|  |  |  |
| --- | --- | --- |
|  | | |
| **Závěrečná studijní práce**  **dokumentace** | | |
| **Deep Q-Learning Pong** | | |
| Samuel Antoš | | |
|  | | |
|  | |  |
| **Obor:** | 18-20-M/01 INFORMAČNÍ TECHNOLOGIE  se zaměřením na počítačové sítě a programování | |
| **Třída:**  **Školní rok:** | IT4  2020/2021 | |

#### Poděkování

*Rád bych poděkoval především doc. Ing. Petru Čermákovi, Ph.D. za objasnění nejasností a složité problematiky.*

Prohlašuji, že jsem závěrečnou práci vypracoval samostatně a uvedl veškeré použité informační zdroje.

Souhlasím, aby tato studijní práce byla použita k výukovým účelům na Střední průmyslové a umělecké škole v Opavě, Praskova 399/8.

V Opavě 31. 12. 2020

*podpis autora práce*

**ANOTACE**

Tato práce popisuje umělé neuronové sítě od základního perceptronu až po složitější vícevrstvé modely. Popsány jsou důležité aktivační funkce, metody učení, přesněji učení s a bez učitele, posilované učení a konkrétně v této oblasti způsob integrace Q-learningu s umělými neuronovými sítěmi (Deep Q-networks). Výsledkem projektu je virtuální prostředí pro učení umělé inteligence, kde se využívají již zmíněné techniky. Prostředí je vytvořené v Kivy na základě hry Pong, ve kterém je pomocí knihovny Keras (Tensorflow API) aplikovaná neuronová síť.

## KLÍČOVÁ SLOVA

Umělá inteligence; umělé neuronové sítě; posilované učení; programování

OBSAH

[Úvod 5](#_Toc61190203)

[1 Umělá neuronová síť 6](#_Toc61190204)

[1.1 Perceptron 6](#_Toc61190205)

[1.2 Přenosová (aktivační) funkce 7](#_Toc61190206)

[1.3 Multi-layer perceptron 7](#_Toc61190207)

[1.4 Typy učení 8](#_Toc61190208)

[1.4.1 Učení s učitelem 8](#_Toc61190209)

[1.4.2 Učení bez učitele 8](#_Toc61190210)

[1.4.3 Posilované učení 8](#_Toc61190211)

[1.5 Q-Learning 9](#_Toc61190212)

[1.5.1 Okamžitá a dlouhodobá odměna 9](#_Toc61190213)

[1.5.2 Explorace a exploatace 10](#_Toc61190214)

[1.5.3 Limitace Q-learningu 10](#_Toc61190215)

[1.6 Deep Q-network 11](#_Toc61190216)

[2 Využité technologie 12](#_Toc61190217)

[2.1 Kivy 12](#_Toc61190218)

[2.2 Keras 12](#_Toc61190219)

[3 Způsoby řešení a použité postupy 13](#_Toc61190220)

[3.1 Prostředí 13](#_Toc61190221)

[3.2 Umělá inteligence 13](#_Toc61190222)

[3.2.1 Struktura modelu 13](#_Toc61190223)

[3.2.2 Agent 14](#_Toc61190224)

[4 Výsledek řešení 15](#_Toc61190225)

[4.1 Funkce aplikace 15](#_Toc61190226)

[4.2 Splněné a nesplněné cíle 15](#_Toc61190227)

[Závěr 16](#_Toc61190228)

[Seznam použitýCH INFORMAČNÍCH ZDROJů 17](#_Toc61190229)

Úvod

Hlavní motivací bylo pochopit základy neuronových sítí a posilovaného učení. Je to velmi zajímavé, ale komplikované téma, které je čím dál více populární. Rozhodl jsem se proto pro studium základů umělé inteligence.

Cílem bylo vytvořit aplikaci, kde bude možné sledovat proces učení umělých neuronových sítí s použitím algoritmu DQN (Deep Q-networks) a možnost si proti nim zahrát.

Tato práce nejprve popisuje problematiku umělých neuronových sítí – základní podobu umělého neuronu (perceptron), jeho parametry a využití. Dále popisuje jeho řetězení do vrstev a vytváření umělých neuronových sítí. Práce rovněž zmiňuje základní přenosové (aplikační) funkce, které zastupují v neuronových sítích velmi důležitou roli. Vysvětleny jsou také různé metody učení, podrobněji je ale popisován algoritmus Q-learning a DQN z něj vycházející, protože právě DQN je využíván v projektu.

