VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

HLUBOKÉ NEURONOVÉ SÍTĚ PRO POSILOVANÉ UČENÍ

DEEP NEURAL NETWORKS FOR REINFORCEMENT LEARNING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **VÁCLAV KOŠÁK**
AUTHOR

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

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2020/2021

Zadání bakalářské práce

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

Zadání:

1. Prostudujte základy neuronových sítí a posilovaného učení.
2. Vytvořte si přehled o současných metodách využívají neuronové sítě a posilované učení.
3. Vyberte konkrétní metodu aplikovatelnou na určitý problém posilovaného učení.
4. Obstarejte si databázi vhodnou pro experimenty.
5. Implementujte navrženou metodu a proveďte experimenty nad datovou sadou.
6. Porovnejte dosažené výsledky a diskutujte možnosti budoucího vývoje.
7. Vytvořte stručný plakát prezentující vaši práci, její cíle a výsledky.

Literatura:

- Peng et al.: DeepMimic: Example-guided Deep Reinforcement Learning of Physics-based Character Skills. ACM Trans. Graph., 2018.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3.

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

Vedoucí práce: **Hradiš Michal, Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 30. října 2020

Abstrakt

Práce popisuje trénovací prostředí pro trénování postavičky chodící po dvou končetinách. Prostředí je implementováno v AI Gym pomocí fyzikálního modelu PyBullet. Úlohy z prostředí jsou řešené pomocí posilovaného učení algoritmem ActorCritic. Každá z úloh je zaměřená na základní pohyby postavičky. Práce ukazuje, pomocí jakých funkcí odměn je algoritmus schopen dospět k řešení úloh.

Abstract

The paper describes a training environment for training a character how to walk. The environment is implemented in Al Gym by using the PyBullet physical model. Tasks from the environment are solved by using reinforcement learning by the ActorCritic algorithm. Each of the tasks is focused on the fundamental movements of the character. The paper show, which reward functions are used by the algorithm to solve given tasks.

Klíčová slova

posilované učení, Actor Critic, AI Gym, PyBullet, PyTorch, robot, strojové učení

Keywords

reinforcement learning, Actor Critic, AI Gym, PyBullet, PyTorch, robot, machine learning

Citace

KOŠÁK, Václav. *Hluboké neuronové sítě pro posilované učení*. Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Michal Hradiš, Ph.D.

Hluboké neuronové sítě pro posilované učení

Prohlášení

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

.....
Václav Košák
10. května 2021

Poděkování

Rád bych poděkoval vedoucímu mé bakalářské práce panu Ing. Michalu Hradišovi, Ph.D. za odborné vedení a poskytnuté konzultace, trpělivost a podnětné návrhy k mé práci. Dále pak rodině za psychickou podporu.

Obsah

Kapitola 1

Úvod

S roboty a pomocníky všeho druhu se můžeme setkávat čím dál častěji. Ve výrobnách jsou roboti podávající věci z jednoho místa na druhé. Roboti na třech a více kolečkách jsou běžnější, protože se nemusí řešit stabilita. Vyvíjí se však i roboti, kteří mají končetiny a mohou se tak pohybovat i na hůře dostupných místech, kde v cestě může být i nějaká překážka. Toto je možné hlavně díky strojovému učení, přesněji posilovanému učení, díky kterému se roboti mohou sami rozhodovat na základě předešlých zkušeností.

Cílem této práce je vytvořit virtuální prostředí s úlohami zaměřujícími se na konkrétní problém. Vytvořil jsem tedy postavičku s dvěma končetinami, kterou jsem učil stát, skákat a také chodit. Pro jednotlivé úlohy s postavičkou je použitý algoritmus z posilovaného učení. Posilované učení je jedna z metod strojového učení, které se učí na základě stavu, ve kterém se nachází, a ohodnocení za vykonané akce. Cílem učení je volit takové akce akce, které vedou k maximální odměně.

Prostředí může být jak virtuální, tak také reálné. Trénování v reálném světě, může ale způsobovat řadu problémů. Je to například fyzické opotřebení robota nebo také obsluha robota po dobu trénování, protože proces učení může trvat dlouho. Díky virtuálnímu prostředí můžeme simulovat to reálné, kde lze algoritmus jednodušeji natrénovat a pak jej jen otestovat v reálném světě.

Práce je členěna do sedmi kapitol. Druhá kapitola popisuje, co je to posilované učení a jeho základní pojmy. Ve třetí kapitole je popsán použitý algoritmus v experimentech. Ve čtvrté kapitole je popsáno prostředí, včetně úloh, které bylo potřeba nadefinovat. Pátá kapitola popisuje experimenty na daných úlohách, jak se algoritmu dařilo a co bylo potřeby vylepšit. Poslední kapitola popisuje závěry celé práce.

Kapitola 2

Existující přístupy

Práce, které se zabývají chůzí již pár bylo napsaných a každá přinesla nějaký nový přístup na řešené úloze.

První zajímavý článek je o učení chůze postavičky, která může mít různou velikost těla[34]. Cílem je tedy naučit univerzální policy, která si poradí se změnou velikosti těla postavičky například i v průběhu simulace. Postavička může měnit délku a šířku noh a ruk, šířku těla, a nebo délku jednotlivých noh, kde jedna z nich může být kratší. Funkce odměn je při učení složená ze 3 částí. První část, zda pohyb imituje referenční pohyby. Druhá část odměny je rozdíl průměrné rychlosti těla a cílové rychlosti těla. Třetí část odměny je o efektivním pracování s energií. Pro naučení obecné policy, která zvládne pracovat s postavičkou, bylo potřeba generovat jednotlivé postavičky, které byly kombinací určitých parametrů, protože i menší změna jednoho z parametrů může měnit strategii volby akce. Některé kombinace parametrů (širší, kratší jedna noha, ...) potřebují delší čas učení než jiné.

Dalším zajímavým přístupem je učení pomocí osnovy. Naučit se něco hned je těžké a je jednodušší se naučit věc postupně. S tímto principem přichází článek o učení pomocí osnovy[35]. Jsou zde 3 druhy postaviček a úloha je pro všechny postavičky stejná. Cílem postavičky je stoupat na kameny, které se před postavičkou postupně generují v různé výšce, vzdálenosti, úhlu natočení atd. Práce přináší různé způsoby učení podle osnovy. Začíná se od jednodušších parametrů po složitější. Když se podaří zvládat určitou úroveň obtížnosti, tedy vzdálenost, úhel natočení atd., tak se parametry posunou a učí se na obtížnějších parametrech.

Kapitola 3

Posilované učení

Posilované učení a celkově strojové učení jde nyní hodně dopředu a aplikuje se prakticky v každém odvětví.

Pojem strojové učení je velice rozsáhlý a obsahuje v sobě hodně podtémat a druhů. Záleží hlavně na trénovacích datech a také na výsledném využití neuronové sítě.

3.1 Rozdíly mezi druhy strojového učení

Práce se zaměřuje na Posilované učení. Pro doplnění a zařazení tohoto tématu do problematiky strojového učení[27], zde jsou další metody, které se využívají. Každý typ je vhodný pro jiná data. Občas se však mohou tyto témata doplňovat.

Učení s učitelem

Cílem učení s učitelem[33] je naučit model predikovat správný štítek na nový vstup. Vstupem při učení je série dvojic, vždy data a štítek. Data může být jakákoliv nasnímaná hodnota – obraz, zvuk, teplota, atd. Štítek je informace, která popisuje data. Během trénování se hledají vzory, které korelují s navrhovanými výstupy. Učení s učitelem může být rozděleno na 2 podkategorie: **klasifikace** a **regrese**. Příkladem úloh může být klasifikace předmětů na fotce, detekce e-mailového spamu.

Učení bez učitele

Na rozdíl od Učení s učitelem jsou u Učení bez učitele[21] na vstupu neanotovaná data. Cílem tedy je zjednodušit data a seskupovat data s podobnými vlastnostmi do několika shluků. Typické úlohy mohou být shlukování objektů, detekce anomálii.

Posilované učení

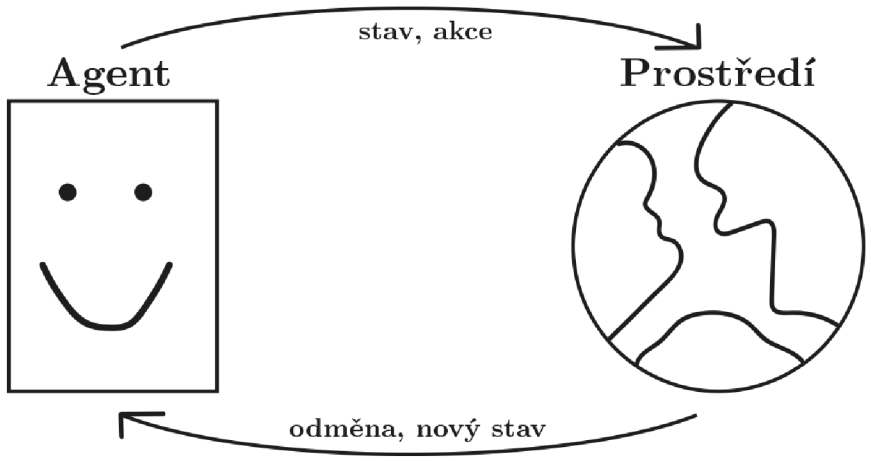
Cílem posilovaného učení je naučit strategii agenta, která vede k maximalizaci sumy průměrné sumy odměn. Strategie se mění na základě pozitivní a negativní zpětné vazby po sérii akcí v externím prostředí. Výstupem je agent, který volí nejlepší akce ve stavu v prostředí. Příklad aplikace může být naučit robota chodit, řídit autonomní vozidlo, hrát deskovou či počítačovou hru.

3.2 Co je to posilované učení

Posilované učení[26, 30] je učení, jehož cílem je naučit agenta, který získá při svém běhu co nejvyšší odměnu. Učení agenta probíhá na základě získané odměny, která byla získaná z provedených akcí, které zvolila policy. Základem posilovaného učení jsou 2 komponenty. **Agent** a **prostředí**. Grafické znázornění jde vidět obrázku 3.1.

Agent je ten, který reaguje na příchozí stavy a volí akce. Prostředí vykonává akce, které dostane od agenta, a vrací odměnu a nový stav. Cílem agenta je dělat akce, které maximalizují celkovou odměnu. Nezáleží tedy na dílčích odměnách, ale na sumě všech získaných odměn. Série menších odměn může vést k vyšší odměně.

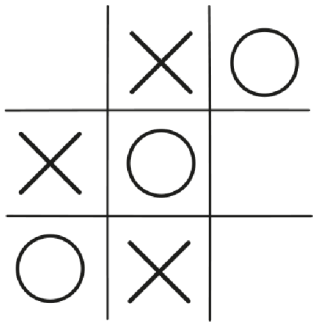
Cílem posilovaného učení je naučit policy agenta, který se učí z jednotlivých zkoušení vytvořit strategii, která ve všech stavech prostředí bude dělat akce, které povedou k nejvyšší odměně.



Obrázek 3.1: Cyklus posilovaného učení, kde mezi sebou komunikuje agent a prostředí. Agent vytváří akce na základě příchozího stavu z prostředí, která obsahuje informaci o stavu prostředí po provedení předchozí akce. Prostředí také vrací odměnu za provedou akci, kterou se agent snaží maximalizovat. Inspirováno z [30].

Pro lepší představu je dobré si to ukázat na příkladu hry piškvorky 3.2. Jedná o hru, kde 2 hráči hrají proti sobě, každý má svůj obrazec, a mají za úkol mít v jakémkoliv směru 3 své obrazce. Hraje se do té doby, než některý z hráčů nemá v jakémkoliv směru 3 své obrazce nebo dokud není pole plné. Hra tedy může skončit 3 scénáři. První scénář je, že vyhraje hráč 1. Druhý scénář, že vyhraje hráč 2, a třetí, že se jedná o remízu, tedy žádný z hráčů nevytvořil řadu 3 stejných obrazců.

Cílem posilovaného učení je naučit funkci, která v jednotlivých stavech bude dělat co nejlepší rozhodnutí, aby vyhrála. Bude tedy postupně hrát a například na konci epizody dostane ohodnocení za hru – kladné body za výhru, za remízu nic, za prohru záporné body. Na základě těchto informací se agent bude zlepšovat a podávat tak lepší výsledky.



Obrázek 3.2: Jedna hra piškvorek v rozehraném stavu, která zobrazuje jednu celou epizodu, a může symbolizovat prostředí. Agent vytváří akce, tedy kam vloží svůj znak (třeba křížek), tak aby vyhrál. Prostředí může být protihráč (s kolečkem), a agentovi vrací nový stav, kde se projeví akce agenta i prostředí. Stav může obsahovat informaci, která políčka jsou zabraná a která ne. Vrací také odměnu za provedenou akci. V tomto případě to může být až na konci.

Epizoda

Samotné učení musí probíhat v nějakém rámci. Epizoda je sekvence kroků, od času 0 až po čas ukončení. Celá epizoda se tedy skládá ze sekvence stavů a akcí, v jejímž průběhu se získává odměna. Cílem je, aby konečná odměna byla co největší.

Epizoda může být ukončena buď prostředím nebo agentem. Epizoda je ukončena prostředím, když se stane něco, co dál neumožňuje pokračovat. Například robot, který se snaží o chůzi, spadne. Agent končí epizodu, když se překoná nastavený počet opakování nebo pokud je dosažená cílová odměna. Toto jsou hodnoty, které jsou variabilní a před trénováním jsou nastavené agentem.

Prostředí

Každý algoritmus se potřebuje rozhodovat na základě informací. Prostředí tedy poskytuje soubor informací, které umožňují agentovi se rozhodnout.

Každá úloha v posilovaném učení má definované prostředí. Jedná se o prostor, kde se všechno odehrává a podle čeho se agent rozhoduje, jakou akci udělá. Prostředí tedy agentovi předává aktuální stav prostředí. Stav je soubor potřebných informací o objektu, se kterým pracujeme. Mohou to být informace ze senzorů, u pohyblivé postavičky poloha končetin, u auta úhel otočení volantu atd.

Akce

Akce je nějaký pokyn, co má stroj, postavička nebo cokoliv jiného v prostředí udělat. Příkladem může být pohnutí končetiny, zatočení volantem, přidání plynu, posunutí ramene atd. Rozlišujeme 2 typy: **Diskrétní** a **spojité**.

Diskrétní akce je číslo, které symbolizuje danou akci. U pohybu do jedné ze čtyř stran může číslo symbolizovat například pohyb doprava. Pokud jsou k akci potřebné ještě nějaké další údaje, například síla, kterou se bude působit v daném směru, tak jsou nadefinované v samotném prostředí.

Dalším typem akce jsou spojité. Akce zde nesymbolizuje určitý pohyb, ale spíše sílu, míru, jakou má akci udělat. Akce je vyjádřena číslem v určitém rozsahu. Na příkladu auta může být akce, jak moc má zrychlovat, jak moc má zatočit volantem na kterou stranu atd.

