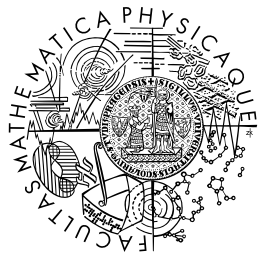


MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Lukáš Eigler

Umělá inteligence pro strategické hry s neúplnou informací

Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: Mgr. Marta Vomlelová, Ph.D.

Studijní program: Informatika

Studijní obor: Umělá inteligence Bc.

Praha 2024

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného titulu.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ....

Podpis autora

Chtěl bych poděkovat své vedoucí bakalářské práce Mgr. Martě Vomlelové, Ph.D. za odborné vedení, za pomoc a rady při zpracování této práce. Dále chci také poděkovat rodině a přítelkyni za neustálou podporu a pomoc.

Název práce: Umělá inteligence pro strategické hry s neúplnou informací

Autor: Lukáš Eigler

Katedra: Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: Mgr. Marta Vomlelová, Ph.D., Katedra teoretické informatiky a matematické logiky

Abstrakt: Tato bakalářská práce poskytuje úvod do teorie her a metod umělé inteligence, speciálně pro hry s neúplnou informací, a hratelnou aplikaci hry na motivy Scotland Yard (Fantom staré Prahy) na náhodně generovaných mapách s možností hrát s člověkem i umělými hráči. Kromě základních konceptů teorie her představuji dvě související bakalářské práce a v tuto chvíli nejsilnější a nejobecnější algoritmus Student of Games. Zaměřuji se na implementaci pokročilého algoritmu umělé inteligence. Práce nabízí dva různé umělé hráče. První využívá heuristiky založené na vlastnostech domény hry, druhý využívá ISMCTS. Hráči jsou v experimentech testováni proti sobě. Z výsledků vyplývá, že jsou výkonnostně různí, což poskytuje uživateli škálu různé síly AI protivníků.

Klíčová slova: Unity, hry s neúplnou informací, Monte Carlo Tree Search (MCTS), CFR, Student of Games, Scotland Yard

Title: Artifical Intelligence for games with incomplete information

Author: Lukáš Eigler

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: Mgr. Marta Vomlelová, Ph.D., Department of Theoretical Computer Science and Mathematical Logic

Abstract: This bachelor thesis provides an introduction to game theory and artificial intelligence methods, specifically for games with incomplete information, and a playable application of the game based on the Scotland Yard on randomly generated maps with the option to play with a human or artificial players. In addition to the basic concepts of game theory, I introduce two related bachelor theses and the currently strongest and most general algorithm, Student of Games. I focus on implementing an advanced artificial intelligence algorithm. The thesis offers two different AI players. The first utilizes heuristics based on the properties of the game domain, while the second utilizes ISMCTS. Players are tested against each other in experiments. The results show that they perform differently, providing the user with a range of AI opponent strengths.

Keywords: Unity, games with imperfect information, Monte Carlo Tree Search (MCTS), CFR, Student of Games, Scotland Yard

Obsah

Úvod **5**

1 Teorie her **7**
1.1 Hra v normální formě 7
1.1.1 Definice 7
1.1.2 Hra s nulovým součtem 8
1.2 Hra v rozšířené formě 9
1.2.1 Definice 9
1.2.2 Strom hry 10
1.2.3 Strategie 10
1.2.4 Pravděpodobnost dosažení 10
1.2.5 Očekávaná odměna 11
1.2.6 Podhra 11
1.3 Optimální strategie 11
1.3.1 Nejlepší odpověď 11
1.3.2 Nash. Ekvilibrium 12
1.3.3 ϵ-Nash. Ekvilibrium 12
1.3.4 Maximin 12
1.3.5 NE a Maximin 13
1.3.6 Regret 13
1.3.7 Regret Matching 14
1.3.8 Counterfactual Regret Minimization 14
1.4 Monte Carlo Tree Search 15
1.4.1 Selekce 15
1.4.2 Expanze 16
1.4.3 Simulace (Algoritmus 3) 16
1.4.4 Zpětná propagace (Algoritmus 4) 16
1.4.5 Závěr algoritmu 16
1.4.6 Pseudokód 17
1.4.7 MCTS a neúplná informace 19
1.4.8 Determinizace 19

2 Související práce **21**
2.1 Související bakalářské práce . . . 21
2.1.1 FIT Michal Sova . . . 21
2.1.2 MUNI Matej Rišňovský . . . 21
2.2 Student of Games . . . 22
2.2.1 Představení . . . 22
2.2.2 Neuronová síť . . . 22
2.2.3 CFR . . . 22
2.2.4 Regret Matching+ . . . 22
2.2.5 Trénování . . . 23

3 Návrh a implementace **25**
3.1 Pravidla hry . . . 25
3.1.1 Cíl hry . . . 25
3.2 Má implementace ISMCTS . . . 25
3.2.1 Vrchol . . . 26
3.2.2 Strom . . . 26
3.2.3 Monte Carlo Tree Search . . . 26
3.2.4 Stav hry . . . 26
3.2.5 Akce . . . 26
3.2.6 Popis hry . . . 27
3.2.7 Hráči . . . 27
3.2.8 Simulátor . . . 27
3.3 Implementace hry . . . 27
3.3.1 Unity . . . 27
3.3.2 Logika hry . . . 27
3.3.3 Pomocné skripty . . . 28
3.3.4 Fantom . . . 28
3.3.5 Detektivové . . . 28
3.3.6 Interface AI . . . 29
3.3.7 Mapa . . . 30
3.3.8 Uživatelské rozhraní . . . 31
3.4 Instalace hry . . . 35

4 Experimenty **37**
4.1 Druhy experimentů . . . 37
4.2 Použité mapy . . . 37
4.3 Náhodný proti heuristice . . . 40
4.4 Heuristika proti ISMCTS . . . 42
4.5 Možná vylepšení ISMCTS . . . 44
4.6 Možná vylepšení heuristiky . . . 44

4.7 Závěr experimentů 44

Závěr **45**

Seznam použité literatury **47**

A Programátorská dokumentace **49**
A.1 Graf 49
A.1.1 Vrchol 49
A.1.2 Hrana 49
A.2 Herní logika 49
A.3 Config 50
A.4 Logy 50
A.5 Ukládání map 50
A.6 Generování map 51
A.7 Hráč 51
A.7.1 AI 51
A.7.2 Reálný hráč 51
A.7.3 Struktura Move 51
A.7.4 Hledání hráčů 51
A.8 Vykreslování postav 52

B Uživatelská dokumentace **53**

Úvod

V roce 1997 poprvé dosáhla umělá inteligence DeepBlue II [1] výhry proti velmistrovi a světovému šampionovi v šachách Garry Kasparovovi. O rok dříve prohrál DeepBlue I proti stejnému oponentovi. DeepBlue II byl postavený na velkém paralelismu. Každou vteřinu ohodnocoval a zkoumal miliony stavů. Oproti DeepBlue I měl také komplexnější ohodnovací funkci pro stavy hry.

Umělí hráči můžou využívat různých metod s různou úspěšností, velmi často mají v nějaké formě zakomponované prohledávání stavového stromu hry. Obecně není možné celý stavový prostor hry uložit do paměti, proto tzv. online algoritmy hledají strategii vždy jen pro aktuální stav. Kvůli velikosti se také využívá metod na zkrácení prohledávané části stromu, např. heuristiky, neuronové sítě či Monte Carlo techniky.

V první kapitole se věnuji seznámení s teorií her, zejména konceptům řešení a informacím užitečným pro hry s neúplnou informací. Informace poskytují vhled do vytváření inteligentních agentů. Dále představuji Monte Carlo techniky a z nich odvozený Monte Carlo Tree Search (MCTS). Ukazuji problémy MCTS a naznačuji jejich řešení. Pro řešení problému neúplné informace popisuji algoritmus Information Set MCTS (ISMCTS).

Další kapitola se zabývá dvěma souvisejícími bakalářskými pracemi.Procházím jejich poznatky a přínosy. Mimo to představuji v tuto chvíli nejsilnější a nejobecnější algoritmus pro hry s neúplnou informací, Student of Games.

Ve třetí kapitole popisuji vývojové prostředí Unity a svou implementaci hry na motivy Scotland Yard (či české verze Fantom staré Prahy). Podrobně procházím strukturu programu. Má implementace umělé inteligence využívá Monte Carlo techniky v algoritmu Information Set Monte Carlo Tree Search (ISMCTS). Představuji i jiného hráče, který využívá pro rozhodování heuristiku vymyšlenou podle vlastností domény hry.

V poslední kapitole porovnávám v experimentech kvalitu hráčů. Pro experimenty jsem připravil tři různé mapy. Z výsledků vyplývá, že úroveň hráčů je různá. Díky různým úrovním poskytují uživateli škálu obtížností při volbě protihráče.

Kapitola 1

Teorie her

Teorie her se zabývá formálním pohledem na hry a hráče modelované strategiemi. Jako hlavní průkopník je považován John von Neumann. Společně s ekonomem Oskarem Morgensternem vydali v roce 1940 knihu Theory of Games and Economic Behavior [2].

Základní pojmy a definice teorie her jsou převzaté, a případně i upravené, ze skript k předmětu Algorithmic game theory (NDMI098) [3] na MFF, která jsou napsaná M. Balkem, a z dizertační práce Search in Imperfect Information Games M. Schmida [4].

1.1 Hra v normální formě

1.1.1 Definice

Hra v normální formě je trojice $(\mathcal{N}, \mathcal{A}, u)$, kde

- $\mathcal{N}$ je konečná množina n hráčů,
- A_i je množina akcí hráče $i \in \mathcal{N}$,
- $\mathcal{A} = A_1 \times ... \times A_n$ je profil akcí,
- $u = (u_1, ..., u_n)$, kde $u_i : \mathcal{A} \to \mathbb{R}$ je funkce odměn (utility/payoff function) pro hráče $i \in \mathcal{N}$ [3].

Strategie

Znakem $\Delta(A)$ značíme množinu všech pravděpodobnostních distribucí na množině A, tj. pro každé $p \in \Delta(A)$ platí, že $\forall a \in A : p(a) >= 0$ a $\sum_{a \in A} p(a) = 1$. Strategie hráče $i \in \mathcal{N}$ je $\pi_i \in \Delta(A_i)$ taková, že hráč i zahraje akci $a \in A_i$ s

$$\begin{bmatrix} (0,0) & (-1,1) & (1,-1) \\ (1,-1) & (0,0) & (-1,1) \\ (-1,1) & (1,-1) & (0,0) \end{bmatrix}$$

Obrázek 1.1 Matice odměn obou hráčů pro hru Kámen nůžky papír

$$\begin{bmatrix} 0 & -1 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 0 \end{bmatrix}$$

Obrázek 1.2 Zjednodušená matice odměn pro hru Kámen nůžky papír

pravděpodobností $\pi_i(a)$. Množina všech strategií hráče i je Π_i. Čistá strategie je strategie, kde hráč i vybere akci $a \in A_i$ s pravděpodobností 1, neboli $\pi_i(b) = 1$ pro $b = a$, ostatní s pravděpodobností 0. Taková strategie je deterministická. Smíšená strategie je strategie, kde π_i je pravděpodobnostní distribuce hráče i přes akce z A_i. Množina smíšených strategií je nadmnožinou množiny čistých strategií. [3]

Velmi užitečný koncept je tzv. support strategie π_i, formálně $supp(\pi_i) = \{a \in A_i | \pi_i(a) > 0\}$[4].

Profil strategií všech hráčů je $\pi = (\pi_1, ..., \pi_n)$, pak značíme π_{-i} profil strategií všech hráčů kromě i-tého.

Hru G dvou hráčů, neboli $G = (\{1, 2\}, \mathcal{A}, u)$, můžeme popsat maticí, příklad na Obrázku 1.1. Řádky matice znázorňují akce hráče 1 (A_1), a sloupce akce hráče 2 (A_2).

