

MASARYKOVA UNIVERZITA
FAKULTA INFORMATIKY



Má Člověče, nezlob se výherní strategii?

DIPLOMOVÁ PRÁCE

Petr Konvalinka

Brno, Jaro 2016

Prohlášení

Prohlašuji, že tato diplomová práce je mým původním autorským dílem, které jsem vypracoval samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování používal nebo z nich čerpal, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

Petr Konvalinka

Vedoucí práce: doc. RNDr. Jiří Barnat, Ph.D.

Poděkování

Rád bych poděkoval doc. RNDr. Jiřímu Barnatovi, Ph.D. za jeho cenné rady a připomínky při vedení této diplomové práce.

Shrnutí

Práce se zabývá hrami se dvěma a půl hráče. Nejprve se definují důležité pojmy a následně se probírají dva algoritmy pro výpočet optimální strategie ve zmíněných hrách. Cílem práce je také navrhnout C++ framework pro řešení takovýchto her. Součástí je i implementace prototypového řešení hry Člověče, nezlob se využívající vytvořený framework.

Klíčová slova

Hry se dvěma a půl hráče, SSG, QT, C++, Člověče, nezlob se

Obsah

1	Úvod	1
2	Diskrétní hry se dvěma a půl hráče	3
2.1	Úvod do teorie her	3
2.2	Tahové hry se dvěma a půl hráče s dokonalou informací	4
2.3	Simple stochastic games	6
2.4	Převod grafu se dvěma a půl hráče na SSG	8
3	Algoritmy pro řešení SSG	12
3.1	Randomizovaný Hoffman–Karp algoritmus	12
3.2	Shapleyho algoritmus	14
4	Framework pro řešení her se dvěma a půl hráče	16
4.1	Požadavky	16
4.2	Návrh	16
4.3	Implementace	18
4.3.1	Použité knihovny	18
	Lp Solve	18
	Eigen	18
4.3.2	Vložení grafu a jeho transformace na SSG	18
4.3.3	Výpočet optimální strategie	19
4.3.4	Vrácení výsledků	20
5	Vývoj prototypu	22
5.1	Požadavky	22
5.1.1	Řešič grafu se dvěma a půl hráči	22
5.1.2	Hra Člověče, nezlob se	23
	Pravidla	24
5.2	Návrh	25
5.2.1	Návrh řešiče her se dvěma a půl hráče	25
5.2.2	Návrh hry Člověče, nezlob se	27
5.2.3	Převod hry Člověče, nezlob se na graf	27
5.3	Implementace	29
5.3.1	Vývojové prostředí QT	30
5.3.2	Struktura aplikace	30
5.3.3	Testování	31
6	Experimentování	32
6.1	Graf se dvěma a půl hráče	32
6.1.1	Ovládání	32

6.1.2	Příklad grafu	33
6.2	Člověče, nezlob se	34
6.2.1	Ovládání	34
6.2.2	Možné kombinace figurek	37
6.2.3	Příklad 1	37
6.2.4	Příklad 2	39
6.2.5	Příklad 3	39
6.2.6	Shrnutí příkladů	39
7	Závěr	43
A	Obsah přiloženého CD	45

Seznam obrázků

- 2.1 Úprava grafu s jednou výstupní hranou 9
- 2.2 Úprava grafu s více než dvěma výstupními hranami 9
- 2.3 Úprava grafu s více než dvěma výstupními hranami 11
- 4.1 Doménový model knihovny SSG 17
- 5.1 Diagram případů užití pro řešič grafu se dvěma a půl hráče 23
- 5.2 Diagram případů užití pro hru Človeče, nezlob se 24
- 5.3 Doménový model pro program GW2.5P 26
- 5.4 Ukázka transformace hry Človeče, nezlob se na graf 29
- 6.1 Vytvořený graf v programu GW2.5P 34
- 6.2 Okno v průběhu výpočtu hry Človeče, nezlob se. 36
- 6.3 Okno po ukončení výpočtu hry Človeče, nezlob se. 36
- 6.4 Příklad 1 použití hry Človeče, nezlob se. 38
- 6.5 Příklad 2 použití hry Človeče, nezlob se. 41
- 6.6 Příklad 3 použití hry Človeče, nezlob se. 42

1 Úvod

Hry se dvěma a půl hráče jsou speciálním druhem her z odvětví *toerie her*. Využívají se zejména v ekonomice, biologii, inženýrství a počítačových vědách. Zachycují chování hráčů ve strategických situacích, ve kterých dva hráči dělají rozhodnutí, které mohou ovlivnit zájmy druhého hráče. Hráči se v těchto situacích řídí podle tzv. strategií. Pokud oba hráči použijí svou nejlepší možnou strategii, mluvíme o tzv. *optimálních strategiích*. Kromě dvou hráčů do hry vstupuje i náhoda, což je tzv. *půlhráč*. Hlavním cílem naší práce je zjistit, jak se nejlépe zachovat ve zmíněných strategických situacích, tedy vypočítat hráčům jejich *optimální strategie*.

Práce se zabývá *hrami se dvěma a půl hráče* a studuje algoritmy pro jejich řešení. Cílem práce je také navrhnout C++ framework pro řešení takovýchto her. Na základě tohoto frameworku pak vytvoříme prototypové řešení hry *Člověče, nezlob se*, které jej bude využívat.

V první kapitole se seznámíme s pojmem *Hry se dvěma a půl hráče* a *Simple Stochastic Games (SSG)*. Vysvětlíme zde teoretické základy nutné pro pochopení dalších částí práce. Popíšeme zde také převod *Her se dvěma a půl hráče* na SSG, pro jejichž výpočet je známo nemálo algoritmů.

Druhá kapitola se zabývá dvěma vybranými algoritmy pro řešení SSG. Algoritmy byly vybrány různým způsobem tak, aby každý využíval jinou techniku. Prvním z nich je *Randomizovaný Hoffman–Karp* algoritmus využívající techniku zlepšování strategie. Druhým je *Shapleyho* iterační algoritmus.

Ve třetí kapitole budeme vyvíjet C++ knihovnu pro řešení *her se dvěma a půl hráče*, která bude využívat dva algoritmy probrané v předchozí kapitole. Knihovna dostane na vstup graf *Hry se dvěma a půl hráče*. Jejím výstupem pak budou vypočítané *optimální strategie* a *optimální hodnoty* (tzv. pravděpodobnosti výhry).

Čtvrtá kapitola se zabývá vývojem prototypu na základě vytvořené knihovny. Prototyp bude mít dvě hlavní funkce. První z nich je možnost sestavení grafu *Hry se dvěma a půl hráče* v interaktivním grafickém prostředí a následné spočtení výsledků. Druhou, důležitější částí, je namodelování deskové hry *Člověče, nezlob se*. Uživatel si bude moci nastavit plán hry. Aplikace hru následně převede na graf *Hry se*

dvěma a půl hráče a vypočítá pravděpodobnost výhry, pokud by hráč táhl jednotlivými figurkami. Zjistíme tedy, kterou figurkou je nejvýhodnější v dané situaci táhnout.

Poslední kapitola se zabývá experimentováním s vyvinutým prototypem. Vyzkoušíme různé příklady hry *Člověče, nezlob se*. Zjistíme, jak dlouho výpočet potrvá a kolik bude mít uzlů graf *Hry se dvěma a půl hráče*, na který se hra transformuje.

2 Diskrétní hry se dvěma a půl hráče

Tato úvodní kapitola nás seznámí s teoretickými základy nutnými pro zvládnutí dalších částí práce.

2.1 Úvod do teorie her

V této podkapitole stručně shrneme pojem *teorie her* a velmi obecně vysvětlíme některé termíny, které jsou s touto vědní disciplínou nutně spjaté, a které budeme používat v dalších částech práce. Nejdůležitější pojmy jsou shrnuty v tabulce 2.1

Teorie her je speciální disciplína aplikované informatiky. Řeší široké spektrum konfliktních rozhodovacích situací s více účastníky – hráči. Pojem *hra* má velmi obecný význam. Každý si jistě představí deskové hry jako *šachy* nebo *Člověče, nezlob se*. Do tohoto oboru můžeme ale zařadit téměř jakoukoli konfliktní situaci mezi jedinci, podniky, armádami, státy, politickými stranami a biologickými druhy. Teoretické modely se pak snaží tyto konflikty nejen analyzovat, ale sestavením matematických modelů a pomocí výpočtů se snaží nalézt co nejlepší strategie pro účastníky takových konfliktů.

V průběhu rozhodovacích situací hráči vybírají **strategii** ze svého prostoru *strategií* podle hodnot výplatní funkce (angl. *payoff*), laicky řečeno: „*Kolik mohu získat*“. Tato výplatní funkce ovšem nezávisí pouze na samotném hráči, ale i na rozhodnutí ostatních hráčů. Výplatní funkce proto musí zohlednit všechny možné kombinace rozhodnutí ostatních hráčů. Např. pokud budeme hrát šachy, není vhodné popojet jezdcem tak, že vytvořím šach, ale soupeř mi jej vezme v následujícím tahu dámou. Strategie, která mi v dané konfliktní situaci zajistí nejvyšší dosažitelnou hodnotu výplatní funkce, se nazývá **optimální strategie**. Právě určení *optimální strategie* je jedním z hlavních výpočetních problémů, které budeme řešit i my v naší práci.

Ve většině her je důležitým předpokladem, že jsou hráči inteligentní (chovají se racionálně) – tedy chtějí maximalizovat hodnotu své výplatní funkce. Jako je tomu např. u hry *Člověče, nezlob se*.

Hry můžeme rozdělit na hry: s *dokonalou informací* (angl. *perfect information*) a *nedokonalou informací* (angl. *imperfect information*). U her

s dokonalou informací může v jednom okamžiku hrát pouze jeden hráč, který zná všechny tahy, které byly provedeny do tohoto okamžiku. Např. u hry *šachy* víme, jakým tahem hrál soupeř a jakým předtím my. Naopak u her s nedokonalou informací mohou hráči nezávisle na sobě.

Dalším dělení her je na hry s *nulovým součtem* (angl. *zero-sum*) a *nenulovým součtem* (angl. *non-zero-sum*). Hru nazveme s *nulovým součtem*, pokud v každé rozhodovací situaci je součet výplatních funkcí roven nule. Jinak řečeno, přesně to, co já získám, tak druhý ztratí (v případě hry dvou hráčů).

V nadpisu této kapitoly se vyskytuje pojem *diskrétní*. Hra je *diskrétní*, pokud je konečná. Tzn., že musí obsahovat konečné množství hráčů, kteří mají na výběr z konečného množství strategií.

Řekneme-li, že hra je *tahová*, znamená to, že se hráči ve hře střídají. Jinak řečeno hráč musí vyčkat, dokud není na řadě.

V naší práci se budeme věnovat diskrétním tahovým hrám s *dokonalou informací* a *nulovým součtem*, jako jsou např. již zmíněné *šachy* nebo *Člověče, nezlob se*.

V této podkapitole je využito informací ze zdrojů [1] a [2].

2.2 Tahové hry se dvěma a půl hráče s dokonalou informací

Naše práce je zaměřená na *tahové hry s dokonalou informací*. Hráči se tedy ve hře střídají a mají dokonalý přehled o hře. Hra probíhá v grafu, kde každý uzel znamená nějaký stav ve hře. Hra může probíhat na nekonečně mnoho kol. V každém kole se stav hry změní pomocí hrany do následujícího uzlu (tzv. následníka). Uzly jsou rozděleny na uzly *hráče 1*, uzly *hráče 2* a *pravděpodobnostní uzly* (to je tzv. půl hráč). V uzlech *hráče 1* vybírá hráč 1, jaký bude následník a v uzlech *hráče 2* zase vybírá hráč 2 svého následníka. V *pravděpodobnostních* uzlech je následník volen podle pravděpodobnostní funkce, která může být, ale také nemusí, rovnoměrně rozložena mezi všechny následníky.

