232)

[Príloha C: 53](#_Toc69573233)

[Príloha D: Obsah Pamäťová karta 54](#_Toc69573234)

# ZOZNAM OBRÁZKOV

Obrázok 1.1:Vizualizácia systému pre učenie posilňovaním...............................................12

Obrázok 1.2: Vizualizácia Q – tabuľky................................................................................13

Obrázok 1.3: Hlboká neurónová sieť...................................................................................15

Obrázok 1.4: Aplikácia filtra................................................................................................17

Obrázok 1.5: Aplikácia poolingu.........................................................................................17

Obrázok 3.1: UML systému pre trénovanie agenta..............................................................32

Obrázok 3.2: Stav prostredia ............................................................................................33

Obrázok 3.3: Transformovaná obzervácia .......................................................................35

Obrázok 3.4: Dizajn neurónovej siete..................................................................................37

Obrázok 3.5: Prvotné skóre dosiahnuté prvým agentom.....................................................38

Obrázok 3.6: Priebeh tréningu s nevhodnými parametrami (modré) 1................................39

Obrázok 3.7: Priebeh tréningu s nevhodnými parametrami 2..............................................39

Obrázok 3.8: Priebeh tréningu s nevhodnými parametrami 3..............................................40

Obrázok 3.9 Najlepšie skóre dosiahnuté prvým agentom....................................................40

Obrázok 3.10: Porovnanie agent 1 (oranžová) vs agent 2 (modrá)......................................41

Obrázok 3.11: Kompletne natrénovaný agent 2...................................................................41

Obrázok 4.1: UML Asistovaného systému..........................................................................43

Obrázok 4.2: GUI Asistovaného systému ..........................................................................44

# Úvod

**„Moorov zákon** je empirické pravidlo, že zložitosť integrovaných obvodov (počet tranzistorov integrovaných na nich) sa zdvojnásobuje približne každé dva roky“ [1]. Toto empirické pravidlo platilo posledných 5 dekád a jeho dôsledkom je obrovský nárast výkonu naších počítačov a aplikácií pre ktoré ich využívame.

V tejto práci priblížime strojové učenie, oblasť štúdie algoritmov ktoré sa dokážu učiť nové veci a zlepšovať sa vo vykonávaní svojej činnosti.

V teoretickej časti tejto práce popíšeme strojové učenie, jeho základné princípy, postupy ktorými sa dá dosiahnuť učenie. Bližšie sa pozrieme na učenie posilňovaním (Reinforcement learning) pomocou umelých neurónových sieti (Artificial neural networks), ich výhody, nevýhody a konkrétne algoritmy. Zbežne popíšeme druhy týchto algoritmov a podrobne vysvetlíme princíp fungovania Advantage Actor Critic (A2C) algortitmu. V závere teoretickej časti spomenieme novšie a výkonnejšie algoritmy ktoré su vylepšením A2C.

V praktickej časti rozanalyzujeme implementáciu A2C algoritmu a jeho vlastnosti. Popíšeme postup pri trénovaní takéhoto algoritmu a vytvoríme jednoduchú aplikácie na ktorej ukážeme možný príklad využitia natrénovaného algoritmu ako asistenta ktorý nám bude radiť pri hraní Pacmana.

# Súčasný stav riešenia problematiky

## Strojové učenie

Strojové učenie je zastrešujúci pojem pre rodinu algoritmov ktoré sú schopné plniť svoju danú úlohu bez toho aby mali explicitne naprogramovaný postup pre jej plnenie.

Pre dosiahnutie tohto cieľa oblasť strojového učenia popisuje metódy a algoritmy ktoré sa dokážu učiť z poskytnutých dát a zlepšovať svoju presnosť pri plnení danej úlohy.

V čase písania tejto práce existujú 3 základné metódy ako pristupovať k učeniu.

### Učenie bez učiteľa

Pri učení bez učiteľa su použíté neoznačené dáta. Algoritmus nevie čo tieto dáta predstavujú. Podľa aplikácie jeho úlohou môže byť:

* Hľadanie spojitosti a štruktúry ktorá sa v daných dátach nachádza.
* Identifikácia výnimiek skrývajúcich sa v daných dátach.

Algoritmy využivajúce túto metódu sa použivajú napríklad v marketingu kde dokážu identifikovať zákazníkov s podobnými atribútmi.

### Učenie s učiteľom

Učenie s učiteľom využíva označené dáta, kde algoritmus dostane konkretný údaj aj s „popisom“ čo daný údaj predstavuje. Algoritmus spracúva tieto dáta a predikuje čo tieto dáta predstavujú. Po vytvorení predikcie svoj odhad porovná s nálepkou a upraví svoj vnútorný model. Po natrenovaní dokáže tento algoritmus robiť predikcie aj pre neoznačené dáta. Učenie s učiteľom sa používa v aplikáciach kde sa na základe historických dát dajú predpovedať budúce udalosti.

### Učenie posilňovaním

Pri učení posilňovaním je algoritmu poskytnuté prostredie v ktorom dokáže vykonávať akcie. V takomto prostredí algoritmus pomocou metódy pokus omyl vykonáva akcie za ktoré je odmenený a jeho cieľom je maximalizovať tieto odmeny.

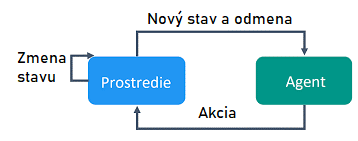
## Učenie posilňovaním

### Základný princíp

Pri učení posilňovaním je našim cieľom vytvoriť algoritmus - agenta ktorý vie vybrať najlepšiu možnú akciu z množiny všetkých možných akcií v danom prostredí.

Pre dosiahnutie tohto cieľa musí mať agent prístup k prostrediu v ktorom može vykonávať kroky. Krok zvyčajne pozostáva z akcie () vybranej agentom ktorá po vstupe do prostredia vyprodukuje stav () čo je aktuálny stav nášho prostredia a odmena () za vykonanie tejto akcie.

Počítačové hry majú častokrát systém v ktorom hráč vykonáva akcie nad herným prostredím a úspešnými akciami zvyšuje svoje skóre. Preto sú hry ukážkovým prostredím pre vývin a testovanie agentov využívajúcich tento princíp učenia.



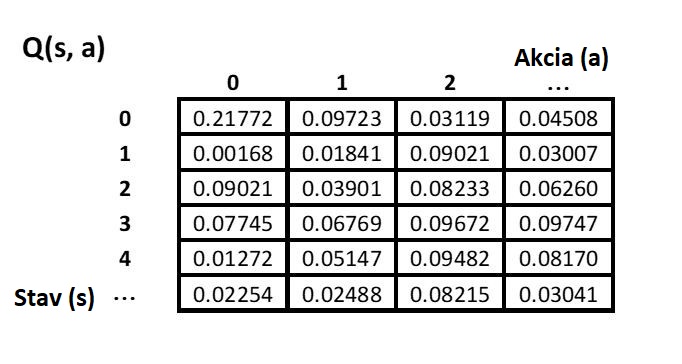
Obrázok 1.1: Vizualizácia systému pre učenie posilňovaním

### Tabuľkové metódy

Existuje viac prístupov k učeniu posilňovaním. Jedným z nich sú tabuľkové metódy. Základom tabuľkových metód je takzvaný algoritmus Q učenia ktorý na učenie využíva Q hodnotu.

**Algoritmus Q učenia**

Hlavným komponentom algoritmu Q učenia je Q tabuľka, čo je dvojrozmerné pole tvaru ktoré sa na začiatku inicializuje – zvyčanie na 0.



Obrázok 1.2: Vizualizácia Q – tabuľky

Agent začína v stave () kde vykoná akciu () za ktorú dostane odmenu ().

Vyberanie akcie môže nastať dvojako: využitím Q tabuľky alebo objavovaním.

Pri objavovaní sa zvolí náhodná akcia pre daný stav. Týmto spôsobom dokáže agent objaviť nové stratégie ktoré nie su zarátané v Q tabuľke.

Pri využívaní Q tabuľky agent vyberá akciu s najvyššou Q hodnotou pre daný stav v ktorom sa nachádza.

Pomer medzi objavovaním a učením sa dá nastaviť pomocou „Epsilon Greedy“ metódy. V jej najjednoduchšej podobe môžme pomer medzi objavovaním a využívaním určiť fixne: napríklad 1 z 10 akcii bude náhodná.

Po vykonaní akcie sa aktualizujú hodnoty Q tabuľky. Na tento účel slúži vzorec pre počítanie Q hodnoty:

Rovnica 1.1: Vzorec pre počítanie Q hodnoty

Rovnicu algoritmu Q učenia môžme rozdeliť na niekoľko častí:

Rovnica 1.2: Dočastný rozdiel (TD)

Najvyššia Q hodnota ktorú je možné získať v nasledujúcom stave (Vybraním najhodnotnejšej akcie tohto stavu) znížená o  Q hodnotu aktuálneho stavu a akcie.

(factor zrážania, gamma) používame na zníženie dôležitosti budúcich Q hodnôt. Pokiaľ je gamma bližšie k 0 tak rozdiel medzi budúcou a aktuálnou Q hodnotou zaváži menej. Pri gamma blížiacej sa k 1 bude rozdiel budúcej vs okamžitej hodnoty viac ovplyvňovať aktuálnu odmenu (zvýši ju) a tým pádom bude agent preferovať budúce odmeny.

Rovnica 1.3: Ovládanie rýchlosti učenia

(Rýchlosť učenia, learning rate, lr, alpha) predstavuje ako veľmi akceptujeme novú hodnotu (najlepšiu z následujúceho stavu) oproti aktuálnej. Vo vzorci vyššie môžme vidieť: Rýchlosťou učenia vynásobime dočasný rozdiel (TD) a aktuálnu Q hodnotu vynásobíme pomocou 1 – rýchlosť učenia. Čím vyššia je naša rýchlosť učenia, tým vyššiu váhu budú mať budúce hodnoty pri aktualizácií Q tabuľky.