Odměna - reward

Odměna je základní prvek u posilovaného učení. Jedná se o zpětnou vazbu od prostředí k agentovi za jednotlivé akce, které agent prostředí předal. Celková odměna získaná za epizodu by se pak dala vypočítat jako:

$$G_t = \sum_{t=0}^{T} t_t. \tag{3.1}$$

Pro agenta je lepší mít odměnu teď než někdy v budoucnu, zároveň matematicky by odměna nemusela konvergovat na určitou hodnotu. Proto je zaveden **discount factor**, pomocí kterého nás nezajímají příliš vzdálené odměny. Jedná se o číslo v rozsahu 0 až 1. Hodnota 0 znamená, že nás zajímají pouze aktuální hodnoty, hodnota 1, že všechny budoucí. Například u hodnoty 0.9 nás zajímá následujících 6, u hodnoty 0.99 následujících 60. Pomocí vzorce lze odměnu vypočítat jako:

$$G_t = \sum_{t=0}^{T} \gamma^i t_t. \tag{3.2}$$

Trajektorie

Trajektorie je sekvence stavů a akcí, které agent získal během jedné epizody. Matematicky to lze vyjádřit jako:

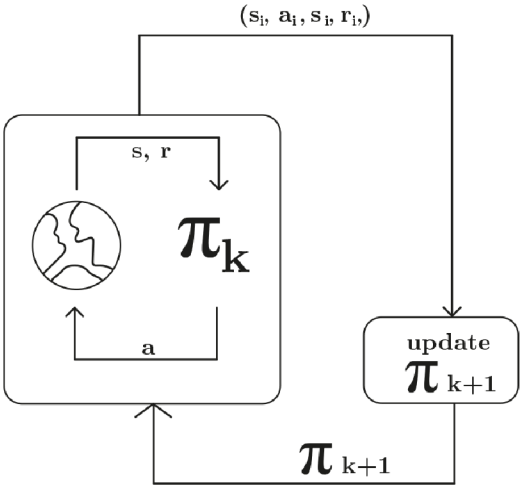
$$\tau = (s_0, a_0, s_1, a_1, ...). \tag{3.3}$$

Off policy / on policy

Akce vytváří agent a provádí se v prostředí. Existuje ale více možností, jakými způsoby se nové akce vytvářejí. Je na to spoustu přístupů a každý algoritmus může využívat jinou metodu. Mezi přístupy patří on-policy, off-policy a offline[10, 25].

on-policy – Zkušenosti jsou sbírány pomocí nejaktuálnější policy. Jedná se o druh online interace. Agent komunikuje s prostředním a sbírá informace o běhu. Pomocí zkušeností se zlepšuje policy. Průběh trénování je zobrazen na obrázku 3.3. **Příklad**: Policy iteration, PPO, Sarsa, atd.

off-policy – U off policy se zkušenosti z jednotlivých epizod ukládají do paměti. Každý další běh s novou policy přidá nové zkušenosti do paměti. U každého algoritmu to může být různě, ale například u DQN se ukládá čtveřice (stav, akce, odměna, nový stav). Tyto data se používají pro trénování nové policy π_{k+1}. Pro novou policy se tedy berou jak nové data z nejnovějšího běhu, tak i ty starší. Vždy se vybere nějaký vzorek a na něm se trénuje. Off-policy lze použít na učení z ukázky, případně i na paralelní učení. Průběh trénování je zobrazen na obrázku 3.4. **Příklad**: Q-learning, DQN, DDQN, atd.



Obrázek 3.3: Průběh trénování založené na on-policy. Agent (symbolizovaný pomocí π) s prostředím nejdříve komunikují mezi sebou, a na konci epizody proběhne trénování policy (π) nad danou epizodou, tedy sekvencí akcí, stavů a odměn. Inspirováno z [10].

offline – Offline posilované učení je speciální. Agent (symbolizovaný pomocí π) zde nekomunikuje s prostředím, není schopen. K dispozici jsou pouze nasbíraná data o fixní velikosti. Cíl je naučit na těchto datech co nejlepší policy (π). Průběh trénování je zobrazen na obrázku 3.5.

3.3 Policy

Agent je ten, který volí akce, jež se mají v prostředí provést. Součástí agenta musí být funkce, která tyto akce vytvoří. Jedná se o policy[18], tedy funkci, která mapuje stavy na akce. Jsou 2 druhy policy: deterministická policy a nedeterministický – stochastická policy[14].

Deterministická policy

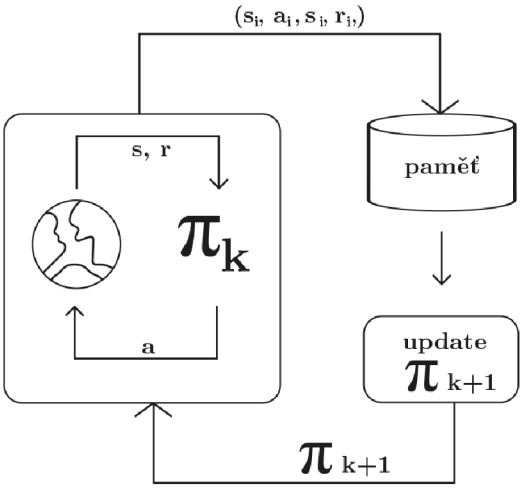
U deterministické policy zvolená akce záleží pouze na stavu. V každém stavu je jasně dané jaká akce bude provedena. Když agent volí rozložení pravděpodobností akcí, tak pravděpodobnost jedné akce je 1, zbytek 0. Deterministická policy lze vyjádřit vzorcem:

$$\pi(s) = a \tag{3.4}$$

Stochastická policy

Výstupem stochastické (náhodné) policy je rozložení pravděpodobnosti pro jednotlivé akce z daného stavu. Pokud bychom znovu vzali příklad auta se 3 akcemi: vlevo, rovně, vpravo, tak výstupem by mohlo být 30%, 10%, 60%. Agent tedy může zvolit jakoukoliv akci, pokud by chtěl maximalizovat odměnu, tak vybere tu nejpravděpodobnější. Stochastická policy lze vyjádřit i vzorcem:

$$\pi(a, s) = \mathbb{P}[A_t = a | S_t = s] \tag{3.5}$$



Obrázek 3.4: Průběh trénování u algoritmů založené na off-policy. Agent (symbolizovaný pomocí π) s prostředím nejdříve komunikuje do konce epizody. Poté je trajektorie uložena do paměti, nad kterou pak probíhá trénování policy (π). Inspirováno z [10].

3.4 Markovovy modely

Téměř každý problém z posilovaného učení lze dosadit do MDP (Markov decision process)[6, 17]. MDP je framework, skládající se ze dvou částí viz. obrázek 3.6, který pomáhá řešit problémy posilovaného učení. Je to matematický systém. Formálně se jedná o pravděpodobnostní automat, příklad takového automatu je na obrázku 3.7.
U MDP jsou 2 vlastnosti: Přechody a pravděpodobnost přechodu.

- **Přechod** – Pohyb z jednoho stavu do druhého
- **Pravděpodobnost přechodu** – Pravděpodobnost, která určuje jakým přechodem se agent dále vydá.

O Markovovův přechod se jedná, pokud současný stav závisí pouze na předchozím stavu. To samé platí, že další stav závisí pouze na současném. Matematicky to lze vyjádřit jako:

$$\mathbb{P}[s_{t+1}|s_t] = \mathbb{P}[s_{t+1}|s_1, ..., s_t] \tag{3.6}$$

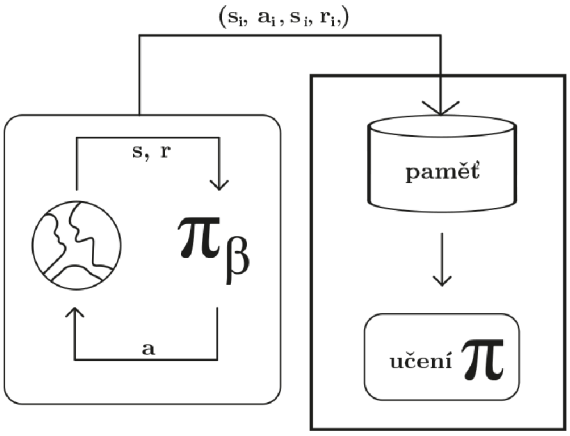
Markovovův proces je definován jako *(S, P)*, kde S jsou stavy a P je pravděpodobnost přechodu. Jedná se o sekvenci stavu $s_1, s_2, ...$, které musí splňovat podmínku markovových přechodů. Pravděpodobnost přechodu $P_{ss'}$ je pravděpodobnost přechodu ze stavu s do stavu s', matematicky se to vyjádří jako:

$$P_{ss'} = \mathbb{P}[s_{t+1} = s_1|s_t = s] \tag{3.7}$$

Pravděpodobnost pro všechny přechody mezi stavy lze pak vyjádřit pomocí matice, jejichž velikost závisí na počtu přechodů a stavů.

MDP[7] je definováno jako $M =< S, A, P, R, \gamma >$, kde jednotlivá písmena mají stejný význam jako u posilovaného učení.

- **S** – množina stavů



Obrázek 3.5: U offline policy trénování probíhá již s nabitými daty, které byly někdy nasbírány pomocí nějaké policy. Nad daty, které jsou uložené v paměti, pak probíhá trénování a cílem je na nich naučit co nejlepší policy. Inspirováno z [10].

- **A** – množina akcí, které si agent může vybrat
- **P** – matice pravděpodobnosti přechodu
- **R** – množina odměn
- γ – discount faktor

3.5 Value Based algoritmy

Value based[13] je základní koncept v posilovaném učení. Odhaduje se, jak dobré je dosáhnout určité stavy či vykonat určité akce. Cíl u value based je optimalizovat hodnotovou funkci $V(s)$, která je definovaná jako funkce s maximální předpokládanou odměnu, kterou agent může získat v jednotlivých stavech.

Zástupcem algoritmu u učení hodnotou může být například Q-learning a jeho různé varianty. Učení hodnotou však může být v některých případech příliš složité, protože se musí prohledat velmi velký prostor stavů. V posilovaném učení tedy funguje často jako součást nějakého algoritmu, který jej kombinuje s nějakým jiným. Příkladem pak může být algoritmus Actor Critic.

Value based můžeme rozdělit na 2 katetegorie: **State value** a **Action value**[19].

State value

Očekávaná celková odměna začíná ve stavu s a koná dle policy. Pokud agent používá policy, tak funkce je pak ve tvaru:

$$V^{\pi}(s) = \mathbb{E}[\sum_{i=1}^{T} \gamma^{i-1} r_i], \forall s \in \mathbb{S} \tag{3.8}$$

Obrázek 3.6: Základní cyklus u MDP (Markov decision process). Skládá ze 2 prvků - Agent a Prostředí. Tyto prvky mezi sebou komunikují a navzájem si vyměňují informace o stavu, akci a odměně.

Ve vzorci 3.8 znak $\mathbb{E}$ (expected) značí očekávanou odměnu. Uvnitř těla je reward discount, tedy parametr pracující s budoucí odměnou.

Action value

Očekávaná hodnota pro agenta, který začíná ve stavu s a vykonává akci. Pak podle policy π. Stav může mít více akcí, bude obsahovat více Q hodnot.

Optimální Q funkce $Q^*(s, a)$ znamená maximální možná Q hodnota pro agenta startující ve stavu s a volí akci a.

Advantage

Rozdíl mezi action value a state value je advantage fuction (A hodnota). Tato hodnota je například součástí algoritmu A2C.

$$A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s) \tag{3.9}$$

Optimální hodnota a policy

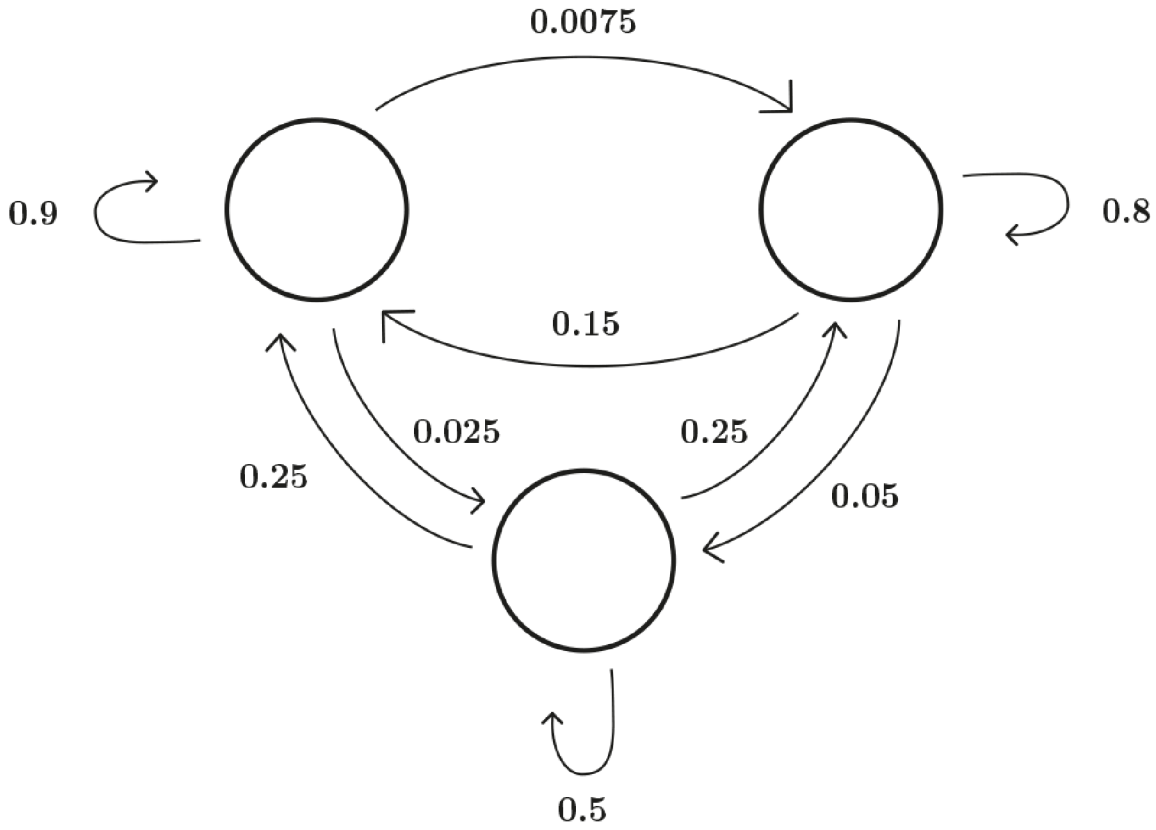
Cílem posilovaného učení je natrénovat síť, která má ideálně dát v každém stavu optimální[19] akci. Rozdíl je znázorněn na obrázku 3.8, kde lze vidět chování random policy. Zvolená akce je vždy náhodná. U optimální policy se jedná o akci, která vede k co největší odměně. Optimální funkce, která produkuje maximální možnou odměnu:

$$V^*(s) = \max_\pi V^\pi(s), Q^*(s, a) = \max_\pi Q^\pi(s, a) \tag{3.10}$$

Optimální policy achieves optimal value functions:

$$\pi_* = \arg\max_\pi V^\pi(s), \pi_* = \arg\max_\pi Q^\pi(s, a) \tag{3.11}$$

Platí tedy i:



Obrázek 3.7: Příklad MDP, kde jednotlivé stavy mohou symbolizovat činnosti během dne s odměnou, a přechody obsahují informaci, jaká je pro daný přechod.

$$\begin{aligned} V_{\pi_*}(s) &= V_*(s) \\ Q_{\pi_*}(s,a) &= Q_*(s,a) \end{aligned} \tag{3.12}$$

Vztah mezi $Q^*(s,a)$ a $V^*(s)$ je vyjádřen jako:

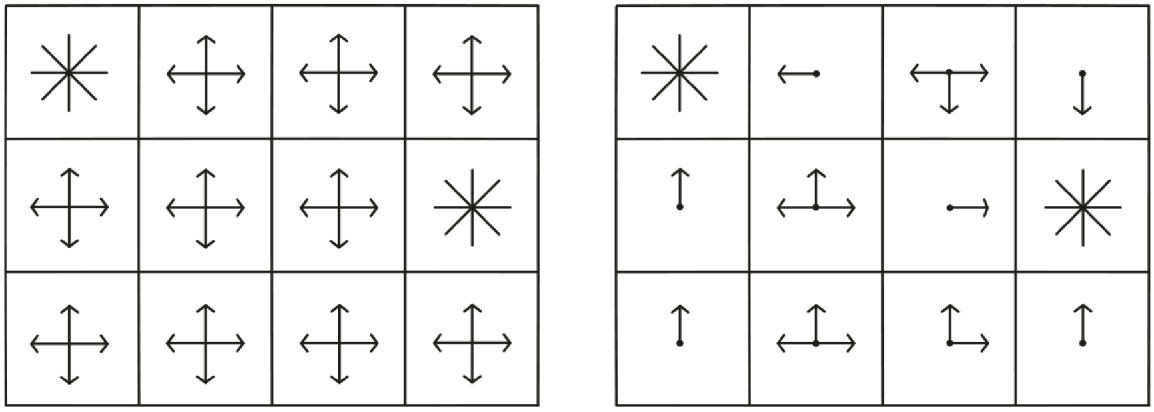
$$V^*(s) = \max_a Q^*(s,a) \tag{3.13}$$

Příklad fungování

Matematicky už to máme odvozené. Nyní pár praktických ukázek, pro dokreslení představy o tom, jak to funguje.

