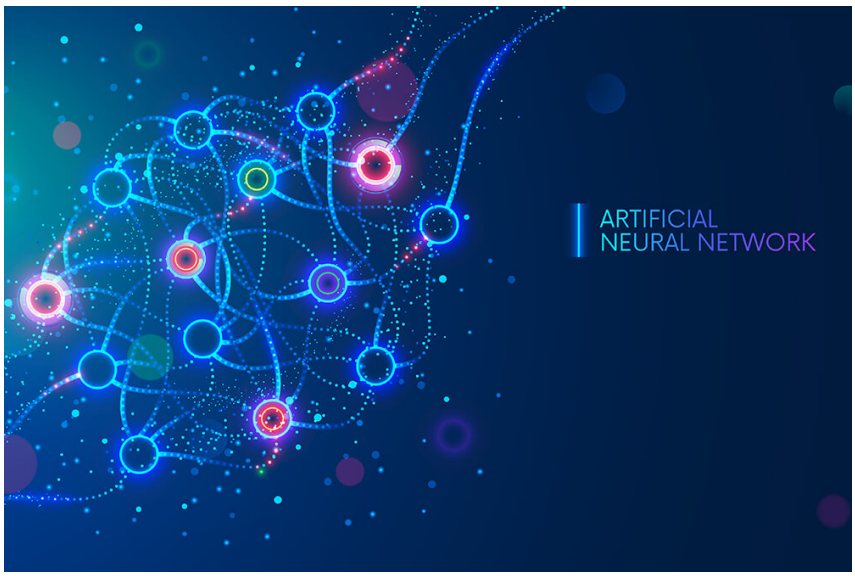
SŠPU
STŘEDNÍ ŠKOLA PRŮMYSLOVÁ
A UMĚLECKÁ, OPAVA

ZÁVĚREČNÁ STUDIJNÍ PRÁCE

dokumentace

NeuralPath Labyrinth



Autor: Lukáš Hrňa

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

Třída: IT4

Školní rok: 2023/24

Poděkování

Prostor k poděkování Tučňákovi.

Prohlášení

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

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

V Opavě 1. 1. 2024

. . . . . .

Podpis autora

Abstrakt

Tento projekt se zaměřuje na aplikaci umělé inteligence pro řešení jednoduchých bludišť. Využíváme PyTorch, knihovnu pro strojové učení, nástroj pro vývoj a testování algoritmů posilovaného učení. Cílem je vyvinout a trénovat model AI, který efektivně prochází prostředím generovaných bludišť. Model je navržen pro rozpoznávání vzorů a navigaci různými konfiguracemi bludišť s cílem najít cestu ven.

Klíčová slova

Umělá inteligence, Posilované učení, PyTorch, Navigace bludištěm, Hluboké neuronové sítě, Prostorová navigace, Algoritmy pro hledání cesty, Generování bludišť

Abstract

This project focuses on the application of artificial intelligence to solve simple mazes. We utilize PyTorch, a machine learning library, a tool for developing and testing reinforcement learning algorithms. The goal is to develop and train an AI model capable of efficiently navigating through the environment of generated mazes. The model is designed to recognize patterns and navigate through various maze configurations to find a way out.

Keywords

Artificial Intelligence, Reinforcement Learning, PyTorch, Maze Navigation, Neural Networks, Spatial Navigation, Pathfinding Algorithms, Maze Generation

Obsah

Úvod

V této závěrečné studijní práci se věnujeme fascinujícímu tématu aplikace umělých neuronových sítí v prostředí 2D bludiště. Cílem práce je prozkoumat a demonstrovat, jak mohou technologie umělé inteligence, zejména neuronové sítě, efektivně řešit složité úlohy navigace a rozhodování v dynamických a neznámých prostředích.

Zaměřujeme se především na teoretické základy neuronových sítí, včetně jejich struktury, typů učení a klíčových algoritmů, jako je Q-learning a jeho rozšíření v Deep Q-networks. Tyto metody poskytují základní rámec pro porozumění tomu, jak AI modely zpracovávají informace, učí se z získaných dat a adaptují se na nové situace v prostředí bludiště.

Praktická část práce zahrnuje vývoj vlastního 2D bludiště, které slouží jako testovací prostředí pro umělou inteligenci. Využíváme programovací jazyk Rust pro generování bludiště a Python s knihovnami jako PyTorch pro vývoj a trénování AI modelu. Dále popisujeme, jakým způsobem AI interaguje s bludištěm, jaké metody a nástroje jsou využívány pro sledování a zaznamenávání jejího chování a jaké výzvy tento proces přináší.

1 UMĚLÉ NEURONOVÉ SÍTĚ

1.1 ÚVOD

Tato kapitola slouží jako úvod do tématu umělých neuronových sítí. Zabývá se vysvětlením základních pojmů, popisem matematického modelu neuronu a jeho schopností klasifikace. Klíčovým prvkem je pochopení matematického modelu jednoho neuronu, což je základ pro pochopení komplexnějších struktur tvořených propojenými neurony v rámci umělých neuronových sítí

Hlavním cílem tohoto projektu je využití a demonstrace principů posilovaného učení, konkrétně metody Q-learningu, pro navigaci umělé inteligence v prostředí bludiště.

