UNIVERZITA PARDUBICE
Fakulta elektrotechniky a informatiky

Optimální strategie hráče

Aleš Laňar

Bakalářská práce
2013

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2012/2013**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Aleš Laňar**
Osobní číslo:	**I09175**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Optimální strategie hráče**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování:

Cílem práce bude sestavit na základě principů teorie her program, který u vybrané deskové společenské hry poradí hráči v každém tahu optimální herní strategii.Vhodná hra musí mít u každého hráče konečný počet strategií, tak aby šlo sestavit výplatní matici hry. Může být zvolena např. hra Pilíře země, Pandemie, atd.

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

Maňas M., Teorie her a konflikt zájmů, Vysoká škola ekonomická v Praze, Nakladatelství Oeconomica, Praha 2002
Říha O., Základy teorie her, JČSMF, Praha 1973

Vedoucí bakalářské práce: **Mgr. Jaroslav Marek, Ph.D.**
Katedra matematiky a fyziky

Datum zadání bakalářské práce: **21. prosince 2012**
Termín odevzdání bakalářské práce: **10. května 2013**

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

Ing. Lukáš Čegan, Ph.D.
vedoucí katedry

V Pardubicích dne 29. března 2013

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 06. 05. 2013 Aleš Laňar

Poděkování

Chtěl bych poděkovat vedoucímu mé bakalářské práce Mgr. Josefu Markovi, Ph.D.
za veškerou pomoc při tvorbě této bakalářské práce a jeho cenné náměty a myšlenky.

Dále také chci poděkovat svým rodičům a mému nejbližšímu okolí za psychickou podporu.

Anotace

Cílem práce je seznámit čtenáře se základními principy teorie her. Po základní definici je čtenáři vysvětlena teorie kooperativních i nekooperativních her za pomoci názorných příkladů. Další část čtenáři vysvětlí a ukáže aplikaci konkrétních postupů na deskové kooperativní hře Pandemic s využitím dříve uvedených definic a postupů z teorie grafů, ale také s využitím aplikace. V poslední části bude čtenář seznámen s popisem, využitím i samotným vývojem aplikace.

Klíčová slova

Pandemic, teorie her, desková hra, kooperativní hra, programování, JAVA, NetBeans IDE, swing

Title

The optimal strategy for player.

Annotation

The goal is to familiarize the reader with the basic principles of game theory. Following a basic definition of game theory, the principles will be further defined by illustrated by cooperative and non-cooperative examples. The board game 'Pandemic' will be used to demonstrate the application and specific procedures endemic to game theory, using previously stated definitions and graph theory. The last section will further familiarize the reader with description, but also with development and application.

Keywords

JAVA, Pandemic, theory of games, board game, cooperative game, programming, swing, NetBeans

Obsah

Seznam zkratek

IDE	Integrated Development Environment
JAR	Java ARchiver
GUI	Graphical User Interface
OOP	Object Oriented Programming
JPEG	Joint Photographic Experts Group
UML	Unified Modeling Language
XML	Extensible Markup Language

Seznam obrázků

Seznam tabulek

Úvod

Teorie her je oborem aplikované matematiky, která se zabývá analýzou konfliktních rozhodovacích situací. Tyto situace lze aplikovat nejen na analýzu herních situací, ale také na běžný život, kdy se dostáváme před nutnost provést rozhodnutí. V takových chvílích, stejně jako samotná teorie her, i my provádíme analýzu jednotlivých možností. V některých momentech však rozhodovací proces není závislý pouze na nás jako na jednotlivci, ale také na ostatních účastnících a jejich pomoci. Tuto situaci simulují kooperativní hry více hráčů a tato problematika je tématem této práce.

Na popisu teorie her se nejvíce podílel John von Neumann (1903 – 1958), který položil základní kameny této disciplíny a je její významnou historickou osobností. V roce 1928 definoval základní pojmy teorie her a v roce 1944 spolu s ekonomem Otakarem Morgensternem (1902 – 1977) vydali první velké dílo o teorii her nazvané *Theory of Games and Economic Behavior* (Teorie her a ekonomického chování). O další posun teorie her se postaral matematik John Forbes Nash (nar. 1928), který rozšířil teorii her o definice kooperativních a nekooperativních her a objevil a definoval rovnováhy nekooperativních her. Za tento přínos obdržel Nobelovu cenu za ekonomii.

Jeho definice rovnováhy nekooperativních her je dodnes známá jako Nashova rovnováha (RASMUSEN, 2006).

Cílem této práce je seznámit čtenáře s vlastní definicí kooperativních her, přičemž mu bude vysvětlen i princip her nekooperativních, včetně aplikace teorie grafů na danou problematiku. Součástí práce je popis aplikace, která demonstruje možnost použití postupů z již zmíněné teorie grafů na konkrétní kooperativní hře.

V prvních čtyřech částech práce jsou popsány samotné definice základních pojmů teorie her a jejich klasifikace dle vlastností. Dále definice nekooperativních her a chování účastníků, důležité pro správné chápání celé problematiky. Hlavní částí je popis kooperativních her a rozhodování hráčů, včetně volby optimální strategie.

Pátá část popisuje samotnou analýzu konkrétní deskové kooperativní hry, a to jak v rovině teoretické, tak praktické.

Šestá část se zabývá popisem Floyd-Warshallova algoritmu včetně ukázky logiky pro jeho použití.

Sedmá část popisuje aplikaci, která simuluje samotný průběh hry a umožňuje zobrazení pravděpodobností nebo zjištění nejkratší možné cesty pomocí Floyd-Warshall algoritmu popsaný v (MIČKA, 2012).

1 Teorie her

Teorie her je matematická disciplína sloužící pro analýzu konfliktních rozhodovacích situací, ve kterých dochází ke střetu zájmů. Teorie her neslouží pouze pro jejich analýzu, ale také pro vytvoření matematického modelu daného konfliktu a pomocí výpočtů k nalezení nejlepší strategie.

1.1 Základní pojmy teorie her

Teorie her definuje a zkoumá situace, ve kterých dochází ke střetu zájmů jednotlivých účastníků, kdy účastníkem nemusíme rozumět pouze jedince, ale například i skupinu jednotlivců či dané prostředí (např. příroda). Každý takovýto účastník střetu se nazývá **hráč**. Cílem každého hráče je, aby v rámci dané situace pro sebe zajistil co nejlepší výsledek. Chování hráče během řešení konfliktní situace je dané podle jeho **inteligence.** Inteligentní hráč volí jednotlivá rozhodnutí tak, aby co nejvíce maximalizoval svou možnou výhru, takového hráče lze nazvat racionálním. Naopak hráč, který svá rozhodnutí volí náhodně, bez přemýšlení nad výší možného zisku, je hráč neinteligentní. V případě, že je hráči výsledek situace lhostejný, jedná se o indiferentního hráče. V praxi se může při daných situacích zdát, že hráč se nechová jako inteligentní, a proto se zavádí označení p-inteligentního hráče, kde se hráč chová jako inteligentní s pravděpodobností p a jako neinteligentní s pravděpodobností $1-p$.

Hra je všeobecně rozhodovací situace, ve které figurují hráči. Proto se hrou nerozumí pouze společenské hry, jako jsou šachy, dáma, karty, apod., ale jakákoli situace, ve které záleží na rozhodnutí. Taková situace může nastat i v běžném životě, ekonomii či sportu a dalších oblastech. Každá hra je definována a popsána souborem pravidel, kterými se hráči řídí.

Možnosti rozhodnutí, která jsou hráči akceptovatelná, se nazývají **strategie.**

1.1.1 Strategie a tahy hráčů

Veškeré přípustné strategie, patří do množiny všech přípustných řešení, z kterých hráči vybírají, nazývána také jako **prostor strategií**. Jako strategii tedy můžeme označit jakékoliv chování hráče. Volba strategie ovlivňuje samotný průběh hry, a proto je cílem každého hráče nalézt svou **optimální strategii.** Optimální strategií je myšleno takové rozhodnutí, které hráči přinese největší užitek ze všech možných alternativ.

Jednotlivé strategie jsou realizovány pomocí jednotlivých **tahů** hráčů. Přesná definice tahu je stanovena pravidly konkrétní hry. V teoretické rovinně lze za tah považovat hráčovu volbu rozhodnutí z množiny všech možných alternativ, které lze realizovat v dané herní situaci. Jednotlivé tahy lze rozdělit na dva základní druhy, tj. tah **osobní** a tah **náhodný.** **Osobním tahem**, rozumíme tah takový, jenž si hráč sám vybral z množiny všech dostupných rozhodnutí, zatímco **náhodným tahem** rozumíme tah, který není vybrán hráčem, ale je vybrán náhodným mechanismem. Osobní tah je například tah v šachu, oproti tomu náhodný tah je pro příklad tah ve hře Člověče, nezlob se, kdy náhodným

mechanismus je hráčům hod kostkou. Zde má hráč celkem 6 alternativ (6-ti stěnná kostka) a pravděpodobnost výběru kterékoliv alternativy je 1/6.

Optimální strategii hráč volí na základě **výplatní funkce**. Tato funkce pro hráče tvoří přepis, udávající důsledek hráčovi volby strategie. Hodnotami této funkce je výhra, resp. prohra.

Ve hře dvou hráčů, kdy má každý hráč konečný počet strategií, lze sestavit tabulku, kterou nazýváme **výplatní matice hry**. Matici tvoří tabulka, jejíž obsah charakterizuje užitek (výhry či prohry) hráčem zvolené strategie. Tabulka obsahuje v záhlaví řádků údaje prvního hráče a v záhlaví sloupců údaje druhého hráče.

Příklad 1.1. *Hru hrají dva hráči A a B. Každý hráč má v ruce 2 karty, každá karta je jedinečná svou číselnou hodnotou. Hráč A drží v ruce karty s hodnotami 4 a 9. Hráč B drží v ruce karty o hodnotách 6 a 8. Ve hře je podstatou současné vyložení jedné karty od každého hráče, kdy vítězem je ten hráč, který vyložil kartu s vyšší hodnotou. Výhra je rovna rozdílu hodnot vyložených karet.*

Tabulka 1 – Výplatní matice hry s kartami pro dva hráče

Hráč A/B	Karta s hodnotou 4	Karta s hodnotou 9
Karta s hodnotou 6	2	-3
Karta s hodnotou 8	4	-1

Tabulka uvádí výsledky jednotlivých rozhodnutí. Pokud hráč A zvolí kartu o hodnotě 6, pak jeho výhra v případě, že hráč B zvolí kartu s hodnotou 4, je rovna hodnotě 2. V případě, že hráč B vyloží kartu hodnoty 9, hráč prohrává se ztrátou 3. Pro hráče A je nejvýhodnější volba karty s hodnotou 8, kdy maximalizuje výhru z 2 na 4 a zároveň v případě prohry minimalizuje prohru z 3 na 1. Tudíž hráčovo optimální rozhodnutí je takové, kdy minimalizuje prohru a maximalizuje výhru, tj. karta 8.

1.2 Rozdělení her dle vlastností

Každá hra, či konfliktní situace, je odlišná od ostatních. Podle jednotlivých charakteristik her je však lze seskupovat do jednotlivých skupin. Hry lze rozdělit na základě toho, kolik máme protihráčů a zda s nimi můžeme spolupracovat či naopak. Dalším důležitým aspektem je to, co vše je možné provést v rámci našeho tahu. Proto je nutné rozdělit konfliktní situace a jejich modely na několik kriterií, abychom byli schopni sestavit konkrétní popis a návod pro danou konfliktní situaci. Tato kritéria jsou popsána níže.

1.2.1 Počet hráčů

Základním a nejjednodušším kritériem je celkový počet hráčů, kteří se účastní dané konfliktní situace. Protože konfliktní situace je důsledkem rozporu neboli konfliktu mezi

více subjekty, je pravidlem, že každá konfliktní situace vyžaduje minimálně dva účastníky. Z tohoto pohledu lze hry rozdělovat na:

- hry dvou hráčů;
- hry *n* hráčů, kde $n > 2$;
- hry s neomezeným počtem hráčů.

Ve hře o *n* hráčích je možné, že hráči vytvoří skupiny nazývané koalice. Význam koalicí je takový, že hráči tvořící skupinu spolupracují na volbě strategie tak, aby vylepšili svou pozici v dané hře. Hráči tvořící koalici mají většinou podobné zájmy.

V případě hry o *n* hráčích mluvíme o konfliktu, u kterého reálně nelze sestavit seznam hráčů z důvodu velkého množství hráčů.

1.2.2 Inteligence jednotlivých hráčů

Hry lze rozdělovat podle toho, jaká je inteligence jednotlivých účastníků[1] konfliktní situace.

- Hry s inteligentními hráči;
- hry s neinteligentními hráči;
- hry s *p*-inteligentními hráči.

1.2.2.1 Hry s neinteligentními hráči

Neinteligentním hráčem můžeme označit například přírodu. U těchto her, kde je např. příroda náhodným mechanismem, rozlišujeme dvě varianty hráčova rozhodování. Pokud inteligentní hráč zná pravděpodobnosti tahů náhodného mechanismu, zjištěné například na základě předchozích zkušeností, hovoříme o **rozhodování při riziku**. Pokud inteligentní hráč pravděpodobnosti nezná, jedná se o **rozhodování při neurčitosti**.

1.2.3 Součet výher

Součet výher je velice důležité kritérium každé konfliktní situace. Součtem výher je myšlena celková hodnota výhry, o kterou se jednotliví hráči podělí. Podle vlastnosti tohoto kritéria lze rozdělit hry na dvě skupiny:

- hry s konstantním součtem výhry;
- hry s proměnným součtem výhry.

