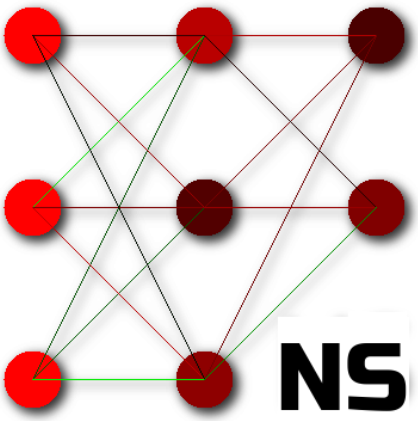
SŠPU
STŘEDNÍ ŠKOLA PRŮMYSLOVÁ
A UMĚLECKÁ, OPAVA

# ZÁVĚREČNÁ STUDIJNÍ PRÁCE

**dokumentace**

# Neurální sítě a jejich učení pomocí metod posilovaného učení

Denis Kurka




**Obor:** 18-20-M/01 INFORMAČNÍ TECHNOLOGIE
se zaměřením na počítačové sítě a programování

**Třída:** IT4
**Školní rok:** 2019/2020

***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. 2019

*podpis autora práce*

## ANOTACE

Práce popisuje umělé neurální sítě od jednoduchých modelů až po vícevrstvé modely. Dále jsou popisovány metody učení bez učitele a učení s učitelem. Dopodrobna se však rozebírá metoda posilovaného učení (reinforcement learning) a konkrétně v této oblasti způsob integrace Q-learningu s umělými neuronovými sítěmi (Deep Q-networks). Výsledkem práce je virtuální prostředí pro učení umělé inteligence, kde se využívají již zmíněné techniky.

## KLÍČOVÁ SLOVA

Umělá inteligence; umělé neuronové sítě; posilované učení; programování

## ANNOTATION

This paper describes artificial neural networks from the simplest models to multilayered models. Learning methods like supervised learning and unsupervised learning are also covered. In a more detailed way, however, only reinforcement learning is then described, mainly Q-learning and it's implementation with artificial neural networks (Deep Q-networks). The final result is the virtual environment for learning artificial neural networks.

## KEYWORDS

Artificial intelligence; artificial neural networks; reinforcement learning; programming

# OBSAH

# ÚVOD

Hlavní motivací bylo vytvořit nástroj pro učení umělých neuronových sítí, kde je možné měnit prostředí, samotné učení i učenou síť. Předat uživateli veškerou kontrolu nad učícím procesem a prostředím. V počítačových hrách se tento žánr označuje jako sandbox. Tyto hry nenutí hráče dělat nic konkrétního, hra jim pouze dává možnosti a prostředí tyto možnosti využít. V minulosti jsem už jednou tuto myšlenku realizoval pomocí umělých neuronových sítí a genetických algoritmů. Toto prostředí jsem nazval NeuralSandbox. Genetické algoritmy mi ale připadaly jako hod kostkou. Začal jsem tedy vyvíjet nové prostředí učící agenta pomocí metod posilovaného učení. Tyto metody jsem začal studovat hlavně kvůli výzkumu společnosti DeepMind a jejich implementaci umělé inteligence do hry StarCraft II. Tuto umělou inteligenci DeepMind nazývá AplhaStar, a je to první umělá inteligence porážející i profesionální hráče v této náročné strategické hře. Rozhodl jsem se tedy pro tento projekt použít jednu z metod učení umělých neuronových sítí publikovanou společností DeepMind, konkrétně algoritmus DQN (Deep Q-networks). Samotné prostředí pro učení umělých neuronových sítí bylo podle algoritmu DQN pojmenováno NeuralQsandbox.

Tato práce nejprve popisuje základní podobu umělého neuronu, různé jeho parametry, využití a limitace. Tato myšlenka je následně rozšířena pomocí řetězení umělých neuronů do vrstev a vytváření umělých neuronových sítí. Zmíněny jsou i různé metody učení, dopodrobna jsou ale popisovány pouze metody využívané v projektu (učení s učitelem, posilované učení). Pro popsání algoritmu DQN bylo nutné zmínit algoritmus Q-learning, ze kterého DQN vychází. Následně je popisován algoritmus DQN a jeho jednotlivé části.

Po teoretické části jsou popsány postupy využité při vývoji NeuralQsandboxu. Následuje vysvětlení funkčnosti a významu jednotlivých částí aplikace.

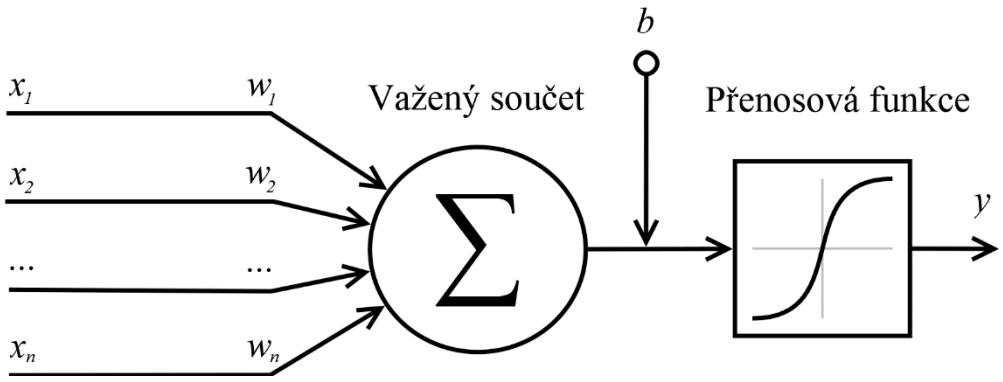
# 1 UMĚLÉ NEURONOVÉ SÍTĚ

Umělá neuronová síť (artificial neural network) je matematický model inspirovaný biologickým nervovým systémem. Půjčuje si poznatky o struktuře a předávání signálu, je ale stále velmi volnou interpretací jeho biologických protějšků.

## 1.1 Perceptron

Perceptron je základ neuronových sítí. Jedná se o pouze jeden neuron, který je možné učit pomocí změn jeho parametrů. Název Perceptron vychází z latinského slova „percipio“, což znamená učit se či pochopit.

