VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

POSILŇOVANÉ UČENIE PRE HRU BOMBERMAN

REINFORCEMENT LEARNING FOR BOMBERMAN TYPE GAME

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **JAKUB ADAMČIAK**
AUTHOR

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

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

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

Zadání bakalářské práce

Student: **Adamčiak Jakub**
Program: Informační technologie
Název: **Posilované učení pro hru typu Bomberman**
Reinforcement Learning for Bomberman Type Game
Kategorie: Umělá inteligence

Zadání:

1. Prostudujte základy neuronových sítí a posilovaného učení.
2. Vytvořte si přehled o současných metodách využívajících neuronové sítě a posilované učení pro řešení kompetitivních her podobných hře Bomberman.
3. Vyberte konkrétní metodu posilovaného učení aplikovatelnou na hry typu Bomberman.
4. Seznamte se s vhodným simulačním prostředím.
5. Implementujte navrženou metodu a proveďte experimenty ve zvoleném simulačním prostředí.
6. Porovnejte dosažené výsledky a diskutujte možnosti budoucího vývoje.
7. Vytvořte stručné video prezentující vaši práci, její cíle a výsledky.

Literatura:

- Ian Goodfellow and Yoshua Bengio and Aaron Courville, Deep Learning, MIT Press, 2016.
- Sergey Levine et al., CS 285 - Deep Reinforcement Learning, UC Berkeley, 2020.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 4.

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

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

Abstrakt

Cieľom tejto bakalárksej práce je návrh, implementácia a trénovanie modelov posilňovaného učenia na hru typu Bomberman. Je postavená na prostredí Bomberland od firmy CoderOne. Toto prostredie bolo vyvinuté za účelom vzdelávania a výskumu v odvetí umelej inteligencie. V tejto práci rozoberám rôzne nastavenia a problémy s implementovaním agenta do prostredia. Vyskúšal som 2 politiky (MLP a CNN), 2 algoritmy (PPO a A2C) a 5 druhov neurónových sietí pre extrakciu vlastností za pomoci knižníc stable baselines 3 a pytorch. Celkový čas trénovania týchto modelov bol dokopy 1207 reálnych hodín, 4168 strojových hodín a 271 miliónov herných krokov. Aj keď bolo trénovanie neúspešné, táto práca ukazuje proces implementácie modelu posilňovaného učenia do prostredia Gym.

Abstract

This bachelor's thesis aims to develop, implement and train reinforcement learning models for a Bomberman-type game. It is based on Bomberland environment from CoderOne. This environment was created for education and research in the field of artificial intelligence. In this thesis I tackle the settings and problems of implementing agent into the environment. I used 2 policies (MLP and CNN), 2 algorithms (PPO and A2C) and 5 setups of neural networks for feature extraction with the use of libraries stable baselines 3 and pytorch. Total training time resulted in 1207 real-world hours, 4168 computing hours and 271 milions of time steps. Although the training was not successful, this thesis shows the process of implementing a reinforcement learning model into a Gym environment.

Kľúčové slová

umelá inteligencia, AI, strojové učenie, ML, posilované učenie, RL, konvolučné neurónové siete, CNN, PPO, A2C, python, stable baselines3, ai-gym, pytorch, hry, bomberman

Keywords

artificial inteligence, AI, machine learning, ML, reinforcement learning, RL, convolutional neural networks, CNN, PPO, A2C, python, stable baselines3, ai-gym, pytorch, games, bomberman

Citácia

ADAMČIAK, Jakub. *Posilňované učenie pre hru Bomberman.* Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Michal Hradiš, Ph.D.

Posilňované učenie pre hru Bomberman

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením doktora Michala Hradiša. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Jakub Adamčiak
11. mája 2022

Poďakovanie

Chcel by som poďakovať Ing. Michalovi Hradišovi, Ph.D. za odborné konzultácie, rady a pomoc pri vypracovávaní tejto bakalárskej práce. Taktiež by som chcel poďakovať ľuďom, ktorí mi pomohli úpravou štylistiky a gramatiky tejto technickej práce a rodine za prejavenú podporu pri štúdiu.

Obsah

Kapitola 1

Úvod

Odkedy existovali počítačové hry, ľudia sa zaoberali tým, ako by mohli hrať proti niekomu, ale bez druhého človeka [6]. Preto sa snažili napodobniť ľudského protihráča. Táto myšlienka dala podnet k prepojeniu umelej inteligencie (AI) a hier. Umelá inteligencia však neznamená, že tento protihráč je inteligentný.

V začiatkoch sa stávalo, že AI hráči (taktiež nazývaní boti) dostávali rôzne výhody oproti ľudským hráčom, aby sa im mohli vyrovnať [20]. Jedna z najčastejších foriem týchto výhod býva napríklad vyššia rýchlosť botov v závodných hrách alebo získanie informácie o pozícii protihráča z enginu herného prostredia, namiesto rôznych zvukových alebo vizuálnych senzorov, ktoré by skôr pripomínali ľudského protihráča.

Táto éra sa však s vývojom čoraz lepšej umelej inteligencie pomaly blíži ku koncu. Hlavne v poslednej dobe prichádzajú na svetlo nové úspechy strojového učenia, ktoré sa vo veľa veciach dokážu vyrovnať človeku a v niektorých ho dokonca dokážu prekonať a to aj bez týchto výhod.

Najúspešnejšie odvetvie strojového učenia na hranie hier je posilňované učenie. S dobrým algoritmom a vhodnými nastaveniami je schopné dosahovať nadľudských výkonov. Takýto model je však potrebné trénovať. Proces trénovania je však niekedy obtiažny, zdĺhavý a taktiež výpočetne náročný.

Toto odvetvie je však stále mladé a tak je potrebné stále skúmať nové prostredia, kde môže byť tento spôsob učenia aplikovaný a to nie len pri hrách, ale aj v reálnom svete. V hernom svete je však integrácia a trénovanie oveľa jednoduchšie oproti tomu reálnemu. V reálnom svete však môžu mať tieto systémy väčšie využitie avšak trénovanie v reálnom svete je oveľa náročnejšie a drahšie. Preto je potrebné stále vyvíjať nové riešenia v týchto herných prostrediach, aby sa raz podarilo simulovať prostredie čo najreálnejšie a teda dostať to najlepšie z oboch svetov, cenu a rýchlosť trénovania v tom virtuálnom a využitie v tom reálnom.

Mojím cieľom v tejto bakalárskej práci je navrhnúť, implementovať a natrénovať model posilňovaného učenia na hre Bomberland. Využijem pri tom algoritmus PPO, ktorým firma OpenAI natrénovala model na jednej z najkomplexnejších hier dnešnej doby [18]. Docielim to využitím knižníc stable baselines 3 a pytorch. Stable baselines 3 je jeden z najpokročilejších frameworkov v oblasti posilňovaného učenia a poskytuje predpripravené algoritmy modelov. Knižnica pytorch poskytuje zase funkcie na tvorbu neurónových sietí, ktoré sú používané v týchto modeloch.

Kapitola 2

Herné prostredie

Táto kapitola bude zameraná na podrobné popísanie prostredia. Je potrebná ku úplnému pochopeniu implementačných detailov interakcií s prostredím. Je určená hlavne pre ľudí nestotožnených s touto hrou alebo jej variáciami.

2.1 Hra Bomberman

Bomberman [11] je arkádová počítačová hra, ktorá je založená v blokovej aréne, kde proti sebe bojujú dvaja alebo viacerí hráči. Arkádová implikuje, že sa jedná o hru, kde je dôležitá koordinácia medzi výstupnými signálmi (zvuk a obraz) a vstupnými signálmi hráča (akcie, ktoré môže v danom hernom prostredí vykonávať, pohyb a iné akcie pomocou stlačení kláves alebo myši).

Cieľom hry je prežiť dlhšie ako nepriateľ. To je možné za pomoci bômb, ktoré sú pokladané na pozícií hráča, ktorý chce bombu položiť. Ak bomba zasiahne ktoréhokoľvek hráča, daný hráč stráca jeden život. Ak prišiel hráč o všetky životy, je z hry vylúčený. Ďalší spôsob ako vyhrať je vyhýbať sa bombám ostatných hráčov a ohňu, ktorý začne vychádzať zo strán na konci hry, väčšinou v špirálovej postupnosti. To znamená, že hráč by sa mal ideálne čo najskôr dostať do stredu, a tiež si ten stred udržať, keďže tam sa oheň dostane ako na posledné miesto.

Prostredie Bomberland

Aj keď bolo popísané, čo to je Bomberman, tak je potrebné vysvetliť aj podrobnejšie detaily tejto variácie.

Bomberland je prostredie inšpirované hrou Bomberman. Toto prostredie bolo vytvorené firmou Coder One [5] a je na ňom postavená táto bakalárska práca. Bolo vytvorené za účelom štúdia a výskumu technológií v oblasti umelej inteligencie. Ukážky a obrázky z tohto prostredia boli získané z datasetu replayov baseline agentov z githubu od užívateľa fxtentacle[1].

Jedná sa o náhodne generované prostredie o rozmere 15x15, kde hráč ovláda za normálnych okolností až 3 herné postavy naraz. Taktiež aj súper má 3 herné postavy. Každá herná postava začína s 3 životmi a víťazí ten hráč, ktorému na konci hry ostane aspoň 1 život. Hra môže taktiež skončiť remízou, ak o posledný život prídu obaja hráči v rovnaký moment (v 1 herný krok = 1 tick).

[1] https://github.com/fxtentacle/gocoder-bomberland-dataset

Obr. 2.1: **Ukážka herného prostredia bomberland.** Na obrázku je zobrazená mechanika konečného ohňa (krajný oheň), výbuchy bômb (okrúhly oheň v krížovom tvare) a nesmrteľnosť postáv po stratení života (priehľadné postavy). Postavy sú rozdelené na rytierov a čarodejníkov. Toto radenie je čisto kozmetické a slúži len na rozlíšenie príslušnosti jednotlivých postáv ku súperiacim tímom. Taktiež je možné vidieť tri druhy prekážkových kociek v hernom prostredí (Kovové kocky, orné kocky a drevené kocky).

Toto prostredie je teda z hľadiska implementácie:

- Diskrétne: herný čas je odkrokovaný. Teda akcie sa vykonávajú po krokoch a kroky sú od seba vzdialené 100 milisekúnd.
- Synchronizované: všetky akcie, ktoré dostane vykonáva v jednom hernom kroku a teda je jedno v akom poradí ich dostane medzi krokmi, vykoná ich aj tak naraz.
- Čiastočne pozorovateľné: niektoré veci sú pred agentom skryté. Napríklad: náhodne sa generujú na rôznych miestach v rôznych časových intervaloch vylepšenia: munícia a pod.

Interakcia s prostredím

Každá jednotka začína s 3 bombami so silou 3. Sila bomby znázorňuje dosah bomby. Bomby sa dajú doplniť za pomoci zdvihnutia kocky typu *ammo* (munícia). Aj dosah bomby sa dá zosilniť zdvihnutím kocky typu *blast power-up* (zosilnenie výbuchu). Bomby vybuchnú najneskôr 100 herných krokov (tickov) od položenia. Ďalšie spôsoby ako sa dá bomba odpáliť skôr sú buď zasiahnutie druhou bombou, čo sa môže stať ihneď po položení, alebo akciou odpálenia bomby (detonate) hráčom, ktorý ju položil, najskôr 5 herných krokov po položení bomby. Hráč si môže vybrať, ktorú zo svojich položených bômb chce odpáliť.

Ak je hráč zasiahnutý ohňom z bomby alebo hru ukončujúcim ohňom, stratí jeden život. V tomto prostredí je však implementovaná mechanika nesmrteľnosti. Hneď po zasiahnutí ohňom sa jednotka stáva nesmrteľnou na niekoľko herných krokov, počas ktorých nestráca životy a môže sa hýbať cez oheň. Odstránenie tejto mechaniky by znamenalo, že jednotka môže zomrieť z plného života už za 3 herné kroky.

V tomto hernom svete tvoria prekážky 3 typy hrou vygenerovaných kociek, cez ktoré sa nedá prejsť.

- Drevené kocky (Wooden Block): dajú sa zničiť bombami a majú 1 život,
- Rudné kocky - Ruda (Ore Block): dajú sa zničiť bombami a majú 3 životy,
- Kovové kocky (Metal Block): nedajú sa zničiť.

Kocky, tak ako aj hráči, stratia jeden život pri každom zasiahnutí výbuchom z bomby. Keď sú zničené, zmiznú a dá sa cez políčko, na ktorých boli, prechádzať.

Ku neprechodným objektom v hernom prostredí patria aj ostatní hráči (či už živí alebo mŕtvi), bomby, okraj mapy a v prípade, že sa na to isté políčko snažia dostať dvaja hráči naraz v jednom momente, nepohne sa ani jeden a políčko ostane prázdne. Je to spôsob, ako vyriešiť synchronizované pohyby v hernom prostredí a teda nikto nedostane prednosť a teda ani výhodu.

V Bomberlande sa niektoré známe vylepšenia z ostatných variácií tejto hry nenachádzajú. Napríklad sa jedná o zrýchlenie jednotky (väčšinou v spojitom prostredí, v diskrétnom by mohlo spôsobiť zlé odkrokovanie, musela by byť akcia rozšírená o počet políčok, ktoré chce jednotka prejsť v danom kroku) alebo hádzanie/kopanie bomby.

Kapitola 3

Posilňované učenie

Táto kapitola bola čerpaná z knihy doktora Suttona a doktora Barta o úvode do posilňovaného učenia. Vysvetľuje čo vlastne je posilňované učenie a popisuje jeho základné časti a princípy.

3.1 Klasifikácia posilňovaného učenia

Posilňované učenie nepatrí ani k forme učenia s učiteľom (supervised learning) ale ani k forme učenia bez učiteľa (unsupervised learning) [22]. Čo to teda je?

Od učenia s učiteľom je posilňované učenie rozdielne v tom, že pri učení s učiteľom máme trénovaciu sadu, ktorá pozostáva z označených príkladov, ktoré sú poskytnuté vonkajším učiteľom. Toto však nie je veľmi praktické, v niektorých prípadoch priam nemožné, pri interakcii s prostredím a jeho meniacimi sa stavmi. A to preto, že často nie sú tieto príklady správne alebo nereprezentujú všetky možné situácie, ktoré môže dané prostredie priniesť [22].

Na druhej strane, od učenia bez učiteľa, je posilňované učenie odlišné zase v tom, že učenie bez učiteľa hľadá skryté podobné štruktúry v neoznačených dátach. Mohlo by sa zdať, že je to to isté, pretože ani jedno sa nespolieha na správnosti príkladov pri trénovaní. Posilňované učenie sa avšak snaží maximalizovať odmeny, nie hľadať skryté štruktúry, aj keď to môže byť užitočné pre agenta, samotné to nerieši získavanie maximálnej možnej odmeny [22].

Znamená to, že posilňované učenie je samostatná forma strojového učenia.

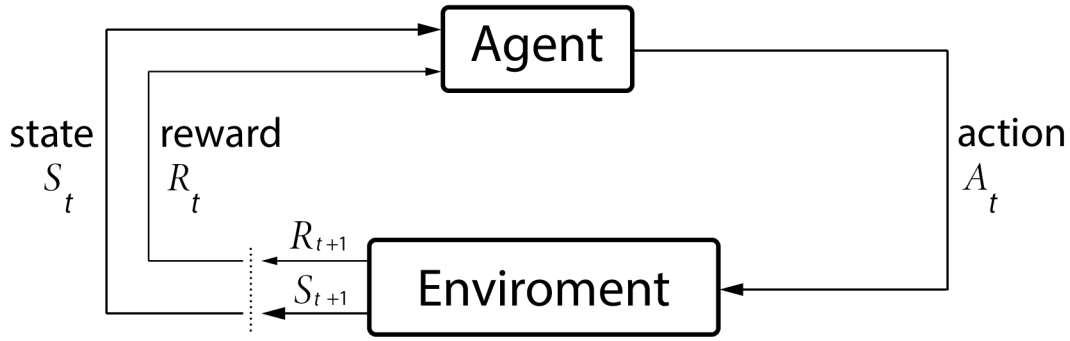
3.2 Časti posilňovaného učenia

Hlavné časti posilňovaného učenia sú agent a prostredie.

Agent môže mať rôzne podoby. Je to program, ktorý vykonáva akcie, ktoré sú mu posielané svojou politikou (policy) na základe informácií, ktoré dokáže pozorovať v danom prostredí [22].