1.2.3.1 Hry s konstantním součtem

O tento druh hry se jedná v případě, že součet výher je pevně daný. To znamená, že tato hodnota není nijak ovlivněna volbou strategií jednotlivých hráčů. Speciální varianta hry

[1] Inteligence hráče je popsána v podkapitole 1.1.

je hra, kde je konstantním součtem výher nula. Tento případ nastává například při hře dvou hráčů, kdy výše výhry jednoho hráče je rovna hodnotě prohře druhého hráče.

Všeobecně lze říci, že pokud označíme výhru každého hráče na konci partie symbolem v_i, kde i =1, 2, ..., n a platí $\sum_{i=1}^{n} v_{i=konst}$, jedná se právě o hru s konstantním součtem.

1.2.3.2 Hry s proměnným součtem

U těchto her není součet výhry pevně daný. Hodnota součtu se mění v závislosti na zvolených strategiích jednotlivých hráčů, tudíž platí $konst = 0$.

1.2.4 Informovanost hráčů

Další charakter hry, podle něhož lze hry rozdělovat, jsou dostupné informace. Informacemi je myšleno to, zda hráči znají jednotlivé tahy a jejich výsledky ostatních hráčů. Proto hry dělíme na:

- hry s dokonalou informací;
- hry s nedokonalou informací.

1.2.4.1 Hry s dokonalou informací

O hře s dokonalou informací mluvíme v případě, že hráči znají veškeré tahy ostatních hráčů a i jejich výsledky. Tento typ her není příliš častý, ale lze se s ním setkat například v kooperativních hrách, kdy je možné veškeré informace s ostatními sdílet a tudíž zjednodušit průběh hry pro koalici.

1.2.4.2 Hry s nedokonalou informací

O tento typ hry se jedná v případě, že hráčům chybí jakákoli informace o jednotlivých tazích či výsledcích ostatních hráčů. Hráči mohou tyto informace poskytovat částečně, či je plně utajit před ostatními spoluhráči.

1.2.4.3 Závislost důsledku volby na informovanosti

Podle toho, jaké informace má hráč k dispozici, je možné dělit hry na:

- deterministické;
- stochastické.

Pokud má hráč informace o tom, jaké důsledky přinese hráči jím zvolená strategie, jedná se o hru **deterministickou**. Pokud ovšem hráč dopředu neví, co volbou rozhodnutí získá či ztratí, hovoříme o hře **stochastické.**

1.2.5 Konečnost strategií

Další rozdělení her lze odvodit dle počtu variant možností pro volbu daného rozhodnutí. Tyto alternativy rozhodnutí jsou v prostoru strategií.[2]

Proto rozdělujeme hry jako:

[2] Prostor strategií byl více probírán v podkapitole 1.1

- konečné**;**
- nekonečné.

Konečné hry jsou takové, které obsahují konkrétní počet možností v prostoru strategií, která lze využít pro výběr rozhodnutí. Pokud počet možností není nijak omezen a tudíž hráči mohou volit nekonečně mnoho variant, hovoříme o **nekonečné hře.**

1.2.6 Cíle a zájmy hráčů

Každý hráč při volbě svého rozhodnutí bere v potaz své vlastní zájmy, které mohou být jak v přímém rozporu se zájmy ostatních hráčů, tak mohou sledovat podobné či dokonce stejné cíle. Proto rozlišujeme:

- antagonistické hry;
- neantagonistické hry.

1.2.6.1 Antagonistické hry

Do této skupiny patří ty hry, kde jsou zájmy jednotlivých hráčů v přímém rozporu. Jako přímý rozpor lze brát výši výhry na úkor ostatních hráčů. To znamená, že do této skupiny zařazujeme hry s konstantním součtem výher.

1.2.6.2 Neantagonistické hry

V této skupině her jsou takové hry, kdy zájmy jednotlivých hráčů nejsou v rozporu či dokonce mohou být podobné nebo stejné. Proto sem patří hry s proměnným součtem výher, u kterých nedochází k rozporu z pohledu výše výhry.

U neantagonistických her často dochází k situacím, kdy je výhodné vytvoření skupiny hráčů, která sleduje společný cíl. V tomto případě hráči společně volí strategii a jsou v kooperaci s ostatními členy skupiny. Z toho pohledu, zda hráči spolupracují či nikoli, dělíme hry na:

- kooperativní;
- nekooperativní.

Kooperativní hry jsou hry, kdy hráči spolupracují ve skupině a sledují společný zájem volbou strategií. Naopak v případě **nekooperativní hry** každý hráč sleduje pouze svůj zájem a volba strategie je řízená pouze přínosem pro samotného hráče.

2 Nekooperativní hry

Neantagonistická nekooperativní hra je taková, kde hráči nemají své zájmy v přímém rozporu a zároveň hra nemá konstantní součet. Pro definici si vezměme dva hráče, hráč A a hráč B. Hráč A má možnost volby z prostoru svých strategií, které označíme $A_1, A_2, \ldots, A_n$, hráč B má strategie označené $B_1, B_2, \ldots, B_m$. Výplaty hráčů při zvolených strategiích označíme jako $M_A(A_i, B_j)$ a $M_B(A_i, B_j)$. U neantagonistického konfliktu, kde není konstantní součet výplat, tudíž platí $M_A(A_i, B_j) + M_B(A_i, B_j) \neq konst$.

Z tohoto důvodu se pro každého hráče vytváří vlastní matice výplat, v případě hry dvou hráčů, proto tyto hry nazýváme **dvojmaticové**.

U nekooperativních her hledáme optimální strategii charakterizovanou pomocí bodu, který nazýváme **Nashův rovnovážný bod**. Tento rovnovážný bod reprezentují strategie A_0 a B_0, jsou to takzvané **rovnovážné strategie**. Obecně pro ně platí, že:

$$M_A(A_i, B_0) \leq M_A(A_0, B_0), \quad (2.1)$$

$$M_B(A_0, B_j) \leq M_B(A_0, B_0). \quad (2.2)$$

U neantagonistických her platí, že pokud se hráč odchýlí od optimální strategie, může si pohoršit nejen na své výplatě, ale zároveň i výplatě protihráče. U antagonistických her platí vždy to, že pokud dojde k ponížení výhry jednoho hráče, dojde ke zvýšení výplaty druhého hráče.

2.1 Vězňovo dilema

Jedná se o nejznámější příklad nekooperativních her. Její zvláštnost je, že optimální řešení, které je výhodné pro oba účastníky konfliktu, existuje. Toto výhodné řešení je však nedostupné z toho důvodu, že porušení dohody vede k vysokému zvýhodnění jednoho účastníka na úkor druhého.

Mějme tuto situaci, hráč A a hráč B nejsou nijak informováni ani ve spojení. Oba hráči mají možnost volby, zda mluvit s vyšetřovateli, či nikoli. Výplatní matice hráčů pak vypadá takto:

Tabulka 2 – Výplatní matice pro vězňovo dilema

	Mluvit	Nic neříkat
Mluvit	(6, 6)	(10, 0)
Nic neříkat	(0, 10)	(2, 2)

Každý hráč chce maximalizovat svou výplatu, v tomto případě chce každý hráč získat nejnižší počet let strávených ve vězení.

Optimální strategií by bylo oboustranné mlčení, kdy by součet výplat byl roven hodnotě 4.

Nejvýhodnější jednostrannou strategií je mlčenlivost jednoho hráče, zatímco druhý hráč promluví. Pak druhý hráč maximalizuje svou výplatu, avšak celkový součet výplat je 10.

Při oboustranném porušení mlčenlivosti, každý z hráčů získá 6 let vězení. V tom případě je suma výplat konfliktu 12.

3 Kooperativní hry

Kooperativní hry jsou nedílnou součástí teorie her. Hry, kde dochází ke kooperaci hráčů, jsou takové, kde hráčům dohoda přinese větší zisk.

3.1 Kooperativní hry dvou hráčů

Pokud daná konfliktní situace dovoluje spolupráci obou hráčů, hráči začínají spolupracovat. V této situaci se účastníci mohou před volbou strategie domlouvat jak o postupu hraní, tak o možném rozdělení výhry. V případě, že se hráči rozhodnou pro spolupráci, nejprve analyzují nastalou situaci, ve které se nacházejí. Poté dojde k dohodě o volbě strategie a případné dohodě o rozdělení výhry.

Základní předpoklad uzavření dohody o kooperaci je to, že spolupráce hráčům zaručí lepší výsledek než v případě, že by hráči hráli každý sám za sebe.

3.1.1 Hry s přenosnou výhrou

Jestliže uzavření dohody přinese lepší výsledek, je ji výhodné uzavřít. Hráči nejprve zjistí, kolik jsou schopni získat bez spolupráce. Výhry hráčů budeme značit jako *v(A)i* a *v(B),* pro které platí, že:

$$v(A) = max_i min_j M_A(A_i, B_j), \qquad (3.1)$$

$$v(B) = max_i min_j M_B(A_i, B_j). \qquad (3.2)$$

Při spolupráci hráčů zjistíme maximální možnou společnou výhru *v(A,B)* jako:

$$v(A,B) = max_i \left[M_A\left(A_i, B_j\right) + M_B(A_i, B_j) \right], \qquad (3.3)$$

což reprezentuje největší číslo v matici A a B. Pokud již známe maximální výhry hráčů bez spolupráce a zároveň maximální společnou výhru, lze zjistit, že se spolupráce vyplatí v případě, že

$$v(A,B) > v(A) + v(B). \qquad (3.4)$$

Potom, co hráči již vědí, že jim spolupráce přinese lepší výsledek, nastává otázka rozdělení výhry. Každý hráč bude sledovat takovou výplatu, která bude rovna či vyšší než výhra, kterou by mohl získat bez spolupráce. Proto výhry hráčů, které označíme a_1 a a_2, musí splňovat následující podmínky:

$$a_1 + a_2 = v(A,B)$$

$$a_1 \geq v(A)$$

$$a_2 \geq v(B)$$

Tyto výhry lze rozdělit několika způsoby. Jde o:

Rozdělení výhry na polovinu, kdy $a_1 = a_2 = \frac{v(A,B)}{2}$. Toto řešení je však často pro hráče nepřijatelné, protože nemusí splňovat podmínku, že každý hráč musí získat minimálně takovou výhru, jakou by byl schopen uhrát sám.

Dělení výhry v poměru přínosu hráčů, je nejvíce spravedlivou možností rozdělení a platí pro ni vztah:

$$a_1 : a_2 = [v(A,B) - v(B) : v(A,B) - v(A)]. \qquad (3.5)$$

Optimální rozdělení, kdy si hráči vezmou částku, kterou by byli schopni uhrát sami a zbytek si rozdělení na polovinu. Pak lze jednotlivé výplaty hráčů psát jako:

$$a_1 = \frac{v(A,B) - v(A) - v(B)}{2} + v(A), \qquad (3.6)$$

$$a_2 = \frac{v(A,B) - v(A) - v(B)}{2} + v(B). \qquad (3.7)$$

3.1.2 Hry s nepřenosnou výhrou

Jedná se o konfliktní situace, kdy je možné mezi hráči uzavřít dohodu o společné strategii, avšak ne o společném využití jejích výsledků. To v praxi znamená takové situace, kdy je přenos výhry buď zakázaný, případně kdy není dostupný žádný mechanismus k přenosu.

Před volbou si hráči musí položit dvě základní otázky, zda je pro ně kooperace výhodná a v případě uzavření dohody, k jakým strategiím se hráči zavazují.

Důležitým aspektem kooperativních her s nepřenosnou výhrou je otázka rozdělování výhry. Hráči si mohou zjistit možné výhry v_1 a v_2 v situaci, kdy nebudou vytvářet spolupráci podle vzorců (3.1) a (3.2). Rozdělování výhry, je v případě her s nepřenosnou výhrou, obtížnější než v případě her s přenosnou výhrou.

Mějme situaci zadanou množinou $M = \{[0,0], [5,5]\}$. Pokud v množině M existuje alespoň jeden prvek označený $[a_1,a_2]$ s vlastností $[a_1, a_2] \neq [v(1), v(2)]$, je kooperace minimálně pro jednu stranu výhodná, protože potencionální výhra přesahuje hodnotu *v(1)*, popř. *v(2)*. Nyní lze zúžit množinu rozdělení na taková, která mohou být realizována.

Definice. Dosažitelné rozdělení $[b_1, b_2] \in M$ nazveme **paretovským rozdělením**, jestliže neexistuje rozdělení $[a_1, a_2] \in M$, kde:

$$(a_1 \geq b_1 \wedge a_2 > b_2) \vee (a_1 > b_1 \wedge a_2 \geq b_2). \qquad (3.8)$$

Množinu všech těchto paretovských rozdělení označíme Ƥ.

Paretovským rozdělením tedy označíme takové rozdělení, kde neexistuje žádné rozdělení, kdy by alespoň jeden hráč získal více a druhý stejně nebo také více než u prvního přijatelného rozdělení.

Většina případů neobsahuje jednoprvkovou množinu Ƥ, tudíž lze obtížně odvodit pojem optimální strategie. Definici optimální strategie také komplikuje fakt, že množina Ƥ nemá tak jednoduchou strukturu, jako je tomu u her s přenosnou výhrou, kdy je jádro[3] konvexní uzavřenou množinou.

[3] Jádrem hry nazýváme množinu všech rozdělení (a_1,a_2), které splňují vztah (3.4).