Perceptron se skládá ze vstupů $x = (x_1, x_2, x_3, \ldots, x_n)$, které jsou násobeny váhami (weights) $w = (w_1, w_2, w_3, \ldots, w_n)$. 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 $f$.



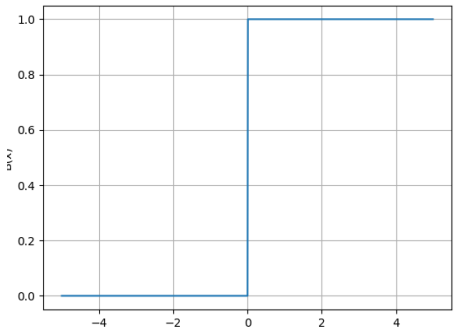

Obrázek 1.1: Schéma perceptronu

Při vstupech $x = (x_1, x_2, x_3, \ldots, x_n)$, kdy počet vstupů označíme jako $n$, můžeme vyjádřit výstup $y$ jako

$$y = f\left(\sum_{i=1}^{n} x_i w_i + b\right).$$

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).

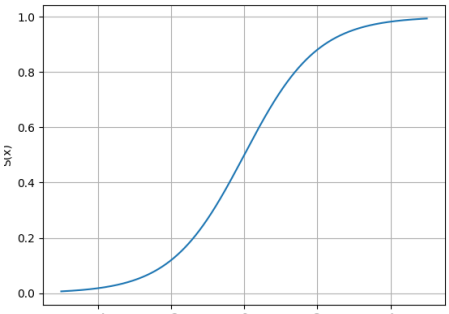
Skoková funkce je asi nejjednodušší aktivační funkce. Její výstup se rovná 0, dokud vstup nepřekročí určitou hranici (například přechod ze záporných do kladných čísel); po překročení hranice se výstup bude rovnat 1. Této funkci se taký přezdívá binární skok (binary step).



Obrázek 1.2: Skoková funkce

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

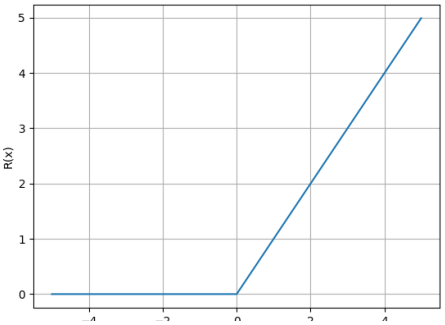
$$S(x) = \frac{1}{1+e^{-x}} .$$



Obrázek 1.3: Logistická funkce

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

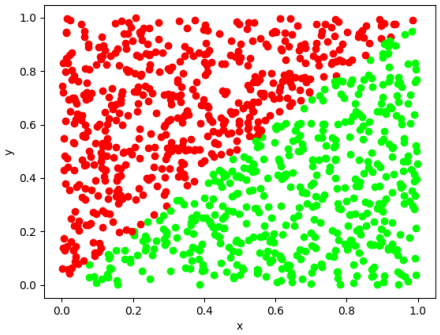
$$R(x) = \frac{x + |x|}{2} .$$



Obrázek 1.4: ReLU funkce

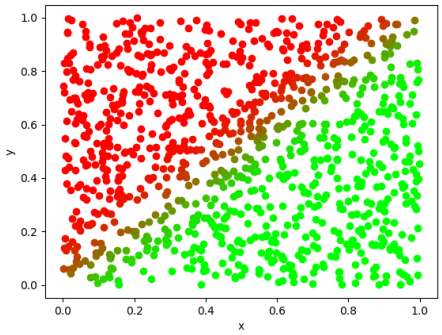
Jednoduchý perceptron dokáže kategorizovat cokoliv, co je lineárně rozdělitelné. Například náhodně vygenerované body, které jsou v rovině rozděleny podle přímky $y = x$. Pro klasifikaci takových dat můžeme použít jednovrstvý perceptronový model, který po učícím procesu budeme testovat na nových bodech neobsažených v učícím procesu.

Rozdělení bodů podle přímky




Obrázek 1.5

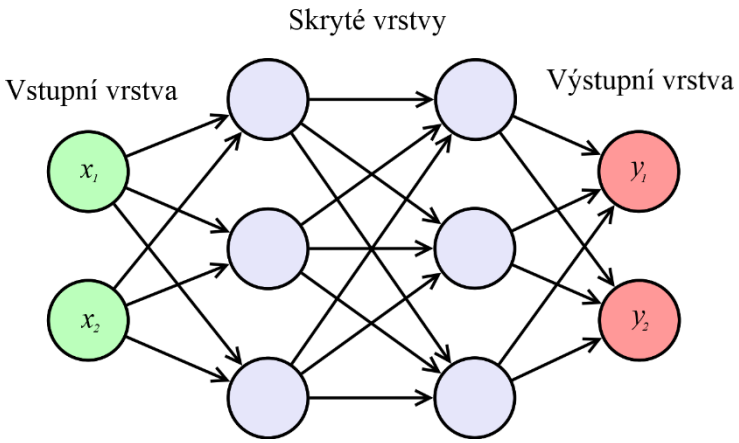
Rozdělení bodů podle perceptronu




Obrázek 1.6

## 1.2 Multi-layer perceptron

Jednovrstvý perceptron není vhodný pro řešení komplexnějších úloh, kde lineární aproximace nestačí. Naštěstí je možné perceptrony řetězit do vrstev, a vytvořit tak výpočetně výkonnější model. Tyto struktury se nazývají Více-vrstvé perceptrony (Multi-layer perceptron). Schéma tohoto modelu je znázorněno na obrázku 1.7.




Obrázek 1.7: 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 dopředí neurální síť (feedforward neural network).