Value based algoritmy[16] jsou dobré, kde je omezený state space. Pochopit algoritmy založené na tomto principu, jak fungují, jde jednoduše na šachovnicovém poli viz. obrázek 3.9, kde se máme dostat z bodu A, a cestou získat co nejvyšší odměnu.

Na začátku je ohodnocení všech stavů nulové. Po procházení se tato pravděpodobnost bude měnit na základě toho, jak je daný stav dobrý, a který vede k co největší odměně. Pro demonstraci počítejme, že pravděpodobnost přechodu nahoru je $0,1$, doprava $0,8$, a dolů $0,1$. Pokud se dotkneme zdi nebo hranic pole, tak se vrátíme zpět do původní polohy.



Obrázek 3.8: Rozdíl mezi náhodnou a optimální policy. V random policy nevíme jaký stav a akce jsou výhodné, a proto probíhají náhodně a výsledná odměna bude pravděpodobně malá. U optimální policy známe kvalitu akcí a stavů a můžeme tak získat maximální možné score.

Discount factor zvolíme jako $0,9$. V poli jsou 2 pole s odměnou, kladná s hodnotou $+1$, a záporná s hodnotou -1.

Trénování a zjišťování, který stav je dobrý, probíhá v epizodách, jak je vidět na obrázku 3.10, kde jsou zobrazené dvě iterace, které již obsahují ohodnocení jednotlivých stavů a také, jaké změny mezi nimi probíhají. Lze vidět, že změny jsou ještě relativně velké a u většího prostoru bude déle trvat, než hodnoty zkonvergují nebo minimálně se přiblíží k hodnotě, která se nemusí již moc měnit.

Nový stav $[2,3]$ v iteraci 4, která je na obrázku 3.10 se vypočítá jako:

$$\begin{aligned} V[2,3] &= 0,8 \cdot 0,83 \cdot 0,9 + 0,1 \cdot 0,51 \cdot 0,9 + 0,1 \cdot (-1) \cdot 0,9 \\ &= 0,5976 + 0,0459 - 0,09 \\ &= 0,55 \end{aligned} \tag{3.14}$$

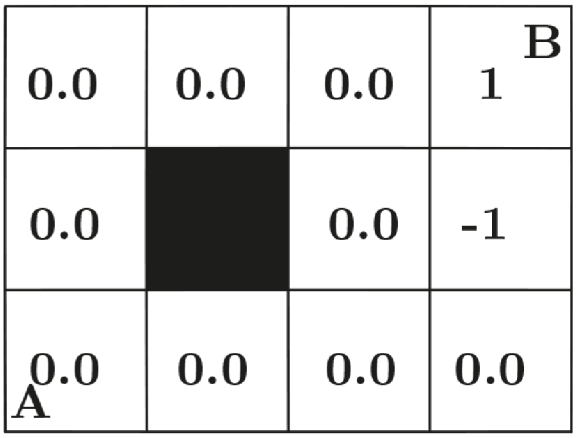
Ve vzorci je na prvním místě pravděpodobnost přechodu. Na druhém místě je aktuální hodnota v daném směru, a na třetím místě discount factor. Na obrázku 3.11 lze vidět iteraci 100, kde hodnoty zkonvergovaly k nějaké hodnotě. Může se už jednat o ideální policy. Nyní lze i pěkně vidět, která z cest je výhodnější, kde je větší ohodnocení stavů.

3.6 Policy based algoritmy

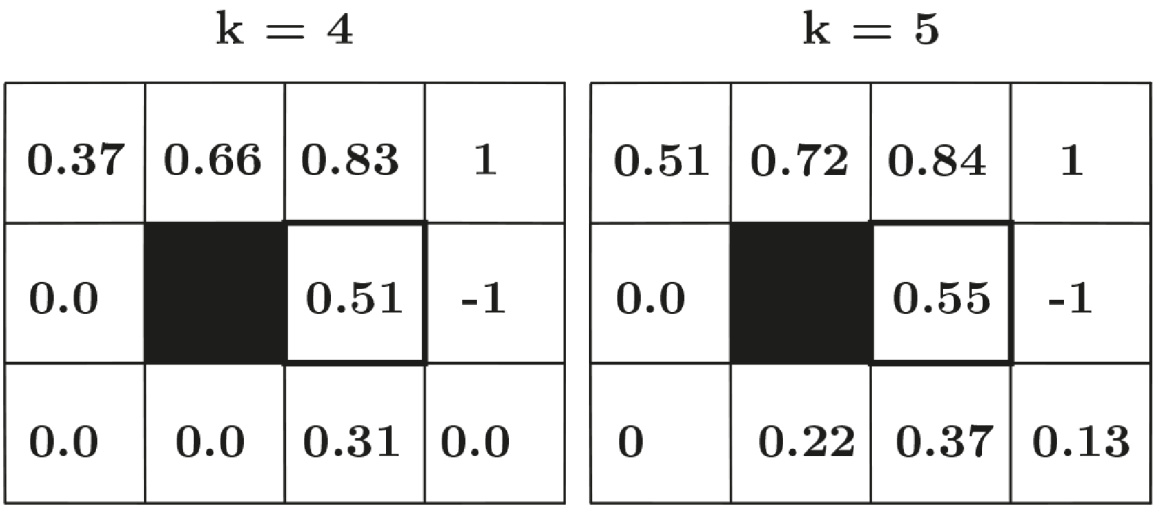
Policy based[31, 24] algoritmy mají za cíl vytvořit hodnotící funkci, jejíž úlohou bude na získaný stav vytvořit akci, která bude dosahovat nejvyšší možné hodnoty. V této metodě máme policy π, síť získá aktuální stav a na jeho základě vyprodukuje rozložení pravděpodobnosti pro jednotlivé akce (pro diskrétní akce). Síť vypočítá pravděpodobnost pomocí vzorce

$$\pi_\theta(a|s) = P[a|s], \tag{3.15}$$

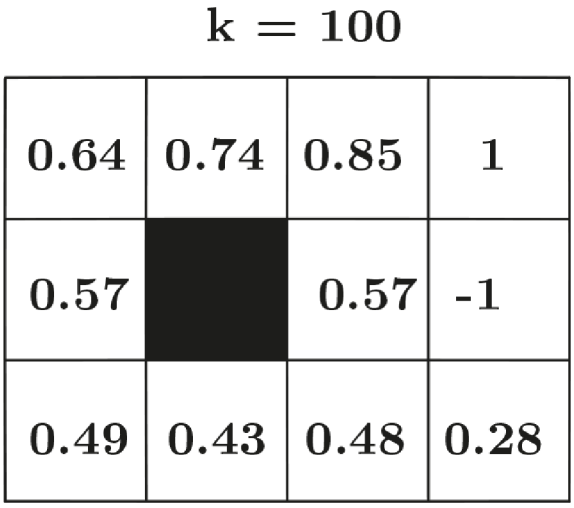
kde θ jsou parametry neuronové sítě. Výstupem je rozložení pravděpodobností akcí a ve stavu s s parametry θ. Průběh je vidět na obrázku 3.12, kde v první části je aktuální stav. Ve druhé části policy pracuje se stavem s a produkuje rozložení pravděpodobností,



Obrázek 3.9: Nejjednodušší úloha value based, kde je cíl dostat se z bodu A a cestou nasbírat co největší odměnu. Čísla v jednotlivých polích, které symbolizují stavy, zobrazují, jak dobrý daný stav je. Toto číslo se v jednotlivých iteracích mění pomocí nějakého value based algoritmu. V poli jsou 2 odměny. Jedna kladná (+1) a záporná (−1). Inspirováno z [16].

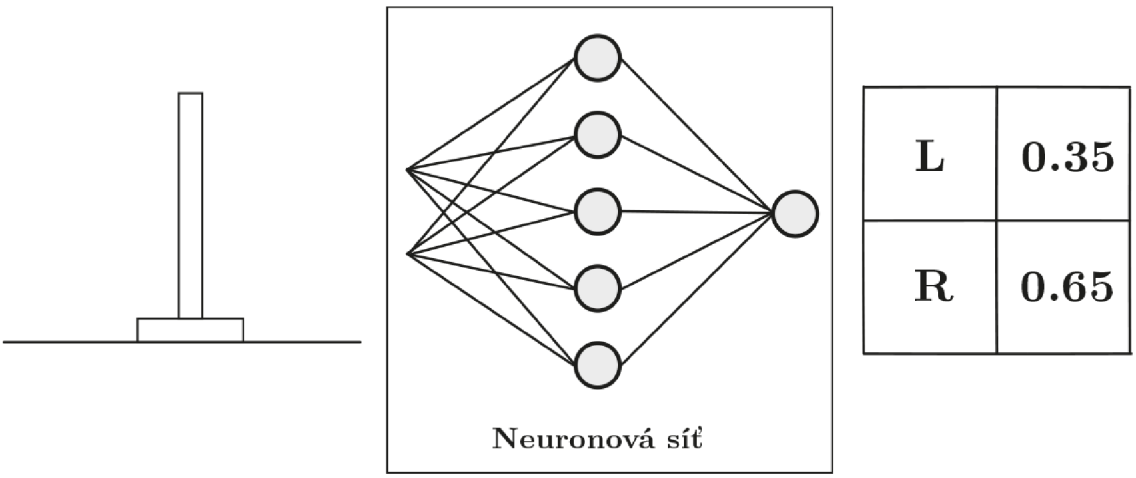


Obrázek 3.10: Ukázka 4. a 5. iterace u základní value based úlohy. Oba obrázky zobrazují ohodnocení stavů a také, v jak se mění ohodnocení v jednotlivých iteracích. Pro dosažení optimální policy by mělo učení probíhat do doby, než hodnoty zkonvergují k určité hodnotě. Inspirováno z [16].



Obrázek 3.11: Ukázka 100. iterace, kde jednotlivé stavy zkonvergovaly na určité hodnoty, které by se v dalších iteracích téměř neměnily. Čísla zobrazují ohodnocení stavů a lze tak vidět cestu, která je nejvýhodnější pro zisk co největší odměny. Inspirováno z [16].

které je zobrazeno ve třetí části. Pokud chceme maximální odměnu, zvolíme akci s nejvyšší pravděpodobností, pokud nás aktuálně nezajímají nejvyšší odměny, ale budoucí možné, tak můžeme zvolit náhodně s reflektováním pravděpodobnosti.



Obrázek 3.12: Zobrazení fungování policy based algoritmu. V první části je úloha v prostředí, která je v nějakém stavu. Druhá část je neuronová síť, která zpracuje stav z prostředí. Ve třetí části je výstup neuronové sítě. Jedná se o rozložení pravděpodobnosti jednotlivých akcí.

Nyní máme policy, která ze zadaného stavu s dokáže vyproduktovat akci a a dává nám nějaké výsledky. Je ale potřeba nějaká funkce, která se bude starat o zlepšování policy π. Funkce by tedy měla hodnotit, jak je naše policy funkce dobrá. Každé konečné MDP má optimální policy (která nám dá největší odměnu). Výpočet u policy based algoritmů probíhá pomocí gradientu[23].

Potřebujeme vědět, jak se agentovi daří v každé epizodě. Hodnotící funkce závisí na score, které je za epizodu získané. Kvalita záleží na prostředí, na odměnách, obecně je ale výše odměny dobré měřítko kvality, tedy jaké dosahuje celkové skore za epizodu. K zjištění musím použít discount faktor tak, abychom počítali s odměnami v nějakém horizontu, tedy s určitým počtem. Definované je to jako:

$$G(\tau_\theta) = \sum_{i=0}^{T} \gamma^t r_t, \tag{3.16}$$

kde τ je trajektorie. Tato suma discount rewards, které agent dostal v aktuální epizodě pomocí policy π.

Loss funkce u policy based funguje na základě odměn. U pozitivních odměn zvyšujeme pravděpodobnost akcí, u záporných naopak. Cílem není mít pravděpodobnost akce 1 nebo 0. Díky tomu bychom přišli o možnost zkoumat prostředí. Akce s pravděpodobností 0 by se nikdy neprovedly a zároveň akce s pravděpodobností 1 by se provedly vždy a neobjevili bychom tak nový stav, který by mohl vést k větší odměně.

Informace, jak dobré je policy, se vypočítá jako:

$$J(\pi_\theta) = \mathbb{E}[G(\tau_\theta)], \tag{3.17}$$

Toto je očekávaná odměna na konci epizody pomocí policy π_θ.

Pokud náhodně inicializujeme θ, spustíme několik epizod a odměny uložíme, s největší pravděpodobností zjistíme, že průměr odměn je nízký, to znamená, že naše policy je špatná. Gradient však ukazuje ve směru největšího růstu. Pokud můžeme vypočítat $J(\pi_\theta)$ s parametry policy, pak můžeme updatovat, policy jako:

$$\theta = \theta + \alpha \nabla_\theta J(\pi_\theta), \tag{3.18}$$

kde α je krok v learning rate[9], který nám říká, jak rychle má probíhat učení. Je to parametr, který ovlivňuje rychlost. Rychlost učení určuje, jak rychle se model přizpůsobuje danému problému. Tato hodnota se pohybuje v rozmezí od 0 do 1. Menší rychlost učení vyžaduje více trénovacích epizod vzhledem k menší váze informací pro změnu dat, vyšší hodnota jich potřebuje méně. Pokud je však míra učení velká, tak se může rychle blížit k neoptimálnímu řešení, zatímco malá rychlost může zkoumat více celý prostor a povede tak k najití optimálního řešení.