1.1.2 Hra s nulovým součtem

Hra s nulovým součtem (zero-sum game) je hra dvou hráčů, kde pro každou dvojici akcí $a = (a_1, a_2) \in (A_1, A_2)$ platí $u_1(a) = -u_2(a)$. Neboli zisk jednoho hráče je ztráta druhého.

Nejčastěji je hra reprezentovaná maticí M, příklad na Obrázku 1.2, kde $M_{i,j} = u_1(i, j)$ pro $i \in A_1, j \in A_2$, tedy utility matice hráče 1. Díky vlastnostem her s nulovým součtem je zřejmé, že utility matice hráče 2 je $-M$, kde $(-M)_{i,j} = u_2(i, j) = -u_1(i, j)$.

1.2 Hra v rozšířené formě

Často ve hrách hrají hráči postupně, oproti tomu u hry v normálním tvaru předpokládáme, že hráči hrají najednou. Hra v rozšířené formě (extensive form game) počítá s tím, že hráči se střídají. Hry se dají reprezentovat orientovaným stromem, kde vrcholy reprezentují stav hry. Z každého vrcholu vede tolik hran, kolik hráč na tahu má možných akcí z daného stavu. Hra začíná v kořenu a končí v listech. Listy mají přiřazené utility hodnoty.

Pokud hráči vždy znají celý aktuální stav hry, jedná se o hru s úplnou informací. Ve stromu jsou reprezentovány všechny možné stavy. Příkladem jsou hry go, šachy či tic-tac-toe. Naopak pokud hráči nemají všechny informace ohledně zahraných akcí ostatních hráčů, či jejich stavu, jedná se o hru s neúplnou informací, např. Poker, Scotland Yard. Hra se, kromě obecného stromu hry, dá reprezentovat stromy hráčů. Strom pro každého hráče obsahuje vrcholy reprezentující informační stav (information state) hráče - všechny možné stavy hry, které jsou z pohledu daného hráče stejné díky neznalosti všech informací, jsou sjednocené do jednoho. Ve stromě se kromě rozhodovacích vrcholů hráčů můžou nacházet i "chance" vrcholy, kde se s nějakou pravděpodobnostní distribucí vybere náhodná akce, např. rozdávání karet, hod kostkou [4].

Simultánní hry dvou hráčů, které se například dají reprezentovat jako hra v normální formě či maticí, se dají také reprezentovat pomocí herního stromu a popisu jako hra v extenzivní formě. Simultánnost je vyřešena rozdělením stavů stromu do informačních stavů, neboli nejdříve zahraje jeden hráč, ale druhý zjistí akci prvního hráče až po svém tahu.

Naštěstí se dá většina definic a konceptů pro hry v normální formě převést či rozšířit i pro hry v rozšířené formě.

1.2.1 Definice

Formálně se hra v rozšířené formě skládá z:

- $\mathcal{N}$ je množina n hráčů,
- $\mathcal{H}$ je množina možných posloupností akcí, neboli historií, prázdná posloupnost $\lambda \in \mathcal{H}$, historii h zřetězenou s akcí a značíme ha, pak každý prefix historie je historie, $ha \in \mathcal{H} \implies h \in \mathcal{H}$,
- $\mathcal{Z} \subset \mathcal{H}$ je množina terminálních historií, takových historií, které nejsou prefix žádné jiné historie,
- pro neterminální historii $h \in \mathcal{H} \setminus \mathcal{Z}$ je $\mathcal{A}(h) = \{a | ha \in \mathcal{H}\}$ množina možných akcí v historii h,

- $u = (u_1, ..., u_n)$, kde $u_i : \mathcal{Z} \to \mathbb{R}$ je utility funkce hráče i,
- funkce $p : \mathcal{H} \setminus \mathcal{Z} \to \mathcal{N} \cup \{c\}$, která každé neterminální historii udělí hráče na tahu nebo náhodu c (např. rozdávání karet),
- funkce f_c, která každé historii $h \in \mathcal{H}$ takové, že $p(h) = c$, přidělí pravděpodobnostní distribuci přes akce v dané historii $\mathcal{A}(h)$, $f_c(h) \in \Delta(\mathcal{A}(h))$,
- pro hráče $i \in \mathcal{N}$ je $\mathcal{I}_i$ informační rozdělení (information partition) historií $h \in \mathcal{H} : p(h) = i$, $I_i \in \mathcal{I}_i$ je informační stav (information state/set) hráče i, který obsahuje pro něj nerozeznatelné historie, také očividně $\forall I_i \in \mathcal{I}_i, \forall h, h' \in I_i : \mathcal{A}(h) = \mathcal{A}(h')$, proto možné akce v informačním stavu I_i značíme $\mathcal{A}(I_i)$. [5]

1.2.2 Strom hry

Strom hry pak obsahuje vrchol pro každou historii v množině všech historií $\mathcal{H}$ s tím, že terminální vrcholy jsou podle terminálních historií $\mathcal{Z}$. Pro každý vrchol daný historií h je daný hrající hráč $p(h)$, jeho možné akce $\mathcal{A}(h)$, akce $a \in \mathcal{A}(h)$ vede do nového vrcholu daného historií $ha \in \mathcal{H}$.

1.2.3 Strategie

Hra v rozšířené formě oproti hře v normální formě už není pouze jednotahová. Hráči se v hraní střídají, hrající hráč je určený funkcí p. Strategie hráče už také pouze neudává pravděpodobnostní rozdělení akcí hráče, ale vrací pravděpodobnostní rozdělení možných akcí v každém informačním stavu. Formálněji je strategie hráče i takové π_i, splňující $\pi_i : I \in \mathcal{I}_i \to \Delta(\mathcal{A}(I))$.

1.2.4 Pravděpodobnost dosažení

Nechť $P^\pi(h)$ je pravděpodobnost, že historie $h \in \mathcal{H}$ nastane, pokud hráči hrají podle π. Pravděpodobnost můžeme rozložit zvlášť na příspěvek každého hráče, $P^\pi(h) = \prod_{i \in \mathcal{N} \cup \{c\}} P_i^\pi(h)$, $P_i^\pi(h)$ pak říká pravděpodobnost, že hráč i zahraje všechny akce a takové, že $h'a$ je vlastní prefix h, kde $p(h') = i$. Nechť P_{-i}^π je součin pravděpodobností všech hráčů kromě i-tého. Pro informační set $I \subseteq \mathcal{H}$, nechť $P^\pi(I) = \sum_{h \in I} P^\pi(h)$ jako pravděpodobnost dosažení informačního setu I při strategickém profilu π [5].

1.2.5 Očekávaná odměna

Pro výpočet očekávané odměny pro hráče i potřebujeme znát strategický profil π (strategie všech hráčů). Očekávaná odměna pak bude součet odměn v terminálním stavu vynásobených pravděpodobností jejich dosažení [5], $u_i(\pi) = \sum_{z \in \mathcal{Z}} P^{\pi}(z) u_i(z)$.

1.2.6 Podhra

Podhra je podproblém celkového problému - hry. Ve hrách s úplnou informací stačí pro řešení podproblému pouze podstrom zakořeněný ve vnitřním vrcholu herního stromu. Samostatný podstrom obsahuje všechny potřebné informace - stav a budoucí stavy.

Oproti tomu v hrách s neúplnou informací to není tak jednoduché. Podhra je identifikovaná veřejným stavem a distribucemi přes vrcholy v informačních stavech všech hráčů v daném veřejném stavu [4].

1.3 Optimální strategie

Při hledání strategií pro hráče se používají buď offline algoritmy nebo online algoritmy. Offline algoritmy spočívají v tom dopředu zjistit strategii a uložit ji tak, aby pro aktuální stav bylo možné získat strategii. Většinou se využívá nějaká forma tabulky, případně implicitní reprezentace. V zásadě jde při získání optimální strategie o vyřešení hry. U složitějších a větších her není možné uložit strategii do tabulky. Online algoritmy proto počítají strategii při hře. Jedna z možností je využití prohledávání herního stromu.

1.3.1 Nejlepší odpověď

Mějme hru a předpokládejme, že strategie hráče i je fixní. Pak nejlepší odpověď (best response) na strategii π_i je π_{-i} taková, že maximalizuje utility hodnotu u_{-i}.

Formálně je nejlepší odpověď na strategii π_i hráče i smíšená strategie, $BR(\pi_i) \overset{def}{=} \operatorname{argmax}_{\pi_{-i} \in \Pi_{-i}} u_{-i}(\pi_i, \pi_{-i})$. [4]

Pro hry dvou hráčů s nulovým součtem je ekvivalentní, když oponent maximalizuje svůj zisk a minimalizuje náš, tedy $\operatorname{argmax}_{\pi_{-i}} u_{-i}(\pi_i, \pi_{-i}) = \operatorname{argmin}_{\pi_{-i}} u_i(\pi_i, \pi_{-i})$. Z této vlastnosti vyplývá, že utility hodnota strategie hrající proti kterékoliv nejlepší odpovědi je unikátní. Hodnota nejlepší odpovědi (best response value) je definovaná jako $BRV_i(\pi_i) = \min_{\pi_{-i}} u_i(\pi_i, \pi_{-i}) = -\max_{\pi_{-i}} u_{-i}(\pi_i, \pi_{-i}) = u_i(\pi_i, BR(\pi_i))$. [4]

Vždy existuje deterministická nejlepší odpověď na strategii [3]. Výpočet pro hry s neúplnou informací je ale složitější než pro hry s úplnou informací. Kvůli

existenci informačních stavů (nerozlišitelných historií pro hráče) nestačí pouze projít zbytek stromu, ale je potřeba do toho počítat i předchozí tahy.

1.3.2 Nash. Ekvilibrium

Nashovo equilibrium (NE) je profil strategií π takový, že $\forall i \in \mathcal{N} : \pi_i$ je nejlepší odpověď na π_{-i}. Slovně řečeno, žádný racionální hráč (sám o sobě) nechce změnit strategii, ve smyslu, že si jinou strategií nemůže pomoci, pokud ostatní hráče zafixujeme a budou stále hrát tu svojí zvolenou strategii. Strategický profil π je NE právě, když $\forall i \in \mathcal{N}, \forall \pi'_i \in \Pi_i : u_i(\pi, \pi_{-i}) \geq u_i(\pi'_i, \pi_{-i})$. Případně je možné NE definovat pomocí nejlepších odpovědí, strategický profil π je NE právě, když $\forall i \in \mathcal{N} : \pi_i$ je nejlepší odpovědí na π_{-i}. [4]

Podle Nashovy věty má každá konečná hra v normální formě s konečným počtem hráčů Nashovo ekvilibrium [3].

1.3.3 ϵ-Nash. Ekvilibrium

U složitějších her není vždy jednoduché najít Nashovo ekvilibrium. Využívá se proto například méně restriktivní pojem ϵ-Nashovo ekvilibrium. Říká, že hráč i při znalosti strategií ostatních hráčů π_{-i} si nemůže pomoci změnou své strategie π_i o více než ϵ. Neboli formálněji je strategický profil π ϵ-Nashovo ekvilibrium právě když, $\forall i \in \mathcal{N}, \forall \pi'_i \in \Pi_i : u_i(\pi_i, \pi_{-i}) \geq u_i(\pi'_i, \pi_{-i}) - \epsilon$. [3]

1.3.4 Maximin

Jednou z možných optimálních strategií je uvažování protihráče, který reaguje nejlepší odpovědí, v hrách s nulovým součtem to zároveň znamená pro nás nejhorší možností. Optimalizování, neboli hledání nejlepší strategie, proti takovému hráči je tzv. Maximin. Proti Nashově ekvilibriu je maximin definovaný pro strategii hráče, ne pro strategický profil všech hráčů. Formálně je Maximin strategie hráče i,

$$\operatorname*{argmax}_{\pi_i \in \Pi_i} \min_{\pi_{-i} \in \Pi_{-i}} u_i(\pi_i, \pi_{-i}) = \operatorname*{argmax}_{\pi_i \in \Pi_i} BRV_i(\pi_i). \tag{1.1}$$

Hodnota maximin strategie je $\underline{v_i} = max_{\pi_i \in \Pi_i} BRV_i(\pi_i)$. Množina všech maximin strategií pro hráče i je $\mathbb{MAXIMIN}_i = \{\pi_i | BRV_i(\pi_i) = \underline{v_i}\}$.[4]

Minimax theorem

V hrách s nulovým součtem platí: [4]

$$\max_{\pi_i} \min_{\pi_{-i}} u_i(\pi_i, \pi_{-i}) = \min_{\pi_{-i}} \max_{\pi_i} u_i(\pi_i, \pi_{-i}) \tag{1.2}$$

Je to jedna ze základních vět teorie her, byla dokázána John von Neumannem v roce 1928 [6].