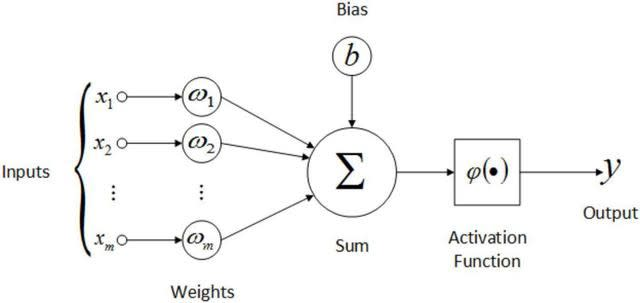
Práce dále seznamuje čtenáře s využitými technologiemi a postupy. Následuje vysvětlení funkčnosti, způsoby řešení, výčet splněných a nesplněných cílů a možná vylepšení.

# Umělá neuronová síť

Umělá neuronová síť je jeden z výpočetních modelů používaných v umělé inteligenci. Jejím vzorem je chování odpovídajících biologických struktur. Umělá neuronová síť je struktura určená pro distribuované paralelní zpracování dat. Skládá se z umělých (nebo také formálních) neuronů, jejichž předobrazem je biologický neuron. Neurony jsou vzájemně propojeny a navzájem si předávají signály a transformují je pomocí určitých přenosových funkcí

## Perceptron

Perceptron je základ neuronových sítí. Jedná se o pouze jeden neuron, který je možné učit pomocí změn jeho parametrů. Perceptron se skládá ze vstupů *x*, které jsou násobeny váhami (weights) *w*. Tyto váhy určující důležitost jednotlivých spojů a jejich celkový podíl na ovlivnění aktivační hodnoty *y*. Následuje sečtení všech vážených spojů (vážený součet), přičtení prahu *b* a vyhodnocení aktivace pomocí přenosové (aktivační) funkce *f*.



Obrázek 1.1: Schéma perceptronu

Při vstupech *x = (x1, x2, x3, …, xn)*, kdy počet vstupů označíme jako *n*, můžeme vyjádřit výstup *y* jako:

## Přenosová (aktivační) funkce

Přenosová (aktivační) funkce určuje aktivaci neuronu; zastupuje tak velmi důležitou roli a její výběr výrazně ovlivní chování celého modelu. Ideální je vybrat jednu z nelineárních funkcí – skokovou, sigmoidální nebo funkci ReLU (rectified linear unit).

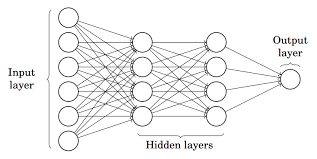
Skoková funkce je asi nejjednodušší aktivační funkce. Její výstup se rovná 0, dokud vstup nepřekročí určitou hranici (například přechod ze záporných do kladných čísel); po překročení hranice se výstup bude rovnat 1. Této funkci se taky přezdívá binární skok (binary step).

Sigmoidální funkce má podobný tvar jako písmeno „S“. Jedna z typů sigmoidálních funkcí je například logistická funkce. Logistickou funkci můžeme definovat jako:

ReLU funkce má také několik různých typů. Její základní podoba má výstup rovný nule, pokud je vstup menší než nula, jinak se výstup rovná vstupu. Tuto podobu lze definovat jako:

## Multi-layer perceptron

Jednovrstvý perceptron není vhodný pro řešení komplexnějších úloh, kde lineární aproxi-mace nestačí. Naštěstí je možné perceptrony řetězit do vrstev, a vytvořit tak výpočetně vý-konnější model. Tyto struktury se nazývají Více-vrstvé perceptrony (Multi-layer per-ceptron). Schéma tohoto modelu je znázorněno na obrázku



Obrázek 1.2: Schéma MLP

Vstupní vrstva nijak nevypočítává svou aktivaci, pouze distribuuje vstupy do první skryté vrstvy, kde se až uplatňují stejné principy jako u jednoduchého perceptronu. Jednotlivé neurony si přeposílají své aktivační hodnoty jako vstupy do další vrstvy, proto tuto síť můžeme označit jako dopřední neurální síť (feedforward neural network).

## Typy učení

Cílem učení neuronové sítě je nastavit síť tak, aby dávala co nejpřesnější výsledky. V biologických sítích jsou zkušenosti uloženy v dendritech. V umělých neuronových sítích jsou zkušenosti uloženy v jejich matematickém ekvivalentu – váhách. Učení neuronové sítě rozlišujeme na učení s učitelem a učení bez učitele. Fáze učení neuronové sítě bývá nazývána adaptivní. Po naučení neuronové sítě je síť ve fázi vybavování.

