Masarykova univerzita
Fakulta informatiky

Hluboké učení v počítačových hrách

Bakalárska práca

Adam Šufliarsky

Brno, jeseň 2019

Masarykova univerzita
Fakulta informatiky

Hluboké učení v počítačových hrách

Bakalárska práca

Adam Šufliarsky

Brno, jeseň 2019

Na tomto mieste sa v tlačenej práci nachádza oficiálne podpísané zadanie práce a vyhlásenie autora školského diela.

Vyhlásenie

Vyhlasujem, že táto bakalárska práca je mojím pôvodným autorským dielom, ktoré som vypracoval samostatne. Všetky zdroje, pramene a literatúru, ktoré som pri vypracovaní používal alebo z nich čerpal, v práci riadne citujem s uvedením úplného odkazu na príslušný zdroj.

Adam Šufliarsky

Vedúci práce: doc. RNDr. Tomáš Brázdil Ph.D.

Poďakovanie

Ďakujem doc. RNDr. Tomášovi Brázdilovi, Ph.D. za vedenie práce a odbornú pomoc.

Zhrnutie

Práca ozrejmuje problematiku hlbokého učenia v počítačových hrách. Predstavuje často používané nástroje v tejto oblasti a popisuje niekoľko už existujúcich riešení. V rámci práce je navrhnutý framework, v ktorom je následne implementovaných niekoľko algoritmov hlbokého učenia, na ktorých sú demonštrované základné postupy a princípy danej oblasti. Práca taktiež experimentálne vyhodnocuje efektívnosť vybraných učiacich algoritmov pri tréningu v rôznych prostrediach (prevažne hrách Atari 2600).

Kľúčové slová

umelá inteligencia, deep learning, Game AI, strojové učenie, počítačové hry

Obsah

Zoznam tabuliek

Zoznam obrázkov

Úvod

Strojové učenie (angl. Machine learning) je vedecká disciplína zaoberajúca sa algoritmami a metódami, ktoré umožňujú programu učiť sa. Program je následne schopný reagovať na rôzne podnety bez toho, aby bol na to explicitne naprogramovaný. Jedným z odvetví strojového učenia je učenie posilňovaním (angl. Reinforcement learning) [1]. Posilňované učenie sa v dnešnej dobe využíva v množstve oblastí, jednou z nich je aj umelá inteligencia v počítačových hrách. Umelá inteligencia je oblasť štúdia v informatike, ktorá simuluje procesy ľudskej inteligencie strojmi, najmä počítačovými systémami. Tieto procesy zahŕňajú učenie, uvažovanie a schopnosť učiť sa z chýb [2]. V kontexte počítačových hier je umelá inteligencia zodpovedná hlavne za adaptívne správanie nehráčskych postáv (NPCs – non-player character) [3]. Cieľom strojového učenia v kontexte umelej inteligencie počítačových hier je schopnosť konkurovať v hrách ľudským hráčom.

Okrem posilňovaného učenia, ktorého vybrané algoritmy sú v rámci práce implementované, existujú aj iné prístupy, ktoré sa dajú v kontexte hlbokého učenia v počítačových hrách využiť. Jedným z nich je supervised learning, v rámci ktorého sú algoritmy trénované za pomoci vstupných dát, v prípade počítačových hier to môžu byť napríklad herné záznamy. Algoritmus sa snaží dosiahnuť schopnosť generalizácie na základe týchto vstupných dát, aby bol schopný korektne reagovať aj na podnety prostredia, ktoré nutne v vstupných dátach neboli [4]. Unsupervised learning na druhej strane hľadá vo vstupných dátach nejaký vzor. Tieto algoritmy analyzujú vlastnosti vstupných dát, ktoré môžu byť následne zoskupované s cieľom vytvoriť úplne nové syntetické dáta s charakteristikami originálnych vstupných dát [4].

Vývojový prístup, ktorého príkladom je neuroevolution, zahŕňa okrem neurónových sietí aj vývojové algoritmy, ktoré majú za úlohu aktualizovať jednotlivé neuróny v sieti [5].

V poslednej dobe sa začali skúmať taktiež hybridné prístupy pri vývoji algoritmov. Hybridný prístup kombinuje metódy hlbokého učenia s inými prístupmi strojového učenia [4].

Cieľom tejto práce je poskytnúť prehľad aktuálneho stavu problematiky strojového učenia v počítačových hrách. Súčasťou práce je

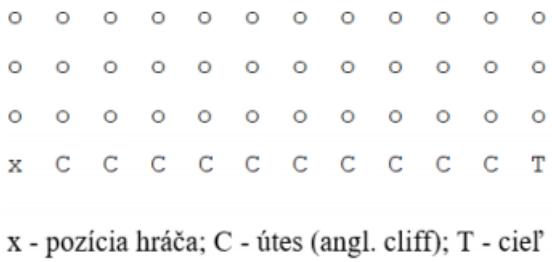
návrh frameworku na uľahčenie implementácie učiacich algoritmov, predstavenie algoritmov Q-learning a SARSA a následná implementácia ich variantov v rôznych prostrediach za pomoci navrhnutého frameworku. Jednotlivé varianty algoritmov sú tiež spúšťané na vybraných hrách Atari 2600 a výsledky týchto meraní sú štatisticky vyhodnotené.

V kapitole 1 sú popísané algoritmy Q-learning a SARSA. Základné princípy sú demonštrované na jednoduchom prostredí. Následne je v kapitole 2 popísaný nástroj OpenAI Gym a jeho základná funkcionalita. Nasleduje kapitola 3, v ktorej sú popísané algoritmy hlbokého učenia vychádzajúce z algoritmov Q-learning a SARSA. Kapitola 4 predstavuje navrhnutý framework, v ktorom sú jednotlivé učiace algoritmy implementované. Postup implementácie podrobnejšie rozoberá kapitola 5. V rámci práce bolo vykonaných aj niekoľko pozorovaní, ktorých výsledky sú prezentované v kapitole 6. V závere práce sú popísané nedoriešené problémy, návrhy na ich riešenia, prínos práce a možnosť jej ďalšieho využitia.

1 Základné učiace algoritmy

1.1 Q-learning

Q-learning je jedným zo základných algoritmov využívaných v odvetví posilňovaného učenia. Písmeno „Q“ v názve znamená kvalita (angl. quality). V danom kontexte reprezentuje kvalitu akcie pri dosahovaní najvyššej možnej odmeny. Algoritmus Q-learning sa učí pomocou vykonaných akcií [6], ktoré sú selektované podľa programátorom určenej taktiky a ohodnotenie akcií je ukladané do dátovej štruktúry, väčšinou tabuľky. Pre účely demonštrácie, ako Q-learning algoritmus funguje, som zvolil prostredie CliffWalking nastroja OpenAI Gym:



Obr. 1.1: Herný plán, CliffWalking

Hrací plán má rozmery 4×12, iniciálna pozícia hráča je $(3, 0)$, pozícia cieľa je $(3, 11)$ a útes je na pozíciách $(3, 1$ až $10)$. Úlohou je dostať sa z iniciálnej pozície do cieľa za čo najmenší počet vykonaných akcií. Každý krok hráča je preto ohodnotený -1. Ak hráč spadne z útesu, akcia je ohodnotená hodnotou -100 a hráč sa vracia na iniciálnu pozíciu. Hra končí, keď hráč dosiahne cieľ.

Pre ľudského hráča by táto hra nepredstavovala žiadnu výzvu, aby ale bol aj Q-learning algoritmus v tejto hre úspešný, musíme si predstaviť niekoľko princípov, ktoré budú následne využívané aj pri implementácii zložitejších algoritmov.

Prvým je model. Model v učiacom algoritme predstavuje dátovú štruktúru, na základe ktorej sa algoritmus rozhoduje, akú akciu v danom stave vykoná. V prípade algoritmu Q-learning, je pre tieto účely postačujúca náhľadová tabuľka. Rozmery náhľadovej tabuľky závisia na prostredí, formát je daný $m \times n$, kde m predstavuje počet

všetkých možných stavov a n predstavuje počet možných akcií. V prípade prostredia CliffWalking je rozmer tabuľky 48 riadkov (počet stavov) a 4 stĺpce (počet možných akcií – hráč sa môže pohybovať štyrmi smermi).

Údaje náhľadovej tabuľky sa získavajú a aktualizujú prieskumom prostredia a využívajú pri prechode hrou (angl. exploration and exploitation). Je veľa možností ako algoritmus bude voliť akciu, ktorá sa v aktuálnom stave vykoná. Najpoužívanejší a taktiež mnou zvolený prístup je ϵ-greedy. ϵ určuje, s akou pravdepodobnosťou agent vykoná náhodnú akciu, čím získa viac informácií o prostredí. Na začiatku behu programu je pravdepodobnosť vykonania náhodnej akcie 1 a postupne sa znižuje, až kým nedosiahne definovanú hodnotu 0,1. V tomto okamihu by mal agent mať už dostatok informácií o prostredí. Učiaci proces prebieha v cykle, je len na programátorovi, či bude pravdepodobnosť náhodnej akcie znižovať po každej vykonanej akcii, alebo až po každej epizóde (hre). Algoritmus teda postupom času čím ďalej, tým viac využíva svoju náhľadovú tabuľku a keď volí nenáhodnú akciu, „pozrie“ do tabuľky na riadok (stav), v ktorom sa aktuálne nachádza a vyberie stĺpec (akciu) s najvyššou hodnotou (Q-value).

Vybratá akcia sa vykoná v nasledujúcom kroku, prostredie sa dostane do nasledujúceho stavu, ostáva už len ohodnotenie akcie. Pre ohodnotenie akcie sa vychádza z Bellmanovej rovnosti a toto ohodnotenie sa nazýva Q-value:

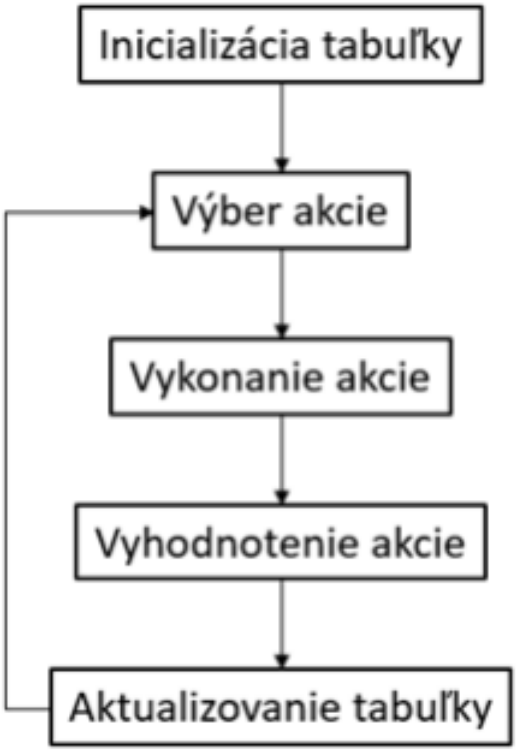
$$Q(s,a) = Q(s,a) + \alpha\left[r + \gamma max_{a'} Q(s',a') - Q(s,a)\right] \tag{1.1}$$

kde a' predstavuje najlepšiu akciu v nasledujúcom stave s'.

α (Learning rate) je hodnota v intervale <0, 1>, ktorá predstavuje rýchlosť učenia, t. j. do akej miery sú hodnoty v tabuľke aktualizované [7].

γ (Discount factor) je hodnota tiež v intervale <0, 1>, určujúca podstatu dlhodobej odmeny. Čím je hodnota bližšie k nule, tým viac sa agent bude sústrediť na okamžitú odmenu [7].

Ako už bolo spomenuté, učiaci proces prebieha v cykle a jeho jednotlivé časti sa dajú jednoznačne pomenovať. Proces učenia Q-learning algoritmu ilustruje nasledujúci obrázok:



Obr. 1.2: Učiaci proces

1.2 SARSA

SARSA je akronym pre State-Action-Reward-State-Action. Na rozdiel od algoritmu Q-learning, v algoritme SARSA máme vopred určenú taktiku ako sa jednotlivé akcie mapujú na jednotlivé stavy počas učiaceho procesu. Vytvárajú sa teda dvojice stav-akcia. Učiaci proces je veľmi podobný ako u algoritmu Q-learning, líšia sa len v aktualizácii modelu, ktorá taktiež vychádza z Bellmanovej rovnosti [8].

$$Q(s,a) = Q(s,a) + \alpha\left[r + \gamma Q(s',a') - Q(s,a)\right] \tag{1.2}$$

kde a' predstavuje akciu zvolenú na základe nejakej stratégie (napr. ϵ-greedy) v nasledujúcom stave s'.

Najlepšie je tento rozdiel vidieť v pseudokóde jednotlivých algoritmov.

Algorithm 1: Q-learning [8]

Inicializuj model;

foreach *epizódu* **do**
  $S \leftarrow$ iniciálny stav prostredia;
  repeat
    Vyber akciu A v stave S na základej nejakej stratégie;
    Vykonaj akciu A a zaznamenaj spätnú väzbu od prostredia - odmenu R, nasledujúci stav S';
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma max_a Q(S',a) - Q(S,A)\big]$;
    $S \leftarrow S'$;
  until *S je konečný stav*;

end

Algorithm 2: SARSA [8]

Inicializuj model;

foreach *epizódu* **do**
  $S \leftarrow$ iniciálny stav prostredia;
  Vyber akciu A v stave S na základej nejakej stratégie;
  repeat
    Vykonaj akciu A a zaznamenaj spätnú väzbu od prostredia - odmenu R, nasledujúci stav S';
    Vyber akciu A' v stave S' na základej nejakej stratégie;
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S',A') - Q(S,A)\big]$;
    $A \leftarrow A'$;
    $S \leftarrow S'$;
  until *S je konečný stav*;

end

V oboch algoritmoch sa akcia vyberá na základe ϵ-greedy stratégie.