1.3.5 NE a Maximin

Nashovo ekvilibrium i Maximin jsou pro hry s nulovým součtem dvou hráčů zajímavé, protože:

(π_1, π_2) je NE $\Leftrightarrow \pi_1 \in \mathbb{MAXIMIN}_1 \wedge \pi_2 \in \mathbb{MAXIMIN}_2$ [4]

Pro hru s nulovým součtem dvou hráčů má využití optimální strategie dobré důvody:

(π_i, π_{-i}) NE : $\forall \pi'_i, \pi'_{-i} : u_i(\pi_i, \pi'_{-i}) + u_{-i}(\pi'_i, \pi_{-i}) \geq 0$,

neboli při použití optimální strategie nemůže hráč prohrát, maximálně remizovat při hře proti optimálnímu hráči. [4]

1.3.6 Regret

Následující tři odstavce používají terminologii článku Regret Minimization in Games with Incomplete Information [5]. Koncept regret (”výčitky”) se využívá v aktuálně velmi úspěšných algoritmech řešení her s neúplnou informací [7]. Regret minimalizace je algoritmus založený na učení. Agent opakovaně volí akce proti protihráči či prostředí. Regret měří rozdíl mezi nastřádaným ziskem agenta proti možnému zisku při hraní nejlepší akce (pro každé opakování). Algoritmus zaručuje dobré vlastnosti i proti nepřátelskému prostředí, které rozhoduje o své akci až po agentovi [3].

Existuje několik druhů regretu. Tak zvaný external regret porovnává aktuální výkon agenta proti jedné nejlepší akci, kterou mohl agent hrát pokaždé. Pak se například používá internal regret, který zkouší strategii agenta pozměnit v právě jedné akci a porovnává kvalitu agenta proti nejlepší takové strategii [3].

Mějme hru v normální formě G. Algoritmus běží v $T \in \mathbb{N}$ iteracích . Budeme uvažovat external regret. Agent (hráč i) si opakovaně volí strategii v každém čase, pak π_i^t je zvolená strategie v čase t. Agent po zahrání dle své strategie dostane odměnu $x^t = (x^t_{a_1}, \ldots) \in \mathcal{R}^{|A_i|}$. Ohodnocení strategie je pouze vážená suma odměn dle pravděpodobností akcí: $v^t = \sum_{a \in A_i} \pi_i^t(a) x_a^t$. Celková odměna do času T je $X^T_{\pi_i} = \sum_{t=1}^T \sum_{a \in A_i} \pi_i^t(a) x_a^t = \sum_{t=1}^T v^t$. External regret akce $a \in A_i$ do času T představuje výčitky nehraní akce a v každém čase $t \leq T$, neboli $R_a^T = \sum_{t=1}^T x_a^t - X^T_{\pi_i}$. Celkový external regret představuje, jak moc litujeme, že jsme nezahráli nejlepší jednotlivou akci, $R^T = \max_{a \in A_i} R_a^T$.

Regret má skvělé teoretické vlastnosti. Průměrný regret je propojený s kvalitou průměrné strategie. Následujícím algoritmem zajistíme, že regret roste pomaleji než lineárně, průměrný regret vůči času se bude blížit k 0. Strategie obou hráčů

při dodržování algoritmu budou spadat do ϵ-Nash. ekvilibria a konvergovat k Nash. ekvilibriu.

1.3.7 Regret Matching

Jeden užitečný algoritmus využívající konceptu regret je regret matching. Hráč si zvolí strategii, kde každou akci hraje proporcionálně podle pozitivního regretu dané akce. Nechť $R_a^{T+} = max(R_a^T, 0)$ pro $a \in A$, R_a^T je akumulovaný regret pro akci a. Strategie hráče pak je $\pi(a) = \frac{R_a^{T+}}{\sum_{a' \in A} R_{a'}^{T+}}$. Pokud je jmenovatel rovný nule, $\sum_{a' \in A} R_{a'}^{T+} = 0$, pak se za strategii zvolí uniformní strategie přes všechny akce hráče [4].

1.3.8 Counterfactual Regret Minimization

Informace v tomto odstavci jsou převzané z článku Regret Minimization in Games with Incomplete Information [5]. Pravděpodobnosti dosažení využité v rovnicích níže jsou definovány v sekci 1.2.4.

Vysvětlení konceptu regret bylo na hře dvou hráčů v normální formě a kvalita aktuální strategie byla srovnávána s čistými strategiemi. Značná část her je ale sekvenční a vícetahová, ne simultánní. Counterfactual regret minimization (CFR) rozdělí regret do menších částí podle informačních stavů na tzv. counterfactual regret, který se dá minimalizovat samostatně po každém informačním stavu. Tato vlastnost platí díky omezení shora na celkový regret sumou counterfactual regretů.

Mějme hráče $i \in \mathcal{N}$ a informační stav $I \in \mathcal{I}_i$. Nechť $u_i(\pi, h)$ je předpokládaná utility hodnota pro danou historii $h \in \mathcal{H}$ a strategický profil π. Nechť $u_i(\pi, I)$ je counterfactual utility hodnota, neboli předpokládaná utility hodnota při dosažení I a každý hráč hraje podle strategického profilu π kromě i-tého, který hraje tak, aby dosáhl informačního stavu I. Formálně, pokud $P^\pi(h, h')$ je pravděpodobnost dosažení historie h' z historie h pomocí strategického profilu π, tak

$$u_i(\pi, I) = \frac{\sum_{h \in I, h' \in \mathcal{Z}} P_{-i}^\pi(h) P^\pi(h, h') u_i(h')}{P^\pi(I)}. \tag{1.3}$$

Pro počítání výčitků nehraní nějaké akce, nechť pro každé $a \in A(I)$ je $\pi|_{I \to a}$ strategický profil identický k π, ale hráč i vždy volí akci a v informačním stavu I. Counterfactual regret pro informační stav I a akci a pak je

$$R_i^T(I, a) = \frac{1}{T} \sum_{t=1}^{T} P_{-i}^{\pi^t}(I)(u_i(\pi^t|_{I \to a}, I) - u_i(\pi^t, I)). \tag{1.4}$$

Celkový counterfactual regret pro informační stav I je pouze maximální přes akce,

$$R_i^T(I) = \max_{a \in A(I)} R_i^T(I, a). \tag{1.5}$$

Nejužitečnější je pozitivní část counterfactual regretu, $R_i^T(I)^+ = \max(R_i^T(I), 0)$

Regret Matching v CFR

Pro každý informační stav $I \in \mathcal{I}_i$ a akci $a \in \mathcal{A}(I)$ si můžeme udržovat $R_i^T(I, a)$. Zvolená strategie v informačním stavu I v čase $T + 1$ je

$$\pi_i^{T+1}(I, a) = \frac{R_i^T(I, a)^+}{\sum_{a \in A(I)} R_i^T(I, a)^+}. \tag{1.6}$$

Při nulovém jmenovali se zvolí uniformní strategie pro daný informační stav.

1.4 Monte Carlo Tree Search

Informace ohledně Monte Carlo technik a samostatného Monte Carlo Tree Search pochazí z dizertační práce Monte Carlo Tree Search for Multi-Player Games [8]. V případech, kdy není možné kvůli velikosti sestrojit herní strom pro celý průběh hry, se dají využít hloubkově omezené stromy s listy ohodnocenými ohodnocovací funkcí. Funkce je připravená podle znalostí z dané domény hry. Vymyšlení správné funkce, případně její výpočet, může být velmi náročné.

Techniky založené na Monte Carlo přístupu místo ohodnocovacích funkcí pro daný list simulují průběhy her. Při simulaci se vybírají akce do doby než hra skončí a je možné získat ohodnocení. Mnohonásobným simulováním se odhaduje ohodnocení listu. Metrika pro ohodnocení může být například procento výher.

Monte Carlo Tree Search (MCTS) je technika, která postupně buduje strom pomocí opakovaných simulací. Díky využití Monte Carlo technik není potřeba ohodnocovací funkce.

Jedna iterace základního algoritmu (Algoritmus 1) se skládá ze čtyř částí: selekce, expanze, simulace a zpětná propagace. Algoritmus běží stanovený počet iterací nebo stanovenou délku běhu.

1.4.1 Selekce

V každé iteraci se z kořene stavěného stromu prochází do listu. Průchod je prováděn podle selekční strategie, která pro každý vrchol vybere jednoho z jeho potomků. Volba selekční strategie určuje v jakém poměru se využívají znalosti (exploitace) a prohledávání prostoru (explorace). Nejčastěji používaná strategie

je UCT (Upper Confidence Bound 1 applied to trees) inspirovaná UCB1 (upper confidence bound, version 1). Využití UCB1 je pro řešení problému multi-armed bandit.

$$v_c = \overline{x_c} + C \times \sqrt{\frac{\log n_p}{n_c}} \tag{1.7}$$

Formule 1.7 z [9] popisuje hodnotu pro potomka c vrcholu p, kde $\overline{x_c}$ je výhernost vrcholu c a n_p, respektive n_c, je počet průchodů skrz p, respektive c. Explorační parametr pro určení poměru exploitace a explorace je C, velmi často se využívá $C = \sqrt{2}$ vycházející přímo z UCB1. Při průchodu se v každém procházeném vrcholu vybere potomek i s maximální hodnotou v_i z možných potomků, dokud se nenarazí na vrchol, který není celý rozvinutý (v budovaném stromu nejsou všichni jeho potomci).

1.4.2 Expanze

Fáze expanze slouží rozrůstání budovaného stromu hry. Nejčastěji se v každé iteraci přidá jeden nový potomek k poslednímu dosaženému vrcholu v selekci. Existují i jiné přístupy, kde se například přidá nový potomek teprve každých n iterací, případně více potomků zároveň.

Selekce a expanze jsou shrnuty v Algoritmu 2.

1.4.3 Simulace (Algoritmus 3)

Z nově přidaného vrcholu se dohraje zbytek hry (rollout/playout) podle zvolené strategie. Často se používá náhodná - uniformní strategie přes možné akce pro každý stav. Výhodou je, že nevyžaduje žádné znalosti domény. Při zakomponování vlastností domény lze dosáhnout kvalitnějších simulací výměnou za složitější výpočet, což může snížit počet iterací, je-li běh časově omezený.

1.4.4 Zpětná propagace (Algoritmus 4)

Simulace končí při dosažení listu stromu hry a získání odměny. Odměna je poté propagovaná podél zvolených vrcholů. Ve vrcholech se ukládá výhra či prohra, případně i remíza, a aktualizuje se výhernost vrcholů a počet navštívení.

1.4.5 Závěr algoritmu

Tyto čtyři fáze se opakují dokud nevyprší čas nebo předem daný počet iterací.

Výběr nejlepší akce se dá uskutečnit výběrem podle výhernosti, počtu navštívení, či jejich kombinací.

1.4.6 Pseudokód

Algoritmy ze sekce 1.4 můžeme shrnout v následujících pseudokódech.