Prostredie nemusí znamenať len veci externé. Napríklad v prípade Bomberlandu sú aj vlastnosti postáv súčasťou prostredia. Vo všeobecnosti to môžu byť všetky dostupné informácie, ktoré agent môže získať, poprípade aj predvídať, respektíve simulovať (hlavne v čiastočne pozorovateľnom prostredí) [22].

Ďalšia dôležitá časť tohto systému je **odmeňovacia funkcia**. Jej hlavnou činnosťou je dávať odmeny alebo tresty agentovi na základe jeho akcií. Je to jedna z najťažších vecí pri implementácii posilňovaného učenia.



Obr. 3.1: **Markovov reťazec** popisujúci interakciu agenta a prostredia v kontexte posilňovaného učenia [22]. Prostredie posiela agentovi akciu, aktuálny stav S a odmenu R, pre čas t, na ktorú agent reaguje akciou A pre čas t, ktoré prostredie vyhodnotí a pošle stav S a odmenu R pre čas $t + 1$.

3.3 Princíp fungovania posilňovaného učenia

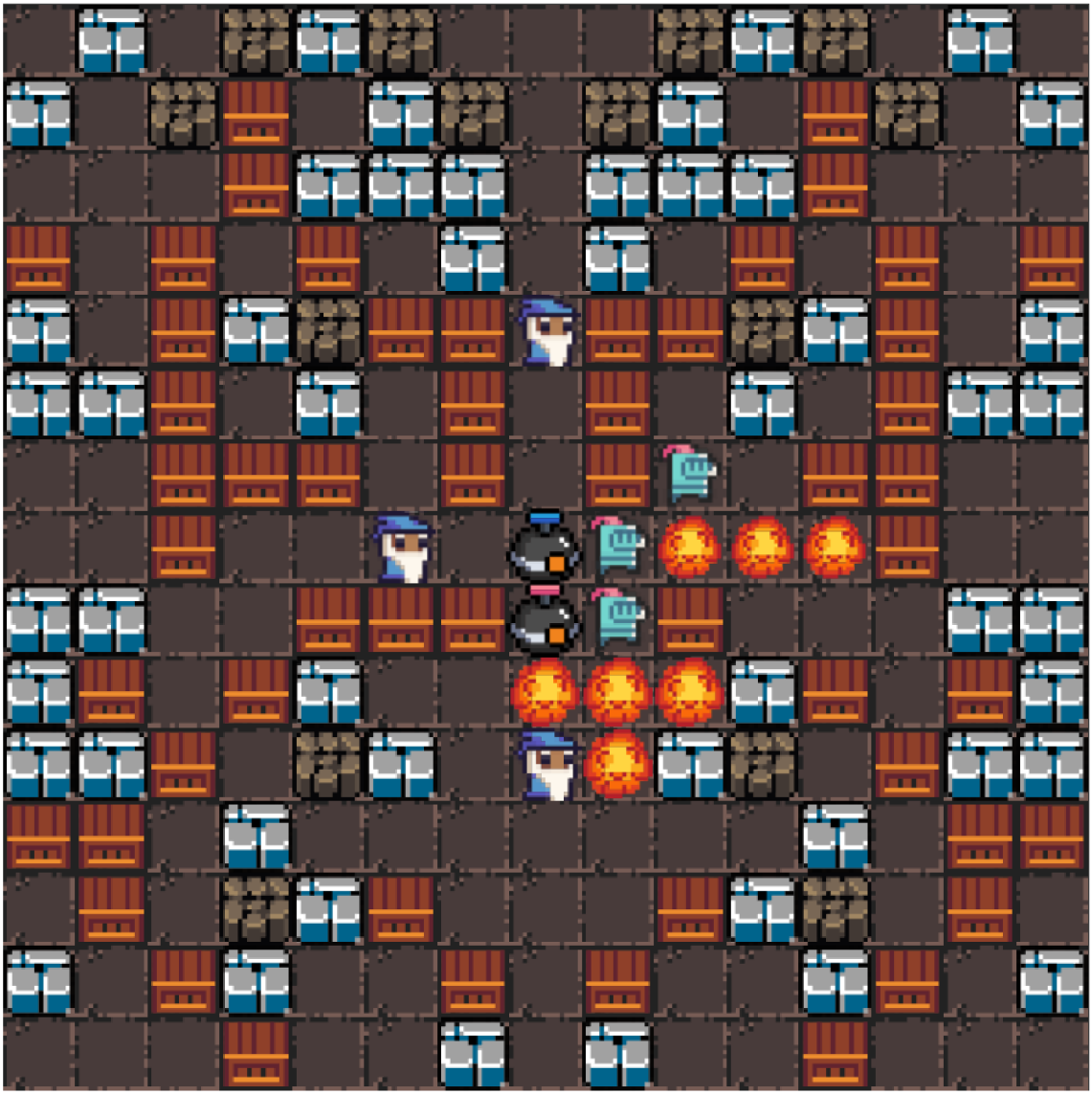
Posilňovanie pozitívneho správania (positive reinforcement) je známa časť psychológie a vychovávania. Používanie odmien a trestov je jedna z najrozšírenejších metód výchovy detí a trénovania zvierat na spoluprácu s človekom [7]. Posilňované učenie v strojovom učení funguje na rovnakom princípe ako posilňovanie pozitívneho správania, z ktorého je odvodené.

Jeden z najznámejších behavioristov Burrhus Frederic Skinner, ktorý sa zaoberal touto tematikou hovorí o tom, že existujú štyri druhy odmien a trestov [7]:

- Pozitívne posilnenie (positive reinforcement): Zavedenie pozitívnej stimulácie (odmeny), keď je dosiahnutý chcený výsledok.
- Pozitívne potrestanie (positive punishment): Zavedenie negatívnej stimulácie (trestu), keď je dosiahnutý nechcený výsledok.
- Negatívne posilnenie (negative reinforcement): Odstránenie negatívnej stimulácie (trestu), keď je dosiahnutý chcený výsledok.
- Negatívne potrestanie (negative punishment): Odstránenie pozitívnej stimulácie (odmeny), keď je dosiahnutý nechcený výsledok.

Táto metóda by mala docieliť zvýšenie výskytu akcií, ktoré vedú ku chcenému výsledku pokiaľ sa subjekt dostane do rovnakej situácie, ako pri trénovaní. Taktiež by mala znížiť pravdepodobnosť, že subjekt vyberie jednu z akcií, ktoré budú viesť ku nechcenému výsledku. B. F. Skinner to pomenoval ako princíp posilňovania (principle of reinforcement) [8].

Obr. 3.2: **Rozhodovanie v Bomberlande. Časť 1** Jednotka, ktorá stojí v strede (kúzelník na pozícii 7,7) vyhodnocuje stav. Drží výhodnú pozíciu a vidí, že sa práve zrodila kocka typu munícia, ku ktorej má priamu cestu. Z predchádzajúceho učenia vie, že sa mu oplatí tieto kocky zbierať.



Obr. 3.3: **Rozhodovanie v Bomberlande. Časť 2** Keďže nechce prenechať pozíciu, na ktorej je, nepriateľovi, čo vie, že je zlé, položí bombu na zem a rozhodne sa ísť po muníciu. Neodpálil však bombu hneď ako to bolo možné (po 5 herných krokoch) ale počkal, kým sa bude vracať späť do stredu a až potom ju odpálil, aby mal čas na návrat. Keď bomba vybuchla, zobrala život nepriateľovi a uvoľnilo sa stredné políčko.

Obr. 3.4: **Rozhodovanie v Bomberlande. Časť 3** Nakoniec sa dokázal vrátiť späť do stredu. Z krátkodobého hľadiska by sa tejto jednotke neoplatilo odísť z tejto pozície avšak z dlhodobého hľadiska sa to veľmi oplatilo. Zobral život nepriateľskej jednotke, získal kocku typu munícia a znovu obsadil výhodnú pozíciu a to bez stratenia vlastného života.

Takže ak agent vykoná akcie, ktoré ho dovedú ku chcenému výsledku, tak ho prostredie odmení. Naopak, ak vykoná akcie, ktoré ho vedú ku nechcenému výsledku, tak ho potrestá. Poprípade, ak nevykoná ideálny krok, tak dostane menšiu odmenu, poprípade žiadnu. Na druhej strane, ak je v zlej situácií ale zlepšuje sa, tak sa zníži alebo odstráni trest [7].

- **Pozitívne posilnenie** (positive reinforcement): Zavedenie pozitívnej stimulácie (odmeny), keď je dosiahnutý chcený výsledok.
- **Pozitívne potrestanie** (positive punishment): Zavedenie negatívnej stimulácie (trestu), keď je dosiahnutý nechcený výsledok.
- **Negatívne posilnenie** (negative reinforcement): Odstránenie negatívnej stimulácie (trestu), keď je dosiahnutý chcený výsledok.
- **Negatívne potrestanie** (negative punishment): Odstránenie pozitívnej stimulácie (odmeny), keď je dosiahnutý nechcený výsledok.

Ďalšie rozdelenie môže byť podľa odstupu dodania odmeny od vykonania akcie a od počtu posilnení. V posilňovanom učení sa tomu hovorí okamžité odmeny (immediate reward) a oneskorené odmeny (delayed reward) [13]. Podľa počtu posilnení ide zase o husté (dense reward) a riedke (sparse reward) odmeňovanie [7].

- **Nepretržitý prehľad** (Continuous schedule): akcia je odmenená alebo potrestaná ihneď a po každom vyskytnutí (kroku). V realite ťažké udržať - nemáme subjekt stále pod dohľadom - avšak v hernom svete je to väčšinou možné.
- **Pevný pomer** (Fixed ratio): akcia je posilnená po pevne stanovenom počte výskytov. Napríklad po každých piatich rovnakých akciách dostane agent odmenu alebo trest.
- **Pevný interval** (Fixed interval): akcia je posilnená v pevne stanovených časových intervaloch, respektíve po pevne stanovenom počte krokov. Napríklad každých desať krokov. Rozdiel od pevného pomeru je v tom, že akcie môžu byť rôzne.
- **Pohyblivý pomer** (Variable ratio): akcia je posilnená po meniacom sa počte výskytov. Napríklad najprv pri jednom, potom pri troch a postupne sa zvyšuje alebo znižuje.
- **Pohyblivý interval** (Variable interval): akcia je posilnená v meniacich sa časových intervaloch, respektíve herných krokoch. Napríklad ak začiatok hry nie je taký dôležitý ako jej koniec, agent bude hodnotený menej často na začiatku, len či ide dobrým smerom, môže teda tvoriť volnejšie stratégie, koniec hry je dôležitejší, bude hodnotený častejšie.

Je dôležité dobre rozvrhnúť veľkosti odmien a taktiež aj za čo ich agent dostáva. V niektorých prípadoch aj kedy odmenu dostáva. Je potrebné sa vyhnúť odmenám, ktoré môžu spôsobiť predlžovanie hry. Môže to byť spôsobené tým, že agent dostane vyššiu odmenu za predlžovanie hry aj keď cieľom je poraziť súpera čo najrýchlejšie. Toto sa dá ošetriť napríklad aplikovaním zliav na odmeny (discount). Odmeny za tú istú vec na začiatku hry sú teda vyššie ako v neskorších štádiách hry.

3.4 Existujúce riešenia

Pri vytváraní tejto bakalárskej práce som taktiež skúmal, aké existujúce riešenia pre podobnú problematiku už existujú. Ako prostredie, kde je jedna z hlavných vlastností to, že sa v ňom nachádza viacero jednotiek, ako priateľských tak aj nepriateľských, tak som sa rozhodol, inšpirovať jednou z najznámejších implementácií posilňovaného učenia v hrách a pravdepodobne aj jednou z najúspešnejších v tomto odbore.

Jedná sa o agenta na hru Dota 2 vytvoreného firmou OpenAI. Tento agent dokázal poraziť najsilnejší tím na svete: Team OG. Jedná sa o obrovský úspech a to z dôvodu, že Dota 2 má veľmi komplexný herný priestor, znázornené na obrázku 3.5. Najzložitejšie vlastnosti sú: veľmi dlhé hry (šach má priemerne 80 krokov za epizódu, pre porovnanie Dota 2 má 20000 krokov za epizódu), čiastočne pozorovateľný priestor (agent nevidí súperov stále, vidí ich len keď sú v priestore, ktorý má viditeľný a teda musí veľa predvídať, kde súperi sú) a vysoko dimenzionálny pozorovací priestor (mnoho entít a vlastností). Taktiež má najvyššie víťazné odmeny zo všetkých e-športov na svete. V roku 2021 na turnaji The International bola celková výhra až 40 miliónov dolárov [1]. Záujem o túto hru je obrovský a teda je tam aj veľká konkurencia. O to väčší úspech je poraziť najlepší tím na svete. Bol uskutočnený aj verejný test, kde Open AI five vyhralo 7,215 hier a prehralo iba 42 hier proti ľudským hráčom. To sa rovná 99.4% úspešnosti.

Tým sa potvrdilo, že posilňované učenie, v kombinácii s hraním proti sebe (self-play), je možné využiť na dosiahnutie nadľudských výkonov. Podarilo sa im to za použitia modifikovaného algoritmu PPO (Proximal Policy Optimization), ktorý využíval zdieľaný LSTM (Long-Short Term Memory) blok, z ktorého potom vychádzali dve plne prepojené neurónové siete pre činiteľa (actor) a kritika (critic) [18]. Jediný problém je, že sa jedná o zložité prostredie a PPO je on-policy algoritmus, čo znamená, že sa učí len z akcií, ktoré vykoná. Tým pádom je učenie časovo a výpočetne náročnejšie. Keďže sa však Bomberland zložitosťou nepribližuje Dote tak som si myslel, že to nebude problém.

Týmto zistením som si vybral algoritmus, ktorý by som chcel použiť. LSTM však nie je vhodné na použitie s obrázkami [3], v Dote 2 používali ako pozorovanie vektory a teda som to musel nahradiť.

Podľa článku Jasona Brownleeho [3], som dospel k záveru, že by mohlo fungovať MLP (Multi-Layer Perceptron) a CNN (Convolutional Neural Network), pretože moje prostredie je reprezentované ako obrázok.

[1] https://www.esportsearnings.com/tournaments

Obr. 3.5: **Nahliadnutie na rozhodovacie možnosti agentov v hre Dota 2.** Aj keď sa jedná o spojité prostredie v spojitom čase, je potrebné ho diskretizovať. To je dosiahnuté tým, že agent vykonáva akciu každý 4. snímok (to je 1 herný krok) [18] a prostredie, je taktiež diskretizované, relatívne podľa entít tak, ako je znázornené na obrázku. Zdroj: https://openai.com/blog/openai-five/

3.5 Použité algoritmy posilňovaného učenia

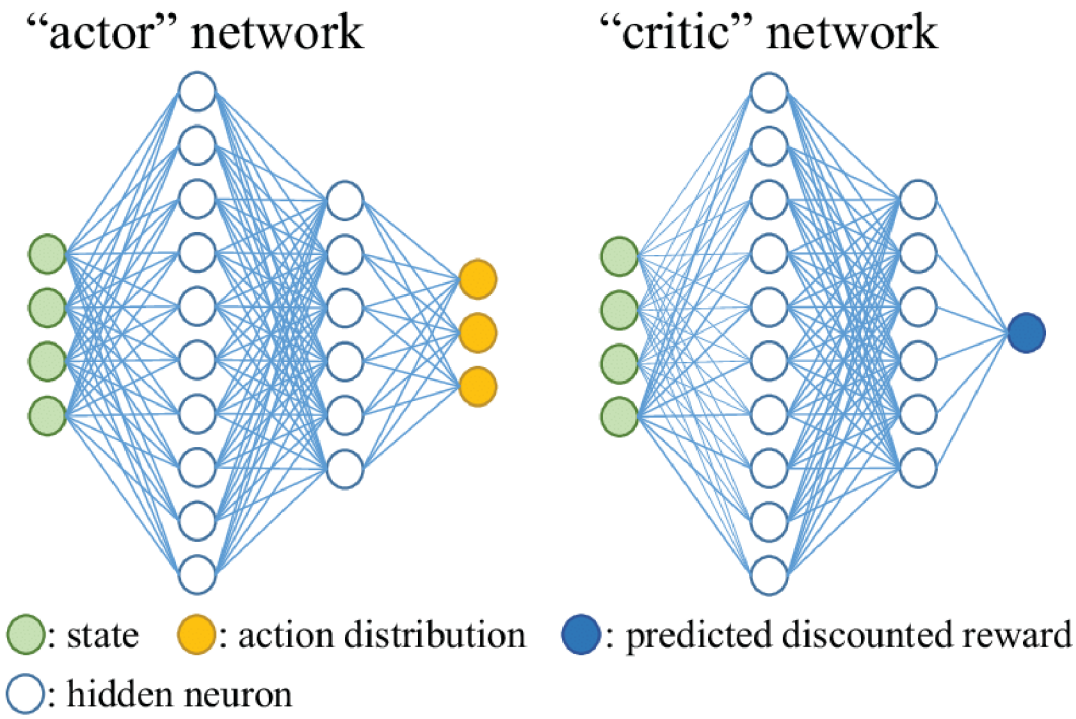
Rátajme so štandardným posilňovaným učením, kde agent interaguje s prostredím ε v diskrétnom čase. V každom časovom kroku t agent dostane stav s_t a vyberie akciu a_t z priestoru akcií A podľa svojej politiky π, kde π je mapovanie stavov s_t ku akciám a_t. Ako návratnú hodnotu dostane agent ďalší stav $s_t + 1$ a skalárnu odmenu r_t. Tento proces pokračuje, dokým agent nedocieli terminálneho, respektíve konečného, stavu. Následne sa celý cyklus začne odznova. Návratová hodnota $R_t = \sum_{k=0}^{\infty} \gamma^k r_t + k$ pozostáva zo všetkých získaných hodnôt v kroku t so zľavovým faktorom $\gamma \in (0, 1]$ [9].