2 Nástroj OpenAI Gym

V rámci bakalárskej práce som využíval súbor nástrojov OpenAI Gym[1] (ďalej Gym), ktorý slúži predovšetkým na vývoj a porovnávanie algoritmov posilňovaného učenia a je plne kompatibilný s rôznymi výpočtovými knižnicami. Gym ponúka zbierku prostredí, na ktorých je možné učiace algoritmy spúšťať. Prostredím môže byť široké spektrum hier, od tých najjednoduchších textových hier, až po komplexné hry Atari 2600.

2.1 Základná funkcionalita

V nástroji Gym je implementovaných niekoľko metód s príslušnou funkcionalitou. Na predstavenie metód a ich návratových hodnôt slúži implementácia agenta, ktorý sa nijak neučí a vykonáva len náhodné akcie:

```
import gym

env = gym.make('MountainCar-v0')
env.reset()
done = False

while not done:
    env.render()
    action = env.action_space.sample()
    state, reward, done, info = env.step(action)

env.close()
```

Pomocou metódy `gym.make` je vytvorené prostredie uložené do premennej `env`, s ktorým neskôr interagujeme. Vstupom tejto metódy je meno prostredia. Každé prostredie má suffix, ktorý označuje pravdepodobnosť opakovania poslednej akcie. Suffix `v0` predstavuje pravdepodobnosť 0,25 a suffix `v4` predstavuje nulovú pravdepodobnosť, že namiesto novej akcie bude zopakovaná tá predošlá. Pred suffix

1. `https://gym.openai.com/docs/`

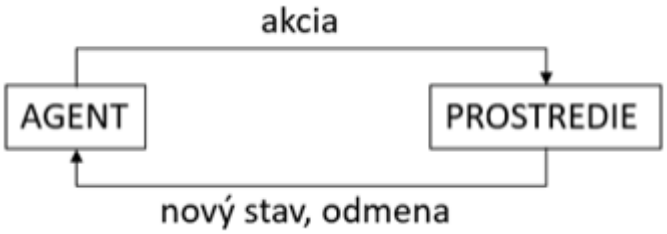
je možné pridať kľúčové slová ako Deterministic a NoFrameskip, ktoré určujú, koľko herných snímkov bude preskočených (Deterministic – 4, NoFrameskip – 0, bez kľúčového slova – výber z intervalu <2, 5>).

Metóda `env.reset` dostáva prostredie do iniciálneho stavu a tento stav vracia ako návratovú hodnotu. V danom agentovi nepotrebujeme s iniciálnym stavom prostredia pracovať, preto si návratovú hodnotu tejto metódy neukladáme.

V hlavnom cykle metóda `env.render` zabezpečuje vykresľovanie stavu prostredia na obrazovku. Táto metóda sa zvykne využívať až po tréningu agenta. Do premennej `action` ukladáme akciu, ktorá bude agentom vykonaná. Slúži na to metóda `env.action_space.sample`. Akcia vrátená touto metódou je náhodná a keďže sa pohybujeme v diskrétnom priestore, typ návratovej hodnoty je celé číslo. Konkrétne v prostredí MountainCar sú akcie kódované nasledovne: 0 - pohyb doľava, 2 - pohyb doprava.

Najvýznamnejšou metódou hlavného cyklu je metóda `env.step`. Ako vstup berie akciu, ktorú agent v danom prostredí vykoná. Výstupom tejto metódy je štvorica. Prvý člen štvorice reprezentuje stav, do ktorého sa prostredie dostalo po vykonaní akcie zadanej vstupom. Druhým členom je odmena, ohodnotenie akcie v stave, v ktorom sa prostredie nachádzalo pred zavolaním funkcie `env.step`. Ohodnotenie je celé číslo s kladnou hodnotou, keď je dosiahnutý cieľ a so zápornou, ak je úlohou dosiahnuť cieľ čo najrýchlejšie. Takéto ohodnotenie sme mohli sledovať pri prostredí CliffWalking, na ktorom sa demonštroval algoritmus Q-learning. Tretím členom štvorice je indikátor konca epizódy, ktorý má hodnotu `True` alebo `False`. Koncom epizódy napríklad môže byť strata všetkých životov. V hore uvedenom kóde koniec epizódy ukončuje hlavný cyklus. Posledným členom štvorice je slovník s informáciami pre programátora ako napríklad počet ostávajúcich životov a pod.

Hlavný cyklus implementuje agent-environment loop, ktorý modeluje, ako prebieha učiaci proces [9]:

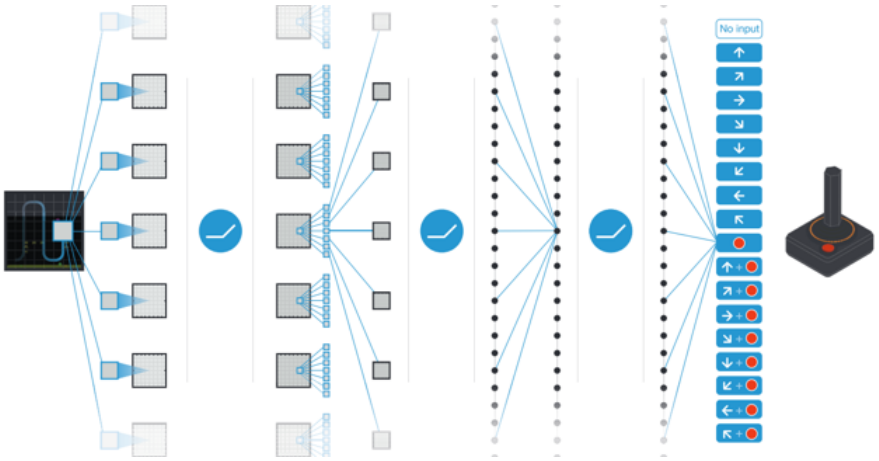


Obr. 2.1: Agent-environment loop

Po hlavnom cykle nasleduje metóda `env.close`, ktorá zavrie vykresľovacie okno. Táto metóda nemá ani vstupnú, ani výstupnú hodnotu.

3 Algoritmy hlbokého učenia

Prívlastok „deep“ v kontexte učiacich algoritmov naznačuje, že dátová štruktúra modelu algoritmu je iného, zložitejšieho typu ako len náhľadová tabuľka, ako tomu bolo v základných verziách algoritmov Q-learning a SARSA. Algoritmy s prívlastkom „deep“ využívajú ako svoj model jednu, alebo viac neurónových sietí. Tieto algoritmy sú teda schopné naučiť sa aj pravidlá zložitejšieho prostredia, ako sú len textové hry (prostredie CliffWalking). V rámci práce budú algoritmy spúšťané na vybraných hrach Atari 2600, ktorých stav nie je reprezentovaný len jedným celým číslom, ako tomu bolo u prostredia CliffWalking, ale reprezentuje ho herná obrazovka. Práve na základe herných snímkov neurónová sieť priraďuje jednotlivým akciám Q-value.



Obr. 3.1: Proces vyhodnocovania stavu neurónovou sieťou [10]

3.1 Deep Q-learning

Deep Q-learning je nosný algoritmus hlbokého učenia v počítačových hrách. Ako napovedá názov, vychádza zo základného algoritmu Q-learning. Samotná technika - využitie neurónových sietí v učiacich algoritmoch, už bola predstavená v druhej polovici 20. storočia [11]. Prevratné bolo využitie tejto techniky v kontexte počítačových hier.

V roku 2013 spoločnosť DeepMind zverejnila vedecký článok s názvom „Playing Atari with Deep Reinforcement Learning“. V článku demonštrovali, ako sa počítač naučil hrať hry Atari 2600 iba za pomoci pixelov hernej obrazovky a odmien v podobe zmien herného skóre. Algoritmus bol spustený na siedmich hrách a dokonca až v troch dosiahol vyššie skóre ako človek [12].

Ďalší článok „Human-level control though deep reinforcement learning“ od rovnakej spoločnosti, bol uverejnený v roku 2015 v britskom vedeckom časopise Nature [13]. V tomto článku rozšírili výskum na vzorku 49 rôznych hier a vo vyše polovici z nich algoritmus dosiahol skóre vyššie ako človek [10].

Postupom času boli na tomto algoritme implementované rôzne vylepšenia, ktoré zlepšovali algoritmu schopnosť učiť sa v niektorých špecifických prostrediach [14, 15, 16]. Základný Deep Q-learning algoritmus sa dá popísať v štyroch krokoch:

1. Zozbieraj a ulož informácie o prostredí do replay bufferu
2. Vyber náhodnú vzorku informácií z replay bufferu (taktiež známe ako Experience replay)
3. Použi vzorku na aktualizovanie neurónovej siete
4. Opakuj 1. - 3.

3.1.1 Reprezentácia herného stavu

Jeden herný snímok nie je postačujúci, nezachytáva pohyb herných objektov (napr. vystrelených projektilov). Bez informácie o tom, ako sa herné objekty pohybujú, by hlboké algoritmy nevedeli správne odhadnúť, aká akcia je v danom stave najideálnejšia. Dá sa to ilustrovať na hre Pong [17], kde cieľom je odraziť na protihráča objekt tak, aby ho nebol schopný odraziť naspäť. Jedna snímka by nemala žiadnu výpovednú hodnotu. Nedalo by sa určiť, či bol objekt odrazený hráčom a letí ku protivníkovi, alebo práve naopak. Na správne vyhodnotenie, akú akciu vykonať, je potrebné tieto informácie poznať. Riešením tohto problému je reprezentovať stav nie jedným, ale štyrmi za sebou idúcimi hernými snímkami, v ktorých sú zahrnuté všetky potrebné informácie prostredníctvom zachytenej zmeny polohy pohybujúcich sa objektov.

3.1.2 Experience replay

Experience Replay (ďalej ER) spočíva v tom, že namiesto aktualizovania modelu (neurónovej siete) na základe len posledného kroku, sú zaznamenávané aj ďalšie relevantné informácie, a to aktuálny stav, nasledujúci stav, odmena, vykonaná akcia a údaj o tom, či je nasledujúci stav konečný. Tieto informácie sa vo vhodnej dátovej štruktúre ukladajú do pamäte konečnej dĺžky. Z tejto pamäte sú vyberané náhodné, nie nutne najnovšie pridané vzorky, na ktorých sa algoritmus učí.

ER je dôležitý preto, že za sebou nasledujúce stavy, si sú veľmi podobné, majú vysokú koreláciu. Líšia sa len vo veľmi malom počte pixelov. Je preto potrebné, aby sa na vstup neurónovej siete dostávali aj staršie stavy, inak by sa učenie zneefektívnilo [10]. Inak povedané, ak je cieľom hry dostať sa do čo najvyššej úrovne, pričom každá úroveň je iná, je nevyhnutné, aby algoritmus „nezabudol" optimálnu stratégiu pre každú z úrovní. V tom algoritmu pomáhajú práve menej aktuálne vzorky, ktoré sú z tejto pamäte náhodne vyberané a predkladané ako vstup neurónovej siete. Takáto pamäť má však veľké pamäťové nároky.

3.1.3 Aktualizácia neurónovej siete

Zatiaľ čo v Q-learning algoritme sme pri aktualizácii Q-value pre daný stav vychádzali z Bellmanovej rovnosti:

$$Q(s,a) = Q(s,a) + \alpha\left[r + \gamma max_{a'} Q(s',a') - Q(s,a)\right] \tag{3.1}$$

kde a' predstavovalo najlepšiu akciu v nasledujúcom stave s', teraz potrebujeme aktualizovať neurónovú sieť. $Q(s,a;\theta)$ už nie je náhľadová tabuľka, ale neurónová sieť s parametrami danými vektorom θ. Cieľom je minimalizovať mean squared error medzi cieľovou a aktuálnou Q-value:

$$L(\theta_i) = \left[(y_i - Q(s,a;\theta_i))^2\right] \tag{3.2}$$

kde

$$y_i = \begin{cases} r \text{ (odmena)} & \text{pre konečný stav} \\ r + \gamma max_{a'} Q(s',a';\theta_{i-1}) & \text{inak} \end{cases} \tag{3.3}$$

čo predstavuje predpis spracovania jednotlivých prvkov vzorky v i-tej iterácii.

Ideálne chceme, aby sa chyba znižovala, teda chceme, aby výstupy neurónovej siete boli čo najbližšie k true Q-values. Preto vykonáme gradient step na stratovej funkcii definovanej vyššie:

$$\nabla_{\theta_i} L(\theta_i) = \alpha\left[\left(r + \gamma max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i)\right) \nabla_{\theta_i} Q(s, a; \theta_i)\right] \tag{3.4}$$

Celý priebeh Deep Q-learning algoritmu je prehľadne zdokumentovaný nasledujúcim pseudokódom:

Algorithm 3: Deep Q-learning + Experience replay [12]

Inicializuj replay buffer;

Inicializuj model;

for *epizóda = 1, M* **do**

  $S_1 \leftarrow$ iniciálna sekvencia 4 herných snímkov $\{x_1\}$;

  $\phi_1 \leftarrow$ spracovaná sekvencia, predstavujúca herný stav $\phi(S_1)$;

  for $t = 1, T$ **do**

    Zvoľ a_t ϵ-greedy metódou;

    Vykonaj akciu a_t a zaznamenaj spätnú väzbu od prostredia - odmenu r_t, ďalší snímok x_{t+1};

    $\phi_{t+1} \leftarrow \phi(S_{t+1})$;

    Ulož dáta $(\phi_t, a_t, r_t, \phi_{t+1})$ do replay bufferu;

    Vyber vzorku prechodov $(\phi_j, a_j, r_j, \phi_{j+1})$ z replay bufferu;

    Nastav

$$y_i = \begin{cases} r_j & \text{pre konečný stav } \phi_{j+1} \\ r_j + \gamma max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{inak} \end{cases};$$

    Vykonaj gradient step $(y_i - Q(\phi_j, a_j; \theta))^2$

  end

end

3.2 Deep double Q-learning

Operátor `max` v základnom Q-learning a Deep Q-learning algoritme využíva rovnaké hodnoty na výber a ohodnotenie akcie. Q-learning a Deep Q-learning algoritmy sú preto náchylné na výber nadhodnotenej akcie (akcie, ktorá má v porovnaní s inými nadštandardne vysoké ohodnotenie), a to vedie k príliš optimistickým odhadom hodnoty Q-value [15]. Myšlienkou Deep double Q-learning algoritmu je dekompozícia výberu a ohodnotenia akcie. Presnosť ohodnotení (Q-values) závisí na akciách, ktoré algoritmus vykonal a na susedných stavoch, ktoré algoritmom už boli preskúmané. Na začiatku tréningu algoritmus nemá dostatok informácií o prostredí a teda nevie, akú akciu má zvoliť. Môže sa teda stať, že algoritmus bude vyberať nevhodné akcie za predpokladu, že je využívaná ϵ-greedy stratégia. Takto vybraté akcie môžu byť postupom času ohodnotené lepšie, ako akcie optimálne.