Definice 2.2.1. *Graf se dvěma a půl hráče* $G = ((V, E), (V_1, V_2, V_P), \delta)$ se skládá z orientovaného grafu (V, E) , množiny vrcholů V , která je rozdělena na tři podmnožiny $V_1, V_2, V_P \subseteq V$, a pravděpodobnostní

2. DISKRÉTNÍ HRY SE DVĚMA A PŮL HRÁČE

<i>hra</i>	rozhodovací situace, konflikt
<i>hráč</i>	účastník konfliktu, rozhodovatel, jedinec, politická strana
<i>strategie</i>	konkrétní alternativa, kterou může hráč zvolit
<i>prostor strategií</i>	seznam všech možných alternativ, které jsou hráči dostupné
<i>výplatní funkce</i>	výsledek hry, tj. užitek, výhra hráče v závislosti na zvolených strategiích
<i>optimální strategie</i>	strategie, která hráč zajistí co nejvyšší užitek
<i>hra s dokonalou informací</i>	v jednom okamžiku může hrát pouze jeden hráč, který zná historii všech rozhodovacích situací
<i>hra s nulovým součtem</i>	hráč získává na úkor ostatních
<i>diskrétní hra</i>	hra je konečná, má konečný počet hráčů, tahů atd.
<i>tahová hra</i>	hráči se v tazích postupně střídají

Tabulka 2.1: Základní pojmy teorie her [1]

přechodové funkce $\delta : V_P \rightarrow \text{Dist}(V)$. $\text{Dist}(V)$ je pravděpodobnostní rozložení na V , $\delta : V \rightarrow [0, 1]$ takové, že $\sum_{v \in V} \delta(v) = 1$. Prostor vrcholů V je konečný. Vrcholy V_1 jsou vrcholy *hráče 1*, vrcholy V_2 jsou vrcholy *hráče 2* a vrcholy V_P jsou *pravděpodobnostní* vrcholy. Pro všechny vrcholy $v \in V$, definujeme $E(v) = \{t \in V \mid (v, t) \in E\}$ jako množinu možných následníků. Je vyžadováno $E(v) \neq \emptyset$ pro všechny nepravděpodobnostní stavy $s \in V_1 \cup V_2$. Ve stavech *hráče 1* vybírá *hráč 1* následníka z $E(s)$. Ve stavech *hráče 2* vybírá *hráč 2* následníka z $E(v)$ a v pravděpodobnostních stavech $S \in S_P$ je následník vybírán podle pravděpodobnostní přechodové funkce $\delta(v)$.

Pro určení vítěze hry musí existovat nějaký cíl. Tedy to, co musí hráč udělat proto, aby vyhrál. Cílem může být např. dosáhnout nějakého stavu anebo se nějakému stavu vyhnout. V našem případě se budeme chtít dostat do nějakého stavu (např. ve hře *Člověče, nezlob se* dostat se do cílového domečku). Budeme tedy používat tzv. *dosažitelné cíle hry*.

Definice 2.2.2. *Dosažitelné cíle* $T \subseteq S$ jsou stavy, kterých musíme ve hře dosáhnout, aby jsme vyhráli.

Další důležité pojmy, jako *hrací strategie* a *hrací hodnoty*, budou vysvětleny v následující kapitole o *Simple stochastic games*.

V této kapitole je čerpáno z [3].

2.3 Simple stochastic games

Simple stochastic games (česky jednoduchá pravděpodobnostní hra, zkráceně SSG) jsou speciální druhem diskrétních her se *dvěma a půl hráče* s dosažitelnými cíli. Pro tento typ hry je známo hodně algoritmů, které dokážou řešit problém nalezení optimální strategie. Narozdíl od *Her se dvěma a půl hráče* obsahuje každý uzel v grafu přesně dvě výstupní hrany. *Graf se dvěma a půl hráče* ale jde převést na SSG graf, více v kapitole 5.4.

Definice 2.3.1. *Simple stochastic games* (SSG) je orientovaný graf $G = (V, E)$ s následujícími vlastnostmi. Množina vrcholů V je spojením množin V_{max} , V_{min} , $V_{average}$ společně se dvěma speciálními uzly nazývanými $0 - sink$ a $1 - sink$. Jeden vrchol z V je nazýván *startovní vrchol* (ve kterém začíná hra). Každý vrchol z V má přesně dvě výstupní hrany, kromě $0 - sink$ vrcholů (ty nemají žádné).

Hra začíná ve startovním vrcholu. Představme si, že hrají dva hráči. Jeden si říká Max a druhý Min. Když je hra ve vrcholu V_{max} , tak je na řadě hráč Max a vybere jednu ze dvou výstupních hran (tedy další vrchol, ve kterém bude hra pokračovat). Když je hra ve vrcholu V_{min} , tak následníka vybírá hráč Min. Pokud je hra ve vrcholu $V_{average}$, tak je následník zvolen náhodně.

Definice 2.3.2. *Maximální a minimální strategie.* Maximální strategie σ je množina hran z E , kde počáteční vrchol v z hrany je z množiny V_{max}

a pro každý takový vrchol v existuje pouze jediná hrana (v, z) v σ . Kdykoli je pak hra ve vrcholu v a hráč Max bude hrát podle strategie σ , tak se hra přesune do vrcholu z . Stejným způsobem je definována minimální strategie τ , pouze vrcholy jsou z V_{min} . Podgrafem grafu G je pak graf $G_{\sigma, \tau}$, který obsahuje pouze hrany z σ a τ .

Definice 2.3.3. *Hodnota $v_{\sigma, \tau}(i)$ každého vrcholu i z grafu G je pravděpodobnost, že hráč Max vyhraje hru, když bude hrát podle strategie σ a hráč Min podle strategie τ . Optimální hodnota vrcholu i je definována jako $\max_{\sigma} \min_{\tau} v_{\sigma, \tau}(i)$.*

Předpoklad hry SSG je, že se zastaví. Znamená to, že v grafu $G_{\sigma, \tau}$ existuje z každého vrcholu cesta do Sink uzlu.

Lemma 2.3.1. Nechť G je SSG graf s n vrcholy a strategiemi σ, τ hráčů Max, Min, vzájemně. Předpokládejme, že vrcholy grafu $G_{\sigma, \tau}$ (kromě sink uzlů) jsou očíslovány $\{1, \dots, t\}$ a nechť vektor hodnot $\bar{v}_{\sigma, \tau} = (v_{\sigma, \tau}(1), \dots, v_{\sigma, \tau}(t))$. Pak existuje $t \times t$ matice Q a t -vektor \bar{b} takový, že $\bar{v}_{\sigma, \tau}$ je unikátním řešením rovnice:

$$\bar{v}_{\sigma, \tau} = Q\bar{v}_{\sigma, \tau} + \bar{b} \quad (2.1)$$

Q je matice, u které ij -tý prvek je q_{ij} , což je pravděpodobnost dosažení vrcholu j z vrcholu i v jednom kroku. \bar{b} je vektor, u kterého i -tý prvek je q_i , což je pravděpodobnost dosažení 1-sinku v jednom kroku.

Důkaz je obsažen v [4].

Pomocí této rovnice můžeme tedy vypočítat hodnoty vrcholů za použití nějakých strategií hráče Max a hráče Min.

Zobecnění hry o více Sink vrcholů. Ve hře je někdy potřeba použít více, než dva Sink vrcholy s různými hodnotami. Hru SSG můžeme zobecnit tak, že může mít více Sink vrcholů. Každý Sink vrchol s má hodnotu $p(s) \in (0, 1)$. Když se tedy hra dostane do vrcholu s , tak hráč Max zvítězí s pravděpodobností $p(s)$. [5]

Definice 2.3.4. *Optimální vektor hodnot \bar{v}_{opt} je vektorem optimálních hodnot vrcholů grafu G . Optimální vektor hodnot je unikátním řešením*

následujících omezení (předpoklad je, že se hra zastaví):[5]

$$\begin{array}{ll}
 v(i) = \max(v(j), v(k)) & \text{Když } i \text{ je Max vrchol s následníky } j \text{ a } k \\
 v(i) = \min(v(j), v(k)) & \text{Když } i \text{ je Min vrchol s následníky } j \text{ a } k \\
 v(i) = 1/2(v(j) + v(k)) & \text{Když } i \text{ je Ave vrchol s následníky } j \text{ a } k \\
 v(i) = p(s) & \text{Když } i \text{ je Sink vrchol}
 \end{array}$$

Definice 2.3.5. Vektor \bar{v} je *stabilní*, pokud pro každý jeho vrchol platí omezení jako definici 2.3.4. *Stabilní* vektor je tedy zároveň *optimální*.

Definice 2.3.6. Max strategie σ je *optimální s respektem* k Min strategii τ , když pro všechny Max vrcholy i s následníky j platí: $v_{\sigma, \tau}(i) \geq v_{\sigma, \tau}(j)$. Stejným způsobem Min strategie τ je *optimální s respektem* k Max strategii σ , když pro všechny Min vrcholy i s následníky j platí: $v_{\sigma, \tau}(i) \leq v_{\sigma, \tau}(j)$.

Definice 2.3.7. Maximální strategie σ je *optimální*, když pro všechny vrcholy i platí, že $\min_{\tau} v_{\sigma, \tau}(i)$ je rovno optimální hodnotě $v(i)$ v grafu G . Podobně Minimální strategie τ je *optimální*, když pro všechny vrcholy i platí, že $\max_{\sigma} v_{\sigma, \tau}(i)$ je rovno optimální hodnotě $v(i)$ v grafu G .

Definice 2.3.8. Vrchol i je *přepínatelný*, pokud má následníky j, k a platí následující: když i je Max vrchol a $v(i) < \max(v(j), v(k))$ nebo když i je Min vrchol a $v(i) > \min(v(j), v(k))$. Pokud strategie $\sigma(\tau)$ obsahuje hranu (i, j) , řekneme, že strategie $\sigma'(\tau')$ je získána *přepnutím* vrcholu i . $\sigma'(\tau') = \sigma(\tau) - \{(i, j)\} + \{(i, k)\}$.

V této kapitole je čerpáno z [5] a [6].

2.4 Převod grafu se dvěma a půl hráče na SSG

Nevýhodou SSG je, že vrcholy obsahují přesně dvě výstupní hrany. V naší práci budeme chtít řešit případy, kdy grafy budou mít i více výstupních hran a nebo pouze jednu. Takovéto grafy tedy musíme převést pomocí následujících úprav na graf SSG.

1. **Vrchol má pouze jednu výstupní hranu.**

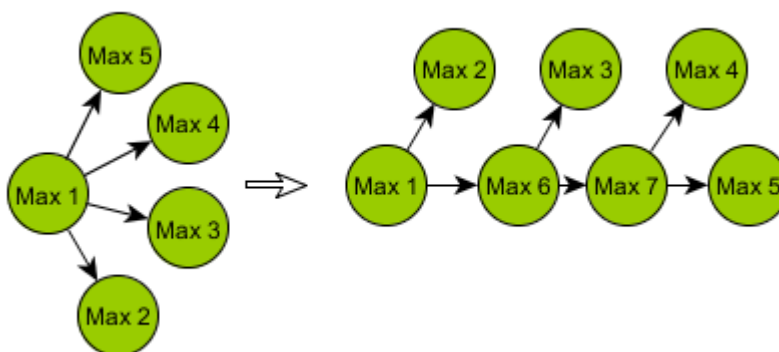
V takovémto případě jednoduše přidáme ještě jednu výstupní hranu, která povede do stejného uzlu. Úprava je stejná pro Max, Min a Ave vrcholy. Viz. obrázek 2.1.



Obrázek 2.1: Úprava grafu s jednou výstupní hranou

2. **Max nebo Min vrchol má více než dvě výstupní hrany.**

Pokud má Max nebo Min uzel (počáteční) více než dvě hrany (následníky), tak provedeme transformaci podle následujícího postupu. Počátečnímu uzlu přidáme prvního z následníků a jako dalšího následníka mu přidáme nově vytvořený uzel. Tomuto nově vytvořenému uzlu dáme druhého následníka počátečního uzlu a pokud zbývá ještě více než jeden následník, tak vytvoříme znovu nový uzel atd. Transformaci můžete vidět na obrázku 2.2, kdy byly nově přidány vrcholy Max 6 a Max 7.



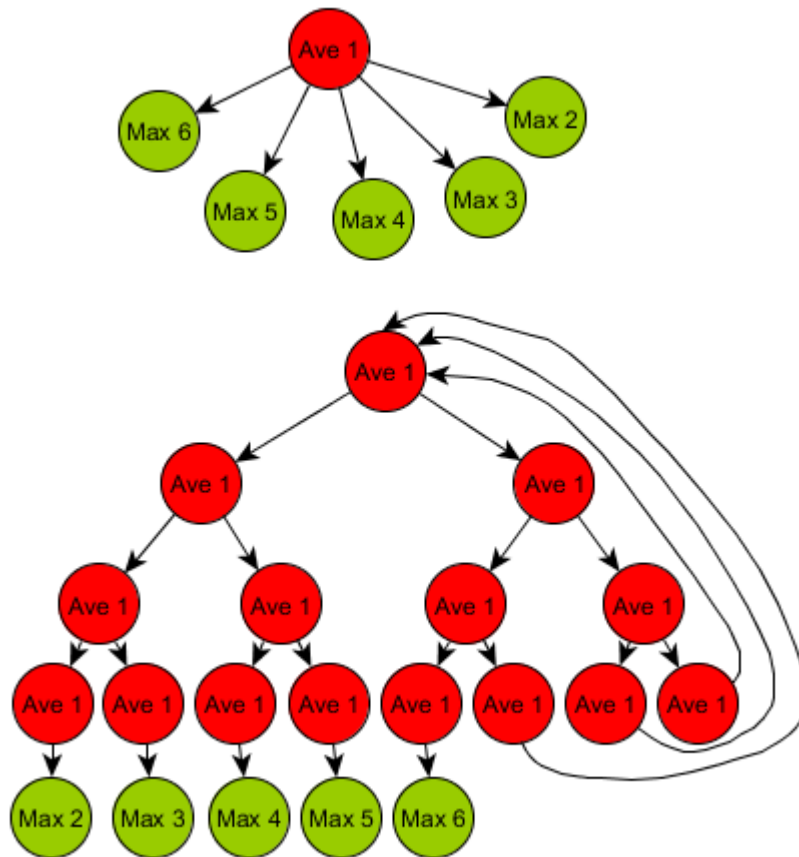
Obrázek 2.2: Úprava grafu s více než dvěma výstupními hranami

3. **Ave vrchol s více než dvěma výstupními hranami.**

Poslední úprava a také nejsložitější je, pokud máme pravděpodobnostní uzel s více než dvěma výstupními hranami. Musíme totiž zajistit, aby pravděpodobnost každého uzlu byla zachována. Když máme čtyři následníky, tak každý z nich má 25% ($1/4$) pravděpodobnost, že hra bude pokračovat na něm.