Algoritmus 1 Monte Carlo Tree Search

```
1:  function MCTS(tree, time)
2:      root ← tree.root
3:      end_time ← current_time + time
4:      while current_time < end_time do
5:          leaf ← Traverse(root) ..... Selekce + Expanze
6:          result ← Rollout(leaf) ..... Simulace hry
7:          BackPropagate(leaf, result) ..... Zpětná propagace
8:      end while
9:      return BestAction(root) ..... Vracení nejlepší akce
10: end function
```

Algoritmus 2 Selekce a expanze

```
1: function TRAVERSE(node)
2:     while IsFullyExpanded(node) do ..... Hledáme nerozvinutý vrchol
3:         node ← BestChild(node) ..... Nejzajímavější potomek podle UCT
4:     end while
5:     return PickUnvisited(node) ..... Vracení nově rozvinutého listu
6: end function
```

Algoritmus 3 Simulace zbytku hry

```
1: function ROLLOUT(node)
2:     while not node.IsTerminial() do .... Simulovat dokud nenarazíme na list
3:         action ← RandomPossibleAction(node) ..... Podle dané strategie
4:         node ← node.Child(action)
5:     end while
6:     return GameValue(node) ..... Získání výsledku hry
7: end function
```

Algoritmus 4 Zpětná propagace

```
1: function BackPropagate(node, value)
2:     node.UpdateValue(value) ..... Aktualizace počtu navštívení a výhernosti
3:     if node is not null then
4:         BackPropagate(node.parent, value) . Rekurzivně voláme až ke kořeni
5:     end if
6: end function
```

1.4.7 MCTS a neúplná informace

Implementace MCTS využívá přesné znalosti aktuálního vrcholu v herním stromu pro budování pomocného stromu a simulování her. Proto je důležitý předpoklad, že hra, na kterou algoritmus používáme, má úplnou informaci (alespoň z pohledu hráče). Hry s neúplnou informací ztěžují situaci díky existenci informačních stavů - hráč neví přesný vrchol stromu, ale pouze podmnožinu možných vrcholů (případně s pravděpodobnostní distribucí přes možné vrcholy).

1.4.8 Determinizace

Jedno z možných řešení situace, kdy se hráč nachází v informačním stavu a nezná přímo svou pozici, je determinizace. Na začátku každé iterace se náhodně, buď uniformně či podle pravděpodobnostní distribuce vyvozené díky znalostem domény, zvolí jeden z možných vrcholů, neboli jedna z možných historií v informačním stavu, pro doplnění informací. Poté stačí použít klasický průběh MCTS iterace, jelikož neznámé informace byly doplněny. Upravený algoritmus má název Information Set MCTS (ISMCTS).

Determinizace má teoretické nedostatky, například tzv. *strategy fusion* [8] - algoritmus nalezne kvalitní strategii pro každou determinizaci zvlášť, nikoliv obecně. Navzdory tomu se v praxi prokázala silnými výsledky, často v karetních hrách.

Determinizace ve hře Fantom

Determinizaci využívají pouze detektivové, jak jsem již zmiňoval, pro Fantoma žádná skrytá informace není. Skrytá informace pro detektivy je pouze pozice Fantoma. V každém kole zjistí, který dopravní prostředek zrovna Fantom využil, můžou tedy vyvodit jeho nové možné pozice. Také v některých kolech se Fantom ukáže a detektivové zjistí jeho reálnou pozici. Na začátku každé iterace se zvolí jedna determinizace (Fantomovi se přiřadí jedna možná pozice) uniformně. Pro každou determinizaci se vytvoří vlastní strom bez skryté informace a pokračuje se pomocí MCTS. Po dokončení všech iterací se přes všechny determinizace aktuálního informačního stavu získá nejlepší akce, například průměrováním.

Kapitola 2

Související práce

2.1 Související bakalářské práce

2.1.1 FIT Michal Sova

Do řešení podobného problému - umělé inteligence ve hře Scotland Yard, se pouštěl i Michal Sova ve své bakalářské práci [10]. Využíval metod strojového učení.

Pro testování navrhl zjednodušenou verzi hry. Úprava hry spočívala v omezení počtu detektivů z původních pěti na pouze dva. Jelikož by Mr. X (obdoba Fantoma v původní verzi) byl poté ve velké výhodě, zmenšila se mapa na mřížku 5x5 (kde každý vrchol je propojen se všemi sousedy). Z dopravních prostředků zůstal pouze jeden a Mr. X přichází o své speciální tahy. Délka hry je zkrácená na 15 tahů z 22. Mr. X se ukáže jednou za tři kola, původně byl viděn přibližně jednou za pět kol.

Na navržené verzi hry byly porovnány dva různé postupy pro umělou inteligenci ve hře - Alfa-beta prořezávání a Monte Carlo Tree Search. Výsledky ukázaly, že procento výher u algoritmu Monte Carlo je nižší než u algoritmu Alfa-beta. Rozšíření hry pro algoritmus Alfa-beta nebylo úspěšné kvůli nedostatku vlastních zdrojů.

2.1.2 MUNI Matej Rišňovský

Ve své bakalářské práci [11] Matej Rišňovský implementoval hru Scotland Yard ve vývojovém prostředí Unity a zkoumal problémy návrhu umělé inteligence z diplomové práce Mária Kudolániho [12].

Za cíl měla práce vytvořit racionální umělou inteligenci, která bude hrát proti hráči, případně s ním. Do hry byly implementované 3 obtížnosti, které do rozhodování s nižší obtížností zavádějí vyšší chybovost.

2.2 Student of Games

2.2.1 Představení

Algoritmus Student of Games pochází z článku Student of Games: A unified learning algorithm for both perfect and imperfect information games [7]. Hlavní myšlenkou je udělat umělou inteligenci obecnější. Původně byly vymýšlené umělé inteligence zaměřené pouze na jednu hru s použitím specifik z domény dané hry. AlphaZero přišlo s možností hraní několika her s plnou informací. Hlavní myšlenka algoritmu Student of Games (Google DeepMind) je zobecnění i pro hry s neúplnou informací. Využívá metod self-play a prohledávání stromu hry. Při učení se snaží konvergovat k Nash. ekvilibriu. Student of Games byl, mimo jiné, úspěšně aplikován na hru Scotland Yard jako příklad hry s neúplnou informací.

2.2.2 Neuronová síť

Místo ohodnocovací funkce využívá Student of Games neuronovou síť. Na vstupu síť dostane veřejný stav hry a pravděpodobnosti privátních stavů hráčů. Jako odpověď vrací pro každý privátní stav ohodnocení pro každého hráče a navíc strategie pro každý privátní stav pro hráče na tahu. Je použita feed-forward network a residual network.

2.2.3 CFR

Student of Games využívá algoritmus CFR pro vymyšlení strategie blížící se k Nash. ekvilibriu, viz sekce 1.3.8.

2.2.4 Regret Matching+

Místo ukládání součtů regretů pomocí Regret Matching, sekce 1.3.8, je zvolen jiný přístup, tzv. Regret Matching+. Nemá žádné lepší teoretické garance, ale v praxi může fungovat lépe. Při udržování regretů se pamatuje pouze nezáporná část. Součet regretů pro informační stav I hráče i a možnou akci a v čase T je $Q^T(I,a) = \max(0, Q^{T-1}(I,a) + R_i^T(I,a))$, kde $R_i^T(I,a)$ je regret hráče i za nezahrání akce a v informačním stavu I do času T. Volba nové strategie v čase $T+1$ je následující $\pi_i^{T+1}(I,a) = Q^T(I,a) / \sum_{b \in A} Q^T(I,b)$.

GT-CFR

Growing-tree CFR je varianta CFR, která postupně zvětšuje prohledávaný strom. Začíná s prvotním stromem obsahujícím aktuální stav a jeho potomky. V každé iteraci probíhají dva kroky: aktualizace regretů a fáze expanze. Aktualizace regretu

probíhá jako u klasického CFR, ale pouze na doposud postaveném stromě. Ve fázi expanze se rozšiřuje strom přidáním nových vrcholů do stromu podle trajektorií při simulování her. Na listových vrcholech se na hodnoty hry dotazuje neuronové sítě.

2.2.5 Trénování

Data pro trénování neuronové sítě se sbírají při self-play (jak z trajektorie, tak při prohledávání stromu akcí ve stavu). V trajektorii se procházené dotazy na neuronovou síť ukládají a solver je prozkoumá podrobněji, případně i rekurzivně přidá další dotazy na vyřešení. Z dotazů a jejich řešení se aktualizuje síť. Řešení dotazů je vlastně řešení "podher" - pomocí GT-CFR (díky tomu rekurzivně vytváří další dotazy - pouze s malou pravděpodobností 0.1 nebo 0.2). Zlepšení sítě se propaguje zespod - nejdříve díky "podhrám" těsně nad listy.

Kapitola 3

Návrh a implementace

3.1 Pravidla hry

Fantom i detektivové mají svůj balíček žetonů, pomocí kterých můžou využívat různé dopravní prostředky pro pohyb po mapě. Žetony jsou nepřenosné.

Pozice v mapě jsou spojené barevnými čarami, které představují trasy jednotlivých dopravních prostředků. Černá barva představuje drožku, zelená taxi a červená tramvaj. Tah končí přesunem pomocí dopravního prostředku na nejbližší místo po trase. Lze pouze využívat dopravní prostředky zvýrazněné po kliknutí na tlačítko s daným dopravním prostředkem za předpokladu, že hráč disponuje příslušným žetonem. Detektivové mají omezený počet žetonů. Při každém tahu odevzdává detektiv Fantomovi žeton odpovídající zvolenému dopravnímu prostředku. Fantom v každém svém tahu použitý žeton ukáže. Fantom se v předem daných kolech objevuje.

3.1.1 Cíl hry

Detektivové zvítězí, podaří-li se některému dorazit na místo (pozici), kde se Fantom ukrývá. Naopak podaří-li se Fantomovi unikat až do závěru hry, stává se vítězem.

3.2 Má implementace ISMCTS

Má implementace umělé inteligence využívá Monte Carlo techniky v algoritmu Information Set Monte Carlo Tree Search (ISMCTS).

Celý algoritmus ISMCTS, sekce 1.4.8, je zapouzdřený v generické třídě, očekává třídu pro stav hry a třídu pro akci. Třída pro ISMCTS dále definuje dvě další třídy: vrchol a samostatný strom.

3.2.1 Vrchol

Třída pro pomocný vrchol na budování stromu pro prohledávání si ukládá potřebné informace ohledně stavu hry, který reprezentuje, statistiky pro samostatné Monte Carlo techniky a odkaz na potomky ve stromě. Přímo si udržuje odkaz na stav hry. Informace potřebné pro fungování ISMCTS jsou počet návštěv, počet výher, který hráč hraje a jestli se jedná o informační set (jestli je počet možných historií větší než jedna) či nikoliv. Odkaz na potomky si drží pomocí slovníku, kde klíčem je vždy akce a hodnotou je vrchol představující následující stav po zahrání dané akce. Kromě toho ještě má odkaz na třídu s popisem hry - přechody mezi stavy ve hře, ohodnocení stavu, možné akce.

3.2.2 Strom

Strom udržuje v kořeni vrchol představující počáteční stav hry, se kterým byl vytvořen. Nemusí se jednat o přímo počáteční stav hry, ale pouze určité podhry. Vnitřně si udržuje generátor náhodných čísel pro simulování a explorační parametr (UCT z formule 1.7).

3.2.3 Monte Carlo Tree Search

Třídy pro vrchol a strom jsou zapouzdřené do jedné třídy, která je generická. Dostane typy pro stav hry a pro akce. Definuje také hlavní funkce pro fungování MCTS, sekce 1.4.6.

3.2.4 Stav hry

V každém vrcholu stromu je uložený stav hry, který vrchol reprezentuje. Stav popisuje všechny důležité informace ohledně hráčů. Mezi důležité informace spadají pozice všech hráčů. Jelikož ale Fantomova pozice nemusí být z pohledu detektivů známá, jsou-li na tahu detektivové, znají pouze množinu možných pozic Fantoma. Pozice detektivů jsou vždy známé. Důležitou součástí jsou žetony. Žetony hráčů jsou veřejné. Kromě informací o pozicích a žetonech si stav udržuje aktuální kolo a kdo je na tahu.

3.2.5 Akce

V MCTS se buduje strom, ve kterém hrají dva hráči - Fantom a detektivové (jako celek), proto jedna akce je buď pohyb Fantoma nebo pohyb všech detektivů. Struktura představující akci pouze obsahuje seznam pohybů všech postav hráče - pro Fantoma pouze jedné, pro detektivy podle jejich počtu. Pohyb je reprezentovaný

strukturou obsahující novou pozici na mapě - číslo vrcholu, a použitý dopravní prostředek.