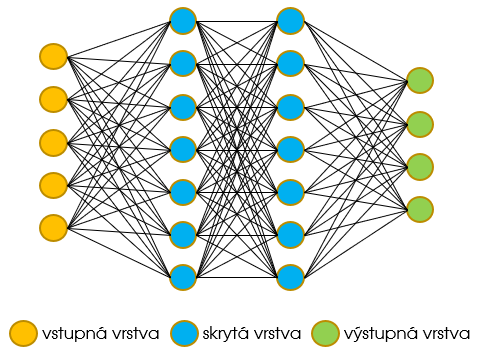
**Stavový priestor**

Kvôli potrebe zostrojenia tabuľky sú tieto metódy efektívne v prostrediach s menším počtom stavov a akcií. Pre zložitejšie aplikácie kde zostrojenie tabuľky neprichádza do úvahy sa používajú metódy využívajúce hlboké neurónové siete.

### Hlboké neurónové siete

Hlboké neurónové siete sú počítačový systém určený na predikciu výstupných hodnôt z poskytnutých vstupných hodnôt. Boli vyvynuté po vzore neurónových sieti v ľudskom mozgu a do istej miery kopírujú ich funkcionalitu.

Umelé neurónové siete sa skladajú z uzlov – umelých neurónov ktoré sú usporiadané po vrstvách. Každá sieť obsahuje vstupnú a výstupnú vrstvu a každý umelý neurón vo vstupnej vrstve je prepojený s každým umelým neurónom vo výstupnej vrstve. Pridaním ďalších vrstiev medzi vstupnú a výstupnú vrstvu – nazývaných skryté vrstvy a ich následným prepojením získame neurónovú sieť ktorú nazývame hlboká neurónová sieť.



Obrázok 1.3: Hlboká neurónová sieť

Vstupné dáta prechádzajú touto sieťou vrstvu po vrstve od vstupnej po výstupnú. Toto sa nazýva dopredná propagácia. Každý umelý neurón dostane pole vstupov z predchádzajúcej vrstvy ktoré vynásobi svojími váhami (). Neurón obsahuje jednu váhu pre každy vstup . Po vynásobení sa pripočíta bias (***b****)* a spustí sa aktivačná funkcia . Výstup (***y***) po aplikovaní aktivačnej funkcie pokračuje do neurónov v ďalšej vrstve až kým sa nedostaneme na konečný výstup z neurónovej siete.

Váhy a bias sú naučitelné parametre ktoré sa nastavujú v procese ktorý nazývame trénovanie neuróvej siete.

Aktivačná funkcia je nelineárna funkcia vďaka ktorej vie neurónová sieť aproximovať komplexné funkcie.

Rovnica 1.4: Vzorec pre počítanie výstupu neurónov

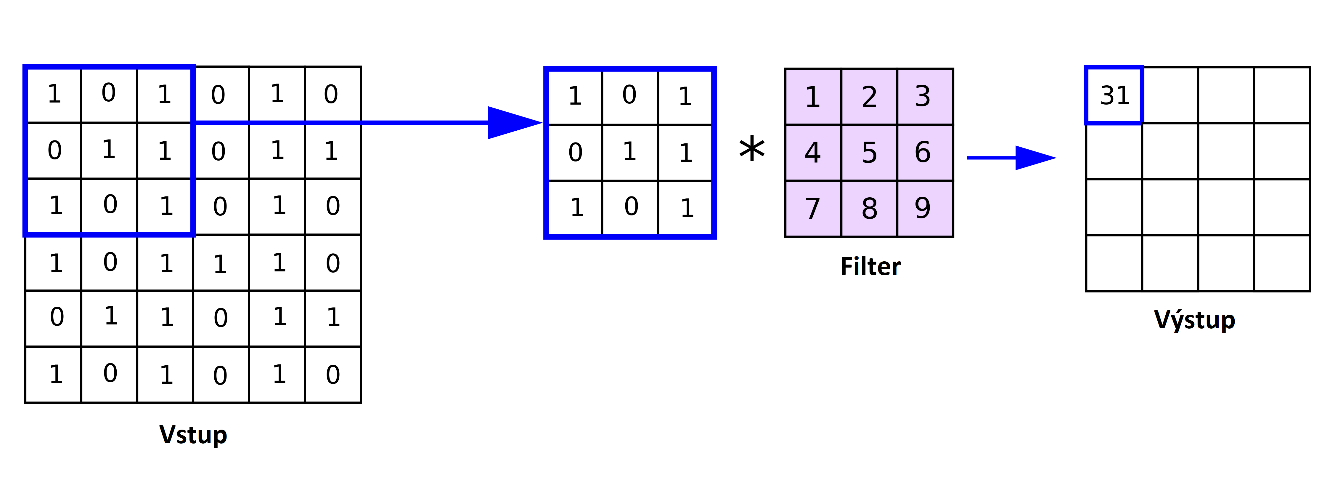
Pri trénovaní neurónovej siete sa určí funkcia ktorá definuje cieľ ktorý chceme dosiahnúť. Môže byť ohodnocovacia (chceme čo najvyššie ohodnotenie - maximalizujeme) alebo stratová (chceme znížť stratu - minimalizujeme) . Po dosadení príslušných hodnôt do funkcie a vypočítaní straty sa táto hodnota spätne propaguje do neurónovej siete a nastavujú sa podľa nej váhy a bias. Princíp propagovania straty je bližšie vysvetlený v 1.2.7 Metódy predikujúce politku – časť Gradient.

### Konvolučné neurónové siete

Konvolučné neurónové siete sú typom hlbokých neurónových sieti ktoré sa používajú pre rozoznávanie a klasifikáciu obrazu a detekciu objektov.

Základ konvolučných neurónových sietí tvoria vrstvy:

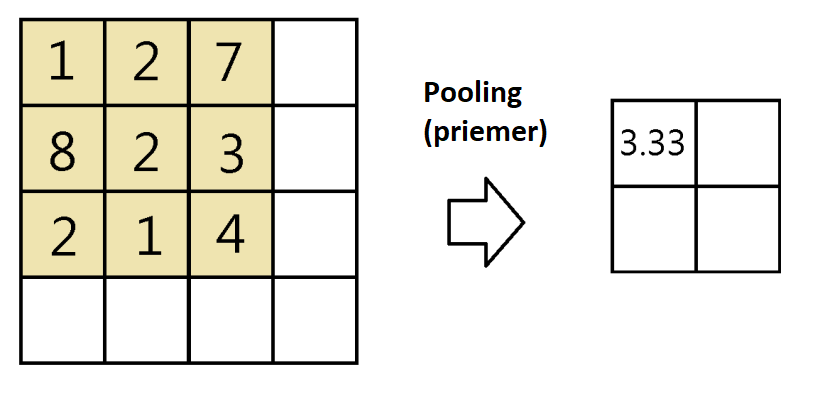
Konvolučná vrstva – používa sa extrahovanie vlastností z obrazu. Pri konvolučnej vrstve sa zadefinuje filter s naučiteľnými parametrami ktorý sa postupne aplikuje na vstupný obraz. Výstupom je nový obraz so zvýraznenými vlastnosťami podľa daného filtra.



Obrázok 1.4: Aplikácia filtra

Pri konvolučnej vrsve sa okrem veľkosti filtra definuje ešte výplň a dĺžka kroku pomocou ktorých sa dá zmenšiť alebo zväčšiť výstupný obraz.

Pooling vrstva – predstavuje ďalší spôsob ako zmenšiť obraz. Oproti konvolučnej vrstve sa pri poolingu neaplikuje filter, ale vybraná oblasť sa zmenší podľa vybraného pravidla (priemer, max, min).



Obrázok 1.5: Aplikácia poolingu

### Algoritmy využívajúce neurónové siete

Pri použití neurónových sieti pre učenie posilňovaním rozlišujeme dva základné prístupy:

**Metódy predikujúce hodnoty -** vychádzajú z algoritmu Q učenia. Pri tomto prístupe vymeníme Q tabuľku za Q funkciu. Neurónová sieť potom slúži ako aproximátor Q funkcie. Základný algoritmus založený na tomto princípe sa nazýva DQN.

**Metódy predikujúce politiku** - pri tomto prístupe neurónová sieť priamo aproximuje politku (funkciu) pre vyberanie akcií.

Zkombinovaním týchto prístupov dostávame Aktor-kritik metódy ktoré využívajú oba prístupy.

### DQN

Vymenením Q tabuľky za hlbokú neurónovú sieť sa mení spôsob aktualizácie Q hodnôt. Teraz namiesto priamého počítania novej Q hodnoty a vkladania ju do tabuľky počítame stratu () ktorá vznikla pri predikcii Q hodnoty a pomocou tejto straty aktualizujeme našu neurónovú sieť.

Rovnica 1.5: Výpočet straty DQN

Podobne ako pri algoritme Q učenia - strata sa počíta ako rozdiel najväčšej možnej Q hodnoty v budúcom stave a Q hodnoty predikovanej pre aktuálny stav.

Pre stabilizovanie učenia sa pri DQN používajú dve rovnaké neurónové siete - cieľová sieť a prediktívna sieť. Cieľova sieť je zmrazená na N krokov a strata je počas týchto krokov spätne propagovaná len cez prediktívnu sieť. Inými slovami: Aktualizujú sa len parametre patriace prediktívnej sieti Po N krokoch sa vypočíta sa nová strata a aktualizuje sa cieľová sieť podľa váh prediktívnej siete. Následne prediktívna sieť pokračuje v učení podľa novej vypočítanej chyby.

Rovnica 1.6: Výpočet straty DQN pri dvoch sietiach

Predstavuje parametre siete ktoré sa pri trénovaní nastavujú. Vľavo sú parametre cieľovej siete a vpravo parametre prediktívnej siete

Ďalším zlepšením učenia je použitie bufferu záznamov. Po sebe nasledujúce stavy sú korelované. Preto vytvoríme buffer do ktorého budeme ukladať naše stavy a pri trénovaní budeme mixovať stavy z prostredia so stavmi ktoré máme uložené.

**Ďalšie algoritmy odvodené z DQN**

**Double DQN –** DQN algoritmus používa tú istú neurónovú sieť pre ohodnotenie akcie Q hodnotou a jej vybratie. Dôsledok toho je občasné „preháňanie“ pri výbere akcii [2]. Double DQN rieši tento problém použitím dvoch sieti kde: Jedna je použitá na vyberanie akcií a druhá na generovanie Q hodnoty.

Rovnica 1.6: Výpočet straty Double DQN

**Duelling DQN** – Duelling DQN počíta Q hodnotu pomocou dvoch konceptov:

Rovnica 1.7: Výpočet straty Duelling DQN

(výhoda) nám vraví o koľko je vyberaná akcia lepšia ako ostatné akcie.

(V hodnota) predstavuje hodnotu stavu v ktorom sa nachádzame.