Ave uzel i má x následníků. Označme si pravděpodobnosti všech následníků ve formě $1/x$. Pak určíme proměnnou t podle vztahu $2^t < x \leq 2^{t+1}$. Uzel i je pak nahrazen kompletním binárním stromem Ave uzlů šířky $t + 1$ s kořenem i . Prvních x listů stromu má výstupní hranu do x následníků, zbývající listy stromu mají hranu do kořene stromu (počátečního uzlu) i . Prvních x listů může být rovnou nahrazeno následníky, abychom ušetřili prostor. [4]

Příklad transformace s pěti následníky můžete vidět na obrázku 2.3.



Obrázek 2.3: Úprava grafu s více než dvěma výstupními hranami

3 Algoritmy pro řešení SSG

V této kapitole uvedeme dva algoritmy pro výpočet optimální strategie v SSG, které později naprogramujeme. Všechny důležité pojmy a definice, které se v algoritmech objeví, jsme vysvětlili v předchozí kapitole. Algoritmy pro výpočet optimální strategie v SSG můžeme rozdělit na čtyři části:

- **Iterační** – algoritmy začínají s určitým iniciálním vektorem a postupně ho v každé iteraci vylepšují.
- **Se zlepšující se strategií** – algoritmy začínají s iniciálními strategiemi σ a τ pro hráče Max a Min. Poté je v každé iteraci strategie jednoho z hráčů zlepšena pomocí *přepnutí vrcholu*.
- **Matematicko–optimalizační** – problém nalezení optimální strategie je redukován na matematicko–optimalizační problém, jako je např. kvadratické programování.
- **Randomizované** – tyto algoritmy jsou většinou kombinací některého z předešlých typů. Využívají i náhodu.

Abychom vyzkoušeli co nejvíce metod, zvolil jsem dva úplně odlišné algoritmy. První je zástupce iteračních algoritmů – *Shapleyho*. Jako druhého jsem zvolil *randomizovaný algoritmus Hoffman–Karp*, který ale využívá i zlepšující se strategii. Pokryjeme tedy tři části algoritmů.

3.1 Randomizovaný Hoffman–Karp algoritmus

Randomizovaný Hoffman–Karp algoritmus je vylepšením základního strategie zlepšujícího algoritmu, který vymysleli pánové Hoffman a Karp. Tento randomizovaný algoritmus je od Anne Condon [6]. Condon ukázala, že v očekávaný počet iterací tohoto algoritmu je $2^{n-f(n)} + 2^{o(n)}$ pro jakoukoli funkci $f(n) = o(n)$, kde n je počet uzlů v grafu.

Na začátku algoritmu se vyberou dvě libovolné *maximální* a *minimální* strategie. Libovolnou strategii můžeme vytvořit např. tak, že do ní dáme vždy první hranu vystupující z vrcholu (každý Max a Min

Algoritmus 1: Randomizovaný Hoffman-Karp algoritmus [6]**Vstup :** Graf SSG G **Výstup:** Optimální pár strategií (σ, τ) a optimální vektor hodnot v_{opt} **begin**Nechť σ, τ jsou libovolné strategie pro hráče MAX, MIN**while** ($\bar{v}_{\sigma, \tau}$ není stabilní) **do**Vyber náhodně a jednotně $2n$ neprázdných podmnožin
z V_{MAX} , které jsou $\bar{v}_{\sigma, \tau}$ -přepínatelnéNechť $\sigma_1, \dots, \sigma_{2n}$ jsou strategie získané přepnutím z těchto
podmnožin

Nechť optimální strategie hráče MIN vzhledem

k $\sigma_1, \dots, \sigma_{2n}$, jsou strategie τ_1, \dots, τ_{2n} Nechť $1 \leq k \leq 2n$ jsou indexy takové, že platí

$$\sum_{x=1}^n \bar{v}_{\sigma_k, \tau_k}(x) \geq \sum_{x=1}^n \bar{v}_{\sigma_l, \tau_l}(x) \text{ pro všechny } 1 \leq l \leq 2n$$

Nechť $\sigma \leftarrow \sigma_k$ a $\tau \leftarrow \tau_k$ **end while****return** strategie σ, τ a optimální vektor $\bar{v}_{\sigma, \tau}$ **end**

uzel má přesně dvě hrany). Dále probíhá cyklus, dokud nezískáme stabilní vektor \bar{v} - tedy zároveň i optimální vektor.

V cyklu nejprve musíme zjistit, které Max uzly jsou přepínatelné. Poté z takovýchto uzlů vytvoříme náhodně $2n$ podmnožin, které nemusí být nutně odlišné. Z každé podmnožiny pak získáme novou strategii σ' , která vznikne přepnutím uzlů, které jsou v podmnožině. Ke každé nově získané strategii σ' musíme vypočítat minimální strategii, která je k ní s respektem optimální. K výpočtu optimální minimální strategie s respektem k určité strategii σ je zvláštní algoritmus 2 popsany níže. Následně ze všech nově získaných strategií σ' vyberu tu, která má největší součet vektoru hodnot $\bar{v}_{\sigma, \tau}$. Pokud zjistím, že vektor hodnot $\bar{v}_{\sigma, \tau}$ podle vybraných strategií je stabilní, tak končím výpočet a vrátím optimální strategii.

Pro výpočet optimální minimální strategie s respektem k maximální strategii σ můžeme použít algoritmus, který počítá optimální vektor hodnot v grafu, který obsahuje pouze Ave a Min uzly. My v grafu sice máme Max uzly, ale každý má pouze jednu výstupní hranu (podle

strategie), takže Max uzly můžeme transformovat na Min uzly (které při výpočtu pouze převezmou hodnotu ze svého následníka).

Algoritmus 2: LP Algoritmus pro SSG pouze s MIN a AVE uzly [4]

Vstup : Graf SSG G pouze s MIN a AVE uzly

Výstup: Optimální vektor hodnot v_{opt}

begin

Maximalizuj $\sum_{x=1}^n v(x)$

podle omezení:

$$\begin{array}{ll} v(x) \leq v(y) & \text{if } x \in V_{MIN}, (x, y) \in E \\ v(x) \leq \frac{1}{2} \cdot v(j) + \frac{1}{2} \cdot v(k) & \text{if } x \in V_{AVE}, (x, y), (x, z) \in E \\ v(x) = p(x) & x \in SINK \\ v(x) \geq 0 & x \in V \end{array}$$

Vyřeš omezení pomocí lineárního programování

return optimální vektor $\bar{v}_{\sigma, \tau}$

end

Algoritmus počítá optimalizační problém lineárního programování s omezeními tak, že *hodnoty vrcholů* maximalizuje.

3.2 Shapleyho algoritmus

Shapleyho algoritmus má mnohem jednodušší princip než *randomizovaný Hoffman–Karp* algoritmus. Algoritmus patří mezi iterační algoritmy. Vymyslel jej matematik Loyd Shapley. Anne Condon [6] ukázala, že v nejhorším případě algoritmus proběhne na $\Omega(2^n)$ iterací, kde n je počet uzlů v grafu. Algoritmus má tedy také exponenciální složitost.

Algoritmus začíná tak, že se nainicializuje *vektor hodnot* podle daných pravidel (viz alg. 3). Následně probíhá cyklus, dokud *vektor hodnot* \bar{v} není *stabilní*. V každé iteraci se pro každý vrchol spočítá nová hodnota v . *Hodnota* Max uzlu je maximum z *hodnot* svých následníků. Podobně *hodnota* Min uzlu je minimum z *hodnot* svých následníků. *Hodnota* Ave uzlu je polovina ze součtu *hodnot* svých synů. *Hodnota* Sink uzlů se nemění.

Nevýhodou tohoto algoritmu je, že nám zároveň nespočítá i *optimální strategii*. Po vypočítání *optimálního vektoru* je tedy musíme ještě

Algoritmus 3: Shapleyho algoritmus [7]**Vstup :** Graf SSG G **Výstup:** optimální vektor hodnot v_{opt} **begin**Inicializujeme následovně vektor hodnot \bar{v} :Pro každé $x \in V$ platí:

$$v(x) = \begin{cases} 1 & \text{if } x \in V_{MAX} \\ 0 & \text{if } x \in V_{MIN} \\ \frac{1}{2} \cdot v(y) + \frac{1}{2} \cdot v(z) & \text{if } x \in V_{AVE}, (x, y), (x, z) \in E \\ p(x) & \text{if } x \in SINK \end{cases}$$

while ($\bar{v}_{\sigma, \tau}$ není stabilní) **do**Nechť v' je definováno následovně:

$$v'(x) = \begin{cases} \max\{v(y), v(z)\} & \text{if } x \in V_{MAX}, (x, y), (x, z) \in E \\ \min\{v(y), v(z)\} & \text{if } x \in V_{MIN}, (x, y), (x, z) \in E \\ \frac{1}{2} \cdot v(y) + \frac{1}{2} \cdot v(z) & \text{if } x \in V_{AVE}, (x, y), (x, z) \in E \\ v(x) & \text{if } x \in SINK \end{cases}$$

Nastav $v \leftarrow v'$ **end while****return** optimální vektor \bar{v} **end**

dopočítat. Můžeme to udělat tím způsobem, že se u každého Max (Min) vrcholu podíváme na optimální hodnoty svých následníků. Optimální strategie pak povede do toho následníka, který bude mít stejnou hodnotu, jako náš Max (Min).

4 Framework pro řešení her se dvěma a půl hráče

Úkolem naší práce je vytvořit framework pro řešení *her se dvěma a půl hráče*. Práce proběhne klasickým způsobem SW vývoje. Nejprve zanalyzujeme požadavky, podle kterých vytvoříme návrh a nakonec knihovnu implementujeme.

4.1 Požadavky

Jako jsme popsali v kapitole 2, hra se *dvěma a půl hráče* je graf, který obsahuje čtyři druhy uzlů – Max, Min, Ave a koncové Sink. V tomto grafu budeme chtít nalézt *optimální strategie* pro oba hráče (jeden je Max a druhý Min). Zároveň budeme chtít vypočítat optimální hodnoty všech uzlů v grafu.

Potřebujeme tedy vytvořit framework (knihovnu), kterému když zadáme graf *hry se dvěma a půl hráče*, tak nám vypočte *optimální strategie* i *optimální hodnoty* všech vrcholů. K výpočtu strategie by měly sloužit alespoň dva algoritmy, abychom si mohli jednoduše ověřit výsledky (od obou algoritmů musí být stejné). Knihovna by měla být nezávislá na platformě, aby se dala jednoduše použít v jakémkoli SW projektu.

