VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

STRATEGICKÁ DESKOVÁ HRA S NEURČITOSTÍ

STRATEGIC GAME WITH UNCERTAINITY

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **MICHAL SOVA**
AUTHOR

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

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Sova Michal**
Program: Informační technologie
Název: **Strategická desková hra s neurčitostí**
Strategic Game with Uncertainity
Kategorie: Umělá inteligence

Zadání:

1. Seznamte se s pravidly deskových her, kde aktuální stav hry bývá pro hráče po většinu času utajený. Mezi takové deskové hry patří například hra "Scotland Yard".
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, to znamená jak pro figuru, která je hledána, tak pro figury, které ji hledají, 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:

1. Russel, S., Norvig, P.: Artificial Intelligence, A Modern Approach, Pearson, 2009

Pro udělení zápočtu za první semestr je požadováno:

- První dva body zadání.

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. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 11. listopadu 2020

Abstrakt

Tato páce řeší autonomní hraní hry Scotland Yard za využití metody strojového učení. Daný problém je vyřešen pomocí algoritmu Monte Carlo tree search. Algoritmus Monte Carlo tree search byl testován proti algoritmu Alfa-beta. Výsledky testování ukázaly, že navržený algoritmus je funkční, ale procento výher u algoritmu Monte Carlo tree search je nižší než u algoritmu Alfa-beta. Výsledkem práce je funkční verze systému, který hraje zjednodušenou verzi hry Scotland Yard autonomně. Dále práce obsahuje rozšíření hry Scotland Yard přidáním agentů a změnou parametrů. Rozšíření této verze pro algoritmus Alfa-beta nebylo úspěšné kvůli nedostatečné kapacity vlastních zdrojů. Naopak algoritmus Monte Carlo tree search se jeví v tomto ohledu úspěšnější.

Abstract

The thesis focuses on creating an autonomous system for the game Scotland Yard by using machine learning method. The problem is solved by algorithm Monte Carlo tree search. Algorithm Monte Carlo tree search was tested against algorithm Alpha-beta. These results showed that Monte Carlo tree search algorithm is operational but win rate of this algorithm is lower than win rate of algorithm Alpha-beta. The resulting system is functional, autonomous and capable of playing the game Scotland Yard on simplified game area. There was an attempt to expand simplified version of the game Scotland Yard. In expanded version algorithm Alpha-beta was not successful because of insufficient computational resources. Algorithm Monte Carlo tree search, on the other hand, was more successful in expanded version.

Klíčová slova

strojové učení, strategické hry, stolní hry, hry s neurčitostí, alfa-beta, Monte Carlo Tree Search (MCTS), Scotland Yard, Go, metody hraní her, neuronové sítě, AlphaGo

Keywords

machine learning, strategic games, board games, games with uncertainity, alpha-beta, Monte Carlo Tree Search (MCTS), Scotland Yard, Go, game theory methods, neural network, AlphaGo

Citace

SOVA, Michal. *Strategická desková hra s neurčitostí*. Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. Ing. František Zbořil, Ph.D.

Strategická desková hra s neurčitostí

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 Sova
10. května 2021

Poděkování

Rád bych poděkoval doc. Ing. Františkovi Zbořilovi, Ph.D. za odbornou pomoc při řešení bakalářské práce.

Obsah

Kapitola 1

Úvod

Deskové hry jsou určeny především pro zabavení skupiny hráčů na dlouhou dobu. Mezi takové hry patří i hra Scotland Yard, která patří spíše do kategorie složitějších deskových her. Díky její náročnosti a složitosti je výzvou pro ni navrhnout a implementovat algoritmus, který by tuto hru hrál autonomně.

Cílem této práce je navrhnout, implementovat a porovnat metodu s ostatními implementovanými metodami, které by byly schopné hrát samostatně hru Scotland Yard. Metoda by měla simulovat lidského hráče a rozhodovat o pohybu figur. Metoda by měla nalézt nejlepší možný tah. Možnost výběru nejlepšího možného tahu by však byla velmi časově a paměťově náročná pro vykonání počítačem. Další možností dané metody je v tomto případě výběr optimálního tahu, který by byl nejlepší za určitou dobu a s určitými prostředky.

V kapitole 2 jsou rozebrány metody, které jsou vhodné k řešení úloh různého typu. Tato kapitola popisuje metody hraní her a řešení úloh použité v této práci.

V kapitole 3 je vysvětlen a popsán princip stolních her. V této kapitole je taktéž vysvětlena další složitá hra Go a přístup řešení algoritmu, který hraje tuto hru autonomně. Zde jsou taktéž vysvětleny principy deskové hry Scotland Yard.

Kapitola 4 je zaměřena na návrh metody, která by hrála hru Scotland Yard a byla by soupeřem lidskému protihráči. Dále je zde vysvětlena již existující metoda pro hraní zjednodušené verze hry Scotland Yard. Kapitola je dále zaměřena na návrh zjednodušení hry Scotland Yard pro účely testování metody.

V kapitole 5 je popsána implementace vybrané metody a experimenty s implementovanými metodami. Experimenty jsou vyhodnoceny a odůvodněny. Dále se kapitola zabývá škálováním zjednodušené verze hry Scotland Yard. Důvodem škálování je ověření implementace metod s rozšířeným modelem, kde metody jsou implementovány pouze na zjednodušeném modelu hry Scotland Yard.

Kapitola 2

Metody řešení úloh

Metod řešení úloh je hodně a každá se hodí nebo postačuje jen na určitý typ problému. V této kapitole jsou uvedeny a vysvětleny některé metody řešení úloh, které se používají k řešení různě složitých problémů. Některé by mohly být vhodné pro řešení problému, kterým se daná práce zabývá. Jiné jsou nezbytné pro pochopení těchto složitějších metod.

Informace z této kapitoly jsou převzaté především ze studijní opory F. V. Zbořila a F. Zbořila [15] a práce V. Neřáda [10]. Dále jsem čerpal z práce S. Russella a P. Norviga [1]. Informace o neuronových sítích jsem čerpal ze stránky IBM [6] a knihy Umělá inteligence [9]. Informace pro metodu Monte Carlo tree search, kterou se práce blíže zabývá, jsem čerpal ze stránek GeeksForGeeks [11], online článku o Monte Carlo tree search [5], článku Monte Carlo Tree Search for the Game of Diplomacy [13] a článku Overview on DeepMind and Its AlphaGo Zero AI [7].

2.1 Metody hraní her

Pro jednoduchost se zaměříme pouze na hry mezi dvěma hráči (příp. týmy hráčů), kteří se po jednotlivých tazích hry pravidelně střídají. Tyto hry můžeme rozdělit na jednoduché hry a hry složité. Jako jednoduché hry můžeme označit takové hry, u kterých lze celý jejich stavový prostor prohledat v rozumném čase. Mezi takové hry patří například hra piškvorky s velikostí pole 3×3, kde počet možných stavů na hráčův tah je nejvýše 9. Většina her je ale složitých, kde nelze v rozumném čase prohledat celý stavový prostor. Tyto metody rozkládají problém na podproblémy a jsou přirozenými metodami, které při řešení obtížných úkolů používá i člověk. V následujících sekcích budou vysvětleny některé stromové metody používané při hraní her. Tyto metody jsou:

- AND/OR grafy
- Minimax algoritmus
- Alfa a Beta řezy (Alpha-beta pruning)

2.1.1 AND/OR grafy

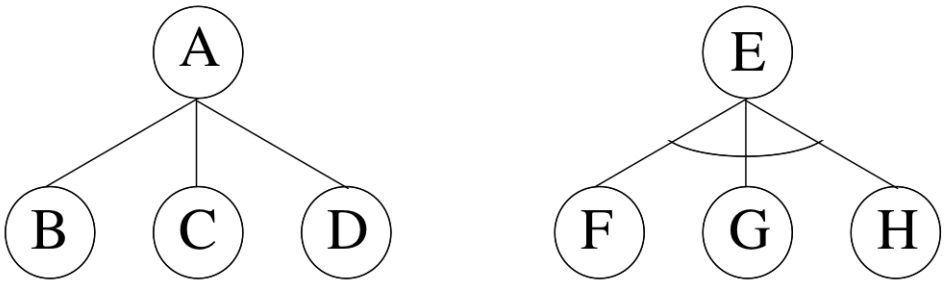
AND/OR graf rozkládá problém na dva podproblémy:

- Problém AND, který je řešitelný pouze pokud všechny jeho podproblémy jsou řešitelné.

- Problém OR, který je řešitelný pokud alespoň jeden z jeho podproblémů je řešitelný.

Tyto stavy si lze představit jako možné tahy jednotlivých hráčů. Stav OR je tah hráče, jelikož mu stačí jen jeden tah ze všech možných tahů k vítězství. Stav AND je tah protihráče, protože může vybrat kterýkoli z možných tahů a hráč musí počítat s tím, že protihráč vybere ten nejlepší. Tato metoda se spíše hodí pro jednoduché hry, jelikož je třeba znát cílové stavy a vygenerovat dostatečný počet koncových stavů. Toto u složitějších her v rozumném čase není možné.

Na obrázku 2.1 lze vidět příklad problému AND (vpravo) a problému OR (vlevo). Uzel A (problém OR) je na tomto obrázku řešitelný, pokud alespoň jeden z uzlů B, C nebo D je řešitelný. Uzel B (problém AND) je na tomto obrázku řešitelný, pokud všchny z uzlů F, G a H jsou řešitelné.



Obrázek 2.1: Ukázka problému OR a AND (Zdroj: [15])

2.1.2 MiniMax algoritmus

Složité hry již nelze řešit čistým AND/OR grafem, jelikož stavový prostor je příliš velký. U těchto her se prohledává pouze do předem daného počtu tahů nebo do koncového stavu, který by rozhodl o řešitelnosti/neřešitelnosti uzlů grafu. Z tohoto důvodu je třeba uzly nějakým způsobem ohodnotit.

Při MiniMax algoritmu je využito AND/OR grafu s tím rozdílem, že algoritmus generuje stavy do zadané hloubky nebo pokud nenajde řešení. Listovým stavům jsou pak přiřazeny hodnoty, na základě vhodně zvolené heuristické funkce. Následně jsou vráceny rodičovským uzlům získané hodnoty. V případě, že se jedná o OR problém, vrací se hodnota největšího z listů. Pokud jde o AND problém, zapíše se do daného uzlu nejmenší hodnota z listů. Tímto způsobem se propagují hodnoty až do kořenového uzlu,tj. aktuálníhu stavu hry. Algoritmus pak vybere tah k uzlu s nejvyšším ohodnocením, tj. tah který je pro daného hráče nejvýhodnější.

2.1.3 Alfa a Beta řezy

Alfa a beta řezy (též Alfa-beta přeřezávání nebo jen Alfa-beta) se v mnohém podobají algoritmu MiniMax. Narozdíl od metody MiniMax metoda Alfa-beta zabraňuje zbytečnému vyšetřování uzlů, které s jistotou nemají vliv na výsledek algoritmu. Alfa řezy zabraňují zbytečnému prohledávání hráčových tahů a beta řezy zabraňují zbytečnému prohledávání protihráčových tahů. K tomu přispívají dvě proměnné, proměnná α a proměnná β.

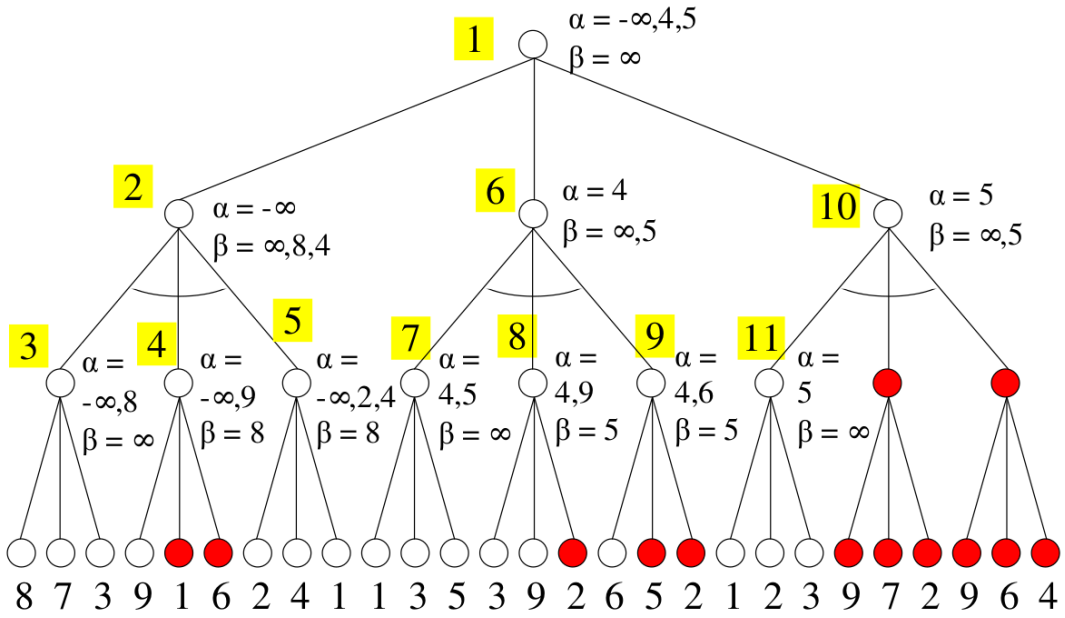
Princip procedury Alfa-beta:

1. Vezměme si daný uzel a nazvěme ho uzlem X.
2. Pokud je uzel X kořenovým uzlem, nastav jeho proměnné $\alpha = -\infty$, $\beta = \infty$ (obecně nastav hodnotu α na minimální možnou hodnotu a hodnotu β na maximální).
3. Pokud je daný uzel X listem (konečným uzlem, nebo uzlem v maximální hloubce) ukonči proceduru a vrať ohodnocení tohoto uzlu.
4. Pokud je daný uzel X typu OR (na tahu je hráč):
   - 4.1 Dokud je $\alpha < \beta$, tak postupně pro následující tah (bezprostředního následníka uzlu X a protihráče) volej proceduru AlfaBeta s aktuálními hodnotami proměnných α a β.
   - 4.2 Po každém vyšetřeném tahu nastav hodnotu proměnné α na maximum z aktuální a navrácené hodnoty.
   - 4.3 Ukonči proceduru, vrať aktuální hodnotu proměnné α a pro kořenový uzel vrať i tah, který vede k nejlépe ohodnocenému bezprostřednímu následníkovi.
5. Pokud je daný uzel X typu AND (na tahu je protihráč):
   - 5.1 Dokud je $\alpha < \beta$, tak postupně pro následující tah (bezprostředního následníka uzlu X a hráče) volej proceduru AlfaBeta s aktuálními hodnotami proměnných α a β.
   - 5.2 Po každém vyšetřeném tahu nastav hodnotu proměnné β na minimum z aktuální a navrácené hodnoty.
   - 5.3 Ukonči proceduru a vrať aktuální hodnotu proměnné β.