Pri Duelling DQN prístupe využívame dve siete. Jednu pre aproximáciu a druhú pre aproximáciu kde na konci ich hodnoty spojíme dokopy. Kvôli povahe implementácie ale spájanie nemôže byť obyčajný súčet je nutné využiť jeden z dvoch trikov popísaných v originálnej publikácií [3], kde je popísaná podrobná špecifikácia algoritmu.

### Metódy predikujúce politiku

Metódy predikujúce politiku majú lepšiu konvergenciu (nájdenie optimálneho stavu). Metódy vyberajú priamo akciu a preto pri menšej zmene politky nastane menšia zmena pri vyberaní akcií. Pri metódach predikujúcich hodnotu môže mať malá zmena Q hodnoty drastický efekt na vybranú akciu (napríklad keď majú dve veľmi rozdielne akcie podobné Q hodnoty).

Politka môže byť deterministická alebo stochastická (metódy predikujúce hodnotu sú len deterministické). To znamená že naša akcia môže byť konkrétna hodnota, alebo rozdelenie pravdepodobností pre dané akcie. Stochastická politika je výhodná v prostrediach ktoré obsahujú elementy neistoty, kde je viac ako jedna správna akcia ktorú môžeme vykonať.

Stochastickým politikám sa darí lepšie v priestoroch s veľkým počtom akcií (metódy predikujúce hodnotu musia ohodnotiť každú akciu pre daný stav) alebo v prípadoch kde je nutné použiť spojité akcie (Napríklad regulovať plynový pedál). Tak isto stochastická politka odstraňuje problém objavovania vs využívania (funkcie). V prípadoch keď majú dve akciu rovnakú pravdepodobnosť môžeme náhodne vybrať jednu z nich, čiže objavovanie je zabudované v princípe tejto metódy a nemusí sa pridávať externe. Avšak kvôli mechanizmu, ktorý používajú pre optimalizáciu sú tieto metódy vzorkovo menej efektívne ako metódy, ktoré predikujú hodnoty.

**Politka**

Politiku označujeme gréckym písmenom s parametrami . Vstupom do politky je akčný priestor a stav prostedia. Výstupom politky je rozdelenie pravdepodobnosti vykonania akcií ***a*** v stave ***s***v prípade stochastickej politky*.*

Rovnica 1.8: Vzorec pre politiku

Naším cieľom je optimalizovať parametre . Na to sa používa ohodnocovacia funkcia politiky **. T**áto funkcia hodnotí očakávanú odmenu ktorú môžeme dosiahnuť podľa danej politky .

V epizodických prostrediach kde je definovaný začiatok a koniec epizódy môžeme brať epizódu ako celok a bude predstavovať kumulovanú zrazenú odmenu za každý krok v epizóde (napríklad najvyššie dosiahnuté skóre na konci hry).

Rovnica 1.9: Vzorec pre ohodnocovaciu funkciu politky 1

V spojitých prostrediach ktoré nemajú začiatok ani koniec sa nemôžeme spoliehať na konkrétne stavy. Preto sa stavy spriemerovávajú a hodnotia sa podľa ich výskytu. Tento spôsob sa ale v praxi nepoužíva. Narážame tu na rovnaký problém ako pri Q tabuľke a to je príliš veľké množstvo stavov ktoré treba preskúmať.

**Kde**

Rovnica 1.10: Vzorec pre ohodnocovaciu funkciu politky 2

je počet výskytov aktuálneho stavu aje počet výskytov všetkých stavov.

Pokiaľ chceme dosiahnúť najvyššiu odmenu za určitý časový úsek tak použijeme tretí spôsob. Znova narážame na problém s počtom stavov, ako pri predchádzajúcom vzorci.

Rovnica 1.11: Vzorec pre ohodnocovaciu funkciu politky 3

pravdepodobnosť že som v stave ***s.***

pravdepodobnosť vybratia akcie ***a***  v stave ***s***  pri nasledovaní politky **.**

Okamžitá odmena.

**Gradient**

Samotná optimalizácia sa vykonáva pomocou gradientov. Gradienty sa počítajú z hodnotiacej alebo stratovej funkcie a určuje zmenu pre naše parametre . V tomto prípade máme hodnotiacu funkciu ktorú chceme maximalizovať, na to sa používa metóda stúpajúceho gradientu (Gradient ascent). V prípade že máme stratu ktorú chceme minimalizovať sa používa metóda klesajúceho gradientu (Gradient descent). Gradient označujeme ako .

V prípade že používame stochastickú politku je naším výstupom rozdelenie pravdepodobnosti pre akcie. Pre výpočet gradientu je ale nutné použiť derivácie nad týmto rozdelením, na čo sa používajú logaritmy [4].

Finálna podoba gradientu ohodnoteného pomocou :

Rovnica 1.12: Vzorec pre gradient

Zmenu parametrov môžeme zapísať ako teda:

Rovnica 1.13: Vzorec pre zmenu parametrov

Konečným cieľom je nájsť nájsť optimálne parametre .

### A2C

A2C patrí do rodiny Aktor-Kritik algoritmov ktoré spájajú výhody metód založených na politike s výhodami metód založených na hodnotách. Využíva dve neurónové siete:

Aktor – vyberá akcie podľa svojej politiky.

Kritik – ohodnocuje akcie vybrané Actorom pomocou tzv. Výhody (A), resp. o koľko je vybraná akcia lepšia ako ostatné akcie. Výhoda sa využíva pri učení politky pre Actora.

Algoritmy založené len na politke ako napríklad posilňujúci algoritmus vykonávajú aktualizácie až na konci epizódy. To znamená že sa učí zo všetkých krokov vykonaných počas epizódy, aj tých menej vhodných, alebo vyslovene zlých. Kvôli tomu potrebujeme pri týchto algoritmoch viac vzoriek než pri algoritmoch predikujúcich hodnotu. Pridaním výhody dokážeme aktualizovať parametre siete kazdých ***t*** krokov algoritmu. Pre A hodnoty potrebujeme ďalšiu neurónovú sieť s vlastnou V funkciou (aproximuje hodnotu stavu) a vlastnými parametrami ***v***. V prostrediach so spracovaním obrazu pomocou konvolučných vrstiev sa dá použíť „dvojhlavá“ architektúra kde konvolučné vrstvy sú zdieľané a z nich vychádzajú dve oddelené plne prepojené vrstvy. Jedna pre Aktora a druhá pre Kritika.

Aktor – politika pre vyberanie akcií:

Kritik – politika pre odhad V hodnoty:

**potom:**

Rovnica 1.14: Vzorec pre výhodu

Čo nám dá výsledný vzorec pre aktualizáciu parametrov:

Rovnica 1.15: Vzorec pre gradient A2C

A2C neukladá záznamy do bufferu ako napríklad DQN, preto sa pre odstránenie korelácií používa paralelné vykonávanie akcií, kde sa neurónová siet pridelí skupine pracovníkov kde každý dostane svoje vlastné prostredie v ktorom každý vykonáva kroky nezávisle na ostatných pracovníkoch. Každých ***t*** krokov sa vykonané akcie, stavy a odmeny spriemerujú a následne sa vykoná aktualizácia a každý pracovník dostane aktualizované parametre.

### A3C

A3C je predchodcom A2C. Pri tejto architektúre používajú pracovníci kópiu globálnej neurónovej siete pomocou ktorej robia kroky vo svojích prostrediach. Keď pracovník vykoná určitý počet krokov, nezávisle na ostatných pracovníkoch tak aktualizuje globálnu sieť pomocou svojích parametrov a skopíruje aktuálny stav aby získal parametre ktoré aktualizovali ostatní pracovníci. Vďaka asynchrónnym aktualizáciam môžu jednotliví pracovníci pracovať vo vlastných vláknach a tým pádom byť efektívnejší čo sa využívania CPU týka. Avšak asynchrónnosť neposkytuje žiadne extra výhody, práve naopak. Podľa OpenAI Baselines synchrónna implementácia (A2C) a využitie GPU vykazuje lepšie výsledky [5].

### PPO

PPO (Proximal Policy Optimization) bol vytvorený v roku 2017 tímom spoločnosti Open AI a v čase písania tejto práce je považovaný za jeden z najlepšich (state of the art) algoritmov pre učenie posilňovaním [6]. Existujú aj efektívnejšie algoritmy ako napríklad ACER, ktorý je ale podstatne zložitejší na implementáciu.

PPO patrí do rodiny Aktor-Kritik algoritmov. Pri A2C algoritme musíme dbať na správne nastavenie parametrov. Pri nevhodnom nastavení môžeme po čase pozorovať stratu schopnosti učiť sa (príklad tohto javu sa nachádza aj v implementačnej časti tejto práce). Tento jav je spôsobený príliž veľkými zmenami pri optimalizácií parametrov. PPO rieši tento problém – vylepšuje stabilitu učenia a znižuje závislosť od nastavených parametrov. Na dosiahnutie tohto cieľa sa počíta rozdiel medzi pravdepodobnosťami vykonania akcie pod novou a starou politikou.

Rovnica 1.16: Vzorec pre gradient PPO

Potom:

Akcia je preferovanejšia pod novou politikou,

Akcia je preferovanejšia pod starou politikou

Vďaka vieme povedať aká veľká zmena v politike nastane. Ďalši krok je ohraničenie veľkosti tejto zmeny. Podľa originálnej publikácie by mala byť táto zmena obmädzená v rozmädzí od 0,8 po 1,2 [7]. Na obmädzenie môžeme použiť jeden z dvoch spôsobov popísaných v publikácií. TRPO alebo tzv. „pristrihnutie“.

Pristrihnutie (clip):

Rovnica 1.17: Rozšírený vzorec pre gradient PPO

Takto dostaneme dva pomery pravdepodobnosti – nepristrihnutý a pristrihnutý pomer v rozsahu , kdeje parameter ktorý predstavuje rozsah pristrihnutia (podľa publikácie ). Následne vezmeme minimum – čiže konzervatívnejšiu hodnotu pre zmenu politiky.

## Knižnice pre neurónové siete

V tejto časti predstavíme dve knižnice pre efektívne písanie algoritmov strojového učenia pomocou neurónových sieti - TensorFlow a Pytorch.

### TensorFlow

[8] [9] Vyvynutý Google Brain tímom a napísaný v programovacích jazykoch: Python, C++ a CUDA. TensorFlow beží na všetkých populárnych operačných systémoch (Linux, Windows, MacOS, iOS, Android). Obsahuje podporu pre využitie CPU, GPU alebo TPU.

