
**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**
**BRNO UNIVERSITY OF TECHNOLOGY**

**FAKULTA INFORMAČNÍCH TECHNOLOGIÍ**
**FACULTY OF INFORMATION TECHNOLOGY**

**ÚSTAV INTELIGENTNÍCH SYSTÉMŮ**
**DEPARTMENT OF INTELLIGENT SYSTEMS**


# HEURISTIKY PRO HRANÍ HRY SCOTLAND YARD

**HEURISTICS FOR THE SCOTLAND YARD BOARD GAME**

**BAKALÁŘSKÁ PRÁCE**
**BACHELOR'S THESIS**


**AUTOR PRÁCE** **MICHAL CEJPEK**
**AUTHOR**

**VEDOUCÍ PRÁCE** **doc. Ing. FRANTIŠEK ZBOŘIL, Ph.D.**
**SUPERVISOR**



**BRNO 2023**

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

# Zadání bakalářské práce

162128

Ústav: Ústav inteligentních systémů (UITS)
Student: **Cejpek Michal**
Program: Informační technologie
Název: **Heuristiky pro hraní hry Scotland Yard**
Kategorie: Umělá inteligence
Akademický rok: 2023/24

Zadání:

1. Seznamte se s pravidly deskové hry typu "Scotland Yard", kdy pozice jedné z figur bývá protihráčům ukázána jen v některých kolech hry. Také nastudujte výsledky, které pro automatické hraní této hry byly dosaženy.
2. Určete metody, které by měly být důvodně vhodné pro realizaci systému, který bude hrát tuto hru autonomně. Zaměřte vedle klasických metod hraní her i metody pro počítačové učení, jako jsou například metody posilovaného učení a hlubokého učení.
3. Pro jednotlivé role figur ve hře implementujte algoritmy řízení a ověřte jejich schopnost plnit zadané cíle.
4. Vyhodnoťte úspěšnost obou stran hry pro různé míry zapojení metod strojového učení a diskutujte zjištěné výsledky.

Literatura:


- Nijssen, J., A., M., Winands, H., M.:"Monte Carlo Tree Search for the Hide-and-Seek Game Scotland Yard", IEEE Transactions on Computational Intelligence and AI in Games 4(4):282 - 294, 2012
- Norvig, P., Russel, S. :"Artificial Intelligence, A Modern Approach", Prentice Hall, 2020
- Daniel Borák: "Heuristic Evaluation in the Scotland Yard Game", Bakalářská práce, 2021, ČVUT


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

Vedoucí práce: **Zbořil František, doc. Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1.11.2023
Termín pro odevzdání: 31.7.2024
Datum schválení: 10.7.2024

# Abstrakt

Tato práce se zabývá možností použití algoritmů hlubokého a posilovaného učení pro řešení problémů s neúplnou informací. Hlavním zkoumaným algoritmem je PPO – Proximal Policy Optimization (optimalizace proximální politiky). K účelu testování vhodnosti algoritmu PPO byla vytvořena zjednodušená implementace hry Scotland Yard a také prostředí pro trénování a testování algoritmů.

Z provedených experimentů této práce vzešlo, že algoritmus PPO je velmi vhodný na řešení problémů s neúplnou informací. Agenti při trénování velmi rychle získali pojem o cílech hry a vybudovali vhodné strategie pro jejich naplnění.



# Abstract

This thesis explores the possibility of using deep and reinforcement learning algorithms to solve problems with incomplete information. The main algorithm studied is PPO – Proximal Policy Optimization. In order to test the suitability of the PPO algorithm, a simplified implementation of the Scotland Yard game was created as well as an environment for training and testing the algorithms.

From performed experiments, it emerged that the PPO algorithm is very suitable for solving problems with incomplete information. The agents very quickly gained a sense of the game's goals and built appropriate strategies to meet those goals through training.



# Klíčová slova

DQN, Hry s neurčitostí, Posilované učení, Proximální optimalizace politiky, Scotland Yard, Umělá inteligence ve hrách

# Keywords

Artificial Intelligence, DQN, Games with uncertainty, Proximal Policy Optimization, Reinforcement Learning, Scotland Yard

# Citace

CEJPEK, Michal. *Heuristiky pro hraní hry Scotland Yard.* Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. Ing. František Zbořil, Ph.D.

# Heuristiky pro hraní hry Scotland Yard

## Prohlášení

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

.....
Michal Cejpek
28. července 2024


## Poděkování

Tímto bych rád poděkoval vedoucímu práce panu doc. Ing. Františku Zbořilovi, Ph.D, za jeho cenné rady, trpělivost a čas při vedení této práce.

# Obsah

# Seznam obrázků

# Kapitola 1

# Úvod

V dnešní době je obor umělé inteligence (AI) stále více všudypřítomný a jeho působení ovlivňuje mnoho aspektů našeho života.

Pokrok umělé inteligence je dobře měřitelný jejím aplikováním v oblasti her. Protože právě hry nabízí jasně definovaná pravidla a cíle, čímž se stávají ideálním testovacím prostorem pro algoritmy umělé inteligence. Výkon AI je ve hrách snadno měřitelný a pokrok dokáže vidět i laik bez žádných složitých grafů, tabulek a výpočtů. Umělá inteligence již dokázala porazit nejlepší hráče v *šachu* [14], *Dota 2* [15], *Go* [4] a spoustě dalších her.

Hra studovaná v této práci se jmenuje *Scotland Yard*. Jedná se o hru pro tři až šest hráčů. V této hře obvykle hraje jeden hráč jako Pan X, který se pokouší uniknout policistům ovládaným ostatními hráči. Policisté však nevědí, kde na herním poli se Pan X nachází. Musí tedy odhadovat jeho pozici a spolupracovat mezi sebou, aby ho mohli polapit. Pozice Pana X je policistům odhalena pouze v určitých kolech. Scotland Yard je ideální hrou ke studování neurčitosti ve hrách hraných systémy umělé inteligence, protože se jedná o hru s nedokonalou informací a k vítězství policistů je zapotřebí spolupráce, strategie a odhadování.

Tato práce zkoumá algoritmy posilovaného učení, jejich použití na hry s neurčitostí a jejich porovnání s klasickými metodami hraní her. Algoritmus optimalizace proximální strategie (Proximal Policy Optimization — PPO) je hlavním zkoumaným algoritmem této práce. PPO je často používán k řešení problémů se spojitými veličinami a ve 3D prostoru. Pro srovnání s jinými algoritmy slouží algoritmus Deep-Q-Learning (DQN). Výslední agenti[1] byli také testováni proti hráči, který náhodně vybírá své akce.

Algoritmus PPO byl vybrán z důvodu jeho úspěšnosti při použití ve složitých hrách s neurčitostí. Toto tvrzení podporuje experiment OpenAI [16], podrobně popsán zde [2]. V tomto experimentu byla optimalizace proximální strategie použita pro učení strategie ve hře typu *schovávaná* s několika agenty a neúplnou informací. Jelikož tento typ hry přesně odpovídá i hře Scotland Yard, zdá se být vhodný i pro řešení této hry. Také další studie [1] a [15] tento výrok dále podporují a dokazují vhodnost algoritmu PPO pro tuto práci.

Zaměření této práce jsem si vybral, protože je mi obor umělé inteligence blízký a vždy jsem se chtěl začít tomuto odvětví více věnovat i po praktické stránce. Algoritmus PPO je pro mě zajímavý a zkušenost s ním by se dala využít v mém dalším pracovním životě.

[1] V kontextu se hrou Scotland Yard je důležité nezaměňovat agenta za policistu. Význam pojmu *agent* je popsáno v kapitole Agent

Pro zpracování práce byly využity tyto hlavní knihovny:

- `Ray.Rlib` — framework pro posilované učení. Obsahuje již zhotovenou implementaci algoritmu PPO a DQN, které byly použity v této práci.
- `PyTorch` — podpůrná knihovna `Ray.Rlib`
- `Gymnasium` — knihovna použita k vytvoření prostředí hry Scotland Yard
- `Pygame` — knihovna pro vytváření uživatelského rozhraní

Práce byla zhotovena pro operační systémy Windows a Python verze 3.10

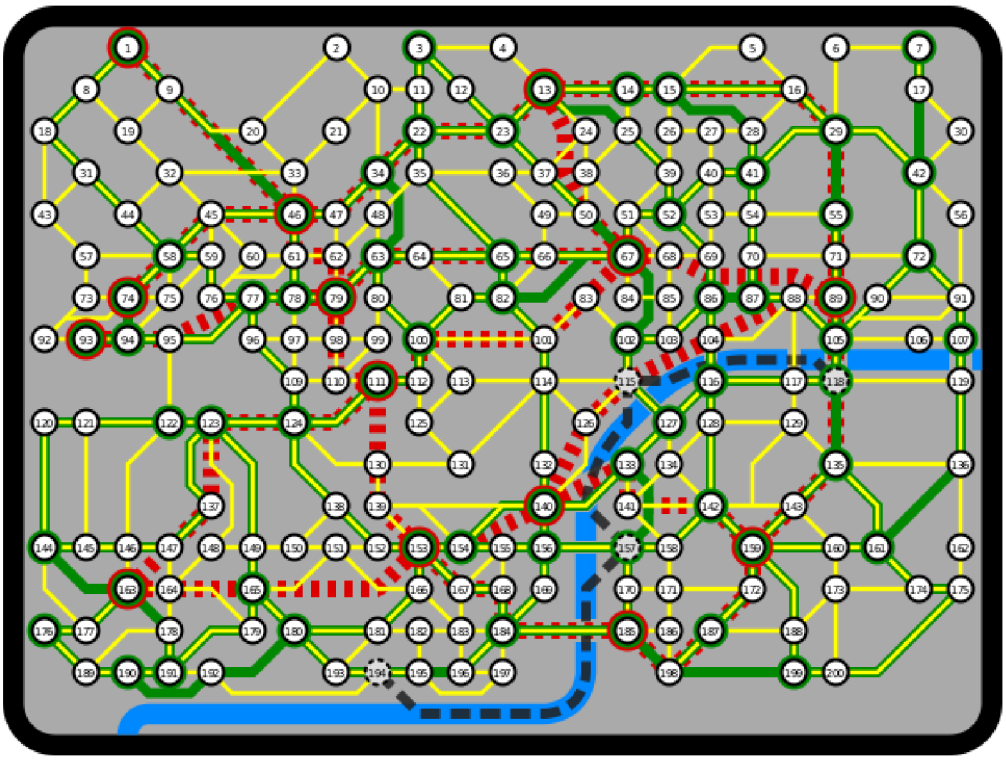
# Kapitola 2

# Teoretický základ

Tato kapitola není encyklopedickým přehledem celého tématu bakalářské práce. Ale jedná se o shrnutí nejdůležitějších relevantních informací a pojmů, důležitých pro tuto práci.

## 2.1 Desková hra Scotland Yard

Scotland Yard je populární hra pro tři a více hráčů, která kombinuje prvky schovávané a hry na honěnou. Jeden hráč hraje za Pana X, který se snaží uniknout policistům, kteří jsou ovládáni zbylými hráči. Hra je u konce, pakliže je Pan X úspěšně chycen policisty (vyhrávají policisté), nebo když je dosaženo maximálního počtu kol, bez toho, aniž by byl Pan X chycen (vyhrává Pan X). Originální hra se odehrává na mapě Londýna.



Obrázek 2.1: Ukázka herní mapy hry Scotland Yard. Zdroj [13]:

Na této herní mapě se nachází 200 polí, která jsou vzájemně propojená cestami. Každá z těchto cest povoluje pouze určitý způsob dopravy (např. pouze taxíkem, pouze autobusem atd.). Jednotliví hráči využívají prvky veřejné dopravy k pohybu po herní ploše, kterými jsou:

- *Taxi*;
- *Autobus*;
- *Metro*;
- *Trajekt*.

Každému hráči je na začátku hry přidělen pouze určitý počet jízdenek na tyto typy dopravních prostředků. K využití dopravy je použita právě tato jízdenka. Pokud již hráč nemá některou z jízdenek, nemůže nadále tento způsob přepravy využívat. Hra se dělí na kola, ve kterých se hráči postupně střídají. Jakmile Pan X provede svůj tah, je odhaleno pouze to, jaký typ dopravy využil, ale neodhaluje, na které místo se přemístil.

Hlavní myšlenkou hry je, že po většinu kol je pozice Pana X policistům utajena. Odhaluje se jim pouze určená kola. To znamená, že policisté pro polapení Pana X musí odhadovat jeho pozici a spolupracovat mezi sebou. Tímto se ze hry Scotland Yard stává hra s nedokonalou informací, jelikož policisté nevidí přesnou pozici Pana X. Tento fakt ji činí vhodnou pro studování metod hraní her s neurčitostí.

Pan X má také možnost použít speciální akci, kterou je *dvojitý tah*, a také vlastní speciální jízdenku, která mu umožňuje použít jakýkoliv druh dopravy. Pokud Pan X použije tuto speciální jízdenku, není policistům odhalen, jaký typ dopravy využil.

## 2.2 Další hry s nedokonalou informací a systémy využívající algoritmus PPO

V oblasti umělé inteligence hraje důležitou roli modelování a řešení her. Hry představují formalizaci konfliktních interakcí mezi aktéry. Klasická teorie her se zaměřuje na hry s úplnou informací, kde mají všechny strany v daném okamžiku přístup ke všem relevantním informacím z herního prostředí. Znají strategie a cíle ostatních hráčů. V praxi se však častěji setkáváme se situacemi, kde jednotlivým stranám chybí některé informace, ať už se jedná o informace z prostředí či cíle soupeře.

Následující hry byly vybrány pro svou složitost a neúplnou informaci, kterou obsahují. V obou případech byly prokázány dobré výsledky s použitím algoritmu PPO. Na základě těchto výsledků i já zvolil algoritmus PPO pro řešení hry Scotland Yard.

### 2.2.1 Dota 2

Dota 2 je velmi komplexní strategická hra odehrávající se v reálném čase. Hraje se ve dvou týmech po pěti hráčích, kteří ovládají postavy s různými unikátními schopnostmi. Schopnosti jsou hlavní součástí hry. Díky schopnostem mohou hrdinové zranit nepřítele, udělat silnější sebe či své spojence a mohou mít i spoustu dalších efektů. Co jim navíc přidává na komplexnosti, je to, že každá schopnost nemůže být po použití znovu použita po určitou dobu.

Ve hře se vyskytuje válečná mlha, která způsobuje, že hráči vidí pouze část mapy ve svém okolí a okolí spřátelených jednotek. To znamená, že nemají úplnou informaci o pozici nepřátelských jednotek.

Dota 2 je pro umělou inteligenci náročnou výzvou z těchto důvodů:

- Složitost hry

  Její stav se mění velmi rychle, a je tedy těžké předpovídat vývoj hry. Agenti tedy musí být schopni rychle reagovat na změny a přizpůsobit jim svoji strategii.

  Pozorovací prostor je obrovský, obsahuje až 16 000 vstupů. Akční prostor je složitý, jelikož lidé hrají Dotu 2 většinou pomocí myši a klávesnice. U OpenAI Five je akční prostor rozdělen na hlavní akce a jejich parametry. Hlavní akce jsou například pohyb, útok, použití schopnosti/ předmětu, zakoupení předmětu a další situační akce. Dostupnost těchto akcí je závislá na stavu hry a je řízena maskou akcí. Tyto akce mají 3 parametry - zpoždění, vybranou jednotku a offset cíle akce. Pokud se tyto akce a jejich parametry zkombinují, výsledkem je 1 837 080 možných akcí.