1.2 TYPY UČENÍ

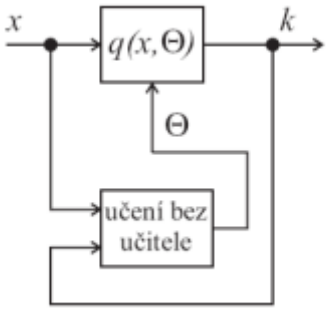
Učení v umělé inteligenci lze rozdělit do několika základních kategorií, z nichž nejvýznamnější jsou učení s učitelem, bez učitele a posilované učení. Každý typ má své specifické charakteristiky a využití.

1.2.1 Unsupervised learning

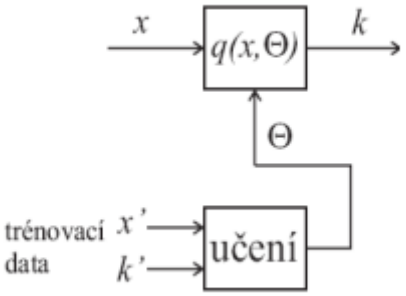
Učení bez učitele, nebo unsupervised learning, pracuje s neoznačenými daty. Cílem je najít skryté vzory nebo struktury v datech, aniž by byly k dispozici předem definované odpovědi nebo labely. Typickými příklady využití jsou klastrování, kde je úkolem najít skupiny podobných prvků v datech, nebo redukce dimenzionality, která slouží k zjednodušení komplexních datových sad. Učení bez učitele je klíčové v situacích, kdy je třeba rozpoznat inherentní strukturu dat bez předchozích znalostí o tom, co data přesně obsahují.

1.2.2 Supervised learning

Učení s učitelem, nebo supervised learning, je proces, kde model umělé inteligence je trénován na datové sadě, která obsahuje jak vstupní data, tak správné odpovědi (labely). Tento přístup



umožňuje modelu naučit se předpovídat výstupy na základě nových vstupů na základě předchozích zkušeností. Tento typ učení je široce využíván v aplikacích jako je klasifikace (například rozpoznávání objektů na obrázcích) a regrese (například předpovídání cen nemovitostí), kde je důležité, aby model byl schopen správně identifikovat a reagovat na různé druhy dat.

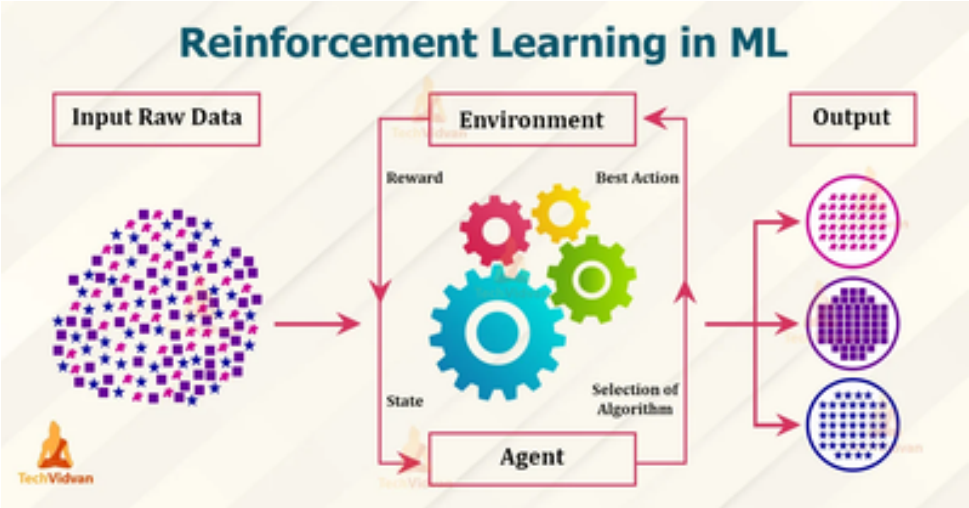


1.2.3 Reinforcement learning

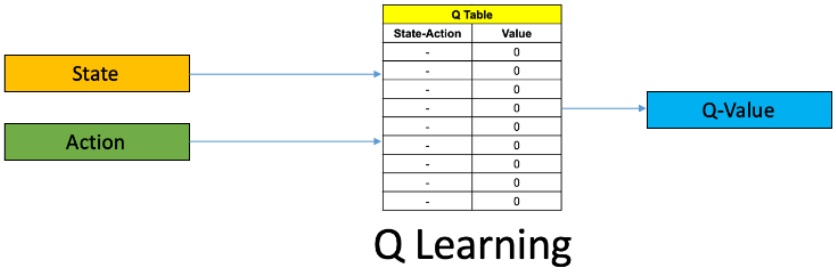
Posilované učení, nebo reinforcement learning, je zase odlišné tím, že se zaměřuje na vývoj algoritmů, které se učí prostřednictvím interakce s prostředím a získávají zpětnou vazbu v podobě odměn nebo trestů. V tomto přístupu je agent, kterým může být například robot nebo software, vystaven prostředí, ve kterém se snaží provádět akce vedoucí k maximální možné odměně. Posilované učení je ideální pro situace, kde jsou rozhodnutí časově závislá a kde je třeba vzít v úvahu dlouhodobé důsledky akcí, jako jsou strategické hry, robotická navigace nebo autonomní řízení vozidel.