* Funkcie poskytované TensorFlow knižnicou:
* Data Flow Graphs – v TensorFlow sú výpočty popísané grafmi prúdenia dát. Uzly v grafe reprezentujú matematické operácie a hrany predstavujú viacrozmerné datasety (tensory). Tento prístup je výhodny v oblasti neurónových sietí kde sa dáta často menia a vykonáva sa nad nimi množstvo operácií. Grafy samotné sa dajú vizualizovať pomocou funkcie nazývanej TensorBoard pre ešte lepšie pochopenie toku dát a vykonaných operacií.
* Knižnica funkcií – TensorFlow obsahuje širokú škálu už implementovaných matematických funkcií pripravených na použitie.
* Podpora pre programovacie jazyky – Python (so stabilnými aktualizáciami), C++, Javascript, Java, Go, Haskell, C#, R, Julia, Scala, Ruby, MATLAB (vo vývoji, nestabilné aktualizácie).

### Pytorch

[8] [10] Považuje sa za kompetitora TensorFlowu a zdieľa s ním množstvo výhod:

* Podpora pre všetky populárne operačné systémy,
* Programovanie v Pythone s dostupným API pre C++
* Silná podpora pre GPU výpočty.
* Synergia s populárnymi Python knižnicami ako Numpy, SciPy, Pandas ...

Ďalšie vlastnosti knižnice:

* TorchScript - metóda pre jednoduché písanie machine learning modelov,
* TorchServe – umožnuje jednoduché nasadzovanie modelov do produkčných prostredí,
* Paralelné výpočty,
* Podpora ONNX štandardu,
* Podpora pre cloud,

V našej práci sme si vybrali knižnicu PyTorch, pretože nám prišla vhodnejšia pre potreby práce z hľadiska implementácie algoritmu.

# algoritmus a2c

V tejto časti si popíšeme našu implementáciu A2C algoritmu. Celá implementácia je realizovaná v jazyku Python s pomocou knižnice PyTorch.

## Agent

Agent obsahuje všetky parametre algoritmu a našu neurónovú sieť. Pri vytvorení sa pokúsi nájsť grafickú kartu. Pokiaľ ju nájde tak výpočty v priebehu učenia sa presunú na ňu. Pokiaľ grafickú kartu nenájde tak výpočty budú pokračovat na CPU. Agent obsahuje 3 metódy - save/load\_model pre ukladanie a načítavanie modelov počas trénovania a learn metódu ktorá je zodpovedná za učenie a ktorú si do hĺbky popíšeme.

### Učenie

Učenie je implementované v metóde learn. Vstupné parametre pre túto metódu sú:

* Workers – Pole pracovníkov
* Iterations – Počet iterácií učenia sa - príkazy od začiatku až po krok optimalizátora a zapísanie štatistík.
* Steps – Počet krokov v prostredí. Tieto kroky sa všetky spracujú vrámcí jednej iterácie.
* Write – Parameter ktorý zapína ukladanie modelu a zbieranie dát.
* Start\_episode – Agenta je možné trénovať na viac krát. V prípade že zbierame dáta chceme vedieť nastaviť štartovaciu epizódu, aby sme vedeli nadviazať na predošle zozbierané dáta.
* Lr\_change\_interval – Počet epizód po ktorých zmeníme našu mieru učenia (lr).

Na začiatku trénovania sa zresetujú všetky prostredia aby sme získali vstupné obzervácie. Potom sa začína prvá iterácia. Na začiatku iterácie sa vytvoria premenné pre zbieranie potrebných údajov vrámci iterácie. Tieto premenné sú tensory tvaru [steps, workers, 1] a naplnia sa nulami. Po vytvorení premenných následujú kroky v prostredí. Vykonanie kroku pozostáva zo zavolania forward metódy našej neurónovej siete nad obzerváciami uloženými v našej premennej. pre vygenerovanie aktor a kritik hodnôt. Aktor

hodnoty následne premeníme na pravdepodobnosti pre akcie v rozmedzí od 0 do 1 ,logaritmy pre neskorší výpočet straty a entropiu ktorá nám pomôže s objavovaním. Ďalej aktualizujeme naše premenné s údajmi za daný krok. V závere kroku vykonáme krok v prostredí pracovníkov. Obzervácie ktoré získame použijeme v ďalšom kroku. Po vykonaní všetkých krokov vrámci iterácie zavoláme forward nad poslednou sadou obzervácií aby sme získali kritik hodnoty.

So všetkými dátami pokope pristúpime k počítaniu výhody:

výhoda = [počet\_krokov, počet\_pracovníkov, 1]

for krok in reversed(počet\_krokov):

kritik[krok + 1] = odmeny[krok] + (gamma \* kritik[stav + 1] \*

neprehrané\_hry[krok])

výhoda[krok] = kritik[krok + 1] - kritik[krok]

výhoda = (výhoda – priemer(výhoda)) / štandardná\_odchylka(výhoda)

Výhoda sa počíta od konca. Najprv vypočítame hodnotu kritika pre následujúci stav. V prípade že sme v danom kroku prehrali hru tak odmeny[krok] bude rovná 0 tak isto ako neprehrané\_hry[krok] a tým pádom aj kritikove ohodnotenie bude 0. Následne vypočítame výhodu ako rozdiel ohodnotenia následujúceho stavu oproti aktuálnemu stavu.

V prípade že v žiadnom kroku neprehráme tak hodnota našeho kritika a tým pádom aj výhoda sa postupne zmenšuje vďaka zrážaciemu faktoru. To znamená že agent získa lepšie ohodnotenie v neskorších krokoch a tým pádom je odmenený za „rozmýšľanie“ nad budúcimi krokmi.

V závere počítania výhody použijeme vzorec pre normalizáciu aby sme znížili hodnotu výhody pre menšie straty a tým pádom menšie zmeny parametrov pri aktualizácií.

V ďalšom kroku vypočítame straty:

strata\_kritik = priemer(výhoda2) \* koeficient\_zrážania\_kritik

strata\_aktor = - priemer(aktor\_log\_pravdepodobnosti \* výhoda)

strata\_entropia = priemer(entropie) \* beta

strata = strata\_kritik + strata\_aktor + strata\_entropia

Straty je možné počítať oddelene a potom ich spojiť ako v príklade vyššie. Tento prístup je výhodný pokiaľ chceme zbierať štatistiky pre jednotlivé straty. V príklade vyššie používame entropiu ktorá pomáha pri objavovaní. Čím istejší si je aktor nejakou akciou, tým väčšia bude entropia a tým väčšia bude naša strata.

optimizátor = vynuluj\_gradienty()

strata.spatna\_propagacia()

optimizátor.krok()

Po vypočítani straty vynulujeme predchádzajúce gradienty, zpropagujeme nové gradienty a spravíme krok optimalizátora ktorý aktualizuje naše parametre. V závere skontrolujeme aktuálnu iterácie a vypíšeme priebeh trénovania, skontrolujeme rýchlosť učenia a uložíme model a štatistiky.

Metódu learn môžeme popísať pseudokódom:

Inicializuj začiatočné premenné

Vygeneruj začiatočné obzervácie *szačiatočné*

For iteracia in iteracie:

Inicializuj premenné pre iteráciu

For step in steps:

Vygeneruj Aktor, Kritik hodnoty z obzervácií

Vypočítaj logaritmy a zapíš hodnoty

For worker in workers:

Krok v prostredí

Zapíš nové obzervácie *st*

Vygeneruj Kritik hodnoty nad novými obzerváciami

Vypočítaj výhodu *At*

Vypočítaj stratu *J(θ)*

Optimalizuj pomocou straty

If iteracia % 100:

Vypíš aktuálny stav do konzoly

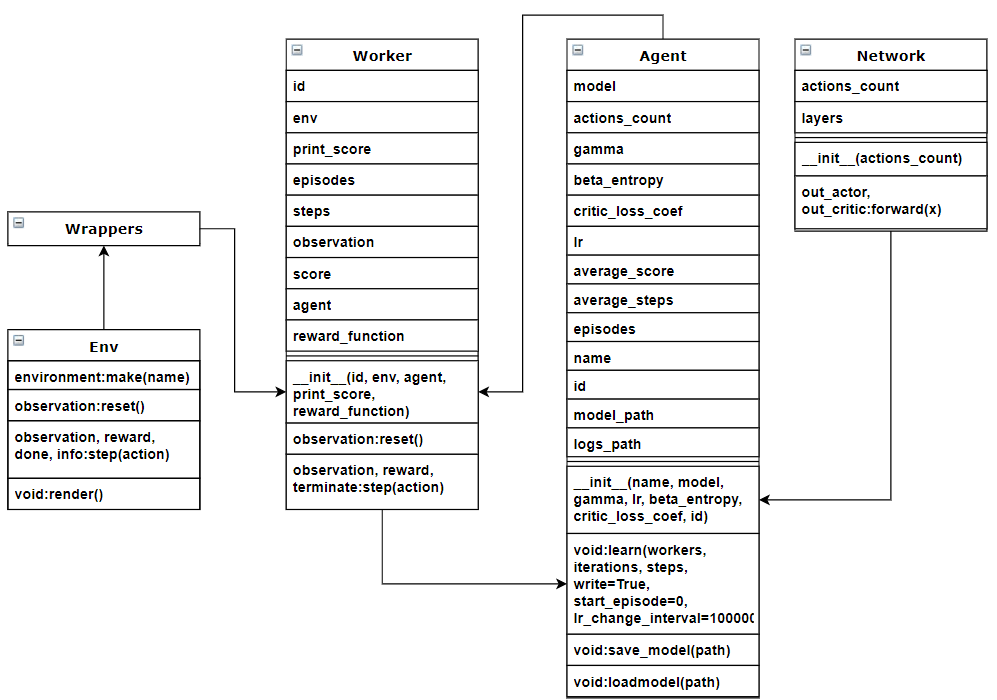
If iteracia % 1000:

Ulož model

If epizoda % Lr\_change\_interval:

Zmeň rýchlosť učenia

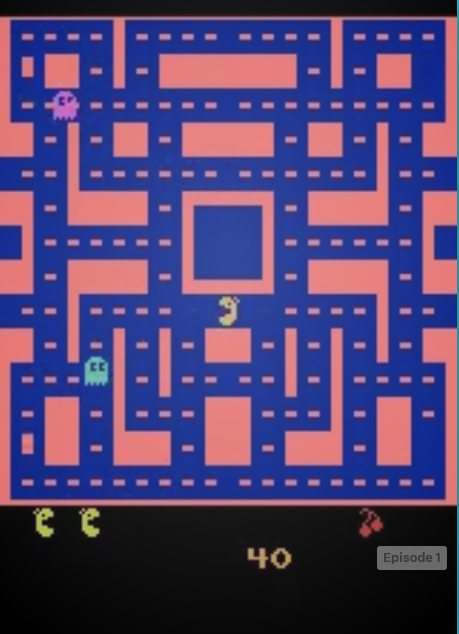
# Trénovanie algoritmu



Obrázok 3.1: UML systému pre trénovanie agenta

## Prostredie

Pre vytvorenie našeho prostredia používame knižnicu „Gym“ od OpenAI. Táto knižnica ponúka širokú škálu prostredí založených na Atari hrách ktoré sú určené špecificky pre trénovanie neurónových sietí. V tejto práci budeme používať prostredie Pacman (Ms-Pacman-v0).