- Nedokonalá informace

  Právě kvůli válečné mlze nemají hráči úplnou informaci o pozici nepřátelských jednotek, a musí tedy pouze odhadovat, kde na mapě se nachází. Nevědí ani, jestli jsou nepřátelé schopni použít své schopnosti a předměty. Ve hře dota 2 vykonávají hráči několik rozhodnutí někdy i několikrát za sekundu.

Dota 2 sama o sobě obsahuje implementaci chování inteligentních agentů, kteří jsou schopni hrát hru na velmi vysoké úrovni. Jejich výkon však není dostatečný a nebyli schopni ani zdaleka porazit zkušené. Použité metody pro vytvoření těchto agentů bohužel nejsou zveřejněny.

Inovaci umělé inteligence v této hře opět přinesla společnost OpenAI a jejich projekt *OpenAI Five* [15]. Tento projekt se zaměřil na vytvoření modelu, který by byl schopen porazit nejlepší hráče světa ve hře Dota 2. OpenAi zvolila pro učení výsledného modelu algoritmus **PPO**. Agent OpenAI Five byl kombinovaně s předchozí verzí trénován asi *55 000* herních let hraním sama proti sobě metodou self-play.

I přes velikost hry je výsledný reakční čas agenta okolo 33 ms, oproti tomu reakční čas profesionálních hráčů her je okolo 120 ms [12]. OpenAi Five porazil tehdejší nejlepší tým světa během exhibice na nejprestižnějším turnaji Dota2 *The International*. V celém exhibičním turnaji proti nejlepším týmům světa z celkových 24 her vyhrál 19 her a pouhých 5 prohrál.

### 2.2.2 Experiment OpenAI

Důležitým zdrojem pro tuto práci byla studie [2] od společnosti OpenAI. Z ní vyplývá, že algoritmus **PPO** je vhodný pro řešení problémů s nedokonalou informací. V dané studii byl algoritmus implementován na komplexní hru typu schovávaná. Ve hře proti sobě hrají dva typy hráčů. Modří hráči se snaží schovat a utéct před červenými. Po herní ploše byly rozestavěny objekty, se kterými hráči mohou interagovat. Mohou je přesouvat a následně „zamrazit“ na místě, takže s objektem nelze pohybovat. Modří hráči se na začátku hry objevují ve své pevnosti, která má několik děr. Červeným hráčům je na začátku hry znemožněn pohyb, což dává modrým hráčům čas připravit se na jejich útok.

Po více než 8 miliónech epizodách učení se modří hráči naučili efektivně blokovat vstup do své pevnosti, takže je červení hráči nebyli schopni dostihnout. Červení hráči se poté adaptovali a naučili se využívat rampy a přelézt opevnění modrých hráčů. Finální strategii se modří naučili po 43 miliónech epizod. Modří hráči na začátku kola ukradli červeným všechny rampy a zabarikádovali se i s nimi v pevnosti. Červení hráči tak neměli žádnou šanci na výhru.

Dokonce se objevily i fascinující strategie, které zneužívaly chyby v prostředí. Například v kódu obsluhující kolize byla chyba, která způsobovala, že při najetí rampy na zeď arény pod určitým úhlem byla rampa rapidní rychlostí vymrštěna do vzduchu. Toho červení hráči zneužili a za pomoci této chyby se vymršťovali do vzduchu, aby překonali zdi pevnosti modrých.

## 2.3 Klíčové koncepty posilovaného učení

Posilované učení (Reinforcement Learning, RL) je oblast strojového učení, která se zaměřuje na učení agentů v dynamickém prostředí. Agent se učí strategii chování, která maximalizuje kumulativní odměnu.

### 2.3.1 Agent

Agent je komplexní entita, která interaguje s prostředím. Prostředí poskytuje agentovi informace o stavu a agent na základě těchto pozorování a své strategie vykonává akce. Ty mohou ovlivnit stav prostředí a agent obdrží odměnu na základě odměnové funkce.

Na základě těchto odměn se agent učí, které akce jsou vhodné, které naopak ne, a upravuje model podle své strategie. Agent většinou volí takové akce, aby maximalizoval kumulativní odměnu.

Pro textový obsah této práce je velmi důležité definovat pojem *agent*. Slovo je podobné slovu *policista*, a v kontextu hry Scotland Yard je tedy možné tyto významy zaměnit. *Slovo agent v této práci výhradně označuje entitu, která je zde popsána, a nikde nevystupuje jako synonymum ke slovu policista.*

### 2.3.2 Pozorovací a akční prostor

**Pozorovací prostor** je jednou z nejdůležitějších součástí posilovaného učení. Bez jeho vhodného zvolení je dosažení strategie s dobrými výsledky velmi obtížné. Pozorovací prostor je množina všech možných pozorování, která může agent získat z prostředí. Je to tedy forma, kterou prostředí předává informace agentovi. Velmi často se aktuální stav pozorovacího prostoru využívá jako stav. Při vytváření prostředí je nutné definovat typ, tvar prostoru a jakých hodnot může nabývat.

Pro jeho podobu je vhodné zvolit standard z knihovny `Gymnasium` [8]. Gymnasium je pokračování knihovny `Gym` od společnosti OpenAI. Tyto knihovny nastavily standard vytváření prostředí pro posilované učení. Dodržování těchto standardů umožňuje použití libovolného algoritmu na jedno prostředí.

**Akční prostor** je, jak název vypovídá, množina všech možných akcí, ze kterých může strategie volit. Často se jedná o konečnou množinu celočíselných hodnot, které reprezentují různé akce. Samozřejmě může být akční prostor i spojitého typu (blíže popsáno v další sekci).

V herním prostředí mohou nastat situace kdy se nějaká akce stane nevalidní, jelikož by její provedení vyústilo v nevalidní stav herního prostředí. V takovém případě je vhodné zvolení této akce zamezit, nejlépe pomocí akční masky. Ta definuje, které akce jsou v momentálním stavu prostředí proveditelné. Lze také tyto akce povolit, ale následně agentovi udělit sníženou odměnu za provedení nevalidní akce.

### 2.3.3 Diskrétní a spojitý prostor

Je-li prostor **diskrétní**, je konečný nebo spočetně nekonečný.

Příkladem takového prostoru je například pozorovací prostor, jehož vstupem je fotka. Ta se dělí na přesně daný počet pixelů, kde každý má dané hodnoty RGB. Diskrétní akční prostory jsou spočetné a často se jedná o konečnou množinu možných akcí. Například akce pohybu *vlevo*, *vpravo*, *nahoru* a *dolů*.

Zato **spojitý prostor** je nekonečný a nespočetný.

Spojitý pozorovací prostor je například pozice robota ve 3D prostoru, nebo jeho rychlost. Spojitý akční prostor může například ovládat sílu stlačení pedálu nebo úhel natočení kola.

### 2.3.4 Prostředí

Je vše, s čím agent interaguje. Prostředí je buď fyzické (entity z reálného světa, ovládání chytré domácnosti, robotické ruky, ovládání reaktoru apod.), nebo virtuální (simulace nebo hra). Prostředí reaguje na zvolené akce agenta, poskytuje mu zpětnou vazbu ve formě odměny. Odměna může být i záporná, pokud agent zvolil velmi nevhodnou akci. Pokud v prostředí existuje více agentů, může mít každý agent jiné pozorování. Díky tomu můžeme například schovat agentovi $A$ určité informace, které agent $B$ vidí.

### 2.3.5 Strategie (Policy)

Pomocí posilovaného učení vzniká strategie. Strategie je matematická funkce, která definuje agentovo chování na základě jeho pozorování (stavu). Snaží se definovat takové chování, které vede k maximální kumulativní odměně. Strategie může být deterministická, nebo stochastická.

#### Deterministická strategie

Deterministická strategie přesně definuje cílový stav přechodu pro každý stav. Agent tedy pro jeden stav vždy volí stejnou akci. Tato strategie je vhodná, pokud je zapotřebí v každém stavu reagovat konzistentně, bez odchylek, například pokud agent ovládá termostat v domě a teplota je pod požadovanou hladinu. Nemůže se stát, aby byla šance, že agent zvolí akci, která teplotu ještě sníží. Další výhoda je, že je jednoduchá na interpretaci a implementaci [5].

Rovnice výsledné akce deterministické strategie je:

$$\pi(s) = a \tag{2.1}$$

- $\pi$ – strategie
- $s$ – stav
- $a$ – akce
- Rovnice tedy značí akci $a$, kterou agent zvolí ve stavu $s$ podle strategie $\pi$

#### Stochastická strategie

Zato stochastická strategie definuje pro každý stav pravděpodobnostní rozdělení nad akcemi. Výsledná akce je tedy náhodná dle pravděpodobnosti zvolení akce. Může tedy nastat situace, kdy ve stejném stavu agent zvolí vždy jinou akci. Tato strategie je vhodná v situacích, kdy je potřeba zkoumat různé strategie a kdy agent nemá úplnou informaci o prostředí. Například tam, kde by deterministická strategie zvolila jasnou akci $a$, stochastická strategie

by mohla s malou pravděpodobností zvolit akci $b$. Tím může dál prozkoumávat výsledek akce $b$ a může zjistit, že je lepší než akce $a$ [5].

Rovnice výsledné akce stochastické strategie je:

$$\pi(a|s) = \mathbb{P}_\pi[A = a|S = s] \tag{2.2}$$

- $\pi$ – strategie
- $s$ – stav
- $a$ – akce
- $\mathbb{P}_\pi$ – pravděpodobnostní rozdělení
- Rovnice tedy značí pravděpodobnost výběru akce $a$, pokud se prostředí nachází ve stavu $s$, podle strategie $\pi$

### 2.3.6 Akce

Akce je aktivita, proces či funkce, kterou agent vykoná ve specifickém stavu [7]. Výsledkem provedení akce je tedy změna z aktuálního stavu do jiného či stejného stavu z množiny možných stavů. Zjednodušeně je to rozhodnutí, které agent vykonává v prostředí, a toto rozhodnutí ovlivňuje prostředí.

Akce může být také nedeterministická či stochastická. Výsledný stav tedy může být jiný pro stejný stav a akci.

### 2.3.7 Odměna

Odměna je hodnota, kterou agent obdrží od prostředí po vybrání akce. Může být kladná, záporná nebo nulová. Dle této zpětné vazby se agent učí, jak moc byla jeho zvolená akce v daném stavu vhodná.

### 2.3.8 Hodnotová funkce stavu $V(s)$

Hodnotová funkce stavu $V(s)$ vyhodnocuje očekávanou kumulativní odměnu, jestliže se agent nachází v tomto stavu. Vyhodnocuje tedy, jak příznivý je daný stav pro agenta.

### 2.3.9 Hodnotová funkce akce $Q(s, a)$

Hodnotová funkce akce $Q(s, a)$ vyhodnocuje očekávanou kumulativní odměnu, pokud se agent nachází ve stavu $s$ a zvolí akci $a$. Vyhodnocuje tedy, jak dobré či špatné je zvolení dané akce v aktuálním stavu.

### 2.3.10 Rovnováha mezi explorací a exploatací

Explorace a exploatace jsou dvě protichůdné strategie, které se vyskytují jak ve strojovém učení, tak i v reálném životě.

> *Kompromis mezi potřebou získávat nové znalosti a potřebou použít již nabyté znalosti k vylepšení výkonnosti je jedním z nejzákladnějších kompromisů v přírodě [3].*

**Exploatace** (exploitation) se snaží vybrat nejlepší možnou akci na bázi známých informací. Tyto informace nemusí být kompletní nebo mohou být zavádějící. Z důvodu nedostatečného trénování či nedostatečného prozkoumávání možností prostředí.

Opačná metoda **explorace** (exploration) usiluje o prozkoumání možností, které nejsou známé a mohly by vést k lepší budoucí odměně. Explorace tedy často zvolí akci, která nemusí být nejlepší, ale může odhalit nové informace, které následovně povedou ke zlepšení exploatace.

### 2.3.11 Druhy informací v teorii her

V rámci umělé inteligence se potýkáme s různými druhy informací. Dělí se na tyto hlavní typy:

***Dokonalá informace*** znamená, že agent ví o prostředí a o ostatních hráčích vše. Například ve hře šachy hráč vidí všechny figury na herní ploše, tedy i ty soupeřovy.

***Kompletní informace*** značí, že agent je obeznámen se strukturou hry a jsou mu také odhaleny odměnové funkce ostatních hráčů. Hráč tedy ví, jakou hru hraje, je obeznámen s jejími pravidly a rozumí tomu, jaké jsou podmínky výhry, a je obeznámen s taktikou ostatních hráčů.

***Nedokonalá informace*** znamená, že agent nemá všechny relevantní informace o prostředí a ostatních hráčích. Například všechny hry, ve kterých hrají hráči zároveň, jsou hry s nedokonalou informací, jelikož hráč v daném okamžiku nezná informaci o tahu ostatních hráčů. Další příklad je hra poker, kde hráč nezná rozdané karty ostatních hráčů. Patří sem také hra Scotland Yard, kde policisté neznají pozici Pana X.

***Neúplná informace*** znamená, že hráč nezná strukturu odměn, podstatu hry nebo její pravidla. Hráč tedy nezná výchozí informace o hře.

# Kapitola 3

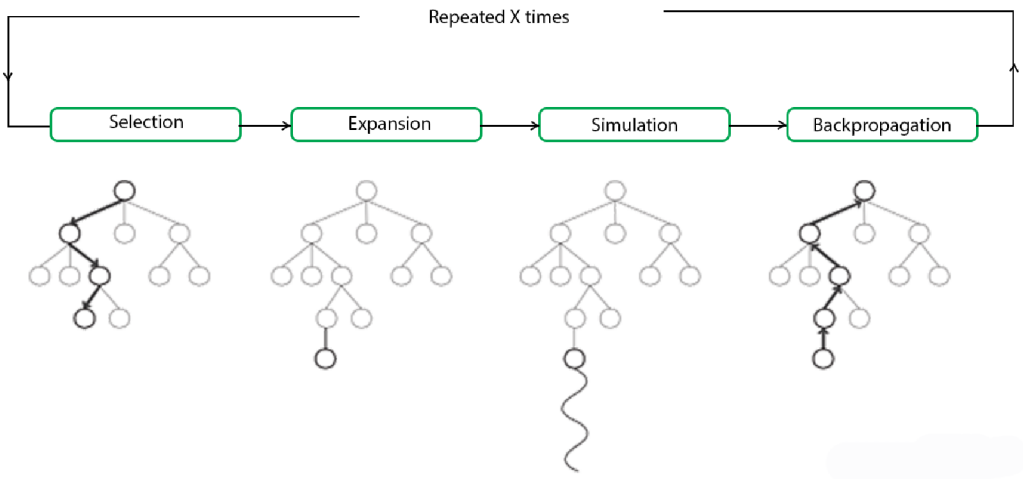
# Algoritmy vhodné ke hraní her s nedokonalou informací

Tato kapitola se zaměřuje na algoritmy, které jsou vhodné pro řešení her s nedokonalou informací a s důrazem na metody posilovaného učení.

## 3.1 Monte Carlo tree search

Metoda Monte Carlo tree search (MCTS) je heuristický algoritmus prohledávání. Kombinuje stromové vyhledávání s principy posilovaného učení. Je často využíván, je-li stavový prostor řešeného problému příliš velký a složitý na to, aby byl prohledán kompletně jinými metodami jako například minimax či alfa-beta prořezávání. Tyto „tradiční“ algoritmy nelze na mnoho problémů použít, jelikož by byly příliš pomalé a náročné na výpočet. Explorací se strom rozrůstá do šířky, zatímco exploatací se strom prohlubuje [17].