4 Rozhodování při riziku a neurčitosti – optimální strategie

V předchozích kapitolách se při uvádění podmínek pro rozhodnutí počítá vždy s inteligentními hráči. Mějme situaci, kdy je první hráč označen jako inteligentní, zatímco druhý jako neinteligentní. Inteligentní hráč volí svá rozhodnutí za účelem maximalizace výhry, zatímco neinteligentní hráč je lhostejný k výši své výhry, a proto má charakter náhodného mechanismu, který není závislý na inteligentním hráči.

Konfliktní situace, jejichž účastníky jsou inteligentní i neinteligentní hráči, je možné popsat hrou v normálním tvaru.

$$\{Q = \{1,2\}; X, Y; M_A(x, y) = M(x, y)\}. \qquad (4.1)$$

Výplatní funkce M_B není uvedena, protože hráč B je lhostejný k výši své výhry, tudíž nemá vliv na počítání. Výplatní matice M_A hráče A, je ovlivněna volbou strategie *y* hráče B. Ten volí svou strategie dle rozložení pravděpodobností *F(y)*, definovanou na množině Y. Hráč A volí svá rozhodnutí v závislosti na tom, zda rozložení pravděpodobností *F(y)* hráče B zná či nezná. Pokud toto rozložení zná, hovoříme o **rozhodování při riziku.** V případě, že rozložení hráč A nezná, hovoříme o **rozhodování při neurčitosti**.

4.1 Optimální strategie u rozhodování při riziku

Definice:

$$S(x) = \int_Y M(x, y)dF(y). \quad (4.2)$$

Strategie $\bar{x}$ je optimální strategií hráče A v případě, že platí:

$$S(\bar{x}) = \max_{x \in X} S(x). \qquad (4.3)$$

Pro hráče A je tudíž optimální strategií ta, která maximalizuje střední hodnotu jeho výhry.

Je-li počet prvků množin X a Y konečný, lze sestavit pro rozhodování jedinou matici:

$$A = \begin{pmatrix} a_{11} & a_{22} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}$$

Pokud neinteligentní hráč B volí v matici i-tý sloupec s pravděpodobností p_i, i = 1, 2, …, n, pak je pro hráče A podle definice optimální ten řádek j, jehož střední hodnota je

$\sum_{i=1}^{n} a_{ij} p_i$ maximální.

4.2 Optimální strategie u rozhodování při neurčitosti

Volba optimální strategie u rozhodování při neurčitosti je komplikovanější než u rozhodování při riziku. Pro rozhodování je zavedeno několik principů, jejichž výsledné

optimální strategie nemusí být při totožné konfliktní situaci stejné. Zde jsou uvedeny čtyři nejznámější strategie.

4.2.1 Princip nedostatečné evidence

Pokud inteligentní hráč A nezná pravděpodobnosti voleb neinteligentního hráče B, je možné říci, že jsou jejich strategie rovnocenné. Proto je tedy nutné, aby hráč A bral volby hráče B se stejnou pravděpodobností. Je nutné předpokládat, že v nekonečné množině Y u hráče B má rozložení *F(y)* nejmenší obsah informace. Informace způsobují již řečený výběr se stejnou pravděpodobností.

Princip nedostatečné evidence říká, že pro optimální strategie $\bar{x}$ platí:

$$S(x) = \int_Y M(x,y)dF(y). \quad (4.4)$$

Za *F(y)* dosadíme rozložení s nejmenším obsahem informace na Y.

Princip nedostatečné evidence pro konečné hry definuje optimální strategie *i* jako maximální hodnotu výrazu $\frac{1}{n}\sum_{j=1}^{n} a_{ij}$.

4.2.2 Princip minimaxu

Princip minimaxu je založen na myšlence, že inteligentní hráč A v každém případě získá alespoň výhru:

$$\min_{y \in Y} M(x,y).$$

Jinými slovy, minimální hodnotu ze své výplatní matice. Princip říká, že optimální strategie je ta, která maximalizuje tento výraz.

Pro konečné hry, lze stanovit optimální strategii *i* jako maximální hodnotu výrazu:

$$\min_j a_{ij}.$$

4.2.3 Princip minimaxu ztráty

Pro tento princip je optimální rozhodnutí takové, které nás vyvaruje velkým ztrátám ve srovnání s rozhodnutím, kdy bychom se rozhodovali podle znalosti strategie druhého neinteligentního hráče.

Při hledání optimální strategie podle principu minimaxu ztrát, je nutné nejprve definovat funkci ztrát *Z(x,y)*:

$$Z(x,y) = M(x,y) - \max_{x \in X} M(x,y). \quad (4.5)$$

Za optimální strategii $\bar{x}$ volíme tu, pro kterou výraz

$$\min_{y \in Y} Z(x,y)$$

nabývá maxima.

Pro konečné hry se nejedná již o funkci, ale mluvíme o matici ztrát, která má prvky:

$$z_{ij} = a_{ij} - \max_k a_{kj}.$$

Optimální strategie i je ta strategie, která maximalizuje řádková minima matice ztrát. Princip minimaxu ztrát zajistí, že zvolená optimální strategie nám zaručí co nejmenší ztráty a podle toho dostupné výhry.

4.2.4 Princip ukazatele optimismu (Hurwiezův)

U toho principu, jindy také nazývaného jako **Hurwiezův ukazatel**, je potřeba zavést konstantu $\alpha, \alpha \in \langle 0,1 \rangle$. Ta zastupuje ukazatel optimismu inteligentního hráče A.

Pokud označíme $A(x) = \max_{y \in Y} M(x,y)$ a $a(x) = \min_{y \in Y} M(x,y)$, pak se za optimální strategii $\bar{x} \in X$ označí ta, pro kterou je výraz:

$$A(x) + (1 - \alpha)a(x)$$

maximální. Pokud $\alpha = 0$, je princip ukazatele shodný s principem minimaxu. Tento princip je podrobněji popsán v (MAŇAS, 1974).

5 Pandemic

Desková kooperativní hra Pandemic, vydaná firmou ALBI Česká republika a.s., je hrou pro 1 až 5 hráčů. Ti mají jeden společný úkol, společnými silami bojovat proti čtyřem nemocem, které se šíří po celém světě. Hráči spolupracují na vynalezení léků na jednotlivé choroby a také zabraňují šíření chorob. Každý hráč je specialistou na určité odvětví a jeho specializace mu dává možnost speciálních schopností.

5.1 Pravidla hry

Kompletní pravidla hry lze najít v přiloženém návodu (LEACOCK) nebo na internetových stránkách vydavatele www.albi.cz.

5.1.1 Obsah hry

Hra obsahuje herní plán, na kterém je zobrazena mapa světa s vybranými městy, plochy pro hrací a infekční karty, ukazatel míry infekce, počítadlo pandemií, vynalezené léky a stručnou nápovědu pro jednotlivé fáze tahu. Mezi městy jsou znázorněná jednotlivá přímá spojení měst, díky nimž hráči vědí, do kterého města se smějí v rámci přímého přejezdu přemístit.

Dále obsahuje čtyři balíčky kostek po 24 kusech, které dle jednotlivých barev určují, o kterou nemoc se jedná, 6 výzkumných stanic, 4 žetony pro vynalezené léky, žeton infekce, žeton pandemie, 59 hracích a 48 infekčních karet, 5 karet rolí a 4 referenční karty.

5.1.1.1 Kostky nemocí

Kostky nemocí se používají pro určování úrovně nemoci v daném městě. Kostek je celkem 96 – 24 od každé barvy. Jednotlivé barvy jsou přiřazeny k nemocem takto:

- žlutá – břišní tyfus;
- červená – SARS;
- modrá – antrax;
- černá – cholera.

5.1.1.2 Hrací karty

Hracích karet je celkově 59. Patří mezi ně karty jednotlivých měst, zvláštních událostí a 6 karet epidemie. Hrací karty měst jsou pro hráče důležité nejen pro možnost rychlé přepravy, ale také proto, že jejich sběrem lze vynalézt lék na konkrétní nemoc. Zvláštní události jsou dobrými pomocníky, jsou to:

- Vládní grant – Postavení výzkumné stanice ve městě dle vlastního výběru.
- Předpověď – Hráč si může vzít prvních šest infekčních karet a vrátit je zpět v libovolném pořadí.

- Přeprava – Umožní hráči přemístit libovolného hráče do libovolného města.
- Klidná noc – V některém dalším kole lze vynechat fázi infikování.
- Odolná populace – Hráč může odstranit libovolnou kartu infekce z odkládacího balíčku.

Tyto zvláštní události lze zahrát kdykoli a jejich zahrání nestojí žádnou akci. Po sejmutí karty epidemie hráč zvyšuje míru infekce, infikuje třemi kostkami město, jehož karta je vespod dobíracího balíčku infekcí, tu pak odloží na odkládací balíček, který je po té zamíchán a vrácen navrch dobíracího balíčku infekcí.

5.1.1.3 Karty infekcí

Infekční karty si hráči lízají a je jimi určeno, které město se má infikovat. Odehrané infekční karty se odkládají do odkládacího balíčku, který je po sejmutí hrací karty epidemie zamíchán a vrácen zpět.

5.1.1.4 Role hráčů

Role hráčů jsou dány náhodným rozdáním karet rolí. Každá role má svou vlastní speciální vlastnost.

- Vědec – potřebuje pouze 4 karty pro vynalezení léku.
- Výzkumník – může provádět akci sdílení znalostí s hráčem ve stejném městě a není limitován nutností měnit pouze kartu města, ve kterém hráči stojí.
- Dispečer – může přesouvat s figurkami ostatních hráčů stejně, jakoby se jednalo o jeho vlastní.
- Operační expert – může stavět výzkumné stanice kdekoli, kde se právě nachází.
- Medik – v rámci tahu léčení nemoci odstraňuje všechny kostky nemoci. V případě, že je již vynalezen lék, odstraňuje všechny kostky nemocí pouhý projetím města.

5.1.1.5 Výzkumné stanice

Výzkumné stanice jsou záchytné body pro hráče. Jejich maximální množství je 6. Výzkumná stanice slouží jako bod, kde lze vynalézt lék na nemoc, či slouží jako cíl akce let raketoplánem, kdy hráči mohou cestovat mezi městy s výzkumnými stanicemi.

5.2 Průběh hry

Celkový průběh hry je dán tímto postupem:

1. Odehrání 4 akcí.
2. Dobrání 2 hracích karet (maximum držených karet je 7).
3. Odehrání role infektora.

5.2.1 Hraní akcí

Hráč má možnost odehrát libovolnou kombinaci 4 akcí, či případně akci vynechat. Role hráčů umožňují hráčům jednotlivé akce modifikovat.

5.2.1.1 Základní akce

- Přejezd – přesun hráče mezi městy, která jsou spojena.
- Přímý let – přesun hráče zahráním hrací karty města, jenž drží. Hráč se přesune do města označeného na kartě a kartu odloží na odkládací balíček.
- Charterový let – hráč zahraje kartu města, ve kterém právě stojí a přesune se do libovolného města na mapě. Tuto kartu odloží na odkládací balíček.
- Let raketoplánem – hráč stojící ve městě s výzkumnou stanicí, se může přemístit do libovolného města s jinou výzkumnou stanicí.

Role dispečera umožňuje odehrání všech těchto akcí s figurkami ostatních hráčů.

5.2.1.2 Speciální akce

- Stavba výzkumné stanice – pokud hráč drží kartu města, ve kterém právě stojí, může v tomto městě postavit výzkumnou stanici. Odehranou kartu odloží na odkládací balíček. Operační expert tuto akci může provádět v libovolném městě.

- Vynalezení léku – pokud hráč stojí ve městě s výzkumnou stanicí a drží 5 karet stejné barvy, může vynalézt lék. Karty odloží na odkládací balíček a označí tuto nemoc za vyléčenou. Tato nemoc se již neinfikuje. Hráči s rolí vědce stačí pouze 4 karty stejné barvy.

- Léčení nemoci – hráči mohou odstranit jednu kostku nemoci ve městě, kde se právě nachází. Pokud je již vynalezen lék, hráči odstraňují veškeré kostky. Medik v rámci jedné akce odstraňuje veškeré kostky nemocí, a pokud je vynalezen lék, medik léčí město pouhým průjezdem města.

- Sdílení znalostí – hráči si mohou vyměnit kartu města, pokud se v tomto městě nacházejí a pokud i hráč, který má kartu obdržet, se nachází ve stejném městě. Výzkumník může darovat jakoukoli kartu, kterou drží, pokud se nachází ve stejném městě s hráčem, s nímž chce výměnu realizovat.

5.2.2 Dobírání karet

Po odehrání všech 4 akcí, si hráč dobírá z balíčku hracích karet po 2 kartách. Každý hráč přitom může držet maximálně 7 hracích karet. Pokud již hráč překračuje množství hracích karet, musí ostatní odložit na odkládací balíček.

Pokud je sejmutou kartou karta zvláštní události, lze ji použít kdykoli v průběhu hry.

V případě, že je mezi dobranými kartami karta epidemie, musí hráč tuto událost zahrát okamžitě. Postup pro zahrání epidemie je popsán na přední straně karty. Hráč nejprve zvyšuje úroveň infekce o jeden stupeň na ukazateli míry infekce. Poté infikuje město, jehož karta je vespod dobíracího balíčku infekcí, třemi kostkami. Tuto kartu hráč odloží

na odkládací balíček karet infekcí, balíček zamíchá a vrátí zpět nahoru na dobírací balíček infekcí.