Díky této proceduře lze snížit množství výpočetního času na daný tah.

Fungování této metody je možné si vysvětlit na příkladu. V obrázku 2.2 lze vidět ukázka stromu metody Alfa-beta. Na tomto obrázku jsou znázorněni 2 hráči, hráč A a hráč B. Hráč A volá proceduru Alfa-beta pro aktuální uzel označený číslem 1 (kořenový uzel) a pro uzel 2 hráče B. Do těchto uzlů, jelikož nejsou listovými (koncovými) uzly, se nastaví hodnoty $\alpha = -\infty$ a $\beta = \infty$. Hráč B volá proceduru Alfa-beta pro uzel 2 a pro uzel 3 hráče A a nastaví hodnoty $\alpha = -\infty$ a $\beta = \infty$ pro dané uzly. Hráč A volá proceduru Alfa-beta pro uzel 3 a pro listové uzly hráče B. Jelikož jsou uzly expandované z uzlu 3 listovými, tyto uzly jsou postupně ohodnoceny Alfa-betou. Jelikož to jsou uzly hráče B (typu OR), hodnota α uzlu 3 je postupně upravována na maximální ohodnocení z jednotlivých koncových uzlů, které jsou bezprostředními následníky uzlu 3. Hodnoty uzlu 3 je pak $\alpha = 8$ a $\beta = \infty$. Hráč A nakonec vrátí hodnotu 8. Uzel 2 pak touto hodnotou nastaví $\beta = 8$, jelikož se jedná o problém AND a $\beta = min(8, \infty)$. Uzel 2 má v tomto okamžiku hodnoty $\alpha = -\infty$ a $\beta = 8$. Hráč B poté volá z uzlu 2 proceduru Alfa-beta pro hráče A a uzlu 4. Hodnoty uzlu 4 jsou nastaveny z uzlu 2 $\alpha = -\infty$ a $\beta = 8$. Hráč A z uzlu 4 volá proceduru postupně pro listové uzly hráče B. První listový uzel hráče B je ohodnocen 9. Tato hodnota se zapíše do hodnoty $\alpha = 9$ uzlu 4. Jelikož je $\alpha \geq \beta$, procedura pro uzel 4 je ukončena a uzlu 2 je vrácena hodnota 9. Jelikož $\beta = min(8, 9)$, zůstane hodnota $\beta = 8$ v uzlu 2 nezměněna. Volá se pak procedura Alfa-beta pro uzel 5. V tomto uzlu je největší ohodnocení koncových uzlů 4. Tato hodnota se vrátí z uzlu 5 do uzlu 2, kde se zapíše do $\beta = min(8, 4)$. Uzel 2 poté vrátí hodnotu β a uzel 1 ji zapíše do $\alpha = 4$. Z uzlu 1 se použije procedura Alfa-beta na uzel 6. Hodnoty uzlu 6 jsou stejné jako hodnoty uzlu 1, $\alpha = 4$ a $\beta = \infty$. Z uzlu 6 se volá procedura Alfa-beta na

uzel 7. V uzlu 7 se zvýší hodnota α z 4 na 5. z uzlu 7 se vrátí hodnota 5, která se zapíše do hodnoty uzlu 6 $\beta = min(5, \infty)$. Hodnoty uzlu 6 jsou $\alpha = 4$ a $\beta = 5$. Následuje uzel 8. V uzlu se postupně ohodnocují listové uzly. Jakmile se narazí na uzel s ohodnocením 9 ($\alpha \geq \beta$), uzel se dále již neprohledává. U uzlu 9 nastane ten samý příklad kdy $\alpha \geq \beta$ a prohledávání uzlu 9 se ukončí hned po ohodnocení prvního listového uzlu s hodnotou 6. Hodnota uzlu 6 $\beta = 5$ se vrátí a uzel 1 ji porovná s aktuální hodnotou $\alpha = max(4, 5)$. Aktuální hodnoty uzlu 1 jsou $\alpha = 5$ a $\beta = \infty$. Z uzlu 1 se zavolá procedura Alfa-beta na uzel 10. Hodnoty tohoto uzlu jsou stejné, jako hodnoty uzlu 1. Z uzlu 10 se zavolá procedura Alfa-beta na uzel 11. Hodnoty uzlu 11 jsou $\alpha = 5$ a $\beta = \infty$. Jelikož žádný z listových uzlů nemá větší ohodnocení jak hodnota uzlu 11 $\alpha = 5$, vrátí uzel 11 hodnotu 5. Hodnota β se v uzlu 10 přepíše na 5 a protože platí $\alpha \geq \beta$, algoritmus již dále ve vyšetřování uzlu nepokračuje. Vrátí se hodnota uzlu 10, která je 5. Vzhedem k tomu že je hodnota α stejná jako navrácená hodnota z uzlu 10, nic se nemění.



Obrázek 2.2: Příklad stromu Alfa a beta (Zdroj: [15])

2.2 Metody pro hraní složitějších her

V již zmíněných metodách může nastat situace, kdy vybraný nejlepší tah nebude nejlepší z dlouhodobého hlediska.

Tak je tomu například u hry Go[1] nebo právě u hry Scotland Yard[2], kdy stavový prostor bývá obrovský i při malé hloubce. V takových případech tyto jednoduché metody nestačí a nejsou vhodné, jelikož výpočet jednoho tahu by trval dlouho a bylo by potřeba velké množství prostředků. U hry Go bylo k takovému případu využita neuronová síť spolu s metodou Monte Carlo tree search 3.2.1. Z tohoto důvodu by mohlo být vhodné použití metody Monte Carlo tree search a použití strojového učení.

[1] podrobněji vysvětleno v kapitole 3.2
[2] pravidla hry vysvětleny v kapitole 3.3

2.2.1 Strojové učení

Strojové učení je disciplína, která je zaměřená na algoritmy a techniky, které na základě zkušeností a dat zlepšují efektivitu inteligentních systémů aniž by člověk měnil daný algoritmus. Strojové učení se používá, pokud neznáme přesné řešení problému nebo nevíme, jakým způsobem se dostaneme k řešení problému. Strojové učení můžeme rozdělit na 3 základní skupiny:

- Učení s učitelem (*supervised learning*) - používá testovací sadu. Testovací sada je označená množina vstupů a předpokládaných výstupů. To umožňuje algoritmu se na základě této množiny dál učit a rozhodovat.
- Učení bez učitele (*unsupervised learning*) - používá neoznačenou množinu vstupů, kde žádná jiná informace není dostupná. Algoritmus spočívá v nalezení podobností této datové sady. Při učení se využívá shlukové analýzy (*clustering*), kde jsou objekty shlukovány na základě podobných vlastností.
- Posilované učení (*reinforcement learning*) - používá ohodnocení daného rozhodnutí nebo dané akce a systém odměn. Odměny mohou být kladné i záporné. Systém se snaží maximalizovat výsledný součet odměn.

2.2.2 Monte Carlo tree search

Monte Carlo tree search (zkráceně MCTS) je metoda, která vyhodnocuje stavy pomocí simulace výsledků hry z daného stavu. MCTS používá tzv. *exploration-exploitation trade-off*. Využívá doposud nejlepší nalezené tahy ale zároveň musí pokračovat s hledáním tahu, který by mohl být lepší, než ten prozatím nejlepší.

Tato metoda by se dala přiřadit k disciplíně strojového učení, konkrétně do skupiny posilovaného učení. MCTS se dá přiřadit do skupiny posilovaného učení, jelikož hodnotí konečný stav, toto hodnocení se snaží maximalizovat a na základě tohoto hodnocení se dále rozhoduje.

Proces algoritmu můžeme rozdělit do 4 následujících kroků:

1. Výběr uzlu (*Selection*): Výběr začíná od kořenového uzlu a pokračuje až k listovým. Vybírají se postupně uzly s nejvyšším potenciálem, dokud se nedojde až k listovým. Vybere se listový uzel s nejlepším potenciálem. Nazvěme vybraný uzel uzlem A.
2. Expanze (*Expansion*): Pokud vybraný uzel A není konečný (konec hry) a pokud uzel A byl již navštíven, algoritmus expanduje daný uzel a vybere náhodně jeden z nově expandovaných uzlů (pokud uzel A nebyl navštíven neexpanduje se a nadále se s ním počítá jako s uzlem B). Tento uzel nazvěme uzlem B.
3. Simulace (*Simulation*): Z uzlu B algoritmus MCTS simuluje tahy náhodně nebo na základě různých strategií. Simulace poběží do koncového stavu (konec hry), nebo do předem definovaného stavu.
4. Zpětná propagace (*Backpropagation*): Hodnota uzlu B je aktualizována na základě výsledného stavu simulace, v nejjednodušším případě hodnota výhry nebo prohry. Algoritmus následně aktualizuje a propaguje hodnotu rodičovských uzlů, tj. uzel bezprostředně nad probíraným uzlem[3], dokud nedojde ke kořenovému uzlu.

[3] v případě uzlu B je rodičovským uzelem uzel A

Tyto 4 kroky se opakují dokud je dostatek prostředků.

MCTS pro výběr uzlu používá v základu vzorec tzv. *Upper Confidence Bound* (UCB), který vypadá následovně:

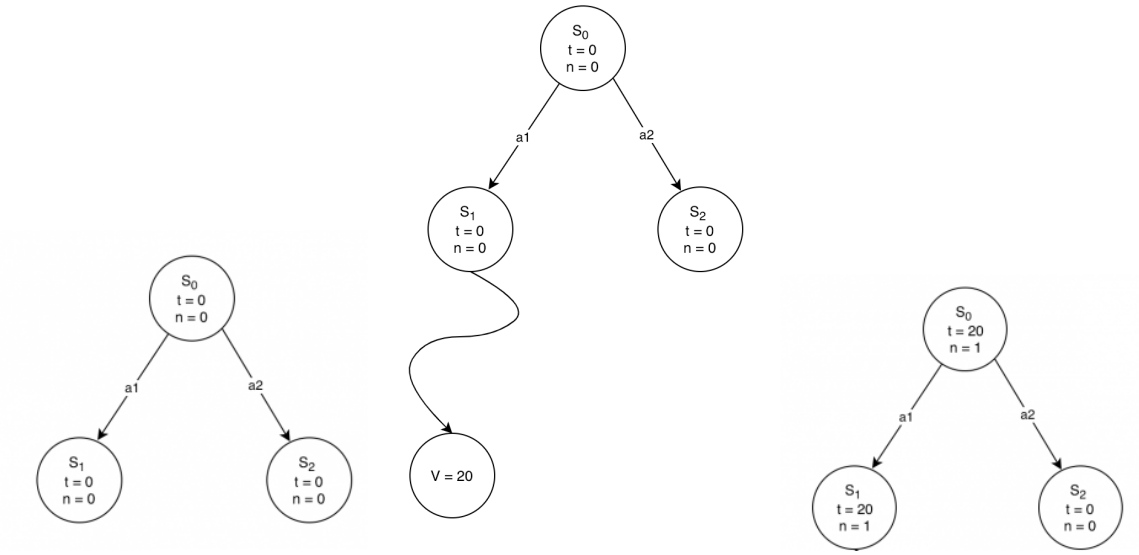
$$UCB1 = V_i + C * \sqrt{\frac{\ln N}{n_i}} \tag{2.1}$$

Kde:

- V_i = vážená odměna/hodnota v daném uzlu
- C = je vhodně zvolená konstanta
- N = kolikrát byl navštíven rodičovský uzel
- n_i = kolikrát byl navštíven daný uzel

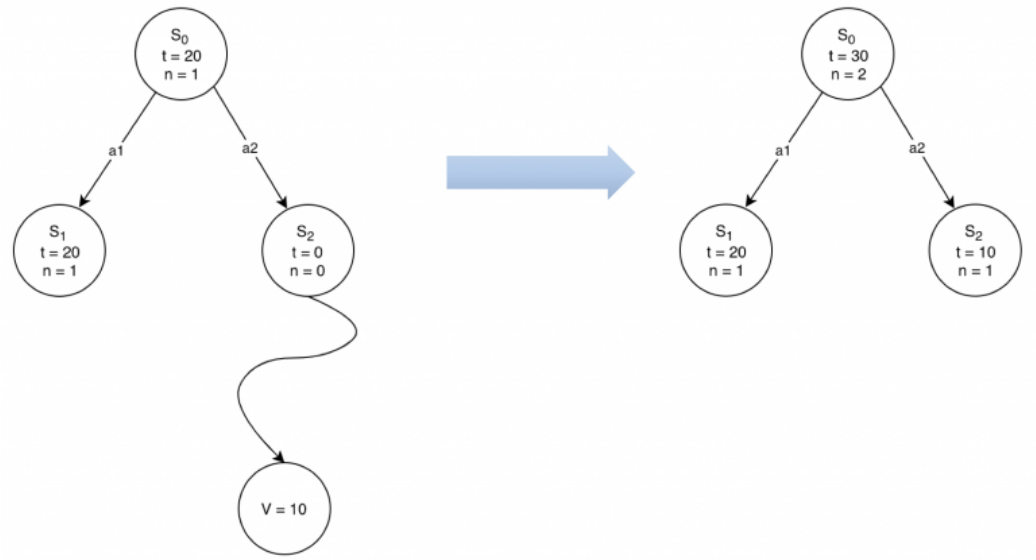
Po vyčerpání přidělených prostředků (vypršení časového limitu nebo dosáhnutí zadaného počtu cyklů) se vybere bezprostřední následník kořenového uzlu s nejvyšší hodnotou/odměnou. Pro efektivní dosažení optimálního tahu je třeba velké množství iterací a ani to nezajistí nalezení nejlepšího tahu.