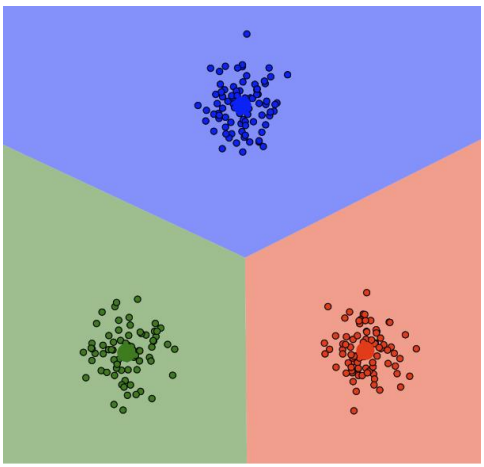
## 1.3 Typy učení

Pro dosažení požadovaného chování (požadovaných výstupů ze sítě) musíme síť učit. To znamená měnit parametry sítě (váhy, prahy) tak, aby se přibližovala našemu požadovanému výsledku. Takových způsobů je spousta, dají se však rozdělit do tří skupin. Učení s učitelem, učení bez učitele a posilované učení.

### 1.3.1 Učení bez učitele

Učení bez učitele (unsupervised learning) je způsob hledání předem neznámých vzorů v určitých datech. U tohoto způsobu nemáme žádnou externí znalost o výstupu a snažíme se ve vstupních datech hledat podobnosti či rozdíly. Učící techniky proto musí samy řídit učící proces (nemají učitele), proto se tomuto přístupu také říká **samoorganizující**. Jedna z možností jak vstupní data rozlišovat je třídit jednotlivé prvky do shluků s podobnými vlastnostmi. Tato podoblast se nazývá shlukování (clustering).

Mezi algoritmy nevyužívající neuronové sítě patří například k-průměr (k-means). K-means vyžaduje dopředu znát počet shluků, do kterých jednotlivé objekty přiřazuje podle střední hodnoty objektu. Střed shluku se zde nazývá centroid. Vizualizace shlukování pomocí algoritmu k-means je možné vidět na obrázku 1.8.



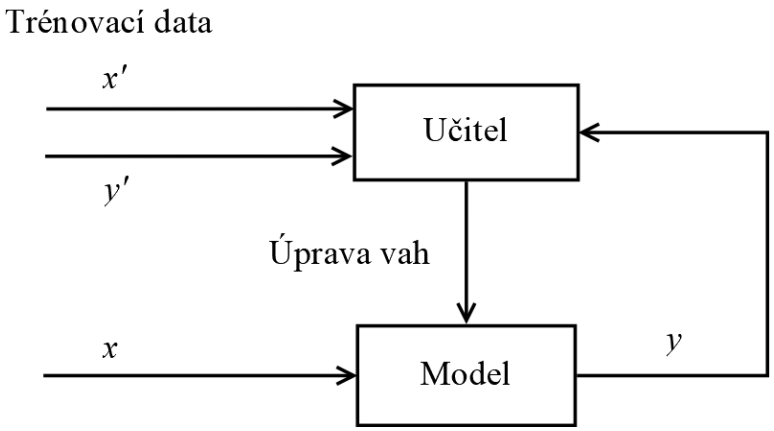
Obrázek 1.8: k-means clustering

Pro lepší porozumění algoritmu k-means doporučuji vyzkoušet internetovou vizualizaci od Naftalina Harrisona [2].

Umělé neurální sítě je také možné učit pomocí učení bez učitele. Toto učení se využívá například s kombinací generativních modelů, které mají za úkol generovat obraz, zvuk nebo jiná data.

### 1.3.2 Učení s učitelem

Učení s učitelem (supervised learning) se snaží naučit model určitého chování podle trénovacích dat. Tato trénovací data (labeled data) se skládají ze vstupů a očekávaných výstupů. Učení samotné se potom skládá z vyhodnocení chyby a úpravy parametrů modelu.




Obrázek 1.9: Schéma učení s učitelem

Pro výpočet chyby můžeme jednoduše odečítat výstup modelu od očekávaného výstupu. Takový přístup ale není ideální a může dojít k nechtěným změnám modelu. Často se proto používá funkce středové kvadratické chyby (mean squared error). Pro počet trénovacích párů $N$, dimenzí výstupu sítě $n$, výstup sítě $y$ a očekávaného výstupu $d$, můžeme chybu $E$ vyjádřit jako

$$E = \frac{1}{2}\sum_{i}^{N}\sum_{j}^{n}(y_{i,j} - d_{i,j})^2 \ .$$

Tuto chybu se budeme snažit minimalizovat a najít tak minimum (nejlépe globální), kde učený model bude co nejlépe schopný odhadovat správný výstup. Pro minimalizace takové funkce ale potřebujeme znát, do jaké míry jednotlivé učící parametry (váhy, prahy, …) ovlivňují tuto funkci. Například pokud váhu $w_1$ zdvojnásobíme a chyba bude čtyřikrát vetší, mohli bychom usoudit, že dvojnásobné zmenšení váhy $w_1$ by způsobilo čtyřnásobné zmenšení chyby. Pokud tuto myšlenku zobecníme, můžeme tvrdit, že od všech parametrů sítě musíme odečíst jejich *vliv* na chybovou funkci. Tento *vliv* nám říká, jakým *směrem* se bude funkce měnit, a proto pro zjištění *vlivu* musíme vypočítat směrnici (gradient). Směrnici zjistíme pomocí derivace chybové funkce vzhledem k jednotlivým vahám. Obecné pravidlo pro váhy $w$ v čase $t$ potom vypadá následovně

$$w_i(t+1) = w_i(t) - \alpha\frac{\partial E}{\partial w_i} \ ,$$

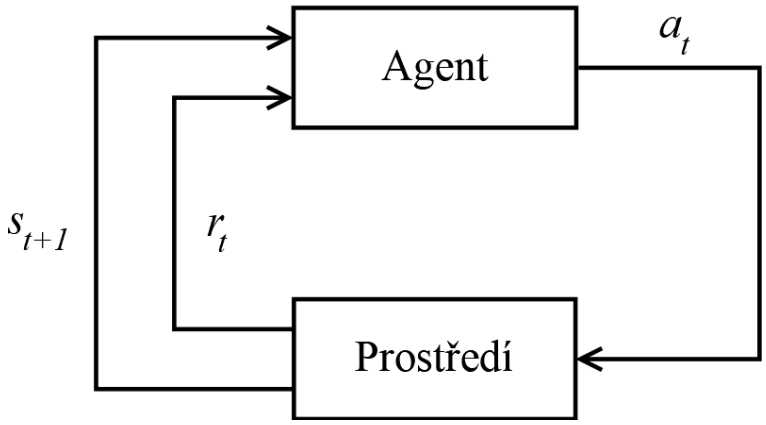
kde α je míra učení, neboli jak moc chceme jednotlivé váhy změnit.