Deep double Q-learning algoritmus tento problém rieši využitím dvoch neurónových sietí s cieľom oddeliť výber akcie od výpočtu Q-value [15]:

$$y_i^{DoubleQ} = \begin{cases} r & \text{pre konečný stav} \\ r + \gamma Q(s', argmax_a Q(s', a'; \theta_i); \theta_{i-1}) & \text{inak} \end{cases} \tag{3.5}$$

čo predstavuje predpis spracovania jednotlivých prvkov vzorky v i-tej iterácii a a' predstavuje najlepšiu akciu v nasledujúcom stave s'.

Výpočet prebieha v niekoľkých krokoch. Neurónová sieť Q-network počíta ohodnotenie každej akcie v stave s', ktorý predstavuje nasledujúci stav po vykonaní akcie a. Z vypočítaných hodnôt sa vyberie najvyššia, ktorá predstavuje akciu najvyššej kvality. Táto akcia je následne ohodnotená neurónovou sieťou Target-network a výsledok predstavuje ohodnotenie danej akcie. Neurónová sieť Target-network nie je ale aktualizovaná tak často ako Q-network, ale aktualizujeme ju v pravidelných intervaloch každých n krokov. Tento počet krokov je zadaný hyperparametrom.

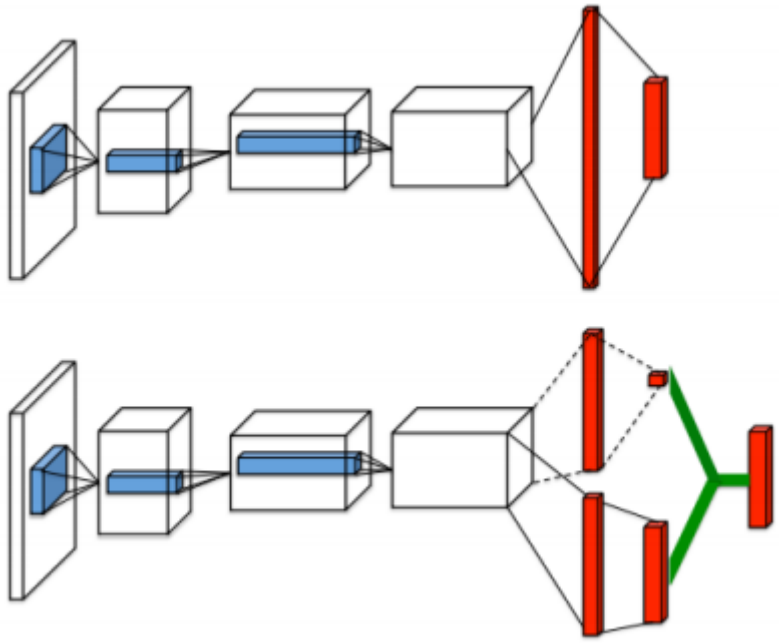
Oproti základnému Deep Q-learning algoritmu táto úprava znižuje nadhodnocovanie akcií a pomáha k stabilnejšiemu a v dlhých pozorovaniach, efektívnejšiemu tréningu [15].

3.3 Architektúra dueling deep Q-network

Q-value v doteraz predstavených algoritmoch reprezentovala kvalitu akcie v danom stave. Architektúra dueling deep Q-network Q-value rozdelila na súčet dvoch hodnôt, z ktorých prvá reprezentuje výhodu, o koľko lepšie je vykonať určitú akciu oproti ostatným akciám a druhá reprezentuje kvalitu stavu [16].

$$Q(s,a;\theta,\alpha,\beta) = A(s,a;\theta,\alpha) + V(s;\theta,\beta) \tag{3.6}$$

Základným princípom Architektúry dueling deep Q-network je mať neurónovú sieť, ktorá oddelene počíta tieto hodnoty a kombinuje ich do jednej až v poslednej vrstve.



Obr. 3.2: Architektúra dueling network [16]

Výhoda Architektúry dueling deep Q-network oproti klasickému Deep Q-learning algoritmu je hlavne v prípadoch, keď množstvo možných akcií je veľké. Algoritmus, ktorý využíva túto architektúru, dokáže ohodnotiť stav bez toho, aby poznal efekt každej akcie v danom stave. Toto je užitočné hlavne v prípade, keď akcie v danom stave

žiadnym zásadným spôsobom neovplyvňujú prostredie. Nie je teda potrebné počítať hodnotu každej akcie [16].

Bolo to prezentované na hre Enduro, v ktorej hráč ovláda auto a jeho cieľom je vyhýbať sa oproti idúcim autám. V tejto hre je potrebné, aby algoritmus reagoval, iba keď sa oproti idúce auto objaví na horizonte. Nie je potrebné ohodnocovať akcie, keď žiadne auto idúce v protismere nie je [16].

Nech α je parameter vetvy siete počítajúcej výhodu, nech β je parameter vetvy siete počítajúcej kvalitu. Potom $Q(s, a; \theta, \alpha, \beta)$ predstavuje ohodnotenie akcie a v stave s. Výpočet prebieha nasledovne:

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + \left(A(s, a; \theta, \alpha) - \frac{1}{\mid A \mid} \sum_{a'} A(s, a'; \theta, \alpha) \right) \tag{3.7}$$

Parametre α a β predstavujú streamy. Stream α (advantage stream) odkazuje na parametre špecifické pre Advantage-network, sieť počítajúcu výhodu danej akcie oproti ostatným akciám. Stream β (value stream) odkazuje na parametre špecifické pre Value-network, sieť počítajúcu kvalitu stavu. Využitím popísanej architektúry by sa mal urýchliť tréning, keďže vieme ohodnotiť stav bez počítania ohodnotenia všetkých akcií v danom stave [16].

3.4 Deep SARSA

Ako algoritmus Deep Q-learning, aj Deep SARSA využíva ako svoj model neurónovú sieť, ktorej vstupom je stav prostredia reprezentovaný štyrmi po sebe idúcimi hernými snímkami a výstupom sú Q-values pre každú akciu [18]. Líši sa ale v dvoch veciach.

Do replay bufferu sa ukladajú dáta v šestici. K pôvodnej pätici, ktorá v algoritme Deep Q-learning pozostávala z aktuálneho stavu, nasledujúceho stavu, odmeny, aktuálne vykonanej akcie a údaja o tom, či je nasledujúci stav konečný, sa pridáva ešte údaj o nasledujúcej vykonanej akcii.

Pozmenená je aj stratová funkcia využívaná pri aktualizácii neurónovej siete. V prípade algoritmu Deep SARSA stratová funkcia vyzerá nasledovne:

$$L(\theta_i) = \left[(y_i - Q(s, a; \theta_i))^2\right] \tag{3.8}$$

kde

$$y_i = \begin{cases} r \text{ (odmena)} & \text{pre konečný stav,} \\ r + \gamma Q(s', a'; \theta_{i-1}) & \text{inak.} \end{cases} \tag{3.9}$$

čo predstavuje predpis spracovania jednotlivých prvkov vzorky v i-tej iterácii a a' predstavuje akciu zvolenú na základe nejakej stratégie (napr. ϵ-greedy) v nasledujúcom stave s'.

Vykonávaný gradient step na stratovej funkcii vyzerá nasledovne:

$$\nabla_{\theta_i} L(\theta_i) = \alpha\left[\left(r + \gamma Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i)\right) \nabla_{\theta_i} Q(s, a; \theta_i)\right] \tag{3.10}$$

Algoritmus Deep SARSA je popísaný nasledujúcim pseudokódom:

Algorithm 4: Deep SARSA + Experience replay [18]

Inicializuj replay buffer;

Inicializuj model;

for *epizóda = 1, M* **do**

  $S_1 \leftarrow$ iniciálna sekvencia 4 herných snímkov $\{x_1\}$;

  $\phi_1 \leftarrow$ spracovaná sekvencia, predstavujúca herný stav $\phi(S_1)$;

  Zvoľ a_1 ϵ-greedy metódou;

  for $t = 1, T$ **do**

    Vykonaj akciu a_t a zaznamenaj spätnú väzbu od prostredia - odmenu r_t, ďalší snímok x_{t+1};

    $\phi_{t+1} \leftarrow \phi(S_{t+1})$;

    Ulož dáta $(\phi_t, a_t, r_t, \phi_{t+1})$ do replay bufferu;

    Vyber vzorku prechodov $(\phi_j, a_j, r_j, \phi_{j+1})$ z replay bufferu;

    Zvoľ a' ϵ-greedy metódou;

    Nastav $y_i = \begin{cases} r_j & \text{pre konečný stav } \phi_{j+1} \\ r_j + \gamma Q(\phi_{j+1}, a'; \theta) & \text{inak} \end{cases}$;

    Vykonaj gradient step $(y_i - Q(\phi_j, a_j; \theta))^2$;

    $a_t \leftarrow a'$

  end

end

3.5 Deep double SARSA

Tak isto ako algoritmus Deep double Q-learning, aj algoritmus Deep double SARSA využíva v procese učenia dve neurónové siete [19]. Dôvod je totožný ako v algoritme Deep double Q-learning. Rozdiel je ale vo výpočte, kde znova vynechávame operátor max, ako tomu bolo v klasickom algoritme SARSA:

$$y_i^{DoubleSARSA} = \begin{cases} r & \text{pre konečný stav} \\ r + \gamma Q(s', Q(s', a'; \theta_i); \theta_{i-1}) & \text{inak} \end{cases} \tag{3.11}$$

čo predstavuje predpis spracovania jednotlivých prvkov vzorky v i-tej iterácii a a' predstavuje akciu zvolenú na základe nejakej stratégie (napr. ϵ-greedy) v nasledujúcom stave s'.

Samozrejme aj v algoritme Deep double SARSA sa do replay bufferu ukladajú dáta vo forme šestíc, ako tomu bolo v algoritme Deep SARSA.

4 Vytvorený framework

4.1 Návrh

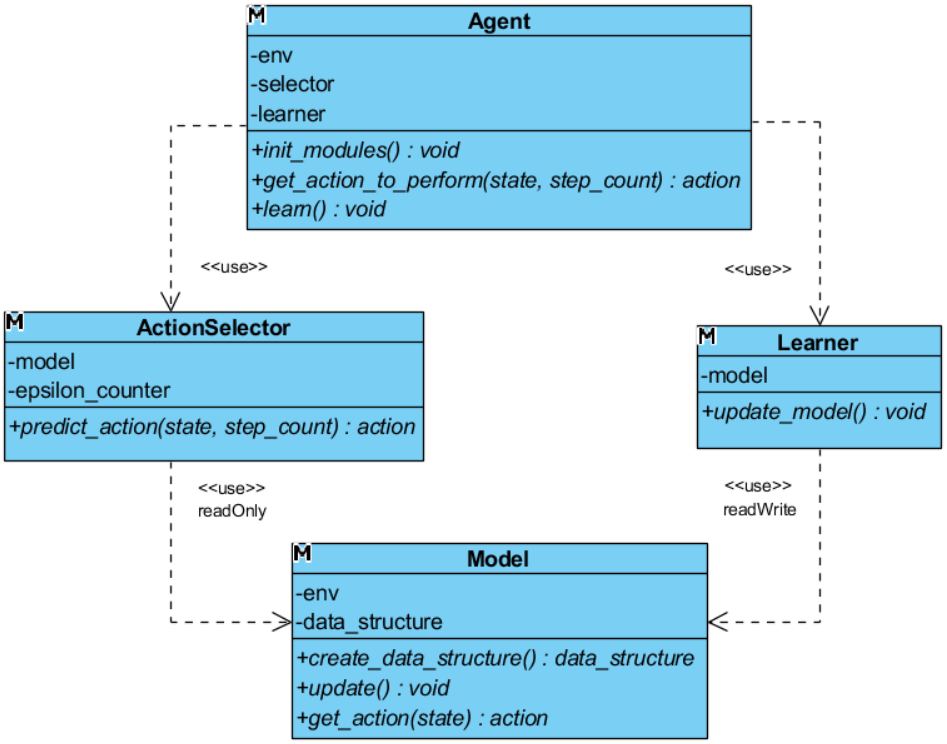
Pred samotnou implementáciou jednotlivých algoritmov som za účelom lepšej prehľadnosti a modularity navrhol framework pozostávajúci zo štyroch tried. Každá trieda zabezpečuje unikátnu funkcionalitu a tieto triedy medzi sebou komunikujú prostredníctvom volania metód.