Na obrázku 2.3 je znázorněn příklad jedné iterace tohoto algoritmu. Na obrázku vlevo je vidět počáteční stav metody MCTS. Metoda začíná v uzlu S_0. Do stavů S_1 a S_2 se dostaneme pomocí akcí $a1$ a $a2$. Hodnota t označuje ohodnocení uzlu a hodnota n značí počet navštívení daného uzlu. Nejdříve se podle metody MCTS vybere uzel s nejvyšším UCB. Jelikož oba uzly S_1 a S_2 ještě nebyly navštíveny, jejich UCB je v obou případech nekonečný. Metoda proto vybere první uzel. Vybraný uzel S_1 je uzlem listovým, který nebyl navštíven ($n = 0$). Proto z tohoto uzlu metoda MCTS provede simulaci a neexpanduje ho. Výsledek simulace lze vidět na obrázku 2.3 uprostřed, kde hodnota $V = 20$ je výsledek provedené simulace. Metoda MCTS poté změní hodnoty nejdříve v S_1 na $t = 20$ a $n = 1$. Následně metoda změní hodnoty v S_0 na $t = 20$ a $n = 1$. Výsledek zpětné propagace (a tudíž první iterace MCTS) lze vidět na obrázku 2.3 vpravo.



Obrázek 2.3: Příklad 1 průběhu algoritmu Monte Carlo tree search (Zdroj: [5])

Na obrázku 2.4 vlevo lze vidět pokračování metody. Metoda MCTS opět začne v kořenovém uzlu S_0. Následně metoda MCTS vybírá uzel s nejvyšším UCB. Jelikož uzel S_2 ještě nebyl navštíven a hodnota UCB je rovna nekonečnu, vybere metoda MCTS uzel S_2. Z tohoto uzlu metoda MCTS provede simulaci, která má výsledek 10. Tato hodnota se propaguje až ke kořenovému uzlu. Hodnoty uzlu S_2 se změní na $t = 10$ a $n = 1$. Hodnoty uzlu S_0 se změní na $t = 20 + 10$ a $n = 1 + 1$. Výsledný stav lze vidět na obrázku 2.4 vpravo. Následně metoda MCTS pokračuje a vypočte nejdříve hodnotu UCB uzlu S_1, která je $20 + 2 * \sqrt{\frac{\ln 2}{1}} = 21.67$. Metoda MCTS vypočte samozřejmě i hodnotu UCB uzlu S_2, která je $10 + 2 * \sqrt{\frac{\ln 2}{1}} = 11.67$. Metoda MCTS si vybere uzel s vyšším UCB, což je uzel S_1. Uzel S_1 není koncovým, tudíž metoda MCTS uzel S_1 expanduje. Následně vybere jeden z expandovaných uzlů. Z tohoto uzlu provede metoda MCTS opět simulaci a výsledek simulace zpropaguje až ke kořenovému uzlu. Tento postup se opakuje dokud je potřeba nebo do vyčerpání prostředků (konečný počet cyklů nebo vymezený časový úsek pro algoritmus). Po vyčerpání prostředků metoda vybere akci $a1$ nebo $a2$ z obrázku 2.4, která má největší hodnotu t.



Obrázek 2.4: Příklad 2 průběhu algoritmu Monte Carlo tree search (Zdroj: [5])

2.2.3 Neuronové sítě

Neuronová síť je model založený na struktuře lidského mozku. Model napodobuje chování a způsob přesunu informace mezi jednotlivými biologickými neurony[4]. Neuronová síť je tvořena vrstvami. Vstupní vrstva, jedna nebo více skrytých vrstev a výstupní vrstva. Každá vrstva je tvořena uzly, formálními neurony. Každý neuron je spojen s dalším neuronem a má svou váhu a práh (*bias, threshold*).

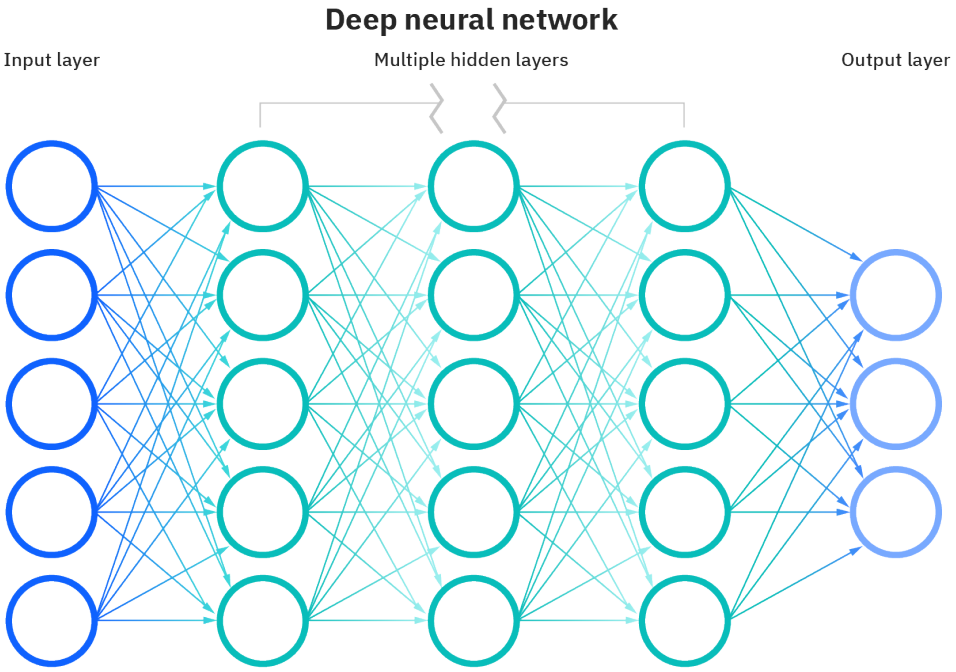
Formální neuron je základní stavební jednotka neuronové sítě. Formální neuron se skládá obecně ze vstupů, práhu a výstupu. Vstupy jsou ohodnoceny váhami, které určují důležitost

[4]Neuron je buňka centrálního nervového systému, která má schopnost přijímat a dekódovat informace ve formě elektrických a chemických signálů a přenášet je do jiných buněk. [https://cs.encyclopedia-titanica.com/significado-de-neurona]

daného vstupu. Váhy mohou být i záporné. Vážený součet vstupních hodnot označuje tzv. potenciál neuronu. Neuron je aktivován, pokud je překročen práh. Výstup závisí na aktivační přenosové funkci $S(\xi)$, kde ξ je potenciál neuronu.

Architektura neuronových sítí je různá, ale jednou z nejznámějších architektur je uspořádání neuronů do vrstev. Každá vrstva může obsahovat tisíce neuronů propojených mezi sebou. Neurony jsou mezi sebou spojeny vždy jedním směrem a to od vstupní vrstvy k výstupní. Tak je tomu na obrázku 2.5, kde je vidět tzv. hluboká neuronová síť (*Deep Neural Network*). Hluboká neuronová síť se liší od normální v architektuře sítě [8]. Rozdílem mezi hlubokou neuronovou sítí a jinou neuronovou sítí je ten, že obsahuje 2 a více vrstev mezi vstupní a výstupní vrstvou. Tyto vrsvy jsou schované a pro okolí nejsou vidět.

Neuronové sítě se musí nejprve učit na vstupních datech, než mohou být použity. Učení neuronové sítě je iterativní proces, kdy je síti představen vstup a na základě předpokládaného výstupu jsou neuronům přerozděleny jejich váhy. Tento proces může trvat hodiny, dny nebo i měsíce.



Obrázek 2.5: Ukázka neuronové sítě (Zdroj: [6])

Kapitola 3

Stolní hry

Existuje velké množství stolních her, některé závisí částečně na pravděpodobnosti, u jiných záleží na strategii. Tato kapitola se bude hlavně zabývat hrami s velkým množstvím možných tahů, osvětlí princip těchto her a problematiku, spojenou právě s velkým množstvím tahů. Aspekty této problematiky je možné připodobnit ke hře Go nebo hře Scotland Yard.

Poslední podkapitola je věnována hře Scotland Yard, které se tato práce blíže zabývá. V této podkapitole je popsáno, čím je hra Scotland Yard specifická, čím se liší od ostatních her a čím je podobná. Navíc jsou v dané podkapitole popsány základní pravidla hry Scotland Yard.

Zdrojem informací pro tuto kapitolu jsou především Overview on DeepMind [7], Mastering the game of Go [12] a článek AlphaZero, a novel Reinforcement Learning Algorithm [2]. Pravidla hry Scotland Yard jsem čerpal ze stránky Ultra Board Games [4].

3.1 Tahové hry s velkým množstvím tahů

Většina tahových her je zaměřena na strategii a promyšlenosti následujícího tahu, který by mohl zajistit vítězství daného hráče. U některých tahových her je každý hráč sám za sebe. Tak je tomu například u hry Go. U jiných hráči musí spolupracovat, aby dosáhnuli společného cíle. Tak je tomu například u hry Scotland Yard. Ve strategii těchto her se musí počítat s možnými tahy hráče a posoudit, které tahy jsou pro něj nejlepší. Těchto možných tahů bývá u her, jako je například hra Go, velmi mnoho. Hráč ve své strategii musí zahrnout svoje tahy, ale taktéž možnosti protihráče. Díky tomu je náročnost výběru tahu ještě vyšší. Mnohdy mají hráči navíc omezenou dobu na samotný tah nebo na celou hru. Touto skutečností je náročnost vykonání tahu ještě umocněna.

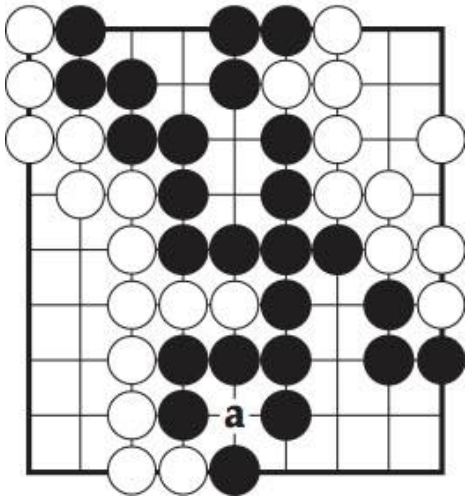
3.2 Hra Go

K deskovým hrám s velkým množstvím možných tahů patří například hra Go [3]. Go je strategická desková hra, která pochází z Číny. Jedná se o jednu z nejstarších deskových her.

Hru hrají 2 hráči na hrací desce, která má 19 řádků a 19 sloupců. Další alternativou hry Go jsou menší hrací desky s velikostí 13×13 a 9×9. Hra začíná s prázdnou hrací deskou. Každý hráč má dostatečné množtví kamenů, jeden bílých, druhý černých. Hráč s černými kameny začíná a hráči se postupně střídají na tahu. Cílem hráče je vytvořit svými kameny teritorium, za které dostávají na konci hry body podle velikosti, které je vyznačeno právě jeho kameny. Taktéž je možné chytit kameny protihráče jejich obklopením. Konec

hry nastává v případě, kdy se jeden z hráčů vzdá. Na konci hry hráči spočítají body. Hráč s více body vyhrává.

Na obrázku 3.1 je ukázka konečné pozice hry, kde černý obklíčil svými kameny bílý kámen. Toto místo je označeno písmenem 'a'. Za teritorium se počítá volné pole obklíčené kameny daného hráče. Černý v tomto případě získal se svými kameny 10 bodů za pravý spodní roh a 5 bodů za horní část hrací plochy. K tomuto skóre je třeba přidat ještě obklíčené kameny bílého hráče. Dohromady tedy černý hráč získal 16 bodů. Bílý hráč sice neobklíčil žádné kameny černého hráče, ale zato má větší teritorium. Bílý za teritorium nakonec získal 17 bodů, čímž vyhrává.



Obrázek 3.1: Ukázka pozice na konci hry Go na ploše 9 × 9 (Zdroj: [3])

3.2.1 Go a AlphaGo

Ačkoli jsou pravidla této hry relativně jednoduchá, možností vykonání jednoho tahu je velmi hodně. Při hře jako je Go je třeba přemýšlet několik tahů dopředu. Vyhodnotit možné vlastní tahy a poté tahy protihráče v tak rozsáhlém stavovém prostoru na několik kol dopředu je pro člověka velmi náročné, což je ideální výzvou pro svět informačních technologií a umělé inteligence.

AlphaGo od společnosti Google DeepMind[1] je první program, který se vyrovnal a předčil člověka ve hře Go. V roce 2015 první publikovaná verze AlphaGo Fan[2] porazila evropského mistra Fan Hui (tehdy 2 dan[3]) 5-0. AlphaGo Fan používala Monte Carlo tree search spolu se dvěma neuronovými sítěmi k vyhodnocení možných tahů.

- *Policy network* trénována na hrách profesionálních hráčů, jejímž výstupem jsou nejzajímavější tahy.
- *Value network* která vyhodnocuje pravděpodobnost výhry na dané pozici.

[1] Společnost pro výzkum a vývoj umělé inteligence [https://deepmind.com]
[2] AlphaGo Fan - pojmenována podle hry s evropským mistrem Fan Hui
[3] dan se dá považovat za herní zdatnost nebo jak dobře kdo hraje v Go, 9 je nejvyšší

Dané neuronové sítě byly natrénovány velkým množstvím člověkem hraných her. Vylepšená verze porazila téměř o rok později mistra 9 dan jménem Lee Sedol 4-1. Tato verze byla nazvána AlphaGo Lee.

Nejnovější verze AlphaGo Zero je rozdílná od svých předchůdců hned v několika směrech. Jedna z největších změn, je způsob trénování. Je trénována čistě hraním sama se sebou bez interakce ze strany člověka, či analýzou her, hraných jinými hráči. Další významná změna oproti svým předchůdcům je jednodušší implementace Monte Carlo tree search, který používá jednu neuronovou síť místo dvou. Vstupní data této neuronové sítě je rozložení hrací plochy a její výstup je distribuce pravděpodobností pro všechny tahy s ohodnocení hrací plochy. Tato pravděpodobnost určuje jakousi zajímavost nebo hratelnost tahu. Čím vyšší má tah pravděpodobnost, tím spíše se daný tah zahraje. Metoda Monte Carlo tree search pomocí simulací a expazí těchto hratelných tahů určí, který z tahů by mohl vést k vítězství. Ohodnocení hrací plochy neuronovou sítí značí, jaká je pro AlphaGo Zero šance na vítězství z této pozice hry. Toto ohodnocení se pohybuje od hodnoty -1 (prohra) do hodnoty 1 (výhra). Pokud neuronová síť vrátí například ohodnocení hry 0.9, myslí si AlphaGo Zero že je daná pozice naprosto výherní.