Hodnota akcie $Q^\pi(s, a) = E[R_t|s_t = s, a]$ je očakávaná návratová hodnota pri vybraní akcie a v stave s pri nasledovaní politiky π. Optimálna hodnota funkcie $Q^*(s, a) = max_\pi Q^\pi(s, a)$ dáva maximálnu hodnotu akcie pre stav s a akciu a dosiahnuteľnou akoukoľvek politikou. Dá sa teda povedať, že hodnota stavu s pod politikou π je definovaná ako $V^\pi(s) = E[R_t|s_t = s]$ a je to teda očakávaná návratová hodnota pri nasledovaní politiky π zo stavu s. Cieľom agenta je teda maximalizovať očakávanú návratovú hodnotu z každého stavu s_t [9].

Optimalizácia proximálnej politiky

Proximal Policy Optimization (PPO) je algoritmus pre posilňované učenie, ktorý vychádza z metód Policy Gradient (Gradient Politiky). Alternuje medzi vzorkovacími dátami cez interakciu s prostredím a optimalizáciou skrytého cieľa za pomoci náhodných stúpaní po

gradiente. Táto metóda vychádza z TRPO (Trust Region Policy Optimization) a mala by byť ľahšia na implementáciu, viac všeobecná a mať lepšiu vzorkovú efektivitu [17].



Obr. 3.6: **Aktor Kritik neurónová sieť v PPO.** Zdroj: [12]

Výhoda Aktor Kritik

Ako alternatívny algoritmus som taktiež vybral Advantage Actor Critic (A2C). Podľa dokumentácie by mal dosahovať rýchlejšieho učenia na procesore ako na grafickej karte. Je rovnako alebo v niektorých prípadoch aj viac efektívny ako algoritmus A3C (Asynchronous Advantage Actor Critic), z ktorého je A2C odvodený [9].

Kritik odhaduje funkciu hodnoty. Toto môže znamenať aj hodnotu funkcie Q alebo hodnotu stavu V. Aktor aktualizuje politiku v navrhovanom smere kritika (tak ako pri policy gradient) [23].

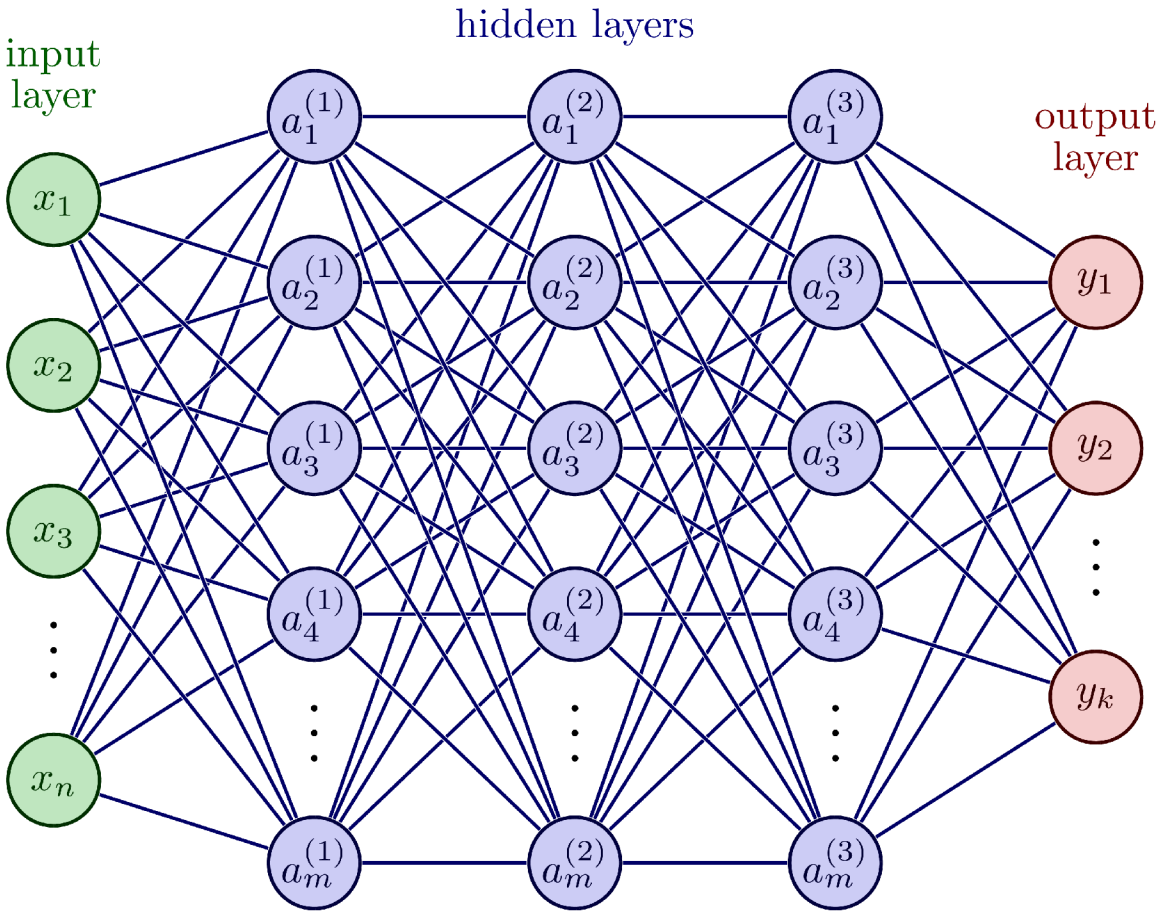
Tento algoritmus som vybral po tom, čo som nemal úspešné výsledky s PPO. Aj keď je A2C horší od PPO, je mu dosť podobný a teda ak by sa mi podarilo nájsť nejakú chybu na ňom, tak by som skôr vedel kde je chyba v PPO.

3.6 Použité neurónové siete

Neurónové siete sa často používajú na vyriešenie problémov spojených so strojovým učením. Sú zložené z mnohých jednoduchých procesných uzlov, ktoré sú spojené a sú vzdialene podobné ľudskému mozgu. Väčšinou sú uložené vo vrstvách. Prepojenia medzi uzlami majú medzi sebou váhy, ktorými sa násobí vstup. Cieľom neurónových sietí je upraviť tieto váhy tak, aby docielili požadovaného výstupu s daným vstupom. To je docielené mnohými po-

súvaniami vpred (feed-forward) a spätnou propagáciou (back propagation) trénovacích dát po celej sieti [4].

Neurónová sieť typu MLP



Obr. 3.7: **Neurónová sieť typu MLP.** Táto sieť je plne prepojená a je to jedna zo základných druhov neurónových sietí. Pri obrázkoch mapuje jeden pixel ku jednému perceptrónu na vstupnej vrstve. Je teda jednoduchá na implementáciu avšak nie je najefektívnejšia na spracovanie obrázkov. Každá vrstva v skrytých vrstvách by taktiež mala byť zakončená nelinearitou, napríklad ReLU (Rectified Linear Unit), ktorá odstráni všetky negatívne hodnoty, ktoré by sa na výstupe danej vrstvy mohli objaviť. Na výstupe sa môžu nachádzať rôzne formáty hodnôt. V niektorých prípadoch sa používajú sigmoid funkcie: napríklad pri binárnych klasifikáciách, v niektorých zase bez aktivačnej funkcie: napríklad pri regresných problémoch. Pri klasifikáciách viac ako dvoch tried je potrebné použiť softmax funkciu, ktorá vydá pravdepodobnosti pre každú klasifikačnú triedu [2]. Zdroj obrázka: [16].

Multi-Layer Perceptron (Viac-Vrstvový Perceptron) je plne prepojená umelá neurónová sieť (ANN, Artificial Neural Network) [21]. Na spracovanie obrázkov je úprava celkom jednoduchá. Obrázky sa narežú po stĺpcoch a uložia pod seba. Pri dvojrozmernom hracom poli o veľkosti 15x15 tým vznikne jednorozmerné pole o veľkosti (15x15)x1, teda 225x1. Následne sa každý pixel priradí ku jednému perceptrónu na vstupnej vrstve. Toto je síce

jednoduché na úpravu prostredia pre vstup do tejto siete, avšak tieto siete bývajú kvôli tomu veľmi veľké. Z týchto dôvodov sa taktiež ťažšie trénujú.

Jedna časť je, že majú veľa váh (weights), ktoré treba aktualizovať pri učení. Druhá časť je to, že ak sa táto neurónová sieť naučí rozpoznávať mačku v ľavom hornom rohu obrázku, tak ju pravdepodobne nerozpozná ak sa objaví na inom mieste. Ak sa začne objavovať na inom mieste, neurónová sieť sa znova začne učiť od začiatku, že býva tam a zabudne, že bývala vľavo hore. Pre tento istý dôvod je ľahšie tento typ siete pretrénovať, respektíve ťažšie natrénovať na zovšeobecňovanie [10]. Z tohto dôvodu je konvolučná neurónová sieť efektívnejšia na riešenie obrázkových vstupov.

Konvolučná neurónová sieť

Convolutional Neural Network (CNN) je narozdiel od MLP len čiastočne prepojená umelá neurónová sieť (obrázok 3.8). Na spracovanie vstupu využíva konvolúcie. Stále je však zakončená plne prepojenou neurónovou časťou, ktorá sa učí. Túto časť tvoria skryté vrstvy. Konvolučné vrstvy využívajú funkciu konvolúcie na vyťahovanie potrebných vlastností, ktoré sú následne predávané do tejto plne prepojenej siete [15].

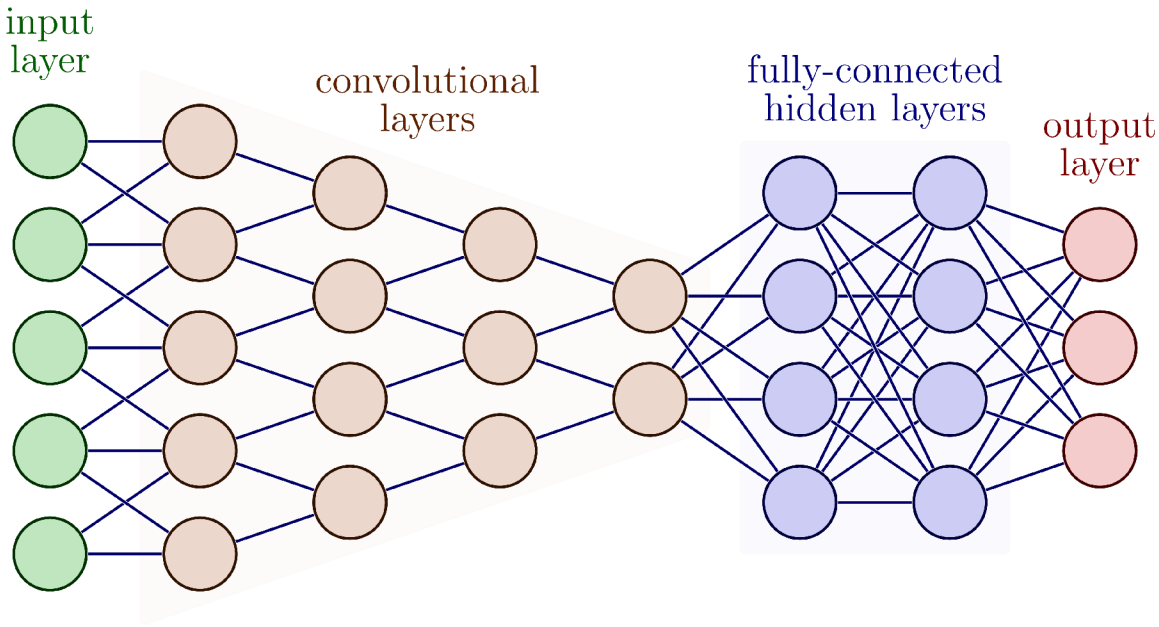
Konvolučné vrstvy fungujú na princípe násobenia vstupu filtrami a výsledok tejto operácie je sčítaný. Konvolúciou sú menené počty kanálov. Môže ich aj zvyšovať aj znižovať. Pre konvolúciu sú dôležité tieto parametre: rozmery vstupného obrazu, rozmery filtra, typ filtra, vyplnenie (padding) a posun (stride) [15]. Ak je potrebné zvýšiť alebo znížiť počet kanálov bez straty informácií, môže sa použiť kernel (filter) o veľkosti 1x1.

Rozmer filtra by mal byť menší ako rozmer vstupného obrazu z dôvodu, že bude tento filter posúvaný po celej ploche vstupného obrázku. Bližší popis pri obrázku 3.9.

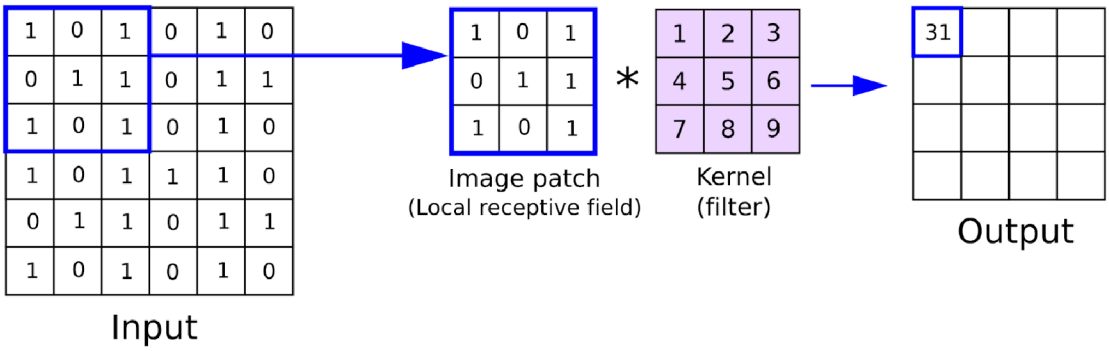
Typ filtra zase určuje, aká vlastnosť bude zvýraznená. Môže obrázok rozmazať, zaostriť, vytiahnuť vertikálne alebo horizontálne čiary (detekcia hrany). Táto funkcia však potrebuje vždy byť zakončená nejakou nelinearitou, respektíve aktivačnou funkciou. Najvyužívanejšia aktivačná funkcia je ReLU: $f(x) = max(0, x)$ (Rectified Linear Unit), ktorá odstráni záporné hodnoty [15].

Taktiež môžu byť využívané rôzne poolingy, ktoré menia rozmery jednotlivých kanálov. Najpoužívanejšie je MaxPooling (Maximálna hodnota) a AveragePooling (Priemerná hodnota). MaxPooling z pola (napríklad 2x2), vyberie najvyššiu hodnotu a uloží ju. Toto efektívne zmenší veľkosť rozlíšenia obrázku na jednu štvrtinu pôvodnej veľkosti. Nevýhoda však je, že sa môžu niektoré informácie stratiť. AveragePooling na druhej strane taktiež znižuje veľkosť rovnako ako MaxPooling, avšak nevyberá najvyššiu hodnotu ale spracuje dané pole na priemernú hodnotu [15].