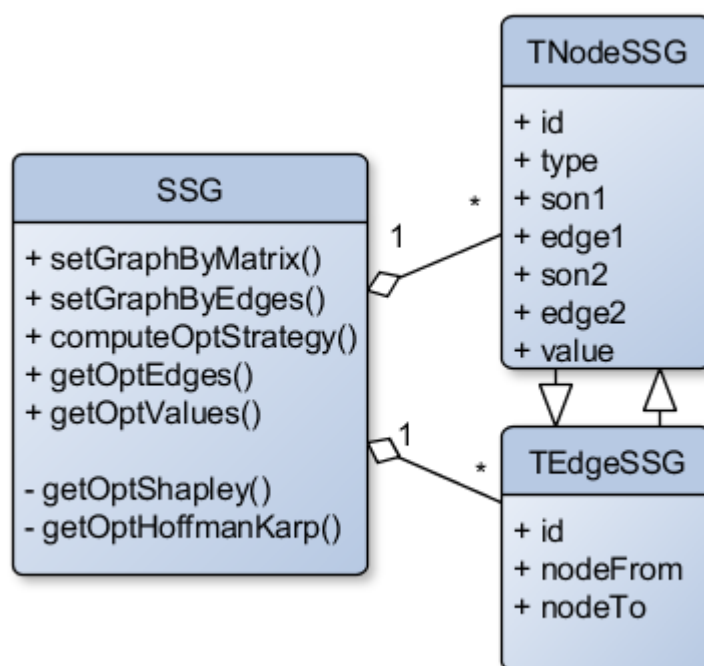
4.2 Návrh

Knihovna se bude skládat z jedné hlavní třídy, kterou nazveme SSG. Pro použití knihovny tedy bude stačit získat jediný objekt této třídy (třídu můžeme tedy implementovat jako návrhový vzor *Jedináček*). Třída SSG musí nějakým způsobem převzít graf, poté jej převést na SSG graf, následně vypočítat výsledek a vrátit jak výslednou *optimální strategii*, tak i *hodnoty* všech vrcholů. Těchto kroků je hodně a v každém může nastat nějaká chyba. Např. při převodu na SSG graf bychom mohli zjistit, že některý z Max uzlů nemá následníka (každý uzel musí mít cestu v grafu do nějakého Sink uzlu). Z tohoto důvodu tedy výpočet nebude probíhat srkze jedinou metodu, ale bude rozdělen na následující čtyři metody, které můžeme nazvat rozhraním:

4. FRAMEWORK PRO ŘEŠENÍ HER SE DVĚMA A PŮL HRÁČE

- `setGraph()` – V parametrech této metody bude graf, který se uvnitř metody transformuje na SSG graf. Vytvoří se zde datové struktury, se kterými bude v dalších metodách probíhat výpočet.
- `computeOptStrategy()` – Pomocí této metody se zavolá výpočet optimálních strategií na graf zadaný předchozí metodou. Který algoritmus bude vybrán, je zvoleno v parametru metody.
- `getOptEdges()` – Vrátí nám *optimální strategie*. Jinak řečeno, ze kterého vrcholu do kterého povede hrana.
- `getOptValues()` – Vrátí nám *optimální hodnoty* všech vrcholů.

Zmíněné metody musí být striktně volány v pořadí, jako jsou zde uvedené (nelze volat metodu na vrácení hodnot, když předtím neproběhl výpočet).



Obrázek 4.1: Doménový model knihovny SSG

Na obrázku 4.1 můžeme vidět doménový diagram knihovny SSG. Obsahuje pouze nejdůležitější metody a atributy. Třída TNodeSSG znázorňuje uzel v SSG grafu a třída TEdgeSSG znázorňuje hranu v grafu. Metoda `setGraph()` byla rozdělena na dvě možnosti. Zaprvé můžeme předat graf pomocí přechodové matice (`setGraphByMatrix()`). Druhou možností je, že ho zadáme pomocí hran (`setGraphByEdges()`).

4.3 Implementace

Framework je naimplementován v čistém jazyce C++, aby šel zapojit do SW projektů na různých platformách a v různých vývojových prostředích, jako je např. Microsoft Visual Studio nebo QT.

4.3.1 Použité knihovny

Lp Solve

K implementaci *Hoffman–Karp* algoritmu je zapotřebí lineárního programování k vyřešení optimačního problému (alg. 2). Základní frameworky jazyka C++ žádný řešič těchto úloh neobsahují, proto jsem hledal jinou volně šiřitelnou knihovnu, kterou bych mohl použít. Vše splnila knihovna Lp Solve. Obsahuje jak grafické prostředí pro řešení lineárních omezení, tak i C++ knihovnu, kterou mohu použít ve zdrojovém kódu.

Eigen

Eigen je volně šiřitelná knihovna jazyka C++ pro lineární algebru. Tedy pro matice, vektory atd. V našem programu potřebujeme násobit matice pro výpočet vektoru \bar{v} v algoritmu *Hoffman–Karp*, proto zde využívám tuto knihovnu.

4.3.2 Vložení grafu a jeho transformace na SSG

Předpoklad pro vložení grafu je, že máme uzly označeny indexy od 0 do $n-1$, kde n je celkový počet uzlů v grafu.

Jak jsem již zmínil, graf lze předat do knihovny SSG dvojím způsobem. První je přes přechodovou matici s následujícími parametry:

`setGraphByMatrix(BMatrix matrix, vector<int> max, vector<int> min, vector<int> ave, TSinks sinks)`. `Int` je typ `Integer`. `Vector` je speciální kontejner s předem neznámou velikostí, který používá paměť z haldy. Nazýváme ho jednoduše pole. `BMatrix` je uživatelsky definovaný typ pro „`Vector Vectorů`“ typu `boolean` – je to tedy matice. Pokud je v grafu hrana z `i`-tého do `j`-tého uzlu, tak matice na pozici `[i,j]` bude `True`. Další parametry – pole integerů `max`, `min` a `ave` nám říkají pomocí indexů, které uzly v grafu jsou typu `Max`, `Min` a `Ave`. Pokud tedy např. pole `max` bude obsahovat čísla `1,4,5`, tak víme, že uzly s indexy `1,4,5` jsou typu `Max`. Posledním parametrem je pole struktur typu `TSink`. Tato struktura obsahuje dvojici atributů – `id` a hodnotu. Když máme třeba strukturu s `id` 2 a hodnotou `0.5` tak to znamená, že v grafu je uzel s indexem 2 typu `Sink` a má hodnotu `0.5`. Všimněte si, že v parametrech není zadáno, kolik je uzlů v grafu. Tento údaj totiž můžeme zjistit jednoduše podle velikosti matice. Musí platit, že počet prvků v polích `max`, `min`, `ave` a `sinks` musí být roven počtu řádků v matici.

Protože zadávání grafu přes matici je někdy zbytečně paměťově náročné – většinou to je tzv. řídká matice, tak vznikla ještě jedna možnost, kterou je pole hran: `setGraphByEdges(int numNodes, vector<StructEdge> edges, vector<int> max, vector<int> min, vector<int> ave, TSinks sinks)`. První parametr `numNodes` udává počet uzlů v grafu. Druhý parametr `edges` je pole struktur. Struktura představuje hranu. Obsahuje dvě čísla, kdy první znamená z kterého indexu hrana pochází a druhé do kterého směřuje. Např. hrana s čísly `1,5` udává, že v grafu je hrana z vrcholu s indexem 1 do vrcholu s indexem 5. Pole `max`, `min`, `ave` a `sinks` se neliší od předchozí metody.

Uvnitř dvou výše zmíněných metod probíhá převod na nové datové struktury, ze kterých se později bude počítat optimální strategie. Při tomto převodu se graf rovnou převádí na *SSG* graf (ve kterém má každý uzel přesně dvě výstupní hrany) pomocí transformací popsaných v kapitole 5.4.

4.3.3 Výpočet optimální strategie

Výpočet optimální strategie může být proveden dvěma způsoby – *randomizovaným Hoffman–Karp* algoritmem a *Shapleyho* algoritmem.

Randomizovaný Hoffman–Karp algoritmus je sestrojen v metodě

`getOptHoffmanKarp()` podle alg. popsaném v kapitole 1. Jako náhodnou strategii zvolí vždy první hranu vycházející z uzlu. Když máme strategie Min a Max, tak musíme vypočítat vektor hodnot. Tento vektor vypočítáme podle rovnice 2.1. V této rovnici se používá maticové násobení, proto zde využijeme již zmíněnou knihovnu pro počítání matic Eigen. Metoda `isStableHoffman` zjistí, zda je vypočtený vektor hodnot optimální/stabilní podle definice 2.3.4. Zároveň se v této metodě počítají *přepínatelné* hrany. Vybrání náhodné množiny obsahující tyto přepínatelné hrany probíhá tak, že se všechny prochází a každá má 50% pravděpodobnost, že bude v množině. Nalezení optimální minimální strategie k dané maximální strategii probíhá v metodě `findOptMinStrategyToMax`. K výpočtu je použito lineární programování podle omezení popsaných v alg. 2. Lineární programování se provádí skrze knihovnu `Lp Solve`.

Shapleyho algoritmus je sestaven v metodě `getOptShapley()`. Výpočet optimálních hodnot probíhá přesně podle předpisového algoritmu 3. Pouze na začátku při inicializaci nejsou známy hodnoty Ave uzlů, proto jim dáme hodnotu 0.5 a pak je znovu inicializujeme již podle předpisového vztahu. Metoda `isStableShapley()` zjistí, zda je vypočtený vektor hodnot optimální/stabilní. Tuto metodu nemůžeme použít stejnou jako u *Hoffman–Karp* algoritmu, protože tam jsme zároveň počítali *přepínatelné* hrany.

4.3.4 Vrácení výsledků

Pokud jsme zavolali metodu pro výpočet optimální strategie a proběhla úspěšně, tak nyní můžeme požádat o výsledky. Pro vrácení optimální strategie slouží metoda `getOptEdges`. Vrácením je pole čísel. Index v tomto poli udává, o který vrchol se jedná. Číslo v poli je vrchol, do kterého optimální hrana jde. Pokud je číslo -1, tak optimální hrana nejde nikam (jedná se o Sink nebo Ave uzel). Lépe to vysvětlíme na příkladu: je nám vráceno pole obsahující čísla 1,-1,0. Znamená to tedy, že uzel s indexem 0 má optimální hranu do uzlu s indexem 1. Uzel s indexem 1 nemá žádnou optimální hranu a uzel s indexem 2 má optimální hranu do vrcholu s indexem 0.

Stejným způsobem probíhá vrácení optimálních hodnot. Zavoláním metody `getOptValues` se vrátí pole desetinných čísel. Když nám např. dojde pole 0.5,0.1,1, tak víme, že vrchol s indexem 0 má opti-

4. FRAMEWORK PRO ŘEŠENÍ HER SE DVĚMA A PŮL HRÁČE

mální hodnotu 0.5, uzel s indexem optimální hodnotu 0.1 a nakonec vrchol s indexem 2 má optimální hodnotu 1.

5 Vývoj prototypu

V této kapitole se budeme zabývat vývojem prototypu, který bude využívat knihovnu SSG popsanou v předchozí kapitole. Prototyp bude program, který bude obsahovat dva druhy funkčnosti. První z nich bude v grafickém prostředí řešit graf *hry se dvěma a půl hráče*. Pomocí této funkčnosti pak budeme moci lépe otestovat knihovnu SSG. Druhou funkčností bude výpočet pravděpodobnosti výhry a určení strategie (kterou figurkou táhnout) u hry *Člověče, nezlob se*. Program nazveme GW2.5P¹. Vývoj programu bude rozdělen na čtyři části – shrnutí požadavků, návrh aplikace, implementace a testování.

5.1 Požadavky

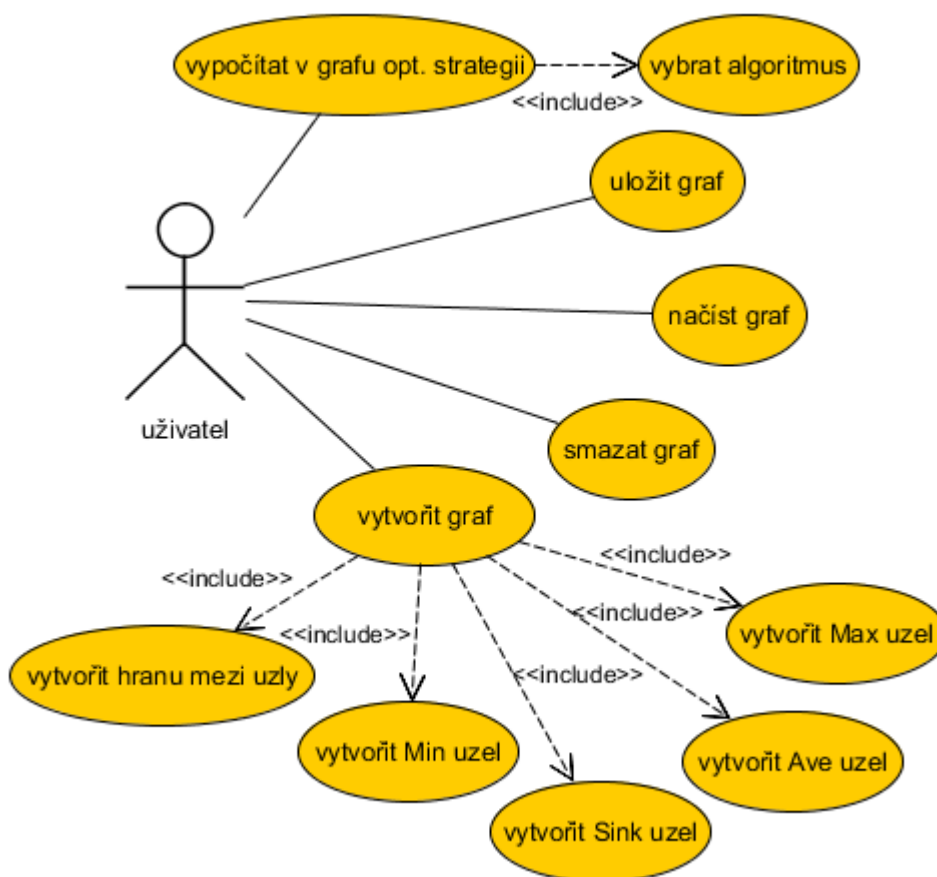
Chceme vytvořit aplikaci s grafickým uživatelským rozhraním. V programu půjde řešit *graf se dvěma a půl hráče* a také hra *Člověče, nezlob se*. Analýzu požadavků tedy rozdělíme na tyto dvě funkčnosti.