5.2.3 Hraní infektora

Infikování je posledním krokem jednoho tahu hráče. Hráč sejme z balíčku infekčních karet stejný počet karet, jako je hodnota ukazatele míry infekce. Každému z těchto měst hráč přidá jednu kostku nemoci.

Pokud má hráč přidat kostku nemoci městu, které již 3 kostky nemoci obsahuje, dochází k **pandemii.** Během pandemie, musí hráč umístit kostky nemoci do všech okolních měst[4]. Pokud některé z těchto měst obsahuje 3 kostky nemoci, celá situace se opakuje. Během každého vzniku pandemie, musí hráči zvýšit hodnotu počítadla pandemií.

5.3 Konec hry

5.3.1 Výhra

Hráči společně vyhrají, pokud se jim podaří vynalézt léky na všechny choroby. Podmínkou vítězství není vyléčení všech měst, hráči vítězí v okamžiku vynalezení čtvrtého léku.

5.3.2 Prohra

Hra končí v okamžiku, kdy nastane jedna z těchto možností:

- Hráč musí infikovat město, ale již není žádná volná kostka dané nemoci.
- Nastane osmá pandemie.
- Již nejsou žádné karty v balíčku hracích karet a hráč musí snímat.

5.4 Předpoklady hry

5.4.1 Příprava hry

Zde budeme uvažovat hraní všech rolí a nejvyšší obtížnost hry. Každý hráč obdrží po dvou hracích kartách. Do zbylých hracích karet jsme náhodně umístili všech 6 karet epidemie. V této situaci je 10 rozdaných hracích karet hráčům a 49 hracích karet, z nichž 6 karet jsou epidemie.

Na začátku každé hry je nutné provést prvotní infikaci, kdy infikujeme celkem 9 měst. První tři infikujeme třemi kostkami, další tři dvěma kostkami a zbylé tři po jedné kostce. Tudíž celková pravděpodobnost pro infikování je $p = \frac{celkový\ počet\ infikovaných\ měst}{celkový\ počet\ měst} = \frac{9}{48}$. Tuto pravděpodobnost lze rozdělit na jednotlivé stupně infekce. U prvních tří karet, kdy se infikuje třemi kostkami, je $p = \frac{3}{48}$. Pravděpodobnost infikování dvěma kostkami je $p = \frac{3}{45}$ a pro infikování jednou kostkou je $p = \frac{3}{42}$. Nyní je v balíčku infekcí 39 karet a celkem bylo již rozdáno 18 kostek nemocí.

[4] Okolní města jsou ta, která mezi sebou mají přímé spojení.

5.4.2 Průběh hry

Během hry je pro hráče důležité sledovat pravděpodobnost vypuknutí pandemie. Pandemie je stav, kdy hráč má přidat kostku nemoci do města, kde již tři kostky nemoci jsou. V takovém případě hráč musí přidat kostku nemoci do všech okolních měst. Pokud i v těchto městech je již stav nemoci maximální, tj. tři kostky, postupuje do dalších propojených měst.

Tato situace může nastat v případě, že hráč získá hrací kartu epidemie. Po provedení všech kroků epidemie se karty měst vracejí na vršek balíčku infekcí, tudíž je již možné počítat pravděpodobnost vypuknutí pandemie. V tu chvíli můžeme určit pravděpodobnost vypuknutí pandemie $p = \frac{počet\ měst\ se\ třemi\ kostkami}{celkový\ počet\ infekčních\ karet\ v\ odkládacím\ balíčku}$.

Tuto pravděpodobnost se hráči během hraní snaží minimalizovat společně s úkolem vynalezení léků.

Pro minimalizaci pravděpodobnosti vypuknutí pandemie, jsou hráči nuceni průběžně léčit města. K léčení je nutné, aby se hráč v daném městě nacházel, tudíž je pro hráče výhodné znát základy teorie grafů, díky které mohou zefektivnit své přesuny mezi městy minimalizací počtu tahů. K tomuto účelu lze využít Floyd-Warshallův algoritmus blíže popsaný v (MIČKA, 2012).

6 Floyd-Warshallův algoritmus

Floyd-Warshallův algoritmus pracuje na orientovaném grafu. Tento graf nesmí obsahovat záporné cykly, což nám zaručí nezápornost hran. Algoritmus nalezne nejkratší cesty možné cesty mezi dvojicí zadaných vrcholů a vybere z nich cestu s nejmenším počtem hran.

Pro tento účel je možné použít *n*-krát Dijkstrův algoritmus, vždy pro každý vrchol. Vzhledem k nutnosti tento algoritmus použít *n*-krát, se velmi zvyšuje časová složitost a zároveň i samotná implementace takového postupu je velice náročná. Floyd-Warshallův algoritmus počítá veškeré vzdálenosti ihned, což vede k nižší časové náročnosti, a také jeho samotná implementace je snadnější, než aplikace Dijsktrova algoritmu pro každý vrchol zvlášť.

Pro použití Floyd-Warshallova algoritmu, je nutné očíslovat jednotlivé vrcholy od 1 do *n*. Vzdálenosti mezi jednotlivými vrcholy, jsou uložené v matici $n \times n$. Princip algoritmu není přímé počítání, ale počítání v *n* iteracích.

Mějme vstupní matici vzdáleností D^{vstup}, která obsahuje vzdálenosti mezi jednotlivými vrcholy. Diagonálu této matice tvoří nuly, protože se jedná o vzdálenost mezi prvkem samotným, a na ostatních indexech jsou konkrétní hodnoty hran mezi jednotlivými vrcholy. V případě, že konkrétní vrcholy nemají společnou hranu, je uvedena hodnota nekonečno. V praxi to znamená, že tato matice obsahuje informace o ohodnocení hran přímo spojených vrcholů, to znamená bez prostředníků.

Během každé iterace algoritmu je matice vzdáleností přepočítávána tak, aby vyjadřovala vzdálenost všech dvojic vrcholů v postupně se zvětšující množině možných prostředníků.

Transformaci matice pro $k \geq 1$ můžeme vyjádřit rekurzivním vztahem:

$$D_{i,j}^{k} = \min \left(D_{i,j}^{k-1}, D_{i,k}^{k-1} + D_{k,j}^{k-1} \right). \qquad (6)$$

Celková asymptotická složitost algoritmu je $O(|U|^3)$.

Princip Floyd-Warshallova algoritmu lze pochopit z následující ukázky pseudokódu:

```
procedure [array] FloydWarshall(D)
    for k in 1 to n do
        for i in 1 to n do
            for j in 1 to n do
                if D[i][j] > D[i][k] + D[k][j] then
                    D[i][j] = D[i][k] + D[k][j]
    return D
```

Více o jeho implementaci a možnostech rozšíření lze zjistit na (Hugue, 2011) nebo (MIČKA, 2012).

7 Aplikace Pandemic – Průvodce

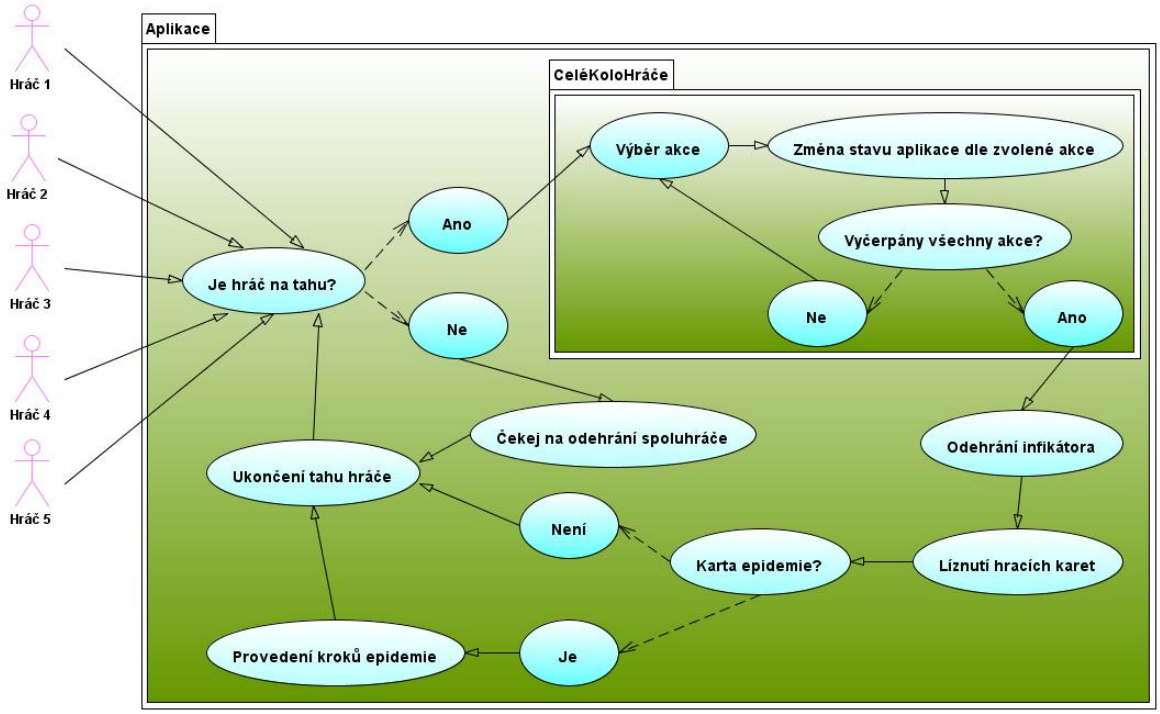
V této bude popsána samotná aplikace a její jednotlivé části.

Aplikace je napsána v programovacím jazyce JAVA v prostředí Netbeans IDE a pro grafické prvky hry jsem použil trial verzi balíčku aplikací společnosti Adobe. Konkrétně se jedná o aplikaci Adobe Photoshop CS6[5].

7.1 Popis aplikace

Aplikace má za úkol možnost odehrání kompletní hry Pandemic včetně zobrazování pravděpodobnosti vypuknutí pandemie a možnosti zobrazení nejkratší možné cesty za použití základní akce přesunutí.

UML diagram užití přibližuje princip samotné aplikace z pohledu samotné hry.



Obrázek 1 – UML diagram užití

7.1.1 Grafika aplikace

7.1.1.1 Mapa světa

Pro vytvoření mapy světa byla jako základ použita mapa světa získaná z internetové databáze vektorové a rastrové grafiky[6]. Mapa byla upravena pomocí nástrojů programu Adobe Photoshop CS6. Postup byl vytažení hran světadílů, vyjmutí vodních ploch do nové vrstvy, překrytí barvou a následné vytlačení a přidání záře na okraj vrstvy se světadíly. Jednotlivé ikony byly tvořeny ručně pomocí křivek a jejich různými variacemi barev, aby

[5] (Adobe, 2013)
[6] (Vectorya.com, 2009)

odpovídaly jednotlivým barvám měst a nemocí originální hry. V pravé straně mapy je prostor nechaný pro tabulku zobrazení údajů měst.

Obrázek 2 – Mapa světa

7.1.2 Rozložení prvků aplikace

Obrázek 3 – Rozložení aplikace

Rozložení aplikace je takové, že mapa světa včetně zobrazení měst umožní hráči lehčí orientaci v aplikaci. V té souvislost je umístěna tabulka výpisu informací o městech na panelu v pravé části samotné mapy. Ovládání aplikace je umístěno v levé dolní části společně s přehledem karet ostatních hráčů. Pomocní ukazatelé a nástroj pro vyhledání nejkratší cesty mezi městy se nacházejí v pravém dolním rohu.

7.1.2.1 Mapa světa a výpis informací měst

Mapa světa se zobrazuje staticky, pouze prvotním vykreslením při spuštění aplikace. Vedle mapy světa je umístěna dynamicky generovaná tabulka, která v každém kole zobrazuje aktuální stav měst. Informace pro jednotlivá města jsou:

- název města;
- aktuálně se zde nacházející hráč/i;
- zda je v tomto městě postavená výzkumná stanice;
- aktuální stav zamoření nemocemi.

7.1.2.2 Ovládací panel akce hráče

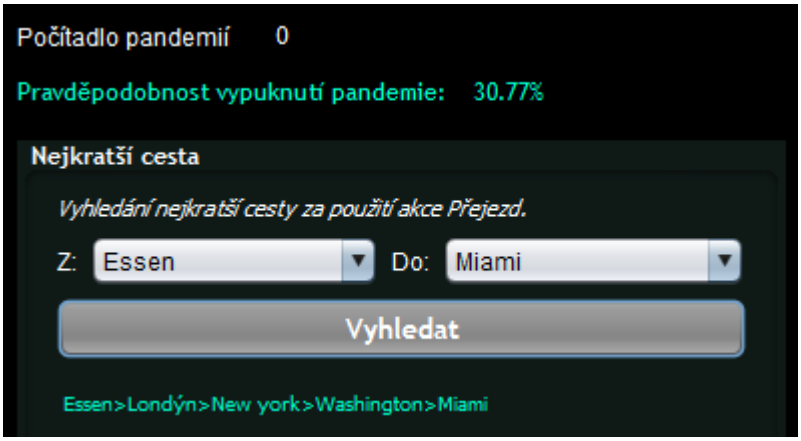
Ovládací panel pro akce hráče obsahuje informaci o pořadí aktuálně prováděné akci, dynamicky plněný seznam pro možnost výběru akce a tlačítko pro výběr akce. Reakcí na vybrání akce je naplnění dvou seznamů, nazvaných specifikace akce. Ty jsou plněny v závislosti na zvolené akci. Zvolená a upřesněná akce se potvrdí tlačítkem pro zahrání tahu.