Pro maximalizaci celkové odměny je nezbytné, aby agent volil optimální akce v závislosti na svém současném stavu. Tento výběr akcí, známý jako strategie nebo politika π , je klíčový pro přiřazení nejvhodnější akce pro každý stav. Existuje mnoho algoritmů pro určení efektivní funkce π , nicméně zde se zaměříme specificky na Q-learning a jeho odvozené algoritmy.

1.3 Q-LEARNING



Q-learning je algoritmus založený na principu posilovaného učení. Na rozdíl od metod, které přímo hledají optimální strategii π (metody policy gradient), Q-learning postupuje nepřímo prostřednictvím tzv. Q-tabulky. Tato tabulka spojuje stavy s akcemi - každý řádek reprezentuje jeden stav a sloupce v tomto řádku různé akce. Agent vybírá akci s nejvyšší hodnotou v tabulce. Tyto hodnoty jsou známé jako Q-hodnoty, kde "Q"reprezentuje "quality"(kvalitu) akce. Algoritmus, který určuje nejlepší akci z Q-tabulky, je označován jako Q-funkce. Aktualizace

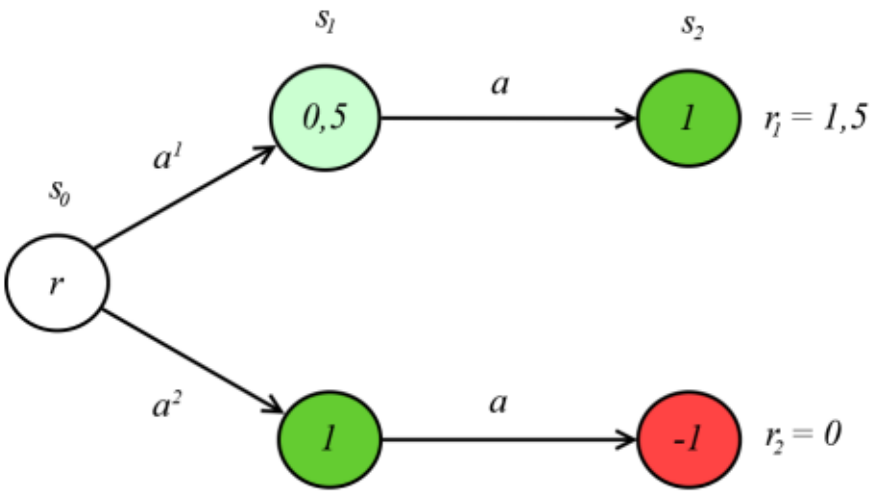


Q-hodnot se provádí na základě odměn, které agent obdrží od prostředí za provedenou akci. Tato odměna, známá jako posílení, může být pozitivní, negativní (trest) nebo neutrální. Když agent obdrží za určitou akci negativní odměnu, dojde k snížení příslušné hodnoty v Q-tabulce, což vede k nižší pravděpodobnosti opětovného výběru této akce. Takto lze postupně eliminovat nežádoucí chování a podporovat chování, které je žádoucí.

1.3.1 Okamžitá a dlouhodobá odměna

Aktualizace Q-hodnot lze rozdělit na dvě zásadní části: okamžitou a dlouhodobou. Okamžitá část je reprezentována odměnou r_t, kterou agent obdrží za provedení akce a_t. K této okamžité odměně je pak přičítána dlouhodobá hodnota, reprezentující očekávané budoucí Q-hodnoty ze stavu s_{t+1}. Cílem je maximalizovat celkovou odměnu, zahrnující nejen aktuální, ale i budoucí akce. Například, v případě dvou možných akcí (a_1, a_2) ve stavu s_0 může být na první pohled

akce s vyšší okamžitou odměnou méně výhodná v dlouhodobém horizontu, pokud by v důsledku jejího provedení následovaly stavy vedoucí k nižším celkovým odměnám.



[h]

Je klíčové najít správnou rovnováhu mezi okamžitou a dlouhodobou odměnou. Úloha dlouhodobé odměny může být upravena pomocí diskontního faktoru (discount factor) γ, který určuje váhu budoucích odměn, zatímco míra učení (learning rate) α ovlivňuje rychlost aktualizace Q-hodnot. Celkovou odměnu r, skládající se z okamžité a dlouhodobé složky, lze vyjádřit jako součet těchto dvou komponent, s přihlédnutím k váze diskontního faktoru a míry učení.

$$r = a.(r_t + \gamma.maxQ(s_{t+1,a})) \tag{1.1}$$

V Q-tabulce se Q-hodnota pro daný stav a akci obvykle nepřepisuje přímo novou odměnou, ale upravuje se na základě již existující hodnoty. Toto se provádí za účelem zachování historických informací o výkonnosti dané akce. Aktualizační pravidlo pro Q-hodnoty tedy bude kombinovat aktuální odměnu s předešlými poznatky o daném stavu a akci, čímž se dosahuje efektivnějšího a přesnějšího učení.

$$Q^{nova}(s_t, a_t) = (1 - \alpha).Q(s_t, at) + r \tag{1.2}$$

1.3.2 Limitace Q-learningu

Při použití Q-tabulky pro mapování stavů na akce v rámci Q-learningu se můžeme setkat s několika komplikacemi. Pro složitější úlohy, kde je stav určen několika proměnnými současně (například v 3D prostoru), může velikost Q-tabulky dosáhnout extrémních rozměrů, což vede k její nepraktičnosti. Navíc, protože stavy musí tvořit konečnou sadu, řešení spojitých stavů se stává problematickým. Další výzvou je manipulace s akcemi: v základním modelu Q-learningu jsou akce pouze přijaty nebo odmítnuty (binární výběr), což znemožňuje spojité řízení výstupu. Tyto problémy mohou být řešeny implementací umělých neuronových sítí, které nabízí flexibilnější a efektivnější přístup.