MCTS se skládá z několika fází:




Obrázek 3.1: Diagram jednotlivých fází MCTS. Zdroj: [17]

- ***Selekce***

  Na základě aktuálního stavu se vybere další stav k prozkoumání. Pro tento výběr se využívají dvě strategie:

  ***Strom s horní mezí spolehlivosti*** (Upper confidence bounds applied to trees, UCT) kombinuje průměrnou hodnotu uzlu a odměnu za exploraci.

  textbf*Chamtivá strategie* $\epsilon$ ($\epsilon$-greedy strategy) vybírá s pravděpodobností $\epsilon$ náhodný uzel, jinak volí uzel s nejvyšší hodnotou. Tato strategie se používá méně často než UCT.

  Obě tyto strategie se snaží o rovnováhu mezi explorací a exploatací.

- ***Expanze***

  V tomto kroku se vyhledávací strom rozšíří o nový uzel, který je výstupem z předchozího kroku.

- ***Simulace***

  Po této fázi je provedena náhodná simulace od nového uzlu až do konečného stavu.

- ***Aktualizace***

  Díky nově nabytým informacím ze simulace se zpětnou propagací aktualizují hodnoty uzlů ve stromě.

Tento algoritmus se skvěle hodí na hry s nedokonalou či neúplnou informací, jelikož se spoléhá na vzorkování pomocí simulací.

## 3.2 Q-learning

Q-learning je jedním z nejznámějších algoritmů posilovaného učení. Učení tohoto algoritmu probíhá bez modelu (model-free) a mimo strategii (off-policy), což znamená, že se učí přímo z interakce s prostředím a nezávisle na strategii, kterou agent používá k volbě akcí.

Jak už název vypovídá, Q-learning se zaměřuje na určení hodnoty hodnotové funkce akce $Q(s, a)$, viz 2.3.9. Tyto hodnoty se uchovávají v Q–tabulce, která na každou kombinaci stavu a akce uchovává hodnotu $Q(s, a)$. Hodnoty v Q–tabulce se iterativně aktualizují na základě získaných odměn. Pokud agent obdrží kladnou odměnu po vykonání akce $a$ ve stavu $s$, zvýší se hodnota $Q(s, a)$. Naopak dostane-li po vykonání této akce zápornou odměnu, hodnota $Q(s, a)$ se sníží.

Řádky Q-tabulky tedy reprezentují stavy a sloupce akce. Po vytvoření jsou všechny $Q$ hodnoty v tabulce inicializovány na nulu. Následně jsou tyto hodnoty iterativně aktualizovány dle zpětné vazby udělené od prostředí ve formě odměny.

Jako u většiny algoritmů je zde potřeba dohlédnout na rovnováhu mezi explorací a exploatací. K tomu se využívá $\epsilon$–greedy strategie viz 3.1.

**Algoritmus 1** Učení Q-learning s využitím $\epsilon$-greedy strategie

1: V Q tabulce inicializuj všechny hodnoty na 0
2: Inicializace $\epsilon$ na 1
3: **pro** každou epizodu **:**
4: **dokud** není dosaženo konečného stavu **:**
5: S pravděpodobností $\epsilon$ vyber náhodnou akci, jinak vyber akci s nejvyšší hodnotou $Q(s, a)$
6: Proveď akci $a$, pozoruj odměnu $r$ a další stav $s'$
7: Aktualizuj hodnotu $Q(s, a)$:
$Q(s, a) = Q(s, a) + \alpha \left(r + \gamma \max_{a'} Q(s', a') - Q(s, a)\right)$
8: **konec cyklu**

- $\alpha$ – rychlost učení
- $\gamma$ – diskontní faktor
- $Q(s, a)$ – hodnota akce $a$ ve stavu $s$
- $r$ – odměna
- $s$ – stav
- $a$ – akce
- $\max_{a'} Q(s', a')$ – maximální hodnota budoucí odměny, pokud se agent nachází ve stavu $s'$ a zvolí nejlepší dostupnou akci $a'$ (hodnota nejlepší akce v následujícím stavu)

Hlavní nevýhodou tohoto algoritmu je právě jeho závislost na Q–tabulce. Tabulka mapuje hodnoty pro každou kombinaci stavu a akce. Je-li však stavový či akční prostor příliš velký, nebo dokonce nekonečný, je tento algoritmus nevhodný, téměř až nepoužitelný. Výsledná Q tabulka by byla neefektivní kvůli své velikosti. Mohlo by se stát, že s nekonečným množstvím kombinací by velikost tabulky rostla do „nekonečna“. Jako řešení byl navržen algoritmus Deep Q-learning.

## 3.3 Deep Q-learning (DQN)

Tato metoda je rozšířením algoritmu Q-learning, který nahrazuje $Q$–tabulku neuronovou sítí určenou k aproximaci hodnotové funkce akce $Q(s, a)$. Díky této aproximaci je možné použít tento algoritmus na problémy s velkým či nekonečným množstvím kombinací akcí a stavů. Při vytvoření neuronové sítě jsou váhy inicializovány náhodně.

Avšak tímto vzniká nový problém, *nestabilita učení*. Ten je řešen dvěma mechanismy:

- **Přehrání zkušenosti (experience replay)**
  Během trénování se ukládají všechny zkušenosti (přechody) do paměti. Zkušenost zahrnuje stav, akci, odměnu a následující stav. Při aktualizaci vah při trénování se náhodně vybírají skupiny zkušeností z této paměti a aktualizují se podle nich váhy sítě. Tímto se snižuje rozdíl mezi jednotlivými aktualizacemi a tím pádem se zvyšuje stabilita učení.
- **Periodická aktualizace**
  Používají se dvě neurnové sítě, jedna hlavní a jedna cílová. Hlavní síť se aktualizuje při každém kroku. Cílová síť se používá k výpočtu hodnoty $Q(s, a)$ a aktualizuje se

méně často než hlavní síť. Aktualizace cílové sítě se provádí periodicky a kopírováním hlavní sítě. Tímto se změny v chování hlavní sítě nepřenášejí okamžitě do cílové sítě a tím se zvyšuje stabilita učení.

**Algoritmus 2** Učení Deep Q-learning s využitím $\epsilon$-greedy strategie, pamětí zkušeností a periodickou aktualizací cílové sítě

Hlavní neuronové síti $Q$ inicializuj váhy ($\theta$) náhodně
Cílovou neuronovou síť $Q^-$ inicializuj váhy jako kopii hlavní sítě: $\theta^- = \theta$
Inicializace $\epsilon$ na 1
**pro** každou epizodu **:**
    **pro** každý krok epizody **:**
        S pravděpodobností $\epsilon$ vyber náhodnou akci, jinak vyber akci s nejvyšší hodnotou $Q(s, a; \theta)$
        Proveď akci $a$, pozoruj odměnu $r$ a další stav $s'$
        Ulož přechod $(s, a, r, s')$ do paměti $D$
        **pokud** paměť zkušeností $D$ obsahuje dostatek vzorků **:**
            Vzorkuj náhodnou skupinu přechodů $(s_j, a_j, r_j, s'_j)$ z $D$
            **pro** každý vzorkovaný přechod $(s_j, a_j, r_j, s'_j)$ **:**
                **Cílová hodnota** $y_j$**:**

$$y_j = \begin{cases} r_j & \text{pokud je } s'_j \text{ konečný stav} \\ r_j + \gamma \max_{a'} Q^-(s'_j, a'; \theta^-) & \text{jinak} \end{cases}$$

                Vypočítej ztrátu a pomocí gradientního vzestupu (gradient ascend) aktualizuj váhy hlavní sítě $\theta$:

$$L(\theta) = (y_j - Q(s_j, a_j; \theta))^2$$

            **konec cyklu**

        Každých $N$ kroků aktualizuj cílovou síť $\theta^-$ kopírováním hlavní sítě $\theta$
    **konec cyklu**
**konec cyklu**

- $\theta$ — váhy hlavní neuronové sítě $Q$
- $\theta^-$ — váhy cílové neuronové sítě $Q^-$
- $D$ — paměť zkušeností
- $N$ — perioda aktualizace cílové sítě
- $\gamma$ — diskontní faktor
- $\gamma \max_{a'} Q^-(s'_j, a'; \theta^-)$ — hodnota $Q$ funkce pro nejlepší akci $a'$ v následujícím stavu $s'_j$, vypočítaná pomocí cílové sítě $Q^-$
- $Q(s_j, a_j; \theta)$ — hodnota $Q$ funkce pro akci $a_j$ ve stavu $s_j$, vypočítaná hlavní neuronovou sítí $Q$
- $L(\theta) = (y_j - Q(s_j, a_j; \theta))^2$ — rozdíl mezi cílovou hodnotou $y_j$ a hodnotou, kterou vypočítala neuronová síť $Q(s_j, a_j; \theta)$
- Hodnoty vah neuronové sítě se upraví tak, aby se minimalizovala ztráta $L(\theta)$

## 3.4 Metody gradientu strategie

Předchozí metody se snažily naučit či aproximovat hodnotovou funkci akce a vždy volily tu nejlepší, to ale vede k deterministické strategii. Často je ale optimální stochastická strategie. Algoritmy z této skupiny se snaží naučit strategii přímo, a to tím způsobem, že upravují parametry strategie, aby maximalizovaly účelovou funkci (objective function). Nejčastěji se využívá neuronová síť, která má na vstupu stav prostředí, na výstupu pravděpodobnostní rozdělení nad akcemi a váhy neuronů jsou právě parametry strategie. Výsledkem metod této skupiny je tedy pravděpodobnostní rozdělení nad akcemi v daném stavu, což znamená, že každá akce má šanci na zvolení.

Účelová funkce je tedy očekávaná odměna při následování strategie a hodnotí strategii. Základní účelová funkce je definována takto:

$$\mathcal{J}(\theta) = \sum_{s \in \mathcal{S}} d_{\pi_\theta}(s) V_{\pi_\theta}(s) \tag{3.1}$$

- $\mathcal{J}(\theta)$ – Účelová funkce.
- $d_{\pi_\theta}(s)$ – distribuční rozdělení stavů. Značí pravděpodobnost, že se agent nachází ve stavu $s$, při následování strategie $\pi_\theta$.
- $V_{\pi_\theta}(s)$ – hodnota stavu $s$ při následování strategie $\pi_\theta$.
- Účelová funkce je tedy součtem hodnot stavů násobených jejich pravděpodobností výskytu.

Hodnotu stavu $V_{\pi_\theta}(s)$ lze dále rozepsat, vznikne tedy výsledná rovnice účelové funkce:

$$\mathcal{J}(\theta) = \sum_{s \in \mathcal{S}} \Big( d_{\pi_\theta}(s) \sum_{a \in \mathcal{A}} \pi(a|s, \theta) Q_\pi(s, a) \Big) \tag{3.2}$$

- $\pi(a|s, \theta)$ — pravděpodobnost volby akce $a$ ve stavu $s$ podle strategie $\pi_\theta$.
- $Q_\pi(s, a)$ — očekávaná odměna, pokud se ve stavu $s$ zvolí akce $a$.

### 3.4.1 Věta o gradientu strategie a aktualizace parametrů

Aby bylo možné zjistit, jakým způsobem upravit parametry strategie k maximalizaci účelové funkce, je potřeba vypočítat ***gradient***[2] – $\nabla \mathcal{J}(\theta)$.

Základní výpočet tohoto gradientu je závislý jak na pravděpodobnosti výběru akce (která je přímo určena strategií), tak na distribučním rozdělení stavů (která je nepřímo určena strategií).

Efekt změn parametrů strategie na zvolené akce lze jednoduše spočítat, ale odhadnout jejich vliv na distribuční rozdělení stavů je složité.
*Věta o gradientu strategie* [19] výrazně zjednodušuje výpočet gradientu tím, že odstraňuje derivace distribučního rozdělení stavů v závislosti na změněných parametrech strategie:

$$\begin{aligned} \nabla_\theta J(\theta) &= \nabla_\theta \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} Q^\pi(s,a) \pi_\theta(a|s) \\ &= \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} Q^\pi(s,a) \nabla_\theta \pi_\theta(a|s) \end{aligned} \tag{3.3}$$

- $\nabla_\theta \pi_\theta(a|s)$ – gradient pravděpodobnosti volby akce $a$ ve stavu $s$ podle strategie $\pi_\theta$. Lze rozepsat jako: $\frac{\partial \pi_\theta(a|s)}{\partial \theta}$.

Tento teorém je základem pro všechny metody gradientu strategie. Parametry strategie se mohou upravovat například pomocí metody **gradientního vzestupu** (gradient ascend):

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\theta) \tag{3.4}$$

- $\theta_{k+1}$ – nové parametry strategie.
- $\theta_k$ – aktuální parametry strategie.
- $\alpha$ – rychlost učení, standardně se volí malá hodnota okolo 0.01.
- $\nabla_\theta J(\theta)$ – gradient účelové funkce, popsaný výše.

Nejzákladnější metodou gradientu strategie je **REINFORCE**. Metoda je velmi podobná algoritmu Monte Carlo, ale využívá gradientu strategie k naivní aktualizaci parametrů. Vypočtený gradient se využije k aktualizaci parametrů strategie, ale změna není nijak omezována. Změny tak mohou být velmi velké, učení tak může být nestabilní, to řeší následující algoritmy.

[2] Gradient značí směr funkce, značí se symbolem $\nabla_\phi$. Počítá se parciální derivací účelové funkce podle $\phi$.

### 3.4.2 Trust Region Policy Optimization (TRPO)

Trénování strategií pomocí výpočtu gradientu je velmi náchylné na změny, kde velká a náhlá změna v jednom kroku může způsobit zásadní změny v chování agenta a omezit další učení optimálnější strategie. Během učení totiž chceme, aby probíhalo plynule, ne skokově. Pokud se strategie změní příliš rychle, protože následovala nejstrmější směr růstu (gradient), může se stát, že mine cestu vedoucí k optimální strategii a uvízne v lokálním optimu.

TRPO je optimalizační algoritmus, který se snaží tento problém řešit tím, že definuje omezení rozdílu mezi novou aktualizovanou strategií $\pi$ a starou strategií kroku $\pi_{\mathbf{old}}$. Tento rozdíl mezi dvěma pravděpodobnostními rozděleními je definován jako Kullback-Leiblerova divergence (K-L divergence)[21].

Vzniká tak region důvěry (trust region), ve kterém musí nová strategie setrvat. Strategie se tedy nemůže skokově změnit na lokální maximum, ale pouze následuje směr gradientu.

Účelová funkce je definována jako:

$$J^{TRPO}(\theta) = \sum_{s \in \mathcal{S}} d_{\pi_{\theta_{\text{old}}}}(s) \sum_{a \in \mathcal{A}} \pi_\theta(a|s) \hat{A}_{\theta_{\text{old}}}(s,a) \tag{3.5}$$

- $\hat{A}_{\theta_{\text{old}}}(s,a)$ – odhad výhody akce $a$ ve stavu $s$. Měří, jak je daná akce lepší než průměrná akce ve stavu $s$. Lze dále rozepsat jako: $Q_{\pi_{\theta_{\text{old}}}}(s,a) - V_{\pi_{\theta_{\text{old}}}}(s)$,

kde rozdíl mezi novou a starou strategií je omezen K-L divergencí:

$$D_{KL}(\pi_{\theta_{\text{old}}} \| \pi_\theta) \leq \delta \tag{3.6}$$

K výpočtu nových parametrů se již nepoužívá gradientní vzestup, ale používají se Hessovy matice. Hessova matice je vždy čtvercového tvaru. Její hodnoty obsahují druhé parciální derivace funkce a popisují tím její lokální zakřivení. Tím, že se hledá lokální maximum pouze v oblasti důvěry, problém se stabilitou je značně redukován.