7.1.2.3 Panel informace o držených kartách ostatními hráči

Panel ostatních karet spoluhráčů tvoří čtveřice uskupení popisku a seznamu. Obsah popisku a k němu vázanému seznam se generuje dynamicky vždy po odehrání čtyř akcí právě hrajícího hráče. V popisku je zobrazena role hráče a v seznamu jsou vypsány všechny jeho držené karty.

7.1.2.4 Informační panel

Informační panel obsahuje informaci o vypočtené pravděpodobnosti možnosti vypuknutí pandemie a panel pro nalezení nejkratší cesty mezi hráčem vybranými městy. Po vybrání měst hráč stisknutím tlačítka vyhledat potvrdí výběr a popisek umístěný pod vyhledávacím tlačítkem zobrazí nejkratší cestu v textové podobě.



Obrázek 4 – Informační panel po vypuknutí epidemie a vyhledání nejkratší cesty

7.2 Jednotlivé části programu

7.2.1 Externí data pro aplikaci

Vzhledem k množství dat, potřebných pro načtení všech základních částí aplikace, jsou data o městech a akcích uložena v souboru XML. Matice spojnic měst ve formě jedniček a nul, je uložena v textovém souboru, který je postupně aplikací načítán.

7.2.1.1 Seznam měst – playingCards.xml

XML soubor *playingCards.xml* udržuje informace o městech. Je použit nejen pro načtení hracích karet, ale také měst a infekčních karet.

Kořenový uzel je pojmenován <*cityCards*> a samotný prvek <*card*> má tuto strukturu:

```
<card>
 <idPCard>ID</idPCard>
 <name>Název</name>
 <region>Barva regionu</region>
</card>
```

7.2.1.2 Seznam akcí

Soubor *actions.xml* obsahuje informace pro jednotlivé akce, které hráči mohou zahrát.

Kořenový uzel je pojmenován <*actions*> a jednotlivé prvky jsou uvedeny jako <*action*>. Soubor má strukturu:

```
<action>
 <actionID>ID</actionID>
 <name>Název akce</name>
 <desc>Popis akce.</desc>
 <class>Třída akce</class>
</action>
```

7.2.1.3 Matice přímého spojení měst

Textový soubor *cityConnectMatrix.txt* obsahuje několik řádků, obsahujících informaci o existenci přímého spojení měst. Pomocí čísla řádku, případně pozice znaku, se asociuje existence přímého spojení znaky 1 či 0 jednotlivým městům.

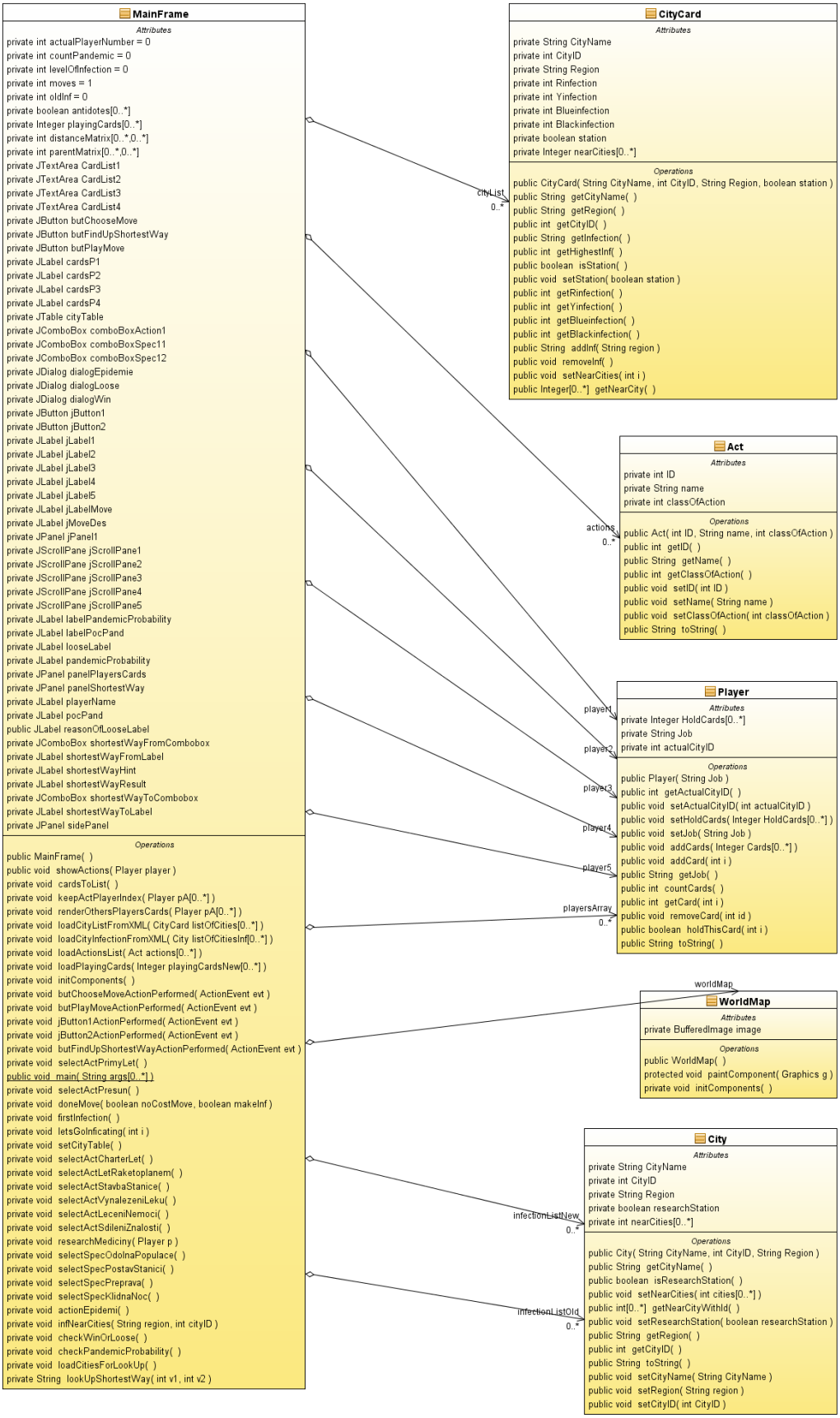
Obsah textového souboru matice spojů, pro prvních 5 měst vypadá následovně:

```
01001001000000000000000001100000000000000000000
10001101100000000000000000000000000000000000000
00010010011000010001000000000000000000000000000
00100000001010000000001000000000000000000000000
11000000000000000000000000100000010000000000000
```

7.2.2 Zdrojová část aplikace

Princip aplikace z pohledu hry, byl již zobrazen v podkapitole 6.1. Tato podkapitola se však věnuje samotné zdrojové části aplikace.

7.2.2.1 UML diagram tříd



Obrázek 5 – UML diagram tříd

Jak lze vidět z uvedeného UML diagramu tříd, aplikaci tvoří hlavní třída MainFrame.java typu JFrame. Tato třída obsahuje a používá objekty, definované v ostatních třídách. Nejdříve zde budou uvedeny jednotlivé třídy, a poté jejich inicializace a využití hlavní třídou MainFrame.java.

7.2.2.2 Třída Act

Tato třída definuje akce, ze kterých hráči vybírají během svých tahů.

Konstruktor:

```
public Act(int ID, String name, int classOfAction) {
        this.ID = ID;
        this.name = name;
        this.classOfAction = classOfAction;
    }
```

Volání konstruktorů nám zajistí, že během vytváření instance této třídy, je nutné zadat jedinečné identifikační číslo akce interpretované jako *ID*, název akce jako textový řetězec *name* a celé číslo *classOfAction*, určující zařazení této akce.

Veškeré atributy jsou *private*, tudíž jejich nastavení či získání je možné pouze pomocí metod **get** a **set**.

Třída obsahuje také přetíženou metodu toString:

```
@Override
    public String toString() {
        return this.name;
    }
```

Tato metoda zařizuje vypsání hodnoty *name* při zobrazování v modelu komponenty typu Combobox.

7.2.2.3 Třída CityCard

Tato třída je používána jak pro hrací a infekční karty, tak i pro jednotlivé prvky v seznamu měst. Tato třída má konstruktor:

```
public CityCard(String CityName, int CityID, String Region, boolean
station) {
        this.CityName = CityName;
        this.CityID = CityID;
        this.Region = Region;
        this.Rinfection = 0;
        this.Yinfection = 0;
        this.Blackinfection = 0;
        this.Blueinfection = 0;
        this.station = station;
        this.nearCities = new ArrayList<>();
    }
```

Při vytvoření instance této třídy je zapotřebí znát název, identifikační číslo, název regionu a pravdivostní hodnotu existence výzkumné stanice. V samotném konstruktoru se již

předem nastavují hodnoty jednotlivých infekcí, označených jako *Rinfection, Yinfection, Blackinfection* a *Blueinfection*, na nulovou hodnotu.

Třída dál obsahuje funkce, jako například:

```
public String getInfection() {
        String inf = "";
        if (Rinfection > 0) {
            inf = inf.concat("Črvn" + Integer.toString(Rinfection) +
";");
        }
        if (Blackinfection > 0) {
            inf = inf.concat("Črn" + Integer.toString(Blackinfection) +
";");
        }
        if (Blueinfection > 0) {
            inf = inf.concat("M" + Integer.toString(Blueinfection) +
";");
        }
        if (Yinfection > 0) {
            inf = inf.concat("Ž" + Integer.toString(Yinfection) + ";");
        }
        if (inf.equals("")) {
            inf = "-";
        }
        return inf;
    }
```

Tato funkce vrací textovou hodnotu výše infekcí, a to tak, že v případě výskytu více druhů infekcí, se tyto hodnoty přidávají k vracejícímu textovému řetězci společně se zkratkou barvy konkrétní nemoci.

Zjištění nejvyšší hodnoty aktuální infekce, zajišťuje funkce:

```
public int getHighestInf() {
        int highest = 0;
        if (highest < Rinfection) {
            highest = Rinfection;
        }
        if (highest < Blackinfection) {
            highest = Blackinfection;
        }
        if (highest < Blueinfection) {
            highest = Blueinfection;
        }
        if (highest < Yinfection) {
            highest = Yinfection;
        }
        return highest;
    }
```

Přidávání infekce městu dle barvy infekční karty zajišťuje funkce *addInf*, definovaná:

```
public String addInf(String region) {
        switch (region) {
            case "Černá":
                if(Blackinfection<3){
```

```
                    Blackinfection++;
                }else{
                    return "Černá";
                }
                break;
            case "Modrá":
                if(Blueinfection<3){
                    Blueinfection++;
                }else{
                    return "Modrá";
                }
                break;
            case "Červená":
                if(Rinfection<3){
                    Rinfection++;
                }else{
                    return "Červená";
                }
                break;
            case "Žlutá":
                if(Yinfection<3){
                    Yinfection++;
                }else{
                    return "Žlutá";
                }
                break;
        }
        return "OK";
    }
```

Tato funkce zajišťuje přidání jednoho bodu infekce dle vstupní barvy. Pokud město již obsahuje 3 body dané infekce, funkce vrací String hodnotu barvy konkrétní infekce. Pokud město neobsahuje 3 body infekce, funkce vrací textový řetězec *OK*. Díky vracející hodnotě funkce aplikace dokáže rozhodovat o přidávání infekcí a vypuknutí případné pandemie.

Pro léčení nemocí slouží metoda *removeInf()*, která pomocí *getHighest()* zjistí nejvyšší infekci a sníží její hodnotu o 1. Metoda má tvar:

```
public void removeInf() {
        if (getHighestInf() == Rinfection) {
            Rinfection--;
        } else if (getHighestInf() == Yinfection) {
            Yinfection--;
        } else if (getHighestInf() == Blueinfection) {
            Blueinfection--;
        } else if (getHighestInf() == Blackinfection) {
            Blackinfection--;
        }
    }
```

7.2.2.4 Třída Player

Tato třída slouží pro tvorbu instancí pro hráče. U hráče jsou důležité tři parametry: role hráče, aktuální pozice a seznam držených karet. Samotnému konstruktoru se předává pouze textový řetězec o roli hráče. Hodnota aktuální pozice se automaticky nastaví na hodnotu 14, udávající pozici pro město Atlanta, dle pravidel hry, a také prázdný ArrayList, který bude udržovat ID držených karet.