Nyní jsme schopní vcelku jednoduše aproximovat gradient $J(\pi_\theta)$ pomocí pár trajektorií a dojít tak k vyšší odměně. Vzorec pro odvozený policy gradient[23] je:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}[\sum_{i=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) G(\tau_\theta)]. \tag{3.19}$$

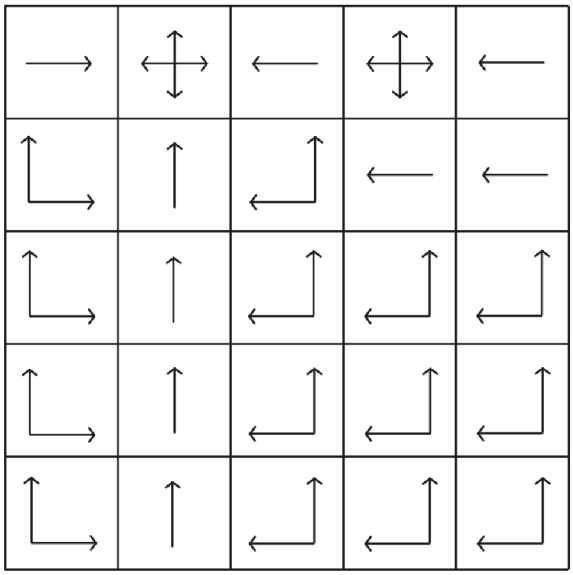
Policy based algoritmy řeší problém s nekonečno akcemi – u value based je to problém, protože se musí počítat kvalita dvojice stavu a akce, aby se vypočítala optimální policy.

U policy based algoritmů není ze začátku problém s poměrem zkoumání a využíváním již nabitých zkušeností. Na začátku policy funkce příliš neví, jaká akce je dobrá a jaká naopak špatná, a tak prozkoumává prostředí. Postupem času, jak se policy učí, tak se snižuje procento akcí, které zkoumají prostředí a naopak roste procento akcí, které vedou k větší odměně. Aby se i v pozdějších fázích zkoumaly nové akce, využívá se entropy[3].

Entropy u posilovaného učení slouží k častějšímu volení nepředvídatelných akcí. Čím větší je entropy hodnota, tím víc náhodných akcí agent provede. Cílem posilovaného učení

je naučit se sekvenci akcí, které vedou k co největší odměně a dochází pak přirozeně k tomu, že se nevolí nové akce a může docházet k tomu, že je dosažené nějaké maximum, což může být pouze lokální maximum a díky tomu, že by algoritmus neprováděl náhodné akce, tak by globální maximum nemusel vůbec najít.

Optimální policy může vypadat jak na na obrázku 3.13, kde je v každém stavu možné vybírat už jen z určitých akcí, které vedou k největší odměně.



Obrázek 3.13: Příklad, jak může vypadat optimální policy. Jednotlivé stavy ve stavech nabízí už jen určité akce, které mohou vést k vyšší odměně.

3.7 Exploration / Exploitation

Jedná se o problém a výzvu v posilovaném učení správně volit poměr mezi zkoumáním (Exporation) a využíváním již nabitých zkušeností z minulosti (Exploitation)[20].

Snaha agenta je získávat co největší odměnu za epizodu. Agent proto preferuje spíše akce, které vybírá s využitím zkušeností z předchozích epizod, u kterých dokáže lépe predikovat vyšší odměnu. Zkoumáním prostoru a dělání náhodných akcí, které nemusí být v daném stavu nejvýhodnější, v budoucnosti však mohou přinést vyšší odměnu.

Agent by měl současně využívat již nabrané zkušenosti pro získání co největší odměny, a zároveň také zkoumat nové akce, které mu mohou v budoucnu přinést mnohem větší odměnu. Strategií je více, například e-greedy, kde se ze začátku provádí více náhodných akcí a postupně během učení se procento náhodných akcí zmenšuje.

3.8 Spojité akce

Spojité akce[5] jsou speciální případ akcí u stochastické policy. Jedná se o akce, kdy agent musí mít na výstupu nějaké reálné číslo. Jako příklad může být, jakou silou pohnout končetinou nebo jak moc zatočit volantem. Tento problém lze řešit více způsoby

- **Rozbití akcí na segmenty** – Máme interval hodnot, ve kterém se hodnoty akcí musí nacházet. Daný interval se rozseká na menší intervaly a vzniknou nám tak diskrétní

akce, které reprezentují daný interval. Toto řešení nefunguje, když je potřeba extrémní preciznost.

- **Na výstupu parametrizovaný model** – Model, který bude dávat na výstupu číslo z určitého intervalu. Existuje tak téměř nekonečno možných výstupů.

U diskrétních akcí nám agent dává na výstup rozložení pravděpodobností jednotlivých akcí. Na vstupu máme stav, na výstupu dimenzi akce a pak pomocí funkce softmax zjistíme pravděpodobnost jednotlivých akcí.

U spojitých akcí máme nekonečno možných akcí, které se však musí nacházet v předem definovaném intervalu. Ideálně i zde potřebujeme podle něčeho akce vybírat a k tomuto účelu se dá využít normální rozložení (gaussova křivka), kde jako výstup učení je rozptyl a směrodatná odchylka a pomocí tohoto jsme schopni získat novou akci.

3.9 Actor critic

Existují 2 základní přístupy – Value based a Policy based. Pomocí value based lze najít nejlepší možnou akci za předpokladu, že je jich konečné množství. Policy bysed je zase dobré pro stochastické a nedeterministické akce. Nevýhodou je však doba trénování.

Každá metoda má svou výhodu i nevýhodu. Algoritmus Actor Critic[12, 28] se snaží tyto dva přístupy spojit. Využívá 2 neuronové sítě. Jednu pro **Actor** a druhou pro **Critic**. Actor je založen na Policy based a vytváří chování agenta. Critic je value based a hodnotí akce, které agent udělal.

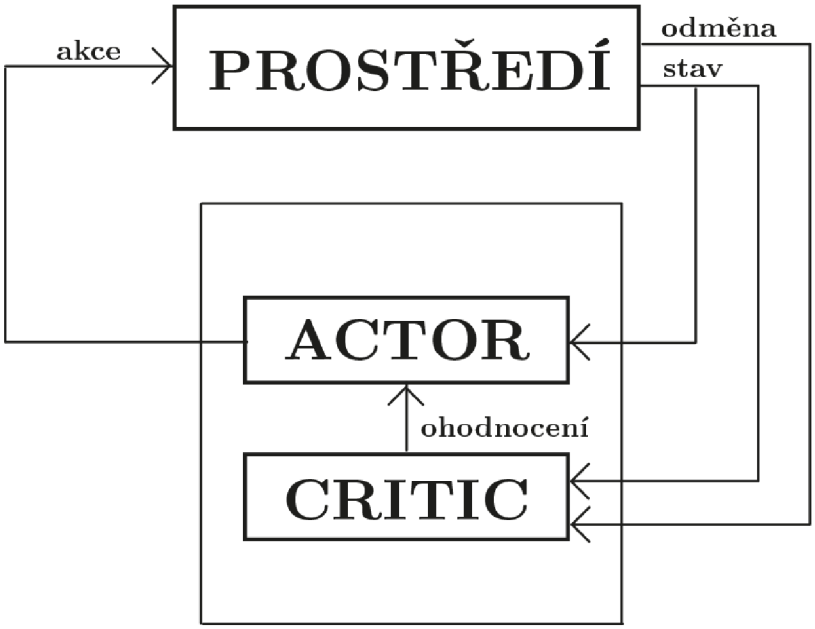
Problém u policy based algoritmů je, že trénování probíhá až na konci epizody a pokud v průběhu byla provedena špatná akce, tak to nemusí jít poznat, protože díky ostatním akcím se odměna zprůměrovala a i tak byla dost dobrá. Pro optimalizaci tedy potřebujeme obrovské množství vzorků. Tento problém se u algoritmu řeší pomocí Value based, tedy pomocí kritika, který hodnotí jednotlivé akce a trénování je tak rychlejší. Samotný algoritmus je také off-policy, tedy trénování probíhá i na minulých epizodách.

Samotné učení si lze představit jako proces, kdy Actor zvolí nějakou akci a kritik tuto akci ohodnotí ve smyslu: „Ano, toto je dobrá akce“, nebo „Ne, toto je špatná akce“. V samotném algoritmu se nepoužívá binární hodnota, ale číselná hodnota od záporných po kladné hodnoty. Pomocí této informace se agent postupně učí. Pro lepší představu je to zobrazené na obrázku 3.14, kde jsou 2 části (Agent a prostředí) a je tam zobrazen princip toku informací.

Actor

V algoritmu Actor Critic má Actor na starost rozhodování, vytváření akcí agenta. Při komunikaci mezi prostředím a agentem, agent využívá Actor, který vytváří akce. Actor je založen na policy based. Normálně se vytváří rozložení pravděpodobností jednotlivých akcí, tak jak normální policy based algoritmus. Průběh je zobrazen na obrázku 3.15. Na vstupu je stav, který jde do neuronové sítě, který stav zpracuje a jako výsledek vyhodí rozložení pravděpodobností akcí. V mém případě Actor vytváří 2 výstupy, směrodatná odchylka a rozptyl, tedy údaje pro normální rozložení, ze kterého se následně vybírá akce.

Actor u spojitých akcí může předpovídat rozptyl, ale ne standardní odchylku. Pokud ji máme definovanou, tak samotný algoritmus může být více stabilní[29].



Obrázek 3.14: Diagram znázorňující fungování algoritmu Actor Critic. Jsou tam 2 složky. Prostředí a Agent. Agent obsahuje Acotr a Critic. Actor je složka, která vytváří akce a komunikuje s prostředím. Kritik nekomunikuje přímo s prostředím, ale dává informace částí Actor o kvalitě akcí, které udělal.

Critic

Critic je část algoritmu hodnotící akce, které Actor udělal, a Actor se na základě této zpětné vazby učí. Critic je value based a výstupem je tedy 1 číslo, hodnocení. Proces jde vidět na obrázku 3.16. Na vstup přijde stav s akcí, poté to projde neuronovou sítí a jako výstup to dá kvalitu akce v daném stavu.

Tato část je volána/používaná při procesu učení. Algoritmus Actor-Critic je off-policy, tedy ukládá si data z jednotlivých epizod do paměti, kterou pak využívá pro trénování.

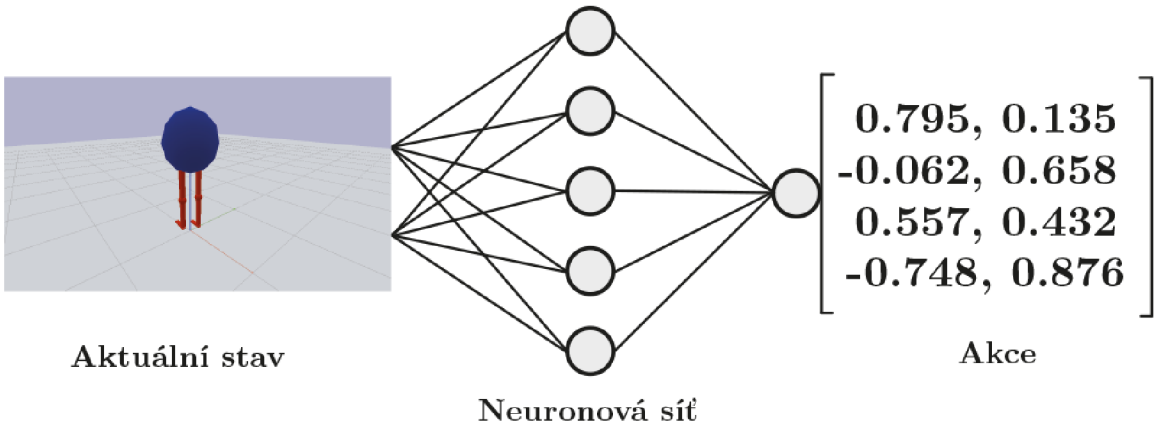
3.10 Soft Actor Critic

V mojí práci využívám algoritmus Actor Critic, konkrétně Soft Actor Critic (SAC)[2], který je definován na úlohy se spojitými akcemi. Cílem SAC není pouze maximalizovat celkové odměny, ale také maximalizovat entropy, což je míra náhodných akcí. To vede k dalšímu zkoumání stavového prostoru a může to později vést k urychlení učení. Může také zabránit, aby se nenašlo příliš brzy nějaké lokální maximum.

Entropy je definovaná jako:

$$H(P) = \mathbb{E}[-\log P(x)]. \tag{3.20}$$

SAC v jeden čase učí policy π_θ a 2 Q-funkce Q_1 a Q_2. Existují 2 verze. V té první se používá fixní hodnota regulačního koeficientu α, v té druhé se hodnota regulačního koeficientu mění v průběhu samotného učení.



Obrázek 3.15: Fungování Actor v algoritmu Actor Critic. V první části je úloha v nějakém stavu, ze kterého informace přecházejí do další části, neuronové sítě, která stav zpracuje. Ve třetí části je výstup neuronové sítě. Pro každou akci je to dvojice – rozptyl a směrodatná odchylka, tedy parametry pro normální rozložení. Inspirováno z [4].

Samotné učení probíhá na vzorcích získaných z jednotlivých epizod. Na začátku učení se vypočítá cíl Q-funkce pro minimalizaci budoucí chyby. Vypočítá se to jako:

$$y(r, s', d) = r + \gamma(1 - d)(\min_{i-1,2} Q_{\phi_{targ}}(s', a') - \alpha \log \pi_\theta(a'|s')), a' \ \pi_\theta(.|s'). \tag{3.21}$$

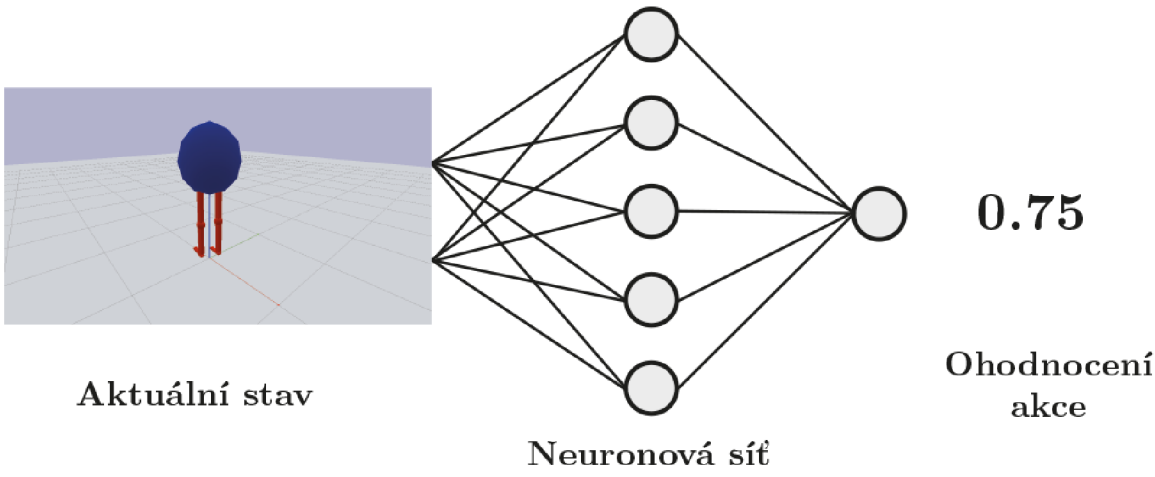
Následně proběhne aktualizace Q-funkce pomocí gradientního sestupu prvního řádu jako:

$$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} (Q_{\phi_i}(s, a) - y(r, s', d))^2, proi = 1, 2. \tag{3.22}$$

Po aktualizaci Q-funkcí je potřeba aktualizovat také policy π, které se aktualizuje pomocí gradient ascent jako:

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} (\min_{i=1,2} Q_{\phi_i}(s, a_\theta(s)) - \alpha \log \pi_\theta(a_\theta(s)|s)), \tag{3.23}$$

kde $a_\theta(s)$ je vzore z $\pi_\theta(.|s)$.