5.1.1 Řešič grafu se dvěma a půl hráči

Uživatel si bude moci vytvořit *graf se dvěma a půl hráče*. K dispozici budou čtyři druhy uzlů – Ave, Max, Min a Sink uzly. Tyto uzly by měly být od sebe barevně rozlišené. Při vytváření Sink uzlu se musí zadat i jeho hodnota. Všechny uzly pak můžeme propojovat hranami. U hrany musí být patrné, který uzel je počáteční a který koncový.

Až si uživatel navolí graf, tak klikne na tlačítko k vypočtení strategie a ta se vypočte. Lze volit mezi dvěma možnými algoritmy výpočtu. V grafu by se pak měly zvýraznit ty hrany, které jsou v optimální strategii. Zároveň u každého uzlu musí být napsána vypočtená optimální hodnota. Diagram případů užití, který shrnuje všechny požadavky, můžete vidět na obrázku 5.1.

1. GW2.5P je zkratka pro Game with two and half players



Obrázek 5.1: Diagram případů užití pro řešič grafu se dvěma a půl hráče

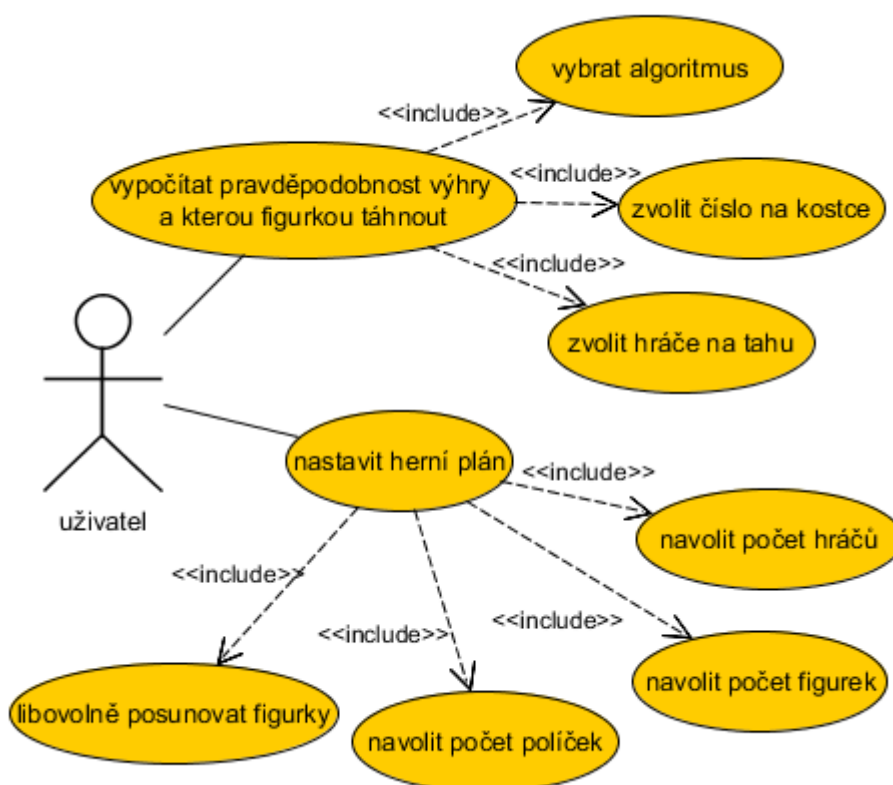
5.1.2 Hra Člověče, nezlob se

Program bude umět vypočítat pravděpodobnost výhry ve hře *Člověče, nezlob se*. Zároveň nám oznámí, kterou figurkou bude nejvýhodnější táhnout. Pravidla hry jsou popsána v následující podkapitole.

Uživatel si nejprve navolí hrací pole. Může zvolit počet hráčů, počet figurek a počet hracích políček. Hrací pole se mu následně objeví v grafickém prostředí. S figurkami se bude moci libovolně hýbat (např. jednu figurku necháme na startu a dvě dáme do cílového do-

mečku).

Po nastavení figurek a určení čísla na kostce, se může zahájit výpočet. Po vypočtení by uživatel měl vědět pravděpodobnosti výher, kdyby táhl jednotlivými figurkami. Figurku s největší pravděpodobností výhry je potřeba zvýraznit. Protože výpočet by mohl trvat delší dobu, je potřeba, aby program „nezamrzl“, ale dával uživateli informaci, v jakém je výpočet stavu. Diagram případů užití ukazuje obrázek 5.2.



Obrázek 5.2: Diagram případů užití pro hru Člověče, nezlob se

Pravidla

Hra *Člověče, nezlob se* je známá po celém světě. (v angl. jazyce je pro ni název Ludo). Princip hry je všude stejný, ale pravidla se různě poně-

kud liší. Řekneme si tedy pravidla, podle kterých ji budeme hrát my – aby jsme hru mohli převést na adekvátní *graf se dvěma a půl hráče*.

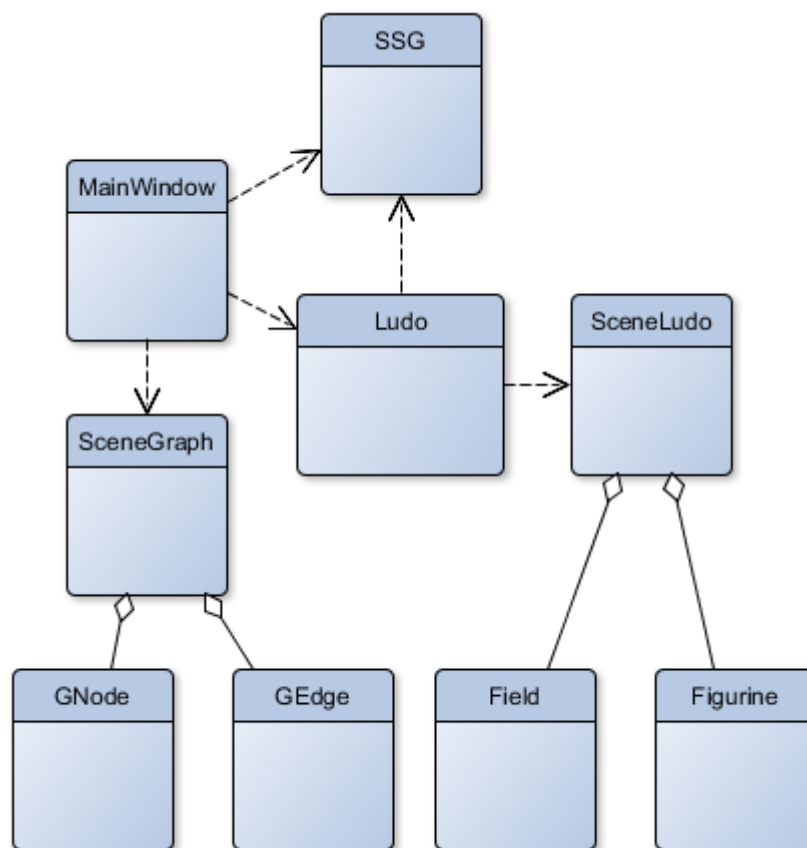
Hru hrají 2 – 4 hráči. Každý z nich má 2 – 4 figurky. Figurky jsou na začátku hry umístěny ve startovních domečcích. Vyhraje ten hráč, který dostane jako první všechny své figurky do cílového domečku. Hráč, který je na tahu, hodí hrací kostkou a podle čísla, které mu padne, může popojet s libovolnou nasazenou figurkou po hracím pláň. Pokud padne šestka, tak může prvně popojet a hází ještě jednou. Aby hráč dostal figurku ze startovního domečku, tak mu musí padnout šestka. Když hráč popojede na políčko, na kterém je figurka soupeře, tak ji vyhodí (je vrácena zpět do jeho startovního domečku). Není dovoleno mít dvě své figurky na stejném políčku. V cílovém domečku může figurka ještě popojíždět, dokud je před ní místo. Pokud hráč, který je na tahu, nemůže popojet žádnou figurkou, tak hraje soupeř.

5.2 Návrh

Program navrhne tak, aby v něm šel řešit *graf se dvěma a půl hráče* a hra *Člověče, nezlob se*. Obě tyto funkčnosti budou obsahovat bílou scénu, na které bude grafika. Mezi těmito dvěma scénami a tedy hrami se bude moci přepínat pomocí záložek. Třída `MainWindow` bude představovat hlavní okno celé aplikace a zahrnovat všechny uživatelem vygenerované události (např. kliknutí na tlačítko). Zároveň se z této třídy budou volat všechny důležité metody. Doménový model s návrhem tříd můžete vidět na obrázku 5.3. Všechny ostatní třídy z modelu zmíníme v další části textu.

5.2.1 Návrh řešiče her se dvěma a půl hráče

Základem bude bílá scéna, na které se bude kreslit graf. Nalevo od scény bude šest přepínacích tlačítek, které nám oznamují, jaké úkony s grafem můžeme právě dělat. Když bude aktivní tlačítko `Max`, tak se do scény po kliknutí vytvoří `Max` uzel. Stejným způsobem budou fungovat tlačítka `Min` a `Ave`. Aktivním tlačítkem `Sink` budeme kreslit `Sink` uzly. Ještě před jejich vytvořením však uživateli vyskočí okénko, kde zadá jeho hodnotu. Teprve poté se vytvoří uzel. Pokud bude zaškrtnuté tlačítko `Edge`, tak můžeme kreslit hrany mezi uzly. Poslední



Obrázek 5.3: Doménový model pro program GW2.5P

bude tlačítko Select. S tímto tlačítkem můžeme označovat, posunovat a mazat uzly v našem grafu.

Na pravé straně od scény bude přepínač, kterým si zvolíme algoritmus pro výpočet optimální strategie. Pod ním pak bude tlačítko, kterým se zmíněná optimální strategie spočítá.

Scénu pro graf implementuje třída SceneGraph. Uzly v grafu představuje třída GNode a hrany třída GEdge.

5.2.2 Návrh hry Člověče, nezlob se

Základem hry bude opět bílá scéna. Pomocí měničů si navolíme počet hráčů, počet figurek a počet prostředních políček. Na scéně se po kliknutí na tlačítko vytvoří herní plán s figurkami. Figurky můžeme tažením posouvat na libovolná políčka na herním plánu. Hra bude obsahovat měnicí box pro číslo udávající, kolik padlo na kostce a také box pro hráče na tahu. Pro hráče na tahu se bude zároveň počítat vítězná strategie. Kliknutím na tlačítko se začne počítat optimální strategie a vítězná pravděpodobnost.

Hru Člověče, nezlob se implementuje třída `Ludo`. V této třídě probíhá transformace hry na graf a následné zavolání metody knihovny SSG pro výpočet optimální strategie. Grafickou scénu hry představuje třída `SceneLudo`. Na scéně se vytváří políčka, které jsou objekty třídy `Field`, a také figurky, které jsou objekty třídy `Figure`.

5.2.3 Převod hry Člověče, nezlob se na graf

V této podkapitole popíšeme jednu z nejdůležitějších částí programu, kterou je transformace hry *Člověče, nezlob se na graf se dvěma a půl hráče*.