Základnou jednotkou je trieda Agent, ktorý sa učí hrať hru. Tento Agent obsahuje nasledujúce moduly:

- Rozhodovací modul (ActionSelector) - selektuje akcie na základe modelu
- Učiaci modul (Learner) - zhromažďuje a predkladá informácie o spätnej väzbe z hry učiacemu algoritmu

Každý z týchto menovaných modulov využíva ešte modul Model, ktorý obsahuje dátovú štruktúru a rozhranie na prácu s ňou, za pomoci ktorej sa agent učí.

Agent, ako aj moduly majú definované rozhranie pozostávajúce z niekoľkých metód, ktoré zaisťujú vyššie opísanú funkcionalitu. Objektový návrh frameworku, ako aj vzťah medzi triedami a spomínané rozhrania znázorňuje nasledujúci class diagram:

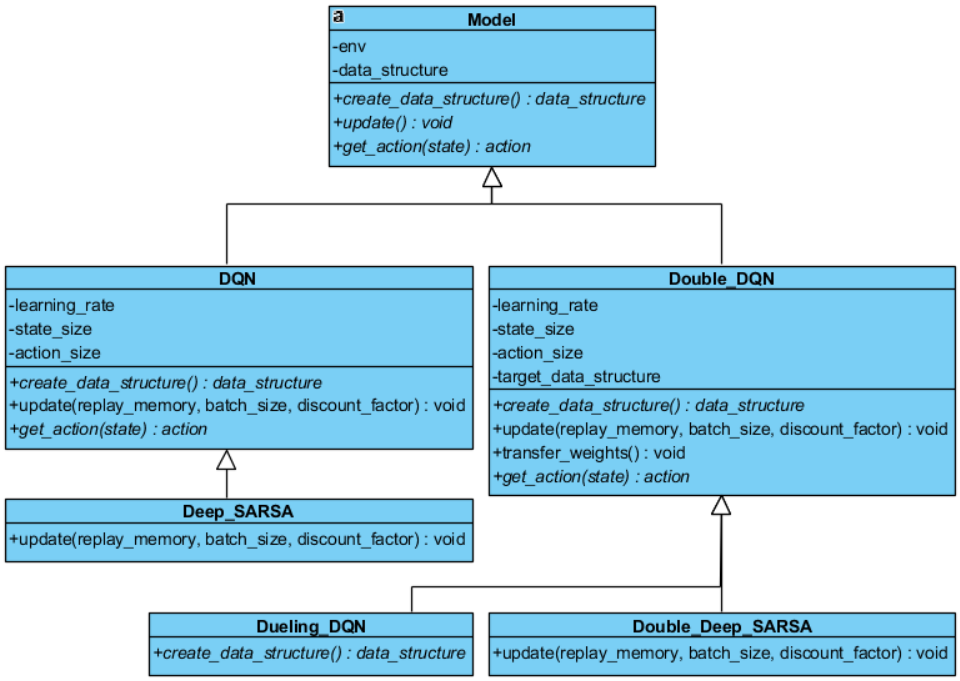


Obr. 4.1: Objektový návrh frameworku

4.2 Objektová hierarchia

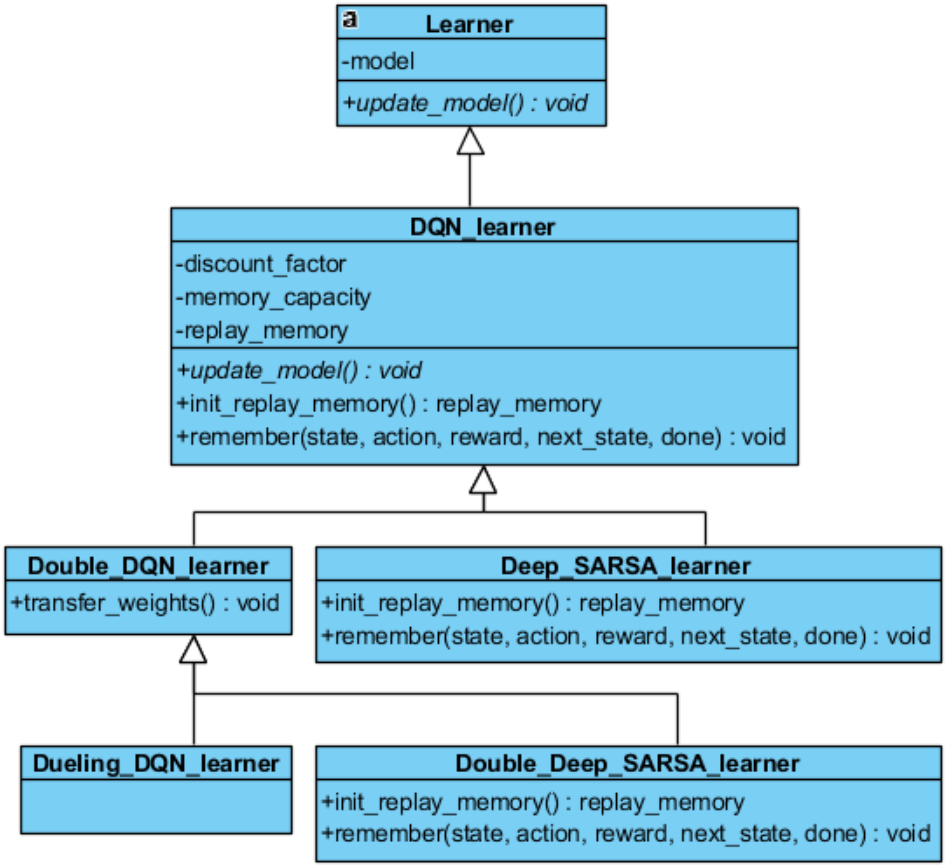
Niektoré moduly, ale aj samotní agenti algoritmov menovaných v kapitole 3, majú veľmi podobnú funkcionalitu, preto bolo možné v daných moduloch zdediť už existujúcu funkcionalitu a len minimálne ju rozšíriť. V tejto kapitole znázorním objektovú hierarchiu jednotlivých modulov agenta, ale aj agenta samotného. Na jednotlivých diagramoch sú znázornené aj prepisovania jednotlivých metód a pridávanie dodatočných atribútov.

Modul Model sa najzásadnejšie líši v algoritmoch, ktoré využívajú odlišný počet neurónových sietí, preto sa aj objektová hierarchia rozvetvila práve na algoritmy s jednou a dvoma neurónovými sieťami.



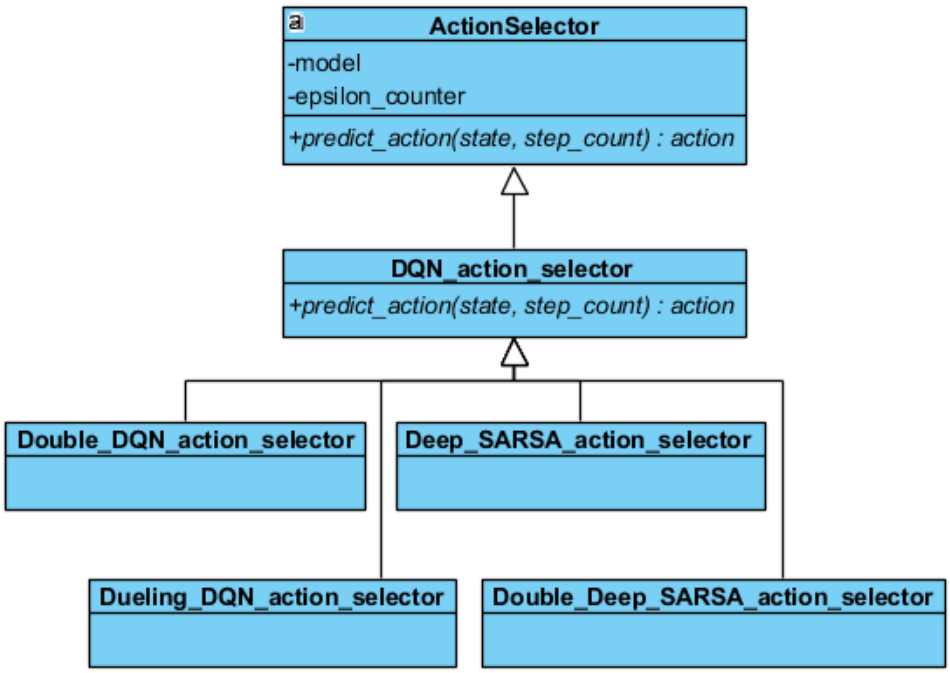
Obr. 4.2: Objektová hierarchia, Model

V učiacom module je väčšina potrebnej funkcionality zahrnutá už v učiacom module algoritmu Deep Q-learning. Väčšiu úlohu ako počet neurónových sietí tu hrá, či sa jedná o variant Q-learning alebo SARSA algoritmu. Ako bolo spomínané v sekcii 3.4, algoritmus SARSA ukladá viac informácií do pamäte ako algoritmus Q-learning.



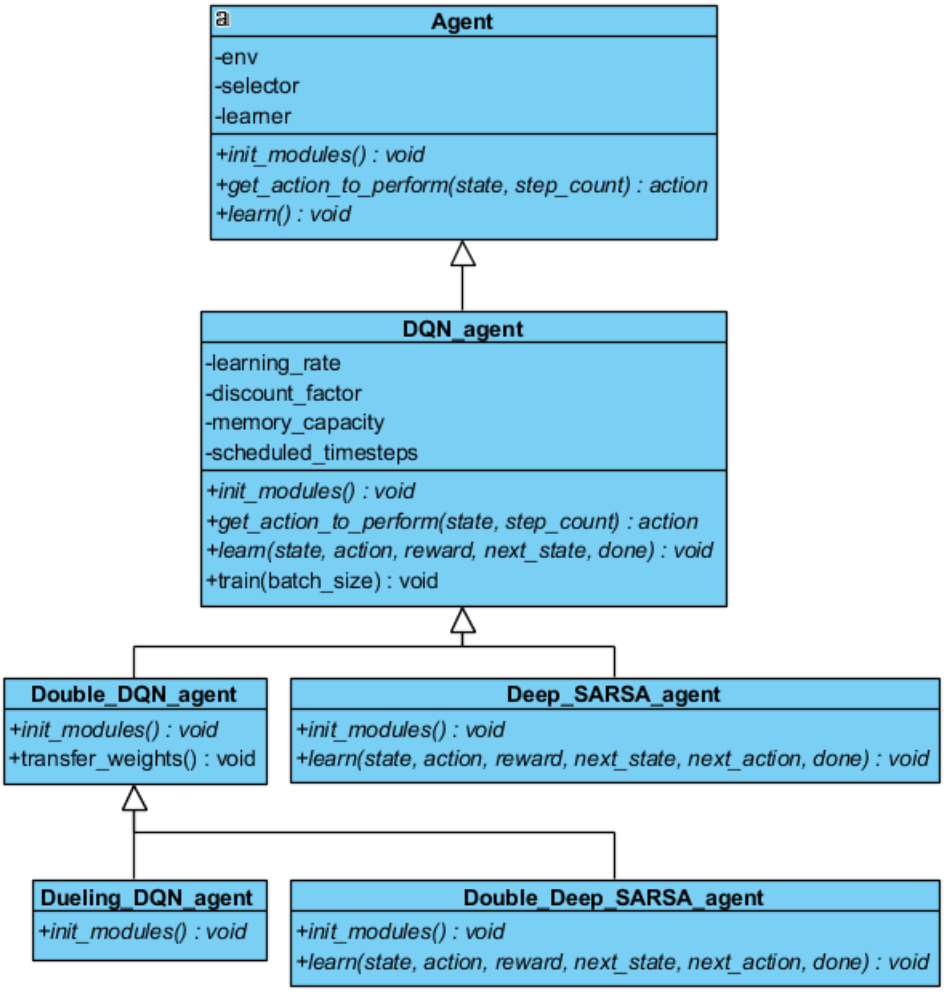
Obr. 4.3: Objektová hierarchia, Learner

V prípade rozhodovacieho modulu je požadovaná funkcionalita zahrnutá už v rozhodovacom module algoritmu Deep Q-learning. Keďže si tento modul drží model len v právomoci readOnly, nebol dôvod v potomkoch jeho metódy nijak prepisovať.



Obr. 4.4: Objektová hierarchia, ActionSelector

V triede Agent je objektová hierarchia podobná ako tomu bolo pri učiacom module. Základná funkcionalita je tiež zahrnutá v agentovi algoritmu Deep Q-learning, je ale potrebné prepísať metódu na inicializáciu modulov pre agenta každého algoritmu, aj keď ich funkcionalita je v niektorých prípadoch veľmi podobná, alebo zhodná.



Obr. 4.5: Objektová hierarchia, Agent

Objektový návrh ako aj objektová hierarchia prezentovaná v tejto kapitole je realizovaná v implementačnej časti práce pri implementácii jednotlivých algoritmov menovaných v kapitole 3.

5 Implementácia algoritmov

V rámci implementačnej časti bakalárskej práce sú implementované všetky algoritmy vymenované v kapitole 3, ako aj základný algoritmus Q-learning na prostredí CliffWalking, ktoré slúžilo na demonštráciu samotného algoritmu. Všetky implementované algoritmy dodržujú objektový návrh frameworku popísaného v predchádzajúcej kapitole.

5.1 Základné algoritmy

Implementácia algoritmu Q-learning je priamočiara, využívajú sa v nej všetky koncepty pomenované v samotnom popise algoritmu. Keďže ide o algoritmus, ktorého dĺžka učiaceho času v prostredí CliffWalking sa pohybuje rádovo v sekundách, po tomto procese je výsledný efekt aj simulovaný priechodom hry v konzole.