Obrázek 3.16: Fungování kritika v algoritmu Actor Critic. V první části je stav v prostředí, kde společně s akcí jde do druhé části, kde tyto informace zpracuje neuronová síť. Ve třetí části je výstup neuronové sítě, tedy ohodnocení akce v daném stavu. Inspirováno z [4].

Kapitola 4

Testovací prostředí

Testovací prostředí je nedílnou součástí posilovaného učení. Je to druhá část, bez které se učení neobejde. Je to ta část, se kterou agent komunikuje, zasílá jí vybrané akce. Prostředí zpátky posílá agentovi ohodnocení a také nový stav, do kterého se prostředí dostalo po provedení akce.

Prostředí si lze představit jako prostor, kde probíhají akce. U hry šachy to bude šachovnicové pole, které má šachové postavičky, u robota to může být reálný svět s robotem. Prostředí tedy nemusí být jen virtuální, ale i reálné, které však učení může ztěžovat a také zpomalovat. Pokud je ale úloha určená do reálného prostředí, je třeba ji v nějaké fázi otestovat.

Reálný svět nabízí spoustu překážek, a tak je jednodušší vytvořit si prostředí virtuální, ve kterém budou třeba zahrnuty všechny parametry reálného světa, jako třeba gravitace. Posilované učení se učí z úspěchů a neúspěchů. Naučit například robota dělat nějaké akce by mohlo být v reálném světě příliš nákladné, protože začátky učení jsou hlavně o neúspěších, které by robota mohly ničit po fyzické stránce a proces učení by se tak mohl zpomalit z důvodu oprav a obecně z fyzického opotřebení. Nemuselo by jít jen o úpravy, ale ladit samotné parametry modelu, kdy se mění velikost nějaké části, lze jednodušeji ve virtuálním prostředí než v tom reálném. Problém by byl i restart úlohy.

Dalším problémem by mohl být čas. Výhodou virtuálního prostředí je, že čas provádění simulace může být rychlejší než reálný čas, a tak model můžeme mnohem rychleji vytrénovat.

Jako prostředí jsem vytvořil prostředí s postavičkou, ve které jsou definované 4 úlohy s různými modifikacemi a rozšířeními.

4.1 Základ prostředí

Každá úloha posilovaného učení musí být definovaná v nějakém prostředí. Vždy záleží na konkrétní úloze, na požadavcích a parametrech úlohy. Z toho pak vyplývá, které nástroje jsou potřeba. Posilované učení již není příliš neznámé, a tak existuje pro tyto úlohy spoustu nástrojů ulehčující základní práci.

Algoritmy jsou mnohdy univerzální a je dobré tedy používat i nástroje na tvorbu prostředí. Tyto nástroje umožňují vytvořit univerzální rozhraní pro komunikaci mezi agentem a prostředím. Mezi nejznámější a také nejpoužívanější patří Gym od OpenAI[1]. Gym jako nástroj nestačí, nabízí jen rozhraní, které zjednodušuje komunikaci s agentem pro tréno-

[1] https://gym.openai.com/

vání. V mém případě je úlohou postavička ve fyzikálním modelu. Pro tuto úlohu je tedy potřeba prostředí. K těmto potřebám poslouží PyBullet[2].

4.2 OpenAI Gym

OpenAI Gym[8] je nástroj nabízející rozhraní pro definování nových úloh nebo jen využití těch stávajících, které jsou v Gym implementované. Tento nástroj vyvíjí organizace Open AI obecně zaměřená na vývoj umělé inteligence. Samotný Gym už obsahuje spoustu předem nadefinovaných prostředí. Jeho součástí jsou jednoduché textové úlohy, staré Atari hry, úlohy jako cartpole a mountaincar, a také různé 3d pohyblivé příšerky. Všechny tyto úlohy lze použít a vytvořit na ně algoritmus, který danou úlohu vyřeší.

Gym mimo jiné umožňuje vytvářet i vlastní prostředí. Každé prostředí se skládá ze sady určitých úkonů, které je potřeba definovat a prostředí je hotové. Jde o inicializační funkci **init**, kde probíhá inicializace velikostí akcí a prostředí. Je zde také inicializace prostředí, které se dále používá, například inicializace fyzikálního modelu. Další je **step**, což je funkce, která se volá v průběhu trénování pro posun. Jako parametr bývá akce, kterou je třeba udělat. Dále pak **reset**. Tato funkce se volá před začátkem trénování a po každé epizodě tak, aby trénování mohlo začít vždy ve výchozím stavu. Dále je možno použít funkci **render** vracející grafické zobrazení aktuálního stavu prostředí.

Proč Gym

Důvod, proč Gym a ne vlastní řešení, je jednoduchost. Stačí nadefinovat těla pár metod, zaregistrovat třídu tak, aby o něm Gym věděla a je to. Pak už stačí jen spustit trénování. Agent si může z prostředí získat potřebné informace, které se při vytváření prostředí nadefinují, a které agent potřebuje ke svému běhu, například jakou dimenzi mají data prostředí a jaké jsou akce.

Dalším důvodem, proč využít Gym je, že při inicializaci prostředí stačí jen změnit název úlohy implementované v Gym a trénování pak může probíhat na nové úloze. Vše by mělo fungovat, protože Gym nabízí obecné přístupy.

4.3 Pybullet

PyBullet[11] je fyzikální simulátor pro hry, vizuální efekty a také pro robotiku a posilované učení. Je to postavené na Bullet Physics SDK, které je napsané v C++ a python verze jen volá funkce pomocí API, které je napsané v C++. Pro vývoj tedy lze použít oba jazyky. V pythonu se jedná o jednoduchý modul, který lze jednoduše nainstalovat jako pip balíček a začít hned používat.

Nástroj je určen k simulaci a umožňuje provádět akce nad modely, které jsou do prostředí vloženy. Příklad akce může být roztočení kola, pohyb končetiny atd. Prostředí také umí dát informaci například o kolizích mezi objekty. V samotném prostředí jsou již předdefinované nějaké modely, například humonoid, ramena, které přenáší objekty atd.

Kromě simulace nástroj nabízí i renderování, a to pomocí TinyRender a OpenGL, tedy zobrazení scény do okna. Dále nabízí také debug nástroje, jako zobrazování textu do scény a čar, které se využijí například při aplikaci externích sil vyvinutých na modely.

[2] https://pybullet.org/

Pybullet jako fyzikální simulátor může fungovat ve více modech. První mod je klasický, kde se bude vykreslovat scéna. Jedná se o mod GUI. Další možností je mod DIRECT. Jde o mod, kdy se nic nevykresluje a díky tomu může celá simulace proběhnout rychleji. Je to ideální pro posilované učení, kde nás přímo nezajímá, jak scéna vypadá, ale jak funguje. A na to nám stačí tento mod. Existuje ještě pár modu, kde je například oddělená simulace a renderování na separátní procesy.

Alternativou k PyBullet je Mujoco[3]. Jedná se o pokročilý simulátor, vyvinutý na University of Washington a je pod licencí, která se platí ročně. V Open AI k němu vytvořili i Python wrappper pro použití v pythonu.

Formáty a způsoby definice objektů

Definovat objekty v Pybullet lze pomocí několika formátů – SDF, URDF, UCJF. Každý formát je lehce specifický a má jiný styl zápisu, všechny jsou ale odvozené od XML. Výsledek by měl by být ze všech formátů stejný, jde tedy hlavně o to, jakým způsobem lze objekty vytvořit. Na výsledném formátu tolik nezáleží, spíše na nástrojích, které zjednoduší vytvoření modelu.

UJCF je formát Mujoco, tedy alternativního fyzikálního modelu, který však lze použít i v PyBullet. Díky Mujoco licenci jsem se tímto formátem nezabýval.

Pro formát **SDF** už existuje nástroj pro modelování. Jedná se o nástroj Gazebo[4], který umožňuje modelování a také testování algoritmů na modelech v různých prostředí. Modelovat v něm lze hlavně jednodušší modely a nedovoluje modelovat postavu ze složitějších tvarů. Narážel jsem na problémy s editací již vytvořených modelů, které nešly editovat a bylo vždy třeba vytvořit nový. Samotný nástroj poskytuje také možnosti simulace.

Další možností je **URDF** formát, který jsem nakonec použil. Nabízí se zde dvě možnosti definice. Ta první je pomocí blenderu[5]. Jedná se o 3D modelovací nástroj, kde lze vytvořit 3D postavu a tu pak pomocí nástroje Phobos[6] převést na URDF model. Vytvořený model se v blender rozdělí na jednotlivé části, přidají se místa ohybů, nadefinují podmínky pro motory a model se vygeneruje. Jedná se tedy o nejjednodušší cestu, jak složitý model převést do Pybullet.

Další možností u **URDF** je psát přímo XML. To je samo o sobě složité kvůli závislostem a správným výpočtům. Existuje však formát XACRO[7], kde lze vytvářet různé makra, které lze postupně volat a provádět výpočty. Při generování se vše vypočítá, importuje a jako výsledek nám vznikne URDF, které pak už jen stačí načíst v pybullet. Jedná se o způsob, jak mít každý řádek definice modelu pod kontrolou a zároveň mít s pomocí maker ulehčenou práci.

Způsoby definice kloubů

Každý model se skládá z částí definujících se pomocí tagů *link*. V těle tagu se dále nastavuje tvar, kolizní prostor a také umístění elementu v prostoru. Jako tvar může být buď kvádr, koule, případně i objekt ve formátu obj, který se do souboru importuje. V případě objektu se může jednat o jakkoliv složitý tvar.

[3] http://www.mujoco.org/
[4] http://gazebosim.org/
[5] https://www.blender.org/
[6] https://github.com/dfki-ric/phobos/
[7] http://wiki.ros.org/xacro/

Model obsahuje jednotlivé části, které musí být něčím spojeny. U URDF souborech se to dělá pomocí tagu *joint*[1] a druhů spojení je více. Rozdíl je v povolených pohybech. Mezi nejzákladnější patří **fixed**, **revolute** a **continuous**, který jsem však nepoužil.

- **Fixed** – spojení, které udělá daný motor nehybný. Aplikovaná akce s tímto spojením nic neudělá. Používá se pro spojení, kde nevyžadujeme pohyb a potřebujeme, aby 2 části držely u sebe.
- **Revolute** – Spojení které umožňuje pohyb ve dvou osách, tedy například pro pohyb končetiny v jedné rovině. Jako parametr se zde nastavuje minimální a maximální úhel, ve kterém se motor (kloub) může nacházet.
- **Continuous** – Něco jako Revolute, akorát bez minimálního a maximálního úhlu. Využití je pro kolo.

Funkce v Pybullet

Obsah všech funkcí obsažených v pybullet, včetně aplikace akcí a získávání informací o modelu, popisuje dokumentace[11]. Zmíním ale jen pár základních, které jsou pro běh nejdůležitější.

Načítání modelu – Pro načítání modelu a nebo jeho prostředí je zde funkce *loadURDF()*, která má své alternativy pro ostatní formáty. Pomocí této funkce se načte model a převede se do vnitřní reprezentace pybullet

Pohyb – Pro pohyb, vykonání akce, je zde funkce *setJointMotorControl2()*, která na zvolený motor aplikuje požadovanou akci. Akce může být pozice, do které se má motor dostat, síla, kterou má působit a nebo rychlost. V URDF je motor definován pomocí tagu *joint*. Jako parametry má nastavené krajní hodnoty, které lze případně nastavit i přímo pomocí pybullet. Akce se nemusí na motory aplikovat zvlášť, ale pomocí pole hodnot. Stačí zadat indexy a síly pro jednotlivé motory a o zbytek se už postará pybullet. Jedná se o funkci *setJointMotorControlArray()*.

Informace modelu vzhledem ke světu – Pro trénování jsou potřeba informace o světě jako kde se model nachází, v jakém úhlu je, jakou rychlostí se pohybuje atd. Na toto jsou funkce *getBasePositionAndOrientation(), getBaseVelocity()*.

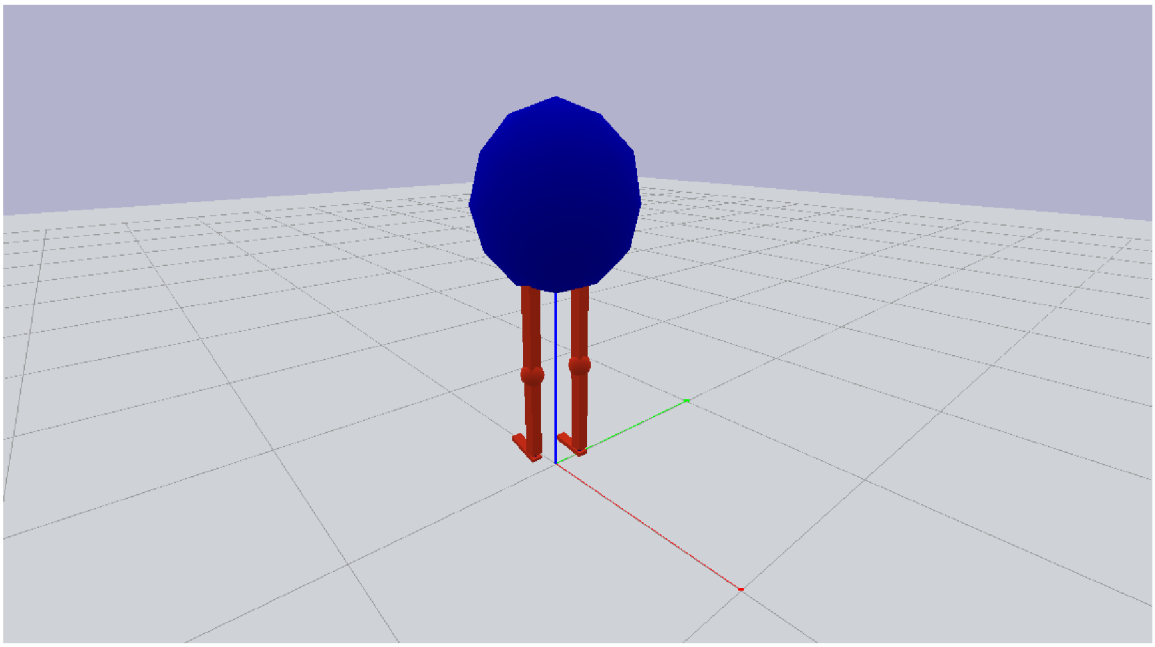
Informace modelu – Agent potřebuje nejen informace o světě, ale také o samotném modelu. Pomocí funkce *getLinkState()* lze získat informace o jednotlivých částech modelu. Informace o motoru, v jaké pozici se motor nachází, slouží funkce *getJointState()*.

Externí síly – Pybullet obsahuje funkce pro aplikaci i externích sil, což může být využito jako rozšíření pro jednotlivé úlohy v prostředí. Funkce pro externí sílu je *applyExternalForce()*. Jako parametry jsou z jakého směru síla působí a také jakou silou.

4.4 Postavička

Pro trénování jsem se rozhodl vytvořit postavičku, která v různých nastaveních bude plnit zadané úlohy. Samotná postavička se skládá z několika částí, které tvoří tělo a pak také nohy. U návrhu předpokládám, že návrh je správný a postavička bude schopná plnit dané úlohy. Tento předpoklad nezaručuje, že neexistuje lepší, ale že je schopná. Postavička je

definovaná pomocí formátu XACRO, který jsem zmiňoval výše, a generuje se do URDF formátu. Výsledná postavička je na obrázku 4.1.