Takto budeme váhy upravovat, dokud nedosáhneme požadovaných výstupů. Díky tomu, že odečítáme směrnici (gradient) od jednotlivých vah, nazývá se tento algoritmus gradient descent (klesáme do lokálního minima). Pro tento algoritmus existuje několik vylepšení. Namísto počítání směrnice přes všechna trénovací data můžeme vytvořit balíčky (batches) a směrnici počítat na nich; tento algoritmus se nazývá stochastic gradient descent (SGD). Jedna z nejnovějších metod, která se ještě navíc snaží „držet směr“ [5], je například algoritmus Adam.

Celý učící algoritmus se potom nazývá **backpropagation**. Tento název byl zvolen, protože při zjišťování gradientu váhy musíte znát i gradient vstupních hodnot a to znamená, že musíte spočítat celou umělou neuronovou síť od zjišťované váhy až po vstupy.

### 1.3.3 Posilované učení

Posilované učení (reinforcement learning) 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 stav $s_t$ a na tento stav reaguje akcí $a_t$. Akce potom ovlivňuje prostředí, které vrací agentovy nový stav $s_{t+1}$ a odměnu za akci $r_t$. Tento proces se opakuje, dokud agent nedosáhne cíle, nebo až je učení přerušeno.




Obrázek 1.10: 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. Algoritmů pro hledání ideální funkce π je celá řada, podrobněji se ale podíváme jenom na algoritmus Q-learning a algoritmy z něho vycházející.

## 1.4 Q-learning

Q-learning je učící algoritmus založený na posilováním 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.

| **Q-tabulka** | $a^0$ | $a^1$ | $a^2$ |
|---|---|---|---|
| $s_0$ | 1.2 | 0.8 | -2 |
| $s_1$ | 3 | 1 | 0.3 |
| … | … | … | … |
| $s_t$ | $a^0$ | $a^1$ | $a^2$ |

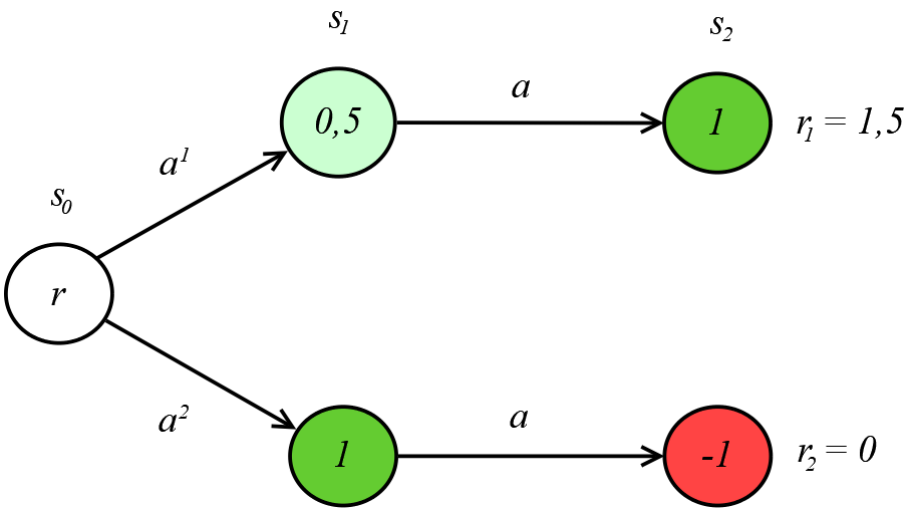
Pokud by se například agent nacházel v prostředí, které by odpovídalo naší Q-tabulce, tak ve stavu $s_0$ by zvolil akci $a^0$, protože její Q-hodnota je nejvyšší.

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 [6]. 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.

### 1.4.1 Okamžitá a dlouhodobá odměna

Aktualizační pravidlo pro Q-hodnoty můžeme rozdělit do dvou částí: na okamžitou a dlouhodobou část. Okamžitá část je odměna $r_t$ za poslední akci $a_t$, Dlouhodobá část přičítá k této odměně ještě budoucí Q-hodnotu (ze stavu $s_{t+1}$). Tímto způsobem se snažíme maximalizovat odměnu i budoucí akce. Například pokud existují dvě možné akce ($a^1$, $a^2$) ve stavu $s_0$, 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 1.11: Dlouhodobá odměna versus okamžitá

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

$$r = \alpha \cdot \left( r_t + \gamma \cdot \max_a Q(s_{t+1}, a) \right).$$

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

$$Q^{nová}(s_t, a_t) = (1-\alpha) \cdot Q(s_t, a_t) + r .$$

### 1.4.2 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 $\varepsilon$-greedy, kde $\varepsilon$ je pravděpodobnost náhodné akce. Tato pravděpodobnost bývá ze začátku velmi vysoká a klesá s časem, nikdy však neklesá na nulu.

### 1.4.3 Limitace Q-learningu

Mapování stavů na akce v Q-tabulce má několik nevýhod. Komplexnější úlohy, kde stav může být definován několika proměnnými najednou (3D prostor), 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.