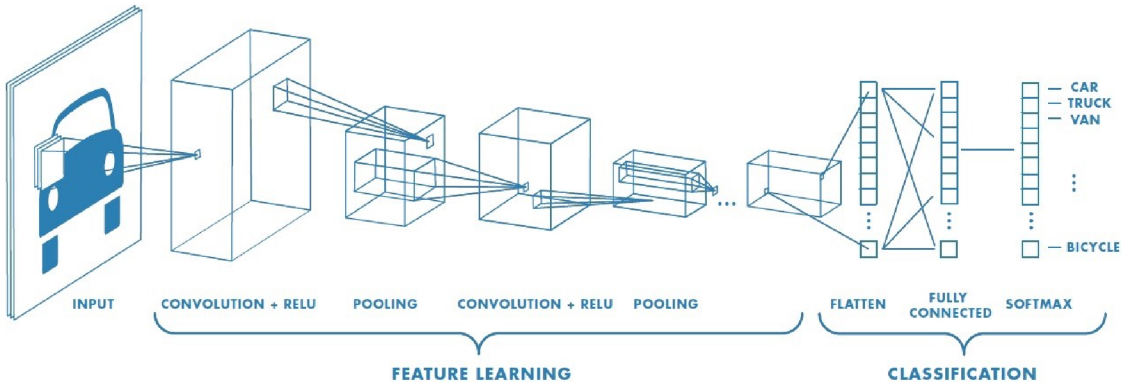
Celý proces konvolučnej neurónovej siete je znázornený na obrázku 3.10.



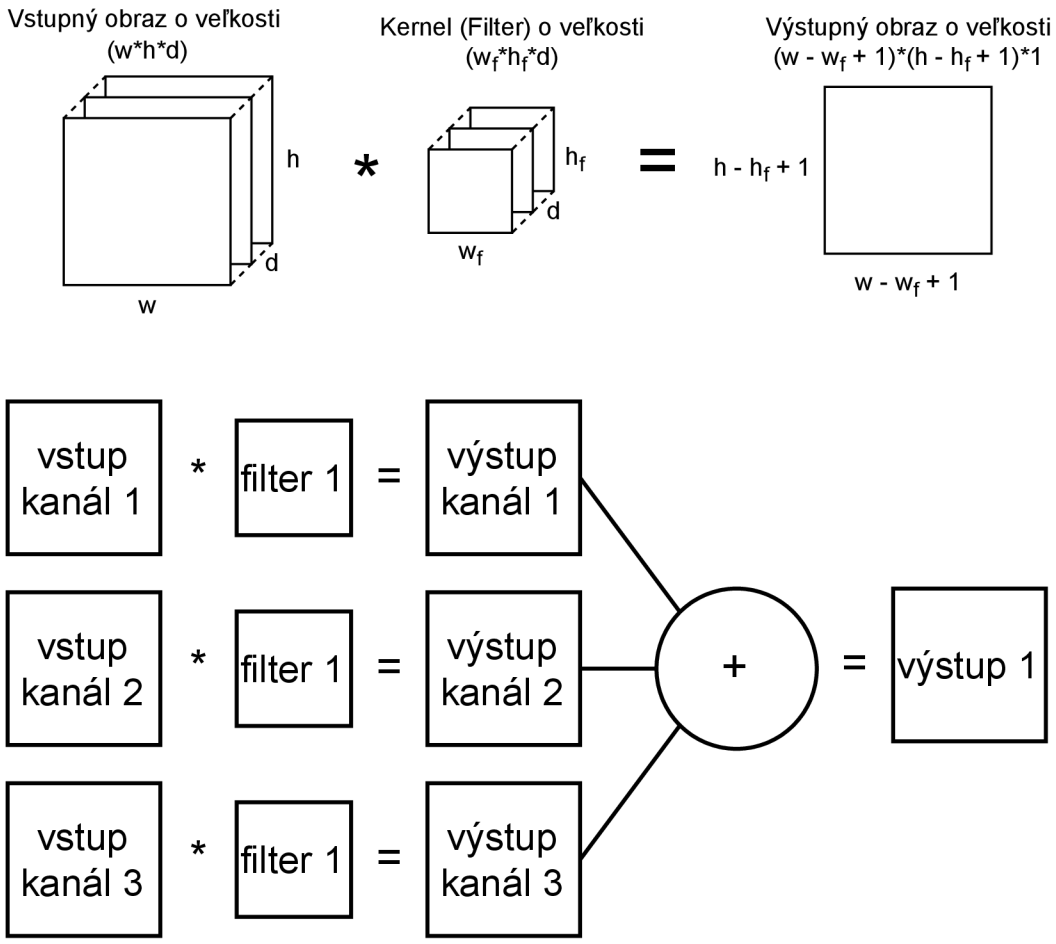
Obr. 3.8: **Konvolučná neurónová sieť.** [16] Na obrázku je znázornená 1D konvolučná sieť. Skladá sa zo štyroch druhov vrstiev. Prvá vrstva je vstupná, nasledujú konvolučné vrstvy, ktoré vedú do plne prepojenej neurónovej siete v skrytých vrstvách a nakoniec výstup. Tu sú vstupné dáta o veľkosti 5, s vyplnením 1, posun 1 a filtrom o veľkosti 3.



Obr. 3.9: **Proces konvolúcie**[14]**.** Filter je posúvaný o hodnotu posunu (stride) po celom obrázku (z ľavého horného rohu až po pravý dolný roh) po riadkoch. Vynásobuje hodnoty obrázku s hodnotami filtra, ktoré následne sčíta a dosadí do nového obrázka na pozíciu stredu filtra v starom obrázku. Aby sa zabránilo strate vlastností na kraji obrázka, môže sa využiť vyplnenie (padding), tak, aby každý pixel bol aspoň raz v strede filtra. Na vypočítanie veľkosti výstupného obrázku sa používa tento vzorec:
$OUTPUT = ((INPUT - KERNEL + 2 * PADDING)/STRIDE) + 1.$



Obr. 3.10: **Vysoký náhľad na spracovanie obrázka konvolučnou neurónovou sieťou[15].** Na obrázku je vykreslený názorný príklad spracovania obrázku konvolučnou neurónovou sieťou. Každá konvolučná vrstva je zakončená aktivačnou funkciou ReLU a následne je vykonaný pooling na zníženie rozlíšenia. Keď je dosiahnutý chcený počet kanálov a rozlíšenie, výstup z extrakcie vlastností je zrovnaný do jednej dimenzie (flatten) a následne predaný plne prepojenej neurónovej sieti.



Obr. 3.11: **Konvolúcia viacerých vrstiev.** Obrázok 1 [15]: Vstupný obraz o veľkosti Šírka w, Výška h a HĺBka, respektíve počet kanálov, d je vynásobená filtrom o veľkosti Šírka Filtra w_f, Výška Filtra h_f a HĺBka (rovnaký počet kanálov ako vo vstupnom obraze) d. Výstupný obraz bude veľkosti Šírka $w - w_f + 1$, Výška $h - h_f + 1$ a HĺBka, teda počet kanálov, 1. Obrázok 2: Keďže sa na obrázku 1 nedalo prehľadne vysvetliť čo je čo, znázornil som to druhým obrázok, ktorý vysvetľuje, že filter je pre všetky vstupné kanály rovnaký. Tie sa následne sčítajú a dostaneme jeden výstupný kanál. Takže keď máme x výstupných kanálov, tak to znázorňuje x použitých filtrov a nie počet vstupných kanálov. Na výstup je následne aplikovaná aktivačná funkcia, ktorá vytvorí potrebnú nelinearitu (odstránenie záporných hodnôt) a tým získame mapu vlastností (feature map).

Kapitola 4

Príprava trénovacieho prostredia

4.1 Komunikácia medzi klientmi a herným serverom

Ako už bolo spomenuté Bomberland je diskrétne a synchronizované prostredie. Komunikácia medzi klientmi a serverom však nie je synchronizovaná, klienti odosielajú svoje akcie v rôznych časoch a taktiež prichádzajú na server v rôznom čase, nie naraz. Server teda vždy čaká 100 ms na všetky akcie a až potom hru odkrokuje a následne pošle všetkým klientom aktualizácie stavu.

Táto komunikácia prebieha vo formáte JSON. Celé stavy, aj tie začiatočné, sú vo formáte JSON a teda sa dá vytvoriť vlastné herné pole. Dajú sa zmeniť rôzne nastavenia, o ktorých sa hovorilo v kapitole 3. Napríklad môžeme zmeniť počet jednotiek, ktoré má každý hráč k dispozícii, kde sa nachádzajú, koľko majú bômb na začiatku hry, silu výbuchov a dokonca aj počet životov.

Aj akcie zo strany klienta sú vo formáte JSON. Viac informácií je dostupných v oficiálnej dokumentácii [5].

Toto je klasický mód fungovania prostredia Bomberland. Taktiež existuje trénovací mód, kde môžu všetky akcie prichádzať z jedného zdroja a herný server len vyhodnocuje všetky akcie a stavy. Je postavený na AI-gym platforme.

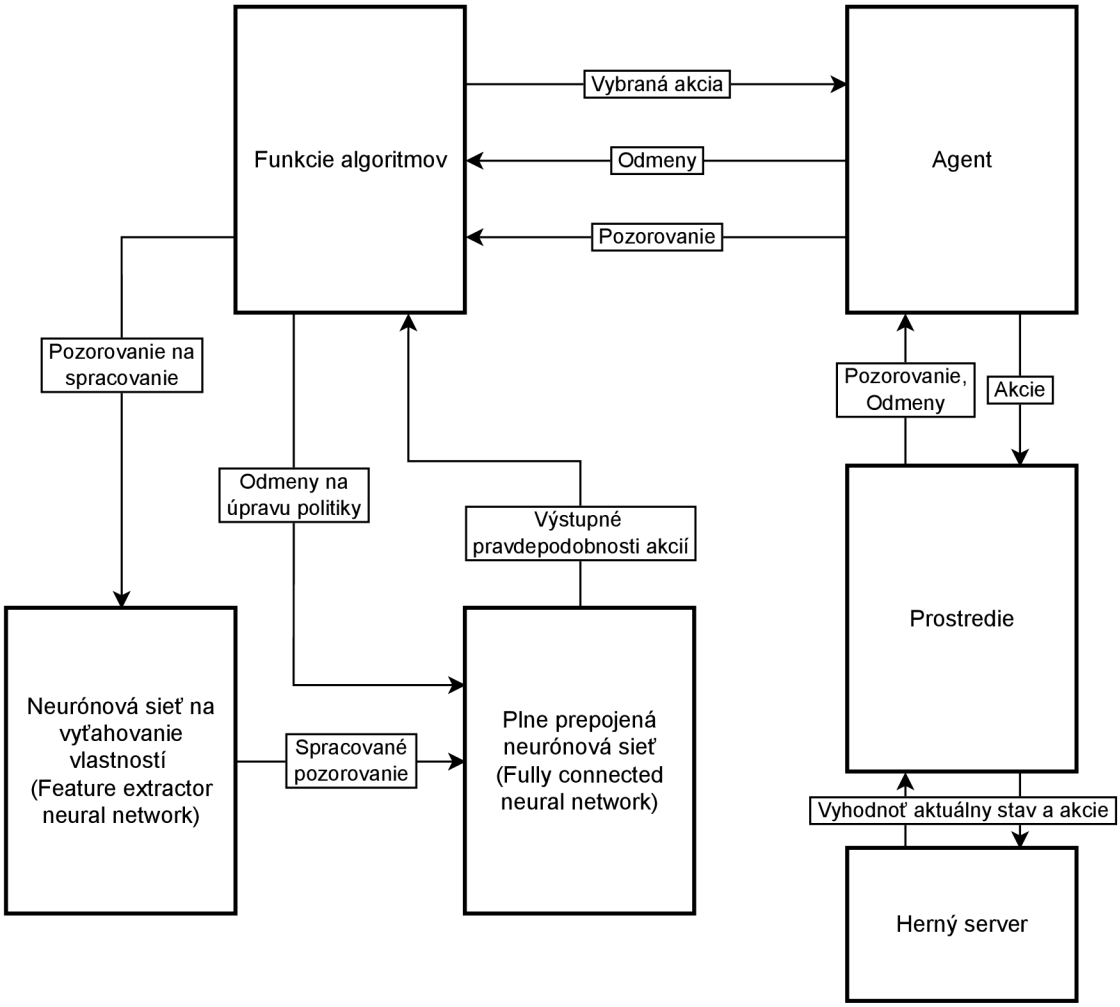
4.2 Platforma AI-gym

Gym (z angličtiny, znamená telocvičňa, miesto kde sa cvičí, respektíve trénuje) je nástroj na vývoj a porovnávanie algoritmov posilňovaného učenia. Nevytvára žiadne predpoklady na štruktúru agenta a je kompatibilný s mnohými numerickými výpočetnými knižnicami, ako napríklad Tensorflow alebo Theano [19].

Knižnica Gym je kolekciou testovacích problémov alebo prostredí, ktoré sa dajú použiť na prácu s rôznymi algoritmami posilňovaného učenia. Vytvára teda rozhranie medzi funkciami štandardizovaných knižníc zo strojového učenia a herným prostredím [19].

Popis rozhrania Gymu

Základné funkcie, ktoré musí rozhranie gym obsahovať sú: **step** (krok), **reset** (vynulovať) a **render** (vykresliť). Taktiež musí obsahovať aj nasledujúce atribúty: **action_space** (priestor akcií), **observation_space** (priestor pozorovania), **metadata** (dáta o dátach, metadáta) a **reward_range** (rozsah odmien).



Obr. 4.1: **Model funkcionality prostredia Bomberland v trénovacom móde gym.** Začiatok tohto cyklu začína v prostredí. Vygeneruje sa náhodné prostredie a pošle sa agentovi. Agent spracuje vstup (prvá iterácia bez odmeny, agent ešte nevykonal akciu), zašle pozorovanie cez funkcie algoritmov do neurónovej siete na extrahovanie vlastností. Spracované pozorovanie sa pošle do plne prepojenej neurónovej siete (časť neurónovej siete, ktorá sa má učiť na základe odmien, neurónová sieť na extrahovanie vlastností sa nemení). Táto sieť spracuje vstup a vráti pravdepodobnosti akcií naspäť do funkcií algoritmov. Tieto pravdepodobnosti spracujú a vrátia agentovi index akcie, ktorú má použiť. Agent zašle index do prostredia, ktoré ho spracuje do formátu na komunikovanie so serverom. Ten vyhodnotí nový stav a pošle ho naspäť prostrediu. Prostredie tentokrát vráti aj odmenu za akciu, ktorú agent vykonal. Na základe tejto odmeny sa postupne bude upravovať (učiť) plne prepojená neurónová sieť.

Funkcie Gymu

Step: Táto funkcia je volaná po vybratí akcie, ktorú chce agent poslať, v aktuálnom kroku, serveru na vyhodnotenie. Jej návratové hodnoty sú **observation** (pozorovanie), **reward** (odmena), **done** (ukončená epizóda) a **info** (dodatočné informácie).

Všetky procesné úkony by mali byť zabezpečené touto funkciou, teda aspoň na ideálne pracovanie s funkciami algoritmov z knižnice stable baselines. Pretože sa táto funkcia volá zvnútra agenta, len s argumentom čísla indexu akcie, ktorú chce vykonať, je nutné ho spracovať do vhodného formátu pre komunikáciu na server4.3. To isté platí aj pre výstupné hodnoty funkcie. Pozorovanie (observation) by teda malo byť spracované do formátu popísanom atribútom priestoru pozorovania 4.3.

Reset: Volanie tejto funkcie, väčšinou po ukončení hry, respektíve dokončenie epizódy, uvedie prostredie do začiatočného stavu a znázorňuje inicializáciu novej hry. Táto funkcia vracia len jednu návratovú hodnotu a to je pozorovanie (observation), ktorá musí byť, tak ako pri návratových hodnotách funkcie *Step*, podľa formátu definovanom atribútom priestoru pozorovania. Táto hodnota sa použije na vykonanie prvého kroku. Funkcia nevracia odmenu, pretože agent ešte nevykonal žiadnu akciu.

Render: Implementácia tejto funkcie je voliteľná. Volá sa pri vykresľovaní prostredia, ktorá je užitočná napríklad pri hľadaní chýb človekom. Môže mať viacero módov, definovaných v atribúte metadata (metadáta) danej triedy.

Priestory

Každé trénovanie začína náhodnými akciami. Priestory slúžia na popísanie týchto vecí aby bolo možné ich využívať v rôznych generických prostrediach, teda zabezpečujú normalizáciu rozhrania prostredí. Sú typu *Space* a popisujú formát validných akcií a pozorovaní. Je možné vyberať z rôznych typov alebo vytvoriť vlastný priestor [19].