Díky tomuto řešení se učení stává robustním a stabilním. Avšak výměnou za to je náročnost výpočtu pomocí Hessových matic a s tím související náročnost na implementaci.

### 3.4.3 Optimalizace proximální strategie (PPO)

Metoda optimalizace proximální strategie (Proximal policy optimization — PPO) byla představena roku 2017 [18]. Tato metoda je vylepšeným následníkem algoritmu TRPO, popsaným v předchozí sekci 3.4.2.

Co výrazně odlišuje tuto metodu od předchůdců, je její jednoduchost a efektivita. Dosahuje lepších výsledků než metoda TRPO a srovnatelných výsledků s metodou ACER. Je však mnohem jednodušší na implementaci, má nižší nároky na výkon a je mnohem lepš, co se týče efektivity dat pro trénování [18]. K dosažení těchto výsledků využívá PPO techniku ořezávání náhradních cílů vloženou do algoritmu TRPO. Vynucuje, aby K-L divergence staré a nové strategie byla v rozmezí $[1-\epsilon, 1+\epsilon]$ kde $\epsilon$ je modifikovatelný parametr.

Nechť $r(\theta)$ je pravděpodobnostní poměr mezi novou a starou strategií.

$$r(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \tag{3.7}$$

Pokud tento vzorec dosadíme do rovnice pro výpočet účelové funkce v algoritmu TRPO, vznikne následovná rovnice:

$$J^{CPI}(\theta) = \sum_{s\in\mathcal{S}} d_{\pi_{\theta_{\text{old}}}}(s) \sum_{a\in\mathcal{A}} \pi_\theta(a|s) r(\theta) \hat{A}_{\theta_{\text{old}}}(s,a) \tag{3.8}$$

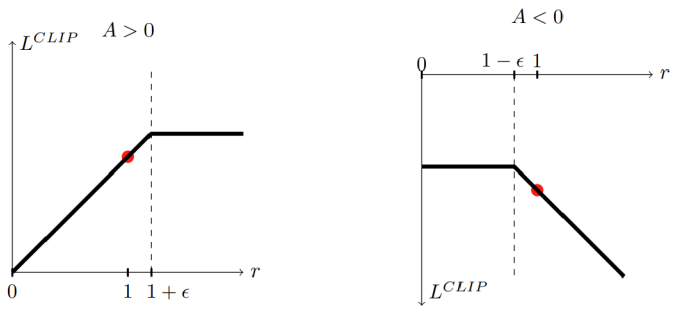
Nyní ale čelíme stejnému problému jako na začátku, kde při gradientním vzestupu může být změna příliš velká. Proto tuto účelovou funkci upravíme a zavedeme ořezávání (clipping).

$$J^{\text{CLIP}}(\theta) = \sum_{s\in\mathcal{S}} d_{\pi_{\theta_{\text{old}}}}(s) \sum_{a\in\mathcal{A}} \pi_{\theta_{\text{old}}}(a|s) \min\left(r(\theta)\hat{A}_{\theta_{\text{old}}}(s,a),\ \text{clip}\left(r(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_{\theta_{\text{old}}}(s,a)\right) \tag{3.9}$$

Jelikož je toto nejdůležitější rovnice z celé prace, znovu zadefinujeme i známé proměnné.

- $\sum_{s\in\mathcal{S}}$ – suma přes všechny stavy
- $d_{\pi_{\theta_{\text{old}}}}(s)$ – distribuční rozdělení stavů. Značí pravděpodobnost, že se agent nachází ve stavu $s$, při následování strategie $\pi_{\theta_{\text{old}}}$.
- $\sum_{a\in\mathcal{A}}$ – suma přes všechny akce
- $\pi_{\theta_{\text{old}}}(a|s)$ – pravděpodobnost volby akce $a$ ve stavu $s$ podle staré strategie $\pi_{\theta_{\text{old}}}$
- $r(\theta)$ – pravděpodobnostní poměr mezi novou a starou strategií
- $\hat{A}_{\theta_{\text{old}}}(s,a)$ – odhad výhody akce $a$ ve stavu $s$,
  vypočtený vzorcem: $Q_{\pi_{\theta_{\text{old}}}}(s,a) - V_{\pi_{\theta_{\text{old}}}}(s)$
- $\epsilon$ – hyperparametr ořezávání
- clip – funkce, která ořízne hodnotu $r(\theta)$ v intervalu $(1-\epsilon, 1+\epsilon)$
- min – funkce, která vybere menší z dvou hodnot

V rovnici se objevuje minimum, to bere menší z ořezané a neořezané hodnoty. Volí minimum, protože jedná pesimisticky (pessimistic bound), a tím volí nejbezpečnější možnost pro aktualizaci parametrů.




Obrázek 3.2: Grafy zobrazují, jak se chová funkce *clip*. Červeným bodem je označen počáteční stav. Levý graf zobrazuje kladný růst gradientu, kde je tedy tento růst omezen. Pravý graf zobrazuje záporné klesání gradientu, kde je toto klesání omezeno. Zdroj: [22]

Všechny výše uvedené vzorce byly kvůli zaměření této práce uvedeny pro diskrétní stavový i akční prostor. PPO však skvěle zvládá i spojité prostory, v případě spojitých prostorů se v rovnicích místo sum přes všechny stavy a akce využije *očekávaná (střední) hodnota*.

Tyto změny tvoří PPO oproti TRPO kompatibilní s metodou gradientního vzestupu, která je oproti výpočtu přes Hessovy matice velice snadná na výpočet. Parametry se tedy aktualizují dle tohoto vzorce

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J^{\text{CLIP}}(\theta) \tag{3.10}$$

PPO má však i své nedostatky. Dle studie [10] se ukázalo, že PPO nefunguje optimálně za 3 podmínek:

1. V prostředí se spojitým prostorem akcí je nemodifikované PPO nestabilní, pokud odměna náhle zmizí mimo ohraničenou podporu.

2. V diskrétním akčním prostoru s řídkými a vysokými odměnami PPO volí neoptimální akce.

3. V době těsně po inicializaci je náchylné k předčasnému volení strategie, pokud je některá z optimálních akcí po inicializaci blízko a snadno dosažitelná.

Tato studie také navrhla tato řešení a prokázala jejich účinnost. Bod 1 a 3 je řešen buď převedením spojitého akčního prostoru na diskrétní nebo zavedením *Beta parametrizace strategie*. *KL regulováním cíle* je řešen bod 2.

Podmínka číslo 1 je v mé implementaci prostředí hry Scotland Yard implicitně neplatná, jelikož akční prostor je diskrétní.
Podmínka číslo 2 je řešena přidáním menších dílčích odměn. Agenti tak získávají odměny i za menší kroky, nejen například za vítězství/ prohru či blízkost, viz podsekce Systém odměn.

**Algoritmus 3** Optimalizace proximální strategie (PPO)

1: Počáteční parametry strategie $\theta_0$, počáteční parametry hodnotové funkce $\phi_0$
2: **pro** iterace $= 0, 1, 2, \ldots$ **:**
3: Shromáždi sadu trajektorií $\mathcal{D}$ spuštěním agentů řízených strategií $\pi_\theta$ v prostředí
4: Vypočítej odměny trajektorií $\hat{R}_t$
5: Vypočítej odhady výhod $\hat{A}_t$ pomocí aktuální hodnotové funkce $V_{\phi_k}$:

$$\hat{A}_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'} - V_{\phi_k}(s_t)$$

6: Aktualizuj parametry strategie $\theta$:

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J^{\text{CLIP}}(\theta)$$

7: **konec cyklu**

- $\sum_{t'=t}^{T}$ – sumace přes všechny kroky trajektorií.
- $\gamma^{t'-t} r_{t'} - V_{\phi_k}(s_t)$ – odhad výhody akce $a_t$ ve stavu $s_t$.
- $k$ – číslo iterace
- $r(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$
  – poměr šance zvolení akce $a_t$ ve stavu $s_t$
  mezi novou a předchozí strategií
- $\epsilon$ – hyperparametr oříznuté funkce.
- $\text{clip}(r(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t$ – ožíznutá ztráta strategie
- $r(\theta)\hat{A}_t$ – ztráta strategie
- min – vybere nejmenší změny, aby nedošlo k nestabilitě učení.
- $\hat{A}_t$ – odhadovaná výhoda akce $a_t$ ve stavu $s_t$
- $\frac{1}{|\mathcal{D}_k|T}$ – počet trajektorií v sadě $\mathcal{D}_k$ a počet kroků $T$ v trajektorii
- $\sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T}$ – sumace přes všechny trajektorie v sadě $\mathcal{D}_k$ a všechny kroky $t$ v trajektorii
- $argmax$ – hledá takový argument, který maximalizuje funkci pomocí optimalizačních metod (Adam, nebo gradientní vzestup)

# Kapitola 4

# Implementace systému pro autonomní hraní hry Scotland Yard

V této kapitole je popsáno navržené řešení a výsledná implementace systému.

## 4.1 Zkoumaná modifikovaná verze hry Scotland Yard

Tato práce využívá modifikovanou verzi hry Scotland Yard, ve které se hráči pohybují po mřížkové herní ploše ve tvaru čtverce. Na mřížce se nachází 15×15 polí. Hráči se po těchto polích mohou pohybovat ortogonálně i diagonálně, vždy však o maximálně 1 pole. Hráč se může rozhodnout nezměnit pozici a zůstat na svém aktuálním poli. K pohybu nejsou potřebné žádné jízdenky.

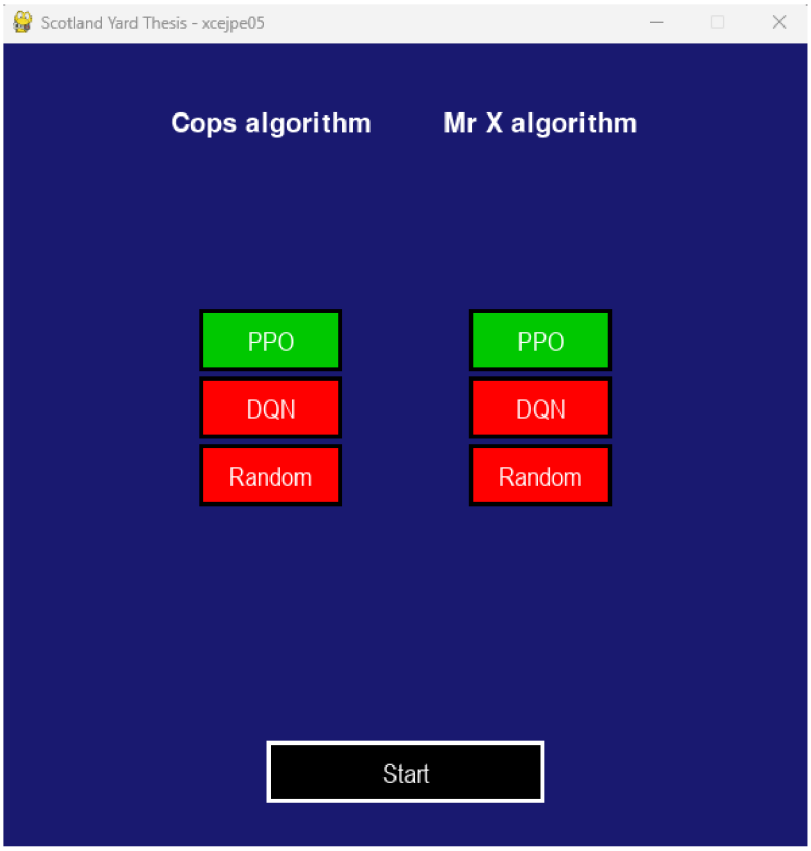
Tato úprava vede k tomu, že se zjednodušil stavový i akční prostor, jelikož se hráči nemusí starat o svoje jízdenky a mohou se pohybovat po herní ploše libovolně. Změny avšak nemění základní podstatu hry, zachovává neurčitost, ale značně zjednodušuje implementaci. Díky tomuto zjednodušení klade trénování nižší nároky na výkon.

Než začne hra, náhodně se vygenerují možné pozice, na kterých mohou Pan X a policisté začínat. Z nich se následně náhodná pozice přidělí jednotlivým hráčům. Poté začíná hra.

Hra se dělí na jednotlivá kola, ve kterých se hráči ve svých tazích střídají. Pro hru byli zvoleni 3 policisté, protože je herní pole velké a 2 policisté by nemuseli mít možnost ho celé pokrýt. V kole hraje jako první Pan X a poté policisté již podle svého očíslování, které jim bylo náhodně přiděleno při vytvoření. Hra končí v okamžiku, kdy policisté chytí Pana X, nebo když Pan X zůstane nepolapen až do konce.

## 4.2 Implementace uživatelského rozhraní a herních mechanismů

Uživatelské rozhraní bylo vytvořeno pomocí knihovny `Pygame`. Hra začíná v menu, kde je momentálně možné vybrat pouze možnost sledování hry mezi dvěma agenty. Uživatel ale může vybrat, jaký algoritmus rozhodování je použit pro jednotlivé agenty. K dispozici jsou algoritmy *PPO*, *DQN* a náhodné chování.




Obrázek 4.1: Menu hry

Samotný kód hry je rozdělen na 3 části. Jednotlivé vrstvy hry jsou tak izolovány a mohou být snadno vyměněny za jiné verze.