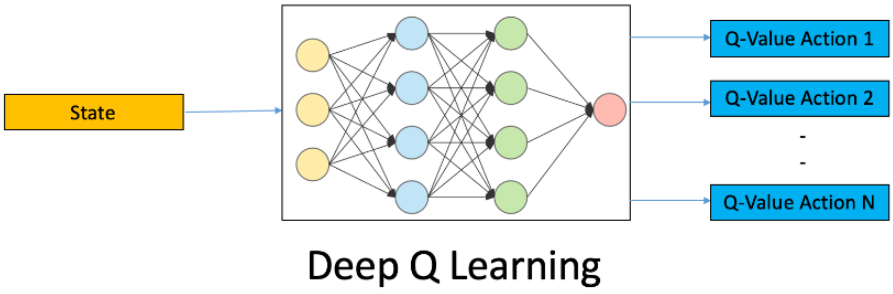
1.3.3 Deep Q-network

Deep Q-network (DQN) je algoritmus, který byl vytvořen společností DeepMind v roce 2015. Tento algoritmus spojuje principy Q-learningu s umělou neuronovou sítí. Základním konceptem DQN je využití neuronové sítě, známé jako Q-síť, která nahrazuje tradiční Q-tabulku. Úkolem této sítě je co nejvíce přiblížit funkci Q-hodnoty. Vstupem do Q-sítě je stav s_t, a výstupem je reakce na tento stav a_t, což je podobné Q-tabulce. Nicméně, proces učení Q-sítě se liší od Q-tabulky, protože zde nelze jednoduše aktualizovat Q-hodnoty. Místo toho se Q-síť musí učit stejným způsobem jako jakákoliv jiná neuronová síť, a to využitím učení s učitelem, kde jsou všechny akce agenta stále hodnoceny pomocí odměňovací funkce a generují trénovací data.

Další rozdíl oproti klasickému Q-learningu je v aktualizačním pravidle. Vzhledem k tomu, že neuronovou síť neměníme přímo, vynecháváme část aktualizačního pravidla, která připočítává starou Q-hodnotu. Toto pravidlo aktualizace pak nabývá nové formy, která efektivně využívá schopnosti neuronové sítě k učení a přizpůsobení.

$$Q^{nova}(s_t, a_t) = r_t + \gamma maxQ(s_{t+1}, a) \tag{1.3}$$

Neuronové sítě jsou efektivnější, když se učí na základě více dat, než jen jednoho výstupu. Efektivnějším přístupem je uchovávání historie předchozích stavů agenta. Zaznamenáváme stav, akci, odměnu a následující stav (s_t, a_t, r_t, s_{t+1}) - všechny elementy potřebné pro výpočet nových Q-hodnot. Během učení pak agent vybírá náhodně uspořádanou vzorku pevně stanovené velikosti z těchto uložených záznamů. Na základě této vzorky se počítají nové Q-hodnoty, které jsou poté uplatněny na neuronovou síť pomocí algoritmu zpětného šíření chyb. Tento proces uchovávání minulých stavů se nazývá metoda opakování zkušeností (experience replay).



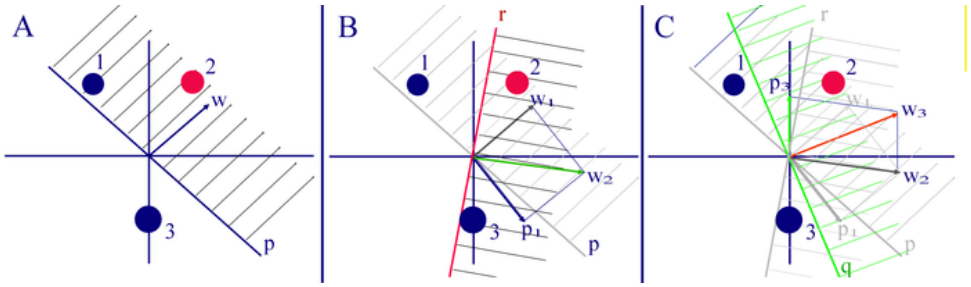
1.4 PERCEPTRON

Perceptron v nejjednodušší podobě je binární klasifikátor, který mapuje vektory vstupů $\mathbf{x} = [x_1, x_2, \ldots, x_n]$ na výstupní hodnoty $f(\xi)$:

$$f(\xi) = \begin{cases} 1 & \text{pro } \mathbf{w} \cdot \mathbf{x} - \theta \geq 0 \\ 0 & \text{jindy} \end{cases}$$

kde $\mathbf{w}$ je vektor vah a θ je práh citlivosti neuronu.