Konstruktor má tvar:

```
public Player(String Job) {
        this.Job = Job;
        HoldCards = new ArrayList<>();
        actualCityID = 14;
    }
```

Tato třída má kromě metod *get* a *set*, metodu pro přidání hodnoty ID karty do seznamu držených karet:

```
public void addCard(int i) {
        this.HoldCards.add(i);
    }
```

Zjištění počtu karet pomocí metoda *size()* seznamu držených karet:

```
public int countCards() {
        return HoldCards.size();
    }
```

Odebírání karet podle zadaného ID hrací karty:

```
public void removeCard(int id) {
        for (int i = 0; i < HoldCards.size(); i++) {
            if (HoldCards.get(i) == id) {
                HoldCards.remove(i);
                break;
            }
        }
    }
```

Odebrání karty dle ID karty s návratovou pravdivostní hodnotou:

```
public boolean holdThisCard(int i) {
        for (int j = 0; j < HoldCards.size(); j++) {
            if (i == HoldCards.get(j)) {
                return true;
            }
        }
        return false;
    }
```

Pro vypsání role hráče v Comboboxu je použita přetížená metoda:

```
@Override
    public String toString() {
        return this.Job;
    }
```

7.2.2.5 Třída WorldMap

Třída WorldMap.java je dědicem třídy JPanel. Tato třída slouží pouze pro vykreslení mapy světa ze souboru. Načtení souboru s mapou světa

```
image = ImageIO.read(new File("pics/worldMap.jpg"));
```

načte soubor v bloku *try* a *catch* do proměnné image typu BufferedImage. Tato proměnná je poté vykreslena pomocí:

```
g.drawImage(image, 0, 0, null);
```

7.2.2.6 Třída MainFrame

Toto je hlavní třída aplikace, která je dědicem třídy JFrame. Tato třída kromě mnoha proměnných obsahuje mnoho funkcí, nutných pro celý běh aplikace. Při spuštění aplikace se jako první přiřadí třída WorldMap, včetně jejího umístění do celkového layoutu JFrame

```
this.worldMap = new WorldMap();
add(worldMap, BorderLayout.CENTER);
```

Všechny komponenty formuláře jsou generovány automaticky pomocí návrháře Netbeans IDE.

Po inicializaci všech proměnných, aplikace načte data z externích souborů. Načtení měst do seznamu měst:

```
private void loadCityListFromXML(ArrayList<CityCard> listOfCities) {
        CityCard insertedCity;
        String name, region;
        int cityID;

        try {
            DocumentBuilderFactory dbf =
DocumentBuilderFactory.newInstance();
            DocumentBuilder db = dbf.newDocumentBuilder();
            Document doc = db.parse(new
FileInputStream("playingCards.xml"));
            doc.getDocumentElement().normalize();

            NodeList nodeLst = doc.getElementsByTagName("card");

            for (int s = 0; s < nodeLst.getLength(); s++) {

                Node node = nodeLst.item(s);

                if (node.getNodeType() == Node.ELEMENT_NODE) {

                    Element elmnt = (Element) node;

                    name =
elmnt.getElementsByTagName("name").item(0).getTextContent();
                    region =
elmnt.getElementsByTagName("region").item(0).getTextContent();
                    cityID =
Integer.parseInt(elmnt.getElementsByTagName("idPCard").item(0).getTextCon
tent());
                    insertedCity = new CityCard(name, cityID, region,
false);
                    listOfCities.add(insertedCity);
                }

            }
        } catch (Exception e) {
            e.printStackTrace();
```

```
        }
    }
```

Následuje načtení textového souboru s maticí existence přímých spojů mezi městy, kde pořadí řádku slouží jako index pro seznam měst, a pořadí symbolu v řádku určuje index přilehlého města v případě, že hodnota symbolu je rovna znaku 1. Metoda má tvar:

```
private void loadNearCities(ArrayList<CityCard> cityList) {
        String s;
        int actCity = 0;
        try (BufferedReader input = new BufferedReader(new
FileReader("cityConnectMatrix.txt"))) {
            while ((s = input.readLine()) != null) {
                for (int i = 0; i < s.length(); i++) {
                    if (s.charAt(i) == '1') {
                        cityList.get(actCity).setNearCities(i + 1);
                    }
                }
                actCity++;
            }
        } catch (IOException e) {
            System.out.println("Chyba na vstupu souboru
cityConnectMatrix.txt");
        }
    }
```

Nyní jsou již města načtená do seznamu, včetně seznamu přilehlých měst. Dále je potřeba nahrát hrací a infekční karty.

Infekční karty jsou načteny naprosto stejně jako města, protože infekční karty jsou karty jednotlivých měst.

Hrací karty se nahrávají ve dvou fázích. Nejprve jsou nahrány karty měst a karty zvláštních událostí:

```
private void loadPlayingCards(ArrayList<Integer> playingCardsNew) {
        try {
            DocumentBuilderFactory dbf =
DocumentBuilderFactory.newInstance();
            DocumentBuilder db = dbf.newDocumentBuilder();
            Document doc = db.parse(new
FileInputStream("playingCards.xml"));
            doc.getDocumentElement().normalize();
            NodeList nodeLst = doc.getElementsByTagName("card");
            for (int s = 0; s < nodeLst.getLength(); s++) {
                Node node = nodeLst.item(s);

                if (node.getNodeType() == Node.ELEMENT_NODE) {
                    Element elmnt = (Element) node;

playingCardsNew.add(Integer.parseInt(elmnt.getElementsByTagName("idPCard"
).item(0).getTextContent()));
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
```

```
        /* Přidání karet Zvláštní události. Jejich ID začínají od 1000 */
        playingCardsNew.add(1000);
        playingCardsNew.add(1001);
        playingCardsNew.add(1002);
        playingCardsNew.add(1003);
        Collections.shuffle(playingCardsNew);
    }
```

Následně jsou přidána jednotlivá čísla zvláštních událostí a nakonec je pomocí kolekce provedeno zamíchání seznamu karet. Toto nahrání je první fází, kdy v balíčku hracích karet prozatím nejsou karty epidemií. Nyní se každému hráči rozdá po dvou kartách pomocí funkce *addCards(ArrayList<Integer> Cards)* třídy *Player*:

```
player1.addCards(playingCards);
```

V další fázi jsou přidány do balíčku hracích karet karty epidemií označené číslem 666 a následně celý balíček hracích karet zamíchán:

```
playingCards.add(666);
Collections.shuffle(playingCards);
```

Dalším krokem přípravy hry je prvotní infikování měst. Tento krok se realizuje pomocí *firstInfection()*, která provede prvotní infikování dle pravidel uvedených v podkapitole 5.4.1.:

```
private void firstInfection() {
        City city;
        for (int j = 0; j < 3; j++) {
            city = infectionListNew.remove(0);
            cityList.get(city.getCityID() - 1).addInf(city.getRegion());
            cityList.get(city.getCityID() - 1).addInf(city.getRegion());
            cityList.get(city.getCityID() - 1).addInf(city.getRegion());
            infectionListOld.add(city);
        }

        for (int j = 0; j < 3; j++) {
            city = infectionListNew.remove(0);
            cityList.get(city.getCityID() - 1).addInf(city.getRegion());
            cityList.get(city.getCityID() - 1).addInf(city.getRegion());
            infectionListOld.add(city);
        }

        for (int j = 0; j < 3; j++) {
            city = infectionListNew.remove(0);
            cityList.get(city.getCityID() - 1).addInf(city.getRegion());
            infectionListOld.add(city);
        }
    }
```

Dále se nastaví výzkumná stanice v úvodním městě Atlanta pomocí:

```
cityList.get(14).setStation(true);
```

Tabulka informací o městech je naplněna metodou *setCityTable()*, která vytvoří model tabulky, do kterého přidává jednotlivá data o městech ve formě elementů modelu. Tento model je po naplnění nastaven tabulce *cityTable*:

```
private void setCityTable() {
        TableModel tableModel;
        String cityName = "", player = "", researchStation = "",
infection = "";
        tableModel = cityTable.getModel();

        for (int i = 0; i < cityList.size(); i++) {
            player = "";
            cityName = cityList.get(i).getCityName();
            if (cityList.get(i).isStation() == true) {
                researchStation = "Ano";
            } else {
                researchStation = "Ne";
            }
            infection = cityList.get(i).getInfection();
            for (int j = 0; j < playersArray.length; j++) {
                if (cityList.get(i).getCityID() - 1 ==
playersArray[j].getActualCityID()) {
                    if (!player.isEmpty()) {
                        if (!player.contains("-")) {
                            player = player.substring(0, 2);
                        }
                        player = player.concat("-");
                        player =
player.concat(playersArray[j].getJob().substring(0, 2));
                    } else {
                        player = playersArray[j].getJob();
                    }
                }
            }
            tableModel.setValueAt(cityName, i, 0);
            tableModel.setValueAt(player, i, 1);
            tableModel.setValueAt(researchStation, i, 2);
            tableModel.setValueAt(infection, i, 3);
        }
        cityTable.setModel(tableModel);
    }
```

Nyní je již téměř vše připraveno pro samotnou hru.

Pro možnost odehrání tahu aktuálním hráčem, je nutná metoda *showActions()*, která vypisuje jednotlivé možné akce do nabídky akcí. Akce kromě přiřazení do JCombobox zjišťuje, zda existují reálné podmínky pro možnost vybraných akcí. Celá funkce pracuje s *DefaultComboboxModel*, který je postupně plněn instancemi třídy Act, kde element modelu obsahuje tuto instanci a při jejím volání ve výpisu je využito přetížené metody *toString*, která zajišťuje reprezentaci jednotlivých akcí pomocí jejich názvu. Metoda nejprve ověří existenci zvláštních událostí u aktuálního hráče podle ID držené karty a v případě výskytu přiřadí název a ID instanci třídy Act, kterou následně přidává do modelu *DefaultComboboxModel*. Po zjištění, zda má hráč možnost zvolit akci zvláštní události, funkce zajišťuje zobrazení základních a speciálních akcí pouze za určitých

předpokladů. Pro možnost zahrání charterového letu je nutné, aby hráč držel kartu města, ve kterém se právě nachází. Podmínka pro tuto akci je návratová hodnota funkce *holdThisCard* použitá takto:

```
playersArray[actualPlayerNumber].holdThisCard(playersArray[actualPlayerNu
mber].getActualCityID() + 1)
```

Let raketoplánem je možný pouze pokud se hráč nachází ve městě s výzkumnou stanicí. Pro tento účel třída CityCard obsahuje funkci isStation s pravdivostní návratovou hodnotou. Podmínka pak vypadá následovně:

```
cityList.get(playersArray[actualPlayerNumber].getActualCityID()).isStatio
n()
```

Akce postavení výzkumné stanice je limitovaná pro hráče podmínkou, že musejí držet kartu konkrétního města, ve kterém chtějí postavit výzkumnou stanici a zároveň v tomto městě stojí. Výjimkou je hráč s rolí operačního experta, který tuto akci může provádět v libovolném městě, ve kterém se právě nachází bez nutnosti držet kartu konkrétního města. Složená podmínka pro všechny hráče s výjimkou operačního experta je:

```
playersArray[actualPlayerNumber].getCard(j) ==
playersArray[actualPlayerNumber].getActualCityID() ||
playersArray[actualPlayerNumber].getJob().equals("Operační expert")
```

Akce pro vynalezení léku je dána podmínkami:

- hráč musí držet 5 karet stejné barvy (pokud je hráčova role vědec, musí držet 4 karty stejné barvy);
- hráč musí stát ve městě s výzkumnou stanicí.

Tyto podmínky po zjištění počtu jednotlivých barev držených hráčem vytvářejí rozhodovací část:

```
if (playersArray[actualPlayerNumber].getJob().equals("Výzkumník")) {
                        if (countOfBlack >= 4 || countOfBlue >= 4 ||
countOfRed >= 4 || countOfYellow >= 4 &&
cityList.get(playersArray[actualPlayerNumber].getActualCityID()).isStatio
n()) {
                            model.addElement(actions.get(i));
                        }
                    } else (countOfBlack >= 5 || countOfBlue >= 5 ||
countOfRed >= 5 || countOfYellow >= 5 &&
cityList.get(playersArray[actualPlayerNumber].getActualCityID()).isStatio
n()) {
                        model.addElement(actions.get(i));
                    }
```

Akce pro léčení nemoci v konkrétním městě se hráči nabízí pouze ve městech, které je již infikované, tudíž podmínka je:

```
cityList.get(playersArray[actualPlayerNumber].getActualCityID()).getHighe
stInf() > 0
```

Modifikace této akce pro roli hráče medika spočívá v tom, že medik odebírá veškeré kostky nemoci v rámci jedné akce.

Zobrazení akce sdílení znalostí je možné, pouze pokud oba hráči stojí ve stejném městě, a zároveň je vyměňovaná karta stejná, jako právě aktuální město. Tato akce není pro roli hráče vědce podmíněna tím, aby vyměňovaná karta byla shodná s aktuálním městem, ve kterém hráči stojí. Pro ostatní hráče je proto nutné splnění podmínky:

```
playersArray[actualPlayerNumber].getCard(j) ==
playersArray[actualPlayerNumber].getActualCityID() – 1
```

7.2.2.7 Akce hráče

Každý hráč má možnost po vybrání akce bližší specifikace akce. To je využíváno jako například definice cílového města při přesunech, stanovení hráče pro akci sdílení znalostí, apod. Po stisknutí tlačítka **Zahrát tah**, aplikace zjišťuje zvolenou akci:

```
Act selectedAction = (Act) comboBoxAction1.getSelectedItem();
```

Dále pomocí funkce *getID()* rozhoduje, jakou funkci má volat.