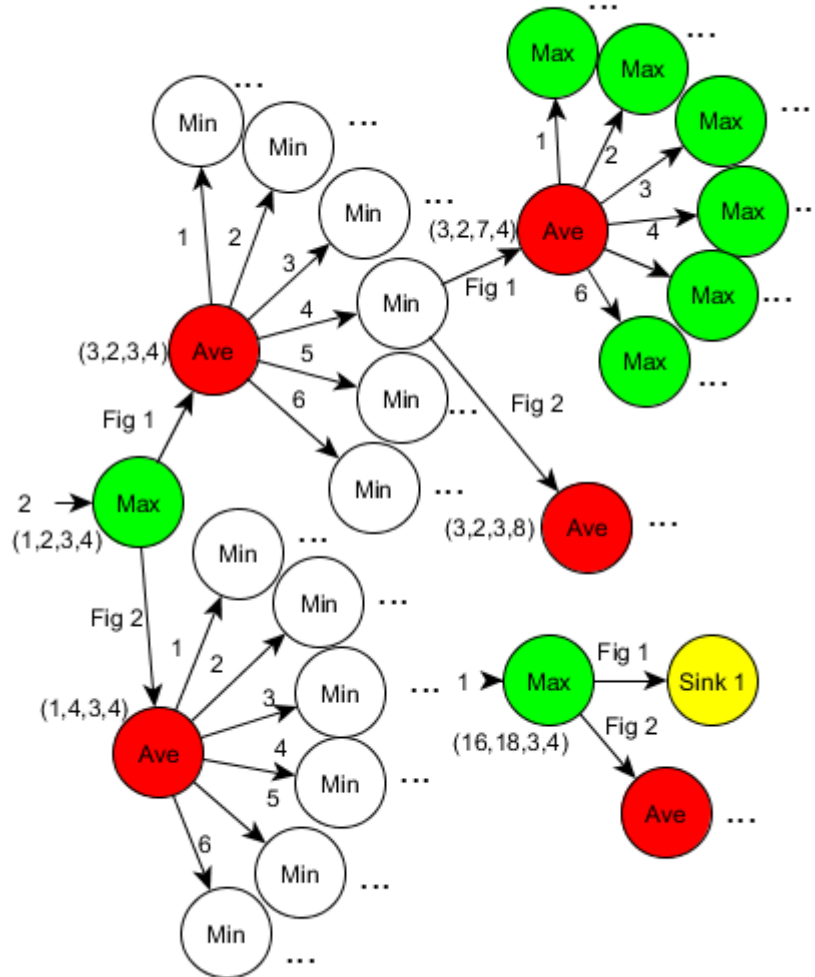
Uzly v grafu budou představovat stav, ve kterém se právě hra nachází. I když hru mohou hrát až čtyři hráči, tak nám stačí dva hráči v grafu – jeden maximální a jeden minimální. Hráč, za kterého hrajeme my, je maximální a ostatní hráči, obrazně řečeno, hrají za minimálního. Půl hráč pak představuje hození kostky – kdy nám padne náhodné číslo od 1 do 6. Stav hry je zaznamenán pomocí rozložení figurek. V každém uzlu grafu bude uloženo umístění všech figurek ve hře pomocí pole čísel. Např. pokud máme dva hráče, kteří mají dvě figurky, tak v uzlu bude čtveřice čísel – první číslo udává polohu figurky 1, druhé udává polohu figurky 2 atd. Poloha figurky je určena pomocí ID políčka, na kterém se nachází. Na startu jsou políčka s ID 0. Prostřední políčka mají ID od 1 do N , kde N je počet prostředních políček. Cílová políčka pak mají ID od N do $(N + \text{kolik je figurek})$.

Z každého MAX nebo MIN uzlu bude vycházet tolik hran, kolik má hráč figurek. Tato hrana bude představovat figurku, se kterou se táhne. Uzly Max a Min obsahují také číslo sdělující, kolik padlo na kostce. Na začátku každého grafu je tedy počáteční uzel Max. Tento uzel má v sobě čísla udávající rozložení figurek a také číslo, které

padlo na kostce. Z něho vystupuje tolik hran, kolik má hráč figurek. Hrany vedou do Ave (pravděpodobnostního) uzlu. Tento uzel představuje hození kostky. Má tedy šest možných vystupujících hran, z nichž každá představuje padlé číslo na kostce. Ave uzel obsahuje také rozložení figurek a informaci o tom, který hráč je na tahu. Nyní hraje hráč Min (pokud tedy v předchozím pokusu nepadla šestka), takže Ave uzel má šest následníků uzlů Min. Z každého Min uzlu poté vedou opět dva Ave uzly, jakožto možné dvě figurky. Celý proces se takto opakuje, dokud nějaký hráč nemá všechny figurky v domečku. V tomto případě je poslední uzel Sink. Pokud má figurky v cíli hráč Max, tak se jedná o 1-Sink (výhra), pokud má v cíli hráč Min, tak se jedná o 0-Sink (prohra). Graf končí vždy některým ze Sink uzlů.

Při hře je možné se dostat do stavu, ve kterém už jsme někdy byli. Např. když máme všechny figurky na startu a několik kol se nám nepodaří hodit šestku, tak se stav hry, ve kterém se nachází hráč Max, nemění. Z tohoto důvodu, když budeme vytvářet Ave uzel, tak se vždy podíváme jestli už neexistuje totožný Ave uzel (se stejným rozložením figurek a hráčem na tahu). V případě že ano, tak nasměrujeme hranu do tohoto Ave uzlu a nevytváříme nový.

Na obrázku 5.4 můžeme vidět začátek grafu, na který se transformovala hra se dvěma hráči se dvěma figurkami a 16 prostředními políčky. První hráč má figurky na pozicích 1 a 2, druhý hráč má figurky na pozicích 3 a 4 (na obrázku je rozložení figurek v závorce). Prvnímu hráči padlo číslo 2. Figurkou 1 se tedy dostaneme do stavu (3,2,3,4) – popojeli jsme figurkou o dvě políčka. Na stejném obrázku v pravém dolním rohu je ukázka závěru hry, kdy hráč hodí číslo 1 a figurkou 1 se dostane do domečku. Protože už druhou figurku v domečku má, tak vyhrává a je zde 1-Sink uzel.



Obrázek 5.4: Ukázka transformace hry Člověče, nezlob se na graf

5.3 Implementace

V této části popíšeme programovou realizaci aplikace a použité nástroje. Při implementaci aplikace se vycházelo z návrhu aplikace.

5.3.1 Vývojové prostředí QT

Jako vývojové prostředí pro naši aplikaci jsem zvolil knihovnu QT² vyvíjenou firmou Digia. QT je multiplatformní nástroj pro vyvíjení softwarových projektů s grafickým uživatelským rozhraním. Aplikace jsou vyvíjeny s nativním vzhledem, takže se vždy přizpůsobí operačnímu systému. QT obsahuje knihovny pro více jazyků (C++, Python, Ruby, Pascal). Já jsem si vybral objektově orientovaný jazyk C++. Tuto vývojovou knihovnu jsem zvolil z důvodu velmi přívětivého prostředí, zejména pak kvůli velké a přehledné dokumentaci. Knihovna QT obsahuje důležitou funkčnost, kterou jsou signály a sloty. Tyto zvláštní metody slouží ke komunikaci mezi objekty. Používají se například při kliknutí na tlačítko – vyšle se signál. Slot je pak metoda, která se provede v reakci na signál – obsluha po stisknutí tlačítka.

Knihovnu jsem použil v nejnovější verzi 5.6.0. Provedl jsem také statickou kompilaci zdrojových kódů QT, takže výsledkem implementace je pouze jediný spustitelný soubor, který má okolo 20 MB. Kdybych nepoužil statickou kompilaci, tak by ke spuštění programu byly vyžadovány knihovny QT, a program by tak zabral o hodně více místa na disku.

Nástroj je dostupný pod LGPL veřejnou licenci.

5.3.2 Struktura aplikace

Hlavní třídou je `MainWindow`. V této třídě se obsluhují kliknutí na veškerá tlačítka v aplikaci.

Scénu řešiče grafu se dvěma a půl hráče implementuje třída `SceneGraph`. Tato třída dědí z `QGraphicsScene`, což je speciální třída knihovny QT pro kreslení různých objektů. Do scény se přidávají uzly – objekty třídy `GNode`, které dědí z `QGraphicsEllipseItem`, která představuje grafickou elipsu (kruh). Uzly v uvnitř sebe obsahují seznam všech vstupujících a vycházejících hran. Hrana mezi uzly je objekt třídy `GEdge`, jenž dědí z `QGraphicsLineItem`, která implementuje grafickou přímku ve scéně. Všechny uzly se přidávají do pole. Když se klikne na tlačítko k vypočtení strategie, tak se přes všechny uzly vytvoří matice přechodů a ta se pošle třídě `SSG`, která výsledek vypočte.

2. Viz <https://www.qt.io/>

Pro hru *Člověče, nezlob se* je vytvořena speciální třída `Ludo`. V této třídě probíhají veškeré úkony týkající se hry, zejména pak převod hry na *graf se dvěma a půl hráče* popsany v kapitole 5.2.3. Scénu představuje třída `SceneLudo` dědicí z třídy `QGraphicsScene`. Na scénu se přidávají políčka, což jsou objekty třídy `TFields`, které jakožto kruhy dědí opět z `QGraphicsEllipseItem`. Kromě políček jsou na scéně ještě figurky – objekty třídy `TFigurines`, které dědí z třídy `QGraphicsPixmapItem`. `QGraphicsPixmapItem` představuje libovolný obrázek, který můžeme vložit do scény.

Když probíhá výpočet strategie u hry *Člověče, nezlob se*, tak se zobrazuje okno informující o stavu výpočtu. Výpočet proto musí probíhat ve speciálním vlákně. Knihovna QT má pro vlákna speciální třídu `QThreads`. Vlákno pak může pomocí zpráv informovat okno o stavu – např. že se dokončil převod na graf.

Všechny třídy a metody programu můžete shlédnout v dokumentaci, která je v příloze.

5.3.3 Testování

Testování probíhalo již při implementaci, kdy byla každá nově přidaná část řádně otestována. V řešiči grafu se dvěma a půl hráče jsem testoval desítky grafů. Od obou algoritmů byly vždy stejné výsledky, což značí, že je výsledek správný.

Při testování hry *Člověče, nezlob se* jsem zjistil, že transformace hry na graf trvá příliš dlouho. Při vytváření grafu se totiž každý nově přidávaný *Ave* uzel porovnává se všemi již vytvořenými, jestli už některý z nich nemá stejnou polohu figurek. Časová náročnost tedy exponenciálně stoupá. Z tohoto důvodu jsem vytvořil hashovací tabulku pro první tři figurky. Když např. budeme mít polohu figurek (1, 2, 3, 4), tak uzel budeme hledat v hashovací tabulce na pozici [1][2][3]. Hashovací tabulka pouze pro tři figurky je z důvodu, že při spuštění programu není jisté, kolik figurek přesně bude (může jich být až 16 – při 4 hráčích a 4 figurkách). Implementace k rozměrného pole o n prvcích by totiž byla velice složitá, ne-li nemožná. Po této úpravě je převod hry na graf několikanásobně rychlejší.

6 Experimentování

Program GW2.5P, který jsme vytvořili, je potřeba otestovat a spustit na různých datech. V této kapitole uvedeme, jak program ovládat a také několik příkladů použití jak hry *Člověče, nezlob se*, tak řešiče grafu se dvěma a půl hráče.

6.1 Graf se dvěma a půl hráče

6.1.1 Ovládání

Po spuštění programu můžeme na horní liště vybírat ze dvou módů:

- mód pro řešení grafu hry se dvěma a půl hráče – lišta s nápisem Graph solver of game with two players.
- mód *Člověče, nezlob se* – lišta s nápisem Ludo.

Pro vytvoření grafu použijeme šest tlačítek na levé straně. Mezi tlačítka se lze přepínat po kliknutí na libovolné z nich. To tlačítko, které je aktivní, nám říká ve kterém stavu se nacházíme. Graf se kreslí do bílého prostoru – scény.

Tlačítka, a tedy i stavy, jsou následující:

- » s obrázkem šipky: V tomto stavu můžeme hýbat s uzly grafu a také kliknutím označit libovolný uzel či hranu. Při prodržení tlačítka CTRL můžeme označit více částí grafu najednou. Označit je můžeme i po tažení myši, kdy se nám tvoří výběrový obdélník. Pokud máme označený uzel nebo hranu, můžeme je smažat kliknutím klávesy DELETE. Samozřejmě se smažou i hrany vedoucí z těchto a do těchto uzlů.
- » s popisem MAX: Po kliknutí do scény se vytvoří MAX uzel, který má zelenou barvu.
- » s popisem MIN: Po kliknutí do scény se vytvoří MIN uzel, který má bílou barvu.

- » s popisem AVE: Po kliknutí do scény se vytvoří *AVE* uzel, který má červenou barvu.
- » s popisem SINK: Po kliknutí do scény se vytvoří *SINK* uzel, který má žlutou barvu. Zde jsme také vyzváni pro zadání hodnoty uzlu. *SINK* uzel může nabývat hodnot od 0 do 1 (1 = výhra, 0 = prohra).
- » s popisem EDGE: V tomto stavu se kreslí hrana z jednoho uzlu do druhého. Hranu kreslíme z počátečního uzlu táhnutím s přidržným levým tlačítkem myši do cílového uzlu.