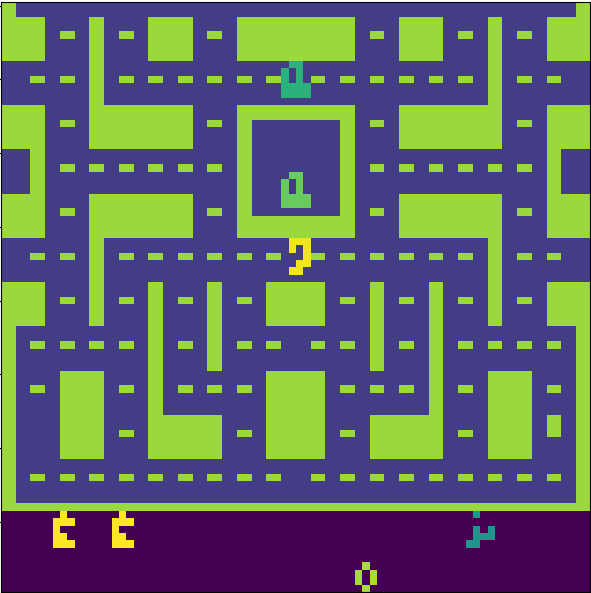
Obrázok 3.2: Stav prostredia

Prostredie sa vytvára pomocou metódy **gym.make(‘Ms-Pacman-v0’)** a inicializuje pomocou **gym.reset()** čo nám zresetuje prostredie do východiskového stavu a vráti prvú obzerváciu (stav prostredia). Následne sa v ňom môžeme vykonávať kroky pomocou metódy **gym.step(akcia)**. Návratová hodnota tejto metódy je obzervácia po vykonaní akcie, odmena za vykonanú akciu, boolean ktorý indikuje či nastal koniec hry a diagnostické informácie pre potreby debbugingu. Vizualizácia prostredia je možná pomocou metódy **gym.render().**

Pre efektívnejšie učenie sme použili obaľovače (Wrapper) pomocou ktorých sme upravili naše prostredie:

* SkipEnv(4) – Preskočí 3 obzervácie, čiže dostaneme každý 4. stav a sumu odmien za dané stavy. Pacman používa malé kroky, preto je výhodné použiť tento wrapper.
* PreProccessFrame – Zmenší našu obzerváciu z veľkosti 210x160x3 na veľkosť 80x80 a odstraní RGB. Výsledná obzervácia je 80x80x1
* MoveImgChannel – Zmení poradie prvkov v tensori. Originálna obzervácia je 80x80x1, pytorch ale potrebuje obzerváciu v tvare 1x80x80
* ScaleFrame – Zmení škálovanie obzervácie.
* BufferWrapper(4) – Vytvorí skupinu 4 po sebe následujúcich obzervácií.

Po spracovaní má naša obzervácia tvar (4,80,80), kde 4 je počet po sebe nasledujúcich snímok pre znázornenie pohybu a 80,80 predstavujú výšku a šírku našej obzervácie.



Obrázok 3.3: Transformovaná obzervácia

Kvôli použitiu nesprávneho obaľovača pre zmenšenie sme odrezali vrchnú časť našeho hracieho prostredia. Napriek tomu sa A2C algoritmus naučil pohybovať v tejto časti.

## Pracovník

Pracovníci majú každý svoje id pre ich lepšie rozpoznanie, svoje vlastné prostredie v ktorom vykonávajú kroky a kópiu našeho agenta, prostredníctvom ktorého vykonávajú dané kroky. Ďalej obsahujú parametre na zbieranie údajov (vykonané epizódy, dosiahnuté skóre ...) a možnosť zadefinovať vlastnú odmenovú funkciu.

V našej práci nepoužívame priamo skóre z prostredia ale spracuvávame ho pomocou osobitnej odmenovej funkcie na hodnotu 1 alebo – 1 podľa toho či je naše skóre lepšie ako to predošlé alebo horšie. Tento prístup je výhodny pretože skóre môžu byť pomerne vysoké. To znamená vysoké zmeny pri nastavovaní parametrov, čo spôsobuje

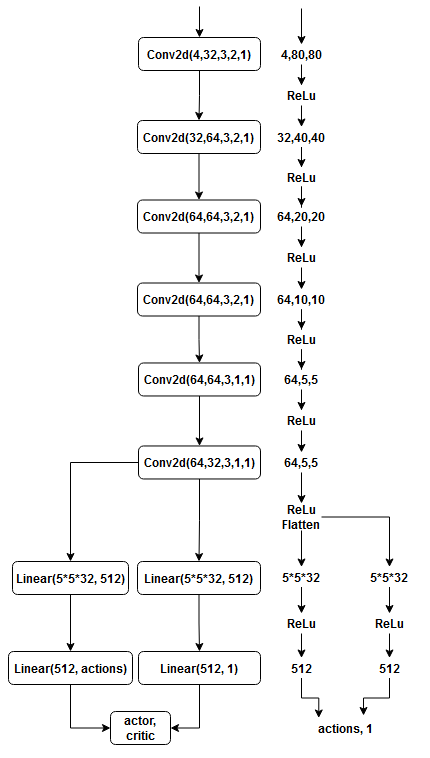
menej stabilný tréning a menej optimálne stratégie. Pri krokoch ktoré normálne negenerujú odmenu bude agent dostávať záporné odmeny. Aby nedostával záporné odmeny, musí si zvoliť cestu ktorá ho k odmene zavedie, nemôže teda robiť bezcieľne kroky. Príklady si ukážeme v časti trénovanie.

## Sieť

Pre vytvorenie našej neurónovej siete sme použili knižnicu Pytorch ktorá umožňuje jednoduchú tvorbu sietí pomocou modulu torch.nn. Pre spracovanie nášho prostredia používame konvolučné neurónové siete, ktoré extrahujú potrebné vlastnosti a zmenšia vstupné data. Výstup z konvolučných sieti smeruje do dvoch plne prepojených vrstiev. Prvá z nich predstavuje Aktora a druhá Kritika. Vyskúšali sme niekoľko dizajnov neurónových sieti ktoré budu bližšie popísane v ďalšej časti. Na obrázku nižšie môžete vidieť dizajn natrénovanej siete ktorú sme použili pre náš program.

Prvých 6 vrstiev sú konvolučné pre 2D obraz zapísané ako **Conv2d(vstupné kanály, výstupné kanály, filter, krok, výplň)**. Prvé 4 vrstvy zmenšia nášu vstupnú obzerváciu o polovicu (krok = 2) a vykonajú aktivačnú funkciu ReLu. Ďalšie 2 vrstvy už rozmery nezmenšujú. Po konvolúcií nasleduje funkcia flatten ktorá nám vytvorí jednorozmerný vstup potrebný pre následujúce lineárne vrstvy.

Lineárne vrstvy sú 4. Zapísané ako **Linear(vstupné neuróny, výstupné neuróny)** Dve pre Aktora a dve pre Kritika. Výstupom sú naše požadované hodnoty. Pri vytvorení používame funkciu ktorá inicializuje parametre tejto siete na náhodné hodnoty v rozmedzí od 0 po 1.



Obrázok 3.4: Dizajn neurónovej siete

## Trénovanie

Prvotné parametre pre trénovanie boli poskytnuté našim školiteľom ktorý má skúsenosti s učením posilňovaním a A2C algoritmom.

Počet pracovníkov: 20

Počet iterácií trénovania: 5000

Počet krokov v prostredí: 10

Počet obzervácií pre jednu inštanciu prostredia: 4

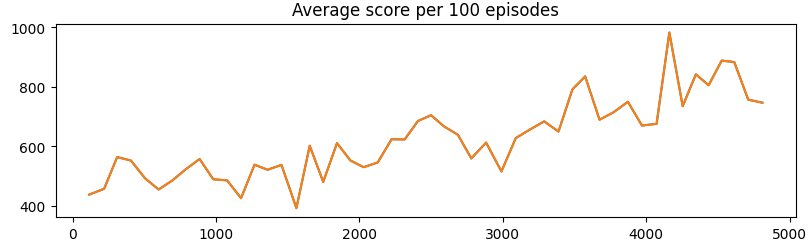
Zrážací faktor: 0,99

Rýchlosť učenia: 0,0005

Beta: 0,01

Koeficient zrážania pre Kritika: 0,5

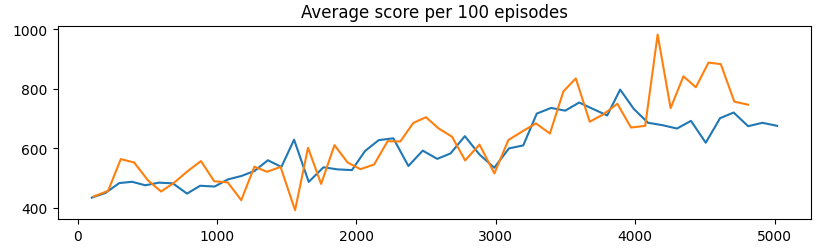
Sieť: 4 konvolúčne vrstvy o veľkosti 64, plne prepojené vrstvy o veľkosti 512 neurónov, výstupné plne prepojené vrstvy.