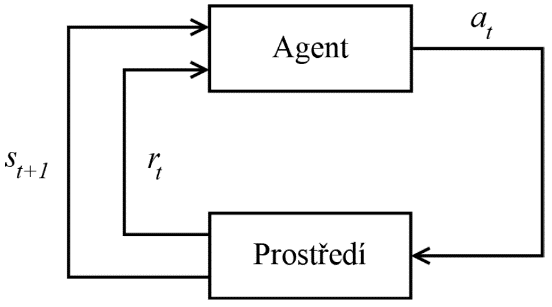
### Učení s učitelem

Podobně jako v biologických sítích je zde využita zpětná vazba. Neuronové síti je předložen vzor. Na základě aktuálního nastavení je zjištěn aktuální výsledek. Ten porovnáme s vyžadovaným výsledkem a určíme chybu. Poté spočítáme nutnou korekci (dle typu neuronové sítě) a upravíme hodnoty vah či prahů, abychom snížili hodnotu chyby. Toto opakujeme až do dosažení námi stanovené minimální chyby. Poté je síť adaptována.

### Učení bez učitele

Při učení bez učitele nevyhodnocujeme výstup. Při tomto učení nám výstup není znám. Síť dostává na vstup sadu vzorů, které si sama třídí. Buď si vzory třídí do skupin a reaguje na typického zástupce, nebo si přizpůsobí topologii vlastnostem vstupu.

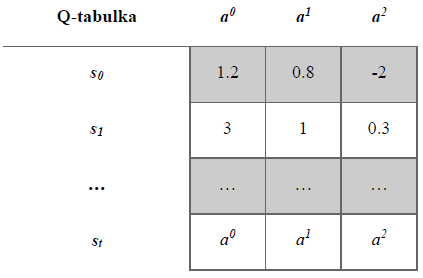
### Posilované učení

Posilované učení stejně jako učení bez učitele nemá žádnou externí znalost výstupů (žádná trénovací data), na rozdíl od učení bez učitele má ale zpětnou vazbu od prostředí. Tato zpětná vazba může být například ve formě odměny – za chtěnou akci kladná a za nechtěnou záporná. Tímto způsobem lze model naučit požadovanému chování definovanému v odměňovací funkci. V kontextu posilovaného učení označujeme učený model jako agenta. Tento agent má určitý stava na tento stav reaguje akcí. Akce potom ovlivňuje prostředí, které vrací agentovy nový stava odměnu za akci.

Obrázek 1.3: Schéma posilovaného učení

Pro dosažení co největší celkové odměny musí agent volit co nejlepší akce vzhledem k svému stavu. Tato strategie (policy) se označuje jako *π* policy a má za úkol ke každému stavu přiřadit co nejlepší možnou akci.

## Q-Learning

Q-learning je učící algoritmus založený na posilovaném učení. Na rozdíl od jiných algoritmů, které hledají funkci *π* přímo (policy gradient metody), Q-learning je metoda nepřímá, ideální funkci *π* hledá pomocí Q-tabulky. V této tabulce se akce mapují na stavy – každý řádek představuje jeden stav, sloupce v tomto řádku představují akce. Agent potom volí takovou akci, která má v této tabulce nejvyšší hodnotu. Těmto hodnotám říkáme Q-values (Q-hodnoty), kde Q označuje kvalitu (quality) akce. Funkce vybírající nejlepší možnou akci z Q-tabulky se nazývá Q-funkce.

Obrázek 1.4: Q-tabulka

Q-hodnoty budeme aktualizovat podle odměn, které agent dostal od prostředí za akci. Tato odměna se nazývá posílení (reinforcement) a může být kladná, záporná (trest), nebo 0. Pokud agent dostal za akci zápornou odměnu, bude její hodnota v Q-tabulce snížená, a tak i pravděpodobnost výskytu této akce. Tímto způsobem můžeme vytěsňovat nechtěné chování a podporovat požadované chování.

### Okamžitá a dlouhodobá odměna

Aktualizační pravidlo pro Q-hodnoty můžeme rozdělit do dvou částí: na okamžitou a dlouhodobou část. Okamžitá část je odměna za poslední akci, Dlouhodobá část přičítá k této odměně ještě budoucí Q-hodnotu. Tímto způsobem se snažíme maximalizovat odměnu i budoucí akce. Například pokud existují dvě možné akce v určitém stavu, tak ta nejlépe hodnocená akce nemusí být (dlouhodobě) nejlepší; agent by se mohl dostat do stavu, kde jeho další akce by vyvolala celkově nižší odměnu.