Ja som zvolil *Box* pre pozorovanie a *Discrete* pre akcie. Box je definovaný ako n dimenzionálne pole, v mojom prípade sú to 3 dimenzie (počet kanálov, výška prostredia, šírka prostredia). Discrete znázorňuje diskrétny priestor, čiže sa jedná o zoznam s veľkosťou n (počet akcií) [19].

GymWrapper

GymWrapper je zabalenie prostredia do štandardov knižnice Gym. Korektnosť implementácie zabalenia (wrapperu) sa dá skontrolovať za pomoci funkcie **check_env** z knižnice stable baselines.

GymWrapper v prostredí Bomberland bol čiastočne pripravený v Starter kite od CodeROne. Jedná sa o funkcie na komunikáciu so serverom. Funkcie na komunikáciu s knižnicou stable baselines 3 som vytváral ja za pomoci dokumentácie ku knižnici stable baselines 3 [1].

4.3 Reprezentácia herného prostredia

Prostredie nemôžeme jednoducho posielať do neurónovej siete v akomkoľvek formáte. Je ho potrebné upraviť.

Úprava prostredia pre vstup do neurónovej siete

Rozhodol som sa prostredie upraviť do podoby šedých (gray-scale) obrázkov. To znamená, že každý kanál o rozmere hracieho pola bude mať hodnoty od 0-254, ktoré budú znázorňovať normalizované hodnoty rôznych vlastností.

Rozhodol som sa dať maximálnu hodnotu 254, pretože knižnica zo stable baselines predpokladala, že keď je maximálna hodnota 255 tak sa jedná o obrázok s 3 kanálmi (RGB) a snažila sa zmeniť tvar tohto vstupu. S tým by nebol problém, avšak funkcia predpokladá, že kanál bude najmenšia dimenzia vo vstupe. Vstup je ale veľkosti 24x15x15 a práve toto robilo problémy, keď sa funkcia snažila zmeniť tvar vstupu na 15x24x15.

Všetky hodnoty sú normalizované. Normalizoval som ich tak, že som si stanovil nejakú maximálnu hodnotu, ktorú môžu vlastnosti dosahovať. A teda hodnoty vlastností sú potom vydelené stanovenou maximálnou hodnotou daného typu a následne vynásobené hodnotou 254.

Kanály jednotiek

Prvých desať kanálov je určených pre vlastnosti jednotiek.

Prvý kanál zobrazuje jednotku, pre ktorú má agent aktuálne vybrať akciu.

Kanály dva až štyri pozostávajú zo zobrazenia všetkých priateľských jednotiek (spolu s tou ovládanou) a v kanáloch päť až deväť zase len nepriateľské jednotky. Hodnoty reprezentujúce vlastnosti (v uvedenom poradí) sú životy, sila bômb jednotiek, počet bômb k dispozícii danej jednotke a nesmrteľnosť. Tieto hodnoty sú umiestnené na pozícii postavy, ktorej táto hodnota patrí.

Desiaty kanál zobrazuje mŕtve jednotky. Je to dôležité z toho dôvodu, že sa cez tieto jednotky nedá prechádzať.

Grafické zobrazenie je na obrázku 4.2.

Kanály kociek

Nasledujúcich deväť kanálov je určených pre kocky.

Prvé tri z týchto kanálov popisujú drevené kocky, kovové kocky a orné kocky.

Drevené a kovové kocky sú reprezentované len ako jednotky (resp. 254, maximálna hodnota). Drevené kocky síce sú rozbitné, no majú iba jeden život. To znamená, že sa buď v hracom poli nachádzajú alebo nie, môžeme to klasifikovať binárne, teda jednotkou alebo nulou. Kovové kocky sa naopak rozbiť nedajú, takže budú mať taktiež konštantnú hodnotu jedna.

Orné kocky sú rozbitné a tak majú normalizovanú hodnotu reprezentujúcu životy jednotlivých kociek. Pôvodne som rozmýšľal nad oddelením orných kociek podľa počtu životov do jednotlivých kanálov. Neskôr som však od toho upustil, kanály by boli väčšinou prázdne a tak sa domnievam, že by to bol zbytočný šum pre neurónovú sieť a tak som sa držal len tejto reprezentácie.

Ďalší kanál v poradí označuje pozície munície. Hodnota týchto kociek označuje dobu expirácie. To znamená, že kocka zmizne a nebude sa dať zobrať.

Bomby sú popísané nasledujúcimi dvoma kanálmi. Sú na pozícií kde bomba leží a majú hodnoty expirácie a sily výbuchu.

Nasledujú vylepšenia výbuchu bomby a taktiež, ako munícia, reprezentujú hodnoty dobu expirácie.

Posledné dva v tejto skupine sú kocky typu oheň. V prvom z týchto dvoch kanálov je hodnota expirácie. Toto je kanál, kde sa objaví oheň z výbuchu bomby a reprezentuje hodnotu, za ktorú zmizne. Druhý kanál zase popisuje len konštantnou jednotkou všetky pozície ohňa. Tu už je aj konečný oheň, ktorý expiráciu nemá. Spojil som ohne z bômb a z konca hry, aby model vedel, že v podstate je to rovnaký typ kocky a týmto políčkam sa má vyhýbať.

Grafické zobrazenie je na obrázku 4.3.

Pomocné kanály

Posledných päť kanálov pozostáva z pomocných funkcií.

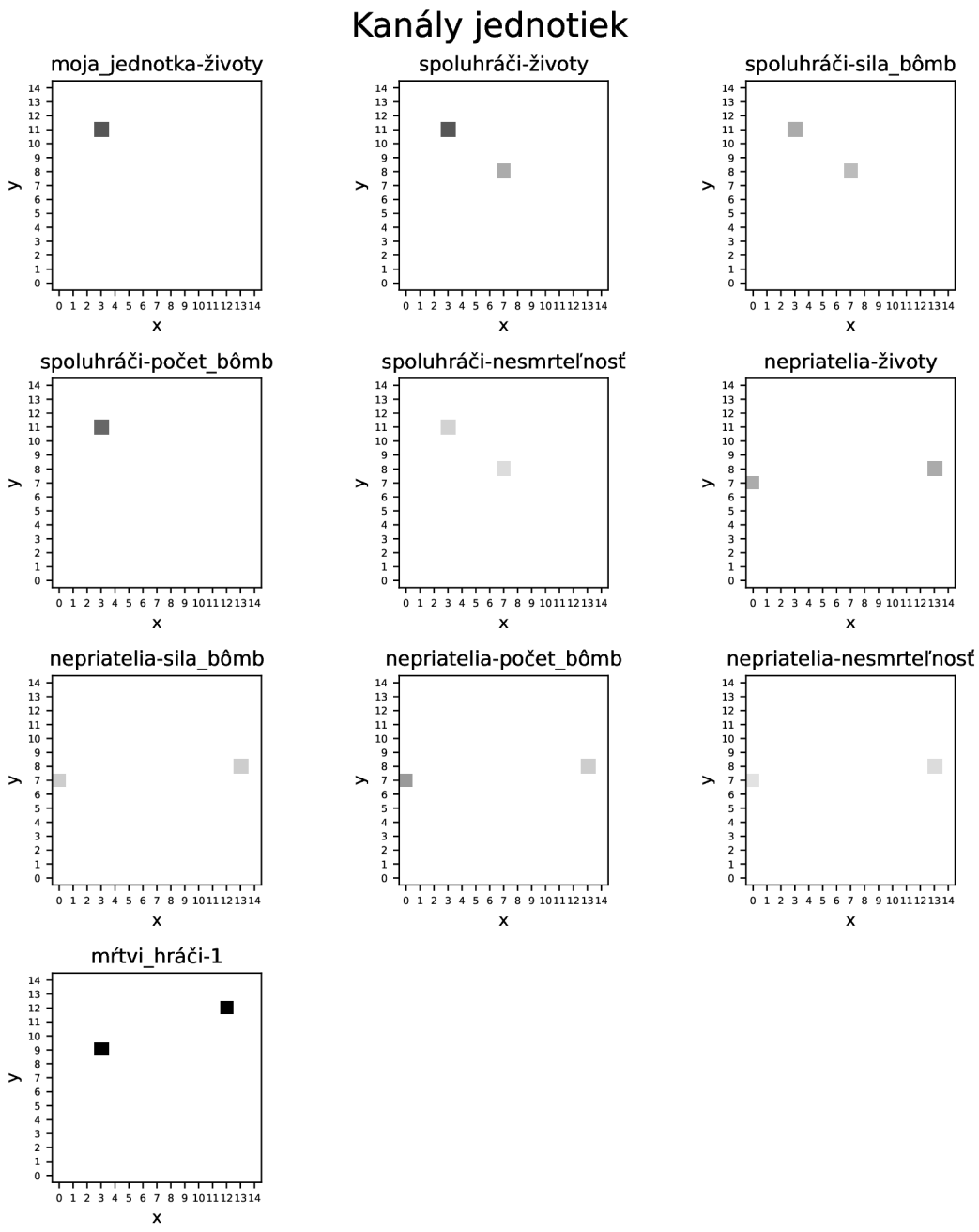
Prvú z nich som nazval výhoda pozície a znázorňuje zjednodušený model pre hracie pole a ktoré kocky na ňom by mala chcieť jednotka obsadiť. Pôvodne bol tento model použitý ako jednoduchá odmeňovacia funkcia.

Kanály dva a tri popisujú lokálnu pozíciu na každej z osí. Mali by slúžiť ku navádzaniu agenta pri tom, kde sa v hracom poli nachádza. V podstate by sa podobne dal použiť aj kanál výhody pozície.

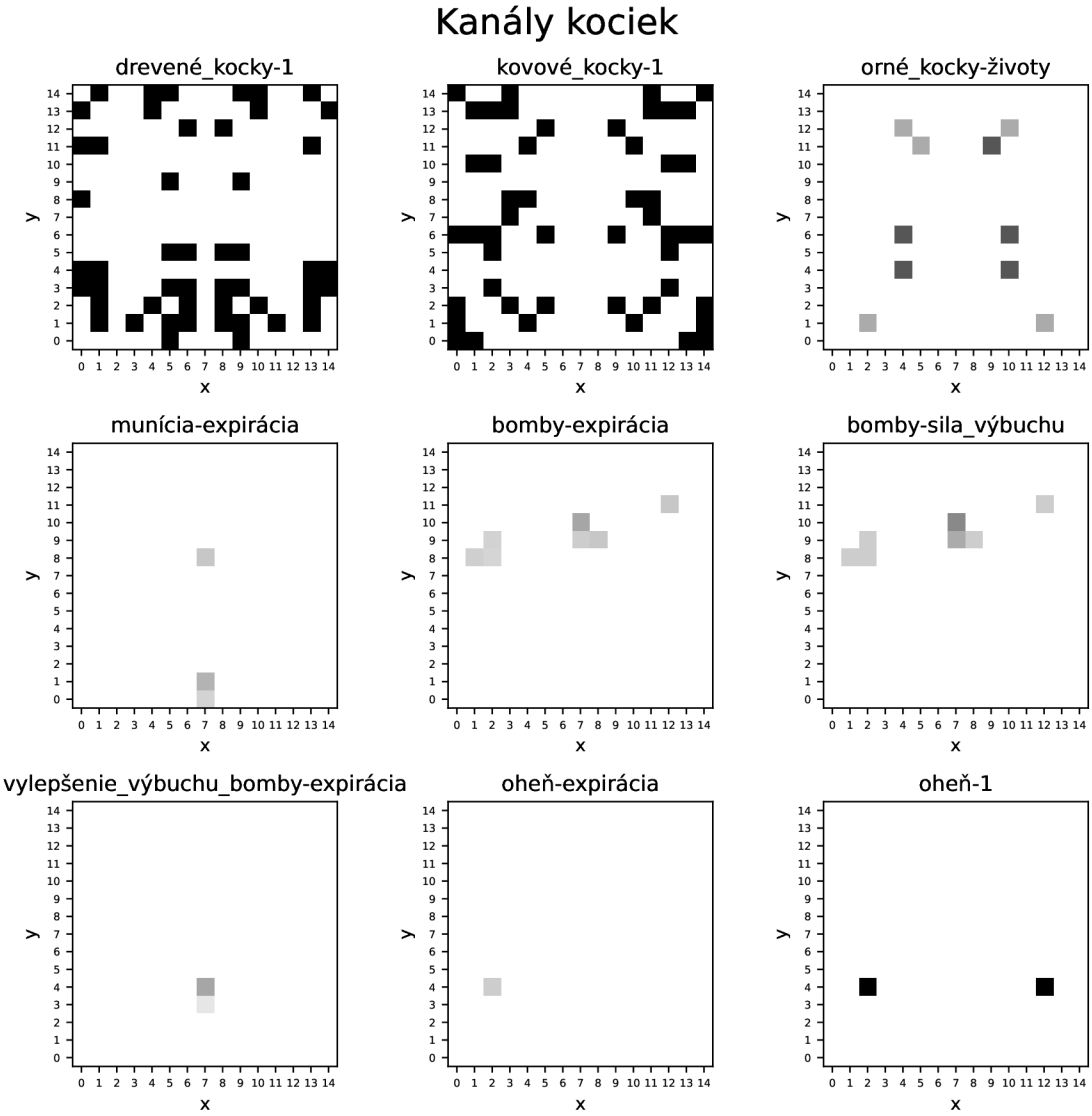
Predposledný kanál označuje, kedy príde konečný oheň do hracieho poľa. Ide v špirálovitej forme z ľavého horného a spodného pravého rohu, na striedanie, každé dva herné kroky.

A posledný kanál jednoducho popisuje, ktorý herný krok je aktuálny. Deliteľ pre normalizáciu ju vypočítaný ako maximálna dĺžka hry. Dá sa vypočítať z herných nastavení, takže zakaždým agent vie, v ktorej fáze hry sa práve nachádza. Avšak v kratších hrách budú hodnoty v tomto kanály rásť rýchlejšie a v dlhších zase pomalšie.

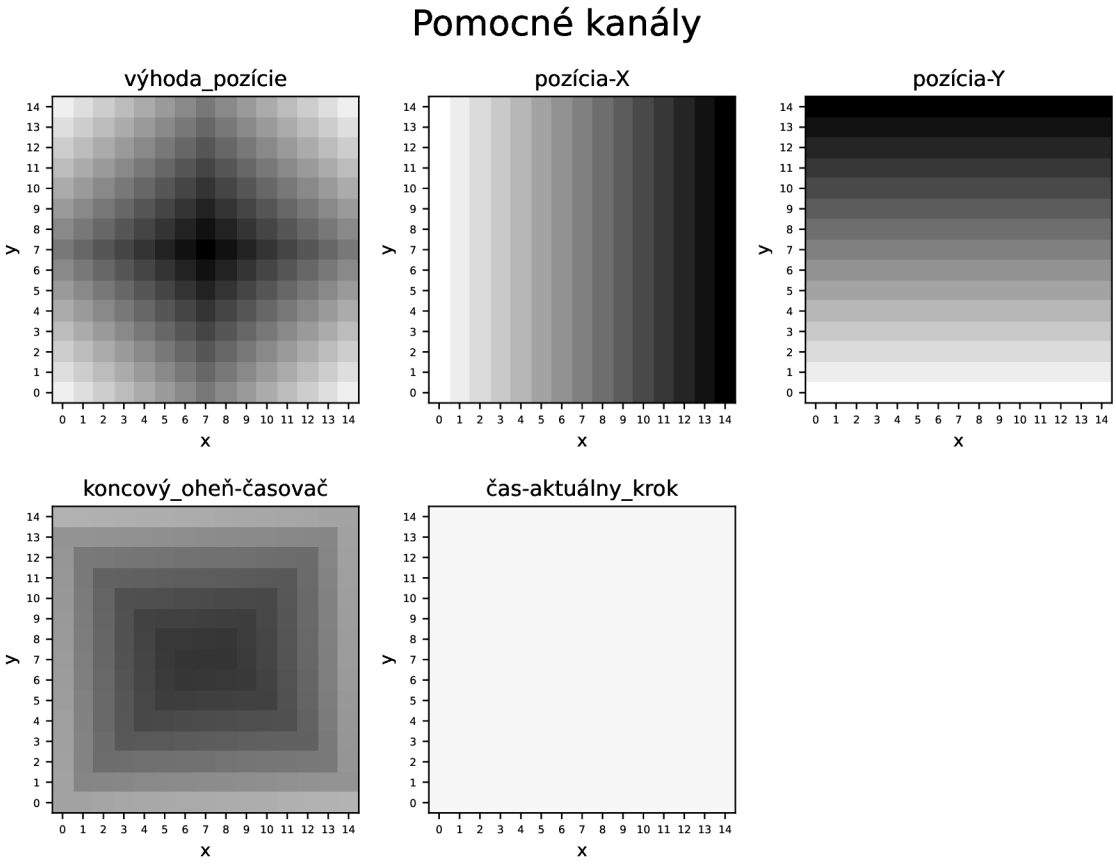
Grafické zobrazenie je na obrázku 4.4.