Obrázek 4.1: Postavička z vytvořeného prostředí. Postavička obsahuje tělo, ze kterého vycházejí 2 končetiny. Každá končetina se skládá ze 3 hlavních částí – chodidla, spodní část nohy a horní část nohy. Každá část je oddělená od druhé kloubem. Postavička umožňuje dohromady 8 různých akcí (pohyb končetin dopředu, dozadu a do boku).

Přesná definice toho prostředí není, protože se může měnit na základě úloh. Základ je však stejný a hlavně se mění jen nastavení jednotlivých parametrů.

Z čeho se skládá postavička

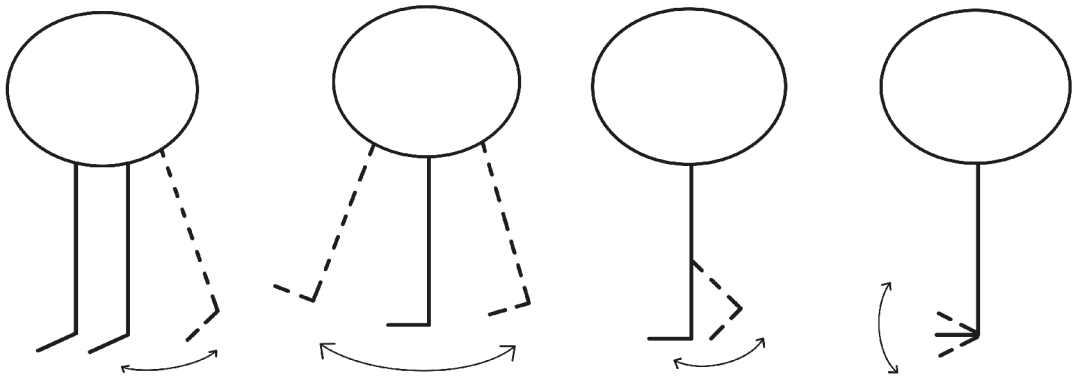
Postavička se skládá z 13 částí. Základem je tělo, které je tvaru koule. Poloměr koule je 0,3 m a má hmotnost 10 kg. Z těla vedou 2 končetiny, které jsou totožné a slouží k pohybu celé postavičky.

Samotná končetina se skládá ze 3 částí, které jsou vždy oddělené klouby. Horní a dolní část končetiny jsou stejné a jedná se o kvádr s rozměry 0,05x0,05x0,4 m a hmotností 3 kg. Chodidlo je také kvádr s rozměry 0,17x0,05x0,02 m a hmotností 1 kg. Každý kloub má definovaný pohyb pouze do jednoho směru, tedy například pohyb končetiny dopředu a dozadu. Pokud v jednom místě jsou potřeba 2 pohyby, jsou definované na jiných částech.

Akce - jaké jsou a jak to funguje

Postavička má 8 motorů, kloubů, se kterými se pohybuje. Každý motor má nastavenou konstantní sílu, kterou končetinu posunuje. Síla se násobí se vstupní informací. Ta říká, jak má být síla veliká a také, jakým směrem se má pohnout. Každý motor má svůj rozsah, ve kterém se může pohybovat. Toto je zobrazené na obrázku 4.2. Jedná se o číslo v rozsahu od -1 do +1, které se násobí s konstantní sílou. Pokud je číslo záporné, tak se posunuje končetina dozadu, pokud je kladné, tak dopředu. Pro každý motor je potřeba tato informace zvlášť,

na vstupu je tedy pole 8 hodnot. Po vynásobení vstupu a sil se zavolá funkce z pybullet, která pohyb vykoná.



Obrázek 4.2: Rozsahy pohybu jednotlivých částí končetin, které postavička obsahuje a se kterými může pohybovat. Umožňuje hýbat s celou končetinou dopředu a dozadu. Může s ní hýbat také do boku. Další pohyb vychází z poloviny končetiny (koleno), kde je umožněn pohyb dozadu a zpět. Nejspodnější část, chodidlo, umožňuje pohyb nahoru a dolů.

Observation - informace pro agenta

Samotné prostředí poskytuje spoustu informací, ale jen některé jsou potřebné pro samotné trénování. Více informací může způsobit větší citlivost a tím lepší trénování. Některé parametry však mohou způsobovat pravý opak. Informace z prostředí jsem si rozdělil do několika kategorii:

- **Informace vzhledem k postavičce** – Prostředí nabízí spoustu informací o postavičce. Některé jsou zbytečné a některé málo relevantní. U postavičky mě zajímá její náklon, jaký má postavička vzhledem k povrchu. Jedná se o ukazatel pádu. Dále mě zajímá rychlost postavičky, z souřadnice (vertikální osa) postavičky, tedy v jaké výšce se postavička nachází. Jedná se o další ukazatel pádu a také zda postavička vyskočila. Souřadnice postavičky mě nezajímají, protože se jedná o údaj, absolutní hodnotu, která nám nedává informaci o okolí. Kdyby se start posunul v některé z os o nějakou vzdálenost, tak by natrénovaný model mohl přestat fungovat.

- **Informace o postavičce** – U samotné postavičky mě zajímají hlavně končetiny, kde zjišťuji relativní pozici končetiny, kterou převádím na informaci, v jaké pozici se nachází vhledem k rozsahu pohybu končetiny. Obsahuje také informaci, zda se chodidla dotýkají povrchu či nikoliv.

- **Dodatečné informace** – Některé úlohy pracují s cílem. Jedná se o bod v prostředí. Jelikož zadávat přesné souřadnice není dobré, tak jsem informaci o cíli zadával pomocí 3 parametrů. První parametr je vzdálenost od postavičky k bodu. Další dva parametry jsou vzdálenost na osách X a Y.

Nastavení fyzikálního modelu

Svět, ve kterém žijeme je spojitý. Svět v pybullet je diskrétní, a tak je potřeba si nadefinovat, jak často bude probíhat dotazování na nové akce. Ve výchozím stavu je v pybullet nastaven

krok simulace 1/240 s, kvůli posilovanému učení jsem to snížil na 1/30 s pro jednodušší a rychlejší trénování.

Další parametr je gravitace. Je zde tedy potřeba definovat požadované zrychlení, v mém případě jsem nastavil $9,8ms^{-2}$, tedy lehce zaokrouhlená hodnota normálního tíhového zrychlení[32]. Ostatní parametry jsou ve výchozím stavu a měly by co nejvíce simulovat reálný svět.

4.5 Další části prostředí

Pro testování není potřeba jenom hlavní model, v mém případě postavička, jsou ale potřeba i ostatní prvky, které umožňují tvořit jednotlivé úlohy.

Povrch prostředí

Pybullet je prostředí, které nabízí fyzikální prostor, ve kterém se může testovat nejen pohyb po povrchu, ale i ve volném prostoru. Je tedy potřeba definovat povrch, po kterém se bude postavička pohybovat. Základní povrch lze nadefinovat pomocí obrazců, jako kvádr, obdélník. Složitější povrch lze pomocí .obj souborů, díky čemuž lze nadefinovat různě vlnitý a tedy i obtížný povrch, který postavička musí zdolat. V mém případě jsem zvolil čistě kvádr, který jsem nadefinovat v Xacro převedený do URDF. Velikost kvádru je pro mě velikost testovacího prostředí.

Povrchu prostředí jde nastavit různé parametry, jako například kluzkost.

Cíl

Součást některých úloh je bod, kam se má postavička dopravit. Tento element se inicializuje vždy při restartu prostředí a to na náhodnou pozici. Pozice je však ošetřená, aby se cíl nevyskytoval příliš blízko a také, aby se nacházel v testovacím prostředí. Souřadnice jsou uložené v prostředí, kde se mimo jiné zjišťuje, zda postavička dorazila k cíli. Cíl je nadefinován pomocí formátu Xacro a převeden do formátu URDF. Jako obrazec byla zvolena koule o poloměru 0.5 m.

4.6 Úlohy v prostředí

Samotné prostředí neobsahuje pouze postavičku s jednou úlohou, ale soubor úloh, které se volí při vytváření prostředí před spuštěním. Vše vychází ze stejného základu, liší se ale funkcí odměn, ukončování epizody atd.

Všechny úlohy obsahují stejnou podmínku a to, zda se postavička nenachází pod nějakým úhlem, který vede k pádu. Další podmínka je, zda tělo postavičky není příliš nízko, což také značí pád. Pokud tedy jedna z těchto podmínek platí, tak se ukončí daná epizoda.

Funkce odměn je obecně definovaná jako:

$$r = r_{alive} + r_{progress} \tag{4.1}$$

Stání

Nezákladnější úloha, kdy má postavička za úkol najít postoj, ve kterém vydrží na místě a nespadne. Na začátku je postavička vygenerovaná na souřadnicích [0, 0] a nesmí se pohnout na jakoukoliv stranu o 0.5 m. Jedná se o prostor, na kterém by měla postavička stát.

Pokud se dostane pryč z prostoru, tak epizoda končí. Odměna za každou akci je konstantní, konkrétně hodnota 2. Matematicky se odměna vyjádří jako:

$$r_{alive} = 2. \tag{4.2}$$

Skákání na místě

Úloha skákání na místě je velmi podobná stání. Na vymezené ploše má za úkol skákat na místě, nevypadnou z plochy a získat co největší odměnu.

Podmínka o ploše je zde podobná. Postavička nesmí překročit na jakoukoliv stranu 0.5 m. Odměna už je složitější. Je potřeba, aby postavička skočila do určité výšky a pak zase zpátky. Takto aby se to cyklicky opakovalo. První část odměny je výškový posun, zda se postavička posunula ve výšce. Druhá část odměny za dosažený bod. Na začátku je nastavena odměna za výskok do určité výšky s hodnotou 200. Po dosažení se tato odměna zruší a nastaví se odměna za dopad, dotknutí oběma chodidly země. Po získání této odměny se odměna zase přepne. Takto to probíhá cyklicky do konce epizody.

Pro lepší odezvu agent dostává i odměnu, když se pohybuje ve správném směru, tedy pokud je postavička na zemi, pak dostává odměnu za posun nahoru. To samé platí, když je cíl na zemi. Pokud se postavička vydá do nesprávného směru, dostává zápornou odměnu.

Odměnu pro skok nahoru lze vyjádřit jako:

$$r_{progress} = |z_{aktualniZVyska}| - |z_{minulaVyska}|. \tag{4.3}$$

Pro dotknutí země je odměna výpočet opačný:

$$r_{progress} = |z_{minulaVyska}| - |z_{aktualniZVyska}|. \tag{4.4}$$

Chůze

V této úloze je za cíl naučit postavičku chodit, libovolným směrem i rychlostí. Kladen je důraz na posun v prostředí. Epizoda končí, když je splněna podmínka indikující pád, další možnost je, pokud dojde na konec prostředí.

Funkce odměn obsahuje 2 části. První část je o posunu od minulé pozice (uražená vzdálenost). Druhá část je o tom, o kolik se posunula od bodu, kde epizoda startovala. Důvod této odměny je ten, aby se postava posunovala nějakým směrem a netvořila „pohyb“ stání na místě. Funkce odměn se matematicky vyjádří jako:

$$\begin{aligned} r_{progress} &= r_{relativniPoposun} + r_{absolutniPosun} \\ r_{relativniPoposun} &= \sqrt{(x_{minule} - x)^2 + (y_{minule} - y)^2} \\ r_{absolutniPosun} &= \sqrt{x^2 + y^2} \end{aligned} \tag{4.5}$$

Chůze s cílem

Chůze s cílem je rozšířená verze pouhé chůze. Cílem této úlohy je dorazit k cíli, který je v prostředí generovaný na náhodné pozici.

Základ odměny je stejný. Uražená vzdálenost od minulého stavu. Další část odměny však tvoří posun směrem k odměně vzhledem k minulé pozici, tedy zda se postavička přibližuje či oddaluje. Po dosažení cílového bodu je stanovena odměna s konstantní hodnotou 50 a zároveň je ukončena epizoda.

Matematicky lze odměnu vyjádřit jako:

$$\begin{aligned} r_{progress} &= r_{relativniPoposun} + r_{posunKCili} \\ r_{relativniPoposun} &= \sqrt{(x_{minule} - x)^2 + (y_{minule} - y)^2} \\ r_{posunKCili} &= \max(d_{minulaVzdalenostKCili} - d_{aktualiniVzdalenostKCili}, 0). \end{aligned} \tag{4.6}$$

Rozšíření

U každé úlohy je možné zapnout rozšíření, které náhodně po dobu epizody a náhodnou velikostí síly v náhodném úhlu působí silou na postavičku, kterou se snaží rozhodit v prováděné úloze. Jedná se o prvek, který dokáže vylepšit trénovací funkci tak, aby byla odolná i vůči náhodným silám.

Je možné zvolit modifikaci samotného modelu. První modifikace je o změně velikosti nohy. Lze ji zmenšit nebo zvětšit. Změna velikosti je vždy o 50% původní velikosti. Druhá modifikace je o změně velikosti těla. Jednou je široké a krátké. Podruhé je modifikace opačná, tělo je dlouhé a úzké.

Kapitola 5

Implementace

Implementace celého řešení je dělána v pythonu a algoritmus, který se využívá pro trénování a také pro experimenty je implementovaný s pomocí knihovny PyTorch[22]. Pytorch je framework pro strojové učení od Facebooku.

Celé řešení obsahuje 4 části, které spolu navzájem nějakým způsobem komunikují.

5.1 Prostředí

Prostředí je definované v AI gym a v projektu reflektuji požadovanou strukturu. Vše se nachází ve složce *customenvs*, která se dále větví na *envs* a *resources*. Ve složce *envs* je soubor monster_env.py, ve kterém je požadovaná struktura. Tento soubor představuje prostředí, komunikuje s agentem a také s fyzikálním modelem. Ve složce *resources* je složka *monster*, ve které se nacházejí všechny potřebné soubory pro fyzikální model. Jedná se vždy o dvojici, definici v *URDF* formátu a soubor, který se stará o načtení a komunikaci. Je zde definice postavičky, povrchu a cíle.

Mezi těmito soubory je i třída prostředí sloužící pro lepší komunikaci, která se však pak převádá na pole hodnot.

5.2 Algoritmus

Implementace algoritmu Actor Critic, kterou v projektu využívám, je ve složce *ActorCritic*. Jedná se o algoritmus inspirovaný z github[15], který je však upraven pro mé použití. Model je definovaný v souboru *model.py* a obsahuje 2 třídy – Actor a Critic.

Actor využívá 2 vrstvy linear vstvy, které se pak využívají na predikci směrodatné odchylky a rozptylu, tedy predikce údajů pro predikci akcí pomocí normálního rozložení. Jedná se také o linear vrstvy. Velikost skryté vrstvy je 256. Velikost výstupu obou vrstev je počet akcí, které prostředí požaduje. Critic je jednodušší a obsahuje 3 linear vrstvy, které vrací 1 výstup v podobě kvality akcí.