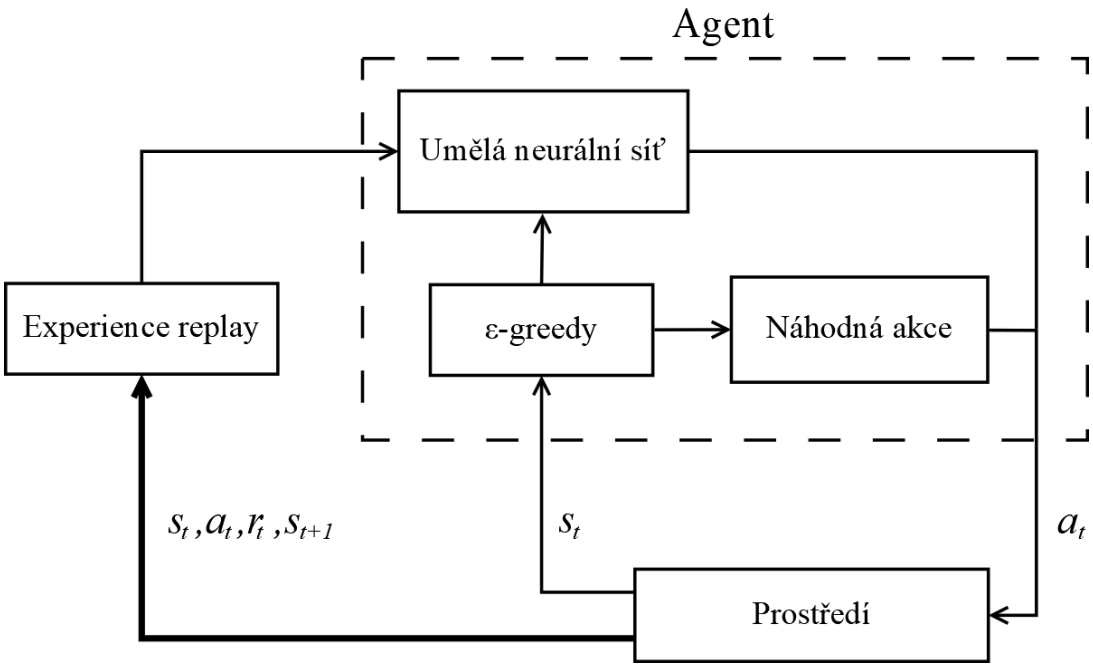
## 1.5 Deep Q-network

Deep Q-network (DQN) je algoritmus kombinující Q-learning a umělou neuronovou síť, publikovaný společností DeepMind v roce 2015 [8]. 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 stav $s_t$ a výstup bude reakce na tento stav $a_t$. 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

$$Q^{nová}(s_t, a_t) = r_t + \gamma \max_a Q(s_{t+1}, a) .$$

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 ($s_t$, $a_t$, $r_t$, $s_{t+1}$), 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**.




Obrázek 1.12: Schéma DQN

# 2 ZPŮSOBY ŘEŠENÍ, VYUŽITÉ POSTUPY A TECHNOLOGIE

Virtuální prostředí pro učení umělých inteligencí **NeuralQSandbox** je napsáno v jazyce Python. Python je jednoduchý na čtení, složité věci se v něm rychle realizují, podporuje paralelizaci a díky rozsáhlým matematickým frameworkům (PyTorch, Tensorflow, …), je ve světě umělých inteligencí často používaný.

## 2.1 Prostředí

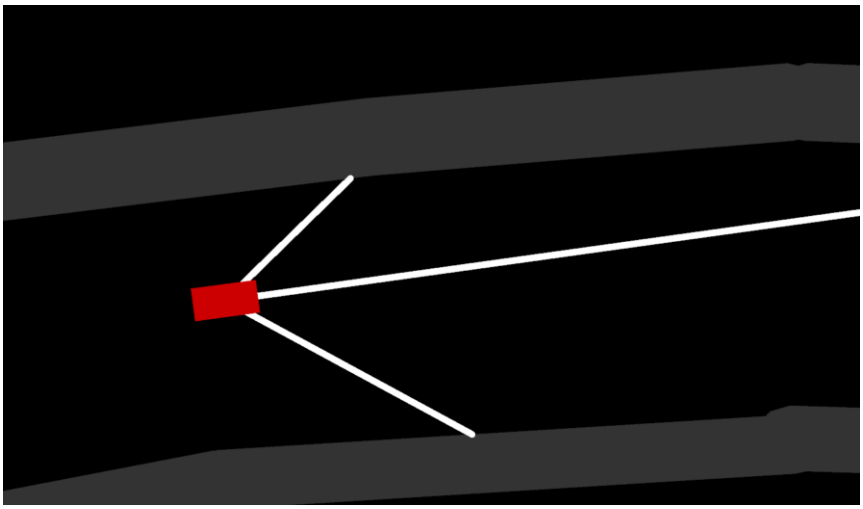
Prostředí je 2D rovina, kde se mohou přidávat statické (zdi, cíl, start) a dynamické (agent, hráč) objekty. Tyto objekty mezi sebou potom interagují a chovají se podle určitých pravidel (fyzika). Toto je implementováno pomocí fyzikální knihovny **Pymunk**. Pymunk je programovací rozhraní (API) mezi Pythonem a fyzikální knihovnou **Chipmunk**. Tato knihovna je napsaná v jazyce C, a je proto velmi rychlá a spolehlivá. Navíc je multiplatformní a v unixových systémech i plně paralelizována.

Vykreslení prostředí a ovládacích prvků včetně grafů, vyskakovacích oken, animací a podobně, je realizováno pomocí knihovny **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.

## 2.2 Umělá inteligence

Umělá inteligence je dosahována pomocí algoritmu DQN s vícevrstvým perceptronem. Pro samotné učení a predikci je použita knihovna **Keras**, což 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.

Agent měří několik vzdáleností od objektů pomocí paprsků. Počet těchto paprsků a úhel jejich natočení je možný měnit v uživatelském rozhraním. Tyto vzdálenosti jsou použity jako stavy pro agenta. Agent na tento stav může reagovat zahnutím vlevo, vpravo nebo neprovádět nic.