Pred samotnou implementáciou učiacich algoritmov v prostrediach Atari 2600 som implementoval algoritmy Deep Q-learning, Deep double Q-learning, Deep SARSA a Deep double SARSA na hrách typu Classic control z nástroja Gym. Neurónová sieť, ktorá bola pre tieto prostredia postačujúca, je podstatne jednoduchšia ako neurónové siete využívané v prostrediach Atari 2600. Algoritmy boli opakovane spúšťané na prostredí CartPole, v ktorom je podstatou udržať rovnováhu stojacej palice na posuvnom vozíku. Hra nie je nijako zložitá, preto už po niekoľkých epizódach je badateľné, že si algoritmus osvojil pravidlá hry a je schopný udržať rovnováhu na viac ako zlomok sekundy. Výsledky vykonaných meraní sú popísane v sekcii 6.2.

5.2 Algoritmy hlbokého učenia v hrách Atari 2600

5.2.1 Reprezentácia stavu

Zo sekcie 3.1.1 už vieme, že je nutné reprezentovať stav nie jedným, ale štyrmi hernými snímkami, aby v nich bola zachovaná informácia o pohybe herných objektov. Pôvodný rozmer jednej hernej plochy v hre Atari 2600 je $210 \times 160 \times 3$, čo v uvedenom poradí predstavuje výšku, šírku a farbu plochy. My tento rozmer upravíme na $84 \times 84 \times 1$, čo znamená, že obraz zmenšíme a prevedieme do odtieňov šedej. Tento

postup bol uskutočnený aj pri pozorovaniach vykonávaných inými subjektami, lebo zmierňuje pamäťové nároky a ďalšou výhodou je, že grafická karta, na ktorej učiaci proces poväčšine beží, efektívnejšie pracuje s objektami, ktoré majú štvorcový charakter.

K dosiahnutiu žiadaného rozmeru som využil princíp obaľovania, ktorý spočíva v tom, že vytvorené prostredie ešte pred tým ako ho použijeme, obalíme niekoľkými ďalšími triedami, ktoré postupne upravujú funkcionalitu pôvodných metód. Na konci procesu dostávame prostredie, na ktorom nám zavolanie metódy `step` vráti stav práve v žiadanom formáte $84 \times 84 \times 4$, čo znamená zväzok štyroch herných snímok. Celý proces sa nachádza v súbore `utils/atari_wrapper.py`.

5.2.2 Štruktúra neurónových sietí

Ďalším krokom bolo zvolenie vhodnej neurónovej siete pre učiace algoritmy. Neurónové siete použité v implementáciách zodpovedajú architektúram dostupným v publikáciách zaoberajúcich sa danou problematikou [10, 15]. Každá z neurónových sietí berie na vstupe stav v tvare $84 \times 84 \times 4$ a výstupom je 18 hodnôt, z ktorých každá reprezentuje jednu akciu, ktorú je možné vykonať na ovládači konzoly Atari 2600.

Tabuľka 5.1: Štruktúra základnej neurónovej siete

Číslo	Vrstva	Vstup	Výstup
1.	Conv2D	$84 \times 84 \times 4$	$20 \times 20 \times 32$
2.	Conv2D	$20 \times 20 \times 32$	$9 \times 9 \times 64$
3.	Conv2D	$9 \times 9 \times 64$	$7 \times 7 \times 64$
4.	Dense	$7 \times 7 \times 64$	512
5.	Dense	512	18

Vylepšenia algoritmu Deep Q-learning a algoritmu SARSA štruktúru neurónovej siete modifikovali len minimálne alebo vôbec. Z rozhrania, ktoré poskytuje trieda Model, je najdôležitejšou metódou metóda update. V nej prebieha samotný proces učenia.

5.2.3 Experience replay

V algoritmoch Deep Q-learning, Deep SARSA a ich variantoch figuruje taktiež princíp Experience replay. Ten som implementoval vytvorením triedy ReplayMemory, ktorej základným stavebným kameňom je dátová štruktúra deque. S triedou reprezentujúcou replay buffer ďalej operuje trieda Learner, ktorá volaním metódy `sample_batch` dávkuje učiacemu algoritmu náhodné vzorky, za pomoci ktorých sa algoritmus následne učí, ako bolo detailnejšie popísane v sekcii 3.1.2. Implementačné detaily daného princípu sú zahrnuté v súboroch `utils/replay_memory.py` a `utils/replay_memory_sarsa.py`.

5.2.4 Výber akcie

Implementácia samotného výberu akcie, konkrétne stratégie ϵ-greedy je priamočiara a pre všetky algoritmy hlbokého učenia rovnaká, ale postupné znižovanie hodnoty ϵ vyžadovalo sofistikovanejší prístup, ktorý je implementovaný za pomoci triedy EpsilonCounter, ktorej konštruktor berie na vstupe počet krokov, iniciálnu a konečnú hodnotu ϵ. Táto trieda lineárne interpoluje hodnoty medzi iniciálnou a konečnou hodnotou a aktuálnu hodnotu vracia pomocou metódy `value`, ktorá berie na vstupe číslo vykonávaného kroku. Implementácia popísaného postupu je zahrnutá v súbore `actionSelector.py`.

5.2.5 Spúšťanie algoritmov hlbokého učenia

V snahe zamedziť opakovaniu kódu som pre potreby spúšťania jednotlivých algoritmov vytvoril samostatný súbor dostupný v priečinku `alg_executor`. Na jeho začiatku je sekcia, kde je možné nastaviť algoritmus ktorého agent bude vytvorený, aká metóda tréningu sa bude vykonávať, na akej hre a samotné hyperparametre. Tento skript pri tréningu taktiež zabezpečuje pravidelné výpisy do súborov.

5.2.6 Využitý programovací jazyk Python

Nástroj Gym, ako aj celá práca je implementovaná v jazyku Python. To so sebou prináša niekoľko pozitív, ale aj negatív. Za pozitívum považujem dostupnosť veľkého množstva nástrojov, ktoré uľahčujú prácu s neurónovými sieťami (knižnica Keras) a poľami, ktorými je

reprezentovaná väčšina dát (knižnica Numpy). Taktiež veľké plus predstavuje pomerne jednoduché spúšťanie algoritmov na grafickej karte. Knižnica Tensorflow, s ktorom pracuje aj knižnica Keras, ponúka dve verzie - CPU a GPU verziu. Ak má stroj, na ktorom algoritmy spúšťame grafickú kartu Nvidia, stačí nainštalovať niekoľko programov a potom sa pri každom spustení budú algoritmy automaticky trénovať na grafickej karte[1].

Na druhej strane jazyk Python nie je vhodný na akýkoľvek zložitejší OOP návrh. Absenciu abstraktných atribútov a metód je nutné riešiť použitím externých nástrojov a knižníc. Knižnica abc pridáva možnosť vytvoriť abstraktnú metódu a skript `utils/better_abc.py`, ktorý rozširuje knižnicu abc, naviac pridáva aj možnosť vytvorenia abstraktného atribútu. V niektorých prípadoch bola problémová aj absencia typov, ktorá zhoršovala čitateľnosť kódu. Bolo nutné obzvlášť dbať na pomenovanie premenných, aby ich názov reflektoval aj typ ich hodnoty.

1. `https://www.tensorflow.org/install/gpu`

6 Merania

V rámci práce som merania vykonával na prostrediach CartPole (Classic Control), Pong a SpaceInvaders (Atari 2600). Vo všetkých prípadoch bolo nutné spracovať textové výpisy jednotlivých učiacich algoritmov. Textové súbory som spracoval za pomoci regulárnych výrazov, ktorými som zo stringov extrahoval číselné hodnoty. S využitím knižnice xlsxwriter som z týchto číselných hodnôt následne vytvoril tabuľky vo formáte .xlsx, ktoré som ďalej spracoval v programe Microsoft Excel.

V prípade prostredia CartPole, boli algoritmy Deep Q-learning, Deep double Q-learning, Deep SARSA a Deep double SARSA spúšťané každý 10 krát. Algoritmus s architektúrou dueling deep Q-network na prostredí CartPole nebol spúšťaný, keďže toto vylepšenie je určené práve pre hry Atari 2600.

V hrách Atari 2600 bol každý algoritmus spúšťaný raz, keďže beh jednotlivých algoritmov bol podstatne časovo náročnejší ako pri prostredí CartPole, ale aj napriek tomu boli zaznamenané zaujímavé výsledky.

Merania boli uskutočnené na hernom notebooku s využitím dedikovanej grafickej karty.

Tabuľka 6.1: Parametre herného notebooku

Procesor	Intel Core i7 6700HQ @ 2,60 GHz Boost 3,50 GHz 4C, 8T
Grafická karta	Nvidia GeForce GTX 970m, 3GB, 1280 CUDA cores
RAM	16GB Dual Channel DDR4, 2133 MHz
Operačný systém	Windows 10 Home

6.1 Hyperparametre a ich vplyv

V rámci algoritmov hlbokého učenia pribudlo k parametrom α (Learning rate) a γ (Discount factor) ešte zopár ďalších. V publikáciách, ktoré popisujú jednotlivé algoritmy, je parameter α v intervale $(0, 0.1>$

a parameter γ v intervale $<0.9, 1)$. Rozhodol som sa preto s týmito parametrami neexperimentovať. Na druhej strane som ale zisťoval vplyv parametrov pridaných exkluzívne pre hlboké učenie: veľkosť replay pamäte, veľkosť vzorky a frekvencia aktualizácie Target-network a minimálna naplnenosť replay pamäte pred začiatkom tréningu.

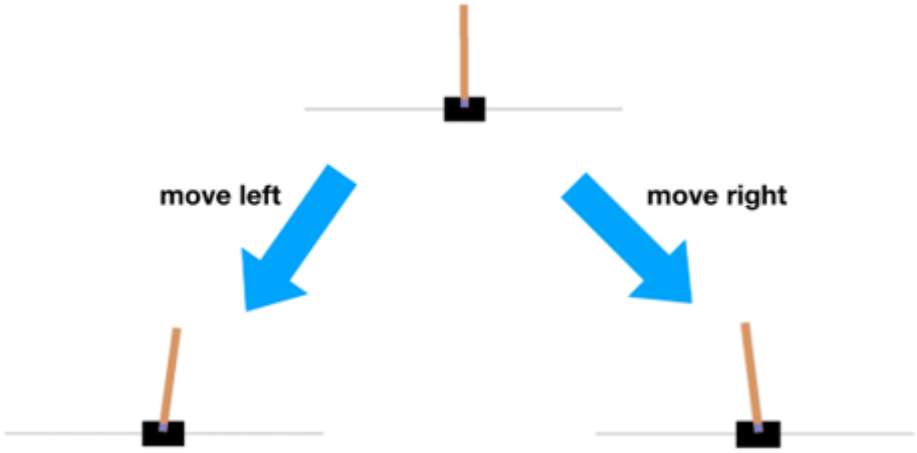
Veľkosť pamäte ovplyvňovala predovšetkým pamäťové nároky jednotlivých algoritmov, čo bolo možné sledovať pri porovaní behu algoritmov v prostrediach Pong a SpaceInvaders, kde som použil rozdielnu veľkosť pamäte. Väčšia pamäť znižovala taktiež pravdepodobnosť, že algoritmus postupom času „zabudne“ optimálne reagovať na staršie stavy. Pri voľbe veľkosti pamäte je preto potrebné dbať na zložitosť prostredia, ale aj na veľkosť dostupnej pamäte na stroji, na ktorom je pozorovanie vykonávané.

Veľkosť vzorky priamo úmerne súvisí s tým, ako dlho bude algoritmus bežať, čo som pozoroval, keď som algoritmy spúštal na rôznych prostrediach s rôznymi hodnotami. Čím väčšiu vzorku používame, tým je proces aktualizácie neurónovej siete časovo náročnejší, môže ale v zložitejších prostrediach dopomôcť k dosiahnutiu lepších výsledkov.

Frekvencia aktualizácie Target-network je hyperparameter využívaný v algoritmoch s prívlastkom double a dueling, ktorý určuje, ako často je aktualizovaná druhá neurónová sieť. Ak by hodnota tohto parametru bola príliš nízka, druhá neurónová sieť by sa stala redundantnou. Naopak, ak by bola hodnota príliš vysoká, ohodnocovali by sme akcie na základe príliš starých dát. Dopad tohto parametra bol spozorovaný pri spuštaní algoritmu Deep double Q-learning v prostredí CartPole.

Parameter určujúci minimálnu naplnenosť replay pamäte pred začiatkom tréningu dopomáha k tomu, aby pred prvou aktualizáciou neurónovej siete, bol v pamäti už určitý počet stavov a tým napomáhame k rôznorodosti vybranej vzorky. Ak by sme algoritmus začali trénovať akonáhle naplnenosť pamäte dosiahne hodnotu veľkosti vzorky, tieto stavy by nasledovali hneď za sebou. Majú vysokú koreláciu a tomu chceme práve určením minimálnej naplnenosti pamäte zamedziť [10].