Celkový podnět neuronu udává vážený součet $\xi = \sum_{i=1}^{n} w_i \cdot x_i - \theta$. Tento celkový podnět bývá označován jako potenciál neuronu. Na potenciál reaguje neuron (perceptron) výstupní odezvou $Z = f(\xi)$, kde f je tzv. přenosová funkce. Je-li přenosová funkce ve tvaru skokové funkce, perceptron funguje, jak již bylo řečeno, jako binární klasifikátor, neuron tedy dělí vstupní prostor na dvě části. Tři fáze učení jednoduchého perceptronu jsou následující:



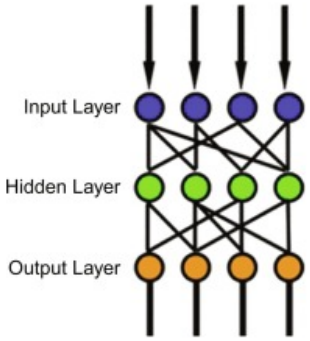
A: Je zvolen náhodně vektor vah $\mathbf{w}$ a k němu určena kolmá rozhodovací hranice p. Zjišťujeme výstup pro bod 1 - leží v oblasti s výstupem 1, i když má ležet v oblasti s výstupem 0.

B: Odečteme vektor bodu 1 od vektoru vah $\mathbf{w_1}$ a získáme nový vektor vah $\mathbf{w_2}$ a k němu příslušnou rozhodovací hranici znázorněnou přímkou r. Bod 2 je umístěn správně, bodu 3 přiřadí síť hodnotu 1 i když má dostat výstup 0.

C: Odečteme vektor bodu 3 od vektoru vah $\mathbf{w_2}$ a získáme nový vektor vah $\mathbf{w_3}$. Nyní je již problém vyřešen - všem bodům je přiřazen odpovídající hodnota výstupu. Řešením problému je tedy vektor vah $\mathbf{w_3}$ s příslušnou rozhodovací hranicí q.

1.5 MULTI LAYER PERCEPTRON(MLP)

MLP je rozšířením dopředné neuronové sítě. Skládá se ze tří typů vrstev - vstupní vrstvy, výstupní vrstvy a skrytých vrstev. Vstupní vrstva přijímá vstupní signál k zpracování. Výstupní vrstva provádí požadované úlohy, jako jsou predikce a klasifikace. Libovolný počet skrytých vrstev, umístěných mezi vstupní a výstupní vrstvou, představuje skutečný výpočetní motor MLP. Podobně jako u dopředné sítě v MLP proudí data směrem dopředu od vstupní k výstupní vrstvě. Neurony v MLP jsou trénovány pomocí algoritmu učení zpětné propagace. MLP jsou navrženy tak, aby aproximovaly jakoukoli spojitou funkci a mohly řešit problémy, které nejsou lineárně separovatelné. Hlavními využitími MLP jsou klasifikace vzorů, rozpoznávání, predikce a aproximace.



$$o_x = G(b_2 + W_2 h_x) \tag{1.4}$$

$$h_x = \Phi(x) = s(b_1 + W_1 x) \tag{1.5}$$

s vektory zkreslení b_1, b_2; maticemi vah W_1, W_2 a aktivačními funkcemi G a s. Sada parametrů k učení je sada $\theta = \{W_1, b_1, W_2, b_2\}$. Typické volby pro s zahrnují tanh funkci s $\tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$ nebo logistickou sigmoidní funkci, s $\text{sigmoid}(a) = \frac{1}{1+e^{-a}}$.

2 VYUŽITÉ POSTUPY A TECHNOLOGIE

V průběhu projektu byly využity následující postupy a technologie:

- **Posilované učení (Reinforcement Learning):** Implementace metody Q-learning pro efektivní rozhodování a adaptaci umělé inteligence v dynamických prostředích, což je klíčové pro navigaci v bludišti.
- **Python a PyTorch:** Použití programovacího jazyka Python ve spojení s PyTorch, oblíbeným nástrojem pro strojové učení. PyTorch je významný svým použitím tensorů, které jsou základem pro modelování a trénování neuronových sítí.
- **Raylib v Rustu pro vykreslení prostředí:** Využití knihovny Raylib v programovacím jazyce Rust pro efektivní vykreslení a vizualizaci 2D bludiště, poskytující hladkou grafiku a interakci s herním prostředím.

2.1 2D BLUDIŠTĚ OVLÁDANÉ UMĚLOU INTELIGENCÍ

2.1.1 Vytvoření a interakce s prostředím

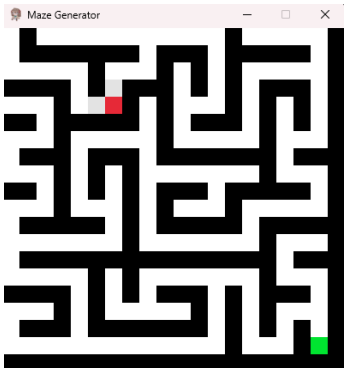
V rámci tohoto projektu je implementováno 2D bludiště, navigované umělou inteligencí. Klíčovou roli v procesu vývoje a tréninku AI modelu hraje PyTorch, pokročilá knihovna pro strojové učení v Pythonu. PyTorch poskytuje efektivní nástroje pro práci s neurálními sítěmi, zvláště díky svým flexibilním a výkonným tensorům.