Hlavní částí je soubor Agent, který se stará o vytváření nových akcí a také o zlepšování modelu. Nové akce má na starost Actor. V učení pomáhají Critic vrsty, kterou jsou tu 4. Všechny tyto vrstvy obsahují optimalizátor Adam s leargning rate 0,001. Pro učení je využívána paměť, která je implementovaná v *memory.py*

Při updatu, kde jako parametr je počet opakování učení, se načtou data z paměti a pak 4 critic vrstvy se postarají o učení. Na začátku se použijí 2 critic pro výpočet cílové Q-hodnoty. Z nich se vybere ta pesimističtější varianta, tedy ta horší. Dále se použijí další

2 critic sítě, které se s použitím hodnot z první části použijí pro učení critic sítě. Na základě takto získaných informací se aktualizuje Actor. Následně pak proběhne i aktualizace prvních dvou critic sítí, které předpověděly cílovou q-hodnotu.

5.3 Soubory pro běh

Samotný běh zajišťuje více souborů. Ten první je *main.py*, který přímá parametry, vybírá algoritmus pro běh atd. Celkově chystá prostředí pro běh. Tento soubor spouštění *run.py*, který má na starost běh, tedy zprostředkování komunikace mezi agentem a prostředím. Dalším důležitým souborem je *parameters.py*, který obsahuje parametry algoritmu pro učení.

Parametry spouštění programu

- **--mode** – Výběr modu pro spuštění (test/train)
- **--render** – Zda se bude vytvářet video z prostředí
- **--visualize** – Zapnutí GUI modu ve fyzikálním modelu, tedy zobrazení okna grafickým zobrazením prostředí
- **--algorithm** – Zvolení algoritmu pro trénování
- **--env-type** – Zvolení úlohy v prostředí
- **--extension** – Rozšíření k trénování, aplikace externích sil na model
- **--modification** – Modifikace modelu, postavičky
- **--load** – Načtení již předtrénovaného modelu

5.4 Spouštění prostředí

Každá úloha, která je v prostředí nadefinovaná se volí na začátku při inicializaci prostředí. Jednotlivé úlohy se spouští pomocí nastavení proměnné s názvem prostředí. Pro Stání je to *standing*, pro skákání na místě *jumping*, pro chůzi *walking*, a pro chůzi s cílem *target_walking*.

Prostředí obsahuje i rozšíření. To se nastavuje také při inicializaci. Pro externí síly je to *external_forces*.

Posledním parametrem při inicializaci je informace, zda chceme zobrazit scénu, případně renderovat video. Ve výchozím stavu je tato volba vypnutá, protože to zpomaluje trénování a není potřeba vidět každou epizodu.

5.5 Grafy

Poslední část je na generování grafů. Při běhu programu vzniká log soubor, ze kterého lze vygenerovat graf. Jako parametr soubor bere soubory, které se mají použít pro vytvoření grafu. Co soubor, to část grafu. Velikost grafu se bere podle nejmenšího z nich. Další parametr je legend, kde se zapisují hodnoty do řetězce oddělené středníkem, které se použijí pro generování legendy.

Kapitola 6

Experimenty

Pro své experimenty jsem zvolil prostředí, které jsem pro tyto účely vytvořil. Součást řešení jsou i soubory zajišťující komunikaci mezi prostředím a agentem. Cílem experimentů bylo zjistit, zda implementovaný algoritmus Soft Actor Critic dokáže natrénovat model v jednotlivých úlohách, ve kterých je nadefinovaná funkce odměn. Natrénovaný model by měl zvládnout požadovaný pohyb. Například v úloze stání zvládnout stát na místě.

Parametry algoritmu byly obecně nastavené s velikostí paměti na 100000 vzorků a trénovalo se vždy po 256 vzorcích z paměti. Parametr učení byl nastaven na 0,001. S hodnotou gamma (discount faktor) probíhalo více testů, kde nejlepších výsledků bylo dosaženo s hodnotou 0,99. Při nižších hodnotách discount faktoru měl algoritmus problém natrénovat model tak, aby dával přijatelné výsledky.

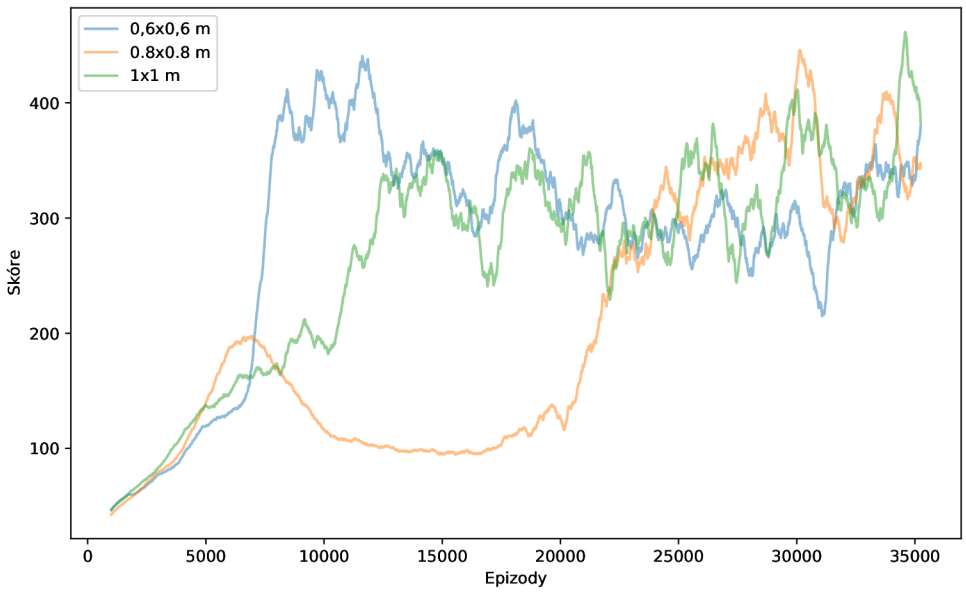
6.1 Stání

První, nejzákladnější úloha, kde je cílem zjistit, zda postavička v prostředí dokáže udělat základní pohyb, a to stání na místě bez toho, aby se nějak více pohnula. Jedná se o základní úlohu, již lze využít v rozšířených úlohách, kde bude například nějaká trasa, která se bude postupně generovat. Pokud zrovna nepůjde kde stoupnout, postavička se bude muset zastavit a stát na místě než bude moct znovu pokračovat.

Na obrázku 6.1 lze vidět graf, kde probíhalo trénování na různé ploše. Základní plocha, kde má postavička stát, je plocha o velikosti 1x1 m (zelená křivka). Druhá křivka je z prostoru 0,8x0,8 m (žlutá křivka), a třetí z 0,6x0,6 m (modrá křivka). Z dlouhodobého hlediska se nejvíce dařilo na největší ploše, kde model dosahoval nejvyšší odměny a postavička byla schopná ustát nějakou dobu na místě. Nejlepší začátek měla postavička na malé ploše (modrá křivka), kde se algoritmu začalo dařit velmi rychle. Zajímavá křivka je prostřední, tedy střední plocha, kde algoritmus měl ze v průběhu mírný výpadek, který však v průběhu dohnal.

6.2 Široké vs dlouhé tělo

V tomto experimentu jsem využil upravené prostředí, jež měnilo velikost postavičky, ukázka jde vidět na obrázku 6.2. Velikost těla je 1x0,25x0,2 m. Cílem experimentu bylo zjistit, zda algoritmu dělá problém, když je tělo větší ve směru jakým probíhá hlavní pohyb, tedy dlouhé a úzké a nebo, jestli mu více sedí tělo široké a krátké.



Obrázek 6.1: Graf ukazující průběh trénování stání postavičky na různé velikosti ploch. Dle grafu jde vidět, že se postavičce nejlépe dařilo na větší ploše. Na menších plochách se jí ale nedařilo hůř.

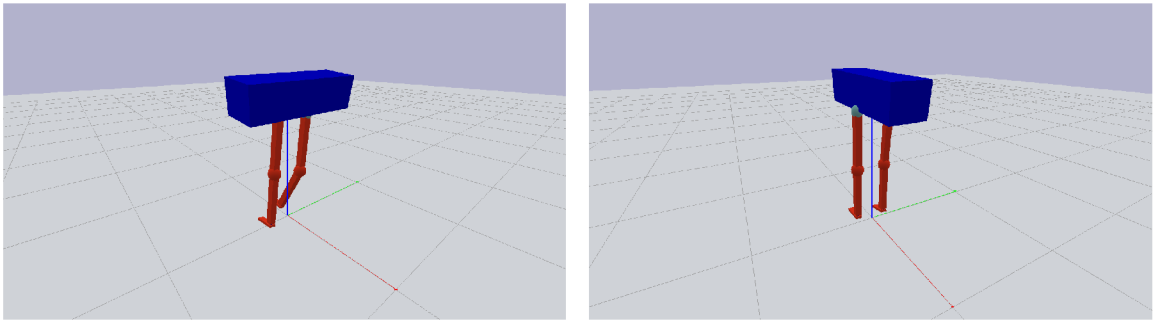
Pro zjednodušení je plocha na trénování ve velikosti 1x1 m. Samotné trénování jde vidět na obrázku 6.3, kde je porovnání postavičky, která je široká a krátká, a úzká a dlouhá. Dle grafu lze pozorovat, že se lépe dařilo postavičce, která byla široká a krátká, měla ale větší výkyvy a pomalu se obě modifikace začaly srovnávat. Důvod proč široká a krátká, že hlavní pohyb probíhá dopředu, případně dozadu a tímto směrem není nic, co by pomáhalo k rychlejšímu pádu.

6.3 Velikost nohy

Další experiment se zaměřoval na velikost chodidla. Zda měla velikost chodidla, zmenšení, případně zvětšení chodidla efekt, ať už pozitivní či negativní. Cílem experimentu bylo zjistit, jestli efekt je znatelný, a zda se vyplatí chodidlo zvětšit, zmenšit, či je v optimální velikosti.

Chodidlo je ta část nohy, které se dotýká podložky a může tak mít vliv na stabilitu celé postavičky, protože styčná plocha je větší a rychlost učení tedy může být větší.

Z grafu 6.4 lze vidět porovnání trénování postavičky s různou velikostí chodidla, které se zvýšilo nebo snížilo o 50% oproti normálu. Zelená křivka, která reprezentuje největší chodidlo, dosahovalo nejvyšších výsledků. Modrá a žlutá křivka byla ze začátku velmi podobná, v průběhu malé chodidlo začalo ztrácet na ostatní, pomalu se ale povedlo dostat na hranici normálního chodidla.



Obrázek 6.2: Ukázka širokého a dlouhého těla, které se používá pro experiment, kde je cílem zjistit, zda trénování probíhá rychleji u širokého a krátkého těla nebo u dlouhého a úzkého těla.

6.4 Schopnost skákat

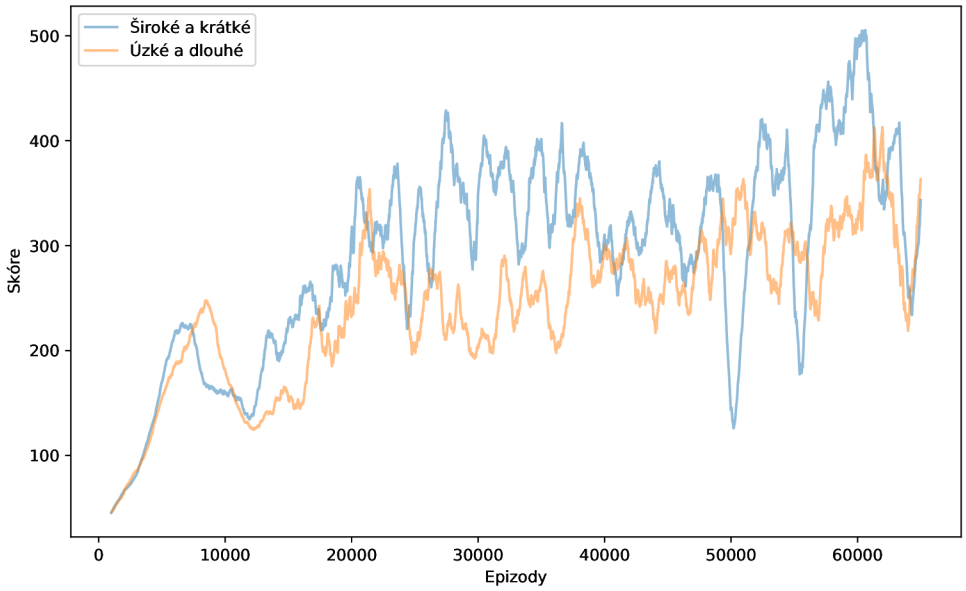
Schopnost skákat je už složitější úloha. Cíl je, aby postavička dokázala být na místě a zároveň vyskočila, tedy změnila svou polohu ve vertikálním směru.

Na obrázku 6.5 je zobrazen průběh trénování skoku, kdy funkce odměn byla založena čistě na dosažené výšce. Průběh samotného trénování měl výkyvy, ale model byl schopen se postupně učit. V průběhu experimentu se trénování dařilo a postavička skočila do požadované výšky a vrátila se zpět. Vyskočit znovu už nezvládla. Postupně však měla postavička tendenci jít pouze nahoru a nejednalo se pak o skákání, ale o velký skok do výšky bez návratu. Pokud se upravila funkce odměn na dosažení horního bodu a pak zpátky za dotyk země, tak trénování nedosahovalo příliš dobrý výsledků.

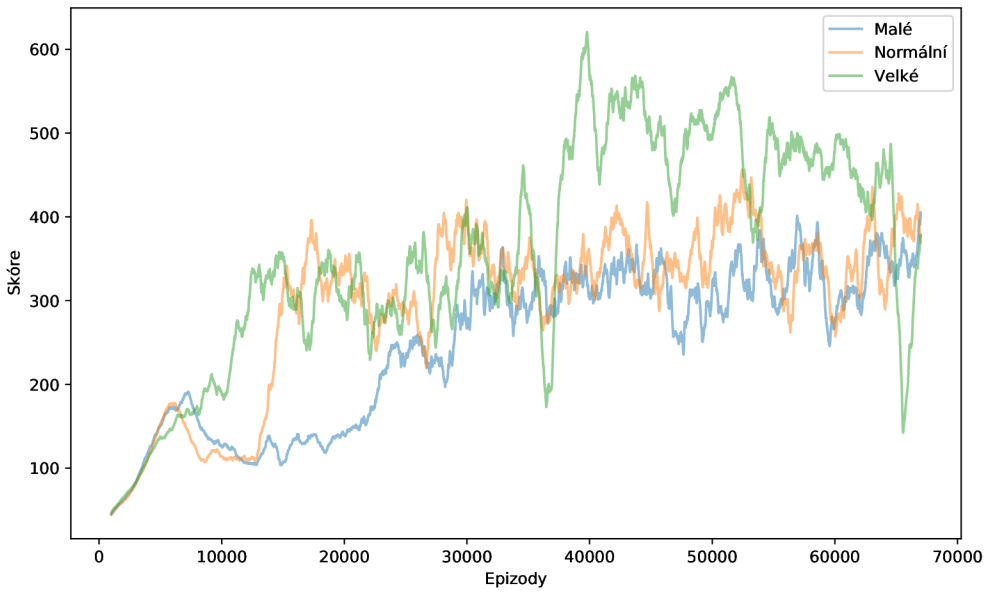
6.5 Chůze

Další experiment se zaměřuje na chůzi, tedy na posun z počátečního místa v jakémkoliv směru, tak aby postavička nestála a měnila svoji pozici. Pohybovat se může kdekoliv po ploše.