Tuto neuronovou síť však bylo třeba natrénovat. Trénování neuronové sítě probíholo podle následujících bodů:

1. Program se nechal hrát si proti sobě několik her. Program nahrával stav hry v každém tahu. Jakmile znal program výsledek dané hry, aktualizoval všechny stavy s daným výsledkem hry (výhra nebo prohra). Nyní jsou vytvořena data pro trénování neuronové sítě.

2. Neuronová síť se poté zkopírovala a kopie této sítě trénovala na datech z předchozího kroku.

3. Kopie a originální neuronová síť poté hráli proti sobě.

4. Vybrala se ta síť, která vyhrála. Síť která prohrála se zahodila.

5. Jde se zpět na krok 1.

Po mnoha iteracích byl vytvořen model neuronové sítě a neuronová síť byla natrénována a je připravená hrát.

3.3 Hra Scotland Yard

Scotland Yard je významná desková hra, při které hráči mohou vybrat 1 z mnoha dostupných tahů. V této části jsou vysvětleny základní pravidla hry a hlavní myšlenka hry. Podrobná pravidla jsou vysvětlena v příloze B

Scotland Yard je desková hra pro 6 hráčů. Hráči si rozdělí úlohy na 5 policistů/detektivů/agentů a jednoho zloděje/padoucha/Mr. X. Hráči se pohybují po různě propojených políčkách na hrací ploše. Velikost hrací plochy je okolo 200 políček. Hrací plocha se dá rozdělit do 4 sekcí spojených mezi sebou mostem nebo jiným zúženým místem. Cílem agentů je chytit Mr. X. Agenti musí spolupracovat, aby chytili Mr. X, zatímco Mr. X se pohybuje dle vlastního uvážení. Hra končí, když agenti chytí Mr. X, nebo ho obklíčí takovým způsobem, že Mr. X nemůže táhnout, aniž by skončil na políčku jednoho z agentů. V tomto případě vyhrávají agenti. V případě, kdy agenti nechytí Mr. X do 22. tahu, vítězí Mr. X.

Agenti i Mr. X se mohou po hrací ploše pohybovat jen na propojené vedlejší políčko a to následujícími způsoby:

- Taxi - nejběžnější prostředek, umožňuje přesun na vedlejší políčko, tvoří většinu cest na hrací ploše
- Autobus - méně běžný, umožňuje cestování na větší vzdálenost
- Metro - nejméně políček, umožňuje cestování přes značnou vzdálenost

Mr. X navíc může využít Trajektu, speciální způsob dopravy určený jen pro Mr. X. Ten spojuje pouze 4 políčka třemi cestami. Každý hráč má omezený, předem určený počet použití daného prostředku cesty. Tento počet je reprezentován jako jízdenky na jednotlivé dopravní prostředky. Hra začíná vylosováním startovních pozicí a následně tahem Mr. X. Následují postupně tahy agentů, kteří se nejlépe předem domluví, jak budou táhnout.

Mr. X je zvýhodněn hned několika způsoby. První výhodou Mr. X je neviditelnost jeho tahů - agenti většinu času nevidí tahy Mr. X, zatímco Mr. X vidí tahy agentů vždy. Agenti vidí tahy Mr. X pouze v 3., 8., 13. a 18. tahu. Agenti navíc vidí způsob dopravy, který Mr. X použil na daný tah. Nicméně Mr. X může použít černou jízdenku, čímž skryje způsob dopravy pro agenty. Agenti na základě toho, co jim bylo poskytnuto, musí vydedukovat pozici Mr. X a obklíčit ho. Tato neurčitost/nepředvídatelnost je klíčovým aspektem hry, kdy agenti odhadují tahy Mr.X a musí spolupracovat a domlouvat se na každém dalším tahu.

Herní plocha je rozdělena do 4 částí, spojené většinou úzkým přechodem. Díky tomuto faktu je optimální strategií agentů odříznout Mr. X od ostatních částí, čímž zvýší své šance na dopadení Mr. X. Agenti jsou navíc znevýhodněni na začátku hry, kdy je pozice Mr. X prozrazena až po 3. tahu. V tomto případě je nejlepší strategií agentů rozdělit si hrací prostor a dostat se blízko metra, kde rychle mohou dohnat Mr. X. Nejlepší strategií Mr. X je dostat se do míst, kde je velké množství možných tahů, aby zmátl agenty a měl co nejvíce cest na útěk.

Kapitola 4

Přístup k řešení hry Scotland Yard

Cílem práce je navrhnout a implementovat metodu, která by hrála tuto hru autonomně. I když existuje již několik mobilních i počítačových verzí hry Scotland Yard, žádná z nich (alespoň z těch, které jsem měl možnost prozkoumat) nehrála za agenty nebo Mr. X autonomě. Kvůli experimentům, iterativním úpravám algoritmu a složitosti hry, jsem se rozhodl zjednodušit pravidla a hrací prostor, tudíž zmenšit stavový prostor a zefektivnit úpravy algoritmu tak, aby měly změny větší vliv na výsledek. Navrhnutý algoritmus by měl být schopný konkurovat lidskému hráči a konkurovat již implementované metodě Alfa-beta.

Tato kapitola popisuje návrh řešení algoritmu pro hraní hry Scotland Yard autonomně. Tento návrh využívá poznatků a zjednodušení hry z práce A. Tulušáka [14].

4.1 Stavový prostor

Desková hra Scotland Yard[1] obsahuje okolo 200 hracích políček. Políčka jsou v průměru různě propojena dalšími 5 hracími políčky. To vychází v průměru na 5 tahů na jednoho agenta. Toto číslo roznásobíme mezi Mr. X a 5 agentů. Z čehož vyplývá, že pouze za jedno kolo je možná kombinace pohybu všech agentů a Mr. X v průměrnu $5^6 = 15625$ možných tahů z aktuálního stavu pro jedno kolo. Ve hře Scotland Yard je kol 22 (nanejvýš). Kdybychom použili například algoritmus MiniMax, velikost stromu a délka výpočtů pro jedno kolo by byla pro běžné hraní nesnesitelná. Obzvláště v našem případě, kde chceme vyzkoušet algoritmus a následně ho upravovat pro potřeby autonomního hraní hry Scotland Yard.

Z tohoto důvodu jsem se rozhodl zjednodušit stavový prostor. Nejprve je třeba zmenšit počet agentů. Ten jsem zmenšil z 5 agentů na 2 agenty. Díky tomuto, když započítáme oba agenty i Mr. X, je z aktuálního stavu v průměru $5^3 = 125$ možných stavů na kolo. Protože je tak málo agentů a Mr. X by v takovém počtu snad nikdy nechytili, je logické zjednodušit i adekvátně hrací plochu. Díky zjednodušení hrací plochy na mřížku o velikosti 5×5 políček, se znevýhodnění počtu agentů minimalizuje. Toto zmenšení a zjednodušení hrací plochy nám zajistí, že nejvíce možných následujících stavů z daného stavu za jedno kolo bude 4^3. Nejmenší stavový prostor bude $2^3 = 8$, jelikož počítáme, že každý hráč bude v rohu hrací plochy a tudíž bude mít pouze 2 možné tahy.

Tuto hru by sice měli hrát 3 hráči, ale pro zjednodušení implementace budeme počítat jen se 2 hráči. První hráč bude hrát za Mr. X a druhý hráč bude hrát za oba agenty. Toto

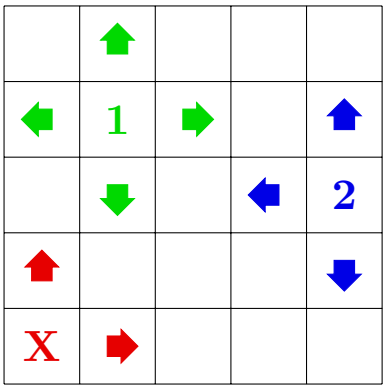
[1] pravidla vysvětlena v kapitole 3.3

zjednodušení umožní jednodušší implementaci algoritmů a použití algoritmů, které jsou určené primárně pro 2 hráče (jako je třeba algoritmus Alfa Beta).

Jedna z posledních úprav je zjednodušení pravidel pro tuto hru. Jelikož je hrací pole tak malé, způsob dopravy zredukujeme pouze na jeden, který je pro všechny stejný. Mr. X dále nemá žádné speciální tahy (jako je například dvojitý tah, kdy Mr. X táhne dvakrát za své kolo). Navíc pro usnadnění práce agentů je Mr. X viděn jednou za 3 kola (původně byl viděn přibližně jednou za 5 kol). Poslední úprava je délka hry, za kterou mají agenti Mr. X chytit. Délka je zmenšena z 22 kol na 15. Tato změna má naopak zvýhodnit Mr. X, jelikož nemusí unikat agentům tak dlouho na tak malém poli.

4.1.1 Shrnutí návrhu prostoru

Na obrázku 4.1 je vidět hrací plocha se dvěma agenty a Mr. X. Agenti jsou označeni čísly 1 a 2, Mr. X je označen písmenem X. Pohyb hráčů bude možný ve 4 směrech: nahoru, dolů, doleva a doprava. Šipky naznačují možný pohyb agentů a Mr. X. Agent 1 se může pohybovat ve čtyřech směrech. Agent 2 je na okraji, proto se může pohybovat jen na 3 políčka. Mr. X je v rohu a proto se může posunout jen na 2 políčka. Mr. X je navíc viděn jednou za 3 kola a jedna hra trvá maximálně 15 kol.



Obrázek 4.1: Herní plocha s možnými pohyby agentů a Mr. X

4.2 Původní řešení za použití Alfa a Beta řezů

Dle postupu A. Tulušáka [14] je využito metody Alfa a Beta řezů. Algoritmus Alfa a Beta řezů je ovšem nejvhodnější při hře 2 hráčů. Toho je dosaženo sloučením agentů do jednoho týmu, a Mr. X do druhého. V tomto případě se dá o týmu agentů uvažovat jako o jednom hráči, který v jednom svém tahu pohne s oběma figurkammi agentů. Tah agentů tak bude mít nanejvýš $4 * 4 = 16$ způsobů (4 tahy pro každého agenta), jak lze táhnout. Tah Mr. X bude mít nanejvýš 4 možnosti jak táhnout.

Agenti vidí Mr. X jednou za 3 tahy. To znamená, že agenti nebudou mít žádné nové informace. Algoritmus alfa-beta sice vypočítává tahy do určené hloubky, ale určuje jen následující tah, který by měl hráč vykonat. Z tohoto důvodu agenti počítají s nejlepším tahem protihráče, tzn. že následující 3 tahy, kdy Mr. X nejde vidět, agenti počítají s nejlepším tahem Mr. X. Proto je strom metody Alfa-beta generován do hloubky 3 kol a metoda je trochu upravena. Alfa-beta si v tomto případě generuje strom jednou za 3 tahy a ze stromu, který si ukládá pro ty 3 tahy, vybírá nejlepší tah pro agenty.

Pohyb Mr. X taktéž určuje metoda Alfa-beta. Narozdíl od agentů, můžeme použít normální proceduru Alfa-bety, která po každém tahu přepočítá nejlepší tah pro Mr. X. Mr. X má navíc jinou hodnotící funkci.

4.3 Návrh metody Monte Carlo tree search

Použití metody Monte Carlo tree search (MCTS) by mohlo být vhodné z důvodu náhodnosti simulací použité u základní metody MCTS. Kvůli téměř neznámému stavu Mr. X, lze jen odhadovat pozici Mr. X. Tuto metodu jsem navíc uvažoval kvůli úspěchu programu AlphaGo (blíže popsáno v kapitole 3.2.1) ve hře Go, který využívá právě algoritmu MCTS a neuronvé sítě. Rozhodl jsem se o otestování samotného algoritmu MCTS, jelikož jsem chtěl zjistit chování a zdatnost algoritmu MCTS v této hře. Algoritmus MCTS v této implementaci nebude používat neuronovou síť narozdíl od AlphaGo 3.2.1, které používá právě kombinaci neuronové sítě a algoritmu MCTS. MCTS by měl být navíc důvodně vhodný kvůli stavovému prostoru u původní hry Scotland Yard, který by byl u předchozí implementace metody alfa-beta velmi rozsáhlý, i když je hloubka stromu jen 3 kola.

Za použití MCTS se sice nestihne najít nejlepší možný tah, ale je možné najít relativně optimální tah během přiřazeného času. Toto se dá využít později při škálování návrhu, kde algoritmus Alfa-beta nejspíše nebude stačit a bude zabírat velké množství výpočetních prostředků. MCTS pro výběr uzlu bude používat tzv. UCT (*Upper Confidence Bound 1 applied to trees*) [13] vzorec, který balancuje exploration-exploitation trade-off a vypadá následovně:

$$UCT = X_j + 2 * C_p * \sqrt{\frac{2 * \ln n}{Visits}} \tag{4.1}$$

Kde:

- $X_j = \frac{win}{Visits}$, kde win je počet výher simulovaných v daném uzlu, $(0 \leq X_j \leq 1)$;
- $C_p = \frac{1}{\sqrt{2}}$;
- $n =$ kolikrát byl navštíven rodičovský uzel;
- $Visits =$ kolikrát byl navštíven daný uzel.

MCTS za těchto podmínek bude hrát proti Alfa-betě. Konkrétně MCTS bude nejdříve hrát za agenty proti Alfa-betě za Mr. X. Poté se mohou role vyměnit. Tímto se ověří zdatnost samotného algoritmu MCTS v této oblasti a funkčnost implementované Alfa-bety.

Kapitola 5

Implementace metody pro hraní her ve hře Scotland Yard

V této kapitole je popsána implemetace konkrétní metody pro hraní her na zjednodušené verzi hry, popsané v předchozí kapitole 4.1. Samotná implementace je uložena ve formě skriptů na paměťovém médiu popsaném v příloze A. Rozhodl jsem se pro využití algoritmu Monte Carlo tree search (MCTS). Algoritmus MCTS jsem nejdříve implementoval pro ovládání agentů. Pro pohyb Mr. X jsem se rozhodl použít již implementovanou metodu Alfa-beta [14]. Později jsem imlementoval algoritmus MCTS i pro Mr. X a využil jsem agenty implementované metodou Alfa-beta. Simulace dvou různých algoritmů ukáže, který z algoritmů pracuje v dané situaci lépe.