Obrázok 3.5: Prvotné skóre dosiahnuté prvým agentom

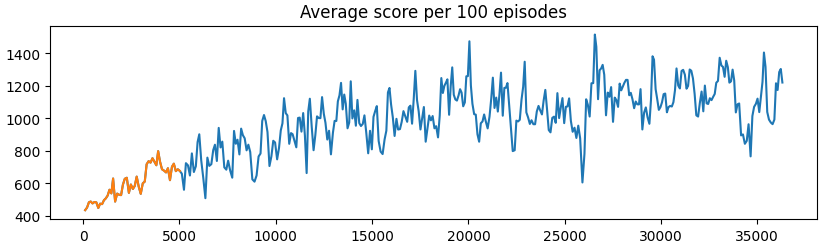
Zbierali sme priemerné skóre za posledných 100 epizód, aby bolo ľahšie rozoznávať trendy pri trénovaní. Zobrazený graf sa vzťahuje na vykonané epizódy, nie iterácie. Tento agent za 5000 iterácií vykonal 4810 epizód kde jedna epizóda predstavuje jednu hru. S aktuálnymi parametrami trvalo vykonanie 5000 iterácií približne 4 hodiny.

V ďalšom príklade sme upravili 2 parametre pre demonštráciu dôležitosti správneho nastavenia parametrov. Zvýšili sme počet krokov z 10 na 20 a znížili počet pracovníkov na 10. Čo znamená že dáta boli o kúsok viac korelované a agent sa aktualizoval po väčšom počte krokov. Tento test bol inšpirovaný článkom [11].



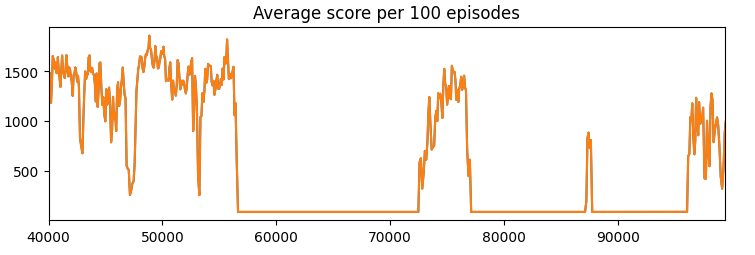
Obrázok 3.6: Priebeh tréningu s nevhodnými parametrami (modré) 1

Pri nižšom počte iterácií sa tento rozdiel prejaví ako pomalšie trénovanie, agent má ale stále potenciál pre zlepšenie.



Obrázok 3.7: Priebeh tréningu s nevhodnými parametrami 2

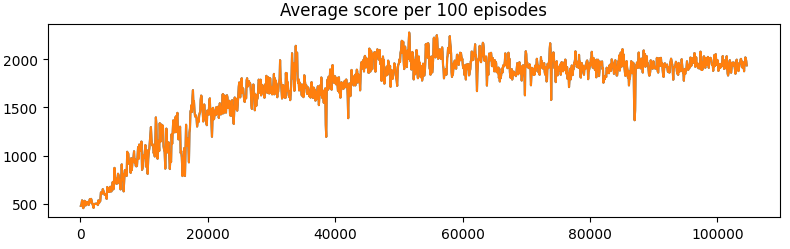
Pri dlhšom trénovaní môžeme vidieť vysoké kolísanie našeho skóre.



Obrázok 3.8: Priebeh tréningu s nevhodnými parametrami 3

Trénovanie je čím ďalej nestabilnejšie až do bodu keď sa agent odnaučí kompletne všetko. Od epizódy 57 000 vykonáva už len jeden úkon stále dookola a jeho dosiahnuté skóre je 90. Neskôr sa mu podarilo znova sa zlepšiť ale nedokázal prekonať svoje predošlé najlepšie skóre a „výpadky“ pokračovali naďalej. Ďalšie vizualizácie sa nachádzajú v prílohe B.

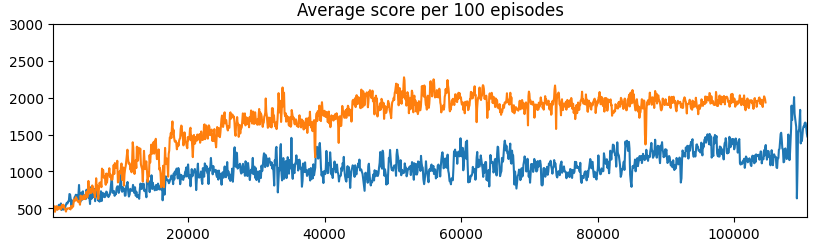
V trénovaní funknčného agenta sme pokračovali s našimi pôvodnými parametrami (20 pracovníkov a 10 krokov), ale rýchlosť učenia sme znížili 5 násobne na hodnotu 0,0001 pre lepšiu stabilitu.



Obrázok 3.9 Najlepšie skóre dosiahnuté prvým agentom

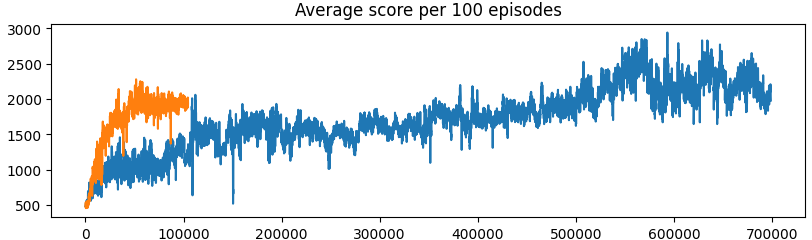
Nášeho pôvodného agenta sa nám podarilo vytrénovať na maximálne priemerné skóre okolo 2000. Pričom na strop možného skóre sme narazili okolo 50 000 epizódy čo je približne 62 000 iterácií. Od tohto bodu už agent nejavil žiadne zlepšenie. Ďalšie vizualizácie sa nachádzajú v prílohe A.

Nakoľko na vyhratie hry potrebujeme minimálne skóre 2500 (získame pozbieraním všetkých „bodíkov“ roztrúsených po mape). Po prekonzultovaní s naším školiteľom sme pridali ďalšie 2 konvolučné vrstvy o veľkosti 64 pričom ostatné parametre sme nechali rovnaké ako predtým.



Obrázok 3.10: Porovnanie agent 1 (oranžová) vs agent 2 (modrá)

Pridaním dvoch vrstiev o veľkosti 64 sme zároveň pridali veľké množstvo nových parametrov ktoré treba nastaviť, preto agent na prvý pohľad vyzerá že sa mu darí horšie.



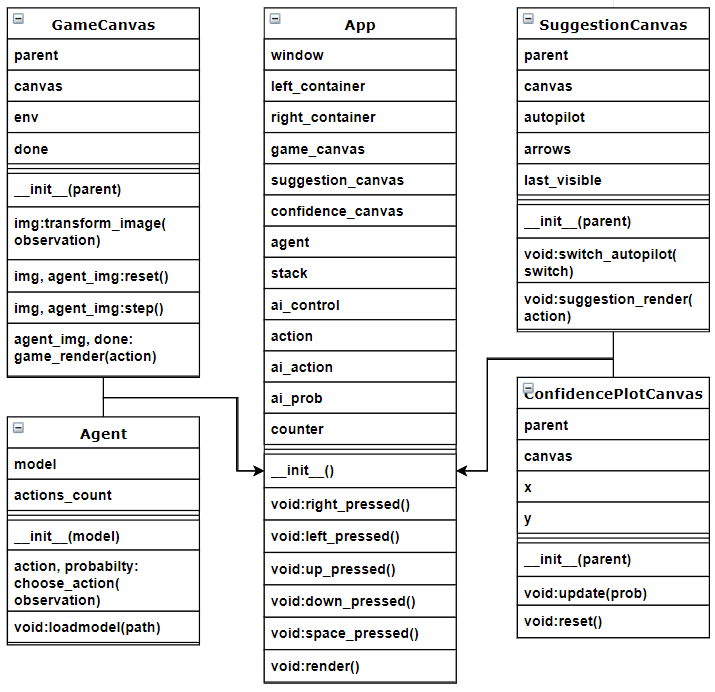
Obrázok 3.11: Kompletne natrénovaný agent 2

Len pridaním 2 konvolučných vrstiev sa počet epizód potrebných pre natrénovanie zväčšil skoro 10 násobne (uvažujeme strop prvého agenta pri 50 000 epizóde). Tento agent už dokáže vyhrať jednu hru (po výhre sa prostredie zresetuje a skóre ostáva). Ďalšie vizualizácie sa nachádzajú v prílohe C.

Natrénovanie tohto agenta (vykonaním 700 000 epizód) trvalo približne 7 celých dní na serveri s CPU Intel Pentium g3220 3GHz a ďalšie 5 celé dni na serveri s CPU Intel i7 6700 4,3GHz.

Kvôli veľkému výkonu potrebnému pre natrénovanie by ďalšie zmeny zabrali príliž veľa času, preto v našej práci použijeme tohto agenta. Podľa Open AI najvyššie dosiahnuté skóre v prostredí Gym Pacman (dosiahnúte agentom) je 5738 a bolo dosiahnuté asynchrónnou implementáciou tohto algoritmu (A3C) [12]. Je však možné dosiahnúť omnoho väčšie skóre (v rádoch 10-tisícov aj 100-tisícov) použitím iných algoritmov [13].