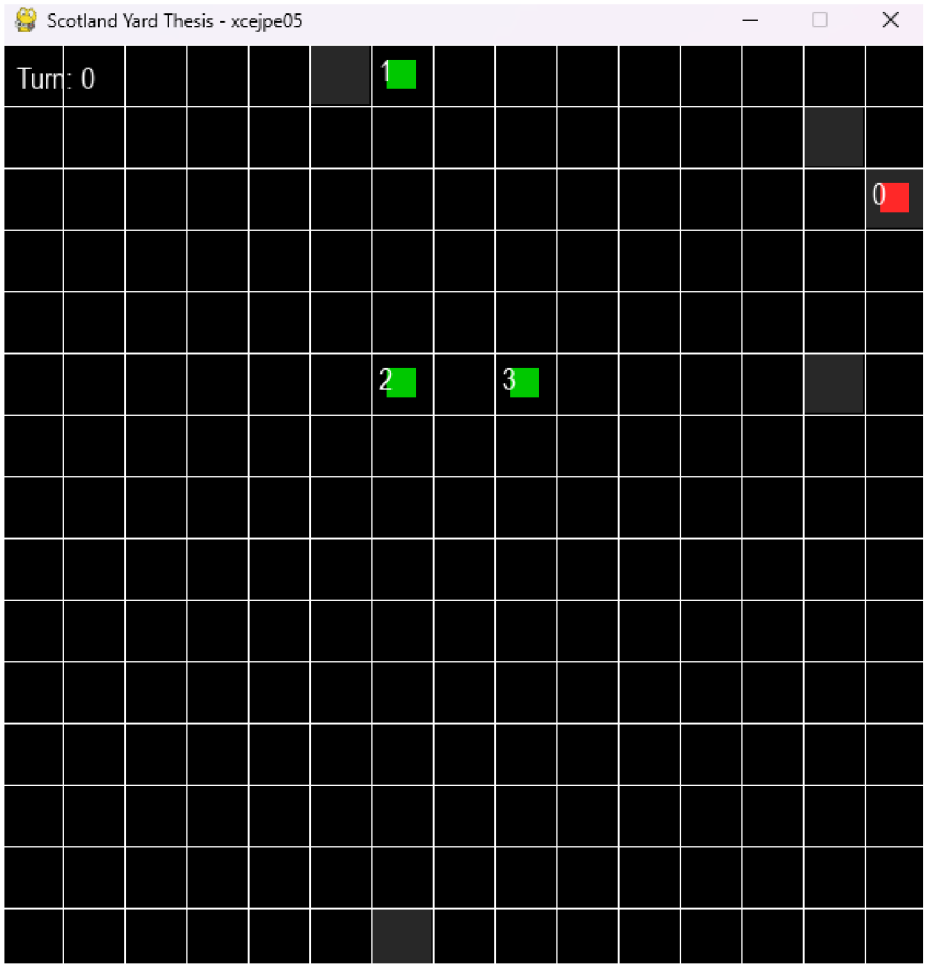
- *GameController* [6]
  Tato třída je zodpovědná za řízení hry. Je zde spuštěna univerzální herní smyčka, která zpracovává uživatelské vstupy, následně provádí aktualizaci stavu aktuální scény a její překreslení.
- *Scény* [6]
  Jednotlivé scény následně definují své chování při aktualizaci a překreslení. Manipulace a přepínání mezi nimi je zajištěno pomocí zásobníků scén. Do něj se ukládají nově otevřené scény a obsluhována je vždy ta nejnovější. Díky tomu, když se vypne aktuální scéna, je aktivována další scéna ze zásobníku.
- *Hra*
  Samotná hra je následovně rozdělena na 2 další části, přičemž každá zpracovává jinou část hry.
  - *Herní logika* – src/game/scotland_yard_game_logic.py
    Zpracovává herní mechanismy, jako je pohyb, zpracování výherních podmínek atd. Zprostředkovává informace o prostředí pro učící se agenty. Funkcionalita hlavní funkce *step* a jak se volí akce, je popsáno v dalších kapitolách (4.3.3 a 4.4).

– *Herní vizualizace* – src/game/scotland_yard_game_visual.py
Vykreslování herních elementů (herní pole, figury atd.).

*Použití herní smyčky, která obsluhuje aktuální scénu, bylo inspirováno výukovým projektem z platformy GitHub [6]. Kód byl ale značně upraven a vylepšen, aby vyhovoval integrování do této práce.*

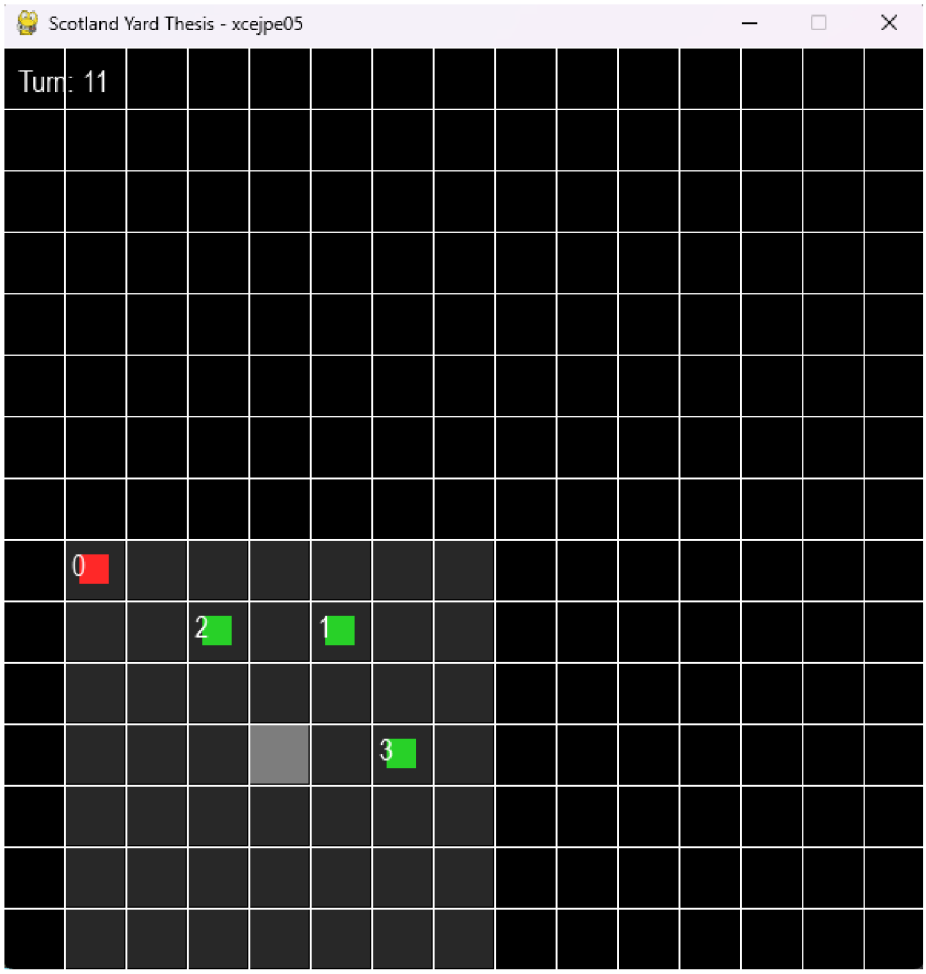
Hra se zapne v pozastaveném stavu, aby bylo možné lépe pozorovat počáteční rozložení hráčů. Pokud je hra pozastavena, je možné pokračovat stisknutím klávesy Space (mezerník). Jakmile hra běží, je možné ji opětovně pozastavit stisknutím klávesy Space (mezerník).

Na herní ploše jsou zobrazeny 4 figury ve formě barevných čtverců. Červený čtverec značí Pana X a zelené čtverce značí policisty. V levém horním rohu okna je zobrazen aktuální stav hry. Pakliže hra stále běží, je zobrazeno číslo aktuálního kola. Jakmile hra skončí, je zobrazeno, kdo hru vyhrál.




Obrázek 4.2: Hra před spuštěním

V obrázku 4.2 je hra v pozastaveném stavu a je zobrazeno počáteční rozložení hráčů. Na obrázku lze vidět tmavě šedé čtverce, které značí možnou pozici Pana X. Na začátku hry je těchto pozic mnoho, jelikož Pan X doposud svoji pozici neodhalil.




Obrázek 4.3: Jedenácté kolo hry

Na obrázku 4.5 je zobrazeno jedenácté kolo hry. Světle šedý a nejvýraznější čtverec značí poslední známou pozici Pana X. Tmavě šedé čtverce opět značí možné pozice Pana X. Na tomto obrázku lze pozorovat, že policisté jsou v blízkosti Pana X a snaží se ho chytit. Všichni policisté se z pozic zobrazených na obrázku 4.2 přesunuli na možné pozice Pana X.

## 4.3 Prostředí a učení agentů

V této kapitole je popsáno prostředí, ve kterém agenti vystupují, jaké informace z něj pozorují a jaké akce mohou provádět. Dále je popsáno, jakým způsobem byli agenti trénováni, jakou formou probíhá a co je výsledkem tohoto trénování.

### 4.3.1 Použitá knihova pro učení

K učení agentů pro hru Scotland Yard byl využit open-source framework `Ray` [20, 11]. Konkrétně byla využita knihovna `Ray.Rllib`. Tato knihovna poskytuje nástroje pro posilované učení a samotné implementace jednotlivých algoritmů, které byly využity v této práci. Rllib dokáže využívat obě populární knihovny pro strojové učení `Tensorflow` a `Pytorch`. Pro tuto práci byla vybrána knihovna `Pytorch`.

Knihovna `Ray` byla zvolena pro podporu učení více modelů současně, je vhodná pro velké projekty a umožňuje možnosti konfigurace jako žádný jiný framework. Obsahuje spoustu implementací algoritmů pro posilované učení, a jelikož je má výsledná implementace modulární, lze jednoduše zaměnit využívaný algoritmus za jiný.

Cenou za tuto flexibilitu, modularitu a abstrakci je ale složitost konfigurace, složitost vytvoření správného prostředí a mnohem větší časová náročnost celého systému. Bez výkonné grafické karty je trénování velmi pomalé.

### 4.3.2 Pozorovací a akční prostor agentů

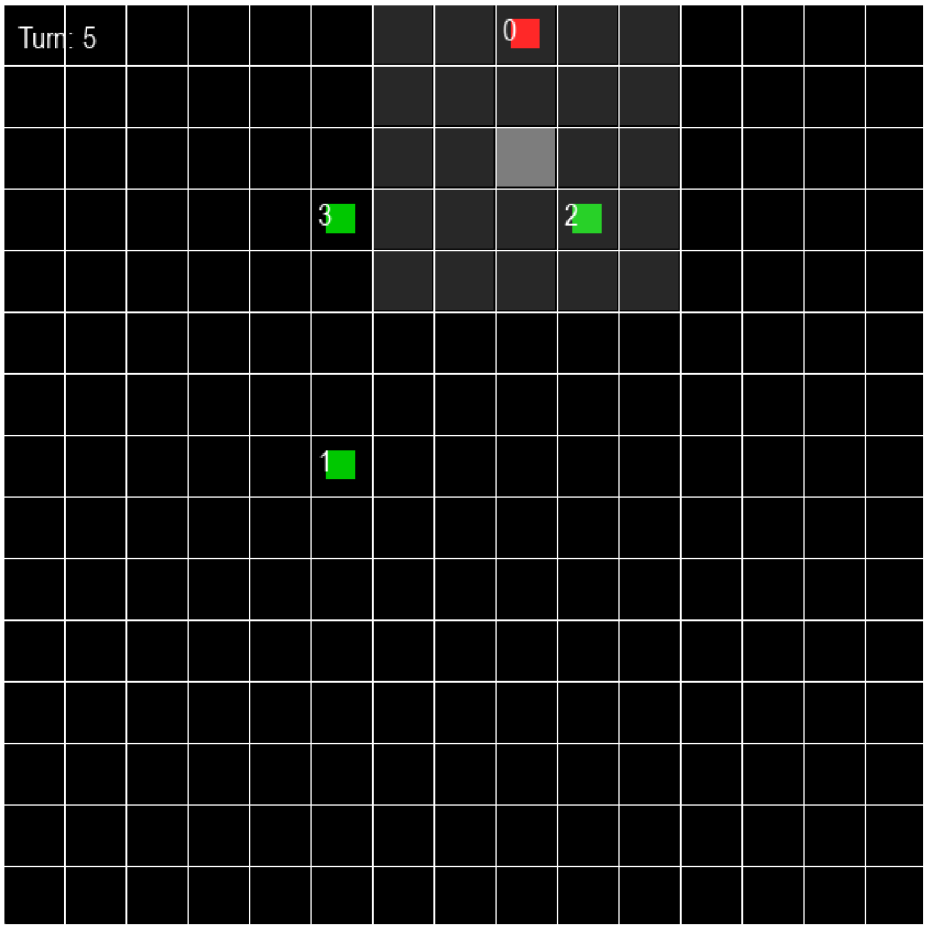
V této práci v prostředí vystupují dva agenti s různými strategiemi a různými pozorovacími prostory. Jeden model definuje chování policistů a druhý model ovládá chování Pana X. Akční prostor je stejný jak pro policisty, tak pro Pana X.

**Pozorovací prostor** agentů je velmi důležitý pro úspěšné naučení dobré strategie. Struktura jednotlivých pozorovacích prostorů je následující:

- **Shodné položky pozorování**
  - Číslo aktuálního kola;
  - Maximální počet kol;
  - Číslo kola, kdy dojde k dalšímu odhalení pozice Pana X;
  - Počet kol, kterých agent stál na místě;
  - Vlastní pozice agenta (x, y);
  - Pozice (x, y) a vzdálenost od poslední známé pozice pana X;
  - Pozice (x, y) zbylých policistů a vzdálenosti od nich.
- **Pozorování, která mají policisté navíc**
  - Pozice (x, y) nejbližšího bodu oblasti zájmu;
  - Vzdálenost od nejbližšího bodu oblasti zájmu;
  - Pravdivostní hodnota, zda je policista v oblasti zájmu.

Agenti tedy nevidí všechna pole, ale jen ta, která jsou důležitá. Klíčový je pozorovací prostor policistů. Ti vidí pouze poslední známou pozici Pana X a *oblast zájmu*, což je oblast, ve které se může Pan X nacházet. Tato oblast je blíže popsána v podkapitole 4.3.4, kde je podrobněji vysvětlena její souvislost se systémem odměn.

Pokud informace obsažená v těchto pozorovacích prostorech není v daný moment definována, je nahrazena hodnotou $-1$. Konkrétně se jedná pouze o hodnoty týkající se poslední veřejně známé pozice Pana X, jelikož v prvních několika kolech není známa.




Obrázek 4.4: Stav hry, pro ukázku vstupů do neuronové sítě

Na obrázku 4.4 je zobrazen stav hry. V tomto stavu byly jednotlivé hodnoty pozorování agentů následující:

$$\text{pan\_x} = \begin{bmatrix} 5, & 24, & 8, & 0, & 8, & 0, & 8, & 2, & 2, & 5, \\ 7, & 7.6, & 9, & 3, & 3.2, & 5, & 4, & 5.7 & & \end{bmatrix}$$

$$\text{policista\_1} = \begin{bmatrix} 5, & 24, & 8, & 0, & 5, & 7, & 8, & 2, & 5.8, & 9, \\ 3, & 5.7, & 5, & 4, & 3, & 6, & 4, & 3.2, & 0 & \end{bmatrix}$$

$$\text{policista\_2} = \begin{bmatrix} 5, & 24, & 8, & 0, & 9, & 3, & 8, & 2, & 1.4, & 5, \\ 7, & 5.7, & 5, & 4, & 5.1, & 9, & 3, & 0, & 1 & \end{bmatrix}$$

$$\text{policista\_3} = \begin{bmatrix} 5, & 24, & 8, & 0, & 5, & 4, & 8, & 2, & 4.5, & 5, \\ 7, & 3, & 9, & 3, & 5.1, & 6, & 4, & 2, & 0 & \end{bmatrix}$$

Například u policisty 1 značí hodnoty následovné informace:

- 5 – aktuální kolo;
- 24 – maximální počet kol;
- 8 – kolo, kdy dojde k dalšímu odhalení pozice Pana X;
- 0 – počet kol, kterých policista stál na místě;
- 5 – x-ová souřadnice pozice policisty 1;
- 7 – y-ová souřadnice pozice policisty 1;
- 8 – x-ová souřadnice poslední známé pozice Pana X;
- 2 – y-ová souřadnice poslední známé pozice Pana X;
- 5.8 – vzdálenost od poslední známé pozice Pana X;
- 9 – x-ová souřadnice pozice policisty 2;
- 3 – y-ová souřadnice pozice policisty 2;
- 5.7 – vzdálenost od policisty 2;
- 5 – x-ová souřadnice pozice policisty 3;
- 4 – y-ová souřadnice pozice policisty 3;
- 3 – vzdálenost od policisty 3;
- 6 – x-ová souřadnice nejbližšího bodu oblasti zájmu;
- 4 – y-ová souřadnice nejbližšího bodu oblasti zájmu;
- 3.2 – vzdálenost od nejbližšího bodu oblasti zájmu;
- 0 – pravdivostní hodnota, zda je policista v oblasti zájmu.