6.2 Prostredie CartPole



Obr. 6.1: CartPole [20]

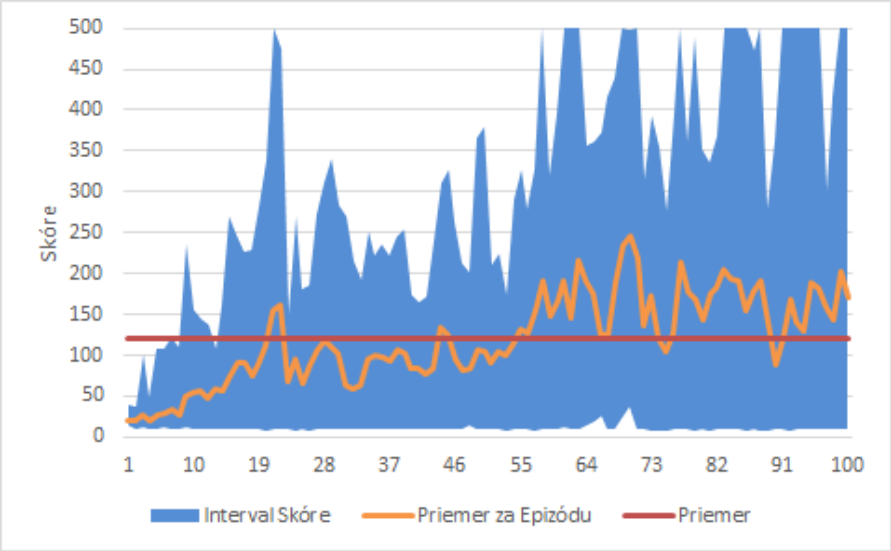
Prostredie CartPole vďaka svojej jednoduchosti a pomerne krátkej dobre tréningu, umožňuje spúšťanie algoritmov opakovane, čím zvyšujeme relevantnosť nameraných výsledkov. Konkrétne v tomto pozorovaní boli algoritmy hlbokého učenia spúšťané desať krát. Z nameraných dát sme následne určili interval skóre, ktoré daný algoritmus dosiahol, vypočítali sme priemernú hodnotu skóre v každej epizóde a priemerné dosiahnuté skóre naprieč všetkými epizódami.

Meranie je špecifické tým, že pri nastavení znižovania hodnoty ϵ (pravdepodobnosť náhodnej akcie) v prvej tretine tréningu na hodnotu 0.1, ako tomu bolo v ostatných meraniach, algoritmy neboli schopné sa učiť. Preto v prostredí CartPole bolo nutné zvoliť agresívnejší prístup znižovania hodnoty ϵ, ako aj zvoliť nižšiu finálnu hodnotu.

Tabuľka 6.2: Zoznam hyperparametrov, CartPole

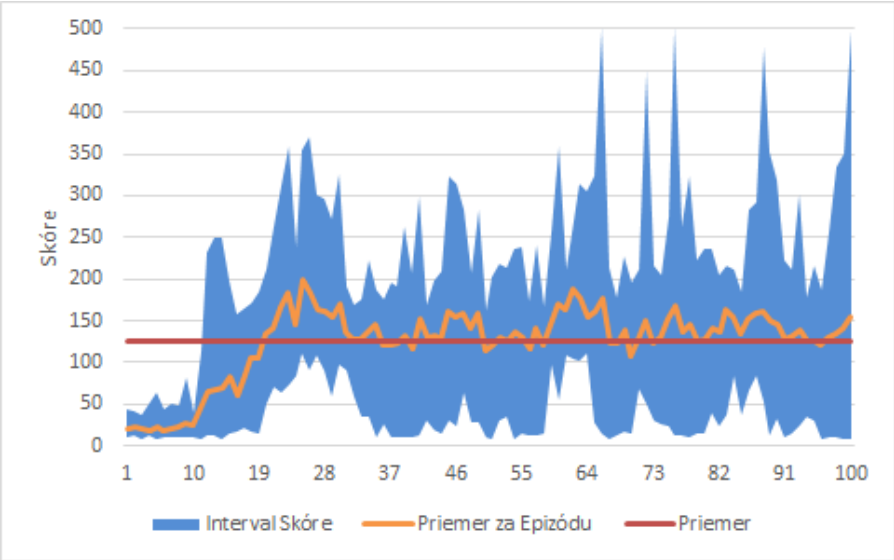
Hyperparameter	Hodnota
Počet opakovaní	10
Learning rate	0,001
Discount factor	0,95
Počet epizód	100
Finálna hodnota ϵ	0,01
Počet akcií za ktorý dosiahne ϵ finálnu hodnotu	900
Frekvencia aktualizácia Target-network	každých 5 epizód
Veľkosť replay pamäte	2000
Veľkosť vzorky	32

Deep Q-learning



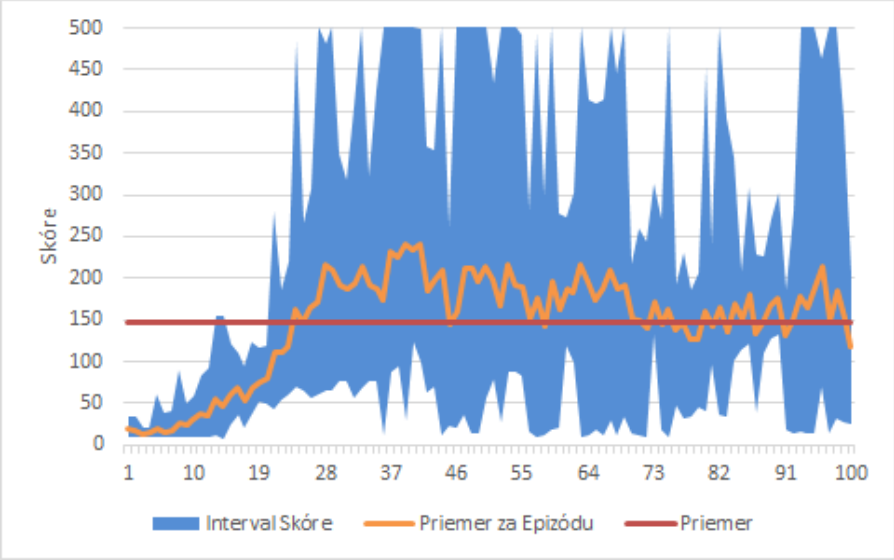
Obr. 6.2: Graf: Deep Q-learning, CartPole

Deep double Q-learning



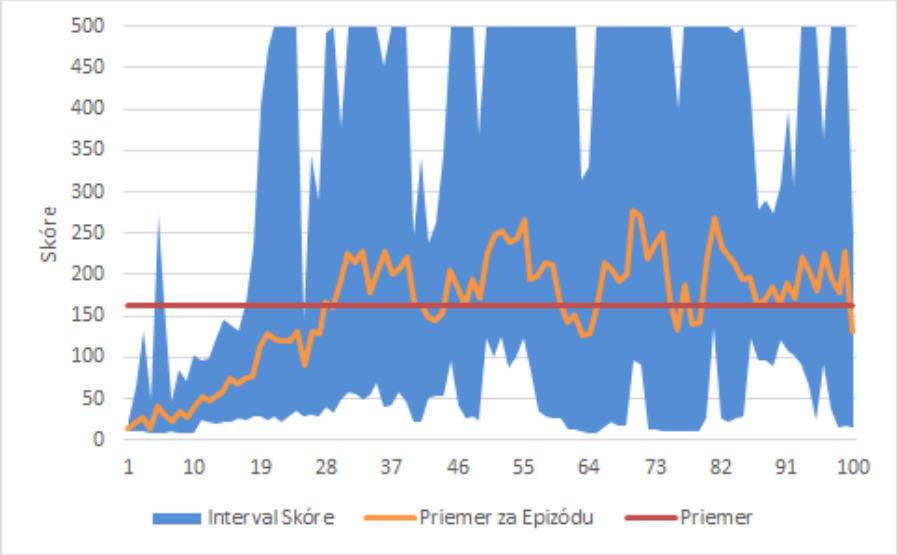
Obr. 6.3: Graf: Deep double Q-learning, CartPole

Deep SARSA



Obr. 6.4: Graf: Deep SARSA, CartPole

Deep double SARSA



Obr. 6.5: Graf: Deep double SARSA, CartPole

Tabuľka 6.3: Porovnanie algoritmov, CartPole

	Priemenrné skóre	Priemenrná dĺžka tréningu (MM:SS)
Deep Q-learning	120,295	28:57
Deep double Q-learning	125,032	32:52
Deep SARSA	147,9	37:06
Deep double SARSA	161,686	44:56

Merania uskutočnené na prostredí CartPole preukázali, že vylepšenia jednotlivých algoritmov predčia ich základné verzie. So zvyšujúcim skóre sa prirodzene zvyšuje aj dĺžka tréningu, keďže cieľom hry je udržať rovnováhu čo najdlhšie. Konkrétne v prostredí CartPole epizóda končí akonáhle agent dosiahne skóre 500. Zaujímavý je taktiež rozdiel v skóre pri porovnaní algoritmov vychádzajúcich z algoritmov Q-learning a SARSA.

6.3 Prostredie Pong

Obr. 6.6: Pong [17]

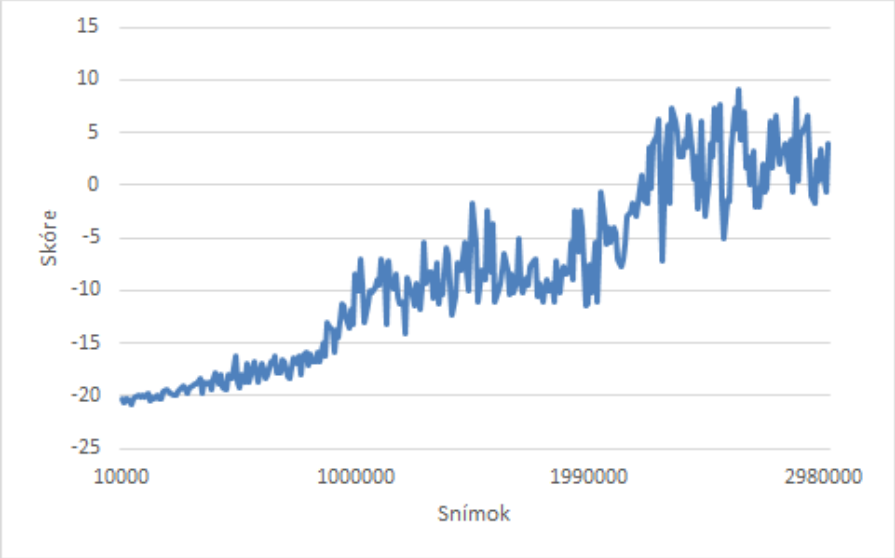
Hra Pong je pomerne jednoduchá na ovládanie. Hráč ovláda odrazovú plochu a platné pohyby sú iba pohyb nahor a pohyb nadol. Úlohou hráča je získať viac bodov ako súper tým, že pomyselnú loptu odrazí takým spôsobom, že súper nebude schopný reagovať a lopta mu prepadne.

V prostredí Pong je každé vyhraté kolo ohodnotené +1 a každé prehraté kolo naopak -1. Každá hra (epizóda) trvá 21 kôl. Rozhodol som sa preto na určenie dĺžky tréningu využiť namiesto počtu epizód, počet herných snímok.

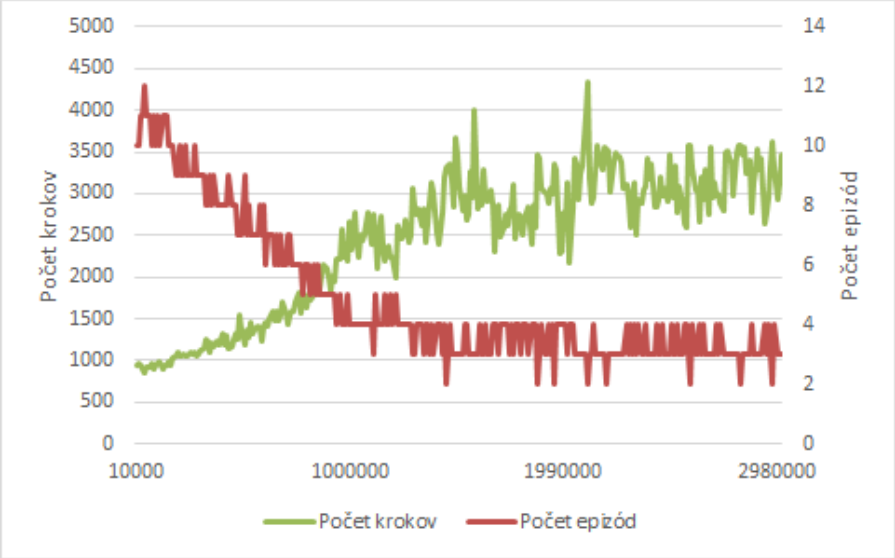
Tabuľka 6.4: Zoznam hyperparametrov, Pong

Hyperparameter	Hodnota
Learning rate	0,00001
Discount factor	0,99
Počet snímok	3 000 000
Finálna hodnota ϵ	0,1
Počet snímok za ktorý dosiahne ϵ finálnu hodnotu	1 000 000
Frekvencia aktualizácia Target-network	každých 10 000 snímok
Veľkosť replay pamäte	20 000
Veľkosť vzorky	32
Minimálna naplnenosť pamäte pred začiatkom tréningu	5000