První část kapitoly se věnuje implementaci prostředí, logiky hry a propojení původního kódu Alfa-bety s novým prostředím. Druhá část je zaměřená na samotnou implementaci metody MCTS. Tato část obsahuje imlementaci, experimenty, problémy a výsledky použití metody MCTS. Třetí část se věnuje zvětšování hracího prostoru, přidávání agentů a úpravě algoritmů pro hraní této hry.

5.1 Vytvoření logiky hry

Kvůli funkcionalitě, přehlednosti a jednoduchosti implementace jsem se rozhodl použít programovací jazyk Python. K výběru tohoto jazyka přispěl i fakt, že původní algoritmus Alfa-bety pro hraní této verze hry je taktéž napsán v jazyce Python. Propojení kódů je díky tomuto mnohem jednodušší.

Pro vytvoření algoritmu je potřeba vytvořit logiku hry a prostředí, ve kterém se bude tato hra odehrávat. Navíc, kromě hraní dvou algoritmů proti sobě a sledování výsledků, je implementována možnost hrát za Mr. X i za agenty samostatně.

5.1.1 Prostředí

Pro funkčnost hry je třeba vytvořit prostředí, které by udržovalo informace o aktuálním stavu hry. Toto prostředí jsem implementoval pomocí klasické třídy v Pythonu. Tato třída obsahuje informace o stavu hry a metody pro manipulacemi s těmito informacemi. Informace o stavu hry jsou následující:

- velikost hracího pole - hrací pole je čtvercová souřadnicová síť s danou velikostí.
- aktuální pozice Mr. X - zapsána pomocí souřadnic.

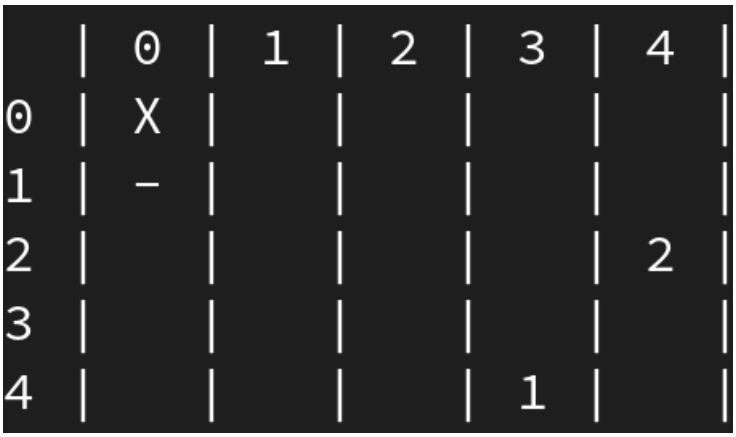
- seznam agentů - seznam aktuálních pozic agentů .
- poslední známá pozice Mr. X - místo, kde byl Mr. X naposledy zviditelněn.
- kdy byl Mr. X naposledy viděn - kolik kol uběho od posledního zviditelnění Mr. X.

Důležité metody pro třídu prostředí jsou:

- *reset* - nastaví velikost hracího pole, nastaví náhodně pozice agentů a Mr. X.
- *set* - nastaví pozice agentů a Mr. X na zadané hodnoty.
- *move* - provede tah daného hráče na novou pozci.
- *print* - vytiskne aktuální stav prostředí ve formě hrací plochy dle obrázku 5.1, kde pomyslné záhlaví v tomto obrázku tvoří čísla od 0 do 4, označující souřadnice. Pozice agentů jsou značeny čísly 1 a 2 v herním poli, pomlčkou je označená poslední známá poloha Mr. X a aktuální známá poloha Mr. X je označená písmenem X.

Třída navíc kontoluje, zda agenti chytili Mr. X.

Na obrázku 5.1 jsou vidět pozice agentů. Pozice agenta 1 je [4, 3], pozice agenta 2 je [2, 4] a aktuálí pozice Mr. X je [0, 0]. Pomlčka na pozici [1, 0] značí poslední známou pozici Mr. X.



Obrázek 5.1: Herní plocha

5.1.2 Logika

Postup pro jednu hru je následující:

1. Vytvoření třídy prostředí.
2. Nastavení prostředí pomocí její metody *reset*. Pozice agentů a Mr. X jsou náhodné.
3. Pro každé kolo hry:
   - 3.1 Vytiskni hrací pole a zkontroluj, zda nevyhrál Mr. X nebo agenti.
   - 3.2 Táhni s Mr. X, dle přednastaveného algoritmu.
   - 3.3 Vytiskni hrací pole a zkontroluj, zda nevyhrál Mr. X nebo agenti.

3.4 Táhni s agenty, dle přednastaveného algoritmu.

3.5 Inkrementuj počet kol, za který byl viděn Mr. X naposledy.

4. Vytiskni výsledky.

Pro více her je třeba pouze zopakovat postup podle počtu her. Při dalších iteracích je možné vynechat první bod postupu, jelikož je třída již vytvořená.

Pro tahy Mr. X nebo agentů je možné si vybrat z následujících možností:

- algoritmus Monte Carlo tree search - algoritmus implementovaný pro agenty i obdobně pro Mr. X
- algoritmus Alfa-beta - implementace agentů je lehce odlišná od implementace Mr. X
- hraní hráčem - možné hraní za Mr. X i za agenty, ovládání je možné pomocí kláves 'w', 'a', 's', 'd', kde 'w' je pohyb směrem nahoru, 's' je pohyb dolů, 'a' je pohyb doleva a 'd' je pohyb doprava
- náhodné generování tahů - Mr. X nebo agenti (nebo obojí) se budou po hrací ploše pohybovat náhodně a jejich tahy nemusí dávat smysl

5.1.3 Propojení metody Alfa-beta s logikou hry

Pro Alfa-betu sice již bylo vytvořeno prostředí a logika hry. Jenomže v této logice se nedalo jednoduše implementovat více typů algoritmu. Proto jsem se rozhodl pro implementaci vlastní logiky s cílem usnadnit si implementaci více algoritmů do stejné hry a propojit ji s algoritmem Alfa-beta.

Pro propojení jsem vytvořil spojovací funkci, která zapouzdřuje Alfa-betu. Zvlášť pro tahy Mr. X a zvlášť pro tahy agentů. Funkce překládá vstupy a výstupy pro Alfa-betu a kontroluje, zda výstup Alfa-bety je správný.

5.2 Použití metody Monte Carlo tree search

Jeden ze základních prvků mého algoritmu je vyhledávací strom. Základní stavební jednotkou stromu je uzel. Algoritmus začíná v kořenovém uzlu. Kořenový uzel označuje vlastně aktuální stav hry. Jeden uzel vyhledávacího stromu obsahuje odkaz na rodičovský uzel, seznam potomků uzlu a další informace, důležité pro běh algoritmu, jako například:

- jeden z možných tahů agentů
- jeden z možných tahů Mr.X
- celkový počet tahů, kolik zbývá do konce hry
- aktuální počet tahů, který udává, kolikátý je to tah v aktuálním uzlu
- kolikrát byl daný uzel navštíven
- ohodnocení daného uzlu, tj. odměna propagovaná ze simulací

Upravený algoritmus MCTS pro hraní zjednodušené verze hry Scotland Yard vypadá následnovně:

1. Algoritmus uloží aktuální pozici agentů a Mr. X jako kořenový uzel.
2. Dokud zbývá dostatek času pro algoritmus MCTS (explicitně zadán parametrem progemu nebo implicitně 1 sekunda):
   2.1 MCTS vybere uzel s nejlepším potenciálem - nejlepším UCB (*Upper Confidence Bound*).
   2.2 Pokud daný uzel je uzlem listovým a byl navštíven alespoň jednou, daný uzel expanduje a vybere první potomek daného uzlu.
   2.3 MCTS odsimuluje výsledek z vybraného uzlu pomocí simulace náhodných tahů.
   2.4 MCTS přičte výsledek simulace a zvýší počet navštívení uzlu ve vybraném uzlu a všech nad ním až ke kořenovému uzlu.
3. MCTS vybere uzel, následující za kořenovým (následující tah), který byl nejvíce navštíven.

5.2.1 Monte Carlo tree search pro pohyb agentů

Metodu Monte Carlo tree search jsem si vybral kvůli určité míře náhody. Množina polí, kde může být Mr. X po pár tazích od zviditelnění je rozsáhlá. Z tohoto důvodu jsem se rozhodl odhadnout aktuální pozici zcela náhodně, tzn. aktuální pozici Mr. X získáme provedením náhodných tahů od poslední známé pozice. Tuto aktuální pozici uložíme společně s aktuálními pozicemi agentů jako kořenový prvek vyhledávacího stromu. Poté je průběh podobný jako u normálního algoritmu MCTS, kdy se prochází vyhledávací strom a vybírá se uzel s největším potenciálem (*Upper Confidence Bound*).

U agentů je navíc rozdílné ohodnocení výsledků simulace. Výherní stav pro agenty znamená, že pozice jednoho z agentů a Mr. X jsou totožné nebo Mr. X nemá volné pole, na které by v dalším jeho tahu vstoupil. Tento výherní stav je ohodnocen 1 jako výhra. Prohra, tzn. Mr. X nebyl chycen do konce hry (15 kol), je ohodnocena hodnotou 0.

5.2.2 Monte Carlo tree search pro pohyb Mr. X

Jelikož Mr. X zná pozici všech agentů, nemusí se, narozdíl od předchozí kapitoly, žádné tahy simulovat. Aktuální pozice agentů i Mr. X se uloží do kořene uzlu a následuje stejný proces jako u verze s agenty. Jediný větší rozdíl je systém ohodnocení simulací. Konec hry, tj. Mr. X nebyl chycen, je opačné u Mr. X. To je ohodnoceno 1 (výhrou). Chycení Mr. X (agent i Mr. X mají stejné pozice nebo Mr. X se nemá kam pohnout) je ohodnoceno 0 (prohrou).

5.2.3 Průběh hry

Hra začíná náhodným vygenerováním pozic agentů a Mr. X tak, aby se žádná z pozic nepřekrývala. Následně se zobrazí hrací plocha s pozicemi hráčů. Agenti se mohou pohybovat libovolně po hrací ploše, každý vždy o jedno pole. Jediné omezení agentů jsou hranice hracího pole a podmínka, že nesmí stát oba agenti na stejném místě. Mr. X se může taktéž pohybovat o jedno pole a taktéž nesmí za hranici hracího pole. Mr. X by se neměl pohnout na pozici obsazenou agenty. Takový stav by znamenal jeho prohru.

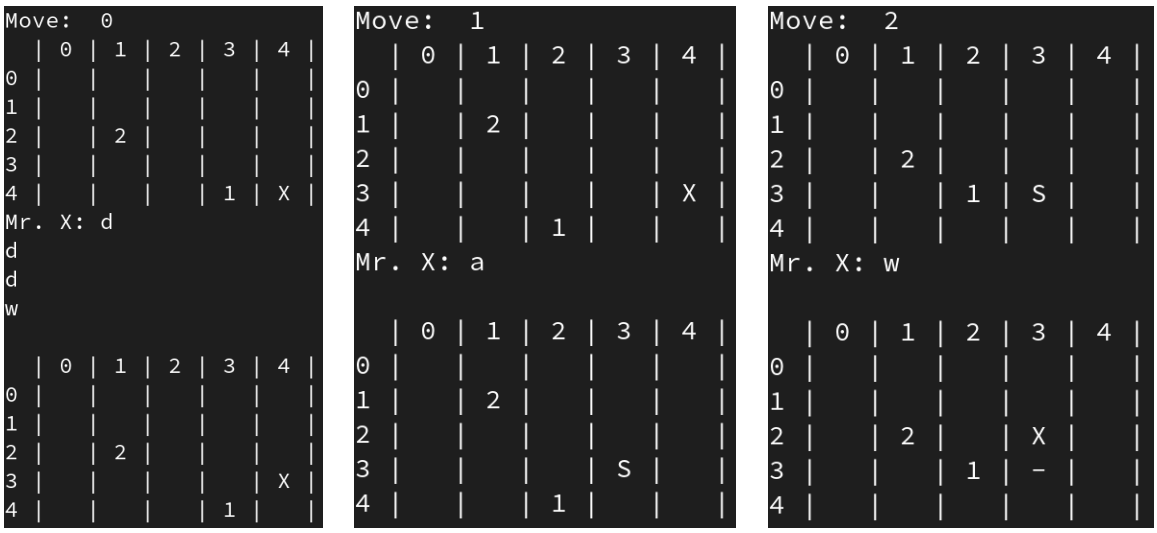
Jako první táhne Mr. X, kterého může ovládat hráč nebo se uplatní jeden z implementovaných algoritmů. Při tahu algoritmu Alfa-beta nebo algoritmu MCTS se vždy přepočítává

daným algoritmem další nejvhodnější tah. Pokud je zvolena možnost hraní za Mr. X hráčem, program čeká, dokud hráč nezadá validní vstup (klávesy 'w', 'a', 's' nebo 'd' možné podle situace[1]).

Jakmile program obdrží validní vstup ze strany hráče nebo algoritmu, je proveden odpovídající tah. Poté je opět zobrazena plocha s aktuální pozicí. Na obrázku 5.2 vlevo lze vidět úvodní stav hry a pokus o nevalidní vstup ze strany hráče. Jakmile hráč zadal validní vstup (klávesa 'w'), je vidět že značka Mr. X (písmeno 'X') se posunula z pozice [4, 4] na pozici [3, 4].

Následuje pohyb agentů. U Alfa-bety se generuje strom možných tahů jednou za 3 tahy a u algoritmu MCTS se tah generuje vždy při zavolání metody. Při ovládání agentů hráčem program vyzve k zadání tahů hráče, podobně jako u Mr. X. Takto program vyzve hráče k zadání tahu pro každého agenta. Na obrázku 5.2 vlevo a uprostřed jdou vidět tahy agentů pomocí algoritmu MCTS. Agent 1 se posunul z pozice [4, 3] na pozici [4, 2] a agent 2 se posunul z pozice [2, 1] na pozici [1, 1].

Z obrázku 5.2 uprostřed a vpravo lze vidět situaci, kdy poslední viděná pozice Mr. X a aktuální pozice Mr. X se překrývají. Tato pozice [3, 3] je označená písmenem 'S'. Poté, co se Mr. X posune do nové pozice, pozice [3, 3] se změní z písmena 'S' na pomlčku ('-'), což označuje poslední známou pozici Mr. X.