Při volbě akce **Přejezd** aplikace přebere informaci o zvoleném městě, zjistí její ID a podle toho změní hráčovo aktuální město.

```
playersArray[actualPlayerNumber].setActualCityID(choosenCity1.getCityID()
- 1);
```

Akce **Přímý let** odebere z hráčových karet kartu aktuálního města a nastaví její ID jako ID aktuálního města.

```
playersArray[actualPlayerNumber].removeCard(choosenCity2.getCityID());
playersArray[actualPlayerNumber].setActualCityID(choosenCity2.getCityID()
- 1);
```

Charterový let odstraní zahranou kartu a nastaví hráčovo aktuální město pomocí ID města, které si zvolil.

```
playersArray[actualPlayerNumber].removeCard(playersArray[actualPlayerNumb
er].getActualCityID());

playersArray[actualPlayerNumber].setActualCityID(choosenCity3.getCityID()
- 1);
```

Let raketoplánem pouze změní hráčovo ID aktuálního města na jím zvolené město.

```
playersArray[actualPlayerNumber].setActualCityID(choosenCity4.getCityID()
- 1);
```

Stavba výzkumné stanice nejdříve zjišťuje, zda je hráčova role operační expert. Pokud ano, hráč pouze změní pravdivostní hodnotu proměnné *researchStation* na *true*. Pokud je hráčova role jiná, kromě postavení stanice je nutné odebrat kartu aktuálního města.

```
cityList.get(playersArray[actualPlayerNumber].getActualCityID()).setStati
on(true);
                    if
(!playersArray[actualPlayerNumber].getJob().equals("Operační expert")) {

playersArray[actualPlayerNumber].removeCard(playersArray[actualPlayerNumb
er].getActualCityID());
                    }
```

Vynalezení léku volá metodu *researchMediciny(Player player)*, která pro daného hráče provádí hledání karet stejné barvy. Počet potřebných karet záleží na roli hráče, pokud se jedná o vědce, je potřebný počet 4, jestliže je to jakýkoli jiný hráč, je nutné držet 5 karet stejné barvy. Poté funkce odstraní pomocí *player.removeCard(int index)* v cyklu všechny tyto karty a nastaví pravdivostní hodnotu v poli pro vynalezené léky na daném indexu na hodnotu *true*.

Léčení nemoci pouze odstraňuje nemoc z aktuálního města pomocí metody *removeInf()*.

Sdílení znalostí dle vybraného hráče nalezne jeho pozici v seznamu všech hráčů a následně tomuto hráči přidá kartu vybranou při specifikaci akce pomocí metody *addCard(int idKarty)*. Poté tuto kartu odebere aktuálnímu hráči ze seznamu držených karet pomocí:

```
playersArray[actualPlayerNumber].removeCard(int idKarty)
```

Metody pro zahrání zvláštních událostí jsou uvedeny v příloze C. Zvláštní události se od jiných akcí liší tím, že nestojí žádný tah při jejich zahrání. Tudíž je v provedení tahu zavedena proměnná *boolean noCostMove*, která umožňuje rozeznat akci, při které se počítadlo odehraných akcí *moves* nezvyšuje.

7.2.2.8 Kontrola stavu hry

Po odehrání 4 akcí hráče, je volána metoda *checkWinOrLoose()*. Tato metoda kontroluje současný stav hry, zda nejsou splněny podmínky pro konec hry[7].

Metoda nejdříve kontroluje, zda hráči vynalezli všechny léky. Pokud jsou všechny léky vynalezeny, hra končí výhrou hráčů. Výhra je oznámena dialogovým oknem. Podmínka se zobrazením dialogu výhry vypadá takto:

```
if(antidotes[0] && antidotes[1] && antidotes[2] && antidotes[3]){
      dialogWin.setVisible(true);
}
```

Pokud hra neskončila výhrou, metoda zjišťuje podmínky pro prohru.

[7] Podmínky pro výhru či prohru jsou popsány v podkapitole 5.3.

Nejprve jsou sečteny jednotlivé barvy nemocí pomocí cyklu:

```
for (int i = 0; i < cityList.size(); i++) {
            blueInf += cityList.get(i).getBlueinfection();
            blackInf += cityList.get(i).getBlackinfection();
            redInf += cityList.get(i).getRinfection();
            yellowInf += cityList.get(i).getYinfection();
        }
```

Po té metoda zkontroluje všechny podmínky prohry hráčů a v případě, že je jedna z těchto podmínek splněna, je JLabel *reasonOfLooseLabel* nastaven text s důvodem prohry. Celá podmínka včetně nastavení odůvodnění a zobrazení dialogu prohry je:

```
if (yellowInf >= 24 || blackInf >= 24 || blueInf >= 24 || redInf >= 24 ||
countPandemic >= 8 || playingCards.isEmpty()) {
            if (countPandemic >= 8) {
                reasonOfLooseLabel.setText("Již proběhlo příliš mnoho
pandemií...");
            }
            if (playingCards.isEmpty()) {
                reasonOfLooseLabel.setText("Již nemáte žádné hrací
karty...");
            }
            if (yellowInf >= 24) {
                reasonOfLooseLabel.setText("Břišní tyfus (žlutá) Vás
porazil...");
            }
            if (blackInf >= 24) {
                reasonOfLooseLabel.setText("Cholera (černá) Vás
porazila...");
            }
            if (blueInf >= 24) {
                reasonOfLooseLabel.setText("Antrax (modrá) Vás
porazil...");
            }
            if (redInf >= 24) {
                reasonOfLooseLabel.setText("SARS (červené) Vás
porazil...");
            }

      dialogLoose.setVisible(true);
}
```

7.2.2.9 Nápověda pro hráče

Hráčům je v aplikaci poskytnuta nápověda pravděpodobnosti průběhu dalšího běhu hry.

První část je výpočet pravděpodobnosti vypuknutí pandemie. To je stav kdy se městu má přidat nemoc, ale město již obsahuje maximální počet. Proto je nutné tyto nemoci umístit do vedlejších měst. Možnost vypuknutí pandemie se objevuje až v případě, že si hráč z balíčku hracích karet vezme kartu epidemie. Po té se karty odehraných infekcí vracejí nahoru na balíček k dobírání, a proto vzniká možnost vypuknutí pandemie.

Metoda pro výpočet pravděpodobnosti vypuknutí pandemie:

```
private void checkPandemicProbability() {
        double countWithThree = 0;
```

```
        double result;

        for (int i = 0; i < cityList.size(); i++) {
            for (int j = 0; j < infectionListNew.size(); j++) {
                if (infectionListNew.get(j).getCityID() ==
cityList.get(i).getCityID()) {
                    if (cityList.get(i).getBlackinfection() == 3) {
                        countWithThree++;
                    }
                    if (cityList.get(i).getBlueinfection() == 3) {
                        countWithThree++;
                    }
                    if (cityList.get(i).getRinfection() == 3) {
                        countWithThree++;
                    }
                    if (cityList.get(i).getYinfection() == 3) {
                        countWithThree++;
                    }
                    break;
                }
            }
        }

        // Výpočet pravděpodobnosti.
        result = (countWithThree / oldInf) * 100.0;
        // Zaokrouhlení na dvě desetinná místa.
        result = Math.round(result * 100.0) / 100.0;
        pandemicProbability.setText(Double.toString(result) + "%");
    }
```

Tato metoda nejdříve prohledává celý seznam měst *cityList* a nalezne počet měst, které mají počet jedné z nemocí roven 3. Poté do proměnné *result* provede výpočet pravděpodobnosti převedený na dvě desetinná místa. Tento výsledek po převedení na textový formát nastaví komponentě *pandemicProbability* typu JLabel:

```
pandemicProbability.setText(Double.toString(result) + "%");
```

Druhá část je hledání nejkratší cesty mezi dvěma městy pomocí Floyd-Warshall algoritmu. Tato metoda využívá dvou komponent JCombobox, díky nimž si hráč vybere ze seznamu dvě města, pro která následně aplikace vypíše nejnižší možný počet jednotlivých přesunů.

Jako první je vytvořeno dvourozměrné pole *private int[][]*. Toto pole slouží jako zdroj jednotlivých spojení měst popsaných v podkapitole 6.2.1.3.

Nejprve je nutné toto pole naplnit daty podle externího souboru *cityConnectMatrix.txt*:

```
try (BufferedReader input = new BufferedReader(new
FileReader("cityConnectMatrix.txt"))) {
            String s;
            int j = 0;

            while ((s = input.readLine()) != null) {
                for (int i = 0; i < s.length(); i++) {
                    if (s.charAt(i) == '0') {
                        // Nastavení nekonečna, není přímý spoj.
                        distanceMatrix[j][i] = 999;
                    }
```

```
                    if (s.charAt(i) == '1') {
                        distanceMatrix[j][i] = 1;
                    }
                }
                j++;
            }
        } catch (IOException e) {
            System.out.println("Chyba na vstupu souboru
cityConnectMatrix.txt");
        }
```

Metoda načte textový soubor a následně cyklem prochází jednotlivé řádky. Druhý vnořený cyklus znak po znaku kontroluje, zda se jedná o 1 nebo 0. V případě, že načtený symbol je 0, matici vzdáleností na indexech dle aktuálního pořadí řádku a pořadí znaku přiřadí hodnotu 999, což nám zajistí, že tato cesta nebude označena za nejkratší, protože nahrazuje nekonečno, označující to, že přímé spojení mezi městy neexistuje. Pokud je načteným znakem 1, znamená to, že mezi těmito městy existuje přímá cesta.

Po inicializaci matic je nutné provést samotný algoritmus:

```
private void floydWarshall(int[][] d) {
        parentMatrix = constructInitialMatixOfPredecessors(d);
        for (int k = 0; k < d.length; k++) {
            for (int i = 0; i < d.length; i++) {
                for (int j = 0; j < d.length; j++) {
                    if (d[i][k] == Integer.MAX_VALUE || d[k][j] ==
Integer.MAX_VALUE) {
                        continue;
                    }
                    if (d[i][j] > d[i][k] + d[k][j]) {
                        d[i][j] = d[i][k] + d[k][j];
                        parentMatrix[i][j] = parentMatrix[k][j];
                    }
                }
            }
        }
    }
```

Matice předků *int[][] parentMatrix* je naplněna pomocí:

```
private int[][] constructInitialMatixOfPredecessors(int[][] d) {
        int[][] p = new int[d.length][d.length];
        for (int i = 0; i < d.length; i++) {
            for (int j = 0; j < d.length; j++) {
                if (d[i][j] != 0 && d[i][j] != Integer.MAX_VALUE) {
                    p[i][j] = i;
                } else {
                    p[i][j] = -1;
                }
            }
        }
        return p;
    }
```

Po provedení algoritmu bude matice vzdáleností obsahovat přepočítané hodnoty, odpovídající počtu nutných přesunů mezi jednotlivými městy, a matice předků údaje pro rekonstrukci cesty.

Samotné vyhledávání nejkratší cestou je provedeno metodou, která do JLabel *shortestWayResult* přidává jednotlivé mezikroky vybrané cesty:

```
private void lookUpShortestWay(int i, int j) {
        if (i == j) {
            shortestWayResult.setText(shortestWayResult.getText() + " " +
cityList.get(i).getCityName());
            System.out.println(i);
        } else if (parentMatrix[i][j] == 0) {
            shortestWayResult.setText(shortestWayResult.getText() +
"Cesta neexistuje.");
            System.out.println("Neexistuje");
        } else {
            lookUpShortestWay(i, parentMatrix[i][j]);
            shortestWayResult.setText(shortestWayResult.getText() + ">" +
cityList.get(j).getCityName());
            System.out.println(j);
        }

    }
```

Závěr

Tato práce mi dokázala, že teorie her jako matematická disciplína má širokou škálu použití a uplatnění v mnoha oborech. Jsou to například ekonomie, politologie nebo sociologie. Získaných poznatků se dá využít v každodenních situacích, a to jak v profesním, tak i osobním životě. Zejména ve chvílích, kdy je člověk nucen přijmout rozhodnutí, které může silně ovlivnit průběh dalších situací.

Čtenář byl v této práci seznámen s historií a vývojem tohoto oboru, jeho základními pojmy a především s problematikou možné spolupráce účastníků dané konfliktní situace. Z tohoto pohledu byla řešena otázka výhody uzavření dohody o spolupráci, volba další strategie a také nástroje pro rozdělování výher. Dále byla čtenáři ukázána možnost využití znalostí z teorie grafů při hledání optimální strategie.

V praktické části této práce byla zpracována kooperativní desková hra Pandemic, jako aplikace naprogramovaná v jazyce JAVA. Vytvořená aplikace zobrazuje pravděpodobnost vypuknutí pandemie a hráči usnadňuje výběr jeho optimální strategie. Pro ještě větší usnadnění výběru optimální strategie, byla vytvořena metoda hledání nejkratší možné cesty, pomocí Floyd-Warshallova algoritmu.

Tato práce mi ukázala náročnost vytvoření aplikace simulující deskovou hru v programovacím jazyce JAVA a zároveň širokou použitelnost knihovny uživatelských komponent Swing pro tvorbu GUI aplikací.

Samotná aplikace má velký potenciál pro rozšíření. Také by při dalším vývoji bylo vhodné provést optimalizaci kódu pro menší paměťovou náročnost. Možná rozšíření a vylepšení mohou být

- doplnění o volbu počtu hráčů a úrovně náročnosti hry;
- možnost vrácení provedeného tahu;
- dynamické zobrazování stavu hry na mapě světa.

Literatura

Adobe, Systems Software Ireland. 2013. Adobe Photoshop. *Adobe Family.* [Online] Adobe Systems Software Ireland Ltd., 2013. [Citace: 20. 12 2012.] http://www.adobe.com/cz/photoshop.html.

ČERNÝ, Jakub. Základní grafové algoritmy. [Online] [Citace: 2. 5 2013.] http://kam.mff.cuni.cz/~kuba/ka/.

Heylighen, F. 1995. The Prisoners' Dilemma. *Pincipia Cybernetica Web.* [Online] 13. 6 1995. [Citace: 26. 3 2013.] http://pespmc1.vub.ac.be/PRISDIL.html.

HRUBÝ, Martin. 2011. THE: Nekooperativní hry v normální formě. [Online] 27. 7 2011. [Citace: 25. 3 2013.] http://www.fit.vutbr.cz/~hrubym/THE/2-normal-form-games.pdf.