3.2.6 Popis hry

Poskytuje přechody mezi stavy a všechny možné akce pro daný stav, případně počet takových akcí. Pro terminální stavy vrací odměnu. Pro účely MCTS má také funkci, která z možných akcí vrací jednu náhodnou s uniformní distribucí.

3.2.7 Hráči

Pro implementaci umělé inteligence je nutné splňovat připravené rozhraní. Samostatní hráči si při hře pouze udržují aktuální informace o stavu hry. Ve svém tahu vytvoří novou instanci MCTS stromu s kořenem v aktuálním stavu. Pomocí třídy MCTS spustí simulaci na předem stanovený čas, případně počet iterací. Po doběhnutí algoritmu pouze vrátí správnou akci. V případě detektivů je situace lehce složitější, jelikož MCTS vrací seznam akcí všech detektivů. Detektivové si tedy zjistí všechny tahy dopředu a postupně je vrací.

3.2.8 Simulátor

Simulátor nabízí možnost simulování jednoho kola či celé hry. Hlavní úkol je rozumným způsobem reprezentovat pravidla hry - legální možnosti tahů, informování hráčů o tazích protihráče, kontrola konce hry.

3.3 Implementace hry

3.3.1 Unity

Unity je moderní herní engine. Poskytuje možnosti vývoje 2D i 3D her. Pro vývoj her obsahuje i fyzikální engine či zvukový engine. Do scén se dají vkládat objekty, které můžou obsahovat různé komponenty. Kromě komponent na vykreslování, kolize a další, je možné k objektu přiložit i skript. Skripty se často píší v jazyce C#. Přidávají objektu další funkcionalitu a chování.

3.3.2 Logika hry

Je to hlavní a nejobsáhlejší třída, obstarává průběh hry. Na začátku hry načte potřebné informace z config souboru a nastaví kameru. Poté hráčům poskytne důležité informace jako počet žetonů a počet detektivů. Po vyrenderování mapy hra začne a spustí se funkce pro průběh hry (game loop). Game loop obsahuje

nekonečný cyklus, kde jeden průběh simuluje jedno kolo hry. V každém kole se aktuálního hráče zeptá na tah, zkontroluje validitu tahu a aktualizuje stav hry. O novém stavu hry jsou oba hráči informování. Game loop je přerušen skončením hry. Hra skončí ve prospěch Fantoma, pokud trvá déle než je stanovená délka hry, pro detektivy, pokud stojí jakýkoliv detektiv na stejné pozici jako Fantom. Když hra neskončila, spustí se další iterace.

3.3.3 Pomocné skripty

Značná část skriptů v souborech hry se stará o pomocné struktury nebo funkčnost grafického rozhraní. Vývojové prostředí Unity nabízí užitečné prvky jako např. tlačítko, textové pole, ale i složitější jako scrollovací okno. Každému takovému prvku je třeba doplnit funkčnost, například pro tlačítka při kliknutí/držení/přejetí myší. Zvolil jsem způsob pomocí připraveného skriptu se správnými objekty a funkcemi. Další z užitečných skriptů se stará o kameru, přidává možnost posouvat ji pomocí šipek a přibližovat či oddalovat pohled, jiný řeší renderování mapy a hráčů.

3.3.4 Fantom

Ve hře je také přidaná základní implementace umělé inteligence pro hráče Fantoma. Využívá pouze heuristiky pro určení nejlepšího pohybu. Daná heuristika pro každý možný pohyb (nový vrchol) Fantoma určí hodnotu a poté vybere tah s maximálním ohodnocením. Pro ohodnocení vrcholu se z něj na mapě zavolá prohledávání do šířky (BFS) a hledá se nejkratší cesta, kterou se jakýkoliv detektiv může dostat na daný vrchol. Myšlenkou heuristiky tedy je vybrat takový vrchol, který je nejdál od nejbližšího detektiva. Nesnaží se přemýšlet a odhadovat další kola, ale pouze najít aktuální, v nějakém směru nejlepší tah. V prvním tahu, když si hráči volí pozice, nevyužívá heuristiky a pouze si vybere náhodný vrchol na mapě.

3.3.5 Detektivové

Hra obsahuje i jednoduchou implementaci hráče za detektivy. V deskové hře za detektivy může hrát více různých hráčů. Předpokládá se určitá úroveň spolupráce mezi hráči - mají společný cíl a nezáleží na tom, který detektiv přímo dopadne Fantoma. Z hlediska umělé inteligence dává smysl reprezentovat takovou skupinu hráčů jako jednoho, který ovládá všechny figurky a snaží se s jakoukoliv dopadnout Fantoma. Využívá základní heuristiky. Když je detektiv na tahu, vybere si vrchol který je na nejkratší cestě k pozici, kde si myslí, že se Fantom nachází. Nejkratší cesta je hledána pomocí prohledávání do šířky a hledá pouze cesty, pro

které má detektiv dostatek žetonů. Nesnaží se nijak vyvodit Fantomovu aktuální pozici z dostupných informací, ale pouze si ji vždy aktualizuje ve chvíli, kdy se Fantom ukáže. Detektivové tedy směřují na poslední Fantomovu viděnou pozici. V prvním kole se zvolí náhodná pozice pro každého detektiva, samozřejmě tak, aby nestáli na stejných pozicích.

3.3.6 Interface AI

Pro možnost rozumného rozšiřování hry o jiné implementace hráčů, Fantoma i detektivů, je připravené rozhraní pomocí interface v jazyce C#, které musí takový hráč splňovat. Zdrojový kód s třídou splňující daný interface stačí umístit mezi soubory hry k ostatním skriptům. Hra automaticky nalezne všechny vhodné třídy a poskytne možnost vybrat si, které budou hrát za Fantoma a detektivy.

Náhled na samotný interface 3.1. Interface pro jazyk C# používá následující funkce:

Listing 3.1 Definice rozhraní pro hráče

```
public interface IFantom : IPlayerBase { }

public interface IDetectives : IPlayerBase { }

public interface IPlayerBase :
IPlayerBase<GameGraphScript,GameNodeScript> { }

// Base interface for all player types
public interface IPlayerBase<MapType, NodeType>
    where MapType : IMap<NodeType>
    where NodeType : INode
{
    // Called when the player's move is valid
    public void PlayIsOK(Move lastMove);

    // Synchronously gets the player's move
    public Move GetMove();
    // Asynchronously gets the player's move
    public Task<Move> GetMoveAsync();

    // Sets the available transport tokens for the player
    public void SetTransports
        (Dictionary<Transport, int> transports);
    // Sets the available transport tokens for the opponent
    public void SetOpponentTransports
        (Dictionary<Transport, int> transports);

    // Factory method to create an instance of the player
    public static IPlayerBase CreateInstance(MapType ggs)
```

```
        => throw new NotImplementedException();

    // Called when the opponent makes a move
    public void OpponentMove(Move move);
}
```

Konfigurace

Při spouštění hry se načtou základní parametry ovlivňující různé vlastnosti a chování hry. Konfigurační soubor je uložen ve formátu JSON pod názvem "config.json" v souborech hry. Samostatný soubor je rozdělený na tři části, kde každá obsahuje parametry k určité části hry.

První část obsažená v položce "graph" udává parametry ovlivňující generování mapy. Generování ovlivňují parametry "Rows" a "Columns" udávající počet řádků a sloupců, a tři parametry obsahující pravděpodobnosti měnící generování: "probOfKeepeingNode", "probTaxi" a "probTram", jejichž použití je více přiblíženo u popisu samostatného generování.

Druhá část je pod položkou "camera". Nachází se v ní parametry, které udávají chování kamery. Mezi parametry patří "moveSpeed", "zoomSpeed", "minZoomDistance" a "maxZoomDistance", kde první parametr ovlivňuje rychlost pohybu kamery při používání šipek, druhý rychlost přibližování pomocí kolečka na myši a poslední dva pouze určují maximální a minimální přiblížení.

Třetí část je pro průběh hry nejdůležitější. V souboru je pod názvem "game". Pod parametry spadá "fantomVisibleTurns", který obsahuje seznam kol, kdy se Fantom zjeví, "fantomTokens" a "detectiveTokens" obsahující seznam dvojic dopravní prostředek a počet žetonů pro určení počátečních žetonů všech hráčů, "numberOfTurns" nastavující délku hry, "aiDelays" pro určení minimální délky tahu hráče hraného umělou inteligencí v milisekundách a "animationSpeed", který nastavuje rychlost animace hráče při přesunu na novou pozici.

Pokud soubor není nalezen, využije se základní, předem připravené nastavení zabudované ve hře.

3.3.7 Mapa

Vrchol mapy

Třída vrchol v sobě obsahuje důležité informace ohledně mapy. Pamatuje si dopravní prostředky, které zde mají zastávku, a vrcholy spojené trasami dopravních prostředků. Poskytuje možnost nastavit a upravit vzhled (sprite) podle zastávek dopravních prostředků a zvýrazňování (naznačení možného přesunu na daný vrchol při kole hráče). Implementuje základní funkce zlehčující práci.

Graf

Hlavní položka je seznam (list) vrcholů. Dále obsahuje odkaz na generátor map. Při generování grafu se buď vygeneruje nová náhodná mapa nebo, pokud je vybraná, se načte uložená mapa. Při vytvoření správně nastaví vzhled (sprite) a vlastnosti každému vrcholu. Přidává další pomocné funkce, mezi které patří i jednoduchá implementace prohledávání do šířky.

Generování mapy

Hra kromě načtení existující mapy nabízí možnost vytvoření nové náhodně vygenerované mapy. Při inicializaci generátoru mapy se načtou potřebné parametry z config souboru.

Mapa je generovaná jako mřížka s daným počtem řádků a sloupců - parametry ”Rows” a ”Columns”. V mřížce každý možný vrchol s pravděpodobností danou parametrem ”probOfKeepeingNode” přežije, jinak je smazaný z mřížky. Nyní má mřížka díry. Každému vrcholu se nyní přiřadí dopravní prostředky, které skrz něj budou procházet. Drožka prochází každým vrcholem a ostatní dopravní prostředky se přiřadí dle pravděpodobností z konfiguračního souboru, taxi dle ”probTaxi” a tramvaj dle ”probTram”. Pro vytvoření hran se prochází znovu všechny vrcholy a pro každý přiřazený dopravní prostředek se pomocí prohledávání do šířky v okolí hledají jiné vrcholy obsahující tento dopravní prostředek, které se spojí hranou.

Mapu je možné nechat uložit do formátu JSON pro případné znovupoužití. Při ukládání se zapamatují důležité informace o vrcholech, podle kterých se dá mapa znovu připravit.

3.3.8 Uživatelské rozhraní

Uživatelské rozhraní hry je připraveno s jednoduchostí v mysli. Vše je přímočaře pojmenované.

Hlavní menu

Hlavní menu na Obrázku 3.1 obsahuje několik důležitých prvků.

Na levé straně obrazovky jsou možnosti výběru hráčů za Fantoma a detektivy. Ve hře jsou připravené tři různé možnosti hráče za Fantoma i detektivy, viz Obrázek 3.2, ale hra podporuje možnost přidání hráčem vytvořené umělé inteligence splňující správné rozhraní. Mezi připravené možnosti u detektivů spadá ”DetectiveReal”, který umožňuje detektivy ovládat ručně při hře, ”DetectiveAI” je umělá inteligence využívající heuristiky pro rozhodování a ”DetectivesAIMCTS” implementující ISMCTS pro rozhodování. Pro Fantoma jsou obdobné.