Obrázek 1.5: Dlouhodobá versus okamžitá odměna

Ideální je však najít rovnováhu mezi okamžitou a dlouhodobou odměnou. Důležitost dlouhodobé odměny můžeme regulovat slevou (discount factor) *γ* a celkovou změnu Q-hodnoty mírou učení (learning rate) *α*. Celkovou odměnu *r* (okamžitou + dlouhodobou) je možné definovat jako:

Korespondující Q-hodnotu v tabulce můžeme přímo přepsat novou odměnou. Tento způsob ale není ideální a mnohem častěji se pouze upravuje již existující Q-hodnota. Celé aktualizační pravidlo potom bude vypadat následovně:

### Explorace a exploatace

Agent se snaží vybírat jen ideální akce a postupně své reakce vylepšuje tak, aby dostal co nejvyšší kumulativní odměnu. Tento způsob se nazývá exploatace, kde snažíme vylepšovat již známý způsob. Pokud ale existuje alternativní způsob řešení, který by eventuálně vedl k vyšší odměně, exploatace ho nikdy nevyzkouší. Jiné způsoby řešení by ze začátku vyvolávaly nízkou odměnu, a tím i menší pravděpodobnost výskytu. Proto musíme prostředí neustále objevovat (explore). Jedna z možností explorace je například provést náhodnou akci místo použití akce z Q-tabulky. Velmi často se zavádí metoda 𝜀-greedy, kde 𝜀 je pravděpodobnost náhodné akce. Tato pravděpodobnost bývá ze začátku velmi vysoká a klesá s časem, nikdy však neklesá na nulu.

### Limitace Q-learningu

Mapování stavů na akce v Q-tabulce má několik nevýhod. Komplexnější úlohy, kde stav může být definován několika proměnnými najednou, by mohly mít Q-tabulku dosahující až nepoužitelných velikostí a vzhledem k tomu, že stavy musí být konečná množina, tak spojité stavy jsou neřešitelným problémem. Další problém jsou akce; v základním Q-learningu jsou akce buď vybrané, anebo ne (binární hodnota). Nelze tedy spojitě kontrolovat výstup. Oba tyto problémy řeší implementace umělých neuronových sítí.

## Deep Q-network

Deep Q-network (DQN) je algoritmus kombinující Q-learning a umělou neuronovou síť. Hlavní myšlenka tohoto algoritmu je nahradit Q-tabulku, umělou neuronovou sítí, které říkáme Q-síť (Q-network). Úkol této sítě bude co nejlépe aproximovat funkci Q-tabulky. Činnost bude v mnoha ohledech podobná, například vstup do Q-sítě bude stava výstup bude reakce na tento stav. Velká změna bude však v učení, kde na rozdíl od Q-tabulky nemůžeme jen změnit Q-hodnotu pomocí aktualizačního pravidla. Q-síť musíme učit jako každou jinou neuronovou síť a vzhledem k tomu, že všechny akce agenta jsou stále hodnoceny odměňovací funkcí (generace trénovacích dat), použijeme učení s učitelem.

Dalším rozdílem oproti klasickému Q-learningu je aktualizační pravidlo. Vzhledem k tomu že umělou neurální síť nijak nepřepisujeme, musíme vynechat část aktualizační pravidla, která přičítá starou Q-hodnotu. Takové pravidlo bude vypadat následovně:

Umělé neuronové sítě není ale ideální učit jen na jedné hodnotě. Mnohem lepší způsob je pamatovat si všechny předchozí stavy agenta. Ukládáme stav, akci, odměnu a budoucí stav, tedy všechny parametry nutné pro spočítání nové Q-hodnoty. Agent při učení vybírá náhodně seřazenou dávku o fixní velikosti z těchto vzpomínek. Následně se na této dávce vypočítají nové Q-hodnoty a pomocí algoritmu backpropagation se aplikují na umělou neurální síť. Tento způsob pamatování si minulých stavů se nazývá experience replay.

# Využité technologie

Aplikace je napsána v jazyce Python, konkrétně je využit Python 3.7.9. Python je jednoduchý na čtení, složité věci se v něm rychle realizují, podporuje paralelizaci a díky rozsáhlým matematickým frameworkům je ve světě umělých inteligencí často používaný. Vykreslení aplikace je realizováno pomocí knihovny Kivy. Pro vytvoření modelu umělé inteligence, jeho učení a predikce je použita knihovna Keras.