Obrázek 5.2: Ukázka prvních 3 tahů hry metody Monte Carlo tree search proti lidskému hráči

5.3 Experimenty s algoritmem Monte Carlo tree search

V kapitole 4.3 jsme si ukázali tzv. UCT (*Upper Confidence Bound 1 applied to trees*) [13] vzorec, který balancuje exploration-exploitation trade-off a který vypadá následovně:

$$UCT = X_j + 2 * C_p * \sqrt{\frac{2 * \ln n}{Visits}} \tag{5.1}$$

[1] pohyb kláves vysvětlen v kapitole 5.1.2

Tento vzorec se dá upravit na:

$$UCT = \frac{w}{Visits} + 2 * \sqrt{\frac{\ln n}{Visits}} \tag{5.2}$$

Kde:

- w = počet výher simulovaných v daném uzlu;
- n = kolikrát byl navštíven rodičovský uzel;
- $Visits$ = kolikrát byl navštíven daný uzel.

Za použití této rovnice a tohoto systému, jsem se rozhodl postavit můj algoritmus MCTS proti algoritmu Alfa-beta.

Alfa-beta je v tomto případě Mr. X. Za agenty hraje MCTS. Testovalo se nejdříve v menším měřítku, tj. 100 her. Jelikož se u MCTS dá nastavit doba po kterou bude algoritmus počítat jeden tah, nastavil jsem tuto hodnotu na 1 sekundu. Tato doba je o něco menší, než by uvažoval normální hráč a ve větším stavovém prostoru by musela být doba delší, ale z důvodu otestování algoritmu je tato doba optimální. U této velikosti hry však postačí 1 sekunda, kdy algoritmus v průměru udělá pár tisíc simulací hry než vrátí tah pro agenty (nebo Mr. X).

V tabulce 5.1 je znázorněna procentuální výhernost agentů u jednotlivých algoritmů. Z tabulky jde vidět, že Alfa-beta hrající za Mr. X porazila MCTS hrající za agenty v 78 případů ze 100. Když se situace obrátila (MCTS hraje za Mr. X a Alfa-beta hraje za agenty), Alfa-beta stále poráží MCTS v 88 případech ze 100. Když porovnáme Alfa-betu hrající za Mr. X s MCTS hrající za Mr. X, jsou výsledky proti Alfa-betě hrající za agenty velmi podobné.

Mr. X	Agenti	úspěšnost agentů (v %)
Alfa-beta	MCTS	22.4
MCTS	Alfa-beta	88.9
Alfa-beta	Alfa-beta	88.7
MCTS	MCTS	63.3

Tabulka 5.1: Výhernost agentů u jednotlivých algoritmů Alfa-beta a MCTS (údaje změřeny na 1000 iterací her)

5.3.1 Úprava systému odměn a rovnice pro výběr uzlu

Z důvodu nízké úspěšnosti agentů jsem se rozhodl upravit systém odměn a rovnici, kterou se vybírají uzly pro expanzi a simulaci. Zanalyzoval jsem tahy agentů řízenými algoritmem MCTS proti Mr. X řízeným algoritmem Alfa beta, jedna z mnoha her je na obrázku 5.3, kde je možné vidět poslední 4 tahy hry. Agent 1 táhne z pozice [3, 2] na pozici [4, 2] a následně zpět na pozici [3, 2]. Obdobně je na tom agent 2, který táhne z pozice [4, 1] na pozici [3, 1] a pak zase zpátky. Toto chování se opakuje do konce hry (který na obrázku nejde vidět). Z těchto tahů jde vidět, že agenti nejdou ani po poslední známé pozici Mr. X ale alternují mezi dvěmi stejnými pozicemi, i když by agenti mohli ještě chytit Mr. X. Proto jsem usoudil, že nejlepší bude do systému odměn nějakým způsobem započítat vzdálenost jednotlivých

agentů od Mr. X. Chování Mr. X, za kterého hraje Alfa-beta, je celkem očekávané. Mr. X čeká až se agenti přiblíží a čeká v co největší vzdálenosti od obou agentů.

Systém odměn v simulaci jsem nejdříve znásobil a odměna pro agenty za nalezení Mr. X tak byla 100. Dále jsem odečetl od výsledku počet tahů, ve kterém agenti chytili Mr. X. Když agenti nechytli Mr. X byli potrestáni zápornou hodnotou -1000. Jedna z posledních úprav, která mě napadla, je odečíst od výsledku vzdálenost agentů od Mr. X. Výslek simulace pak vrátil hodnotu x_i, která byla upravována podle následujících kritérií:

- Počáteční stav návratové hodnoty x_i je 0.
- Pokud agenti chytili Mr. X nebo Mr. X nemá kam jít, pak algoritmus přičte k hodnotě x_i výraz (100 - aktuální počet tahů v simulaci) $\times$ 50 a vrátí hodnotu x_i. Hodnota 100 a 50 jsou mnou zvolené konstany, které zaručí kladnost výsledku a vyváží do jisté míry trest v podobě vzdálenosti.
- Pokud Mr. X utekl odečte MCTS od hodnoty x_i 1000 a vrátí hodnotu x_i.
- V každém kroku simulace algoritmus odečte od hodnoty x_i vzdálenost každého agenta od Mr. X a to následujícím způsobem:
  a) Vzdálenost Mr. X a agenta je větší než počet zbývajících tahů (tj. agent nikdy nemůže Mr. X chytit), odečti od x_i hodnotu $(d + 10) \times 5$, kde d je vzdálenost agenta a Mr. X. Konstanty 10 a 5 mají v tomto případně znevýhodnit další postup z uzlu, jelikož minimálně 1 z agentů již nemůže chytit Mr. X.
  b) Vzdálenost Mr. X a agenta je menší než počet zbývajících tahů (tj. agent může chytit Mr. X), odečti od x_i hodnotu (d), kde d je vzdálenost agenta a Mr. X.

Výsledné hodnoty simulací se pak sčítají v jednotlivých uzlech spolu počtem průchodů uzlem při zpětné propagaci. Při tazích Mr. X se vzdálenost od agentů přičítá, jinak se hodnoty na konci simulací přičítají nebo odčítají obráceně, jak tomu je u agentů.

Rovnici

$$UCT = \frac{w}{Visits} + 2 * \sqrt{\frac{\ln n}{Visits}} \tag{5.3}$$

jsem musel upravit. Již nelze počítat pouze s poměrem vyhraných her, ale musíme počítat se součtem výsledků simulací. Proto jsem danou rovnici upravil do tvaru

$$UCB = X + 200 * \sqrt{\frac{\ln n}{V}} \tag{5.4}$$

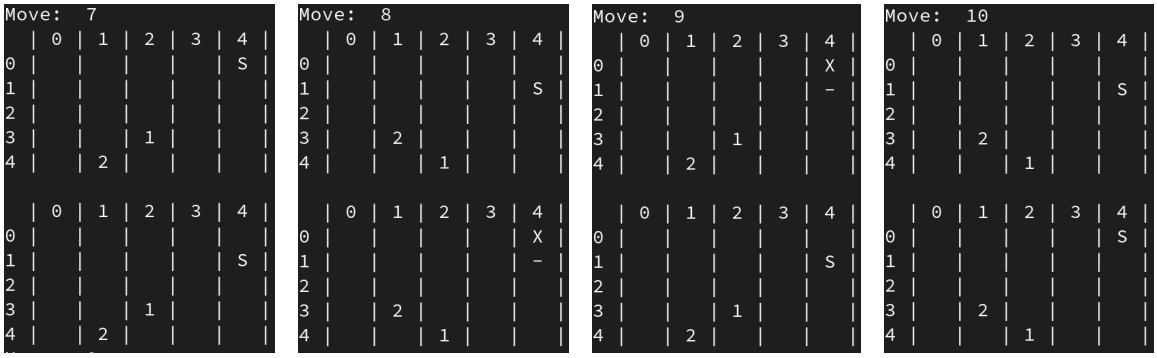
Kde UCT jsem zaměnil za UCB a kde:

- X = součet všech provedených simulací z daného uzlu;
- n = počet provedených simulací;
- V = kolikrát byl navštíven daný uzel.

S těmito parametry byly provedeny testy, jejichž výsledky lze vidět v tabulce 5.2. Z tabulky lze vidět zhoršení výsledků. Zhoršení je i u hraní MCTS jako Mr. X. Jelikož je úspěšnost agentů u vcelku nízká, zkusil jsem postavit MCTS proti algoritmu, který zahraje vždy náhodný tah. V těchto situacích MCTS v naprosté většině vyhrává. To, že je MCTS lepší než náhodné tahy naznačují také první a poslední řádek tabulky 5.2, kde agenti s náhodným výběrem tahů jsou výrazně horší proti Alfa-betě, než agenti táhnoucí pomocí algoritmu MCTS.

Mr. X	Agenti	úspěšnost agentů (v %)
Alfa-beta	MCTS	20.8
MCTS	Alfa-beta	91.0
Náhodný	MCTS	94.9
Alfa-beta	Náhodný	6.9

Tabulka 5.2: Výhernost agentů algoritmů Alfa-beta, upravenému MCTS a náhodném tahu (údaje změřeny na 1000 iterací her)



Obrázek 5.3: Ukázka 4 tahů hry metody MCTS jako agenti 1 a 2 proti metodě Alfa-beta jako Mr. X

5.3.2 Zvýšení času pro algoritmus Monte Carlo tree search

Rozhodl jsem se ještě zvýšit čas pro vypočtení tahu pro algoritmus MCTS. To by mohlo zvýšit úspěšnost algoritmu MCTS, jelikož algoritmus má více času pro simulaci více stavů. Čas jsem se rozhodl zvýšit z 1 sekundy na tah na 10 sekund na tah.

Z tabulky 5.3 lze vidět v prvních 2 řádcích upravenou verzi MCTS. Ta se s větším časovým úsekem ještě zhoršila. V posledních dvou řádcích lze vidět verze MCTS s původním vzorcem pro výpočet nejlepšího uzlu a původním systémem odměn, kdy se počítalo pouze počet výher z daného uzlu. Původní verze prokázala u agentů, řízených algoritmem MCTS, mírné zlepšení oproti testu, kdy algoritmus MCTS počítal pouze 1 sekundu. Taktéž je tomu u Mr. X hrající podle algoritmu MCTS, kde Mr. X vyhrál v 10 případech ze 100. Kdežto když MCTS počítalo jen 1 sekundu, Mr. X vyhrál v 8 případech ze 100. Toto zlepšení je ale málo patrné a výměna tolika času za tak malé zlepšení je celkově nevýhodná.

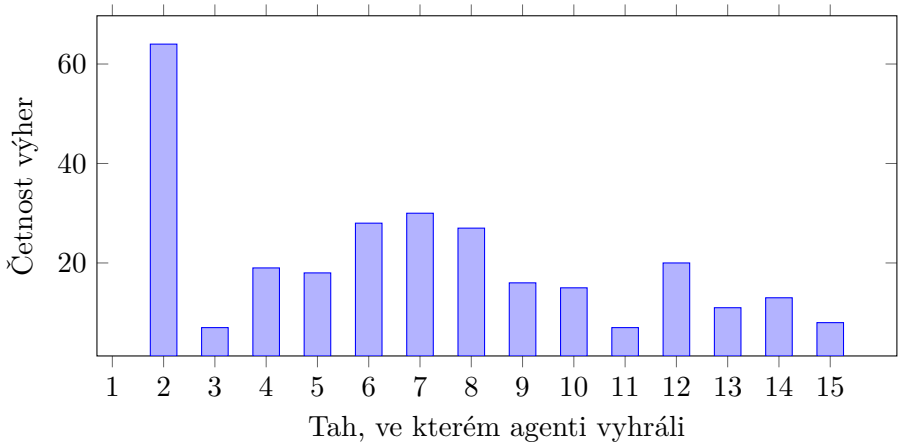
Mr. X	Agenti	úspěšnost agentů (v %)
Alfa-beta	MCTS (upravený)	13.0
MCTS (upravený)	Alfa-beta	93.0
Alfa-beta	MCTS (UCT)	27.0
MCTS (UCT)	Alfa-beta	90.0

Tabulka 5.3: Výhernost agentů algoritmu MCTS, který počítá 10 sekund 1 tah, proti algoritmu Alfa-beta (údaje změřeny na 100 iterací her)

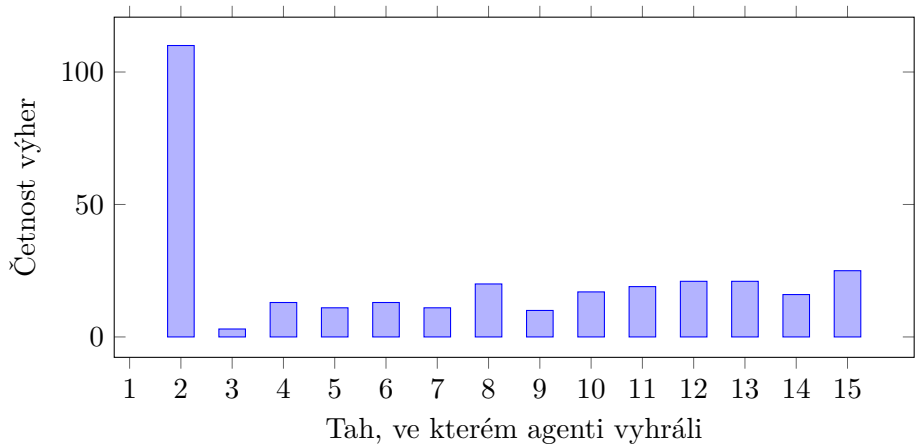
5.3.3 Vyhodnocení experimentů

Výsledky experimentů algoritmů Alfa-beta s MCTS vedly k úpravě algoritmu MCTS. Tyto úpravy měly snahu zlepšit chování MCTS, které se občas zaseklo. To je vidět na obrázku 5.3, kde agenti opakují své tahy. Opakování tahů měla omezit nová funkce pro výběr uzlů ve vyhledávacím stromu MCTS a systém odměn v simulaci MCTS. To se bohužel nepovedlo a MCTS samo o sobě nemůže konkurovat algoritmu Alfa-beta v takovém měřítku. Alfa-beta poráží MCTS jako Mr. X i při kontrole agentů. Prodloužení času stráveného MCTS na výběr tahu nijak výrazné nepomohlo zlepšení algoritmu MCTS.