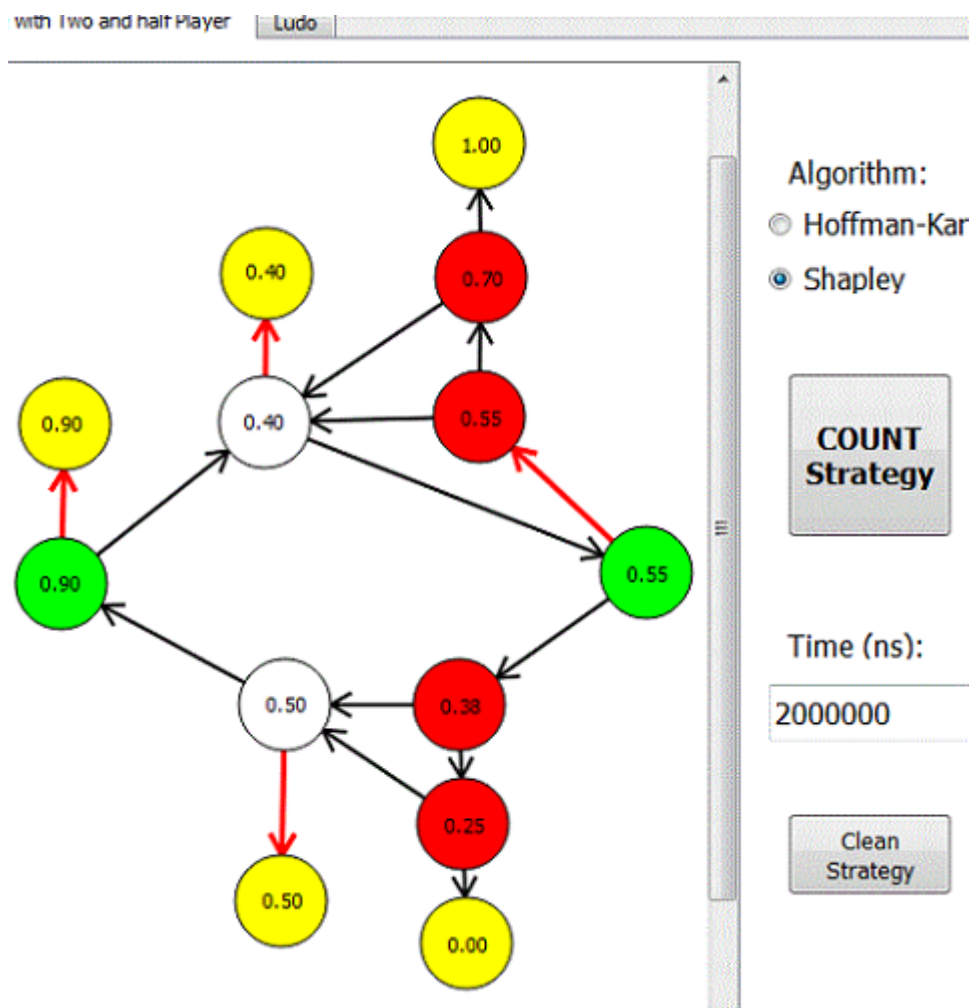
Pokud máme vytvořený graf, tak můžeme vypočítat strategii a hodnoty uzlů. To provedeme kliknutím na tlačítko *COUNT Strategy*. Ještě předtím si ale můžeme zvolit jednoduchým přepínačem jeden ze dvou algoritmů, pomocí něhož bude probíhat výpočet.

Výpočtený výsledek se zobrazí následovně. Hodnota uzlu se zobrazí uvnitř, stejně jako tomu je u *SINK* uzlu. Všechny hrany, které jsou obsaženy v optimální strategii, se zvýrazní červenou barvou. Čas, jak dlouho trval výpočet, je obsažen v kolonce s popisem *Time*. Výpočtené hodnoty a strategie lze smazat pomocí tlačítka *Clean Strategy*.

V menu programu *File* pak můžeme grafy ukládat a znovu je načítat. Grafy se ukládají ve formátu *XML*.

6.1.2 Příklad grafu

Tento příklad zároveň otestuje správnost našeho programu. Proto jsem vytvořil graf totožný jako v [6, s. 8], abychom mohli porovnat výsledky. Graf s vypočtenou strategií vidíte na obrázku 6.1. Pomocí *Shapleyho* algoritmu výpočet trval 2 000 000 ns, tedy cca. 2 ms. S algoritmem *Hoffman–Karp* to proběhlo o něco déle – cca. 53 ms. Výsledky od obou algoritmů jsou totožné a zároveň se také shodují s výsledky z [6, s. 8]. Správné hodnoty uzlů můžeme rozpoznat i pouhým pohledem. Každý *MAX* uzel obsahuje maximum ze svých synů. *MIN* uzel obsahuje naopak minimum. *AVE* uzel pak obsahuje součet hodnot synů podělený jejich počtem. Také zde můžeme vidět, že optimální strategie vede do uzlu, který má stejnou hodnotu.



Obrázek 6.1: Vytvořený graf v programu GW2. 5P

6.2 Člověče, nezlob se

6.2.1 Ovládání

Ke hře *Člověče, nezlob se* se dostaneme, pokud po spuštění programu na horní liště zvolíme Ludo.

Hru musíme nejdříve nastavit pomocí měničů po levé straně:

- » **Players** – zde nastavíme počet hráčů ve hře, je možné nastavit 2 – 4 hráče.
- » **Figurines** – zde nastavíme počet figurek, které mají hráči k dispozici, je možné nastavit 2 – 4 figurky.
- » **Fields** – tímto nastavíme počet políček na hlavním hracím poli, je možné nastavit 4 – 52 políček.

Po nastavení hry klikneme na tlačítko SET NEW GAME PLAN, čímž se nám nastaví hrací plán s figurkami. Tažením s přidrženým levým tlačítkem myši můžeme přesunovat figurky na libovolná místa na hracím plánu, kromě daných omezení, kdy např. nepůjde přesunout modrá figurka do zeleného domečku.

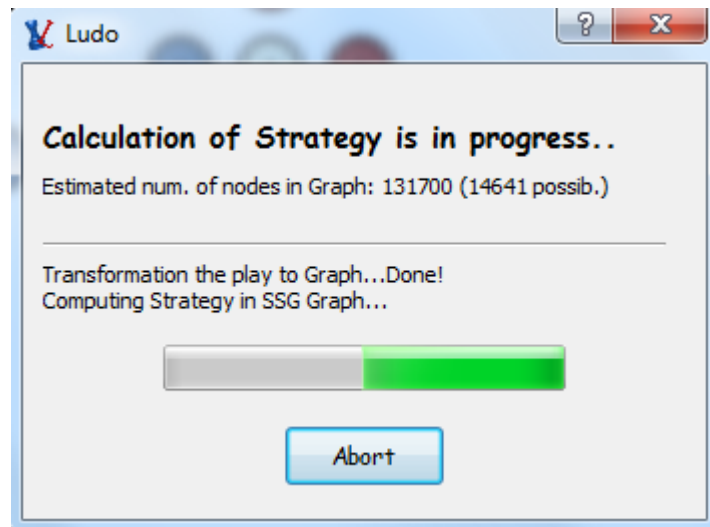
Na pravé straně nastavíme, který hráč je na tahu¹ a jaké číslo padlo na kostce. Nejprve zde byla možnost také zvolit i algoritmus výpočtu, ale po důkladném otestování jsem zjistil, že algoritmus *Hoffman-Karp* je nepoužitelný. Používá totiž maticové násobení, které je velice časově i paměťově náročné a již při nejmenší možné hře – tedy dvě figurky a dva hráči, by výpočet trval několik hodin. Proto se v této hře používá pouze *Shapleyho* algoritmus.

Kliknutím na tlačítko COUNT Strategy se začne počítat pravděpodobnost výhry hráče, který je na tahu a také optimální strategie - tedy kterou figurkou je nejvýhodnější táhnout.

Při výpočtu nás informační okno informuje o jeho průběhu, jak můžete vidět na obrázku 6.2. Výpočet můžeme kdykoli zrušit kliknutím na tlačítko Abort.

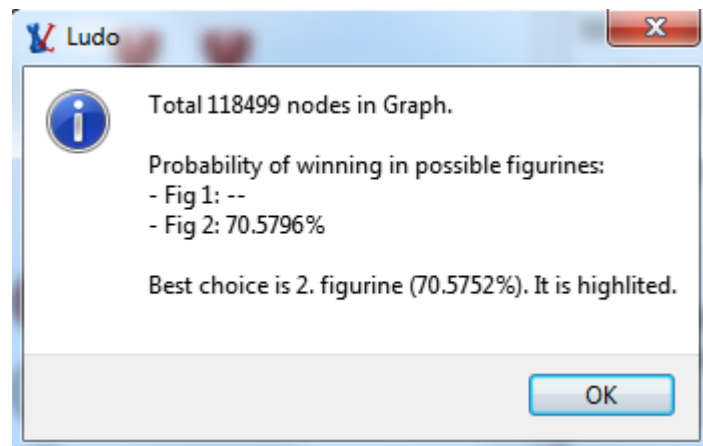
Odhadovaný počet uzlů (Estimated num. of nodes in Graph) bere v úvahu i to, na kterém místě se figurky nachází. V závorce za odhadovaným počtem uzlů je číslo, které nám říká, kolik kombinací figurek může nastat. Více v 6.2.2. Pokud odhadovaný počet přesáhne 5 000 000 uzlů, zobrazí se varování. Při tomto počtu uzlů totiž program zabírá již téměř 1,5 GB fyzické paměti a mohl by tedy spadnout kvůli jejímu nedostatku.

1. Pro tohoto hráče se počítá i pravděpodobnost výhry



Obrázek 6.2: Okno v průběhu výpočtu hry Človeče, nezlob se.

Když se výpočet dokončí, tak se nám zobrazí ještě jedno okénko s výsledky (viz obr. 6.3). Nemohu-li figurkou v aktuálním tahu táhnout, jsou u ní pomlčky. Figurka, která má největší pravděpodobnost výhry, se zvýrazní. Po pravé straně programu se nám také zobrazí výsledná pravděpodobnost výhry a celkový čas výpočtu.



Obrázek 6.3: Okno po ukončení výpočtu hry Človeče, nezlob se.

6.2.2 Možné kombinace figurek

Délka výpočtu optimální strategie je přímoúměrná počtu uzlů v grafu, na který se hra transformuje. Graf zahrnuje všechny možné případy, do kterých se hra může dostat. Aby jsme shruba věděli, jak dlouho bude výpočet trvat a zda počítač obsahuje dostatečné množství fyzické paměti, tak program spočítá, kolik kombinací figurek může nastat.

Pokud je některá figurka v cílovém domečku, tak již nemůže být vyhozena, proto se nám mnohonásobně zmenší počet možných kombinací.

Pro lepší vysvětlení uvedeme příklad: mějme dva hráče, každý se dvěma figurkami a 8 prostředních políček. Jedna figurka je v cílovém domečku, ostatní 3 jsou někde v poli. Figurky, které nejsou v cíli, mohou být v domečku, dále na 8 prostředních políčkách a nakonec na 2 políčkách v cíli. Celkem se tedy jedna figurka může v průběhu hry objevit na 11 políčkách. V našem případě je tedy kombinace následující²: $1 \cdot 11 \cdot 11 \cdot 11 = 1331$ možností rozestavení figurek v průběhu hry. Výsledek není úplně přesný, protože dvě figurky nemohou být na stejném políčku, ale nám to k hrubému odhadu stačí.

6.2.3 Příklad 1

V naší práci si ukážeme celkem tři příklady použití hry Člověče, nezlob se. Bohužel neexistuje možnost zjistit, jestli jsou naše vypočtené pravděpodobnosti správné. Hra se transformuje na *Graf se dvěma a půl hráči*, který může obsahovat několik tisíc, někdy i miliónů uzlů. Musíme se tedy spoléhat na svou logickou úvahu. Např. na začátku hry se dvěma hráči by asi každý očekával, že pravděpodobnost výhry je kolem 50%.

V našem prvním příkladě zkusíme tedy zjistit pravděpodobnost výhry na začátku hry, kdy jsou všechny figurky ještě v domečku. Budeme mít tři hráče se dvěma figurkami a čtyři prostřední políčka. Počet možných kombinací je tedy $7^6 = 117649$.

Rozložení figurek a výsledek můžete vidět na obr. 6.4. Pravděpodobnost výhry 32.97% je očekávaná. Máme totiž zhruba třetinovou šanci na výhru. Počet uzlů v *Grafu se dvěma a půl hráči* je 378668. Vý-

2. První číslo je 1, protože je figurka v cíli

počet trval 29 sekund. Více v tabulce 6.1.

The screenshot shows a Ludo game interface. At the top, there are tabs for "Two and half Player" and "Ludo". The game board is visible, showing pieces for Blue, Red, and Green. The Blue player has two pieces on the board, Red has two, and Green has two. The dice is currently showing a 3. The control panel on the right includes the following elements:

- Algorithm:** Hoffman-Kar (unselected), Shapley (selected).
- Who is on the turn:** Blue (selected).
- On the dice is:** 3.
- COUNT Strategy** button.
- Winning probability (%):** 32.9754.
- Time (sec):** 29.0480.
- Clean Strategy** button.