Deep Q-learning

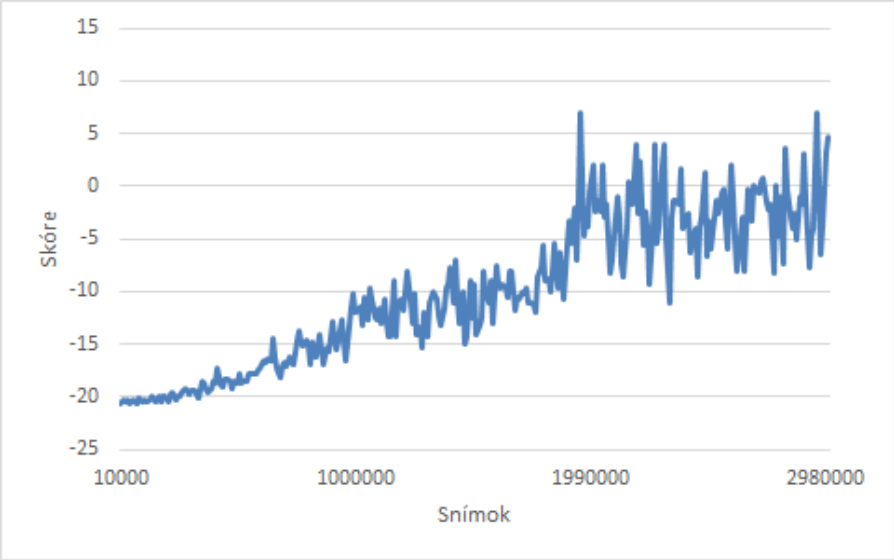


Obr. 6.7: Graf: Deep Q-learning, Pong skóre

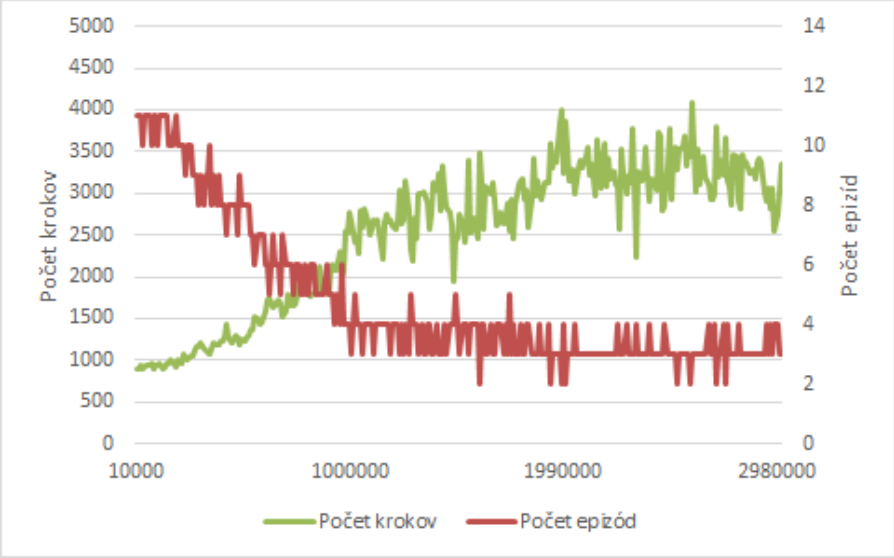


Obr. 6.8: Graf: Deep Q-learning, Pong epizódy

Deep double Q-learning

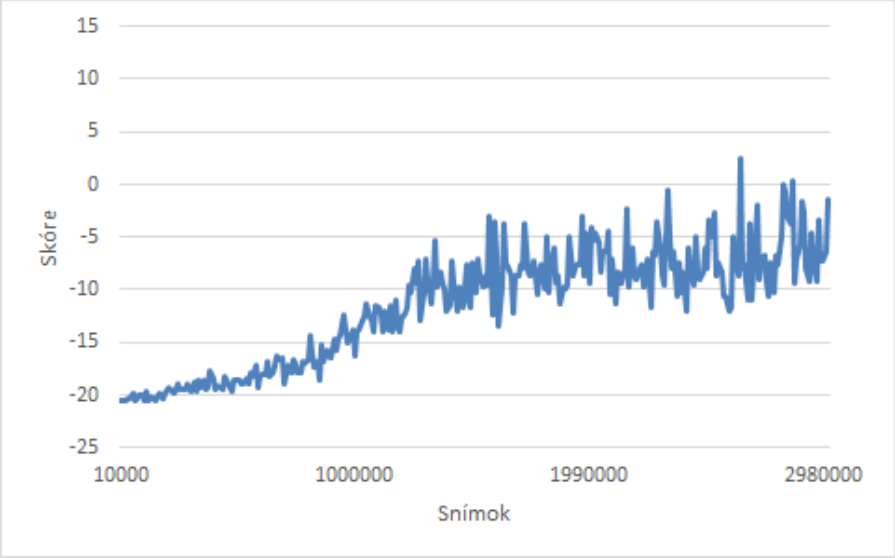


Obr. 6.9: Graf: Deep double Q-learning, Pong skóre

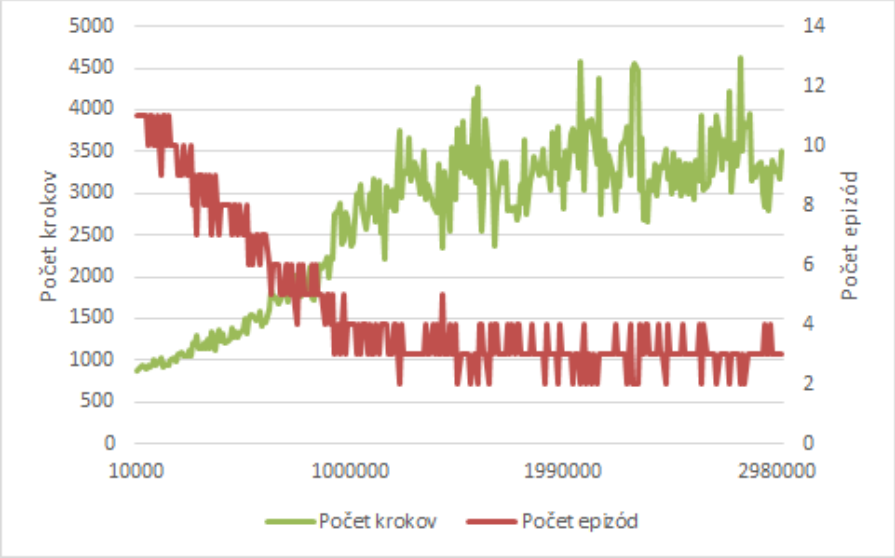


Obr. 6.10: Graf: Deep double Q-learning, Pong epizódy

Architektúra dueling deep Q-network

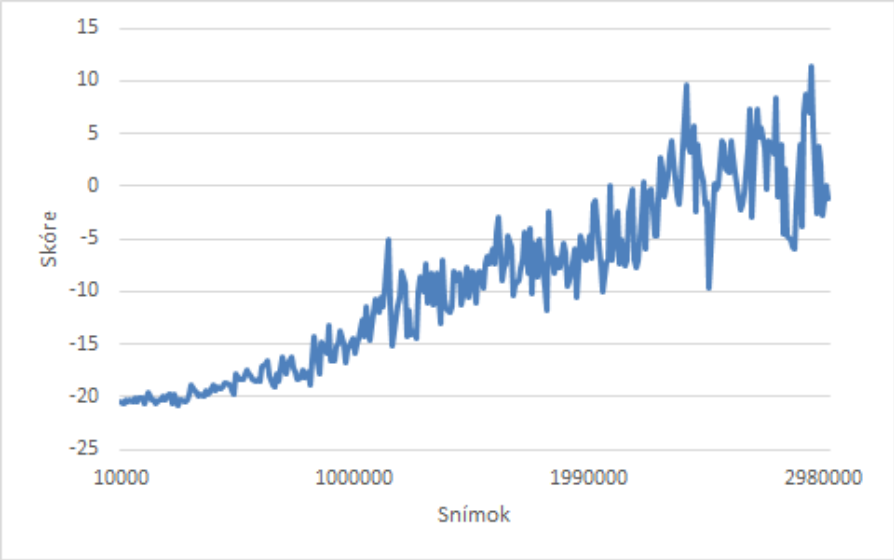


Obr. 6.11: Graf: Architektúra dueling deep Q-network, Pong skóre



Obr. 6.12: Graf: Architektúra dueling deep Q-network, Pong epizódy

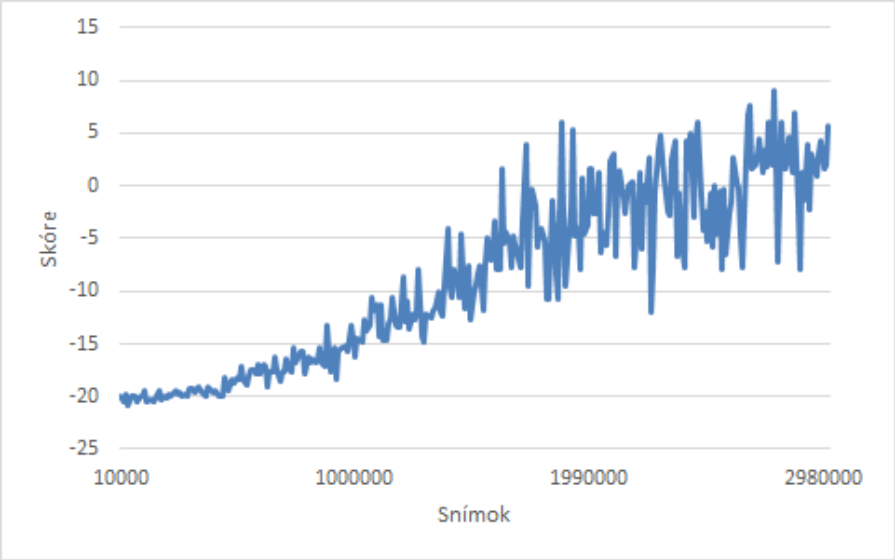
Deep SARSA



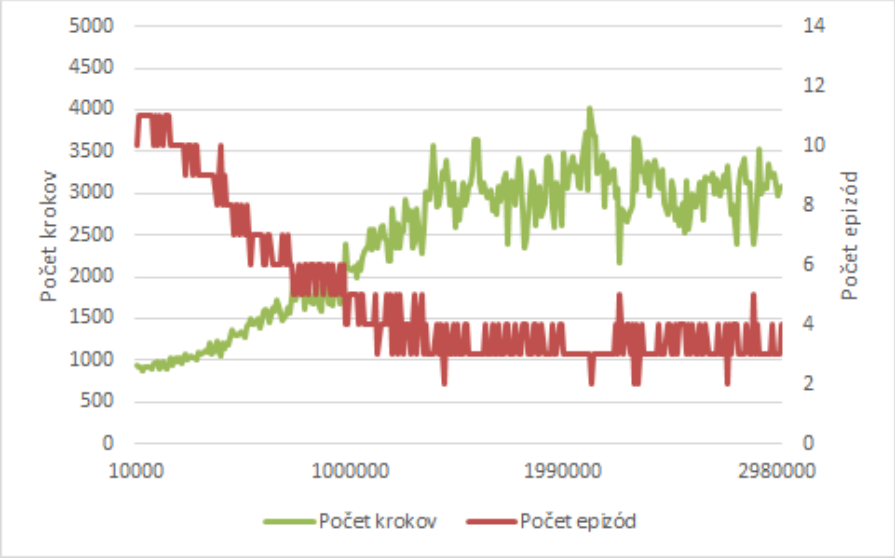
Obr. 6.13: Graf: Deep SARSA, Pong skóre

Obr. 6.14: Graf: Deep SARSA, Pong epizódy

Deep double SARSA



Obr. 6.15: Graf: Deep double SARSA, Pong skóre



Obr. 6.16: Graf: Deep double SARSA, Pong epizódy

Porovnanie algoritmov

Tabuľka 6.5: Porovnanie algoritmov, Pong

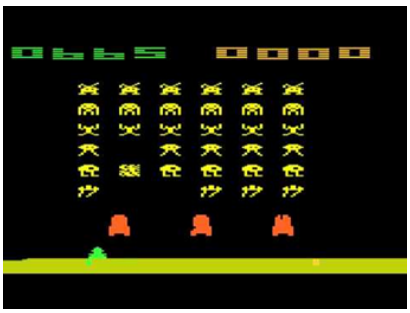
	Max skóre	Priem. skóre	Max dĺžka epizódy	Priem. dĺžka epizódy	Dĺžka tréningu H:M:S
DQN	9	-8,45732	4330	2449,293	28:03:02
DDQN	7	-10,0756	4086	2507,144	31:37:26
DDDQN	2,5	-11,5497	4628	2680,654	32:29:04
DSARSA	11,333	-9,19682	4185,5	2442,333	28:26:10
DDSARSA	9	-8,86843	4013,333	2432,85	28:26:58

Keďže algoritmy boli spúšťané iba po dobu 3M herných snímok, nie je možné tieto výsledky priamo porovnať s inými pozorovaniami, ktoré boli vykonané po dobu 10M [12], 50M [10] a 200M [15] herných snímok, na oveľa výkonnejších strojoch. Aj napriek pomerne krátkej dobre tréningu v porovnaní s inými pozorovaniami, je zrejmé, že sa agent každého algoritmu učil. V hre Pong kladné skóre na konci epizódy naznačuje, že hráč, v tomto prípade algoritmus, kolo vyhral. Algoritmom sa podarilo dosiahnuť kladné skóre už v zhruba dvoch tretinách tréningu a postupom času skóre stále narastalo. Môžeme teda predpokladať, že ak by tréning trval dlhšie, dosiahli by sme priaznivejšie aj priemerné skóre.

S dosiahnutým skóre súvisí aj dĺžka jednotlivých hier. So zvyšujúcim skóre sa zvyšovala aj dĺžka a znižoval počet vykonaných hier v meranom intervale. Zaujímavé sú obzvlášť výsledky algoritmu Dueling deep Q-learning, ktorý dosiahol najvyššiu priemernú dĺžku epizódy, ale najnižšie skóre. Algoritmus teda nevedel „zasadiť víťazný úder“.

Dĺžka tréningu jednotlivých algoritmov sa pohybovala rádovo okolo tridsiatich hodín, čo je alarmujúce číslo. Ak by som chcel na svojom počítači zopakovať pozorovanie na 200M herných snímkach, trvalo by to zhruba 1980 hodín, čo je 82 a pol dňa len na vykonanie pozorovania jedného algoritmu. Je preto žiadúce, vykonávať pokusy väčších rozmerov na adekvátne výkonných strojoch.