Obr. 4.2: **Kanály jednotiek.** Slúžia ako vstup pre neurónovú sieť.



Obr. 4.3: **Kanály kociek.**



Obr. 4.4: **Pomocné kanály.**

Spracovanie výstupu z neurónovej siete

Výstup z neurónovej siete je spracovaný funkciami knižnice použitého algoritmu, teda stable baselines. Z funkcie model.predict(obs) následne vyjde už len číselný index akcie v poli akcií. Táto hodnota musí byť upravená pred poslaním na herný server na spracovanie. Robím to vkladaním potrebných údajov do JSON šablón, ktoré sú formátované do dátovej štruktúry slovník. Všetky tieto slovníky nakoniec vložím do dátovej štruktúry list.

```
[
    {
        "action": {
            "type": "move",
            "unit_id": unit_id,
            "move": u_action
        },
        "agent_id": agent_id
    },
    {
        "action": {
            "type": "bomb",
            "unit_id": unit_id
        },
        "agent_id": agent_id
    },
    {
        "action": {
            "type": "detonate",
            "unit_id": unit_id,
            "coordinates": bomb_coords
        },
        "agent_id": agent_id
    }
]
```

Akčný priestor som navrhol tak, aby agent mohol vykonávať nasledujúce akcie: Pohyb vpravo, pohyb vľavo, pohyb hore, pohyb dole, položenie bomby, odpálenie 3. bomby od konca, odpálenie 2. bomby od konca, odpálenie 1. bomby od konca a stáť na mieste. Napadlo mi zakomponovať až tri možnosti odpálenia bomby, pretože je tam možnosť výberu, ktorú bombu chce agent odpáliť. V originálnom starter kite sa nachádzala možnosť odpáliť len poslednú položenú bombu, to však nemusí byť práve tá bomba, ktorú by agent chcel odpáliť.

Kapitola 5

Príprava a trénovanie agenta

Po vybraní frameworku stable baselines 3 a vybraní algoritmov, ktoré chcem použiť v mojom agentovi, bolo implementovanie prípravy a vytvorenie agenta relatívne priamočiare. Po inicializácií agenta s vybraným algoritmom z ponuky knižnice a vybranou politikou algoritmu, bol agent ihneď pripravený na učenie.

To ale neplatilo pri prostredí, ktoré pozostávalo z asynchrónnych funkcií pythonu. Tie sa totižto môžu volať len z ďalších asynchrónnych funkcií. Po dlhom skúmaní možností, som uvážil, že bude najlepšie využiť funkcie z knižnice asyncio, ktorá umožňuje vytvoriť cyklus pre asynchrónne funkcie, aby sa navonok správali ako synchrónne. Je to dôležité z dôvodu, že funkcie modelu agenta sú všetky synchrónne a nevedia zabaliť volanie asynchrónnych funkcií gymu do funkcií asyncio. Toto riešenie pravdepodobne nie je ideálne avšak v čase riešenia som na lepší spôsob neprišiel. Po zabalení asynchrónnych funkcií bolo aj prostredie pripravené na komunikáciu s agentom.

5.1 Proces trénovania

Začatie trénovania bolo taktiež relatívne jednoduché. Prebieha na základe volania funkcie learn. Táto funkcia následne vykonáva všetky potrebné kroky ku učeniu modelu. Volá funkcie step a reset z priradeného prostredia. Pri čítaní rôznych fór (zdroj chýba) som sa dočítal, že prednastavené hyperparametre by mali fungovať v podstate na akomkoľvek prostredí, avšak ich bližšie nastavenie by malo zvýšiť výkon a efektivitu agenta. Takže som použil tieto nastavenia a začal trénovanie.

1. iterácia trénovania

- MLP politika
- PPO algoritmus
- žiadne nastavené hyperparametre
- sparse odmeňovacia funkcia: len na konci hry
- fixné pôvodné prostredie
- bez pohybu nepriateľov
- kanály: bez pomocných kanálov 4.4

Na začiatku som bol veľmi optimistický a tak som chcel vyskúšať riedku (sparse) metódu odmeňovania. Je postavená na odmeňovaní agenta až na konci hry, podľa toho, či vyhrá

alebo prehrá. V teórií je toto najlepší spôsob, ako nájsť ideálnu hernú stratégiu, pokiaľ nechceme agenta nijak ovplyvniť ľudským pozorovaním prostredia. Ako som ale zistil, je taktiež veľmi pomalý, pretože agent hrá naslepo a dostane odmenu až po (v priemere) 350+ herných krokov. Taktiež je veľmi ťažké pre agenta usúdiť, ktoré kroky presne, ho viedli ku víťazstvu. Odmeňovacia funkcia vyzerala nejak takto:

```
reward = (100*n_my_units + 10*sum_my_hp)
punishment = (100*n_enemy_units + 10*sum_enemy_hp)

if done:
    return (reward - punishment)
return 0
```

Model sa za 2 dni vôbec nič nenaučil (predvídateľne), preto som sa rozhodol túto iteráciu ukončiť.

2. iterácia trénovania

- MLP politika
- PPO algoritmus
- žiadne nastavené hyperparametre
- dense odmeňovacia funkcia: každý krok + discount
- fixné pôvodné prostredie
- bez pohybu nepriateľov
- kanály: bez pomocných kanálov 4.4

Druhá iterácia pozostávala z rovnakých nastavení až na fakt, že odmenu dostával agent po každom kroku. Odmeňovacia funkcia sa teda trochu upravila. Taktiež mi napadlo rovno implementovať zľavu na odmenu:

```
reward = (100*n_my_units + 10*sum_my_hp)
punishment = (100*n_enemy_units + 10*sum_enemy_hp)

to_return = ((reward - punishment) - last_reward)
last_reward = (reward - punishment)
if done:
    to_return = to_return * (MAX_GAME_LENGTH/current_tick+1)
return to_return
```

Takže každý krok dostane rozdiel odmeny aktuálnej od predchádzajúcej a to preto, aby nedostával odmenu stále, keď je vo výhode. Dostane ju len v momente, kedy túto výhodu nadobudne. Ak je hra ukončená, je vynásobená maximálnou dĺžkou, ktorá je vydelená aktuálnym časovým krokom. Teda ak agent prehrá veľmi skoro v hre, je veľmi veľká negatívna odmena ale ak veľmi skoro vyhrá, je odmena zase veľmi veľká v pozitívnych číslach.

3. až 6. iterácia

- MLP politika
- PPO algoritmus
- žiadne nastavené hyperparametre
- dense odmeňovacia funkcia: každý krok + discount + odmena za pozíciu
- zjednodušené prostredie: málo až žiadne prekážky
- bez pohybu nepriateľov (1v1, 1v3): postavený nepriateľ do stredu
- kanály: bez pomocných kanálov 4.4
- náhodná začiatočná pozícia

Niekde medzi týmito iteráciami som prišiel na to, že môžem zaznamenávať logovanie v TensorBoarde. A teda mať aj lepší prehľad o tom, ako sa trénovanie vyvíja. Doteraz bolo trénovanie odvodené z testovania a vypisovaním vykonaných pohybov.

Jedny z najvýznamnejších zmien v týchto iteráciách boli úpravy na odmeňovacej funkcii a prostredí. Taktiež som pridal aj klipovanie odmeny, čo si však nemyslím, že malo akýkoľvek vplyv na trénovanie. Je však vhodné to ponechať z dôvodu normalizácie.

Podstatnejšia zmena odmeňovacej funkcie spočívala v odmeňovaní agenta, ak sa približoval bližšie ku stredu. Je vypočítaná manhattanskou vzdialenosťou, kde hodnota aktuálnej pozície $H = |x_1 - x_2| + |y_1 - y_2|$ je definovaná ako súčet absolútnych hodnôt rozdielov x a y súradníc pozície od x a y súradníc stredu.

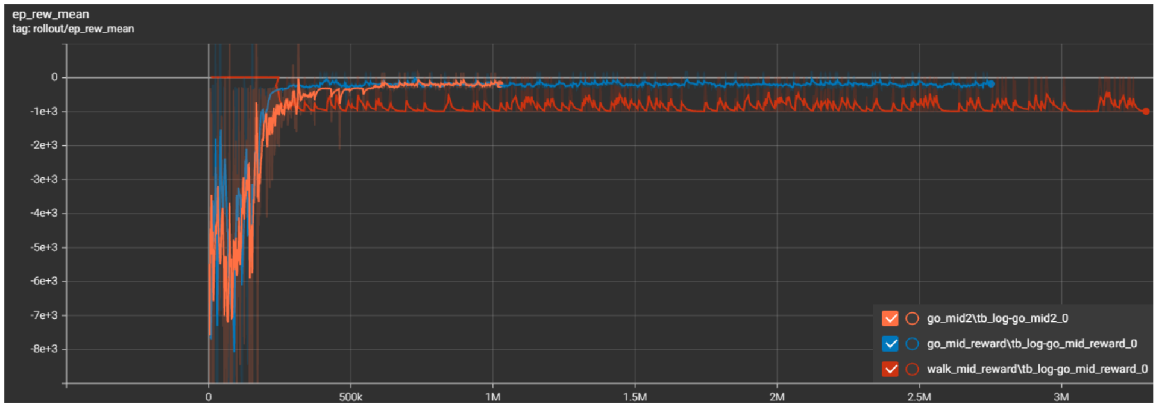
Zmena prostredia spočívala v jeho zjednodušení, odstránil som všetky (alebo skoro všetky) prekážky a skúšal trénovať týmto štýlom.

Nová funkcia teda vyzerala približne takto:

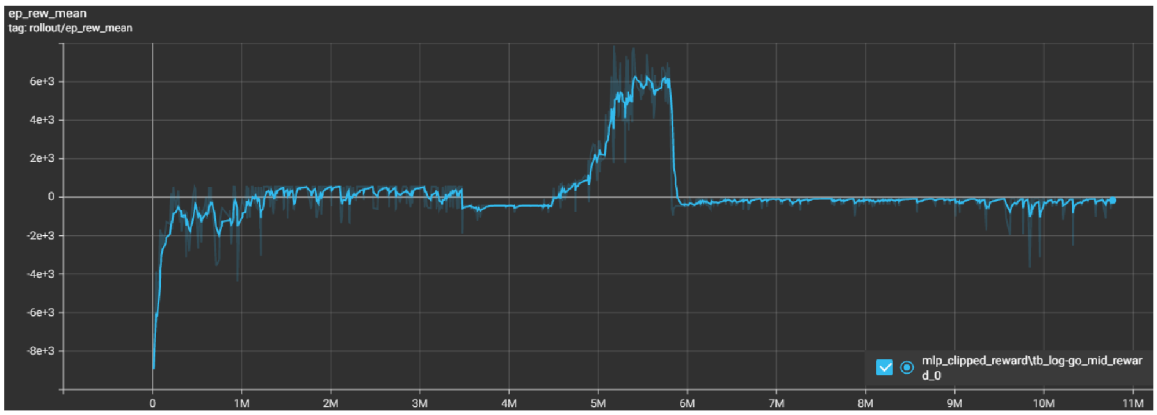
```
position_reward = position_advantage[my_y][my_x]

reward = (100*n_my_units + 10*sum_my_hp)
punishment = (100*n_enemy_units + 10*sum_enemy_hp)

to_return = ((reward - punishment) - last_reward) + position_reward
last_reward = (reward - punishment) + position_reward

if done:
    to_return = to_return * (MAX_GAME_LENGTH/current_tick+1)
return to_return
```

Teda to isté ako naposledy ale aj s pričítanou výhodou pozície. Agent sa naučil v prázdnom poli chodiť do stredu, to mi však nestačilo a tak som pridal nepriateľskú jednotku do stredu poľa nech donútim môjho agenta pokladať bomby, inak bude stále dookola prehrávať. Naučil sa pokladať bombu no iba na to isté miesto, aj keď bolo iné voľné miesto bližšie. Podľa mňa sa tento agent pretrénoval a keď som vložil na toto miesto kocku, nedokázal nájsť cestu okolo prekážky. Podľa záznamov z tensorboardu sa traja zo štyroch agentov trénovali dokopy 31 hodín (nie paralelne). Posledný, opisovaný model, sa učil okolo 87 hodín, teda 3 dni a 15 hodín, za ktoré dosiahol 10.77 miliónov herných krokov.

Všetky zobrazené grafy z tensorboardu majú na osi Y priemernú odmenu za epizódu a na osi X počet herných krokov (pokiaľ nie je napísané pod grafom ináč). Grafy nemajú rovnaké herné nastavenia, takže ich porovnávaním nedostaneme správne porovnanie výsledkov. Každý graf by mal byť braný len v skupine, v ktorej sa nachádza, respektíve porovnávať len modely vrámci jedného grafu. Ak je nejaký nezmyselný skok v grafe, je to pravdepodobne zapríčinené tým, že som začal trénovanie a potom si všimol niekde chybu, poprípade som niečo chcel vyskúšať s inými nastaveniami a nechcel som začínať trénovanie od začiatku.



Obr. 5.1: **Prvé 3 nahrané MLP modely.** Ako je vidno na týchto grafoch, všetky modely sa prestali učiť medzi 250 až 500 tisíc hernými krokmi.



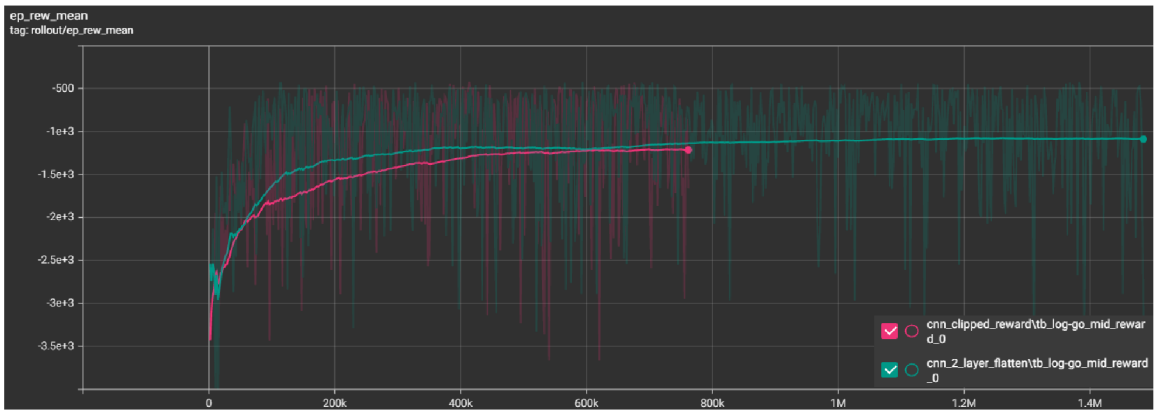
Obr. 5.2: **Posledný nahraný MLP model.** Asi jediný model, ktorý dokázal položiť a odpáliť bombu pri nepriateľskej jednotke. Nanešťastie, nedokázal toto zovšeobecniť a obísť prekážku.

7. iterácia trénovania

- CNN politika
- PPO algoritmus
- žiadne nastavené hyperparametre
- dense odmeňovacia funkcia: každý krok + discount + odmena za pozíciu
- zjednodušené prostredie: málo až žiadne prekážky

- bez pohybu nepriateľov (1v1, 1v3): postavený nepriateľ do stredu
- kanály: bez pomocných kanálov 4.4
- náhodná začiatočná pozícia

CNN však nebolo možné spustiť tak jednoducho ako MLP. Bolo potrebné navrhnúť vlastnú neurónovú sieť na vyťahovanie vlastností (feature extractor), pretože vstavaná CNN vyžadovala najmenší možný rozmer 36x36. Prvú navrhnutú neurónovú sieť som si nanešťastie neuložil a nepamätám si, ako vyzerala. Mám ju však uloženú v tensorboarde a na grafe je vidno, že sa prestali zvyšovať odmeny okolo rozmedzia 200 až 400 tisíc herných krokov.



Obr. 5.3: **Prvé dva CNN modely.** Ako je vidno na grafe tieto modely nedospeli ani ku jednej výhre (všetky, aj nespriemerované, dáta sú pod 0). Tu som si uvedomil, že asi nie je najlepší nápad dávať statického nepriateľa presne do stredu. Agent by totižto musel vykonať veľmi špecifické ťahy aby sa mu podarilo zvíťaziť.