2.1.2 Ovládání bludiště

Interakce mezi AI a bludištěm je realizována prostřednictvím knihovny pyautogui v Pythonu. Tato knihovna umožňuje AI ovládat herní prostředí simulací klávesových a myšových akcí, což je nezbytné pro dynamické testování a experimenty.

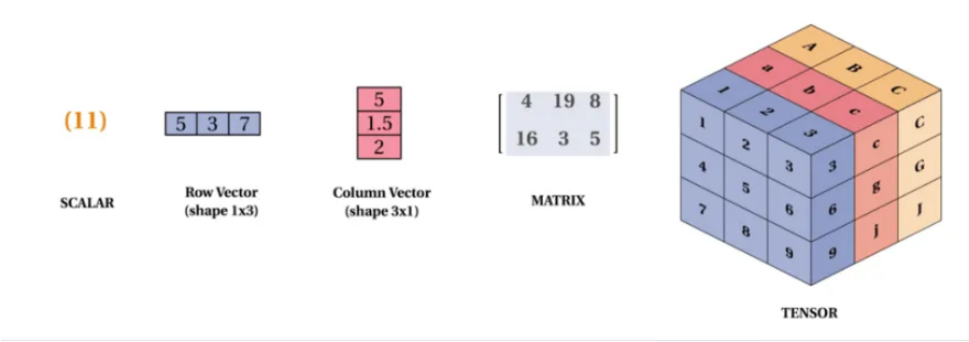
2.1.3 Vykreslení prostředí

Bludiště je vykresleno pomocí knihovny Raylib v jazyce Rust, která poskytuje efektivní a snadné prostředky pro tvorbu her a vizualizaci herních prvků. Tento výběr zaručuje hladké a realistické grafické zobrazení prostředí pro AI. [h]



2.1.4 Tensorová reprezentace v PyTorch

Tensor je základním stavebním kamenem v PyTorch, poskytující mnohovrstevnou, multidimenzionální matici. Tensory v PyTorch slouží jako kontejnery pro data, která jsou zpracovávána a analyzována neuronovými sítěmi. Díky své vysoké efektivitě a flexibilitě jsou tensory nezbytné pro optimalizaci výpočtů a umožňují efektivně manipulovat s velkými datovými sadami.



[h]

2.1.5 Využití tensorů v AI navigaci

V kontextu AI navigace bludištěm jsou tensory v PyTorch využívány pro reprezentaci a zpracování stavů, akcí a odměn v procesu učení. Tensory umožňují rychlé a efektivní výpočty potřebné pro dynamické aktualizace Q-hodnot v procesu Q-learningu a zajišťují efektivní trénink a optimalizaci neuronových sítí.

3 Řešení problému

3.1 Generování Bludiště pro AI

Vývoj prostředí pro umělou inteligenci zahrnuje tvorbu bludiště, které AI musí prozkoumat a řešit. Následující fragment kódu v jazyce Rust demonstruje algoritmus pro generování bludiště. Tento algoritmus je základním kamenem pro vytváření dynamických a výzvou bohatých prostředí, ve kterých se AI může učit a rozvíjet.

Kód 3.1: Funkce pro generování bludiště v Rustu

```
fn generate_maze(&mut self, x: usize, y: usize) {
        self.grid[y][x] = 0;
        let mut dirs = [(0, -1), (1, 0), (0, 1), (-1, 0)];

        dirs.shuffle(&mut self.rng);

        for (dx, dy) in dirs {
                let nx = (x as i32 + dx * 2) as usize;
                let ny = (y as i32 + dy * 2) as usize;

         if nx < GRID_WIDTH as usize && ny < GRID_HEIGHT
          as usize && self.grid[ny][nx] == 1 {
                self.grid[y.wrapping_add(dy as usize)]
                [x.wrapping_add(dx as usize)] = 0;
                self.generate_maze(nx, ny);
                }
        }
}
```

Tento algoritmus využívá rekurzivní způsob k procházení a otevírání cest v mřížce, která reprezentuje bludiště, čímž vytváří cesty a slepé uličky. Výsledná struktura bludiště pak poskytuje komplexní prostředí pro testování a vývoj algoritmů umělé inteligence.

3.2 Ukládáni a práce s daty

Pro ukládání dat o pohybech v bludišti je implementována funkce `write_to_csv`, která přijímá data o pohybech a boolean hodnotu indikující, zda bylo bludiště úspěšně dokončeno. Tato funkce vytvoří CSV soubor a zapíše do něj hlavičku a následně záznamy o jednotlivých pohybech.

3.2.1 Implementace funkce write_to_csv

Kód 3.2: Funkce pro zápis do CSV souboru

```
fn write_to_csv(moves_data: &MovesData, is_completed: bool) {
        let file_path = "moves_data.csv";
        let mut file = File::create(file_path);

        // Write the header
        writeln!(file, "move;possible_moves;completion_message")

        // Process and write moves data to CSV
        for (move_key, possible_moves) in &moves_data.moves {
                // ... (zde pokracuje logika pro zapis dat)
        }
}
```

Tato funkce přepisuje stávající CSV soubor při každém volání, což zajišťuje, že data jsou aktuální. V případě, že je bludiště dokončeno, poslední záznam v CSV bude obsahovat speciální značku indikující úspěšné dokončení.