6.4 Prostredie SpaceInvaders



Obr. 6.17: SpaceInvaders [21]

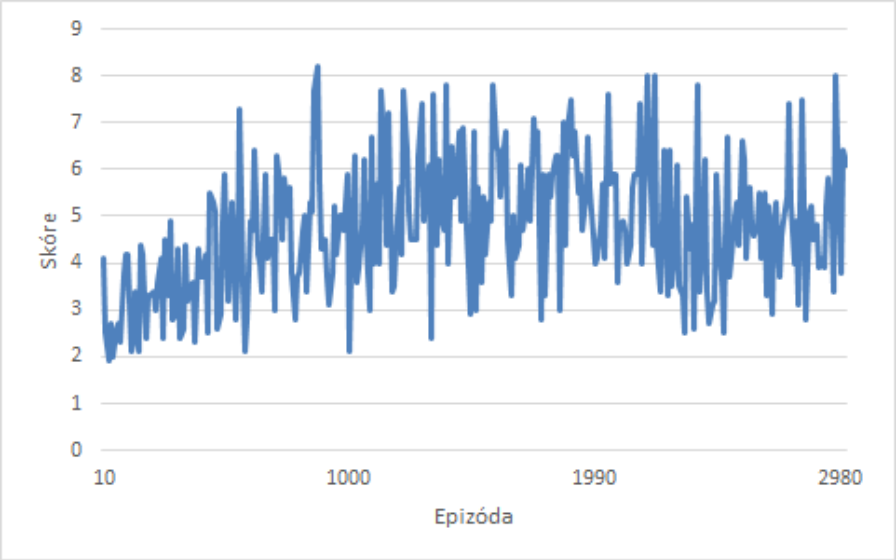
V hre SpaceInvaders sa k jednoduchému pohybu pridáva aj streľba. Dané prostredie som vybral preto, že v publikácii popisujúcej Dueling deep Q-learning bolo dosiahnuté úctyhodné zlepšenie oproti klasickej verzii Deep Q-learning [16]. V mnou vykonanom pozorovaní som v tomto prostredí zvolil prístup určenia dĺžky tréningu pomocou počtu epizód. Keďže hráč má len tri životy, dĺžka jednej epizódy bola podstatne kratšia, ako tomu bolo v prostredí Pong.

Tabuľka 6.6: Zoznam hyperparametrov, SpaceInvaders

Hyperparameter	Hodnota
Learning rate	0,001
Discount factor	0,95
Počet epizód	3000
Finálna hodnota ϵ	0,1
Počet epizód za ktorý dosiahne ϵ finálnu hodnotu	1000
Frekvencia aktualizácia Target-network	každých 10 epizód
Veľkosť replay pamäte	40 000
Veľkosť vzorky	64
Minimálna naplnenosť pamäte pred začiatkom tréningu	2500

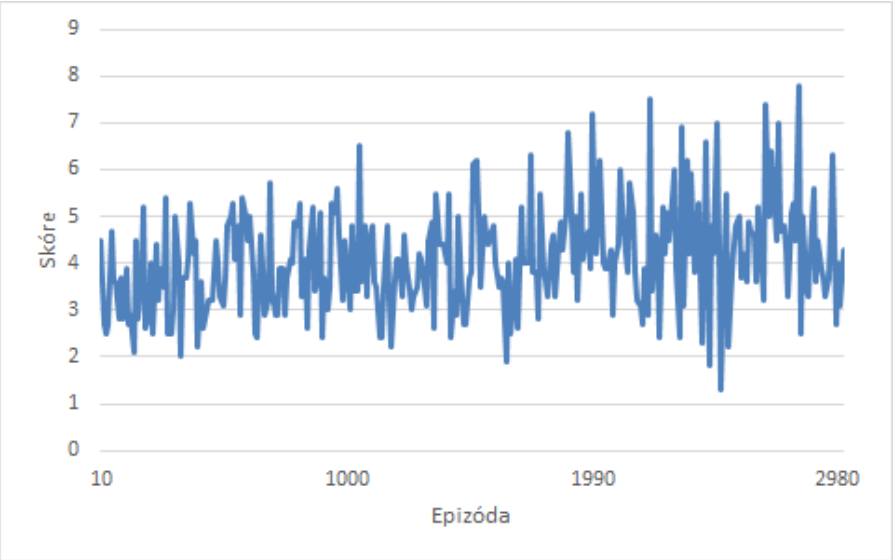
Žiaľ počet epizód bol nedostatočný aj napriek zvýšenej veľkosti replay pamäte a veľkosti vzorky. Nie je badateľné zlepšovanie (učenie) sa jednotlivých algoritmov v danom prostredí. Ako príklad uvádzam výsledky algoritmov Deep Q-learning a Dueling deep Q-learning, kde mal byť rozdiel v dosiahnutom skóre jasne viditeľný v prospech algoritmu Dueling deep Q-learning [16]. Je vhodné poznamenať, že hodnoty odmien nereflektujú reálne zmeny skóre v hre, ale každý zásah nepriateľa je ohodnotený hodnotou +1, aby bolo pripadne možné na jednotlivých hrách algoritmy ľahšie porovnať.

Deep Q-learning



Obr. 6.18: Graf: Deep Q-learning, SpaceInvaders skóre

Architektúra dueling deep Q-learing



Obr. 6.19: Graf: Architektúra dueling deep Q-network, SpaceInvaders skóre

Tabuľka 6.7: Dĺžky meraní algoritmov, SpaceInvaders

Algoritmus	Dĺžka merania (H:M:S)
Deep Q-learning	10:32:42
Deep double Q-learning	11:37:41
Architektúra dueling deep Q-network	11:17:17
Deep SARSA	9:41:25
Deep double SARSA	9:46:12

Ako už bolo konštatované, dĺžka tréningu jednotlivých algoritmov nebola postačujúca na to, aby bolo viditeľné, že sa algoritmus hru naučil hrať. Pri určovaní dĺžky tréningu počtom epizód v hrách Atari 2600 nevieme dopredu predpokladať, aký dlhý tréning bude. Naviac môžeme vidieť, že sa algoritmy SARSA vykonávali kratšie aj napriek zložitejšiemu výpočtu v porovnaní s algoritmom Deep Q-learning.

Bolo to zapríčinené náhodnými sekvenciami krátkych epizód, čo je nežiadúce správanie, ktoré je možné eliminovať práve určením dĺžky tréningu pomocou počtu herných snímkov.

7 Záver

Práca sa zaoberá zložitým a veľmi rýchlo sa rozvíjajúcim odvetvím informatiky. V oblasti hlbokého učenia v počítačových hrách osobitne platí, že to, čo je dnes novinkou, už zajtra bude zastarané. Našim zámerom teda nebolo vymyslieť niečo prevratné, skôr sme sa zamerali na návrh nástroja, ktorý by implementácie jednotlivých algoritmov uľahčil.

Všetky algoritmy popísané v rámci práce boli implementované práve podľa navrhnutého frameworku. Naviac boli algoritmy spúšťané na rozličných prostrediach. Merania sa rozsahovo nepribližovali meraniam popísaným vo vedeckých publikáciách, ale aj napriek tomu sme v niektorých meraniach dosiahli zaujímavé výsledky. Je nutné poznamenať, že rozsah meraní je úzko spätý s dostupným technickým vybavením.

Ak by sa nám do budúcna podarilo eliminovať problém nízkej výkonnosti technického vybavenia a priblížili by sme sa výkonnostne o niečo viac k strojom, na ktorých boli vykonávané merania popísané vo vedeckých publikáciách, bolo by možné merania s nadobudnutými vedomosťami zopakovať vo väčšom rozsahu, čo by určite viedlo k ešte zaujímavejším výsledkom.

Bibliografia

1. KURAMA, V. *Introduction to machine learning* [online]. Medium - Towards Data Science, 2017 [cit. 2019-05-20]. Dostupné z: https://towardsdatascience.com/introduction-to-machine-learning-db7c668822c4.
2. KUMAR, CH. *Artificial Intelligence: Definition, Types, Examples, Technologies* [online]. Medium, 2015 [cit. 2018-06-11]. Dostupné z: https://medium.com/@chethankumargn/artificial-intelligence-definition-typesexamples-technologies-962ea75c7b9b.
3. YANNAKAKIS N, G. *Game AI revisited* [online]. IT University of Copenhagen, Copenhagen, Denmark: Center for Computer Games Research, 2012 [cit. 2019-05-20]. Dostupné z: https://web.archive.org/web/20140808052149/http://aida.ii.uam.es/teaching/videojuegos/wp-content/uploads/course_files_12_13/gameAI.pdf.
4. JUSTESEN, N.; BONTRAGER, P.; TOGELIUS, J.; RISI, S. *Deep Learning for Video Game Playing* [online]. IT University of Copenhagen, Copenhagen, New York University, New York, 2019 [cit. 2019-12-07]. Dostupné z: https://arxiv.org/pdf/1708.07902.pdf.
5. STANLEY, K. *Neuroevolution: A different kind of deep learning* [online]. O'Reilly, 2017 [cit. 2019-12-07]. Dostupné z: https://www.oreilly.com/radar/neuroevolution-a-different-kind-of-deep-learning/.
6. VIOLANTE, A. *Simple Reinforcement Learning: Q-learning* [online]. Medium - Towards Data Science, 2019 [cit. 2019-05-20]. Dostupné z: https://towardsdatascience.com/simple-reinforcement-learning-q-learning-fcddc4b6fe56.
7. SATWIK, K.; BRENDAN, M. *Reinforcement Q-Learning from Scratch in Python with OpenAI Gym* [online]. LEARNDATASCI [cit. 2019-05-20]. Dostupné z: https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/.

8. KUMAR, V. *Reinforcement learning: Temporal-Difference, SARSA, Q-Learning and Expected SARSA in python* [online]. Medium - Towards Data Science, 2019 [cit. 2019-12-07]. Dostupné z: https://towardsdatascience.com/reinforcement-learning-temporal-difference-sarsa-q-learning-expected-sarsa-on-python-9fecfda7467e.

9. ADL. *A brief introduction to reinforcement learning* [online]. Medium - freeCodeChamp, 2018 [cit. 2019-05-20]. Dostupné z: https://medium.freecodecamp.org/a-brief-introduction-to-reinforcement-learning-7799af5840db.

10. MNIH, V. et al. *Human-level control through deep reinforcement learning* [online]. London, Macmillan Journals ltd: Nature Publishing Group [cit. 2019-05-20]. Dostupné z: https://web.stanford.edu/class/psych209/Readings/MnihEtAlHassibis15NatureControlDeepRL.pdf.

11. SCHMIDHUBER, J. *Deep learning in neural networks: An overview* [online]. University of Lugano a SUPSI, Switzerland: Elsevier, 2015 [cit. 2019-12-07]. Dostupné z: https://www2.econ.iastate.edu/tesfatsi/DeepLearningInNeuralNetworksOverview.JSchmidhuber2015.pdf.

12. MNIH, W.; KAVUKCUOGLU, K.; SILVER, D.; GRAVES, A.; ANTONOGLOU, I.; WIERSTRA, D.; RIEDMILLER, D. *Playing Atari with Deep Reinforcement Learning* [online]. DeepMind Technologies, 2013 [cit. 2019-05-20]. Dostupné z: https://arxiv.org/pdf/1312.5602.pdf.

13. MATIISEN, T. *Demystifying Deep Reinforcement Learning* [online]. IntelAI [cit. 2019-05-20]. Dostupné z: https://www.intel.ai/demystifying-deep-reinforcement-learning/#gs.blxi10.

14. SIMONINI, T. *Improvements in Deep Q Learning: Dueling Double DQN, Prioritized Experience Replay, and fixed Q-targets* [online]. Medium - freeCodeChamp, 2018 [cit. 2019-05-20]. Dostupné z: https://medium.freecodecamp.org/improvements-in-deep-q-learning-dueling-double-dqn-prioritized-experience-replay-and-fixed-58b130cc5682.

15. HASSLET, H. van; GUEZ, A.; SILVER, D. *Deep Reinforcement Learning with Double Q-learning* [online]. Google DeepMind, 2015 [cit. 2019-05-20]. Dostupné z: https://arxiv.org/pdf/1509.06461.pdf.

16. WANG, Z.; SCHAUL, T.; HESSEL, M.; HASSLET, H. van; LANCTOT, M.; FREITAS, N. de. *Dueling Network Architectures for Deep Reinforcement Learning* [online]. London, UK: Google DeepMind, 2016 [cit. 2019-05-20]. Dostupné z: https://arxiv.org/pdf/1511.06581.pdf.

17. KARPATHY, A. *Deep Reinforcement Learning: Pong from Pixels* [online]. 2016 [cit. 2019-05-20]. Dostupné z: http://karpathy.github.io/2016/05/31/rl/.

18. ZHAO, D.; WANG, H.; SHAO, K.; ZHU, Y. *Deep Reinforcement Learning with Experience Replay Based on SARSA* [online]. Institute of Automation Chinese Academy of Sciences, Beijing 100190, China, 2016 [cit. 2019-12-07]. Dostupné z: https://ieeexplore-ieee-org.ezproxy.muni.cz/stamp/stamp.jsp?tp=&arnumber=7849837&tag=1.

19. GANGER, M.; DURYEA, E.; HU, W. *Double Sarsa and Double Expected Sarsa with Shallow and Deep Learning* [online]. Computer Science Department, Houghton College, Houghton, NY, USA: Journal of Data Analysis a Information Processing, 2016 [cit. 2019-12-07]. Dostupné z: https://file.scirp.org/pdf/JDAIP_2016101714072270.pdf.

20. PHY, V. *Reinforcement Learning Concept on Cart-Pole with DQN* [online]. Medium - Towards Data Science, 2019 [cit. 2019-12-07]. Dostupné z: https://towardsdatascience.com/reinforcement-learning-concept-on-cart-pole-with-dqn-799105ca670.

21. AQUALUNGGAMEREVIEWS. *Space Invaders Atari 2600 Review* [online]. Youtube, 2015 [cit. 2019-12-07]. Dostupné z: https://www.youtube.com/watch?v=AiH_3iUAzFU.