## Kivy

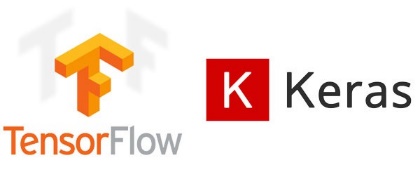
Kivy je multiplatformní grafická knihovna, akcelerovaná pomocí grafického rozhraní OpenGL. Další výhodou knihovny Kivy je její přístup. Na rozdíl od ostatních knihoven, kde základem je opakující se vykreslovací funkce, Kivy klade spíše důraz na změny. Celý kód je tak přehlednější, jednodušší, a hlavně rychlejší.

Obrázek 2.1: Kivy

## TensorFlow, Keras

TensorFlow je bezplatná a otevřená softwarová knihovna pro strojové učení. Může být použit v celé řadě úkolů, ale má zvláštní zaměření na trénink a odvozování hlubokých neuronových sítí. TensorFlow je symbolická matematická knihovna založená na toku dat a diferencovatelném programování. Konkrétně jsem využil TensorFlow 2.1.0.

Knihovna Keras je nadstavba knihovny Tensorflow specializovaná na umělé neurální sítě a umělou inteligenci. Použití takové knihovny nejen zjednodušuje celý učící proces, ale také ho zrychluje. Jádro Tensorflowu je totiž napsáno v jazyce C a je plně paralelizováno. Dále je možné využívat i více prostředků najednou (GPU, CPU, TPU). Aplikace samotná primárně využívá procesor, využití jiných prostředků je ale možné.



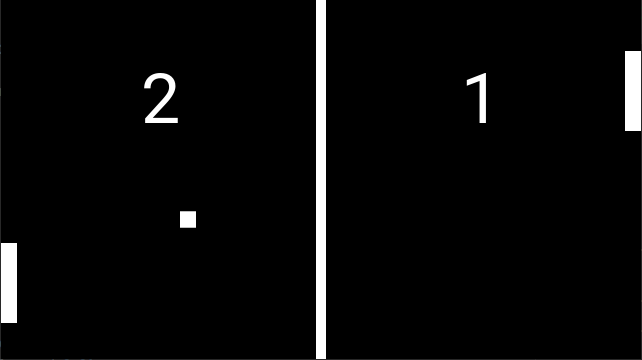
Obrázek 2.2: TensorFlow a Keras

# Způsoby řešení a použité postupy

## Prostředí

Prostředí aplikace je inspirováno hrou Pong. Ve 2D rovině se nachází míček a dvě pálky. Do středu se umístí míček, který dostane určitou rychlost náhodným směrem. Pokud míček narazí do stěny, odrazí se. Pálky jsou umístěny na protějších stranách prostředí a lze s nimi pohybovat nahoru a dolů po celé výšce prostředí. Každý hráč ovládá jednu pálku a má za úkol odrazit míček. Pokud se mu to nepodaří, protihráč získá bod, míček se opět umístí na střed a pokračuje se. Hráči mohou určit směr pohybu míčku po odražení od pálky tím, kterou částí pálky míček odrazí. Střed pálky odrazí míček rovně, kraje pálky odrazí míček víc do boku. Po každém odražení od pálky se míček trochu zrychlí, dokud se nedostane na určitou maximální rychlost, pak už nezrychluje.

Obrázek 3.1: Prostředí 1



## Umělá inteligence

Umělá inteligence je dosahována pomocí algoritmu DQN s vícevrstvým perceptronem. Agent získává od prostředí informace o pozici obou pálek, míčku a směru pohybu míčku. Tyto informace jsou použity jako stavy pro agenta. Agent na tento stav může reagovat pohybem nahoru, dolů nebo neprovádět nic. Agent je kladně odměňován za odražení míčku a za získání bodu, negativní odměnu dostává, pokud neodrazí míček nebo protihráč odrazí míček.

Agent si každý krok ukládá stavy, akce a odměnu. Uložené páry stavů a akcí budu dále označovat jako *vzpomínky*. Takto si agent vytváří databázi vzpomínek, které jsou ohodnoceny odměnou. Ze vzpomínek je náhodně vybrána určitá dávka (batch). Na jednotlivých vzpomínkách se počítá nová Q-hodnota a ta se potom pomocí algoritmu backpropagation aplikuje na umělou neurální síť agenta (experience replay). Správně by tento proces měl probíhat každý krok, ale kvůli vysokým nárokům na výkon je prováděn každým třetím krokem.