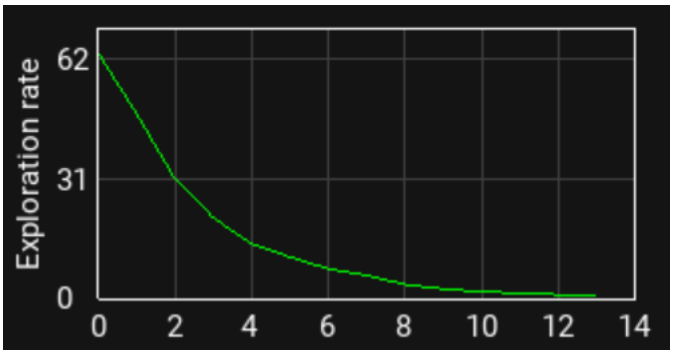


Obrázek 2.1: Agent a měření vzdálenosti

Agent je následně kladně odměňován za rychlost a čas přežití, negativní odměnu dostává za náraz a příliš blízké míjení zdí. Takto definované odměny fungují na uzavřené prostory a agent se postupně naučí zatáčet a reagovat na různé situace. Ideální by však bylo odměňovat agenta ještě za postup k cíli. Cíl ale není při vytváření prostředí nutný, a proto je zvolen obecnější způsob odměny.

Agent si každý fyzikální 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 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 fyzikální krok *t*, ale kvůli vysokým nárokům na výkon jsou prováděny každý třetí fyzikální krok.

Samozřejmě je dále implementovaná i explorační metoda $\varepsilon$-greedy, kde pravděpodobnost $\varepsilon$ se snižuje každý třetí fyzikální krok hned po výpočtu Q-hodnoty.




Obrázek 2.2: Zanesení explorace do grafu

## 2.3 Paralelizace

Projekt původně nebyl paralelizován a fungoval pouze na hlavním vlákně. S narůstající komplexitou ale bylo nutné zavést aspoň částečnou paralelizaci. Knihovny jako Pymunk a Tensorflow jsou sice plně paralelizovány, ale pokud jejich využití blokuje v hlavním vlákně jiný proces (vykreslování), tak tyto knihovny nejsou schopny využít plně svou kapacitu. Ideálně by tyto knihovny měly mít své vlákno, ale kvůli původně špatnému návrhu aplikace toto nebylo možné. Nakonec byla implementována aspoň částečná paralelizace. Hlavní vlákno vykresluje grafickou část a zpracovává uživatelské vstupy, druhé výpočetní vlákno zajišťuje fyzikální kroky a učení umělé neuronové sítě.

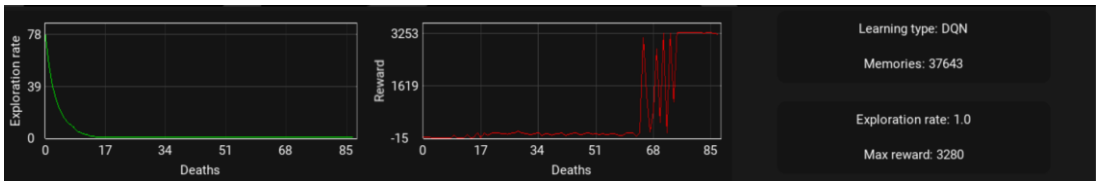
# 3 VYSLEDEK ŘEŠENÍ

## 3.1 Základní funkce aplikace

Aplikace NeuralQsandbox obsahuje několik různých módů, každý z těchto módů má svojí funkčnost a uživatel mezi jednotlivými módy přepíná pomocí grafického uživatelského rozhraní. Mezi aktivní módy patří: mód učení, testování a hraní. Při změně aktivního módu je nutný reset prostředí a učící smyčky, proto jsou tyto módy označovány jako *aktivní*. Mezi pasivní módy patří: editor úrovní, editor umělé neuronové sítě a nastavení. Uživatel může používat jednotlivé pasivní módy bez ohledu na aktivní (dokonce i editor úrovní). Například při učícím procesu může uživatel přepnout aplikaci do editoru úrovní, cokoliv změnit a přepnout zpět. Aplikace se sice pozastaví a počká na úpravy v editoru, ale jinak nebude funkčnost nijak ovlivněna.

## 3.2 Mód učení

Mód učení zajišťuje rozhraní mezi uživatelem a učícím procesem agenta. Pokud neexistuje již nahraný model (importovaný, dříve vytvořený), tak ho mód učení vytvoří a začne s učícím procesem. Informace o modelu a učení je možné zobrazit v kartě „Model info“, kde se v průběhu učení budou aktualizovat hodnoty. Tento přehled modelu je dostupný neustále i v jiných módech, avšak jenom v módu učení je aktualizován.




Obrázek 3.1: Model info

Mód učení nemá žádný cíl nebo konec, neustále učí agenta, dokud není přerušen uživatelem. Při přerušení učení se učený model uloží a při opětovném zapnutí učícího procesu učení pokračuje dále, nezačíná znovu (všechny vzpomínky jsou zachovány). Pokud ale uživatel resetuje model nebo importuje nový, učení se resetuje (vzpomínky, $\varepsilon$) a začíná od znovu.

Učený model není možné exportovat celý, pouze jeho umělou neurální síť. Proto učení na importovaném modelu začíná od začátku (nemá vzpomínky).

## 3.3 Mód testování

Pro otestování učeného modelu je možné kdykoliv opustit učící proces a zapnout mód testování. Tento mód sváže grafické vykreslování s fyzikou (1 snímek = 1 krok), tak aby rychlost agenta nebyla zkreslená, a používá pouze akce agenta bez náhodných akcí. V tomto módu také nedochází k učení, agent pouze projíždí úroveň pro testovací účely. Karta s informacemi o modelu zůstává stále stejná.

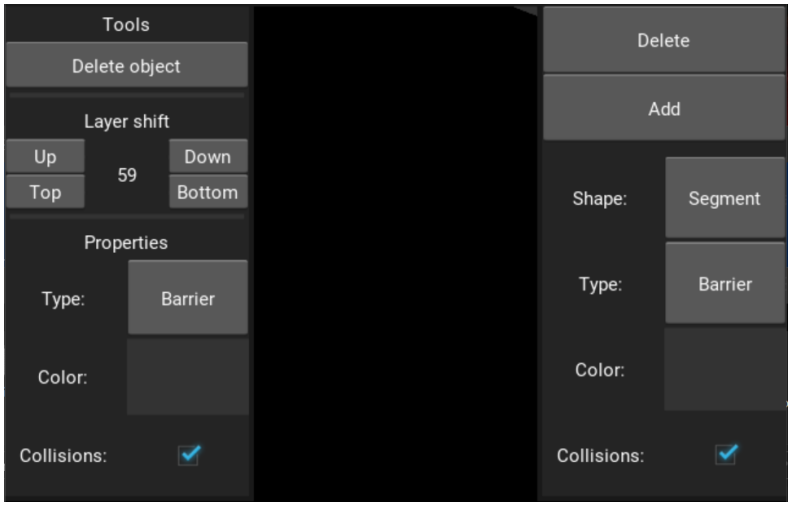
Mód testování dokáže také testovat importované modely, bez zapínání módu učení. Aplikace má v sobě několik již naučených modelů, které je možné vyzkoušet.