**Akční prostor** agentů reprezentuje možné akce, které mohou agenti provést. Tyto akce jsou stejné jak pro policisty, tak pro Pana X. Agenti se mohou hýbat ortogonálně i diagonálně, a také můžou zůstat na místě. Celkově tedy agent může volit z 9 akcí.

### 4.3.3 Trénování agentů pomocí algoritmu DQN a PPO

Trénování je proces, při kterém agenti zlepšují své schopnosti a strategie na základě zpětné vazby z prostředí. V případě této práce dochází k modifikaci vah neuronů ve skrytých vrstvách neuronových sítí.

Během trénování je současně modifikován jak model pro policisty, tak model pro Pana X. Jedná se tedy o posilované učení s více agenty **(Marl)**. V tomto podoboru posilovaného učení koexistuje vícero agentů ve stejném prostředí. Mohou mít stejné či rozdílné cíle.

V případě této práce tedy 3 agenti reprezentují policisty se stejným cílem. Jeden agent reprezentuje Pana X, který má na rozdíl od zbylých agentů opačný cíl. I proto jsem zvolil framework Ray, který jako jeden z mála tyto možnosti podporuje.

#### Neuronové sítě ve výsledných modelech

V modelech PPO i DQN hraje hlavní roli neuronová síť. Vstupem do těchto neuronových sítí je aktuální stav, v případě mé práce je to viditelné pozorování agentů v prostředí, popsané v předchozí části. Tyto vstupy jsou následně zpracovány skrytými vrstvami v neuronových

sítích. Struktura vnitřních vrstev a výstup neuronové sítě modelů jsou rozdílné pro oba algoritmy.

**Neuronová síť v modelu trénovaném algoritmem PPO**

- Vstup
  - Viditelný stav prostředí.
- Struktura
  - Jedná se o dopředně propojenou neuronovou síť, kde se informace šíří pouze jedním směrem.
  - Vstupní vrstva má 18 vstupů pro Pana X a 19 pro policisty.
  - Obsahuje obsahuje dvě skryté vrstvy po 256 neuronech [256, 256]. Aktivační funkce je ReLU, která napomáhá k učení komplexních závislostí.
  - 9 možných výstupů (1 pro každou akci).
- Výstup
  - Výstupem této neuronové sítě je pravděpodobnostní rozdělení nad všemi akcemi, což znamená, že výstupem je pravděpodobnost zvolení pro každou akci, viz kapitola o PPO 3.4.3.
  - Jedná se tedy o stochastickou strategii, kde každá akce má určitou šanci na zvolení, přičemž některé akce mají větší šanci a jiné menší.

**Neuronová síť v modelu trénovaném algoritmem DQN**

- Vstup
  - Viditelný stav prostředí
- Struktura
  - Jedná se o dopředně propojenou neuronovou síť, kde se informace šíří pouze jedním směrem
  - Vstupní vrstva má 18 vstupů pro Pana X a 19 pro policisty
  - Dvě skryté vrstvy, kde první má 128 neuronů a druhá 64 [128, 64]. Aktivační funkce je ReLU, která napomáhá k učení komplexních závislostí.
  - 9 možných výstupů (1 pro každou akci)
- Výstup
  - Výstupem této neuronové sítě je ohodnocení všech akcí v aktuálním stavu (hodnotová funkce Q), viz kapitola o DQN 3.3.
  - Používám $\epsilon$-greedy variantu, kde agent tyto hodnoty porovná a vybere akci s největší Q hodnotou, nebo náhodnou akci. Šanci na zvolení náhodné akce určuje právě $\epsilon$. Během trénování je $\epsilon$ postupně snižováno od 100% šance na náhodnou akci k 5% šanci na náhodnou akci. Vysoké epsilon vede k exploraci a nízké k exploataci. Ve hře je poté $\epsilon$ nastaveno na 0

Využívám tedy implementace algoritmu PPO i DQN, které jsou obsaženy v knihovně `Ray.Rlib`. PPO využívá základní parametry trénování bez žádné modifikace parametrů,

jelikož se ukázalo, že agenti se chovají optimálně a hru pochopili i bez zásahů do výchozích hodnot hyperparametrů. To, že nebylo třeba modifikovat parametry, je i díky tomu, že PPO těchto hyperparametrů ani moc nemá, jeho konfigurace je tedy velmi jednoduchá oproti jiným algoritmům.

Zato DQN agenti se chovali *neoptimálně* i přes ladění hyperparametrů. S konfiguračními parametry `Ray.rlibu` a hyperparametry DQN jsem dlouho experimentoval. Měnila se velikost vnitřních vrstev, počet neuronů, rychlost učení, velikost paměti, velikost dávky učení a další. Agenti se i přesto naučili pouze stát na místě, přestože toto chování bylo velmi penalizováno. Pro vyřešení tohoto problému je nyní agentům udělována penalizace, pokud se 5 kroků po sobě nepohnou. Penalizace je rovna zápornému počtu kol nečinnosti. Postupně tedy roste a pohybem je resetována. Tato expanze odměn vedla k vylepšení strategie agentů DQN a nadále nestáli pouze na jednom místě. Bohužel však přes veškeré snahy agenti DQN stále nedosahují ani zdaleka tak dobrých výsledků jako agenti PPO.

Trénování modelů je obsaženo v samostatných skriptech, spustitelných z příkazové řádky:

- TrainerDQN.py
  Spuštění: `python TrainerDQN.py`

  --backup-folder [string]: složka pro záložní kopie; default: None

  --load-folder [string]: načte model z dané složky; default: trained_models_dqn

  --save-folder [string]: uloží model do dané složky; default: trained_models_dqn

  --num-iterations [int]: počet iterací trénování; default: 50

  --save-interval [int]: interval ukládání modelu

  --no-verbose: zákaz výpisu průběhu trénování; default: False

- TrainerPPO.py
- Spuštění: `python TrainerPPO.py`
  - Argumenty jsou stejné jako u TrainerDQN.py

Tyto spustitelné skripty obsahují třídy a konfigurace pro trénování modelů. Při spuštění skriptu se načte aktuální model, pokud existuje, a pokračuje v trénování tohoto modelu. Pokud model neexistuje, je vytvořen nový model a začíná se s trénováním od začátku. Model je uložen každých 5 iterací trénování. Při dlouhém trénování doporučuji spustit script s parametrem `--do-backup`, který zároveň periodicky tvoří záložní kopie modelů. Během několikadenního trénování může dojít k výpadku proudu či jiné chybě a model může být poškozen. Během běhu skriptu se vypisuje aktuální stav trénování (pokud je nastaveno `-verbose`):

- Číslo aktuální iterace;
- Celkový čas trénování;
- Počet epizod v iteraci a průměrná odměna v aktuální iteraci;
- U algoritmu DQN se vypisuje aktuální hodnota epsilon.

### Řešení zvolení nevalidních akcí

Hra Scotland Yard má omezené herní pole. Tím pádem je jasné, že pokud se hráči vyskytují na okraji herního pole, nemohou zvolit takovou akci, která by je posunula mimo

toto herní pole. Policisté také nemohou zvolit akci, která by je posunula na stejné pole, ve kterém již stojí jiný policista. K vyřešení tohoto problému se nejčastěji využívá tzv. maska akcí (*Action mask*). Framework Ray tuto možnost v omezené míře podporuje. Vytvořené prostředí je ale potřeba obalit v obalové třídě, která tuto funkcionalitu zprostředkuje. Bohužel se mi nepodařilo tuto funkcionalitu spojit s dalšími požadavky systému, které také vyžadují zabalení do jiné třídy. Mezi těmito požadavky je fungování více aktérů s různými strategiemi v jednom prostředí a různé pozorovací prostory agentů.

Agent tedy může při trénování zvolit nevalidní akci, ale je za jejich zvolení velmi penalizován. Agenti ji tedy volí opravdu zřídka, a to jen na základě velmi malé explorační šance a pouze na počátku trénování. Pokud se i tak stane, je ve hře implementovaná funkcionalita, která se pokusí znovu vygenerovat další akce, dokud vygenerovaná akce není validní. Aby se zamezilo nekonečnému cyklu čekání na validní akci, je po 100 pokusech vygenerována náhodná validní akce. Takto uměle generovaná akce nebyla za celou dobu testování agentů PPO potřeba, avšak ze statistického hlediska může tato situace nastat.

Zavedení tohoto omezení pomocí systému odměn vedlo k positivním výsledkům a agenti se naučili, jaké akce jsou nevalidní, a nepoužívají je.

### 4.3.4 Systém odměn

Jednotlivým agentům jsou udělovány odměny na základě jejich chování. Jelikož policisté a Pan X mají odlišné a protichůdné strategie, i jejich odměny jsou odlišné. Odměny, které agenti dostávají, byly navrženy odhadem a laděny postupným experimentováním.

Stejné odměny, které dostávají oba agenti:

- Odměna za přímou výhru: 50 (nepřímá výhra: 30)
  - Nepřímá výhra je pro policisty, kteří se přímo nepodíleli na chycení Pana X.
- Prohra: $-50$
- Nevalidní akce: $-20$
- Neaktivita: $-0.5 \times$ počet kol nečinnosti $<-2.5, -12>$
  - Neaktivita začíná po 5 kolech stání na místě.
  - Tato odměna byla přidána pro podporu učení algoritmu DQN.

Odměna pro Pana X navíc obsahuje:

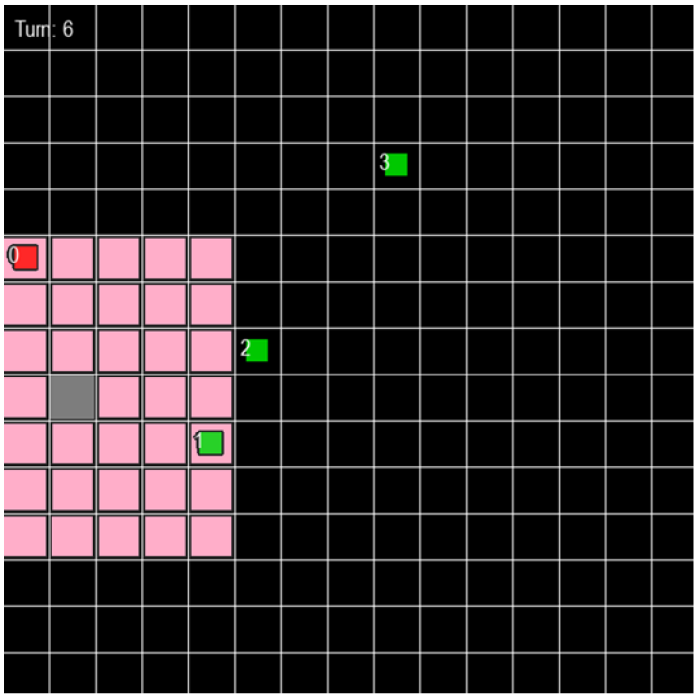
- Vzdálenost od nejbližšího policisty: $<-20, 20>$
  - Odměna v závislosti na nejbližším policistovi se ukázala jako lepší než odměna za vzdálenost od všech policistů. Pan X v opačném případě tolik neřešil, když byl blízko jednoho policisty, jelikož odměna byla zprůměrována ze vzdáleností všech policistů.

Odměna pro policisty navíc obsahuje:

- Vzdálenost od oblasti zájmu: $<-8, 10>$ (Pokud je policista v oblasti zájmu, dostává konstantní odměnu 12). Oblast zájmu je blíže popsána v následující podkapitole.

### Oblast zájmu

Policisté znají s jistotou pouze poslední odhalenou pozici Pana X. Okolo tohoto bodu je vytvořena oblast zájmů, ve které se Pan X může nacházet. Funguje zde analogie s reálným světem, kde při hledání osoby známe její poslední známou pozici a její maximální rychlost a můžeme tak vytvořit oblast, která stojí za prohledání. Policisté vidí bod této oblasti, který je jim nejbližší, tedy vstupní bod. Čím blíže je policista k tomuto bodu, tím větší odměnu získává. Pokud je od oblasti velmi vzdálený, obdrží penalizaci. Jakmile je policista v oblasti zájmu, dostává konstantní odměnu za to, že se v této oblasti pohybuje.




Obrázek 4.5: Zvýrazněná oblast zájmu policistů (růžově zvýrazněné pole).

Oblast zájmu je vypočtena na základě poslední známé pozice Pana X. Na této pozici je vytvořen čtverec, jehož velikost je závislá na počtu kol, které uplynuly od posledního odhalení Pana X. Pokud byl Pan X odhalen v aktuálním kole, je velikost čtverce rovna nule a oblast zájmu je tedy pouze 1 pole. Další kolo se velikost čtverce zvětší o jedno pole na každou stranu, obsahuje tedy 9 polí.

PPO algoritmus může v prostředích s diskrétními akcemi volit velmi neoptimální akce, pokud má příliš málo podstatných odměn[10]. Účel této odměny je podpořit učení policistů a zamezit tomuto problému. Policisté tak získávají více dílčích odměn, ne pouze jednu velkou odměnu za chycení Pana X. Přidáním této dílčí odměny za interakci s touto oblastí se výsledná strategie značně zlepšila a agenti dosahovali lepších výsledků.

Další variantou k předání této oblasti policistům, bylo přidat do pozorování policistů pozice všech polí, kde se může Pan X nacházet. Toto by však bylo méně efektivní, jelikož by se pozorovací prostor výrazně zvětšil, což by zvýšilo náročnost výpočtů a zpomalilo by se učení. Naštěstí agenti dosahovali dobrých výsledků již se zjednodušenou variantou.

## 4.4 Systém řízení pro hru Scotland Yard

Systém řízení je zodpovědný za řízení chodu agentů a celé hry Scotland Yard stejně jako za trénování agentů. Systém řízení tedy můžeme rozdělit na dvě různé části podle toho, zda kontroluje trénování, nebo o hraní hry. V obou variantách je hlavní smyčka, která volá hlavní metodu **step**. Tato metoda následně volá i metodu **play_turn**, která řídí chod hry.

### 4.4.1 Systém řízení během trénování

**Algoritmus 4** *step()* metoda prostředí, která se využívá při trénování

**vstup:** hrající agent, akce pro agenta
**výstup:** pozorování pro dalšího agenta, odměna

**pokud** akce je nevalidní **:**
    další agent = agent
    odměna = −20
    **return** stejný stav, odměna

game.play_turn(akce)
nový stav = environment.get_state()
odměna = environment.get_reward()
další agent = environment.get_next_player()
**return** nový stav, odměna

**Algoritmus 5** Průběh trénování ve frameworku Ray

**vstup:** počet iterací, velikost dávky, prostředí, model
**výstup:** model

dávky = []
**pro** i in počet iterací **:**
    **pro** pracovníka in pracovníci **:**
        velikost_dávky = 0
        mini_dávka = []
        **dokud** velikost_dávky < max_velikost_dávky **:**
            stav = prostředí.získej_stav()
            akce = model.získej_akci(stav)
            nový stav, odměna = prostředí.step(akce)
            mini_dávka.append(stav, akce, nový stav, odměna)
            velikost_dávky += 1
        dávky.append(mini_dávka)
    **konec cyklu**
    model.update(dávky)
**konec cyklu**
**return** model

Za řízení trénování agentů je zodpovědný framework `Ray`. Vícero procesů je spuštěno současně a každý proces má svoji instanci prostředí. Každý proces sbírá data z hraní agentů a následně je posílá do hlavního procesu, který je zodpovědný za zpracování těchto údajů a trénování. Prostředí má svoji funkci step, která je volána v každém kroku sbírání dat.