### Struktura modelu

Vstupní vrstva obsahuje šest neuronů (pozice levé pálky na ose y, pozice pravé pálky na ose y, pozice míčku na ose x, pozice míčku na ose y, rychlost míčku na ose x, rychlost míčku na ose y). Dále jsou v modelu dvě skryté vrstvy o 256 neuronech a výstupní vrstva, která obsahuje tři neurony (akce nahoru, dolů, nedělej nic). Aktivační funkce pro výstupní a skryté vrstvy je ReLU. Model využívá optimizer Adam a chybovou funkci MSE.

**def** build\_dqn(lr, n\_actions, input\_dims, fc1\_dims, fc2\_dims):

    model = Sequential([

                Dense(fc1\_dims, input\_shape=(input\_dims,)),

                Activation('relu'),

                Dense(fc2\_dims),

                Activation('relu'),

                Dense(n\_actions)])

    model.compile(optimizer=Adam(lr=lr), loss='mse')

**return** model

# Výsledek řešení

## Funkce aplikace

Po spuštění aplikace si uživatel může vytvořit nebo nahrát již vytvořenou neuronovou síť a dále zvolit, které pálky se budou ovládat lidskými hráči a které budou ovládány zvolenými neuronovými sítěmi. Pak může uživatel zahájit hru. Zvolené neuronové sítě se v průběhu hry neustále učí. Uživatel tak má možnost sledovat, jak se neuronová síť stále zlepšuje.

## Splněné a nesplněné cíle

Seznam cílů:

* Pochopení základů umělé inteligence
* Pochopení základů DQN
* Funkční aplikace
* Funkční umělá neuronová síť
* DQN
* Možnost sledovat, jak se umělá neuronová síť učí
* Možnost zahrát si proti umělé neuronové síti
* Ukládání modelů

# Závěr

Cílem projektu bylo pochopit teorii ohledně posilovaného učení umělých neuronových sítí a vytvoření aplikace, kde bude možné sledovat proces jejich učení a možnost si proti nim zahrát.

Aplikace v momentálním stavu umožňuje uživateli vytváření modelů, sledovat proces jejich učení přímo ve hře a kdykoli si proti nim zahrát. Cíl práce byl tedy zcela naplněn.

Naskytují se další možná vylepšení, jak grafické, tak především možnost uživatele měnit strukturu a veškeré parametry algoritmu DQN. Tato vylepšení bych někdy v budoucnu rád přidal.

Praktické řešení: <https://github.com/traktor312/zaverecny_projekt>

Seznam použitýCH INFORMAČNÍCH ZDROJů

[1] 3Blue1Brown. Neural networks [online], YouTube. Aug 1, 2018 [cit. 2020-12-28]. Dostupné z: https://www.youtube.com/playlist?list=PLZHQObOWTQDNU6R1\_67000Dx\_ZCJB-3pi

[2] BROWNLEE, Jason. Gentle Introduction to the Adam Optimatization Algorithm for Deep Learning [online]. Aug 20, 2020 [cit. 2020-12-28]. Dostupné z: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

[3] JULIANI, Arthur. Simple Reinforcement Learning with Tensorflow [online]. Aug 25, 2016 [cit. 2020-12-28]. Dostupné z: https://medium.com/emergent-future/sim-ple-reinforcement-learning-with-tensorflow-part-0-q-learning-with-tables-and-neural-networks-d195264329d0

[4] KANTOR, Jan. Učení bez učitele [online]. Brno, 2008 [cit. 2020-12-28]. Dostupné z: https://core.ac.uk/download/pdf/30296714.pdf. Diplomová Práce. VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ. Vedoucí práce Ing. PETR HONZÍK, Ph.D.

[5] KLŮJ, Jan. Obecná umělá inteligence pro hraní her [online]. Praha, 2017 [cit. 2020-12-28]. Dostupné z: https://dspace.cuni.cz/handle/20.500.11956/90569. Diplomová práce. Univerzita Karlova, Matematicko-fyzikální fakulta.

[6] SENO, Takuma. *Deep Reinforcement Learning* [online]. Oct 20, 2017 [cit. 2020-12-28]. Dostupné z: https://towardsdatascience.com/welcome-to-deep-reinforce-ment-learning-part-1-dqn-c3cab4d41b6b

[7] The Kivy Authors. Welcome to Kivy – Kivy 2.0.0 documentation [online]. [cit. 2020-12-28]. Dostupné z: https://kivy.org/doc/stable