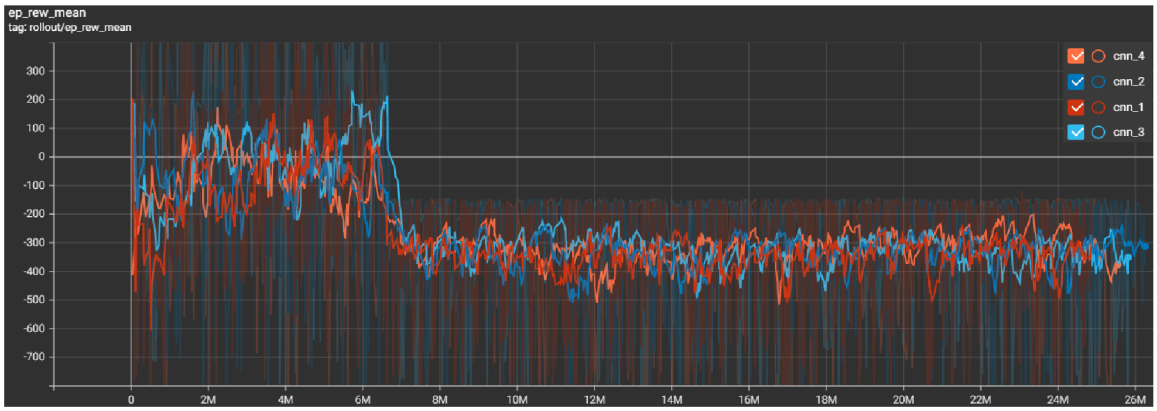
8. iterácia trénovania

- CNN politika: 4 nové neurónové siete na vyťahovanie vlastností
- PPO algoritmus
- žiadne nastavené hyperparametre
- dense odmeňovacia funkcia: každý krok + discount + odmena za pozíciu
- zjednodušené prostredie: 40 prekážok, náhodné generovanie prostredia
- bez pohybu nepriateľov (3v3): postavený nepriateľ do stredu, spoluhráči sa nehýbu
- kanály: pridaný kanál na výhodu pozície
- náhodná začiatočná pozícia

Tu začína časť, kedy som začal trénovať viac modelov paralelne. Každý proces bežal v osobitnom docker kontajneri takže by sa nemali ovplyvňovať, čo sa týka dosiahnutých výsledkov. Jediné v čom sa ovplyvňovali bol výkon.

Tu sú popísané neurónové siete na vyťahovanie vlastností (feature extractors). Každý proces mal aktivovanú inú neurónovú sieť. Prvé tri využívajú aj AdaptiveAvgPooling na rozmer 1x1 ako poslednú vrstvu (pred vyhladením do jednej dimenzie). Posledný je bez adaptive poolingu.

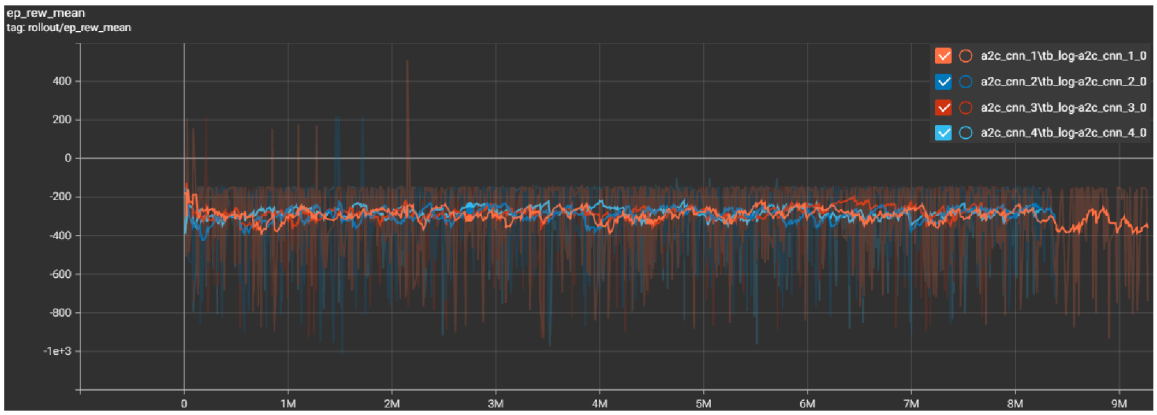
```
self.nn_options = {
    1: nn.Sequential(
        nn.Conv2d(n_input_channels, 16, kernel_size=3, padding=1, stride=1),
        nn.ReLU(),
        nn.Conv2d(16, 16, kernel_size=3, padding=1, stride=1),
        nn.ReLU(),
        nn.Conv2d(16, 32, kernel_size=3, padding=1, stride=1),
        nn.ReLU(),
        nn.AdaptiveAvgPool2d(1),
        nn.Flatten()),

    2: nn.Sequential(
        nn.Conv2d(n_input_channels, 16, kernel_size=3, padding=1, stride=1),
        nn.ReLU(),
        nn.Conv2d(16, 32, kernel_size=3, padding=1, stride=2),
        nn.ReLU(),
        nn.Conv2d(32, 32, kernel_size=3, padding=1, stride=1),
        nn.ReLU(),
        nn.AdaptiveAvgPool2d(1),
        nn.Flatten()),

    3: nn.Sequential(
        nn.Conv2d(n_input_channels, 16, kernel_size=3, padding=1, stride=1),
        nn.ReLU(),
        nn.Conv2d(16, 32, kernel_size=3, padding=1, stride=2),
        nn.ReLU(),
        nn.Conv2d(32, 64, kernel_size=3, padding=1, stride=2),
        nn.ReLU(),
        nn.AdaptiveAvgPool2d(1),
        nn.Flatten()),

    4: nn.Sequential(
        nn.Conv2d(n_input_channels, 16, kernel_size=3, padding=1, stride=1),
        nn.ReLU(),
        nn.Conv2d(16, 32, kernel_size=3, padding=1, stride=2),
        nn.ReLU(),
        nn.Conv2d(32, 64, kernel_size=3, padding=1, stride=2),
        nn.ReLU(),
        nn.Flatten()),
}
```



Obr. 5.4: **Originálny tréning CNN sietí.** Okolo kroku 6 a pol milióna bolo zmenené nastavenie, na znovu položenie nehýbajúceho sa nepriateľa do stredu hracieho pola. Predpokladal som, že tým vyriešim aspoň to, že sa agenti hýbu stále len do jedného smeru, alebo len stoja na mieste (nie kombinácia týchto dvoch).

9. iterácia trénovania

- CNN politika: 4 neurónové siete na vyťahovanie vlastností
- A2C algoritmus
- žiadne nastavené hyperparametre
- dense odmeňovacia funkcia: každý krok + discount + odmena za pozíciu
- zjednodušené prostredie: 40 prekážok, náhodné generovanie prostredia
- bez pohybu nepriateľov (3v3): postavený nepriateľ do stredu, spoluhráči sa nehýbu
- kanály: pridaný kanál na výhodu pozície
- náhodná začiatočná pozícia

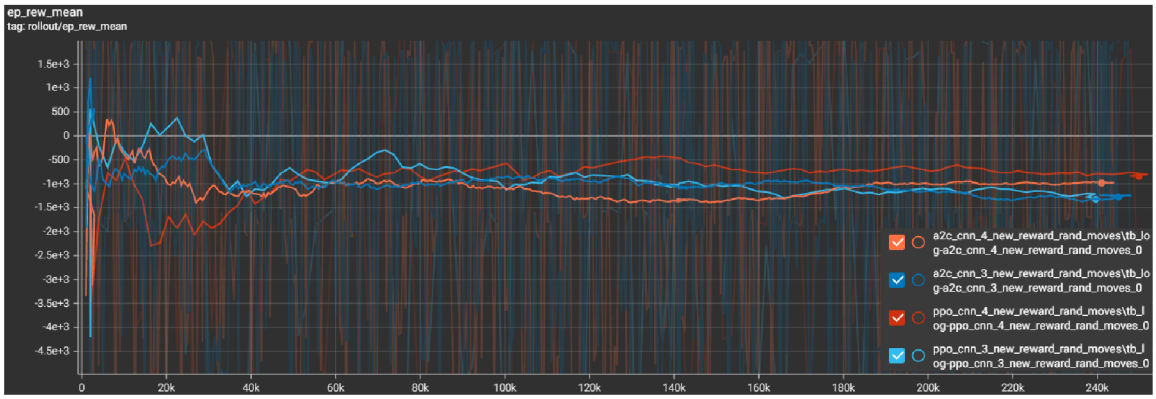


Obr. 5.5: **Prvé trénovanie s A2C, stále na CNN.** V podstate sa jedná o identické výsledky ako v predchádzajúcej iterácii s PPO. Po neúspešnom trénovaní na 24 miliónoch krokoch som začal používať algoritmus A2C. Po 10 miliónoch som trénovanie ukončil s rovnakým výsledkom.

10. iterácia trénovania

- CNN politika: neurónové siete 3 a 4 na vyťahovanie vlastností 5.1
- PPO a A2C algoritmus
- žiadne nastavené hyperparametre
- dense odmeňovacia funkcia: každý krok + discount
- zjednodušené prostredie: 40 prekážok, náhodné generovanie prostredia
- náhodný pohyb nepriateľov (3v3): spoluhráči sa nehýbu
- kanály: pridaný kanál na výhodu pozície
- náhodná začiatočná pozícia

Keďže sa trénovanie nikam nehýbalo, pokúsil som sa naštartovať náhodný pohyb nepriateľských jednotiek. Jednotky mali definované aby sa 40% času hýbali a 60% času stáli. Dôvodom je, že som chcel aby chvíľu stáli aby bolo ľahšie ich prípadne zasiahnuť bombou. Taktiež som tu odstránil odmenu za pozíciu.



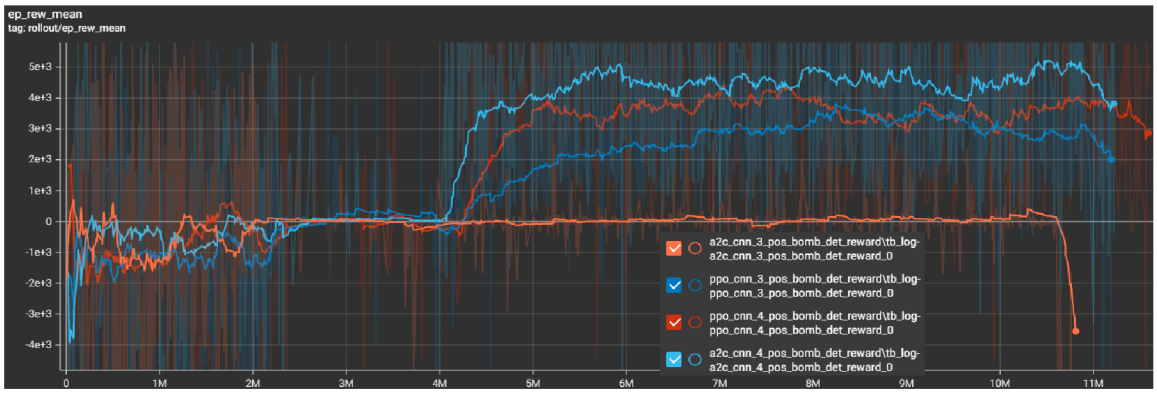
Obr. 5.6: **Prvé trénovanie s náhodným pohybom nepriateľov.** Z predchádzajúcich trénovaní som odhadol, že by sa tento model aj tak už po 250 tisíc ťahoch nezačal nič učiť tak som prešiel na ďalší experiment.

Trénovanie trvalo len okolo 250 tisíc krokov, keďže som nevidel žiadnu zmenu a na základe predchádzajúcich trénovaní som videl, že sa to zhruba okolo tohto kroku už moc nevyvíjalo, skúsil som niečo iné. Stále sa model snaží vykonávať rovnaké akcie a to buď opakovane pohyb do jednej strany alebo nevykonávať žiadnu akciu. Je to zvláštne, pretože nemali žiadnu penalizáciu za pohyb a to ani zlý. Je jasné, že sa naučia nedávať bomby, pretože ich zrejme väčšinou trafili a hra sa skončila. Je možné, že nikoho nimi niekoľko kôl netrafili, trafili seba a tak od nich navždy upustili.

11. iterácia trénovania

- CNN politika: neurónové siete 3 a 4 na vyťahovanie vlastností 5.1
- PPO a A2C algoritmus
- žiadne nastavené hyperparametre

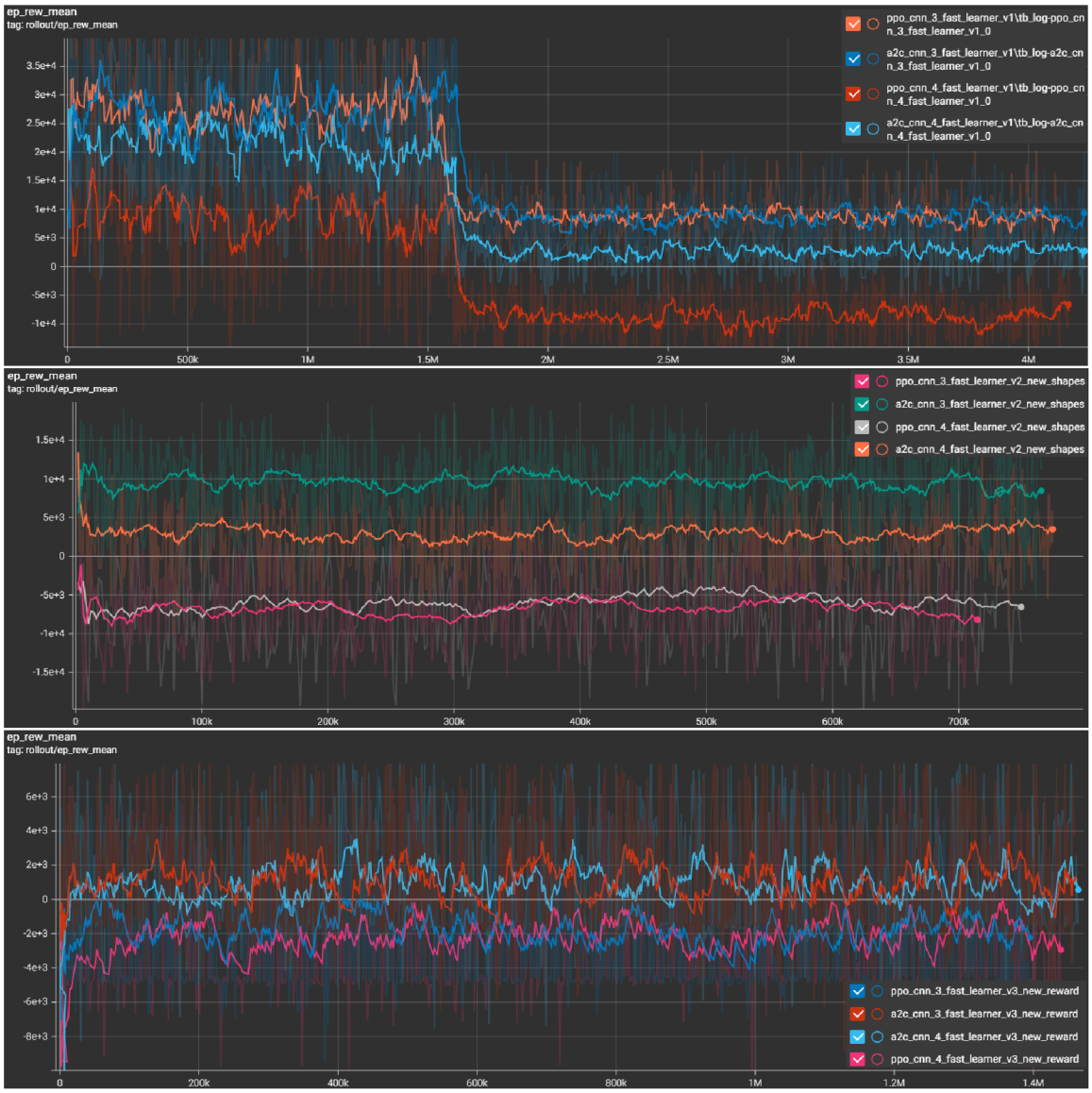
- dense odmeňovacia funkcia: každý krok + discount + odmena za položenie bomby a úspešnú detonáciu a dostáva vyššiu odmenu ak zoberie život, neráta sa počet jednotiek počas trvania hry, ráta sa až na konci, pridaná odmena pozície
- zjednodušené prostredie: 40 prekážok, náhodné generovanie prostredia
- náhodný pohyb nepriateľov (3v3): spoluhráči sa nehýbu
- kanály: pridaný kanál na výhodu pozície + X a Y pozícia
- náhodná začiatočná pozícia



Obr. 5.7: **Implementovaná odmena za položenie bomby a detonáciu bomby.** Modely sa zase bohužiaľ nenaučili pokladať bomby a vlastne ani vyhrávať hru, aj keď to vyzerá podľa odmien ináč. Model A2C_cnn_4 (tyrkysová) sa naučil stáť na mieste a dosiahol tým winrate 39% na 560 hrách. Model PPO_cnn_4 (červená) sa naučil chodiť stále iba doprava (s pár výchylkami náhodných akcií) a dosiahol 28% winrate na 550 hrách. Testovanie prebieha trochu ináč ako trénovanie v tom, že všetci traja agenti využívajú natrénovanú policy, pri trénovaní vykonáva akcie iba jeden agent.