Této metodě je předána zvolená akce dle strategie tohoto učení a metoda zkontroluje, zda je akce validní. Pokud je validní, prostředí ji provede. V opačném případě je agent penalizován za špatnou akci a funkce step je volána znovu pro stejného agenta, dokud není zvolena validní akce. Po provedení akce tedy metoda vypočte nový stav prostředí a odměny za provedenou akci. Nové pozorování z prostředí předá dalším agentům, kteří mají hrát. Vždy je tedy této metodě *step* předána akce pro agenty, kteří obdrželi pozorování. Tím se naznačuje, kdo hraje jako další a kdo má čekat na svůj tah. Tyto dávky odehraných her jsou poté předány hlavnímu procesu, který na základě použitéhu algoritmu pro učení upraví váhy své neuronové sítě, viz kapitola o algoritmu PPO (3.4.3) a DQN (3.3).

### 4.4.2 Systém řízení během hraní hry

**Algoritmus 6** Základní herní smyčka

```
scény = fronta()
scény.přidej_scénu(Úvodní_scéna)

dokud je fronta scén neprázdná :
    zpracuj_uživatelský_vstup()
    pokud uživatel stiskl klávesu esc :
        scény.odeber_scénu()

    scéna = scény.první_scéna()
    scéna.aktualizuj_scénu(uživatelský_vstup)
    scéna.zobraz_scénu()
```

Hlavní herní smyčka obsahuje frontu scén. Scény jsou třídy, které obsahují implementace metod pro aktualizaci a zobrazení scény. V této frontě jsou postupně přidávány scény, které se mají zobrazit. Jakmile je scéna odstraněna, intuitivně se zobrazí minulá scéna.

**Algoritmus 7** Aktualizace scény hry Scotland Yard

```
pokud hra není pozastavena :
    game.play_turn()
    pokud hra skončila :
        Zobraz konec hry a za 2 sekundy resetuj hru
```

**Algoritmus 8** Metoda *play_turn()*

```
vstup: hrající agent, akce pro agenta

pokud není akce :
    pocet_pokusu = 0
    akce = None
    dokud je akce nevalidní :
        pokud pocet_nevalidnich_akci < max_pocet_nevalidnich_akci :
            akce = model.získej_akci(stav, agent)
            pocet_nevalidnich_akci += 1
        jinak:
            akce = náhodná_validní_akce

    return akce

agent.move(akce)
agent = get_next_player()
```

Tato metoda obsluhuje jedno kolo hry. Vstupem je agent, který má hrát, a akce, kterou má provést. Pokud není akce, znamená to, že neprobíhá trénování a akci vygeneruje model.

Zde je metoda Step volána v každém kroku herní smyčky. Tato metoda již nedostává akci agentů jako vstup při trénování, ale musí si ji sama vygenerovat. Metoda step tedy volá metodu pro získání akce od hrajících agentů.

Agentům je předán aktuální stav a ti ho pošlou na vstup neuronové sítě svého modelu a na základě výsledků vrátí vybranou akci. Metoda následně zkontroluje, zda je akce validní, a pokud není, vygeneruje novou. Samozřejmě se může teoreticky stát, že agenti budou volit nevalidní akci vždy a program by se mohl zacyklit. Řešení tohoto problému bylo popsáno v kapitole Řešení zvolení nevalidních akcí (4.3.3).

Hlavní metoda step nyní získala akci pro agenty a následně tuto akci provede. Na základě jejího provedení se upraví stav prostředí a funkce step je volána znovu, dokud nenastane konec hry.

### 4.4.3 Pořadí tahů v jednom kole

V průběhu realizace práce jsem experimentoval s různými variantami průběhu kola hry. V **prvotní verzi, kde agenti hrají zároveň** se hra dělila pouze na kola a funkce step obsluhovala celé toto kolo. Ve funkci step tedy byly volány metody pro získání akce pro všechny agenty, které se nakonec vykonaly najednou.

V této variantě ještě agenti nemohli zůstat na aktuální pozici. Pan X si tak vybudoval zajímavou strategii. Místo aby se od policistů držel co nejdále, přibližoval se k nim a využíval toho, že všichni hrají současně. Vždy volil akci, která ho posunula na pole, kde byl policista. Věděl totiž, že na daném místě již další kolo policista být nemůže, protože nemohl zůstat na stejném poli.

Tento styl ale neodpovídal tomu jak, se skutečně hra hraje, protože všichni hráči nemohou hrát současně. Proto jsem navrhl **finální verzi, kde se agenti střídají**. Zde již byla implementována možnost stání na místě a agenti se střídají v tahu. Hra je tedy členěna na kola a ta jsou dále členěna na tahy jednotlivých hráčů. Tato verze již více odpovídá reálné hře.

# Kapitola 5

# Experimenty

Pro ověření vlastností algoritmu PPO byly provedeny dva experimenty. Experimenty byly prováděny na systému s následujícími parametry:

- Operační systém: Windows 10 (64-bit)
- Procesor: Intel Core i5–9300H
- Paměť: 16 GB DDR4
- Grafická karta: NVIDIA GeForce GTX 1660 Ti

Agenti DQN i PPO se učili ve stejném prostředí, za stejných počátečních podmínek, měli totožný stavový i akční prostor a od prostředí získávaly stejné odměny.

Pro uskutečnění těchto experimentů byly vytvořeny skripty v jazyce Python. Jeden pro sběr dat a druhý pro jejich následné vyhodnocení a zpracování do grafů a textové podoby.

## 5.1 Experiment 1: Pozorování vývoje chování během tréninku

Experiment trénování spočívá v ukázce, jak se agenti postupem trénování zlepšují. U agentů DQN a PPO je očekáván nárůst průměrné odměny a snížení průměrné vzdálenosti policistů od Pana X. Experiment v průběhu tréninku periodicky provádí simulace her mezi agenty a zaznamenává jejich průběh.

Hlavními sledovanými parametry jsou:

- Průměrná odměna Pana X;
- Průměrná odměna policistů;
- Průměrná vzdálenost policistů od Pana X;
- Počet vyhraných her Pana X;
- Počet vyhraných her policistů.

Experiment sleduje vývoj těchto hodnot po **1000 iterací**. To znamená, že na konci experimentu podstoupili všichni agenti přesně 1000 iterací tréninku

Simulační hry byly prováděny ve všech možných kombinacích algoritmů. Zkoušeny tedy byly všechny algoritmy jak v roli Pana X, tak v roli policistů. Simulace byly spouštěny po různém počtu trénovacích iterací v závislosti na aktuálním počtu již provedených iterací.

Počet simulací byl následovný:

- 50 simulací každých 10 iterací, do 100 trénovacích iterací;
- 50 simulací každých 20 iterací, od 100 do 500 trénovacích iterací;
- 50 simulací každých 50 iterací, od 500 do 1000 trénovacích iterací.

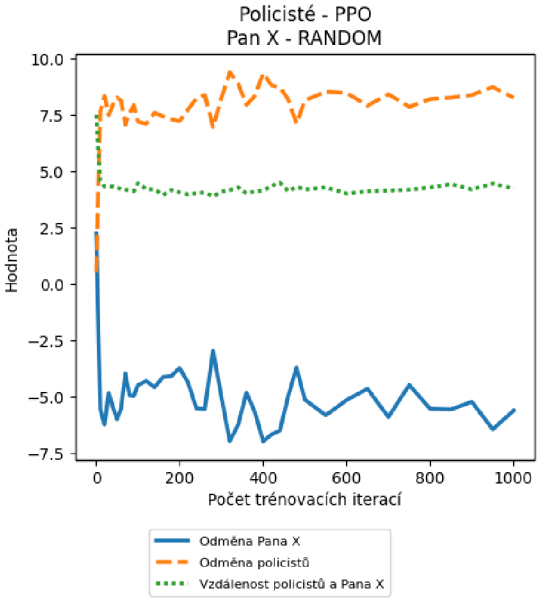
Při učení je nejzajímavější sledovat vývoj chování agentů na začátku trénování. Proto byly simulace prováděny s větší četností právě na začátku a postupem trénování byla četnost snižována pro snížení výpočetní a časové zátěže simulací. Tím bylo zajištěno, že experiment správně a efektivně vyobrazuje, jak rychle a jestli vůbec byli agenti schopni pochopit cíl hry. Sběr dat pro tento experiment trval přibližně 18 hodin. Při četnějším sběru dat by rapidně narostla časová náročnost experimentu a dle testovacích spuštění a výpočtů by trval i několik dní.

### 5.1.1 Výsledky experimentu

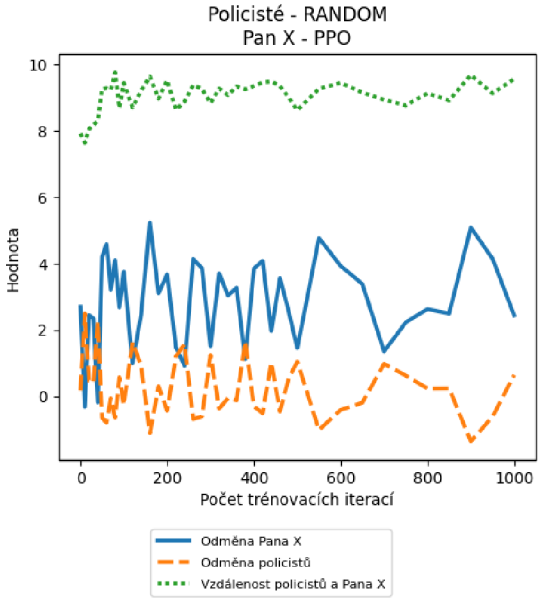
Při sledování výsledků tohoto experimentu je důležité si uvědomit, že díky základní premise hry Scotland Yard je pro policisty vyhrát mnohem složitější než pro Pana X, protože policisté dokážou vyhrát pouze tehdy, pokud chytí Pana X, zato Pan X může vyhrát jenom na základě uplynutí maximálního počtu kol.

Všechny následující grafy vyobrazují průměrnou odměnu Pana X (modrá nepřerušovaná linie) a policistů (oranžová přerušovaná linie), průměrnou vzdálenost policistů od Pana X (zelená tečkovaná linie), a to vše v závislosti na počtu trénovacích iterací.

#### Srovnání s náhodným agentem




Obrázek 5.1: Graf simulace her mezi policisty trénovanými pomocí PPO a Panem X volícím náhodné akce

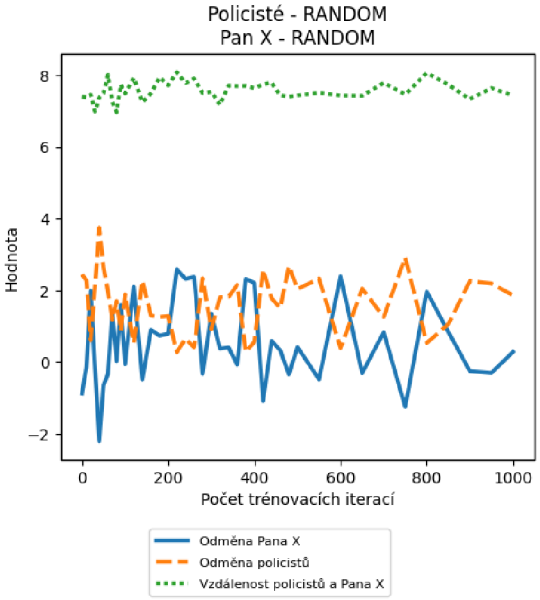



Obrázek 5.2: Graf simulace her mezi Panem X trénovaným pomocí PPO a policisty volícími náhodné akce

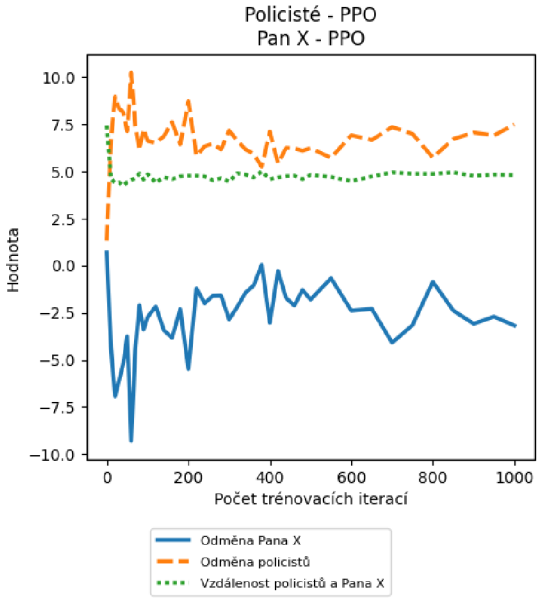
Jak lze vidět na grafu 5.1, policisté jsou velmi úspěšní ve své strategii. Vzdálenost mezi policisty trénovanými pomocí PPO a Panem X, který volí náhodné akce, je po celou dobu velmi malá. Odměna, kterou policisté získávají, je velmi vysoká. Odměna Pana X je naopak velmi nízká.

Na grafu lze také sledovat, že průměrná vzdálenost policistů od Pana X na začátku trénování rapidně klesá, stejně tak klesá odměna Pana X tím, že se zlepšuje strategie policistů. Po tomto rapidním poklesu se vzdálenost policistů od Pana X stabilizuje a odměna Pana X se drží na nízké hodnotě.

Naopak z grafu 5.2 lze pozorovat, že je odměna Pana X kladná a až na odchylky vyšší než odměna policistů, kteří se chovají náhodně. Odměna policistů se pohybuje kolem nuly a odměna Pana X je stále kladná. Také vzdálenost policistů od Pana X je skoro dvojnásobná než v předchozím případě.




Obrázek 5.3: Graf simulace her mezi policisty a Panem X, kde oba volí náhodné akce




Obrázek 5.4: Graf simulace her, kde oba agenti volí akce dle modelu PPO

Obrázek 5.5: Srovnání PPO agentů s agenty volící náhodné akce

Na srovnání 5.5 lze vidět, že jak PPO, tak náhodný algoritmus udržují velice stabilní vzdálenost. U náhodného chování je to kvůli základní pravděpodobnosti, která musí být průměrně stejná. U grafu agentů PPO to vypovídá o tom, že policisté dokážou stabilně odhadovat pozici Pana X a přinejmenším se pohybovat v jeho blízkém okolí. Z grafu 5.4 lze pozorovat, že odměna policistů a Pana X kolísá. Frekvence tohoto kolísání se postupně jemně zmenšuje. Z toho odvozuji, že toto kolísání vypovídá o tom, že se agenti postupně učí strategii protivníka a snaží se podle ní upravit svoji strategii, aby byli opět lepší. To značí, že se obě strategie pomalu přibližují k optimu. Předpokládám, že dalším trénováním by se tento trend potvrdil. Avšak k tomu by byl zapotřebí větší výpočetní výkon a spousta času. Ale jak lze vidět z výsledků experimentu, i přes relativně malý počet trénovacích iterací se objevuje očekávané chování a rozdíl s náhodným chováním je markantní.