## 3.4 Mód hraní

Mód hraní umožňuje uživateli, vyzkoušet si ovládaní agenta pomocí klávesnice. Tento mód je užitečný pro testování nových úrovní.

## 3.5 Editor úrovní

Uživatel má možnost upravit nebo vytvořit úroveň, pomocí editoru úrovní. Tento editor je uživateli dostupný neustále, ať už chce změnit barvu, přidat objekt nebo vše smazat. Při vstupu do editoru se objeví panel s nástroji, v tomto panelu je možné přepínat mezi přidáváním a odebíráním objektů. Při přidávání je také možno nastavit tvar objektu, jeho typ (bariera, start, cíl), barvu a kolize. Při nechtěné změně je možné používat zkratku „CTRL+Z“, která vrací změny. Pro změnu pozice objektu ho stačí v editoru uchopit myší a přesunout na novou pozici (u tvaru segment je také možné použít dvojklik pro změnu pozice koncových bodů). Pro změnu jiných parametrů je třeba na objekt kliknout a po objevení editačního panelu je možné upravit vlastnosti objektu.

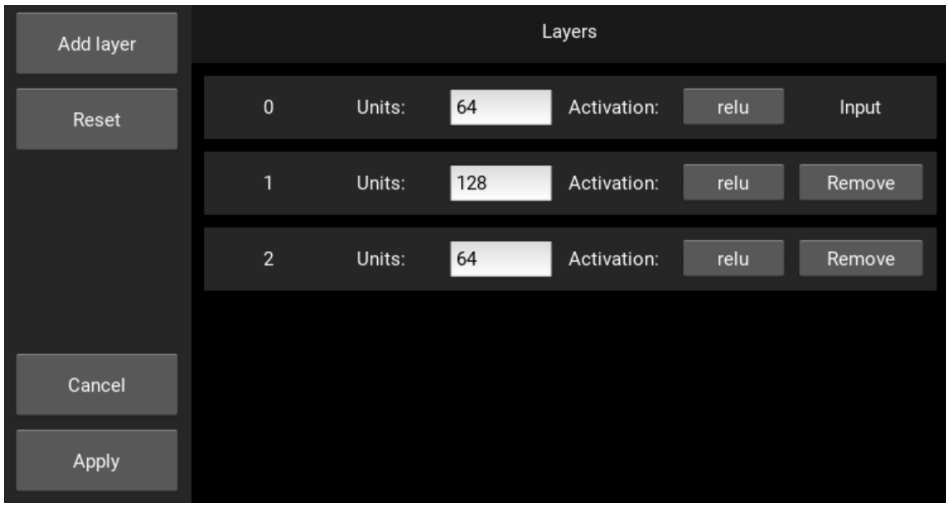



Obrázek 3.2: Panely pro ovládání editoru úrovní

Úrovně je možné exportovat a importovat. Exportované úrovně jsou ve formě binárního souboru vytvořeného pomocí modulu Pickle. Aplikace má uloženo několik již vytvořených úrovní, které je možné načíst a vyzkoušet.

## 3.6 Editor umělé neurální sítě

Pomocí editoru umělé neurální sítě je možné změnit počet vrstev a vlastnosti jednotlivých vrstev. Tyhle změny vedou ke změně chování modelu a jeho učení. Uživatel má takto možnost, měnit mozek agenta a pozorovat následky na jeho chování. Pro aktualizaci umělé neuronové sítě je ale nutné vytvořit nový model.




Obrázek 3.3: Editor umělé neuronové sítě

## 3.7 Nastavení

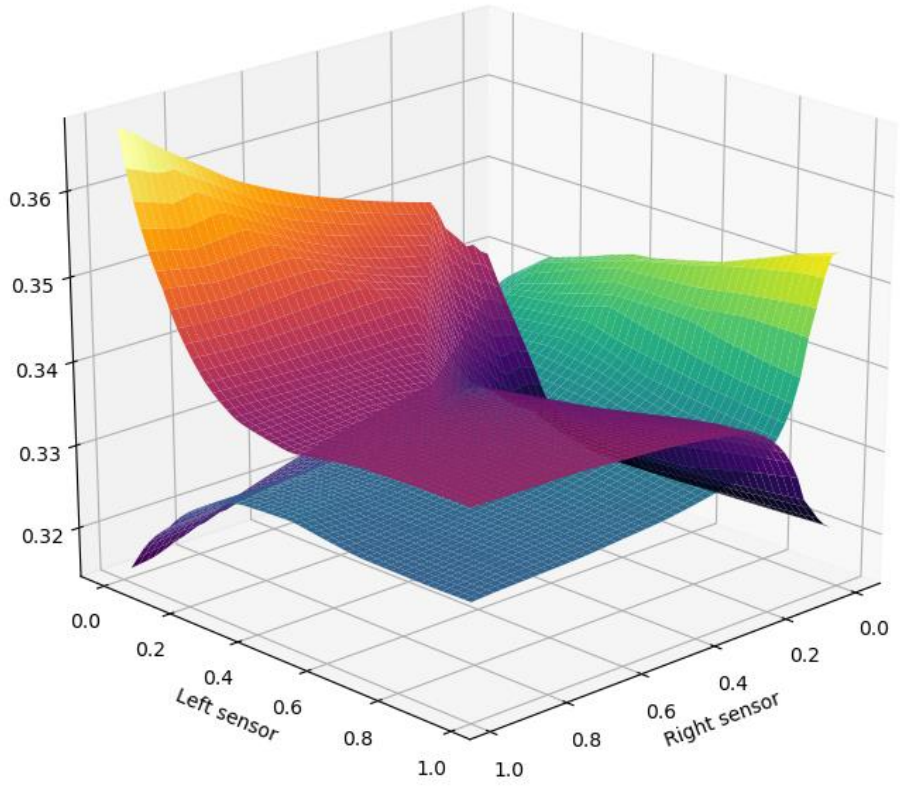
Algoritmus učení DQN má mnoho parametrů, které se musí před začátkem učení nastavit. Uživatel má možnost upravit tyto učící parametry a zkoušet jaký to bude mít vliv na agenta. Aplikace má zabudované odzkoušené hodnoty, které je vždy možné nahrát. Pomocí nastavení je také možné ovlivnit počet, náklon a vykreslování paprsků agenta.