12. až 14. iterácia trénovania

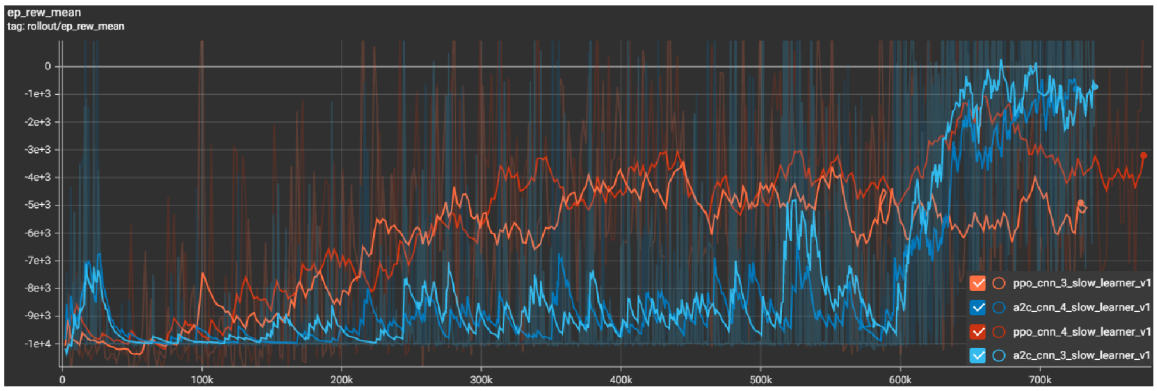
- CNN politika: neurónové siete 3 a 4 na vyťahovanie vlastností 5.1
- PPO a A2C algoritmus
- od 12. iterácie: Zvýšený learning rate (PPO: 0.0003 -> 0.01, A2C: 0.0007 -> 0.01)
- dense odmeňovacia funkcia: od 14. iterácie upravené hodnoty a opravená chyba v trénovaní, odpočítavanie odmien za vybranie zlej akcie
- zjednodušené prostredie: 40 prekážok, náhodné generovanie prostredia
- náhodný pohyb nepriateľov (3v3): spoluhráči sa nehýbu
- od 13. iterácie všetky pomocné kanály (vzorec koncového ohňa a aktuálny čas hry)
- náhodná začiatočná pozícia



Obr. 5.8: **Experiment so zvýšeným learning rate.**

15. iterácia trénovania

- CNN politika: neurónové siete 3 a 4 na vyťahovanie vlastností 5.1
- PPO a A2C algoritmus
- Znížený learning rate (PPO: 0.0003 -> 0.00005, A2C: 0.0007 -> 0.00005)
- dense odmeňovacia funkcia: upravené hodnoty a opravená chyba v trénovaní, odpočítavanie odmien za vybranie zlej akcie
- zjednodušené prostredie: 40 prekážok, náhodné generovanie prostredia
- náhodný pohyb nepriateľov (3v3): spoluhráči sa nehýbu
- všetky pomocné kanály (vzorec koncového ohňa a aktuálny čas hry)
- náhodná začiatočná pozícia



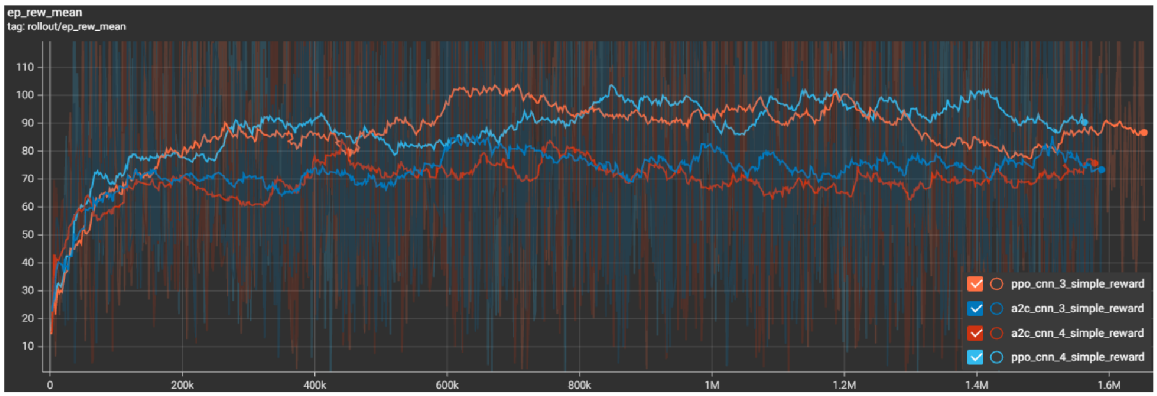
Obr. 5.9: **Experiment so zníženým learning rate.**

16. a 17. iterácia trénovania

- CNN politika: neurónové siete 3 a 4 na vyťahovanie vlastností 5.1
- PPO a A2C algoritmus
- 2. iterácia: Zvýšený learning rate (PPO: 0.0003 -> 0.001, A2C: 0.0007 -> 0.001)
- dense odmeňovacia funkcia: úplne zjednodušená odmena za pohyb
- zjednodušené prostredie: 40 prekážok, náhodné generovanie prostredia
- náhodný pohyb nepriateľov (3v3), spoluhráči sa nehýbu
- všetky pomocné kanály (vzorec koncového ohňa a aktuálny čas hry)
- náhodná začiatočná pozícia

Posledný experiment. Chcel som ním zistiť, či je celý čas chyba v mojej odmeňovacej funkcií. Takže v 16. iterácii som spravil úplne novú odmeňovaciu funkciu, ktorá sa podobala tej na absolútnom začiatku trénovania. V podstate spočíva v tom, že ak sa agent hýbe (a teda aj žije) dostáva odmenu.

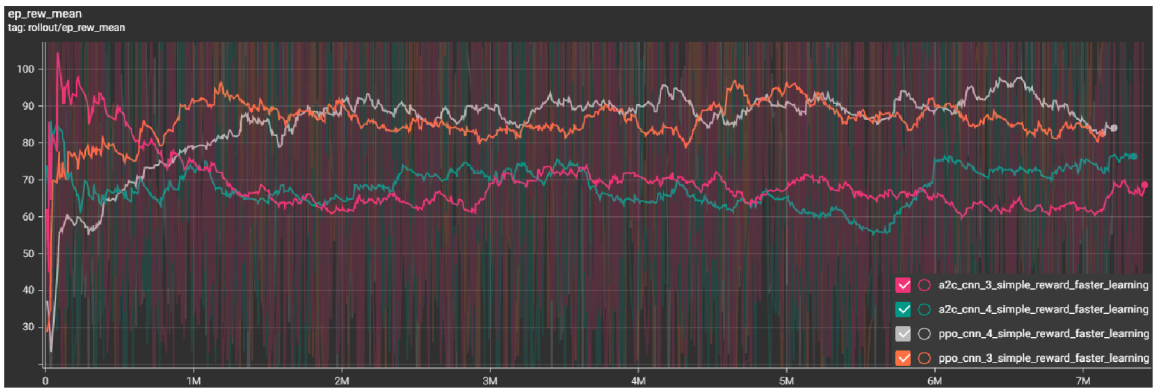
```
if current_location != last_location:
    return 2
return 0
```



Obr. 5.10: **Experiment s jednoduchou odmeňovacou funkciou 1.**

Toto však tvorilo chybu v testovaní, tak som ju upravil v 17. iterácii do tejto podoby:

```
if not done:
    if current_location != last_location:
        return 2
    return 0
else:
    final_count = n_my_units - n_enemy_units
    if final_count > 0:
        return 1 #win
    elif final_count == 0:
        return 0 #draw
    else:
        return -1 #lose
```



Obr. 5.11: **Experiment s jednoduchou odmeňovacou funkciou 2.**

Výsledky testu boli také, že sa agent len hýbal v podstate na tom istom mieste, kde sa narodil, doľava a doprava alebo hore a dole.

Kapitola 6

Záver

Cieľom práce bola integrácia, návrh, implementácia a trénovanie modelov posilňovaného učenia do herného prostredia typu Bomberman.

Modely vytvorené knižnicou stable baselines 3 sa mi podarilo integrovať s vybraným prostredím. Navrhol a naimplementoval som rôzne riešenia, ktoré by mali umožniť učenie. Patrí medzi ne napríklad výber algoritmov, politík, neurónových sietí a ich návrh, tvorba odmeňovacích funkcií a iných herných nastavení. Ďalšou dôležitou časťou je aj úprava prostredia do formátu čitateľného neurónovými sieťami a reprezentácia výstupu neurónovej siete pre vstup do herného servera na spracovanie týchto akcií. V rámci implementácie som vyskúšal dve politiky (MLP a CNN), dva algoritmy (PPO a A2C) a päť neurónových sietí na extrakciu vlastností za pomoci knižníc stable baselines 3 a pytorch.

Celkový čas trénovania týchto modelov bol dokopy 1207 reálnych hodín, 4168 strojových hodín a 271 miliónov herných krokov. Aj napriek tomuto úsiliu sa mi nepodarilo natrénovať úspešný model, ktorý by prekročil výhernú hranicu 50% proti náhodnému protihráčovi. Protihráči mali viacero verzií. Niektorí boli postavení do stredu, iní mali náhodne vygenerované začiatočné súradnice a nakoniec protihráči, ktorí vykonávali náhodné pohyby.

Treba taktiež poznamenať, že sa jednalo o náhodne generované prostredie, kde agent začínal vždy z inej začiatočnej pozície a každé herné prostredie vyzeralo inak a aj nepriatelia v ňom sa pohybovali náhodne. Toto mohol byť jeden z dôvodov, prečo sa agent nedokázal úspešne natrénovať. Prostredie bolo príliš zložité.

Ďalší dôvod môže byť taktiež zlá práca s prostredím. Prostredie bežalo v rozmedzí medzi 10 až 30 herných krokov za sekundu. To je na akékoľvek trénovanie veľmi málo, čoho som si nebol vedomí v čase trénovania. Zistil som to až ku koncu práce, keď sme s doktorom Hradišom skúšali vlastné veľmi zjednodušené prostredie, kde sa kroky za sekundu pohybovali v rozmedziach 20 až 200 krát vyšších. V tomto prostredí sa mi z časových dôvodov už nepodarilo natrénovať fungujúci model.

Keďže hra Bomberman nie je stále vyriešená a toto je moja prvá práca v tomto odvetví, nedokázal som identifikovať presné chyby, ktoré zapríčinili neúspech trénovania. Ak by som mal túto prácu začať nanovo s tým, čo viem teraz, určite by som si navrhol vlastné zjednodušené prostredie, ktoré by fungovalo rovnako ako Bomberland. Som si istý, že by som dostal vyššie rýchlosti pri trénovaní a teda aj viac výsledkov a experimentov. Taktiež by som sa viac zameral na nastavovanie hyperparametrov a parametrov neurónových sietí vo vnútri modelov. Pri písaní tejto práce mi napadlo aj viac generalizovať kanály a tým zredukovať ich počet. To by však pravdepodobne nemalo vplyv na funkcionalitu, len efektivitu.

Literatúra

[1] Baselines, S. *Stable Baselines Documentation* [online]. 2019 [cit. 2022-05-01]. Dostupné z: https://stable-baselines.readthedocs.io/en/master/guide/.

[2] Brownlee, J. *Crash Course On Multi-Layer Perceptron Neural Networks* [online]. 2016 [cit. 2022-05-08]. Dostupné z: https://machinelearningmastery.com/neural-networks-crash-course/.

[3] Brownlee, J. *When to Use MLP, CNN, and RNN Neural Networks* [online]. 2018 [cit. 2022-05-07]. Dostupné z: https://machinelearningmastery.com/when-to-use-mlp-cnn-and-rnn-neural-networks/.

[4] Chandrakant, K. *Reinforcement Learning with Neural Network* [online]. 2020 [cit. 2022-05-08]. Dostupné z: https://www.baeldung.com/cs/reinforcement-learning-neural-network.

[5] CoderOne. *Bomberland Documentation* [online]. 2022 [cit. 2022-05-01]. Dostupné z: https://www.gocoder.one/docs.

[6] Copeland, B. J. *The Modern History of Computing* [online]. 2000 [cit. 2022-05-08]. Dostupné z: https://plato.stanford.edu/entries/computing-history/.

[7] Courtney E. Ackerman, M. *Positive Reinforcement in Psychology* [online]. 2022 [cit. 2022-05-04]. Dostupné z: https://positivepsychology.com/positive-reinforcement-psychology/.

[8] Daniel L. Schacter, D. M. W. *Psychology.* Second edition. Worth Publishers, 2010. ISBN 978-1-4292-3719-2.

[9] DeepMind, G. *Asynchronous Methods for Deep Reinforcement Learning* [online]. 2016 [cit. 2022-05-07]. Dostupné z: https://arxiv.org/pdf/1602.01783.pdf.

[10] Dinesh. *CNN vs MLP for Image Classification* [online]. 2019 [cit. 2022-05-08]. Dostupné z: https://medium.com/analytics-vidhya/cnn-convolutional-neural-network-8d0a292b4498.

[11] Fandom. *Bomberman* [online]. 2012 [cit. 2022-05-02]. Dostupné z: https://bomberman.fandom.com/wiki/Bomberman_(series).

[12] Feng Ren, H. T. *Applying deep reinforcement learning to active flow control in weakly turbulent conditions* [online]. 2021 [cit. 2022-05-10]. Dostupné z: https://www.researchgate.net/publication/350209594_Applying_deep_reinforcement_learning_to_active_flow_control_in_weakly_turbulent_c

[13] Kaelbling, L. P. *Delayed Reward* [online]. 1996 [cit. 2022-05-05]. Dostupné z: https://www.cs.cmu.edu/afs/cs/project/jair/pub/volume4/kaelbling96a-html/node16.html.

[14] LENDAVE, V. *What Is A Convolutional Layer?* [online]. 2021 [cit. 2022-05-08]. Dostupné z: https://analyticsindiamag.com/what-is-a-convolutional-layer/.

[15] Mhossain. *Easy way to understand Convolutional Neural Network: It's Easy!!!* [online]. 2019 [cit. 2022-05-08]. Dostupné z: https://medium.com/@_moazzemhossain/easy-way-to-understand-convolutional-neural-network-its-easy-d9b7c0c5adb3.

[16] Neutelings, I. *Neural networks* [online]. 2021 [cit. 2022-05-08]. Dostupné z: https://tikz.net/neural_networks/.

[17] OpenAI. *Proximal Policy Optimization Algorithms* [online]. 2017 [cit. 2022-05-07]. Dostupné z: https://arxiv.org/pdf/1707.06347.pdf.

[18] OpenAI. *Dota 2 with Large Scale Deep Reinforcement Learning* [online]. 2019 [cit. 2022-05-02]. Dostupné z: https://cdn.openai.com/dota-2.pdf.

[19] OpenAI. *AI Gym Documentation* [online]. 2022 [cit. 2022-05-01]. Dostupné z: https://gym.openai.com/docs/.

[20] Rabin, S. *AI Game Programming Wisdom*. 1. vyd. Charles River Media, 2002. ISBN 1-58450-077-8.

[21] Simplilearn. *What is Perceptron: A Beginners Guide for Perceptron* [online]. 2022 [cit. 2022-05-08]. Dostupné z: https://www.simplilearn.com/tutorials/deep-learning-tutorial/perceptron.

[22] Sutton, R. S. a Barto, A. G. *Reinforcement Learning: An Introduction* [online]. The MIT Press, Cambridge, Massachusetts, London, England, 2014, 2015 [cit. 2022-04-28]. Dostupné z: https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf.

[23] Yoon, C. *Understanding Actor Critic Methods and A2C* [online]. 2019 [cit. 2022-05-07]. Dostupné z: https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f.