Hugue, Michelle. 2011. Lecture 24: Floyd-Warshall Algorithm. *CMSC 351 Algorithms.* [Online] University Of Maryland, 31. 8 2011. [Citace: 20. 3 2013.] http://www.cs.umd.edu/~meesh/351/mount/lectures/lect24-floyd-warshall.pdf.

LEACOCK, Matt. Pravidla hry Pandemic. *Pravidla hry Pandemic.* Praha : ALBI Česká republika a.s.

MAŇAS, Miroslav. 2002. *Teorie her a konflikt zájmů.* Praha : Nakladatelství Oeconomica, 2002. ISBN 80-245-0450-2.

—. 1974. *Teorie her a optimální rozhodování.* Praha : Nakladatelství technické literatury, 1974. SNTL.

MIČKA, Pavel. 2012. Floyd-Warshallův algoritmus. *Algoritmy.net.* [Online] 2012. [Citace: 27. 4 2013.] http://www.algoritmy.net/article/5207/Floyd-Warshalluv-algoritmus.

Oracle, Corporation. 2011. Package javax.swing. *Oracle Documentation.* [Online] Oracle Corporation, 2011. [Citace: 14. 4 2013.] http://docs.oracle.com/javase/6/docs/api/javax/swing/package-summary.html.

RASMUSEN, Eric. 2006. *Games and Information: An Introduction to Game Theory, 4th Edition.* místo neznámé : Wiley-Blackwell, 2006. ISBN 978-1-4051-3666-2.

Vectorya.com. 2009. Free Vector World Map. *VectorYa.com.* [Online] VectorYa.com, 2009. [Citace: 3. 4 2013.] http://vectorya.com/freevectors/art-designs/free-vector-world-map/.

WAMPLER, Dean. 2011. *Functional Programming for Java Developers.* Cambridge : O'Reilly Media, 2011. ISBN:978-1-4493-1103-2.

Příloha A – Obsah CD-ROM

Kořenový adresář

- **autorun.inf** – Informace pro automatické spuštění CD.
- **LanarA_OptimalniStrategie_JM_2013.pdf** – Bakalářská práce ve formátu PDF.
- **prezentace.pdf** – Prezentace pro obhajobu.
- **readme.txt** – Obsah CD v textové podobě.
- **Pandemic/** - Adresář aplikace.
    - **pics/** - Adresář pro obrázky načítané aplikací.
        - **worldMap.jpg** – Obrázek mapy světa.
    - **actions.xml** – Zdrojový XML soubor pro načítání akcí.
    - **cityConnectMatrix.txt** – Textový soubor s maticí přímých spojení měst.
    - **Pandemic.jar** – JAR soubor aplikace.
    - **playingCards.xml** – Zdrojový XML soubor pro načítání karet a měst.

Příloha B – Soubor akcí actions.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<actions>
    <action>
        <!-- Dispečer posunuje i figurkami ostatních hráčů. Může
přesouvat figurku vzdáleně tam kde již jiný hráč stojí.  -->
        <actionID>1</actionID>
        <name>Přejezd</name>
        <desc>Přesun figurky do vedlejšího města.</desc>
        <class>1</class>
    </action>
    <action>
        <actionID>2</actionID>
        <name>Přímý let</name>
        <desc>Přesun do města, jehož kartu držíme.</desc>
        <class>1</class>
    </action>
    <action>
        <actionID>3</actionID>
        <name>Charterový let</name>
        <desc>Přesun figurky do libovolného města, pokud hráč s)tojí ve
městě, jehož kartu drží.</desc>
        <class>1</class>
    </action>
    <action>
        <actionID>4</actionID>
        <name>Let raketoplánem</name>
        <desc>Přesun figurky mezi městy s výzkumnými stanicemi.</desc>
        <class>1</class>
    </action>
    <action>
        <!-- Operační expert může postavit stanici bez držení karty. -->
        <actionID>5</actionID>
        <name>Stavba výzkumné stanice</name>
        <desc>Postavení výzkumné stanice ve městě, kde hráč stojí a má
její kartu.</desc>
        <class>2</class>
    </action>
    <action>
        <!-- Vědec potřebuje pouze 4 karty stejné barvy. -->
        <actionID>6</actionID>
        <name>Vynalezení léku</name>
        <desc>Vynalezení léku, pokud hráč stojí ve městě s výzkumnou
stanicí a drží 5 karet stejné barvy.</desc>
        <class>2</class>
    </action>
    <action>
        <actionID>7</actionID>
        <name>Lečení nemoci</name>
        <!-- Medik odstraňuje všechny body nemoce jedné barvy, pokud je
vynalezen lék, lečí všechny pouhým projitím města. -->
        <desc>Odstranění jednoho bodu nemoce, kde hráč stojí. Pokud je
již vynalezen lék, odstranění všech bodů.</desc>
        <class>2</class>
    </action>
    <action>
        <!-- Výzkumník může předávat karty v jakémkoliv městě, kde oba
hráči stojí. -->
```

```
        <actionID>8</actionID>
        <name>Sdílení znalostí</name>
        <desc>Výměna karty hráče s hráčem. Oba stojí ve stejném městě a
předává se karta daného města.</desc>
        <class>2</class>
    </action>

</actions>
```

Příloha C – Zdrojový kód pro akce souboru MainFrame.java

```
private void butPlayMoveActionPerformed(java.awt.event.ActionEvent evt) {
        Act selectedAction = (Act) comboBoxAction1.getSelectedItem();
        boolean noCostMove = false;
        boolean makeInf = true;
        if (comboBoxSpec11.isEnabled()) {
            switch (selectedAction.getID()) {
                case 1:
                    City choosenCity1 = (City)
comboBoxSpec11.getSelectedItem();

playersArray[actualPlayerNumber].setActualCityID(choosenCity1.getCityID()
- 1);
                    break;
                case 2:
                    City choosenCity2 = (City)
comboBoxSpec11.getSelectedItem();

playersArray[actualPlayerNumber].removeCard(choosenCity2.getCityID());

playersArray[actualPlayerNumber].setActualCityID(choosenCity2.getCityID()
- 1);
                    break;
                case 3:
                    City choosenCity3 = (City)
comboBoxSpec11.getSelectedItem();

playersArray[actualPlayerNumber].removeCard(playersArray[actualPlayerNumb
er].getActualCityID());

playersArray[actualPlayerNumber].setActualCityID(choosenCity3.getCityID()
- 1);
                    break;
                case 4:
                    City choosenCity4 = (City)
comboBoxSpec11.getSelectedItem();

playersArray[actualPlayerNumber].setActualCityID(choosenCity4.getCityID()
- 1);
                    break;
                case 5:

cityList.get(playersArray[actualPlayerNumber].getActualCityID()).setStati
on(true);
                    if
(!playersArray[actualPlayerNumber].getJob().equals("Operační expert")) {

playersArray[actualPlayerNumber].removeCard(playersArray[actualPlayerNumb
er].getActualCityID());
                    }
                    break;
                case 6:
                    researchMediciny(playersArray[actualPlayerNumber]);
                    break;
                case 7:

cityList.get(playersArray[actualPlayerNumber].getActualCityID()).removeIn
f();
                    break;
```

```
                case 8:
                    City chooCity5 = (City)
comboBoxSpec11.getSelectedItem();
                    Player pl = (Player)
comboBoxSpec12.getSelectedItem();
                    for (int i = 0; i < playersArray.length; i++) {
                        if (playersArray[i].getJob().equals(pl.getJob()))
{

playersArray[i].addCard(chooCity5.getCityID());

playersArray[actualPlayerNumber].removeCard(chooCity5.getCityID());
                        }
                    }
                    break;
                case 1000:
                    City choosenCity6 = (City)
comboBoxSpec11.getSelectedItem();
                    infectionListOld.remove(choosenCity6.getCityID() -
1);
                    noCostMove = true;
                    playersArray[actualPlayerNumber].removeCard(1000);
                    break;
                case 1001:
                    City choosenCity7 = (City)
comboBoxSpec11.getSelectedItem();
                    cityList.get(choosenCity7.getCityID() -
1).setStation(true);
                    noCostMove = true;
                    playersArray[actualPlayerNumber].removeCard(1001);
                    break;
                case 1002:
                    City choosenCity8 = (City)
comboBoxSpec11.getSelectedItem();
                    Player choosenPlayer = (Player)
comboBoxSpec12.getSelectedItem();

                    for (int i = 0; i < playersArray.length; i++) {
                        if
(playersArray[i].getJob().equals(choosenPlayer.getJob())) {

playersArray[i].setActualCityID(choosenCity8.getCityID() - 1);
                        }
                    }
                    noCostMove = true;
                    playersArray[actualPlayerNumber].removeCard(1002);
                    break;
                case 1003:
                    makeInf = false;
                    playersArray[actualPlayerNumber].removeCard(1003);
                    break;
            }
            comboBoxSpec11.disable();
            comboBoxSpec12.disable();

            doneMove(noCostMove, makeInf);
        }
```

Příloha D – Mapa světa s městy

Příloha E – Ukázka aplikace v průběhu hry

Příloha F – Tabulka přímých spojů pro černá města

	Čennaj	Dillí	Istanbul	Alžír	Kalkata	Teherán	Bagdád	Bombaj	Karáčí	Moskva	Káhira	Rijád
Čennaj	0	1	0	0	1	0	0	1	0	0	0	0
Dillí	1	0	0	0	1	1	0	1	1	0	0	0
Istanbul	0	0	0	1	0	0	1	0	0	1	1	0
Alžír	0	0	1	0	0	0	0	0	0	0	1	0
Kalkata	1	1	0	0	0	0	0	0	0	0	0	0
Teherán	0	1	0	0	0	0	1	0	1	1	0	0
Bagdád	0	0	1	0	0	1	0	0	1	0	1	1
Bombaj	1	1	0	0	0	0	0	0	1	0	0	0
Karáčí	0	1	0	0	0	1	1	1	0	0	0	1
Moskva	0	0	1	0	0	1	0	0	0	0	0	0
Káhira	0	0	1	1	0	0	1	0	0	0	0	1
Rijád	0	0	0	0	0	0	1	0	1	0	1	0

Příloha G – Tabulka přímých spojů pro modrá města

	Madrid	Washington	Atlanta	Petrohrad	Toronto	San Francisco	Essen	Milán	Chicago	New York	Paříž	Londýn
Madrid	0	0	0	0	0	0	0	0	0	1	1	1
Washington	0	0	1	0	1	0	0	0	0	1	0	0
Atlanta	0	1	0	0	0	0	0	0	1	0	0	0
Petrohrad	0	0	0	0	0	0	1	0	0	0	0	0
Toronto	0	1	0	0	0	0	0	0	1	1	0	0
San Francisco	0	0	0	0	0	0	0	0	1	0	0	0
Essen	0	0	0	1	0	0	0	1	0	0	1	1
Milán	0	0	0	0	0	0	1	0	0	0	1	0
Chicago	0	0	1	0	1	1	0	0	0	0	0	0
New York	1	1	0	0	1	0	0	0	0	0	0	1
Paříž	1	0	0	0	0	0	1	1	0	0	0	1
Londýn	1	0	0	0	0	0	1	0	0	1	1	0

Příloha H – Tabulka přímých spojů pro žlutá města

	Santiago	Buenos Aires	Bogota	Miami	Johannesburg	Lagos	Lima	Mexico City	Sao Paulo	Los Angeles	Chartúm	Kinshasa
Santiago	0	0	0	0	0	0	1	0	0	0	0	0
Buenos Aires	0	0	1	0	0	0	0	0	1	0	0	0
Bogota	0	1	0	1	0	0	1	1	1	0	0	0
Miami	0	0	1	0	0	0	0	1	0	0	0	0
Johannesburg	0	0	0	0	0	0	0	0	0	0	1	1
Lagos	0	0	0	0	0	0	0	0	1	0	1	1
Lima	1	0	1	0	0	0	0	1	0	0	0	0
Mexico City	0	0	1	1	0	0	1	0	0	1	0	0
Sao Paulo	0	1	1	0	0	1	0	0	0	0	0	0
Los Angeles	0	0	0	0	0	0	0	1	0	0	0	0
Chartúm	0	0	0	0	1	1	0	0	0	0	0	1
Kinshasa	0	0	0	0	1	1	0	0	0	0	1	0

Příloha I – Tabulka přímých spojů pro červená města

	Taipei	Ho Či Minovo Město	Bangkok	Jakarta	Manila	Tokio	Šanghaj	Peking	Sydney	Hong Kong	Soul	Ósaka
Taipei	0	0	0	0	1	0	1	0	0	1	0	1
Ho Či Minovo Město	0	0	1	1	1	0	0	0	0	1	0	0
Bangkok	0	1	0	1	0	0	0	0	0	1	0	0
Jakarta	0	1	1	0	0	0	0	0	1	0	0	0
Manila	1	1	0	0	0	0	0	0	1	1	0	0
Tokio	0	0	0	0	0	0	1	0	0	0	1	1
Šanghaj	1	0	0	0	0	1	0	1	0	1	1	0
Peking	0	0	0	0	0	0	1	0	0	0	1	0
Sydney	0	0	0	1	1	0	0	0	0	0	0	0
Hong Kong	1	1	1	0	1	0	1	0	0	0	0	0
Soul	0	0	0	0	0	1	1	1	0	0	0	0
Ósaka	1	0	0	0	0	1	0	0	0	0	0	0