# Program

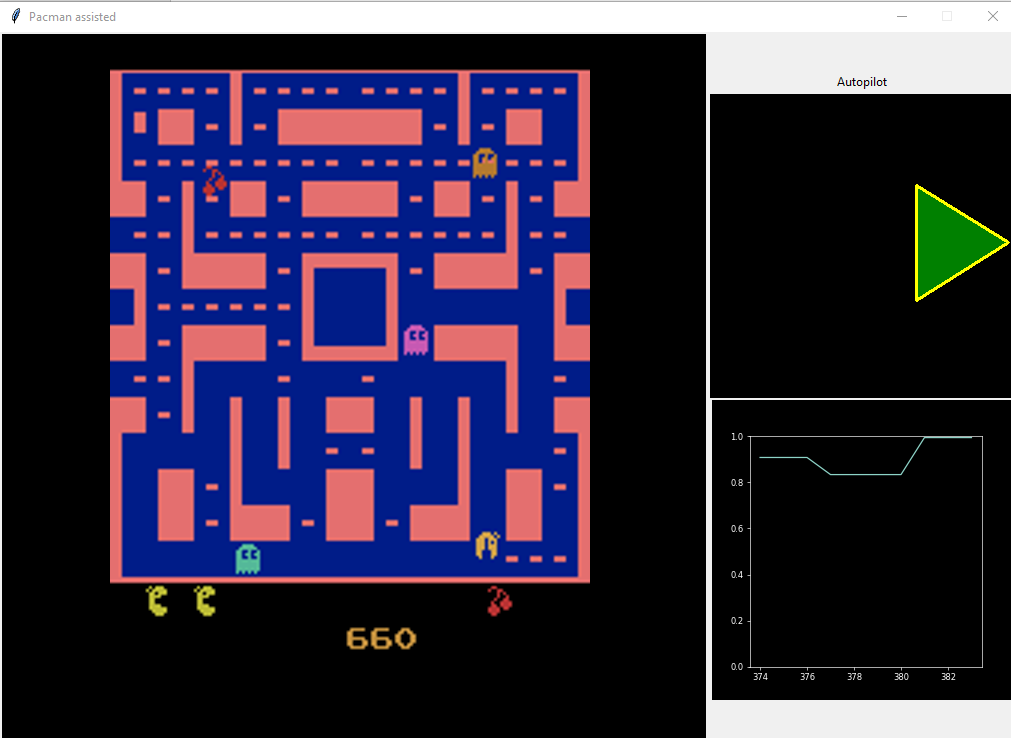


Obrázok 4.1: UML Asistovaného systému

Pre splnenie cieľa tejto práce sme vytvorili aplikáciu do ktorej sme zasadili naše prostredie pacmana a implementovali ovládací mechanizmus pre ľudského hráča, nášho A2C agenta a asistenčné mechanizmy. Aplikácia je implementovaná v jazyku Python. Na používateľské rozhranie sme použíli knižnicu Tkinkter.

Asistenčný mechanizmus je viacstupňový:

* V pravo na obrazovke máme okno v ktorom sa zobrazujú akcie ktoré bý v aktuálnom stave vykonal náš agent.
* V ľavom spodnom rohu je potom graf ktorý znázorňuje ako veľmi si je agent istý danou akciou.
* Stlačením medzerníka môžeme predať ovládanie nášmu agentovi. Po opätovnom stlačení sa ovládanie prepne naspäť na hráča.



Obrázok 4.2: GUI Asistovaného systému

## Hrací Agent

V aplikácií sme použili zjednodušenú verziu agenta ktorého sme použili pre trénovanie.

Hrací agent obsahuje len model našej neurónovej siete. Kedže je už natrénovaný tak nepotrebujeme implementovať parametre pre učenie ani premenné pre zaznamenávanie štatistík. Tak isto nepotrebujeme metódu pre učenie alebo ukladanie modelu. Namiesto metódy pre učenie sme vytvorili jednoduchú metódu pre vyberanie akcie pre danú obzerváciu ktorá nám vráti aj percentuálnu pravdepodobnosť danej akcie, alebo inými slovami ako veľmi si je náš agent istý danou akciou.

## Hra

Trieda Hra (GameCanvas) obaľuje prostredie Gym pacman. V hre je implementovány krok a reset originálneho gym prostredia. Pre zobrazenie Pacman prostredia sme implementovali vlastnú render metódu (game\_render), kde: Krok v prostredí vráti obzerváciu, čo je tensor veľkosti (3x210x160). Túto obzerváciu následne pretransformujeme na obrázok pomocou knižnice PIL a zasadíme ho do herného plátna. Po zasadení sa originálna obzervácia pretransformuje pomocou wrappera na tensor veľkosti (1x80x80) aby ho mohol spracovať náš agent.

## Konzultant

Trieda Konzultant (SuggestionCanvas) obsahuje plátno v ktorom sú zadefinované 4 šípky - jednu pre každý smer a kruh ktorý predstavuje nič nerobenie. Podľa akcie agenta sa na plátne vždy zobrazí jedna zo šípok, alebo kruh. Ďalej nás konzultant informuje o tom či hru práve ovláda hráč alebo agent pomocou nápisu „Autopilot“.

## Graf istoty

Graf istoty (ConfidencePlotCanvas) implementuje graf pomocou knižnice matplotlib v ktorom je zobrazená istota našeho agenta za posledných 10 akcií.

## Aplikácia

Aplikácia je trieda ktorá spája dokopy všetky predchádzajúce triedy. Aplikácia obsahuje hlavné okno našeho GUI do ktorého sú zasadené ostatné moduly (hra, konzultant, graf istoty) a definuje ich pozície.

V aplikácií sú zadefinované metódy pre ovládanie hráča pomocou šípiek na klávesnici a medzerník na prepínanie medzi ovládaním hráčom a ovládaním agentom.

Po vytvorení tejto triedy sa zavolá centrálna metóda render(), ktorá vola metódy ostatných modulov a následne samú seba s 1ms prestávkou.

V render metóde sa zároveň skladá pole obzervácií pre agenta, ktorý kvôli trénovacím wrapperom potrebuje každú 4. snímku. To má za následok že hra môže byť hraná s 60 FPS rýchlosťou ale rady od agenta sa aktualizujú s 15 FPS rýchlosťou aby boli ľahšie registrovateľné hráčom.

# Záver

Tento záver bude rozdelený na niekoľko častí:

* *Trénovanie agenta*
* *Posúdenie implementácie nášho asistovaného programu*
* *Posúdenie úspešnosti/prínosu agenta pri hraní.*

*Trénovanie agenta*

V tejto práci sa nám podarilo úspešne implementovať A2C agenta. Agent je schopný učiť sa a pri správnom nastavení parametrov je učenie rýchle a stabilné. Bohužial ale nastaviť správne parametre je „alchýmia“, v ktorej meníte hodnoty parametrov a pozorujete aký to má vplyv na učenie daného agenta. Niektoré parametre ako napríklad zrážací faktor používajú vo väčšine prípadov rovnakú hodnotu (0,99). Iné parametre ako napríklad rýchlosť učenia alebo počet neurónov a vrstiev v našej neurónovej sieti majú veľké množstvo rôznych konfigurácií, kde niektoré sú lepšie než iné. Zistiť ale tie správne sa dá len pomocou metódy pokus a omyl, čo je v niektorých prípadoch komplikované ako sme demonštrovali v časti *3.4 Trénovanie* kde sa zle nastavené parametre naplno prejavili až okolo epizódy 55 000 a náš agent sa odnaučil kompletne všetko a jeho skóre padlo na hodnotu 60 čo je asi 4x menej ako sa dá dosiahnuť pomocou náhodných akcií. Natrénovanie tejto demonštrácie trvalo približne týžden na hardware popísanom na konci kapitoly, z tohto dôvodu sme nemohli vyskúšať veľké množstvo konfigurácií a ako finálnu sme použili konfiguráciu ktorá po natrénovaní (skoro 2 týždňe) dokáže získať skóre v rozmedzí typicky od 2000 do 3500. Pri týchto skóre sme narazili na strop a náš agent sa prestal ďalej zlepšovať. Agentov štýl hrania je získať čo najviac malých bodíkov čo najrýchlejšie, nesústredil sa špeciálne na zabíjanie duchov alebo zbieranie čerešní, vďaka použitiu osobnitnej odmenovej funkcie. S týmto prístupom sa dá najrýchlejšie vyhrať kolo a postúpiť do ďalšieho. Agent však stále dokáže spraviť chýbný krok a nechať sa zabiť duchom. Tu sme spozorovali že čím menej životov náš agent má tým viac riskuje. Pri poslednom živote zvykne zomrieť do 10 sekúnd. Tak isto čím viac bodíkov zozbiera tým menej jeho ďalšie akcie dávajú zmysel. Pre ďalšie zlepšenie našeho agenta je potrebné vyskúšať ďalšie konfigurácie parametrov.

*Posúdenie implementácie nášho asistovaného programu*

Vďaka použitiu knižnice zameranej na strojové učenie a neurónove siete bolo jednoduché vytvoriť agenta, natrénovať ho a následne uložiť našu natrénovanú sieť a neskôr ju importovať do našej asistovanej aplikácie. Vďaka použitiu wrapperov sme zjedodušili prostredie (zmenšili ho a odstránili farby) pre našeho agenta aby sa mohol učiť efektívnejšie, použili sme ale nesprávny wrapper pre orezovanie obrazu a odrezali sme z vrchu cca 1/5 hracej plochy. Túto chybu sme si uvedomili až po natrénovaní a písaní našej asistovanej aplikácie. Agent sa naučil hrať aj v tomto priestore ktorý nevidí. Myslíme si ale že v prípade použitia správneho wrappera by bolo učenie trocha efektívnejšie, nevieme však do akej miery.

*Posúdenie úspešnosti/prínosu agenta pri hraní.*

Našu aplikáciu sme dali odtestovať 5 ľudom, z čoho dvaja nehrajú hry vôbec. Test bol vykonaný na 10 hrách kde v prvých 5 hrách nemohli používať autopilot ale mohli sledovať rady od agenta a ďalšich 5 hrách mohli používať aj autopilota. Z pozorovaní sme ale zistili že Pacman aj keď ideálny pre demonštráciu (vďaka svojej zložitosti) výkonu A2C algoritmu nijako nebenefituje z jeho použitia. Kedže pri hraní pacmana môže hráč zmeniť smer každú chvíľu a samotná hra si vyžaduje sústredenie, naši hráči nepoužívali rady od agenta ani autopilota.