3.2.2 Zpracování dat a jejich sledování

Systém sledování souborů je klíčový pro automatizaci procesu zpracování dat. V našem případě jsme implementovali třídu `CSVFileMonitor` v Pythonu, která nám umožňuje monitorovat a reagovat na změny v CSV souboru v reálném čase. Tato třída využívá knihovnu `watchdog`, která poskytuje široké možnosti pro sledování souborového systému, a knihovnu `csv` pro zpracování CSV souborů. Knihovna `watchdog` byla vybrána pro svou efektivitu a přenositelnost mezi různými operačními systémy, zatímco `csv` je standardním nástrojem pro práci s CSV soubory v Pythonu, což zajišťuje snadnou manipulaci s daty.

3.2.3 Inicializace a konfigurace monitoru

Monitorovací třída `CSVFileMonitor` je inicializována s cestou k sledovanému souboru a adresáři. Může být také konfigurována s callback funkcí, která se vyvolá, když dojde k detekci změny v souboru.

Kód 3.3: Inicializace monitoru souboru

```
class CSVFileMonitor:
def __init__(self, file_path, directory_path, callback=None):
self.file_path = file_path
self.directory_path = directory_path
self.last_processed_line = None
self.observer = Observer()
self.callback = callback
```

3.2.4 Detekce změn v souboru

Pro detekci změn v souboru používáme metodu `has_file_updated`, která porovnává čas poslední modifikace souboru s uloženým časem poslední kontroly. Pokud dojde ke změně, aktualizuje se čas poslední kontroly a metoda vrací `True`, což signalizuje, že soubor byl upraven.

Kód 3.4: Detekce změn v CSV souboru

```
def has_file_updated(self):
current_modification_time = os.path.getmtime(self.file_path)
if current_modification_time > self._last_file_modification_time:
self._last_file_modification_time = current_modification_time
return True
return False
```

Tato metoda je zásadní pro efektivní sledování souboru bez nepotřebného čtení celého souboru při každém průchodu kontrolní smyčkou. Pouze změny, které jsou skutečně zjištěny, vedou k dalšímu zpracování, čímž se optimalizuje výkon a reaktivita systému.

3.2.5 Sledování a reakce na události souborového systému

Pro sledování událostí souborového systému využíváme knihovnu `watchdog`. Instance `Observer` z knihovny je použita k naplánování sledování, což umožňuje spustit předdefinovanou akci vždy, když dojde ke změně ve sledovaném souboru.

Kód 3.5: Sledovani a reakce na zmeny souboru

```
def start(self):
event_handler = FileSystemEventHandler()
event_handler.on_modified = lambda event: \
self.process_file() if event.src_path == self.file_path else None
self.observer.schedule(event_handler, path=self.directory_path, r
self.observer.start()
```

Tato podsekce podrobně popisuje mechanismus sledování a reakce na změny v CSV souboru, který je základem pro dynamické zpracování dat v reálném čase. Výběr knihoven `watchdog` a `csv` je zdůvodněn jejich vysokou efektivitou a snadným použitím v Pythonu, což umožňuje flexibilní a robustní řešení pro monitorování souborů.

3.2.6 Získání informací o aktuálním stavu hry

Pro získání důležitých informací o aktuálním stavu hry jsou využívány dvě funkce: `get_possible_moves` a `get_completion_message`.

Funkce `get_possible_moves` vrací seznam možných pohybů v aktuálním kroku hry. Tento seznam je extrahován z posledního zpracovaného řádku CSV souboru a je určen k informování AI o dostupných možnostech pohybu.

Kód 3.6: Získání seznamu možných pohybů

```
def get_possible_moves(self):
if self.last_processed_line:
return self.parse_list(self.last_processed_line[1])
return None
```

Druhá funkce, `get_completion_message`, zjišťuje, zda bylo bludiště úspěšně dokončeno. Tato informace je také extrahována z posledního řádku CSV souboru a je používána k určení, zda AI dosáhla cíle.

Kód 3.7: Zjištění dokončení bludiště

```
def get_completion_message(self):
if self.last_processed_line:
return int(self.last_processed_line[2])
return None
```

Tyto funkce jsou zásadní pro dynamickou interakci AI s prostředím, poskytují nezbytné informace pro rozhodování a strategii hry.

3.3 Implementace Algoritmu Q-Learning

Algoritmus Q-learning je implementován několika zásadními funkcemi, které zajišťují různé aspekty učícího se procesu. Níže jsou detailněji popsány klíčové funkce použité ve skriptu:

3.3.1 Funkce Epsilon-Greedy Policy

```
def epsilon_greedy_policy(state, valid_moves, epsilon):
if random.uniform(0, 1) < epsilon:
action = random.choice(valid_moves)
print(f"Exploring: Chosen action {action} from {valid_moves}")
return action
else:
valid_q_values = Q_table[state, valid_moves]
action = valid_moves[torch.argmax(valid_q_values).item()]
print(f"Exploiting: Chosen action {action} with Q-values {valid_q
return action
```

Tato funkce rozhoduje mezi prozkoumáním nových možností a využitím stávajících znalostí. Rozhodnutí je založeno na proměnné '$exploration_prob$'. Funkce náhodně vybere akci pro prozkoumání nebo akci s nejvyšší hodnotou Q pro využití znalostí.