Nezabrala ani změna funkce pro vyhledávání uzlu ze stromu MCTS. Na grafech 5.4 a 5.5 lze vidět ve kterém tahu agenti chytili Mr. X. Ve druhém tahu chytili Mr. X jen díky náhodě, jelikož Mr. X by neměl být ještě viděn. Původní algoritmus MCTS dle grafu 5.4 chytá Mr. X nejčastěji okolo tahu 7 a tahu 13. Upravená verze algoritmu MCTS chytá Mr. X mnohem rovnoměrněji.



Obrázek 5.4: Graf znázorňující četnost výher agentů ovládaných původním MCTS (řízeným podle počtu výher) proti algoritmu Alfa-beta



Obrázek 5.5: Graf znázorňující četnost výher agentů ovládaných upraveným MCTS (řízeným vzdáleností agentů od Mr. X) proti algoritmu Alfa-beta

5.4 Škálování návrhu

Alfa-beta nachází lepší tahy a obecně hraje lépe za agenty i za Mr. X. Proto jsem se rozhodl začít škálovat tuto zjednodušenou verzi hry Scotland Yard, na které jsme doposud hráli. Cílem bylo zjistit, jak si jednotlivé algoritmy povedou s větším herním prostorem, delší hrou nebo větším počtem agentů. Tímto počínáním jsem chtěl zjistit, jestli Alfa-beta nebo Monte Carlo tree search by mohli hrát hru Scotland Yard autonomně a s rozumnými prostředky a ne jen zjednodušení této hry.

Kvůli tomu jsem upravil program tak, aby bylo možné vyzkoušet hraní této hry.

5.4.1 Zvýšení počtu tahů

Zvýšení počtu tahů nemá na algoritmus Alfa-beta žádný vliv, jelikož Alfa-beta vypočítává tah Mr. X i agentů vždy do hloubky 3. Na algoritmus MCTS má změna délky hry minimální vliv. V simulaci hry algoritmu MCTS se musí jen změnit statická délka hry za dynamickou. Jinak vše zůstane stejné.

Zvýšení počtu tahů a tím prodloužení hry vede teoreticky k větší šanci chytit Mr. X. To znamená, že oba algoritmy Alfa-beta i MCTS by měli zvýšit své úspěšnosti v počtu chycení Mr. X.

Porovnáním údajů z tabulky 5.4 lze vyčíst zlepšení agentů u obou algoritmů. Nejlepší podíl zlepšení má však metoda MCTS pro agenty, jež hrála proti MCTS hrající za Mr. X. To je způsobeno tím, že Mr. X za kterého hraje MCTS je horší než Mr. X za kterého hraje Alfa-beta.

Mr. X	Agenti	úspěšnost agentů (v %)	
		pro 30 tahů	pro 15 tahů
Alfa-beta	MCTS	27.7	22.4
MCTS	Alfa-beta	92.7	88.9
Alfa-beta	Alfa-beta	94.0	88.7
MCTS	MCTS	91.3	63.3

Tabulka 5.4: Výhernost algoritmu MCTS a algoritmu Alfa-beta ve 30 a 15 tazích pro porovnání (údaje změřeny na 1000 iterací her)

5.4.2 Přidávání agentů

Pro přidání více agentů bylo potřeba i zvětšit hrací pole. V programu bylo třeba změnit statickou velikost pole na dynamickou. Tato velikost se dá potom získat ze třídy prostředí, která tuto hodnotu uchovává pro výpis hrací plochy. Pro fungování programu bylo ještě potřeba upravit algoritmus Alfa-bety A. Tulušáka [14], která obsahovala taktéž statickou velikost hrací plochy. Informace o velikosti hrací plochy se předává přes funkci, která komunikuje s Alfa-betou.

Pro přidání více agentů musela být přepsána funkce pro generování možných tahů agentů z aktuální pozice agentů. V původní funkci se nejdříve vygenerovaly seznamy možných nových pozic pro oba agenty a ty se potom spojovali tak, aby nebyli agenti na stejném místě. Navíc je potřeba zkontrolovat, zda agent 1 netáhl na pozici, která je okupována agentem 2. Jinak řečeno agent 1 by normálně táhnul jako první a nesmí tímto tahem skončit na aktuální pozici agenta 2. Pro více agentů je třeba tuto funkci upravit. Ve funkci

se postupně generuje seznam možných tahů pro jednotlivé agenty. Rozdíl oproti předchozí verzi je okamžitá kontrola, zda agent náhodou netáhne na aktuální pozici jiného agenta. Ve funkci se pak udělá nad tahy jednotlivých agentů kartézský součin[2], tj. množina (seznam) všech možných stavů, jak mohou táhnout agenti. Jeden stav obsahuje na prvním místě vždy možný tah (možnou pozici) prvního agenta, druhý prvek obsahuje druhou možnou pozici a tak dále. Na konec je potřeba odstranit stavy, kdy agenti skončí na stejném místě.

Je potřeba navíc upravit algoritmus Alfa-beta, jelikož hodnotící funkce počítá jen se dvěma agenty. Funkce vypadá následovně [14]: Pro každý uzel v cestě od kořenového ke koncovému algoritmus přičte hodnotu funkce $f_1(i)$, $f_2(i)$ resp. $f_3(i)$ a $f_4(i)$ a to následovně:

- Když je uzel výherní pro agenta 1, algoritmus použije funkci $f_1(i)$:

$$f_1(i) = (3 - h) * 50 - d(x, a_2) * h \tag{5.5}$$

- Když je uzel výherní pro agenta 2, algoritmus použije funkci $f_2(i)$:

$$f_2(i) = (3 - h) * 50 - d(x, a_1) * h \tag{5.6}$$

- Když uzel není výherní, algoritmus použije funkci $f_3(i)$ a $f_4(i)$:

$$f_3(i) = -min(d(x, a_1), d(x, a_2)) \tag{5.7}$$

$$f_4(i) = -max(d(x, a_1), d(x, a_2))/2 \tag{5.8}$$

Kde

- h je hloubka zanoření
- x je pozice Mr. X
- a_1, a_2 jsou pozice agentů
- $d(x, a)$ je vzdálenost agenta od Mr. X

Tuto funkci jsem upravil následujícím způsobem pro obecně n agentů:

- Když je uzel výherní pro agenta x, algoritmus použije funkci $f_a(a_i)$ a následně $f_1(i)$:

$$f_a(a_i) = (3 - h) * 50 - d(x, a_i) * h \tag{5.9}$$

$$f_1(i) = \frac{f_a(a_1) + f_a(a_2) + ... + f_a(a_x - 1) + f_a(a_x + 1) + ... + f_a(a_n)}{n - 1} \tag{5.10}$$

- Když uzel není výherní, algoritmus použije funkci $f_2(i)$ a $f_3(i)$:

$$f_2(i) = -min(d(x, a_1), d(x, a_2), ..., d(x, a_n)) \tag{5.11}$$

$$f_3(i) = \frac{-max(d(x, a_1), d(x, a_2), ..., d(x, a_n))}{n} \tag{5.12}$$

[2] Jsou dány množiny A a B. Kartézským součinem množiny A s množinou B rozumíme množinu všech uspořádaných dvojic takových, že první prvek uspořádané dvojice je prvkem množiny A a druhý prvek uspořádané dvojice je prvek množiny B. [https://portal.matematickabiologie.cz/index.php?pg=zaklady-informatiky-pro-biology–teoreticke-zaklady-informatiky–teorie-mnozin–kartezsky-soucin]

Kde

- h je hloubka zanoření
- x je pozice Mr. X
- a_i je pozice agenta
- $d(x, a_i)$ je vzdálenost daného agenta od Mr. X

ď Prakticky to znamená, že u výherního uzlu se udělá aritmetický průměr funkcí ostatních agentů (těch, kteří nechytili Mr. X).

Kvůli zvětšení počtu agentů je třeba adekvátně zvětšit hrací plochu. Rozhodl jsem se zvětšit hrací plochu na pole o velikost 7×7. Prodloužení hry nebylo třeba, jelikož se agenti ještě mohou za hru dostat z jednoho rohu do druhého.

Začal jsem se zvýšením agentů ze dvou na tři. Z tabulky 5.5 jde vidět, že agenti se ve všech případech zhoršili (v porovnání s tabulkou 5.1). Alfa-beta se zhoršila výrazně.

Alfa-beta se navíc zhoršila i časově. Jedna hra, která měla 14 kol (což je 28 tahů), algoritmu Alfa-bety proti Alfa-betě trvala 1 minutu a 22 sekund. Alfa beta potřebovala na tah skoro 3 sekundy (82 sekund / 28 tahů). Původní hra algoritmu Alfa-beta proti Alfa-betě, která trvala 15 kol při 2 agentech a původním hracím polem (5×5), trvala 0.7 sekund. Z čehož vyplývá, že na 1 tah Alfa-beta potřebovala méně než desetinu sekundy. Oproti MCTS, které v této verzi potřebuje stále jen 1 sekundu, je Alfa-beta v tomto ohledu horší. Alfa-beta nicméně pořád poráží MCTS o znatelný rozdíl ve výhernosti při hraní za agenty i za Mr. X.

Jelikož se Alfa-beta zhoršila časově více než desetinásobně přidáním jednoho agenta, rozhodl jsem se přidat ještě jednoho agenta. Kvůli tomu bylo nutné zvětšit hrací plochu, aby měl Mr. X stejnou šanci na útěk. Zvoliljsem hrací pole 15×15 polí. Kvůli velikosti pole je třeba prodloužit délku hry. Proto jsem prodloužil délku hry na 35 kol. Délku počítání jsem nechal u algoritmu MCTS stejnou. U algoritmu Alfa-beta již nebylo třeba nic měnit. Po spuštění hry MCTS hrající za agenty proti Alfa-betě hrající za Mr. X, si strom Alfa-bety začal postupně zabírat více než 16GB paměti RAM[3] počítače. Z nedostačujících prostředků (nejspíše málo paměti) pro běh algoritmu Alfa-beta byl test předčasně ukončen operačním systémem. Mezitím stihly ale proběhnout 2 hry, které trvaly více než 4 hodiny. Při hře trvající 35 kol, kde Alfa-beta hrála za Mr. X proti algoritmu MCTS hrající za agenty (měl 1 sekundu pro vykonání tahu), potřebuje Alfa-beta k vykonání tahu více než 3 minuty pro výpočet a expanzi všech možných stavů.

Pro představu stavový prostor v hloubce 3 u nejhoršího případu (každý agent i Mr. X má 4 možnosti jak táhnout) můžeme dopočítat následujícím způsobem (hrajeme za Mr. X):

- Po rozgenerování kořenového uzlu (aktuálního stavu), kdy je na řadě Mr. X, může jet jen 4 směry.
- Tyto 4 stavy roznásobíme počtem možných tahů agentů, kterých je v nejhorším případě $4^4 = 256$. Vznikne nám tedy $4 \times 256 = 1024$ nových stavů, což můžeme zjednodušit na 10^3. Toto je jen hloubka 1.

[3] též operační paměť, paměť RAM znamená Random Access Memory a je součást počítače, která umožňuje ukládat a získávat informace kdykoli během běhu různých procesů. [https://cs.you7behappy.com/hard-disk-vs-ram-1188]

- Následuje opět tah Mr. X a agentů, z čehož můžeme použít předchozích výsledků a říct, že počet těchto stavů je zaokrouhleně 10^3 nových stavů. Tyto stavy roznásobíme a dostaneme $10^3 \times 10^3 = 10^6$. Nyní jsme vygenerovali listové uzly stromu pro hloubku 2.

- Následuje hloubka 3, opět s tahem Mr. X a agentů, z čehož můžeme opět použít předchozích výsledků a říct, že počet těchto stavů je zaokrouhleně 10^3. Tyto stavy opět roznásobíme a dostaneme $10^6 \times 10^3 = 10^9$.

Z tohoto výpočtu můžeme říct, že v nejhorším případě je třeba vygenerovat 10^9 listových uzlů stromu, pro které ještě Alfa-beta musí vypočítat určité ohodnocení.

Rozhodl jsem se alespoň otestovat algoritmus MCTS proti metodě, která hraje vždy náhodné tahy. Z tabulky 5.6 lze vidět, že algoritmus MCTS je lepší než hraní s náhodnými tahy s agenty i Mr. X. MCTS vyhrálo přibližně v 48 případech ze 100, když algoritmus hrál za agenty. Náhodné tahy za agenty přinesly úspěšnost agnetům v 28 případech ze 100, když hráli proti Mr. X řízeného algoritmem MCTS. I když MCTS má jen 1 sekundu na výběr tahu, MCTS je stále lepší než zcela náhodné generování tahu.

Mr. X	Agenti	úspěšnost agentů (v %)
Alfa-beta	MCTS	15.3
MCTS	Alfa-beta	75.2
Alfa-beta	Alfa-beta	58.8
MCTS	MCTS	54.0

Tabulka 5.5: Výhernost agentů algoritmu MCTS proti algoritmu Alfa-beta pro 3 agenty (údaje změřeny na 1000 iterací her)

Mr. X	Agenti	úspěšnost agentů (v %)
MCTS	MCTS	48.5
Náhodný	MCTS	48.4
MCTS	Náhodný	26.7
Náhodný	Náhodný	38.6

Tabulka 5.6: Výhernost agentů algoritmu MCTS proti náhodným tahům pro 4 agenty (údaje změřeny na 1000 iterací her)

Kapitola 6

Závěr

Cílem této práce bylo navrhnout, implementovat a porovnat funkčnost metody s ostatními metodami, které již byly implementovány.

Splnit tento cíl se podařilo, ovšem výsledky implementované metody byly málo uspokojivé. Zvolená metoda je metoda Monte Carlo tree search. Tato metodu jsem postavil proti metodě Alfa-beta.

Nejprve jsem se seznámil s metodami, které jsou účinné v řešení určité úlohy. Následně jsem se seznámil s principy a pravidly tahových her. Zaměřil jsem se na pravidla hry Go a hry Scotland Yard. Hru Go jsem si vybral kvůli již úspěšně implementované metody pro hraní této hry autonomně. Domníval jsem se, že jedna z metod by mohla být vhodná pro realizaci algoritmu, který by hrál hru Scotland Yard autonomně.