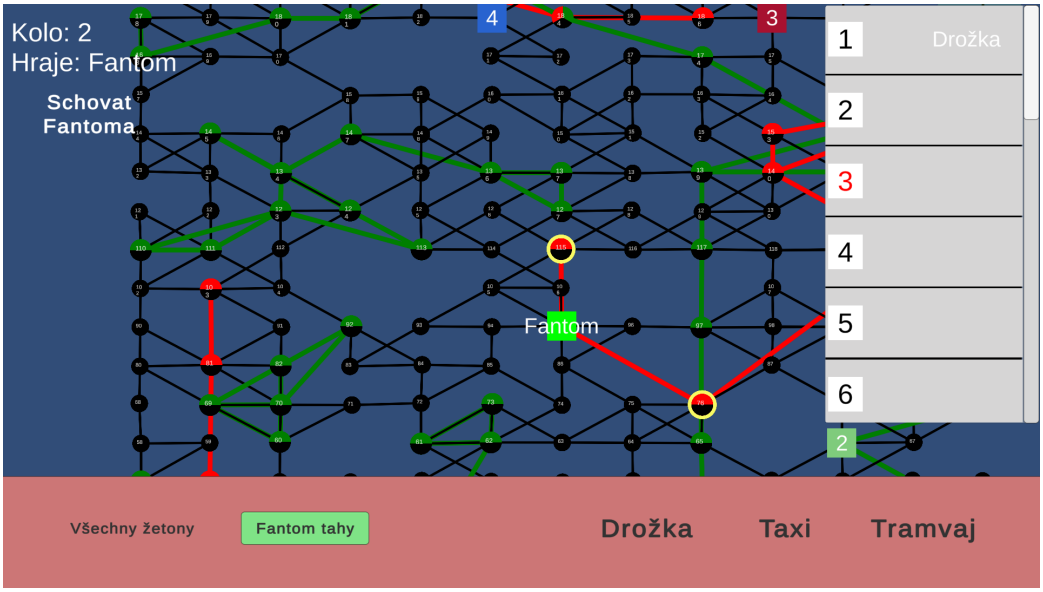
Na pravé straně se nachází tři tlačítka. Tlačítko ”UKONČIT” pouze vypne spuštěnou hru. Při kliknutí na tlačítko ”MAPY” se zobrazí možnost výběru uložených map ve formátu JSON v souborech hry. Po vybrání mapy se v následující hře použije. Pokud žádná mapa není zvolená, vygeneruje se náhodná. Poslední tlačítko ”HRÁT” spustí hru se zvoleným nastavením hráčů a mapy.

Obrázek 3.1 Hlavní menu

Obrázek 3.2 Výběr hráče

Herní obrazovka

Při hře vidí hráč mapu hry a důležité informace o hře viz Obrázek 3.3. Vlevo nahoře se nachází aktuální kolo, kdo je na tahu (Fantom či detektiv) a tlačítko ”Schovat Fantoma”, které pouze zneviditelní či naopak ukáže Fantoma, je určeno pro hraní dvou lidí proti sobě, pro hraní s umělou inteligencí se nedoporučuje využívat. Vlevo dole jsou dvě tlačítka, ”Všechny žetony”, které zobrazí tabulku s počtem aktuálních žetonů dopravních prostředků pro každého hráče, a ”Fantom tahy” zobrazující historii Fantomových tahů, která se objeví vpravo, aktuálně je zobrazená. Vpravo dole jsou tři tlačítka pro dopravní prostředky, pomocí kterých se hráč, zvolil-li si ovládání Fantoma či detektivů a je právě na tahu, vybírá dopravní prostředek na cestování. Po zvolení dopravního prostředku se zvýrazní možné tahy žlutým kruhem (ukázka po zvolení Tramvaje na Obrázku 3.3)



Obrázek 3.3 Pohled při hře, vpravo historie tahů Fantoma

Konec hry

Po dohrání hry se objeví menu nabízející ukončení hry pomocí ”UKONČIT”, hrát znovu pomocí ”HRÁT ZNOVU” a uložení mapy do správného JSON formátu mezi soubory hry pomocí ”ULOŽIT MAPU” (Obrázek 3.4).

Obrázek 3.4 Menu po dohrání hry

3.4 Instalace hry

Vše potřebné zajistí k práci přiložený FantomInstaller.exe. Vytvoří novou složku s hrou ve stejné složce, ve které se nachází. Hra je spustitelná pomocí souboru “Fantom.exe”.

Kapitola 4

Experimenty

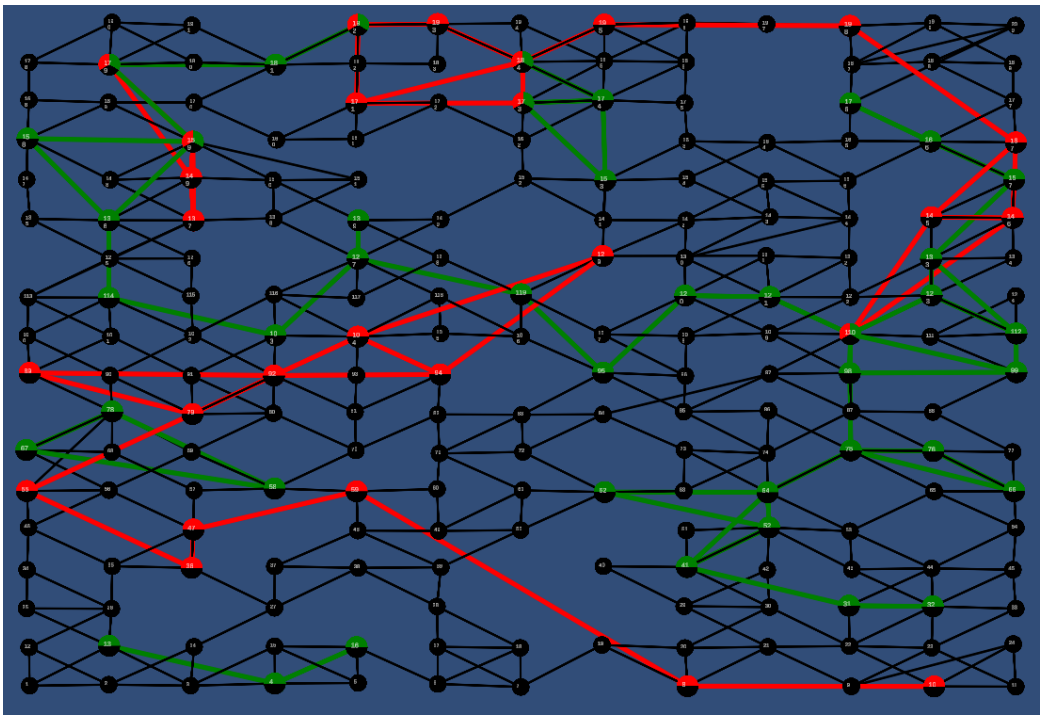
4.1 Druhy experimentů

Experimenty začnu porovnáním náhodného hráče proti hráči využívajícím heuristiku pro rozhodování, dle sekce 3.3.4 a sekce 3.3.5. Lepší z těchto dvou hráčů, tj. ten který vyhraje více her, postoupí a odehraje další hry proti hráči, který využívá ISMCTS, dle mé implementace ze sekce 3.2. Úvodní zápasy budou hrány v obou možnostech, tj. Fantom náhodný hráč proti detektivům s heuristikou a Fantom s heuristikou proti náhodným detektivům. Celý turnaj bude proveden na třech různých mapách.

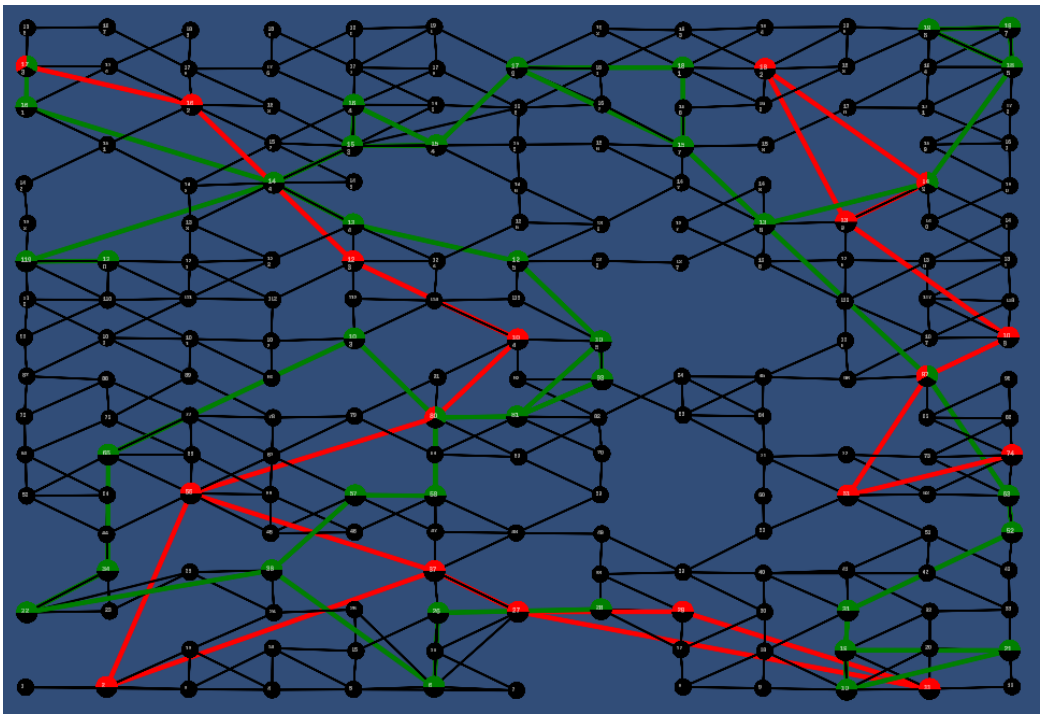
Parametry hry jsou předem stanovené. Délka hry je 24 kol. Kola, kdy se Fantom musí ukázat jsou následující: 3., 8., 13., 18. a 21. Úvodní žetony každého detektiva jsou: drožka 10krát, taxi 9krát a tramvaj 5krát, úvodní žetony Fantoma jsou: drožka 4krát, taxi 3krát a tramvaj 3krát. Všechny souboje provedu 50krát. V každém tahu má hráč s Monte Carlo technikami pro nalezení akce nastavenou délku simulování na 2,5 sekundy. Vyšší čas nabízí větší množství simulací, čímž by mohl být odhad nejlepší akce vylepšen.

4.2 Použité mapy

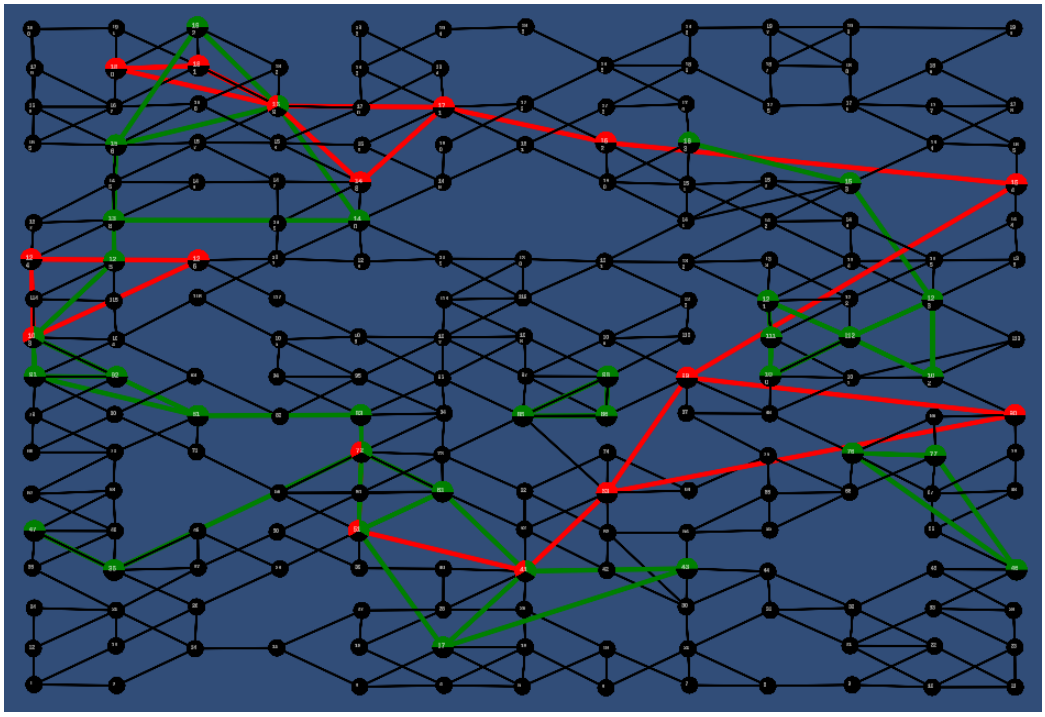
Zvolené mapy, Obrázky 4.1, 4.2 a 4.3, jsou náhodně vygenerované z implementace hry. Kvůli náhodnému generování mohou často vznikat mapy výhodné pro jednu stranu. Výběr map nedoprovázelo testování férovosti, experimenty popisují funkčnost v prostředí hry.