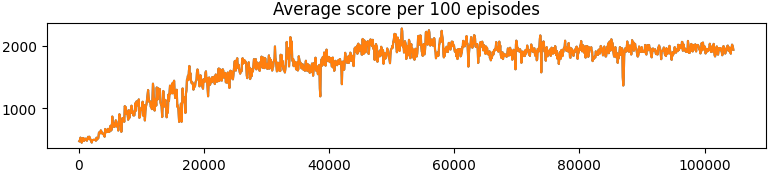
# Zoznam použitej literatúry

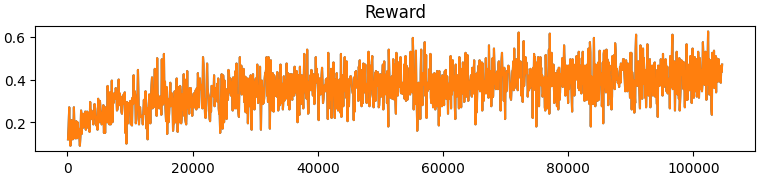
|  |  |
| --- | --- |
| [1] | „sk.wikipedia.org,“ [Online]. Available: https://sk.wikipedia.org/wiki/Moorov\_z%C3%A1kon. |
| [2] | A. G. S. Hado van Hasselt, „https://arxiv.org,“ Google DeepMind, 8 December 2015. [Online]. Available: https://arxiv.org/pdf/1509.06461.pdf. |
| [3] | T. S. H. v. H. L. d. F. Ziyu Wang, „https://arxiv.org,“ Google DeepMind, London, UK, 5 April 2016. [Online]. Available: https://arxiv.org/pdf/1511.06581.pdf. |
| [4] | Shakir, „http://blog.shakirm.com,“ 22 November 2015. [Online]. Available: http://blog.shakirm.com/2015/11/machine-learning-trick-of-the-day-5-log-derivative-trick/. |
| [5] | E. M. L. R. S. Yuhuai Wu, „https://openai.com,“ OpenAI, 18 August 2019. [Online]. Available: https://openai.com/blog/baselines-acktr-a2c/. |
| [6] | O. K. W. D. R. John Schulman, „https://openai.com,“ OpenAI, 20 July 2017. [Online]. Available: https://openai.com/blog/openai-baselines-ppo/. |
| [7] | F. W. D. R. K. John Schulman, „https://arxiv.org,“ OpenAI, 28 August 2017. [Online]. Available: https://arxiv.org/pdf/1707.06347.pdf. |
| [8] | „ https://marutitech.com/top-8-deep-learning-frameworks/,“ [Online]. Available: https://marutitech.com. |
| [9] | D. Causevic, „https://www.toptal.com,“ [Online]. Available: https://www.toptal.com/machine-learning/tensorflow-machine-learning-tutorial. |
| [10] | „https://pytorch.org,“ [Online]. Available: https://pytorch.org/features/. |
| [11] | J. Zhanson, „ http://blog.jzhanson.com,“ 28 May 2018. [Online]. Available: http://blog.jzhanson.com/blog/rl/project/2018/05/28/breakout.html. |
| [12] | „https://gym.openai.com,“ [Online]. Available: https://gym.openai.com/evaluations/eval\_kpL9bSsS4GXsYb9HuEfew/. |
| [13] | „https://paperswithcode.com,“ [Online]. Available: https://paperswithcode.com/sota/atari-games-on-atari-2600-ms-pacman. |

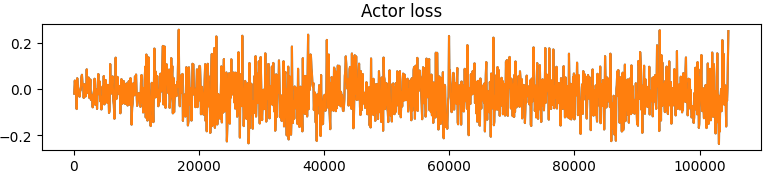
Prílohy

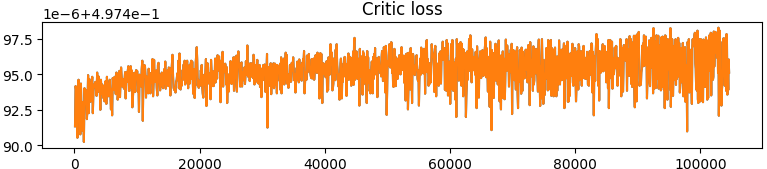
## Príloha A:

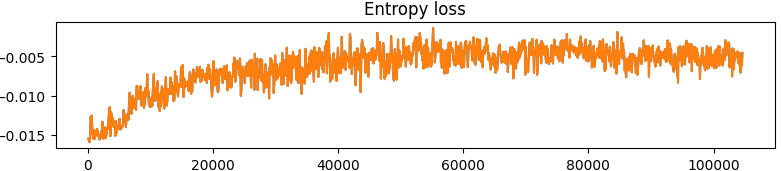
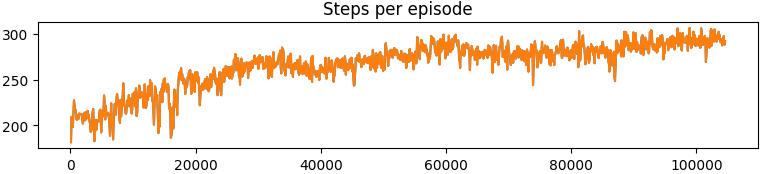
Vizualizácie prvotného agenta:





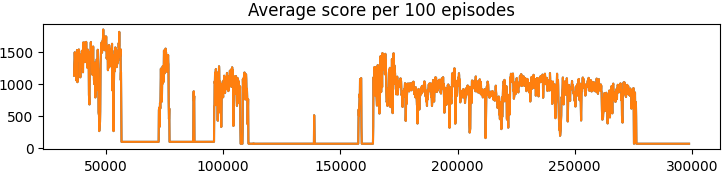


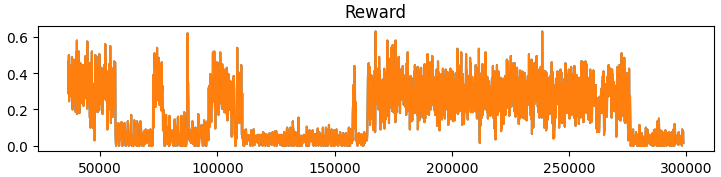


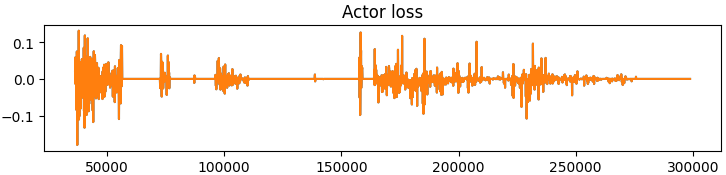
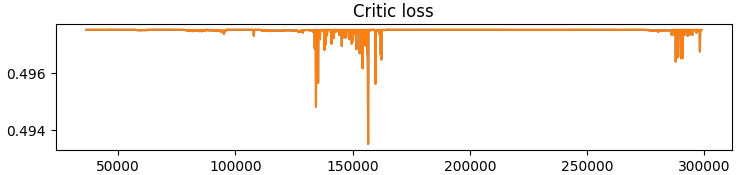
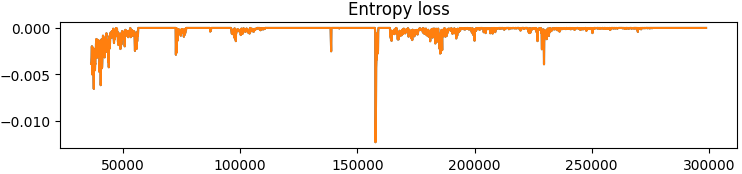
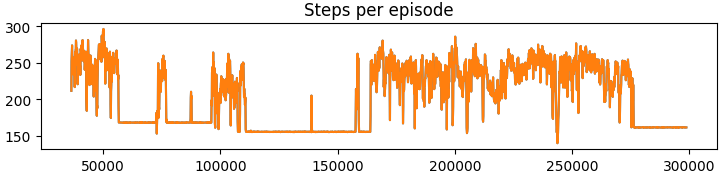
 

## Príloha B:

Vizualizácie zle nastaveného agenta:

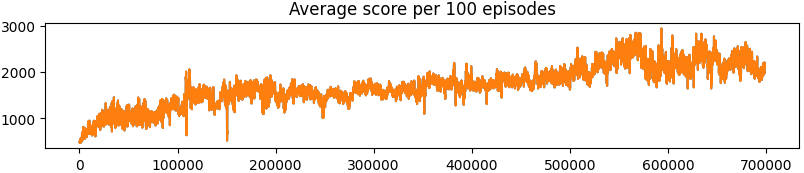
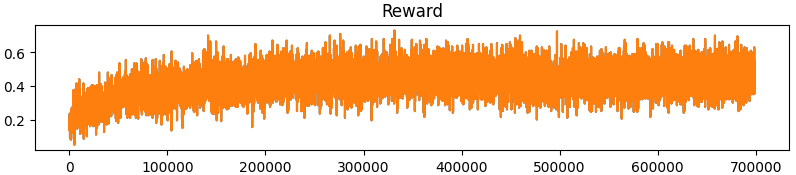
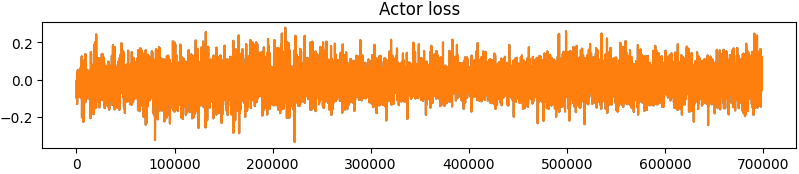
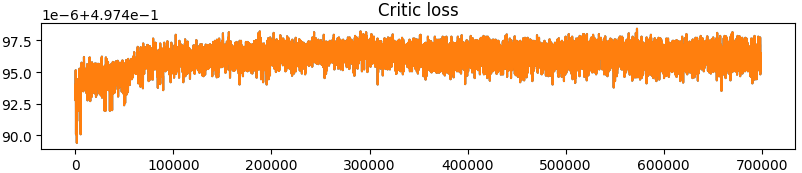
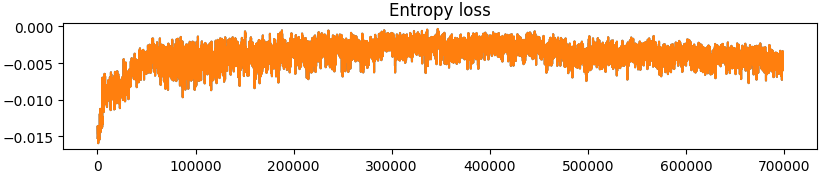
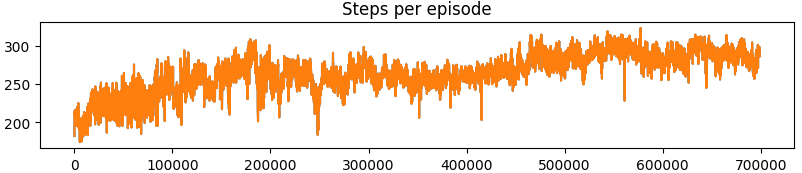




## Príloha C:

Vizualizácie finálneho agenta:

## Príloha D: Obsah Pamäťová karta

Priložená pamäťová karta obsahuje:

* Práca v elektronickej podobe (formát PDF)
* Kód práce a zozbierané štatistiky