Pro realizaci jsem zvolil metodu Monte Carlo tree search. Tato metoda je jedna z metod implementovaných u programu AlphaGo (kapitola 3.2.1) hrající tuto hru autonomně. Algoritmus pro hraní touto metodou jsem navrhl pro agenty i Mr. X. Navrhl jsem i úpravu algoritmu, která se ukázala hrát obdobně, jako u původní verze.

Před postavením implementované metody Monte Carlo tree search (MCTS) proti jiné metodě bylo třeba zjednodušit herní prostor a pravidla hry Scotland Yard. Při postavení metody MCTS proti již implementované metodě Alfa-beta se ukázalo, že metoda MCTS se při zjednodušení hry Scotland Yard nevyrovná algoritmu Alfa-beta ani jako Mr. X ani při hraní za agenty.

Z tohoto důvodu bylo třeba ověřit funkčnost Alfa-beta ve větším měřítku. Postupným zvětšováním velikosti hrací plochy, délky hry a počtu agentů jsem chtěl ověřit funkčnost metod MCTS a Alfa-beta. Alfa-beta stále dominovala i ve větším měřítku, jenomže při přidávání více agentů Alfa-beta začínala používat nadměrné množství prostředků. Kvůli paměťové náročnosti, časové náročnosti a nedostatečné kapacity vlastních prostředků, jsem musel testy s Alfa-betou ukončit. Stejné testy s MCTS se ukázaly z tohoto pohledu velmi příznivé. Metoda MCTS se dokáže časově omezit. Zahraje nejlepší tah v daném časovém intervalu.

Z toho vyplývá, že samotná metoda MCTS nestačí a je třeba ji využít jako podpůrnou metodu k jiné metodě. Podobně jako je tomu u programu AlphaGo, který využívá algoritmu MCTS spolu s neuronovými sítěmi.

Literatura

[1] , S. J. R. a Norvig, P. *Artificial Intelligence: A Modern Approach. 3rd ed.* 3. vyd. New Jersey: Pearson Education, Inc., 2010. ISBN 978-0-13-604259-4.

[2] Aguayo, C. *AlphaZero, a novel Reinforcement Learning Algorithm, in JavaScript* [online]. Listopad 2020 [cit. 2021-5-06]. Dostupné z: https://towardsdatascience.com/alphazero-a-novel-reinforcement-learning-algorithm-deployed-in-javascript-56018503ad18.

[3] *How to Play* [online]. Říjen 2017 [cit. 2021-5-03]. Dostupné z: https://www.britgo.org/intro/intro2.html.

[4] *Scotland Yard Game Rules* [online]. 2020 [cit. 2021-5-03]. Dostupné z: https://www.ultraboardgames.com/scotland-yard/game-rules.php.

[5] Choudhary, A. *Monte Carlo Tree Search Tutorial: DeepMind AlphaGo* [online]. Leden 2019 [cit. 2021-5-03]. Dostupné z: https://www.analyticsvidhya.com/blog/2019/01/monte-carlo-tree-search-introduction-algorithm-deepmind-alphago/.

[6] Education, I. C. *What are Neural Networks?* [online]. Srpen 2020 [cit. 2021-5-03]. Dostupné z: https://www.ibm.com/cloud/learn/neural-networks.

[7] Holcomb, S. D., Porter, W. K., Ault, S. V., Mao, G. a Wang, J. Overview on DeepMind and Its AlphaGo Zero AI. In: *Proceedings of the 2018 International Conference on Big Data and Education*. New York, NY, USA: Association for Computing Machinery, 2018, s. 67–71. ICBDE '18. DOI: 10.1145/3206157.3206174. ISBN 9781450363587. Dostupné z: https://doi.org/10.1145/3206157.3206174.

[8] Johnson, J. *What's a Deep Neural Network? Deep Nets Explained* [online]. Červenec 2020 [cit. 2021-5-09]. Dostupné z: https://www.bmc.com/blogs/deep-neural-network/.

[9] Mařík, V., Lažanský, J. a Štěpánková, O. *Umělá inteligence*. 1. vyd. Praha: Academia, 1993. ISBN 80-200-0496-3.

[10] Neřád, V. *Umělá inteligence pro hraní her*. Brno, 2012. Bakalářská práce. FIT VUT v Brně. Dostupné z: https://dspace.vutbr.cz/bitstream/handle/11012/55269/final-thesis.pdf?sequence=6&isAllowed=y.

[11] Roy, R. *ML: Monte Carlo Tree Search (MCTS)* [online]. 2019 [cit. 2021-5-03]. Dostupné z: https://www.geeksforgeeks.org/ml-monte-carlo-tree-search-mcts/.

[12] Silver, D., Schrittwieser, J. a al, K. S. et. Mastering the game of Go without human knowledge. *Nature*. 2017, č. 550, s. 354–359. DOI: 10.1038/nature24270.

[13] Theodoridis, A. a Chalkiadakis, G. Monte Carlo Tree Search for the Game of Diplomacy. In: *11th Hellenic Conference on Artificial Intelligence*. New York, NY, USA: Association for Computing Machinery, 2020, s. 16–25. SETN 2020. DOI: 10.1145/3411408.3411413. ISBN 9781450388788. Dostupné z: https://doi.org/10.1145/3411408.3411413.

[14] Tulušák, A. *Strategická desková hra s neurčitostí*. Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií.

[15] Zbořil, F. V. a Zbořil, F. *Základy umělé inteligence: Studijní opora*. Květen 2012.

Příloha A

Obsah přiloženého paměťového média

Přiložené paměťové médium obsahuje kromě samotné bakalářské práce soubory a skripty potřebné pro běh programu, který hraje zjednodušenou verzi hry Scotland Yard autonomně. Na paměťovém médiu ve složce `Python-3.9.5` je také program Python, potřebný pro chod skriptů. Pokyny pro instalaci a spuštění tohoto programu jsou uvedeny v souboru `README.rst`, umístěného ve složce `Python-3.9.5`. Ve složce `monte-carlo` se nachází soubor `README.md`, který obsahuje popis spouštění skriptu, argumentů skriptu a výčet náležitých souborů. Skript se spouští z příkazové řádky a argumenty ovlivňují chování skriptu.

Seznam souborů ve složce `monte-carlo`

Zde je uveden seznam skriptů potřebných pro fungování porgramu, který hraje zjednodušenou verzi hry Scotland Yard autonomně. Soubory převzaté a upravené jsou přejaty ze zdrojových kódů A. Tulušáka [14]. Tyto soubory jsou upraveny a místo upravení je okomentováno.

Převzaté a upravené soubory:

- alfa_beta.py
- alfa_beta_mrx.py
- alfa_beta_node.py
- environment.py

Vlastní soubory:

- README.md
- main.py
- mc_environment.py
- monte_carlo.py
- alfa_beta_wrapper.py
- player.py

Příloha B

Scotland Yard - Pravidla hry

Námět hry

Zloděj, zvaný Mr. X, se snaží setřást své pronásledovatele v Londýně. Utíká před nimi taxíkem, autobusem nebo metrem. Jen ten nejmoudřejší detektiv (agent) ho může chytit.

Mr. X se snaží až do konce hry o to, aby svým pronásledovatelům unikl a skryl místo svého pobytu. Agenti se naopak snaží správně si vyložit tahy Mr. X a vystopovat ho.

Cíl hry

Mr. X se snaží uniknout svým pronásledovatelům a zůstat neobjevený, dokud agenti nevyčerpají všechny své tahy. Agenti se snaží najít Mr. X tím, že se dostanou na zastávku, na které se právě nachází Mr. X.

Příprava

Hráči se dohodnou, kdo převezme roli Mr. X. (Tip: pro tuto roli jsou potřebné pevné nervy, proto by ji měl vzít zkušený hráč). Ostatní hráči převezmou roli agentů.

Mr. X obdrží:

- 1 bílou hrací figurku
- clonu pro Mr. X (zabrání, aby detektivové viděli, kam se Mr. X dívá)
- tabulku jízd s vloženým papírem
- tužku
- lístky:
  - 5x černý lístek
  - 2x dvojitý ťah

Každý agent obdrží:

- 1 barevnou herní figurku podle vlastní volby a příslušnou tabulku a lístky
- lístky, které položí na svoji tabulku s lístky:

- 4x metro
- 8x autobus
- 11x taxi

Startovní postavení

Aby jste si určili startovní pozice, roztřiďte startovací karty podle jejich zadních stran (D a X). Promíchejte zvlášť karty D a zvlášť karty X a položte je přikryté na stůl. Každý agent si vytáhne jednu startovací kartu s písmenem D na zadní straně a postaví svou hrací figurku na odpovídající stanici. Nyní si Mr. X skrytě vybere startovací kartu s písmenem X na zadní straně a podívá se na ni. Svoji figurku ale nepostaví na hrací desku.

Průběh hry

Hraje se až 22 kol. Jedno kolo se skládá z toho, že Mr. X nejprve vykoná svůj tah a nakonec v libovolném pořadí všichni agenti. K tomu musí Mr. X a každý agent změnit své postavení. Každý agent použije pro použitou trasu lístek, který přidá ze své tabulky lístků do všeobecných zásob. Mr. X si bere svoje lístky vždy ze všeobecných zásob.

Jak se táhne

Každé místo na hrací desce je stanice pro 1 až 3 dopravní prostředky (taxi, autobus, metro). Barvy stanic označují, jaké dopravní prostředky odtud vycházejí a kde zastavují. Aby jste mohli použít nějaký dopravní prostředek, musí stát hrací figurka na stanici pro tento dopravní prostředek (barva dopravního prostředku je zahrnutá v barvě stanice).

- *Taxíkem* (žlutá) se dostanete na každé místo na hrací desce. Trasa, kterou můžete projít, je ale krátká - můžete se posunout jen k dalšímu místu (podél žluté linky).
- *Autobus* (tyrkysová) jde jen z míst s tyrkysovým půlkruhem ve stanici - s ním můžete projít po autobusových linkách o něco delší trasy.
- *Metro* (červená) jde po červených linkách a může rychle projít velké vzdálenosti.

Na hrací desce je ale jen málo stanic metra (místa s červeným vnitřním čtvercem ve stanici). Hráč použije lístek odpovídající barvou a posune se k další stanici. Ušlou trasu můžete projít znovu zpět při dalším tahu. Všechny figurky se můžou přesouvat jen na volná místa. Jestliže Mr. X nemůže táhnout na žádné volné místo, prohrál. Jestliže agent táhne na místo, na kterém se nachází Mr. X, pak Mr. X taky prohrál. Agenti nesmí nikdy stát na stejném místě.

Hrací tahy Mr. X

Mr. X vykonává své tahy skrytě. Vybere si pro to tajně novou stanici, která je nějakou linkou přímo spojená s aktuální pozicí. Zapíše číslo nové stanice na další volné pole své tabulky jízd. Svůj zápis přikryje použitým lístkem. Agenti tedy vědí, jakým dopravním prostředkem se Mr. X přesouval, ale nevědí, kam. Když je při následujícím tahu hráče opět na řadě, táhne z této zaznamenané stanice dál k další stanici, kterou si vybere atd.

Hrací tahy agentů

Po té, co Mr. X ukončí svůj tah, příjdou na řadu v libovolném pořadí agenti. Protože agenti mají společný cíl, měli by se hráči dobře dohodnout na svých tazích při hře. Každý agent odevzdá do zásob svůj právě použitý lístek a přesune svou figurku na zvolenou stanici dopravního prostředku, který si vybere.

V průběhu hry platí:

- Agenti mají omezené zásoby lístků. Jestliže nemá agent již žádný lístek na jeden z dopravních prostředků, nemůže tento dopravní prostředek používat.
- Jestliže již agent nemá vůbec žádné lístky nebo už svými lístky nemůže táhnout, musí hru opustit.
- Agenti si mezi sebou nesmí vyměňovat lístky.
- Lístky agentů leží vždy viditelně, aby Mr. X mohl zjistit, jaké dopravní prostředky mají jeho pronásledovatelé ještě k dispozici.

Speciální tahy Mr. X

Prozrazení polohy Mr.X

Mr.X se musí ukázat v pravidelných intervalech, a to po svém 3., 8., 13, 18. a 24. tahu. Tahy, ve kterých se musí objevit, jsou označené na tabuli jízd zakroužkovanými čísly a větším polem na psaní. Stejně jako jindy vykoná Mr. X svůj zápis na tabulku jízd a položí na ni svůj lístek. Potom postaví svou hrací figurku na aktuální pozici. Nyní mají agenti velkou šanci Mr. X obklíčit nebo chytit. Není ale moc času, protože už při dalším tahu si Mr. X vezme svou hrací figurku z desky a zmizí.

Dvojiný tah

Jestliže hraje Mr. X s lístkem na dvojitý tah, může v tomto kole navštívit hned dvě místa a to v jakékoliv povolené kombinaci 2 dopravních prostředků. Zaznamená obě stanice na tabulce jízd (do 2 samostatných polí) a položí na ni 2 lístky použitých dopravních prostředků. Lístek na dvojitý tah se dostane ze hry. Jestliže je prvním místem místo, na kterém se musí Mr. X objevit, ukáže se tam, ale zmizí hned s druhým tahem. Protože se dvojitý tah hraje přímo za sebou jako 2 normální hrací tahy, nesmí Mr. X ani při prvním ani při druhém tahu vstoupit na pole, na kterém stojí agent. V jednom kole může Mr. X použít jen 1 lístek na dvojitý tah.

Černé lístky

Mr. X může místo normálního lístku použít černý lístek, který platí pro jakýkoliv dopravní prostředek. S černým lístkem může Mr. X (a jen Mr. X) taky použít trajekt (a tak jít např. z bodu 157 do bodu 115). Toto spojení (černé linky) není možné použít s žádným jiným lístkem. Jestliže Mr. X použije černý lístek, neobrží agenti žádné upozornění na to, jaký dopravní prostředek Mr. X mohl použít. Černé lístky se mohou používat také v rámci dvojitých tahů. Jako obvykle se odkladájí na tabulku jízd.

Konec hry

Agenti vyhrávají, jestliže:

- se v jakémkoliv okamžiku v průběhu hry ocitnou agent a Mr. X současně na stejné stanici. V tomto případě se musí Mr. X nechat vidět.

Mr. X vyhrává, jestliže:

- stihne do konce 22. kola hry projít Londýnem bez toho, aby ho agenti chytili. Kolo je ukončené až tehdy, kdy nakonec ještě jednou táhnou agenti.
- už žádný z agentů nemůže táhnout.