Nastavení učících parametrů se aktualizuje pouze při vytváření modelu, pokud jíž existuje, je nutné ho smazat a vytvořit nový. Nový model je možné vytvořit v kartě „Model“.

## 3.8 Výsledky učení

Při správně zvolených učících parametrech je umělá neuronová síť schopná konvergovat během několika minut, což je oproti genetickým algoritmům velmi rychlé a efektivní. Naučené sítě jsou také schopny korektně reagovat na nové úrovně, prokazují tak schopnost generalizace. Graf na obrázku 3.4 znázorňuje závislost akce na stavech naučené umělé neurální sítě. Stavy jsou znázorněny rovinou *xy*, kde osa *x* znázorňuje měřenou vzdálenost levého sensoru a osa *y* sensoru pravého. V ose *z* (výška) je vynesen poměr predikované hodnoty akce k součtu hodnot všech akcí pro každý stav. Oranžová plocha znázorňuje akci vpravo

(*vpravo / (vlevo + vpravo + nic)*) a zelená plocha akci vlevo (*vlevo / (vlevo + vpravo + nic)*). Výška těchto akcí nám určuje *jistotu* modelu.




Obrázek 3.4: Závislost bočních senzorů a akcí

Na zobrazených datech můžeme pozorovat chování modelu. Když se levý senzor blíží k nule, má model větší tendenci odbočovat doprava a pokud se pravý senzor blíží k nule, reaguje spíše odbočením doleva. Zobrazené data jsou ale zkreslená, nezobrazují totiž akci bez odbočování a stavy jsou omezené pouze dvěma bočními senzory, ostatní stavy jsou konstantní.

# Závěr

Cílem projektu bylo pochopit a uplatnit teorii ohledně posilovaného učení umělých neuronových sítí a vytvořit konfigurovatelné prostředí, kde uživatel může ovládat učení, prostředí a umělou neuronovou síť.

Aplikace v momentálním stavu umožňuje uživateli zcela měnit prostředí, parametry algoritmu DQN a dokonce i měnit strukturu umělé neuronové sítě. Uživatel je takto schopen pozorovat chování agentů při změnách v prostředí nebo velikosti jejich sítě. Cíl práce byl tedy zcela naplněn.

V průběhu vývoje se ale naskytli vylepšení, které bych rád v budoucnu přidal. K těmto vylepšením patří například možnost měnit algoritmus učení, více stavových informací a grafických prvků celkově.

# SEZNAM POUŽITÝCH INFORMAČNÍCH ZDROJŮ


[1] BOČÁN, Hynek. *Posilované učení pro hraní robotického fotbalu* [online]. Brno, 2017 [cit. 2019-12-22]. Dostupné z: https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=158429. Bakalářská práce. VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ. Vedoucí práce Doc. RNDr. PAVEL SMRŽ, Ph.D.

[2] HARRIS, Naftali. *Visualizing K-Means Clustering* [online]. January 19, 2014 [cit. 2019-12-22]. Dostupné z: https://www.naftaliharris.com/blog/visualizing-k-means-clustering/

[3] JULIANI, Arthur. *Simple Reinforcement Learning with Tensorflow* [online]. Aug 25, 2016 [cit. 2019-12-22]. Dostupné z: https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-0-q-learning-with-tables-and-neural-networks-d195264329d0

[4] KANTOR, Jan. *Učení bez učitele* [online]. Brno, 2008 [cit. 2019-12-22]. 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. 2019-12-22]. Dostupné z: https://dspace.cuni.cz/handle/20.500.11956/90569. Diplomová práce. Univerzita Karlova, Matematicko-fyzikální fakulta.

[6] LIN, Long-Ji. *Self-Improving Reactive Agents Based On Reinforcement Learning, Planning and Teaching* [online]. School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213: Kluwer Academic, 1992 [cit. 2019-12-24]. Dostupné z: http://www.incompleteideas.net/lin-92.pdf

[7] MAINI, Vishal. *Machine Learning for Humans* [online]. Aug 19, 2017 [cit. 2019-12-22]. Dostupné z: https://medium.com/machine-learning-for-humans/unsupervised-learning-f45587588294

[8] MNIH, Volodymyr et al. *Human-level control through deep reinforcement learning* [online]. London: Macmillan Publishers, 2015 [cit. 2019-12-22]. Dostupné z: http://files.davidqiu.com//research/nature14236.pdf

[9] MNIH, Volodymyr et al. *Playing Atari with Deep Reinforcement Learning* [online]. 2013 [cit. 2019-12-22]. Dostupné z: https://arxiv.org/pdf/1312.5602.pdf

[10] RIVLIN, Or. *Reinforcement Learning with Hindsight Experience Replay* [online]. Jan 31, 2019 [cit. 2019-12-22]. Dostupné z: https://towardsdatascience.com/reinforcement-learning-with-hindsight-experience-replay-1fee5704f2f8

[11] SENO, Takuma. *Deep Reinforcement Learning* [online]. Oct 20, 2017 [cit. 2019-12-22]. Dostupné z: https://towardsdatascience.com/welcome-to-deep-reinforcement-learning-part-1-dqn-c3cab4d41b6b

[12] ZHANG, Shangtong a Richard S. SUTTON. *A Deeper Look at Experience Replay* [online]. University of Alberta, 2018 [cit. 2019-12-22].