Obrázek 6.4: Příklad 1 použití hry Človeče, nezlob se.

6.2.4 Příklad 2

V tomto příkladu vyzkoušíme, zda je lepší vyhodit soupeřovu figurku, nebo raději s druhou svojí figurkou zajet do domečku. Máme dva hráče a každý z nich má tři figurky. Dvě figurky jsou ale v domečku, přičemž jedna je v domečku na prvním místě, takže se může ještě posunout dopředu (má tedy tři možnosti). Prostředních políček je dvanáct. Počet možných kombinací je tedy $3 \cdot 16 \cdot 16 \cdot 1 \cdot 16 \cdot 16 = 196608$. Výsledek vidíte na obr. 6.5. Výhodnější je vyhodit soupeřovu figurku – to máme pravděpodobnost výhry 67.32%. Kdybychom táhli druhou figurkou, tak máme pravděpodobnost výhry pouze 53%.

Celkový počet uzlů v grafu je 1 436 431. Takto velké množství uzlů se projevilo i na spotřebované fyzické paměti. Program během výpočtu potřeboval 740 MB. To už je opravdu velké číslo. Zvláště pokud víme, že běžný OS Windows povoluje maximálně okolo 1900 MB paměti pro jeden program.

6.2.5 Příklad 3

V posledním příkladě zkusíme nastavit všechny čtyři hráče a čtyři figurky. Rozestavení volíme tak, aby program nespádl kvůli nedostatku paměti. Proto u každého hráče dáme tři figurky do cílového domečku. Jednu figurku v cíli ale necháme pohnutelnou, u černého hráče pak dvě pohnutelné. Rozestavení figurek vidíte na obr. 6.6. Počet možných kombinací, když vynecháme sedm nepohnutelných figurek v domečku, je: $11 \cdot 2 \cdot 11 \cdot 2 \cdot 11 \cdot 2 \cdot 11 \cdot 3 \cdot 2 = 702768$.

Pravděpodobnost výhry je 21.9%. Pravděpodobnost druhé figurky, kterou můžeme popojet v cíli, je pouze nepatrně menší – 21.6%. Tento výsledek je také očekávaný, protože při čtyřech hráčích je na začátku hry pravděpodobnost asi 25%. Nyní jsou ovšem na řadě ale další hráči, kteří mi mohou figurku vyhodit, proto je pravděpodobnost nižší.

6.2.6 Shrnutí příkladů

Uvedené příklady nelze nijak přesněji ověřit, ale všechny dopadly podle původních očekávání. Rychlost výpočtu při tolika uzlech dopadla naopak nad očekávání dobře. Bohužel ale musíme dávat pozor, jak volíme figurky, jelikož již při třech hráčích je počet možných

kombinací tak velký, že počet uzlů přesáhne milión. Program může mít nedostatek fyzické paměti a spadnout.

V tabulce 6.1 jsou shrnuty výsledky a měření všech tří příkladů.

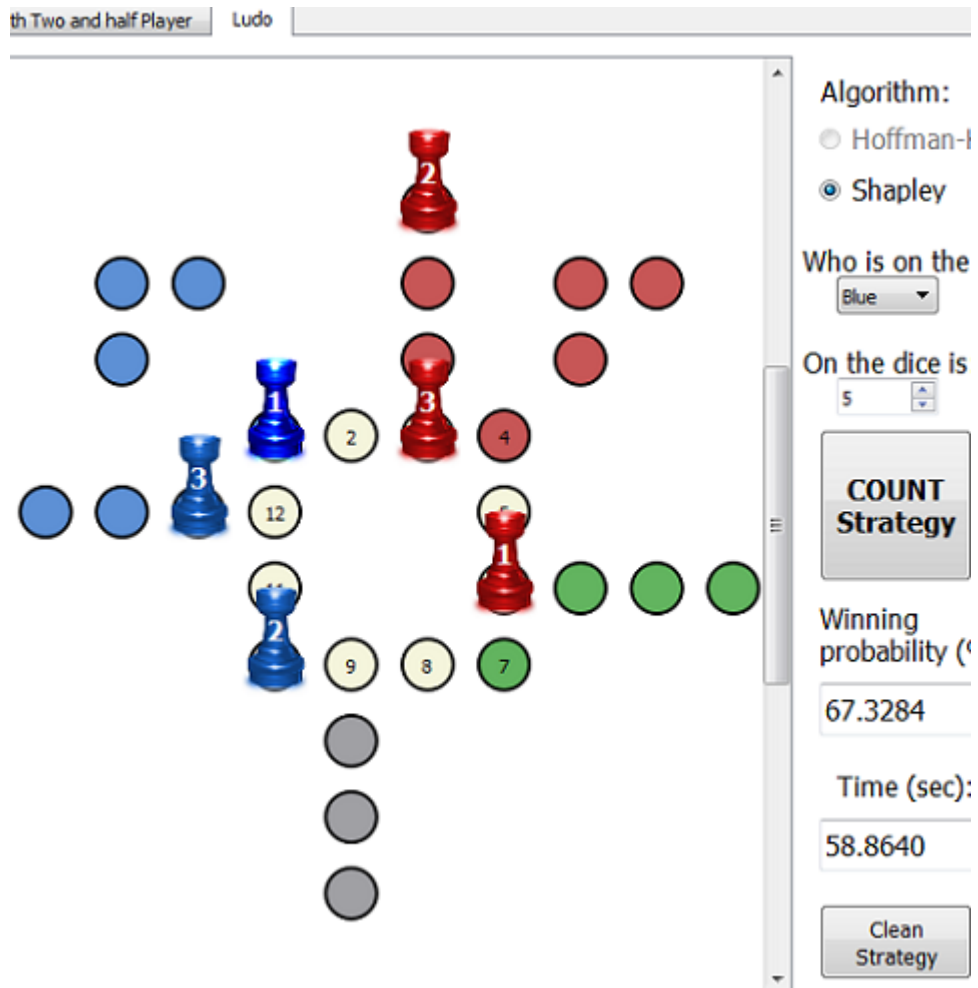
	Výhra ³	Počet uzlů ⁴	Délka výpočtu	RAM ⁵
Příklad 1	32.97%	117 649	29 sek.	170 MB
Příklad 2	67.33%	1 436 431	58 sek.	740 MB
Příklad 3	21.9%	2 542 725	1097 sek.	1266 MB

Tabulka 6.1: Shrnutí příkladů hry Člověče, nezlob se

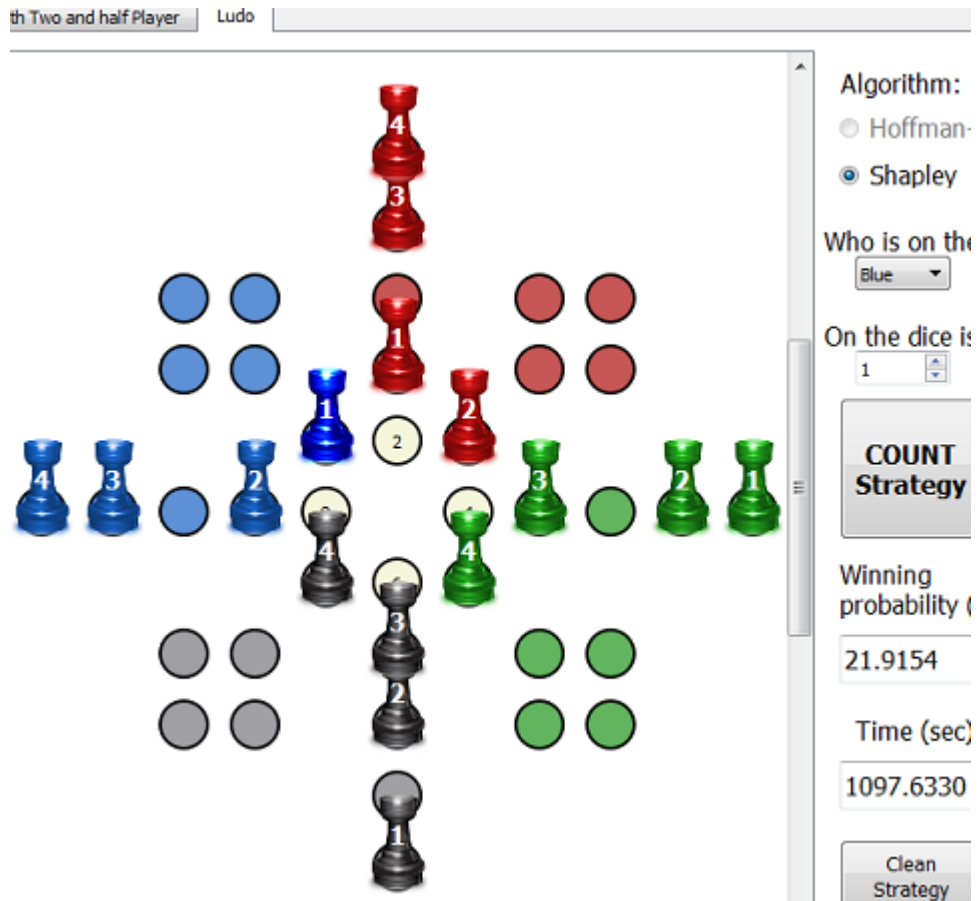
3. Pravděpodobnost výhry pro hráče na tahu

4. Počet uzlů po transformaci v *Grafu se dvěma a půl hráči*

5. Kolik zabíral program fyzické paměti v době výpočtu. Měřeno programem Process Explorer



Obrázek 6.5: Příklad 2 použití hry Človeče, nezlob se.



Obrázek 6.6: Příklad 3 použití hry Človeče, nezlob se.

7 Závěr

Cílem práce bylo nastudovat algoritmy pro řešení *her se dvěma a půl hráče*. Na základě nastudovaných algoritmů bylo úkolem vytvořit framework pro řešení těchto her. Součástí práce bylo také vytvořit funkční prototyp se hrou *Člověče, nezlob se* využívající vytvořenou knihovnu.

Při studiu jsem zjistil, že řešením *her se dvěma a půl hráči* se zabývají tzv. SSG (*simple stochastic games*). U těchto her je známo velké množství algoritmů pro jejich řešení. Vybral jsem dva algoritmy – *Randomizovaný Hoffman–Karp* a *Shapleyho*. Tyto algoritmy jsem naimplementoval do frameworku nazvaného SSG.

Navrhl jsem a naimplementoval také funkční program s názvem GW2.5P, který využívá vytvořený framework. Program má dvě hlavní funkčnosti. První je grafické řešení grafu *her se dvěma a půl hráče*, kde si uživatel může vytvořit graf a program mu vypočte optimální strategii. Druhou funkčností je hra *Člověče, nezlob se*. Hra se na základě navoleného hracího pole a figurek transformuje na SSG graf a dokáže tak spočítat pravděpodobnost výhry u vybraného hráče. Již při nejmenším počtu hráčů a figurek však graf obsahuje několik tisíc uzlů. Při větším počtu figurek, kdy uzlů bude několik milionů, může nastat problém s nedostatečnou fyzickou pamětí. Program ale dokáže odhadnout počet uzlů v grafu a uživatele před tímto možným scénářem varovat.

Knihovna SSG je plně funkční a je k dispozici pro různé projekty psané v jazyce C++ vyžadující spočítat optimální strategii ve *hrách se dvěma a půl hráče*.

Bibliografie

- [1] Martin Dlouhý a Petr Fiala. *Úvod do teorie her*. Oeconomica, 2007.
- [2] Theodore L. Turocy a B. Von Stengel. *Game Theory*. Tech. zpr. Texas A&M University, London School of Economics, 2001.
- [3] Krishnendu Chatterjee a Thomas A. Henzinger. "A survey of stochastic ω -regular games". In: *Journal of Computer and System Sciences* (2011).
- [4] Anne Condon. *The complexity of stochastic games*. Tech. zpr. Computer Sciences Department, University of Wisconsin-Madison, 1992.
- [5] Rafal Somla. "New algorithms for solving simple stochastic games." In: *Electronic Notes in Theoretical Computer Science* (2005).
- [6] Anne Condon. *On algorithms for simple stochastic games*. Tech. zpr. Computer Sciences Department, University of Wisconsin-Madison, 1993.
- [7] Elena Valkanova. "Algorithms for Simple Stochastic Games". Dipl. University of South Florida, 2009.

A Obsah příloženého CD

- thesis.zip – PDF a zdrojové texty diplomové práce
- src.zip – zdrojové soubory k programu GW2 . 5P
- bin.zip – spustitelný program GW2 . 5P
- doc.zip – dokumentace k programu GW2 . 5P