Obrázek 4.1 Mapa 1



Obrázek 4.2 Mapa 2



Obrázek 4.3 Mapa 3

4.3 Náhodný proti heuristice

Nejdříve se proti sobě utkají náhodný hráč s hráčem využívající heuristiky dle sekce 3.3.4 a sekce 3.3.5. Náhodný hráč samozřejmě nijak nevyužívá znalosti a vlastnosti hry. Naopak hráč využívající heuristiku se snaží použít nejlepší možný tah s aktuálními znalostmi.

Z výsledků v tabulkách 4.1 a 4.2 je očividné, že se vyplatí využívat známé informace místo náhodného chození po mapě. Přesto se detektivům, kteří využívají heuristiku, daří proti náhodnému hráči více než Fantomovi. Detektivové s heuristikou neprohráli ani jednu hru. Důvodem nejspíše bude, že Fantom chce pro výhru využívat neznalosti detektivů, také náhodnými pohyby nijak netrestá vlastnosti heuristiky detektivů - směřují pouze k poslednímu známému místu Fantoma, jelikož se může zdržovat dlouze v blízkém okolí.

Na histogramech 4.4 a 4.5 je zobrazená četnost výherních kol z pohledů detektivů přes všechny tři testované mapy.

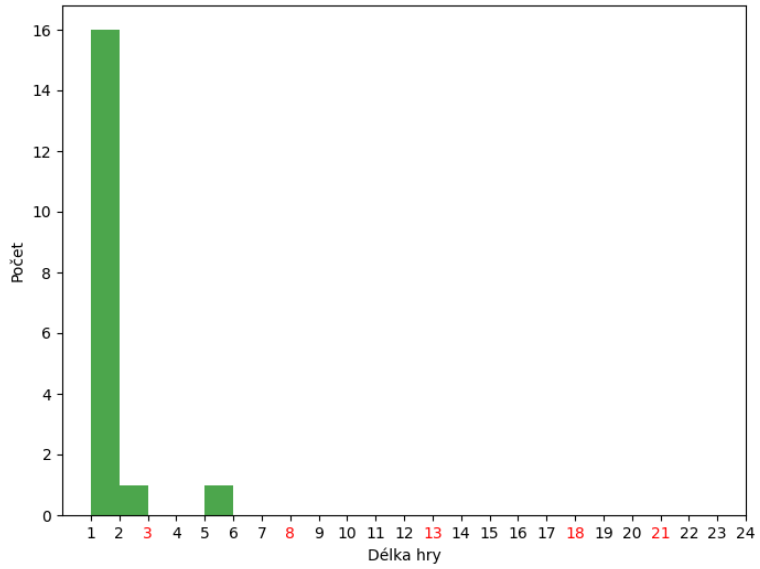
Detektivové hrající náhodně proti chytřejšímu hráči využívající heuristiky, Obrázek 4.4, vyhrávali většinou díky náhodnosti prvního kola Fantoma - první tah volí čistě náhodně. V pozdějších kolech se díky heuristice Fantom mohl detektivům vyhýbat. Oproti tomu detektivové hrající dle heuristiky, Obrázek 4.5, nejčastěji vyhráli v prvních 12ti kolech. Heuristika je proti náhodnému hráči dostatečně silná, že náhodný Fantom skoro nikdy nevyhraje.

Mapa	Detektivové	Fantom
Mapa 1	9	41
Mapa 2	5	45
Mapa 3	4	46

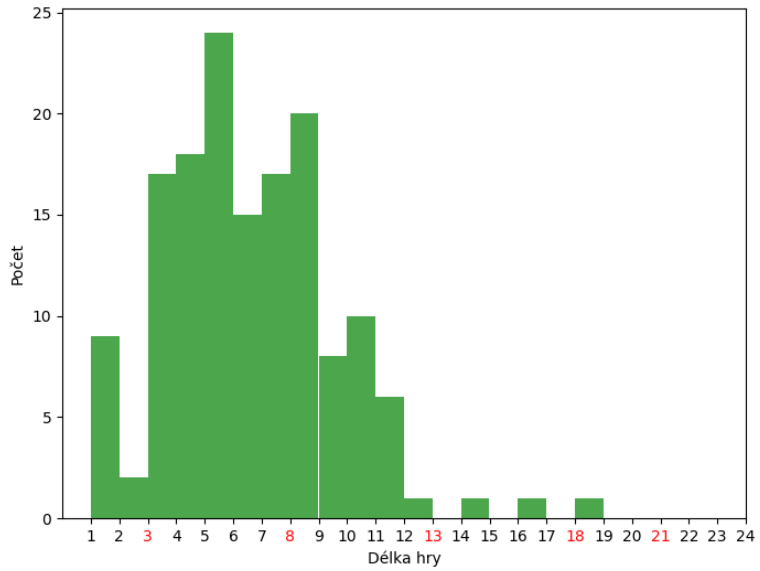
Tabulka 4.1 Počet výher detektivů (náhodný) a Fantoma (heuristika)

Mapa	Detektivové	Fantom
Mapa 1	50	0
Mapa 2	50	0
Mapa 3	50	0

Tabulka 4.2 Počet výher detektivů (heuristika) a Fantoma (náhodný)



Obrázek 4.4 Histogram délky her, kdy vyhráli detektivové. Fantom (heuristika) vs. detektivové (náhodný).



Obrázek 4.5 Histogram délky her, kdy vyhráli detektivové. Fantom (náhodný) vs. detektivové (heuristika).

4.4 Heuristika proti ISMCTS

Proti hráči využívající ISMCTS se utká hráč využívající heuristiku - výherce předešlého kola. Hráč se pomocí ISMCTS snaží simulovat průběhy her a ze simulací odvodit nejlepší tah.

Na histogramech 4.6 a 4.7 je zobrazená četnost výherních kol z pohledů detektivů přes všechny tři testované mapy.

Ve hrách, kdy Fantom využívá ISMCTS proti detektivům s heuristikou, Obrázek 4.6, vychází, že detektivové nejčastěji vyhrají kolem poloviny délky hry - mezi 7. a 18. kolem. Oproti náhodnému Fantomovi trvá detektivům déle, než se jim podaří Fantoma dopadnout. Pokud se hra dostane do posledních kol, je možné, že detektivům také dochází žetony a přichází o možnosti dopadení Fantoma. Oproti tomu detektivové využívající ISMCTS proti Fantomovi s heuristikou hrají hůře, Obrázek 4.7. Nejčastěji vyhrají díky náhodnému prvnímu kolu heuristiky, jinak v průběhu hry vyhrávají spíše ve druhé polovině.

Mapa	Detektivové	Fantom
Mapa 1	17	33
Mapa 2	20	30
Mapa 3	19	31

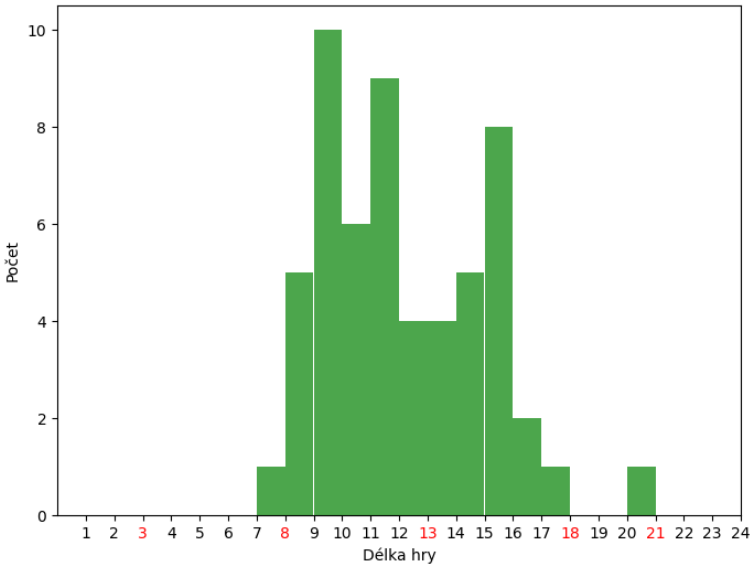
Tabulka 4.3 Počet výher detektivů (heuristika) a Fantoma (ISMCTS)

Mapa	Detektivové	Fantom
Mapa 1	4	46
Mapa 2	9	41
Mapa 3	2	48

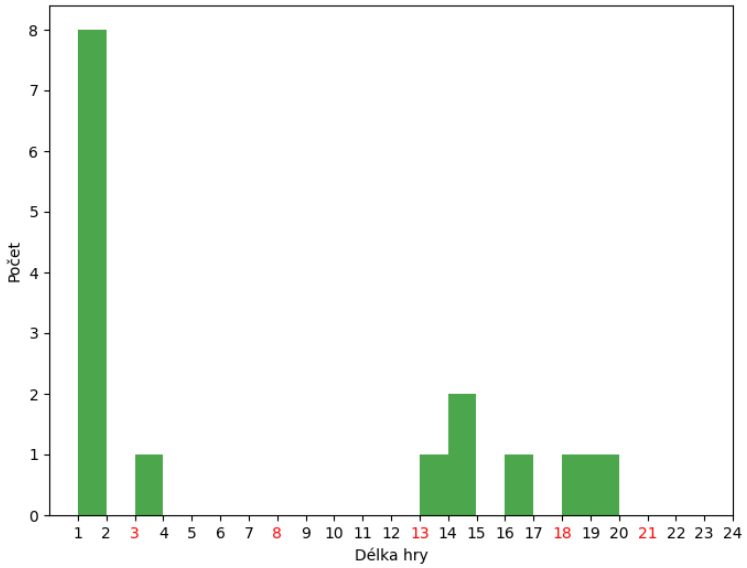
Tabulka 4.4 Počet výher detektivů (ISMCTS) a Fantoma (heuristika)

Z výsledků v tabulkách 4.3 a 4.4 vyplývá, že Fantom využívající ISMCTS je lepší než hráč s heuristikou. Proti tomu, detektivové využívající ISMCTS dosahují velmi nízké úspěšnosti. Proti Fantomovi s heuristikou vyhrají méně než 10 % her.

Důvodů může být několik. Jedním z možných problémů je již zmíněný *strategy fusion* [8], který popisuje situaci, kde se algoritmus pokusí najít nejlepší strategii pro každou determinizaci zvlášť, místo hledání nejlepší strategie pro všechny determinizace zároveň. Proti Fantomovi používají detektivové 5 figurek, mají tedy řádově více možných akcí.



Obrázek 4.6 Histogram délky her, kdy vyhráli detektivové. Fantom (ISMCTS) vs. detektivové (heuristika).



Obrázek 4.7 Histogram délky her, kdy vyhráli detektivové. Fantom (heuristika) vs. detektivové (ISMCTS).

4.5 Možná vylepšení ISMCTS

Zlepšení algoritmu může poskytnout kvalitnější strategie, například se toho dá dosáhnout zefektivněním implementace algoritmu. Jiným způsobem je využít znalost domény hry. Zabudovat znalosti lze několika způsoby. Například je možné determinizace vybírat nikoliv uniformě z informačního stavu, ale podle pravděpodobnostní distribuce určující pravděpodobnost každého stavu. Pravděpodobnější determinizace budou častěji zvoleny a budou mít možnost více ovlivnit získanou strategii. Při iteraci je možné zvolit chytřejší simulační (rollout) strategii, která reprezentuje chování chytřejšího hráče místo náhodného.