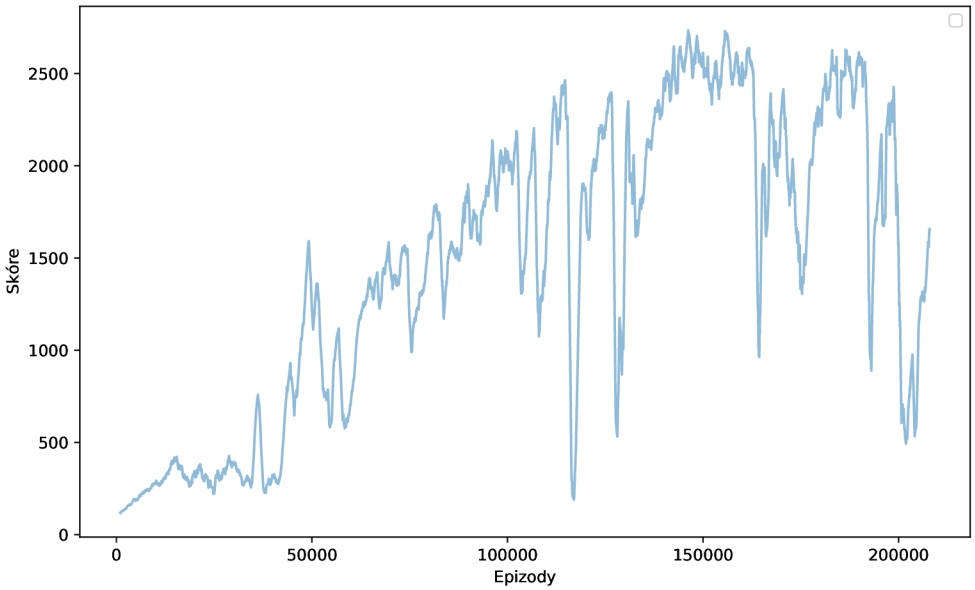
Na obrázku 6.6 jde vidět průběh učení chůze. Průběh získaných odměn je strmý a pak se to uklidí a postupně se model zlepšuje. Samotný výsledek však není nijak extra dobrý. Postavička se umí hnout, ale posunuje se po ploše pouze o malou vzdálenost a samotný pohyb není příliš vzhledný.



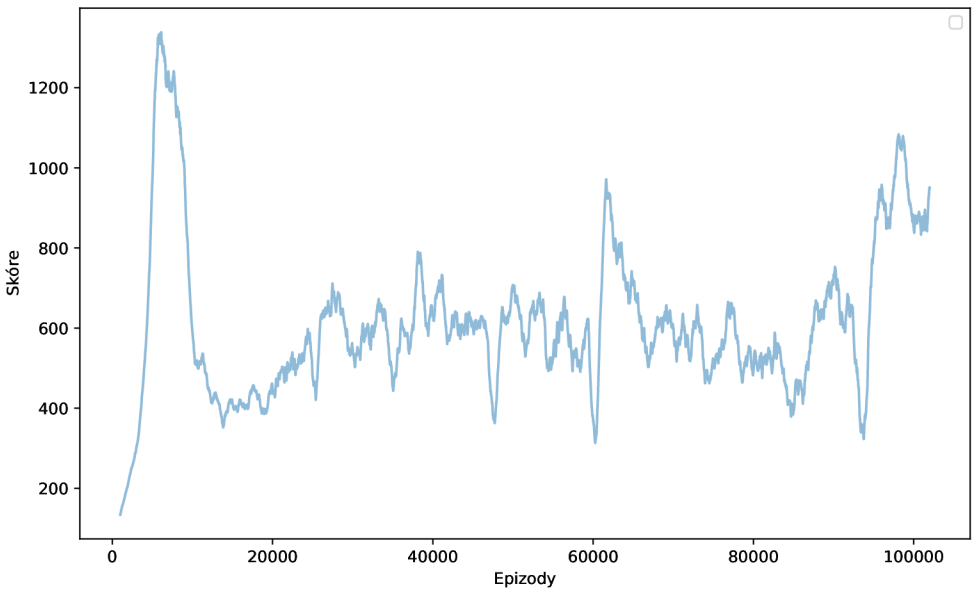
Obrázek 6.3: Graf ukazující jak dařilo modifikacím postavičky. Jedna modifikace byla krátká a široká, druhá dlouhá a úzká. Dle grafu vyšších hodnot dosahovala širší a kratší postavička, v některých částech trénování měla větší výkyvy.



Obrázek 6.4: Graf zobrazující průběh trénování postavičky, která měla jinak veliké chodidlo. Velikost od normálu byla snížena nebo zvýšena o 50%. Na zelené křivce, tedy největším chodidle jde vidět, že trénování probíhalo nejrychleji a dosahovalo nejlepších výsledků. Normální velikost a malá se od sebe příliš neliší.



Obrázek 6.5: Graf zobrazuje průběh získaných odměn při trénování skákání postavičky, která umí vyskočit do požadované výšky a zase spadnout zpět na zem. Graf obsahuje spoustu výkyvů, ale výsledný model splňuje základní funkčnost.



Obrázek 6.6: Graf zobrazující průběh trénování chůze postavičky. Graf zobrazuje průběh odměn, který je ze začátku velice strmý. Postavička se umí pohybovat, pohyb je však velice malý.

Kapitola 7

Závěr

V práci jsem se zabýval strojovým učením, konkrétně posilovaným učením a vysvětlil základní pojmy a principy, a s jakými daty algoritmus pracuje. Vysvětlil jsem jeden z algoritmů posilovaného učení, jež používám v experimentech.

Cílem této práce bylo vytvořit postavičku, která se pohybuje v prostředí a má za úkol plnit úkoly zadané v úlohách. Implementoval jsem ji pomocí AI gym a fyzikálního modelu PyBullet. Postavička se skládá ze dvou končetin, jimiž ovládá svůj pohyb. Úlohy zadané v prostředí jsou zaměřené na zvládnutí základních pohybů jako stání, skákání a chůzi. V experimentech jsem se zaměřil i na samotnou postavičku, kde jsem testoval efekt velikosti jednotlivých částí těla. Zadané úlohy jsem řešil pomocí nadefinovaného algoritmu Actor-Critic.

Experimenty ukázaly, že postavička je schopná s nadefinovanou funkcí odměn v nějaké míře plnit úlohy. Z experimentů se zjistilo, s jakými parametry je úloha splnitelná a jaké fyzické vlastnosti postavičky vedly k lepším výsledkům. Mnohdy model obsahuje ještě spoustu chyb zamezujících ukázkovým pohybům. Bylo by přínosné rozšířit funkci odměn o penalizační funkci. Například by hlídala využití končetin a jejich polohu, aby byla vždy co nejpřirozenější a tak napodobovala co nejvíce pohyb člověka, případně učit chodit postavičku za pomocí referenčních pohybů. Ideálně by měl model pracovat i s určitou energií a při použití vyšší síly by mohl být penalizován. U reálného modelu by postavička neměla neomezené množství energie, jako v mém případě. S energií by bylo potřeba zacházet tak, aby vydržela co nejdéle. Aby se uvažovaná akce například rozložila do menších akcí, kde se využije méně energie, než do jedné velké akce, kde by bylo potřeba hodně energie.

Literatura

[1] *Joint* [online]. 2018-11-05 [cit. 2021-30-04]. Dostupné z: http://wiki.ros.org/urdf/XML/joint/.

[2] *Soft Actor-Critic* [online]. 2018 [cit. 2021-09-05]. Dostupné z: https://spinningup.openai.com/en/latest/algorithms/sac.html.

[3] ARGERICH, M. F. *Entropy Regularization in Reinforcement Learning* [online]. Květen 2020 [cit. 2021-30-04]. Dostupné z: https://towardsdatascience.com/entropy-regularization-in-reinforcement-learning-a6fa6d7598df.

[4] BALSYS, R. *BipedalWalker-v3 with Continuous Proximal Policy Optimization* [online]. Listopad 2020 [cit. 2021-30-04]. Dostupné z: https://pylessons.com/BipedalWalker-v3-PPO/.

[5] BARBERA, V. la. *Continuous control with A2C and Gaussian Policies —MuJoCo , PyTorch and C++* [online]. Duben 2020 [cit. 2021-30-04]. Dostupné z: https://medium.com/@vittoriolabarbera/continuous-control-with-a2c-and-gaussian-policies-mujoco-pytorch-and-c-4221ec8ba024.

[6] BLACKBURN. *Reinforcement Learning : Markov-Decision Process (Part 1)* [online]. Červenec 2019 [cit. 2021-30-04]. Dostupné z: https://towardsdatascience.com/introduction-to-reinforcement-learning-markov-decision-process-44c533ebf8da.

[7] BLACKBURN. *Reinforcement Learning : Markov-Decision Process (Part 1)* [online]. Červenec 2019 [cit. 2021-09-05]. Dostupné z: https://towardsdatascience.com/introduction-to-reinforcement-learning-markov-decision-process-44c533ebf8da.

[8] BROCKMAN, G., CHEUNG, V., PETTERSSON, L., SCHNEIDER, J., SCHULMAN, J. et al. *OpenAI Gym*. 2016.

[9] BROWNLEE, J. *Understand the Impact of Learning Rate on Neural Network Performance* [online]. Leden 2019 [cit. 2021-30-04]. Dostupné z: https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/.

[10] CHILAMKURTHY, K. *Off-policy vs On-Policy vs Offline Reinforcement Learning Demystified!* [online]. Listopad 2020 [cit. 2021-30-04]. Dostupné z: https://kowshikchilamkurthy.medium.com/off-policy-vs-on-policy-vs-offline-reinforcement-learning-demystified-f7f87e275b48.

[11] COUMANS, E. a BAI, Y. *PyBullet, a Python module for physics simulation for games, robotics and machine learning* [http://pybullet.org]. 2016–2020.

[12] Degris, T., White, M. a Sutton, R. S. Off-policy actor-critic. *ArXiv preprint arXiv:1205.4839*. 2012.

[13] Deshpande, M. *Deep Reinforcement Learning: Value-based Methods* [online]. Prosinec 2018 [cit. 2021-30-04]. Dostupné z: https://mohitd.github.io/2018/12/23/deep-rl-value-methods.html.

[14] Deshpande, M. *Deep Reinforcement Learning: Policy-based Methods* [online]. Leden 2019 [cit. 2021-30-04]. Dostupné z: https://mohitd.github.io/2019/01/20/deep-rl-policy-methods.html.

[15] Dittert, S. *PyTorch Implementation of Soft-Actor-Critic-and-Extensions* [https://github.com/BY571/Soft-Actor-Critic-and-Extensions]. GitHub, 2020.

[16] Hui, J. *RL — Value Learning* [online]. Listopad 2018 [cit. 2021-30-04]. Dostupné z: https://jonathan-hui.medium.com/rl-value-learning-24f52b49c36d.

[17] Jagtap, R. *Understanding Markov Decision Process (MDP)* [online]. Září 2020 [cit. 2021-09-05]. Dostupné z: https://towardsdatascience.com/understanding-the-markov-decision-process-mdp-8f838510f150.

[18] Karunakaran, D. *Key concepts in Reinforcement Learning* [online]. Březen 2020 [cit. 2021-30-04]. Dostupné z: https://medium.com/intro-to-artificial-intelligence/key-concepts-in-reinforcement-learning-2af715dfbfa.

[19] Karunakaran, D. *Q-learning: a value-based reinforcement learning algorithm* [online]. Září 2020 [cit. 2021-30-04]. Dostupné z: https://medium.com/intro-to-artificial-intelligence/q-learning-a-value-based-reinforcement-learning-algorithm-272706d835cf.

[20] Lindwurm, E. *Intuition: Exploration vs Exploitation* [online]. Listopad 2019 [cit. 2021-30-04]. Dostupné z: https://towardsdatascience.com/intuition-exploration-vs-exploitation-c645a1d37c7a.

[21] Mishra, S. *Unsupervised Learning and Data Clustering* [online]. Březen 2017 [cit. 2021-09-05]. Dostupné z: https://towardsdatascience.com/unsupervised-learning-and-data-clustering-eeecb78b422a.

[22] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J. et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: Wallach, H., Larochelle, H., Beygelzimer, A., Alché Buc, F. d', Fox, E. et al., ed. *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, s. 8024–8035. Dostupné z: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[23] Saad, M. *Policy Gradient Theorem* [online]. Listopad 2020 [cit. 2021-30-04]. Dostupné z: https://able.bio/markhsaad/policy-gradient-theorem--ec6f9fd9.

[24] Silver, D. *Lectures on Reinforcement Learning* [url: https://www.davidsilver.uk/teaching/]. 2015.

[25] Suran, A. *On-Policy v/s Off-Policy Learning* [online]. Červenec 2018 [cit. 2021-30-04]. Dostupné z: https://towardsdatascience.com/on-policy-v-s-off-policy-learning-75089916bc2f.

[26] Sutton, R. S. a Barto, A. G. *Reinforcement learning: An introduction*. 2. vyd. MIT press, 2018. ISBN 9780262039246.

[27] Torres, J. *A gentle introduction to Deep Reinforcement Learning* [online]. Květen 2020 [cit. 2021-30-04]. Dostupné z: https://towardsdatascience.com/drl-01-a-gentle-introduction-to-deep-reinforcement-learning-405b79866bf4.

[28] Tovar, A. D. *Advantage Actor Critic continuous case implementation* [online]. Březen 2020 [cit. 2021-30-04]. Dostupné z: https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f.

[29] Tovar, A. D. *Advantage Actor Critic continuous case implementation* [online]. Březen 2020 [cit. 2021-30-04]. Dostupné z: https://medium.com/deeplearningmadeeasy/advantage-actor-critic-continuous-case-implementation-f55ce5da6b4c.

[30] Weng, L. *A (Long) Peek into Reinforcement Learning* [online]. Únor 2018 [cit. 2021-30-04]. Dostupné z: https://lilianweng.github.io/lil-log/2018/02/19/a-long-peek-into-reinforcement-learning.html.

[31] Weng, L. *Policy Gradient Algorithms* [online]. Duben 2018 [cit. 2021-30-04]. Dostupné z: https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html.

[32] Wikipedie. *Tíhové zrychlení — Wikipedie: Otevřená encyklopedie*. 2020. [Online]. Dostupné z: https://cs.wikipedia.org/w/index.php?title=T%C3%ADhov%C3%A9_zrychlen%C3%AD&oldid=18523201.

[33] Wilson, A. *A Brief Introduction to Supervised Learning* [online]. Září 2019 [cit. 2021-09-05]. Dostupné z: https://towardsdatascience.com/a-brief-introduction-to-supervised-learning-54a3e3932590.

[34] Won, J. a Lee, J. Learning Body Shape Variation in Physics-based Characters. *ACM Trans. Graph.* 2019, sv. 38, č. 6.

[35] Xie, Z., Ling, H. Y., Kim, N. H. a Panne, M. van de. ALLSTEPS: Curriculum-driven Learning of Stepping Stone Skills. In: *Proc. ACM SIGGRAPH / Eurographics Symposium on Computer Animation*. 2020.

Příloha A

Obsah přiloženého CD

- **src** – Zdrojové kódy programu – algoritmus, prostředí, natrénované modely
- **readme.md** – Popis jak se spouští program
- **plakat.jpg** – Plakát
- **bp.pdf** – Pdf verze bakalářské práce
- **latex** – Zdrojové soubory bakalářské práce v LaTeX
- **video** – Ukázky natrénovaných modelů

Příloha B

Plakát

Obrázek B.1: Plakát