3.3.2 Funkce Provedení Pohybu

```
def execute_move(action):
if action == 0:    # Up
pyautogui.press('w')
elif action == 1: # Left
pyautogui.press('a')
elif action == 2: # Down
pyautogui.press('s')
elif action == 3: # Right
pyautogui.press('d')
```

Tato funkce překládá vybranou akci na příkaz stisku klávesy pomocí 'pyautogui'. Je to způsob, jakým agent interaguje s prostředím - každá akce (0 až 3) odpovídá pohybu v určitém směru.

3.3.3 Funkce Získání Dalšího Stavu

```
def get_next_state(current_x, current_y, action, valid_moves):
if action in valid_moves:
if action == 0:    # Up
current_y = max(current_y - 1, 0)
elif action == 1: # Left
current_x = max(current_x - 1, 0)
elif action == 2: # Down
current_y = min(current_y + 1, 18)
elif action == 3: # Right
current_x = min(current_x + 1, 18)
return current_x, current_y
```

Funkce vypočítá, kam se agent přesune po provedení akce. Bere v úvahu aktuální pozici (X, Y) a akci, a upravuje pozici agenta v rámci definovaných hranic (např. v 10x10 mřížce).

3.3.4 Aktualizace Q-Learning

```
with torch.no_grad():
        Q_table[current_state, action] = (
        Q_table[current_state, action] +
        learning_rate * (reward + discount_factor *
         torch.max(Q_table[next_state]) -
        Q_table[current_state, action])
        )
```

Tento úsek kódu aktualizuje Q-tabulku pomocí algoritmu Q-learning. Hodnoty v Q-tabulce reprezentují odhadovanou odměnu za provedení určitých akcí v určitých stavech. Aktualizace umožňuje agentovi se učit z jeho zkušeností.

ZÁVĚR

V rámci této práce bylo naším hlavním cílem prozkoumat aplikaci Q-learningu, metody posilovaného učení, pro simulaci navigace umělé inteligence v bludišti. Soustředili jsme se přitom na základní principy této metody a její praktickou implementaci v kontextu AI. Praktická demonstrace modelu AI, který řeší bludiště pomocí Q-learningu, odhalila jak obrovský potenciál tohoto přístupu, tak i některé výzvy, které s sebou přináší. Tato práce tak nabízí podrobný pohled na využití posilovaného učení v kontextu řešení konkrétního problému a demonstruje jeho užitečnost v praxi.

LITERATURA

[1] WIKIPEDIA. *Umělá neuronová síť* [online]. San Francisco (CA): Wikimedia Foundation, 2021- [cit. 2023-11-11]. Dostupne z: https://cs.wikipedia.org/wiki/Umela_neuronova_sit

[2] REHOR, Václav. *Umělé neuronové sítě* [online]. Praha: ČVUT, 2021 [cit. 2023-11-11]. Dostupné z: https://users.fit.cvut.cz/~rehorto2/otevrena-fakulta/neural-networks.html

[3] *Typy umělých neuronových sítí* [pdf]. Plzeň: Západočeská univerzita, 2021 [cit. 2023-11-11]. Dostupné z: https://www.kky.zcu.cz/uploads/courses/nses/2_Typy_umelych_neuronovych_siti-faze_jejich_cinnosti-pouziti.pdf

[4] *Neuronové sítě - jednotlivý neuron* [pdf]. Brno: Masarykova univerzita, 2021 [cit. 2023-11-11]. Dostupné z: https://is.muni.cz/www/98951/41610771/43823411/43823458/Analyza_a_hodnoc/44563149/04Neuronove_site_-_jednotlivy_neuronV1_.pdf

[5] *PyTorch* [online]. [cit. 2023-11-11]. Dostupné z: https://pytorch.org

[6] *Popis ANN* [pdf]. Ostrava: VŠB - Technická univerzita Ostrava, 2021 [cit. 2023-11-11]. Dostupné z: http://homel.vsb.cz/~rep75/Predmety/AI/ANN/BackPropagation/Popis%20ANN.pdf

[7] OPENAI. *OpenAI Research and Technology* [online]. San Francisco (CA): OpenAI LLP, 2023- [cit. 2024-01-10]. Dostupné z: https://openai.com/research/

[8] GOOGLE AI. *Google AI* [online]. Mountain View (CA): Google LLC, 2023- [cit. 2024-01-10]. Dostupné z: https://ai.google/

[9] GORDON, R. *AI agents help explain other AI systems*. San Francisco (CA): MIT Schwarzman College of Computing, 2024-01-03. Dostupné z: https://computing.mit.edu

[10] AJAY, A. and AGRWAL, P. *Compositional Foundation Models for Hierarchical Planning*. Massachusetts: Massachusetts Institute of Technology, 2023. Dostupné z: https://arxiv.org

[11] AJAY, A. and AGRWAL, P. *HiP project*. Massachusetts: Computer Science and Artificial Intelligence Laboratory, MIT, 2023. Dostupné z: https://hierarchical-planning-foundation-model.github.io

[12] MIT NEWS. *Multiple AI models help robots execute complex plans more transparently*. Massachusetts: Massachusetts Institute of Technology, 2024-01-03. Dostupné z: https://news.mit.edu