4.6 Možná vylepšení heuristiky

Zlepšení výsledků je možné dosáhnout upravením heuristiky pro první kolo. Kdyby v prvním kole nebyl výběr náhodný, bude hráč silnější. Například Fantom využívající heuristiky velmi často prohrál v prvním kole proti detektivům s ISMCTS, histogram 4.6.

4.7 Závěr experimentů

Vidíme, že mapy a parametry hry jsou nastaveny tak, že detektivové i Fantom mají naději na výhru. Záleží na jejich strategii a strategii oponenta. Navržení umělí hráči, využívající heuristiky či ISMCTS, poskytují škálu různě silných soupeřů pro lidského hráče.

Závěr

V rámci práce jsem implementoval hru na motivy Fantom staré Prahy a dva druhy umělých agentů a popsal základy teorie her a některé jejich aplikace. Popis dvou souvisejících bakalářských prací a velmi aktuálního článku mi poskytl zajímavé poznatky a vhled do řešení daného problému.

Implementoval jsem hru v herním engine Unity. Součástí práce je i hra samotná a její popis. Pro vývoj hry bylo potřebné získat znalosti o herním engine Unity a aplikovat je.

Hlavním cílem této práce bylo implementovat algoritmus umělé inteligence do hry. V rámci práce jsem implementoval dva druhy umělých agentů. První využívá heuristiky vymyšlené na základě domény hry, druhý naopak využívá Monte Carlo techniky v algoritmu Information Set Monte Carlo Tree Search (ISMCTS), které přímo nevyužívají žádné znalosti domény.

Pro otestování funkčnosti a úrovně umělých inteligencí jsem připravil menší turnaj. V prvním kole se utkala umělá inteligence využívající heuristiku proti náhodnému hráči. Heuristika vyhrála v obou případech, jak za Fantoma, tak za detektivy. Ve druhém kole se vítěz prvního kola utkal s hráčem používajícím ISMCTS. Výsledky byly smíšené, za Fantoma vyhrál hráč s ISMCTS, za detektivy naopak heuristika. Celý turnaj probíhal na třech různých mapách, v každém kole se provedlo 50 her pro obě možnosti umělých inteligencí.

Z výsledků experimentů vidíme, že umělí hráči hrají se snahou vyhrát. Díky výhrám na obou stranách můžeme usoudit, že zvolené mapy a parametry umožňují vyhrát jak Fantomovi, tak detektivům. Výherce je hráč s lepší strategií vůči jeho oponentovi. Algoritmus ISMCTS, který je nejen složitý, ale i aktuální, se prokázal být funkčním.

Implementace hry nabízí možnost za Fantoma i detektivy zvolit reálného i umělého hráče, umožňuje tedy hrát lidským hráčům proti sobě, ale i proti různě silným soupeřům umělé inteligence.

Seznam použité literatury

[1] Murray Campbell, A.Joseph Hoane a Feng-hsiung Hsu. "Deep Blue". In: *Artificial Intelligence* 134.1 (2002), s. 57–83. ISSN: 0004-3702. DOI: https://doi.org/10.1016/S0004-3702(01)00129-1. URL: https://www.sciencedirect.com/science/article/pii/S0004370201001291.

[2] J. von Neumann a O. Morgenstern. *Theory of games and economic behavior*. Princeton University Press, 1947.

[3] Martin Balko. *Algorithmic game theory (NDMI098), Lecture notes*. 2024. URL: https://kam.mff.cuni.cz/~balko/ath2324/main.pdf.

[4] Martin Schmid. *Search in Imperfect Information Games*. 2021. arXiv: 2111.05884 [cs.AI].

[5] Martin Zinkevich et al. "Regret Minimization in Games with Incomplete Information". In: *Advances in Neural Information Processing Systems*. Ed. J. Platt et al. Sv. 20. Curran Associates, Inc., 2007. URL: https://proceedings.neurips.cc/paper_files/paper/2007/file/08d98638c6fcd194a4b1e6992063e944-Paper.pdf.

[6] Tinne Hoff Kjeldsen. "John von Neumann's Conception of the Minimax Theorem: A Journey Through Different Mathematical Contexts". In: *Archive for History of Exact Sciences* 56.1 (2001), s. 39–68. ISSN: 00039519, 14320657. URL: http://www.jstor.org/stable/41134130 (cit. 07. 03. 2024).

[7] Martin Schmid et al. "Student of Games: A unified learning algorithm for both perfect and imperfect information games". In: *Science Advances* 9.46 (lis. 2023). ISSN: 2375-2548. DOI: 10.1126/sciadv.adg3256. URL: http://dx.doi.org/10.1126/sciadv.adg3256.

[8] L L G Soete et al. "Monte-Carlo Tree Search for Multi-Player Games". In: 2013. URL: https://api.semanticscholar.org/CorpusID:196010916.

[9] Volodymyr Kuleshov a Doina Precup. "Algorithms for multi-armed bandit problems". In: *Journal of Machine Learning Research* 1 (ún. 2014).

[10] Michal Sova. *Strategická desková hra s neurčitostí*. Čeština. Bakalářská práce. Brno, CZ, 2021. URL: https://www.fit.vut.cz/study/thesis/23706/.

[11] Matej Rišňovský. *Implementácia umelej inteligencie do hry Catch the Phantom [online]*. Bakalářská práce. SUPERVISOR : Roman Stoklasa. 2020 [cit. 2024-03-04]. URL: https://is.muni.cz/th/or4dk/.

[12] Mário Kudoláni. "Umělá inteligence pro hru Catch the Phantom [online]". SUPERVISOR : Roman Stoklasa. Diplomová práce. Masarykova univerzita, Fakulta informatiky, Brno, 2018 [cit. 2024-04-07]. URL: https://is.muni.cz/th/duubn/.

Příloha A

Programátorská dokumentace

A.1 Graf

Třída "GameGraphScript" je klíčovým prvkem herního prostředí, zajišťujícím řízení interakcí mezi herním grafem a jeho uzly. Zodpovídá za generování grafu na základě nových nebo existujících dat, přiřazování identifikátorů uzlů, jejich vizuální reprezentaci a generování hran mezi nimi. Umožňuje ukládání dat grafu a nabízí funkci hledání nejkratší cesty mezi uzly (BFS).

A.1.1 Vrchol

Třída "GameNodeScript" je klíčovým prvkem herního prostředí, zodpovědným za reprezentaci uzlů v herním grafu, jejich vizuální zobrazení a interakci. Umožňuje nastavovat ID, pozici, spojené uzly a transportní typy, a dále provádět ověřování spojení, manipulovat se zvýrazňovačem a volat logiku na základě interakce. Struktura "NodeData" slouží k ukládání informací o uzlech.

A.1.2 Hrana

Třída "GameEdgeScript" pouze drží texturu, kterou LineRenderer vykreslí.

A.2 Herní logika

Třída "GameLogicScript" představuje jádro herní logiky a spravuje průběh hry, interakce mezi hráči a grafem hry. Obsahuje odkazy na různé herní komponenty, včetně grafu hry, správce historie Fantoma, ovladače kamery, vykreslovače figurek, textu označujícího tah a panelu konce hry. Třída řídí hráče a tahy, nastavuje viditelnost hráčů a tokenů, implementuje vizuální efekty a animace tahu hráčů,

spravuje informace o poloze hráčů a tokeny, kontroluje podmínky ukončení hry a zajišťuje hladký průběh herní smyčky. Kromě toho obsahuje metody pro správu konce hry, zobrazuje konečný panel a ukládá statistiky. Třída také zahrnuje utility pro správu cest k souborům.

A.3 Config

Struktura "Config" zahrnuje několik vnořených struktur pro jednotlivé konfigurační sekce. Každá konfigurační sekce obsahuje několik proměnných s hodnotami nastavení. Metody BasicConfig, LoadConfig, LoadConfigOrBasic a SeeStructure umožňují manipulaci s konfiguracemi.

A.4 Logy

Rozhraní "IGameLogger" definuje metody pro logování informací o průběhu hry, chybách a dalších událostech.

Třída "EmptyLogger" implementuje rozhraní "IGameLogger" a všechny logy zahazuje, tj. neprovádí žádnou skutečnou operaci.

Třída "FileLogger" také implementuje rozhraní "IGameLogger" a slouží k logování informací do souboru. Konstruktor třídy "FileLogger" inicializuje logovací mechanismus a vytváří soubor, pokud neexistuje. Metody LogTurn, LogError a LogInfo slouží k zaznamenávání různých typů zpráv do souboru. Metody WriteLine a Write slouží k zápisu zpráv do souboru, s ohledem na vláknovou bezpečnost pomocí uzamčení (lock).

A.5 Ukládání map

Třída "MapData" je odvozena od třídy ScriptableObject a slouží k uchování dat mapy. Obsahuje informace o JSON datovém řetězci a zda je mapa vybrána.

Struktura "NodeDataHolder" obsahuje informace o uzlech na mapě, včetně jejich dopravních spojení a pozic.

Struktura "StringPosition" slouží k uchování pozice jako řetězce pro JSON serializaci.

Struktura "EnumIntPair" uchovává páry klíč-hodnota, kde klíč je celočíselný identifikátor a hodnota je seznam celočíselných hodnot.

Struktura "EnumDictionary" uchovává slovník, který mapuje klíče na seznam hodnot.

Třída “NodeConverter” obsahuje metody pro konverzi instancí třídy GameNodeScript na JSON. Tato třída umožňuje převést uzly mapy a seznam uzlů na JSON formát.

A.6 Generování map

Rozhraní IGraphGenerator definuje metody pro generování uzlů a načítání konfigurace.

Třída SimpleGraphGenerator implementuje rozhraní IGraphGenerator a provádí generování uzlů a hran v grafu na základě zadané konfigurace. Tato třída obsahuje metody pro generování uzlů, včetně zpracování existujících uzlů a vytváření hran mezi nimi. Taktéž zde jsou implementovány metody pro zpracování konfigurace a vytváření spojení mezi uzly.

A.7 Hráč

A.7.1 AI

Rozhraní “IFantom” a “IDetectives” slouží jako kontrakty pro hráče Fantoma a detektivů. Obě tato rozhraní dědí od základního rozhraní “IPlayerBase”, které obsahuje metody pro správu tahů, nastavení dostupných transportních tokenů a reakci na tahy protivníka.

A.7.2 Reálný hráč

Třída “RealBase” je abstraktní třídou pro reálné hráče a obsahuje logiku pro reakci na kliknutí na herní uzel.

A.7.3 Struktura Move

Struktura “Move” reprezentuje tah hráče a obsahuje informace o pozici a druhu dopravního prostředku.

A.7.4 Hledání hráčů

Třída “PlayerFinder” je utilitní třídou, která umožňuje nalézt typy implementující dané rozhraní v určitém assembly. Obsahuje metodu “GetTypesWithInterface”, která vrací seznam typů, které implementují specifikované rozhraní “T”. Dále obsahuje metodu “GetClassNamesFromTypes”, která z kolekce typů získá seznam názvů těchto typů.

Třída “TypeLoaderExtensions” obsahuje rozšíření pro načítání typů z assembly. Metoda “GetLoadableTypes” umožňuje získat načitatelné typy z dané assembly a zároveň zachytává výjimky, které mohou nastat během načítání typů.

A.8 Vykreslování postav

Třída “FigureRendererScript” se stará o vykreslování a pohyb herních figurek (detektivů a Fantoma). Figurky jsou reprezentovány pomocí sprite a barev. Třída obsahuje metody pro inicializaci a přípravu figurek, a také pro animovaný pohyb figurek na cílovou pozici.

Dále třída “ColorUtility” slouží pro operace spojené s barvami, například pro výpočet podobnosti barev a zjišťování, zda je barva příliš světlá pro bílý text.

Příloha B

Uživatelská dokumentace