### Závěr experimentu pro algoritmus PPO

Každá možná kombinace využití algoritmů pro policisty a Pana X byla během experimentu testována celkově 2050×. Z tohoto testování vzešly kromě grafů vývoje chování také údaje zaznamenávající výsledky jednotlivých her. Tyto výsledky byly zpracovány do tabulek, které zobrazují procentuální počet výher jednotlivých stran v závislosti na použitém algoritmu.

| Policisté | Pan X | | |
|---|---|---|---|
| | PPO | Random | DQN |
| PPO | 38 % | 67 % | 65 % |
| Random | 11 % | 20 % | 18 % |
| DQN | 10 % | 17 % | 19 % |

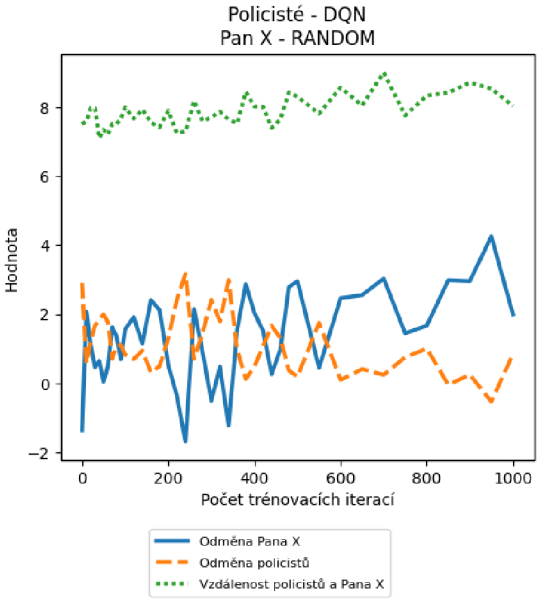
Tabulka 5.1: Zobrazuje procentuální počet výher policistů proti Pánovi X s vybraným algoritmem během experimentu

| Pan X | Policisté | | |
|---|---|---|---|
| | PPO | Random | DQN |
| PPO | 62 % | 89 % | 90 % |
| Random | 33 % | 80 % | 83 % |
| DQN | 35 % | 82 % | 81 % |

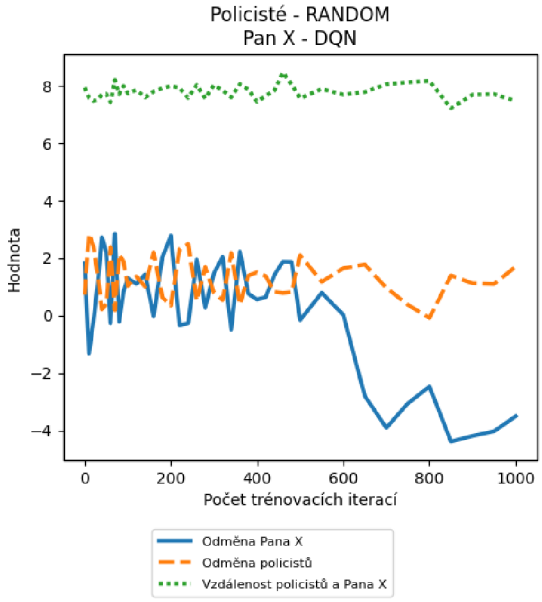
Tabulka 5.2: Zobrazuje procentuální počet výher Pana X proti policistům s vybraným algoritmem během experimentu

Pokud zde opět porovnáme výsledky, kdy proti sobě hráli dva náhodní agenti a agenti trénovaní pomocí PPO, lze vidět, že agenti trénovaní pomocí PPO mají výrazně lepší výsledky.

### Ostatní výsledky




Obrázek 5.6: Graf simulace hry mezi policisty trénovanými pomocí DQN a Panem X volícím náhodné akce




Obrázek 5.7: Graf simulace hry mezi Panem X trénovaným pomocí DQN a policisty volícími náhodné akce

Jak již jsem zmiňoval, výsledky trénování agentů pomocí algoritmu DQN bohužel nemají dobré výsledky. Výsledné chování se velmi podobá náhodnému agentovi. Z předchozích prací je známo, že by algoritmus DQN měl mít znatelně lepší výsledky než náhodná metoda [9].

Algoritmus DQN je velice citlivý na hyperparametry a je možné, že bylo zvoleno nevhodné nastavení. Potřebuje také mnohem více trénovacích iterací než PPO. Je tedy možné, že 1000 iterací není dostatečné pro naučení dobré strategie pro algoritmus DQN. Jelikož jsou ale trénovací iterace PPO časově náročnější než iterace DQN, nebylo možné sladit jejich počet.

*Trénování z tohoto experimentu poslouží jako základ pro další experiment, kde budou simulovány hry s již natrénovanými agenty.*

## 5.2 Experiment 2: Simulace hry s již natrénovanými agenty

Tento experiment byl proveden za účelem ověření, zda jsou agenti trénovaní pomocí algoritmu PPO schopni pochopit cíl hry a naučit se optimální strategii. Porovnává, jestli se natrénované modely z experimentu 1 výrazně liší od agenta, který volí pouze náhodné akce, a tím prokázat, že se chová způsobem, který vede k vítězství.

Pro ověření skutečného výkonu agentů byly provedeny simulace hry mezi již natrénovanými agenty. Ačkoli již předchozí experiment dokazuje, že agenti trénovaní pomocí algoritmu PPO jsou schopni pochopit cíl hry, je důležité získat skutečná data o výkonu jednotlivých metod bez zkreslených výsledků kvůli trénování.

Následující tabulky byly získány z 10000 simulací pro každou kombinaci již natrénovaných algoritmů.

| Policisté | Pan X | | |
|---|---|---|---|
| | PPO | Random | DQN |
| PPO | 26 % | 72 % | 64 % |
| Random | 10 % | 19 % | 18 % |
| DQN | 8 % | 17 % | 17 % |

Tabulka 5.3: Zobrazuje procentuální počet výher policistů proti Pánovi X s vybraným algoritmem

| Pan X | Policisté | | |
|---|---|---|---|
| | PPO | Random | DQN |
| PPO | 74 % | 90 % | 92 % |
| Random | 28 % | 81 % | 82 % |
| DQN | 36 % | 82 % | 83 % |

Tabulka 5.4: Zobrazuje procentuální počet výher Pana X proti policistům s vybraným algoritmem

Z tabulky lze vyčíst, že policisté trénovaní pomocí PPO vyhráli proti Pánovi X s náhodným chováním 72 % her. V porovnání s policisty volícími náhodné akce je to výrazně lepší výsledek. Ti proti Pánovi X s náhodným chováním vyhráli pouze 19 % her.

Z her vzešly také tyto informace:

- Průměrná vzdálenost mezi Panem X a policisty (oba PPO): 4.779692656488397;
- Průměrná vzdálenost mezi Panem X a policisty (oba náhodné chování): 7.631191103120987;
- Průměrná vzdálenost mezi Panem X a policisty (oba DQN): 7.869671633109524.

Z tohoto experimentu lze tedy vyvodit, že agenti trénovaní pomocí algoritmu PPO byli schopni se naučit optimální strategii i ve hře s neúplnou informací.

# Kapitola 6

# Závěr

Provedené experimenty ukázaly, že algoritmus PPO je skutečně vhodný na hry s neúplnou informací, jelikož policisté trénování pomocí PPO dokázali odhadovat přibližnou pozici Pana X a měli mnohem větší úspěšnost než náhodní agenti. Agenti dokázali pochopit koncept a cíl hry Scotland Yard. Naučili se v ní chovat optimálně, přestože začínali s nulovými znalostmi o prostředí. Bylo příjemným překvapením, že již po pouhých 10 iteracích trénování byli agenti schopni ukazovat lepší výsledky než agent s náhodným chováním.

## Možná vylepšení

Bohužel se během experimentu nepovedlo separovat výslednou strategii od frameworku `Ray.Rlib` a extrahovat ji do podoby, kde by již `Rlib` nebyl potřeba pro její využití. Při spouštění hry se tedy načítá i celý framework `Ray.Rlib`, toto velmi zpomaluje načítání hry, kdy prvotní načtení trvá několik sekund. Toto a spoustu dalších potíží jsou důvody, proč zvolení knihovny Ray.Rllib zpětně lituji. Měl jsem spíše algoritmus PPO implementovat svépomocí s využitím `Pytorch` či `Tensorflow`, nebo využít knihovnu `CleanRl` a učení více strategií naráz vyřešit jiným způsobem.

Původním plánem bylo také realizovat možnost hraní hry mezi lidmi a agenty. Tuto implementaci jsem ale přes problémy s experimenty a algoritmem DQN nestihl dokončit. Mám za to, že by tato možnost nadále prohloubila výsledky této studie o zajímavé experimenty, kde by bylo možné sledovat, jak si agent PPO vede proti lidskému protivníkovi.

Agenti trénovaní pomocí algoritmu DQN měli velmi špatné výsledky. Je vyloučeno, že by byla chyba v prostředí, udělování odměn či v pozorování, jelikož se agenti trénovaní pomocí PPO učili za stejných podmínek a jejich výsledky byly dobré. Je možné, že i přes hledání optimálních parametrů s pomocí `Ray.Tune` byla zvolena nevhodná konfigurace. Byť jsem s těmito parametry sálodlouze experimentoval, nebylo dosaženo uspokojivých výsledků. Navrhuji změnu stavového a akčního prostoru, aby byl kompatibilní s verzemi studentů z minulých let, kteří realizovali bakalářskou práci na podobné téma, a následně využít jejich implementaci pro další experimenty [9].

Dále je zde možnost rozšířit hratelnost hry o jednotlivé druhy dopravy a k nim příslušící jízdenky. Tím by výrazně vzrostla složitost hry a stavový prostor. Jak ale ukázala práce OpenAI [15], algoritmus PPO nemá problém s obrovským stavovým i akčním prostorem a naučit se i komplikované hry.

# Literatura


[1] Baes, J. *Application of Reinforcement Learning Algorithms to the Card Game Manille*. 2022. Diplomová práce. UGent. Faculteit Ingenieurswetenschappen en Architectuur.

[2] Baker, B.; Kanitscheider, I.; Markov, T.; Wu, Y.; Powell, G. et al. *Emergent Tool Use From Multi-Agent Autocurricula*. 2020. Dostupné z: https://arxiv.org/abs/1909.07528.

[3] Berger Tal, O.; Nathan, J.; Meron, E. a Saltz, D. The Exploration-Exploitation Dilemma: A Multidisciplinary Framework. *PLOS ONE*. 1. vyd. Public Library of Science, Duben 2014, sv. 9, č. 4, s. 1–8. Dostupné z: https://doi.org/10.1371/journal.pone.0095693.

[4] Borowiec, S. *AlphaGo seals 4-1 victory over Go grandmaster Lee Sedol* online. 2019. Dostupné z: https://www.theguardian.com/technology/2016/mar/15/googles-alphago-seals-4-1-victory-over-grandmaster-lee-sedol. [cit. 2024-03-18].

[5] Carr, T. *Policies in Reinforcement Learning* online. March 2024. Dostupné z: https://www.baeldung.com/cs/rl-deterministic-vs-stochastic-policies. [cit. 2024-03-20].

[6] CDcodes), C. Y. channel:. *Game states* online. June 2021. Dostupné z: https://github.com/ChristianD37/YoutubeTutorials/tree/master. [cit. 2024-03-20].

[7] Cognilytica. *Action - Cognilytica* online. Unknown. Dostupné z: https://www.cognilytica.com/glossary/action/. [cit. 2024-05-02].

[8] Foundation, F. *Gymnasium*. 2024. Dostupné z: https://gymnasium.farama.org/.

[9] Hrkľová, Z. *Metody hlubokého učení pro strojové hraní hry Scotland Yard [online]*. Brno, CZ, 2023 [cit. 2024-05-04]. Bakalárska práca. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://theses.cz/id/811r0e/.

[10] Hsu, C. C.-Y.; Mendler Dünner, C. a Hardt, M. *Revisiting Design Choices in Proximal Policy Optimization*. 2020. Dostupné z: https://arxiv.org/abs/2009.10897.

[11] Liang, E.; Liaw, R.; Moritz, P.; Nishihara, R.; Fox, R. et al. *RLlib: Abstractions for Distributed Reinforcement Learning*. 2018.

[12] Lindner, J. *Statistics About The Average Reaction Time* online. February 2024. Dostupné z: https://gitnux.org/average-reaction-time/. [cit. 2024-03-25].

[13] Mliu92. *Scotland Yard schematic* online. 2023. Dostupné z: https://commons.wikimedia.org/wiki/File:Scotland_Yard_schematic.svg. [cit. 2024-03-20].

[14] Newborn, M. *Kasparov versus deep blue: computer chess comes of age*. 1. vyd. Springer-VerlagBerlin, Heidelberg, 1996. ISBN 978-0-387-94820-1.

[15] OpenAI; :; Berner, C.; Brockman, G.; Chan, B. et al. *Dota 2 with Large Scale Deep Reinforcement Learning*. 2019. Dostupné z: https://arxiv.org/abs/1912.06680.

[16] OpenAI. Emergent tool use from multi-agent interaction. online, September 2019. Dostupné z: https://openai.com/research/emergent-tool-use. [cit. 2024-03-27].

[17] Roy, R. *ML | Monte Carlo Tree Search (MCTS)* online. May 2023. Dostupné z: https://www.geeksforgeeks.org/ml-monte-carlo-tree-search-mcts. [cit. 2024-03-25].

[18] Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A. a Klimov, O. *Proximal Policy Optimization Algorithms*. 2017. Dostupné z: https://arxiv.org/abs/1707.06347.

[19] Sutton, R. S.; McAllester, D.; Singh, S. a Mansour, Y. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In: Solla, S.; Leen, T. a Müller, K., ed. *Advances in Neural Information Processing Systems*. MIT Press, 1999, sv. 12. Dostupné z: https://proceedings.neurips.cc/paper_files/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf.

[20] Team, T. R. *Ray Documentation*. 2024. Dostupné z: https://docs.ray.io.

[21] Weng, L. *From GAN to WGAN*. 2017. Dostupné z: https://lilianweng.github.io/posts/2017-08-20-gan/#kullbackleibler-and-jensenshannon-divergence.

[22] Weng, L. *A (Long) Peek into Reinforcement Learning* online. February 2018. Dostupné z: https://lilianweng.github.io/posts/2018-02-19-rl-overview/#key-concepts. [cit. 2024-03-20].

# Příloha A

# Obsah přiloženého paměťového média

- `src/` – Složka se zdrojovými kódy aplikace.
- `text/` – Složka s zdrojovými kódy textu práce v jazyce LaTeX.
- `trained_models_dqn/` – Složka s natrénovaným modelem DQN.
- `trained_models_ppo/` – Složka s natrénovaným modelem PPO
- `thesis.pdf` – Soubor s textem práce.
- `thesis_print.pdf` – Soubor s textem práce pro tisk.
- `README.md` – README soubor pro tuto práci.
- `simulations/` – Složka s výsledky simulací.
- `arial.ttf` – Písmo pro grafické rozhraní
- `main.py` – Soubor, jehož spuštěním se spustí grafické rozhraní hry Scotland Yard.
- `TrainerDQN.py` – Soubor obstarávající trénování modelu DQN.
- `TrainerPPO.py` – Soubor obstarávající trénování modelu PPO.
- `tune_dqn.py` – Tune byl použit pro hledání nejlepších hyperparametrů pro model DQN.
- `tune_ppo.py` – Tune byl použit pro hledání nejlepších hyperparametrů pro model PPO.
- `requirements.txt` – Soubor s python balíčky potřebnými pro spuštění aplikace.